SINGAPORE UNIVERSITY OF TECHNOLOGY AND DESIGN

長庚大學 Chang Gung University

Systolic Array Based Convolutional Neural Network Accelerator

CHUA Shi Hui

Advisor: TEO Tee Hui
Co-advisor: WEY I-Chyn

SUTD-CGU Dual Masters Programme in Nano-Electronic Engineering and Design (NEED)

March 3, 2022

Outline

1. Introduction
2. Literature Review
3. Implementation
4. Results
5. Conclusion

Section 1

Introduction

Motivation

- With the advancement in machine learning, widespread adoption of the technology in diverse applications have been on the rise
- Convolutional Neural Networks (CNNs) possess a particular edge over its predecessor, the multi-layer perceptron (MLP), due to its weight sharing features that allows the CNN to use less parameters
- Many of the existing hardware accelerators focus on obtaining higher performance but often comes at a higher energy cost
- Power efficiency is crucial, especially when the hardware accelerator has the potential to be widely deployed, since the energy used will be compounded

Motivation

- The focus will hence be on implementing an energy efficient algorithm for a CNN accelerator via systolic architecture
- The flexibility of reconfigurable hardware such as the FPGA is preferred because it can be easily reprogrammed and redeployed
- A cost-optimized lightweight solution is implemented through low end FPGA hardware so as to allow for greater accessibility
- The solution will incorporate the whole CNN network and build upon the larger set of functions available in machine learning in order to accommodate a wider variety of applications

Section 2

Literature Review

In-datacenter Performance Analysis of a Tensor Processing Unit (TPU) [1], Google, Norman P. Jouppi

- TPUs are ASICs that make use of Matrix Multiply Units to accelerate the inference of neural networks
- The Matrix Multiply Unit contains 256×256 Multiply-Accumulate (MAC) units that can perform 8-bit MAC operations each on signed or unsigned integers
- Fixed array shape of 256×256 so TPU is not optimized for smaller CNNs

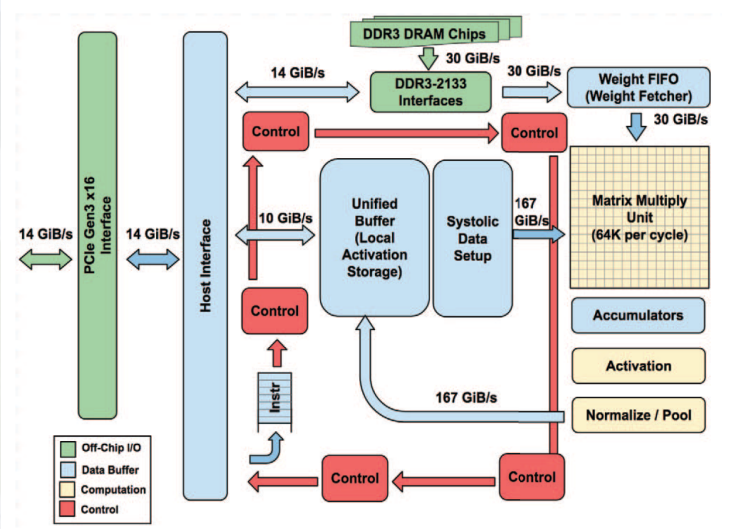

Figure 1: TPU Block Diagram, as illustrated in [1]

In-datacenter Performance Analysis of a Tensor Processing Unit (TPU) [1], Google, Norman P. Jouppi

- Weight stationary systolic dataflow used for reduced memory access
- Data flows in from the left and weights are preloaded from the top
- MAC operation flows through the matrix diagonally
- Performance degrades slightly when matrix size increase because each step takes longer to complete

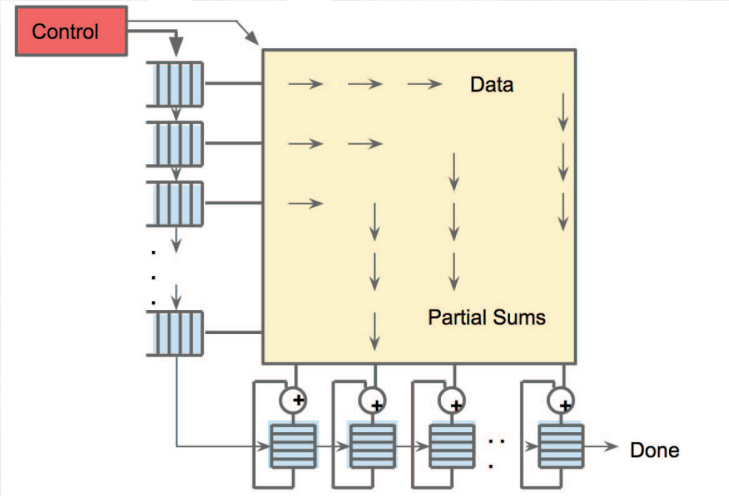

Figure 2: Matrix Multiply Unit dataflow, as illustrated in [1]

Eyeriss An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks [2], MIT, Vivienne Sze

- Eyeriss is an ASIC using systolic dataflow in a 12×14 PE Array
- Proposed new processing dataflow, Row Stationary (RS)
- Run Length Compression (RLC) is done to compress consecutive zero data into a 5-bit number
- Only ReLU function supported

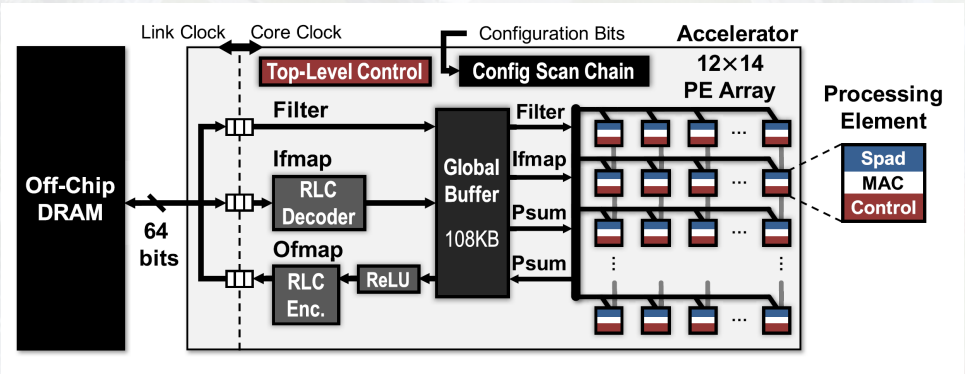

Figure 3: Eyeriss system architecture, as illustrated in [2]

Eyeriss An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks [2], MIT, Vivienne Sze

- Row Stationary reduces both filter and input movements
- Weights are passed across rows, inputs are passed diagonally and psum will flow vertically
- This reduces higher level memory accesses by preloading data onto the local scratchpads

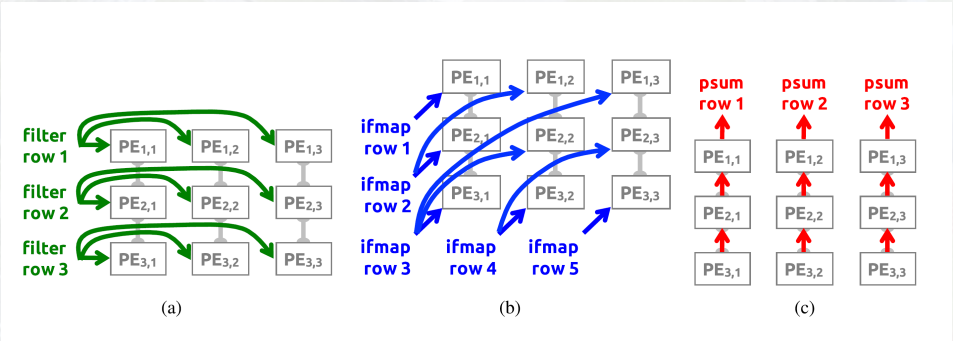

Figure 4: Eyeriss dataflow, as illustrated in [2]

FPGA-based accelerator for convolution operations [3]

- [3] used the Weight Stationary systolic dataflow for convolutional operations
- Huge FIFO buffers of almost 3 times of PE array size
- Only convolution done, no activation or other CNN functions

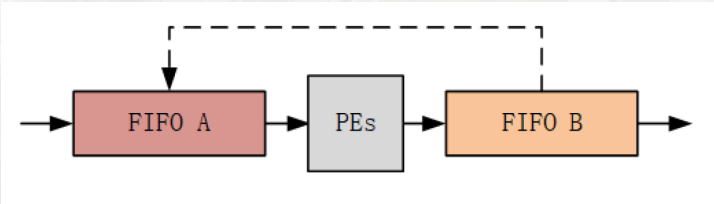

Figure 5: System Design, as illustrated in [3]

- Implemented on Zedboard FPGA
- 8×4 PE array used

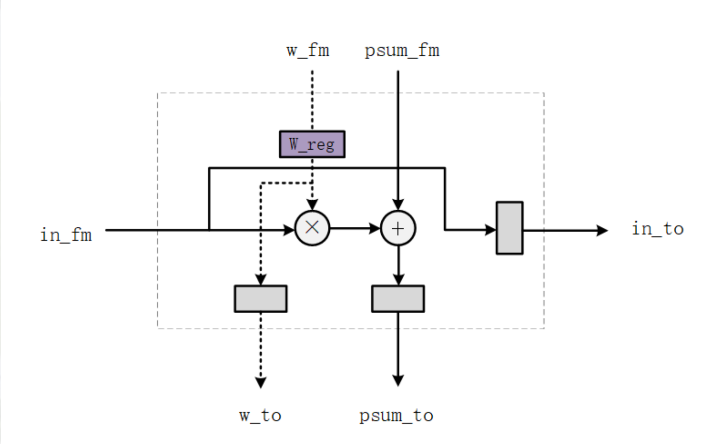

Figure 6: Processing Element Design, as illustrated in [1]

High-performance Convolutional Neural Network Accelerator Based on Systolic Arrays and Quantization [4]

- Output stationary dataflow used with 3×3 array
- Output position stays constant while weights and inputs are shifted into the array
- Quantized 8-bit data used
- Weight is broadcast to all PEs at once
- Implemented on Zynq-XC7Z035

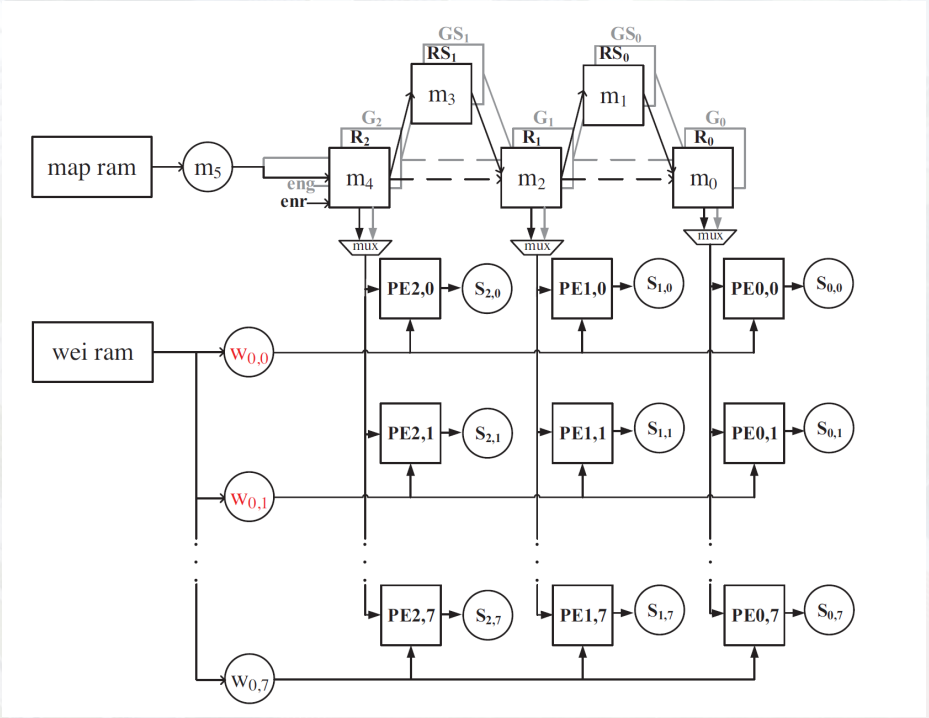

Figure 7: Convolutional Unit Design, as illustrated in [4]

Section 3

Implementation

Implemented Network Topology

Implemented network is shown in Figure 8. It consists of the following layers:

- Convolutional: 5×5 kernel, done via weight stationary systolic array
- ReLU Activation Function, with options for Sigmoid, Tanh and Exponential functions
- Max Pooling Function, with option for Average Pooling Function
- Fully Connected Layer, with Argmax Function for output prediction

Cost efficient PYNQ-Z2 FPGA was used for lightweight model implementation

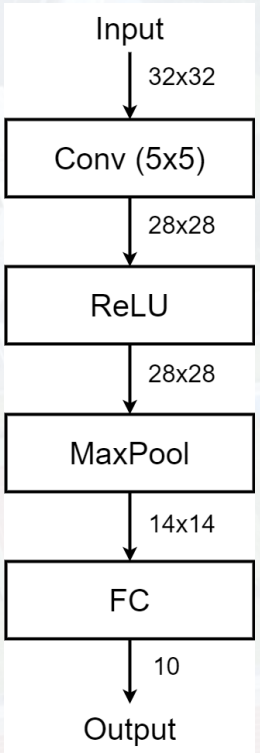

Figure 8: Implemented Network

System Block Diagram

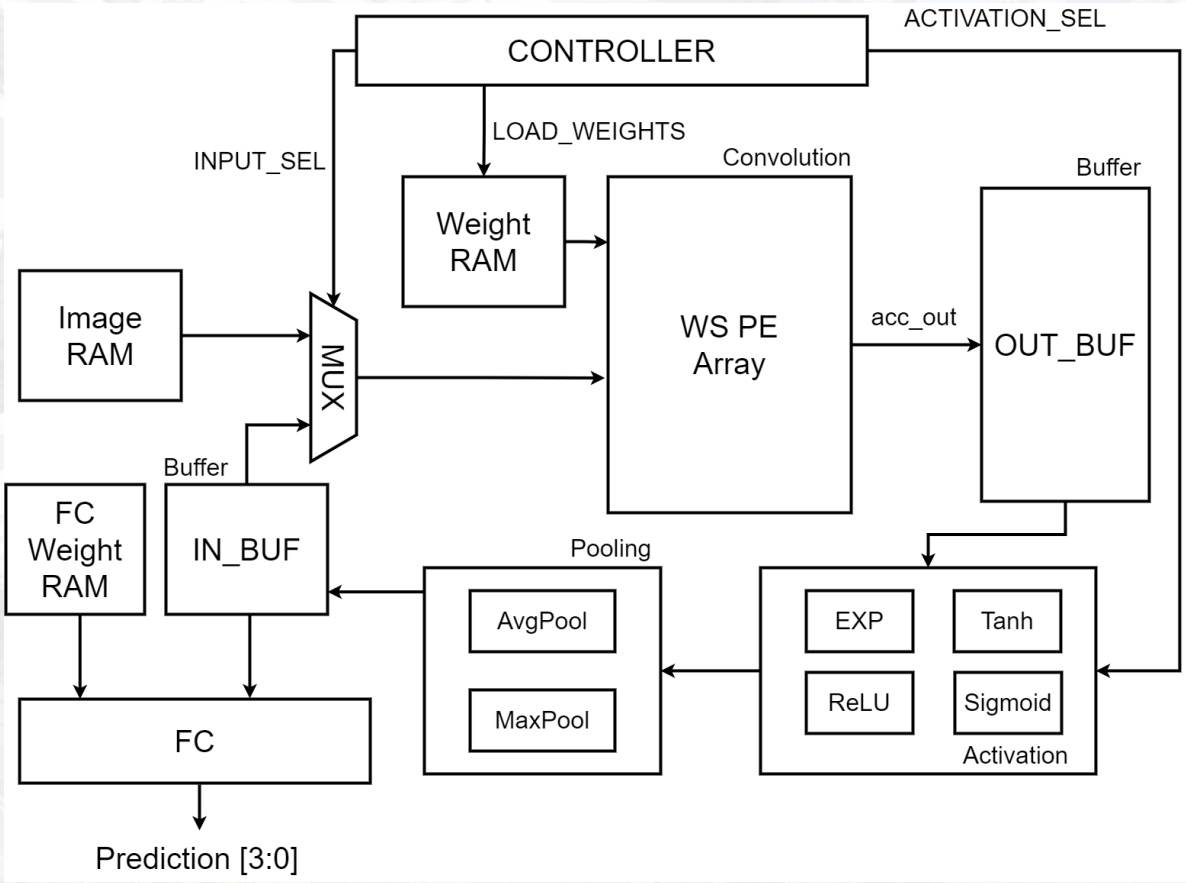

Figure 9: System Block Diagram

Weight and Image RAM

- Contains preloaded data files stored in memory
- Weight RAM outputs the weights to all the PEs as well as the initial bias value all at once
- Image RAM schedules the input to the PE array for systolic dataflow according to Equations (1) to (3)
- The dataflow will iterate through p and q per clock cycle

$$addrx1 = (q) \cdot IMG\_WIDTH + p - 1 \quad (1)$$

$$addrx2 = (1 + q) \cdot IMG\_WIDTH + p - 2 \quad (2)$$

$$addrx3 = (2 + q) \cdot IMG\_WIDTH + p - 3 \quad (3)$$

$$max\_p = IMG\_WIDTH + KERNEL\_SIZE \quad (4)$$

$$max\_q = IMG\_WIDTH - KERNEL\_SIZE + 1 \quad (5)$$

Weight Stationary Processing Element Design

Figure 10 shows the design of a single processing element with the following functions:

- Weight preload with a weight register
- Input and weight multiplication
- Addition of a partial sum
- Input forwarding

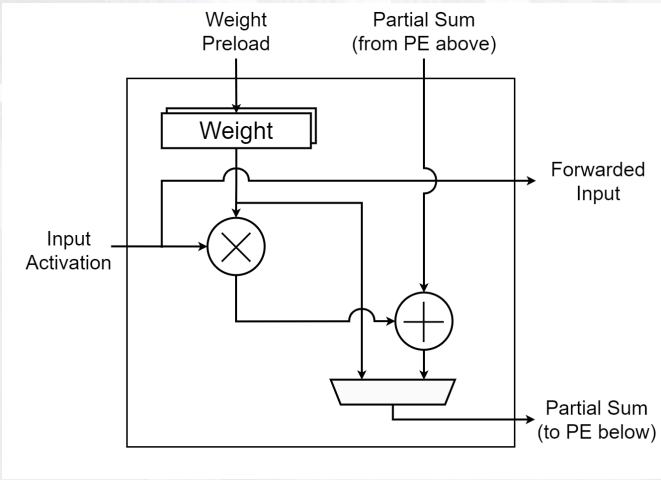

Figure 10: Weight Stationary Processing Element Design

5 by 5 PE Block Diagram

Each PE from Figure 10 is combined to form a larger PE Array shown in Figure 11.

- Preloaded weight registers
- Only 5 total signals for the input, to be cascaded down to the other PEs
- Addition of one bias per PE array at the first PE
- Input forwarding
- Accumulator to collect the outputs

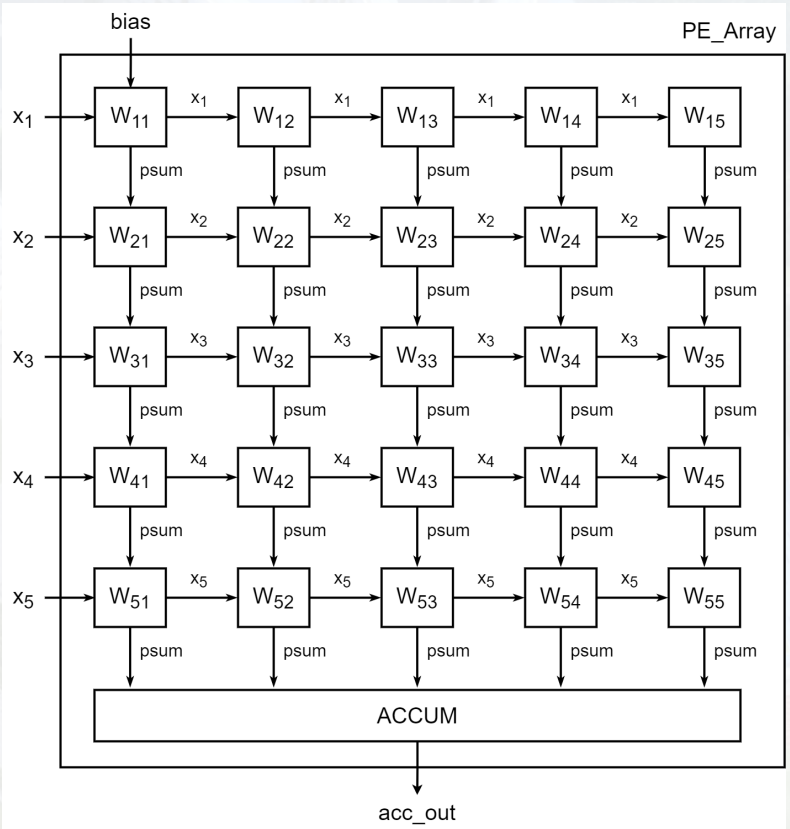

Figure 11: 5×5 Processing Element Array

3 by 3 PE Array Block Design

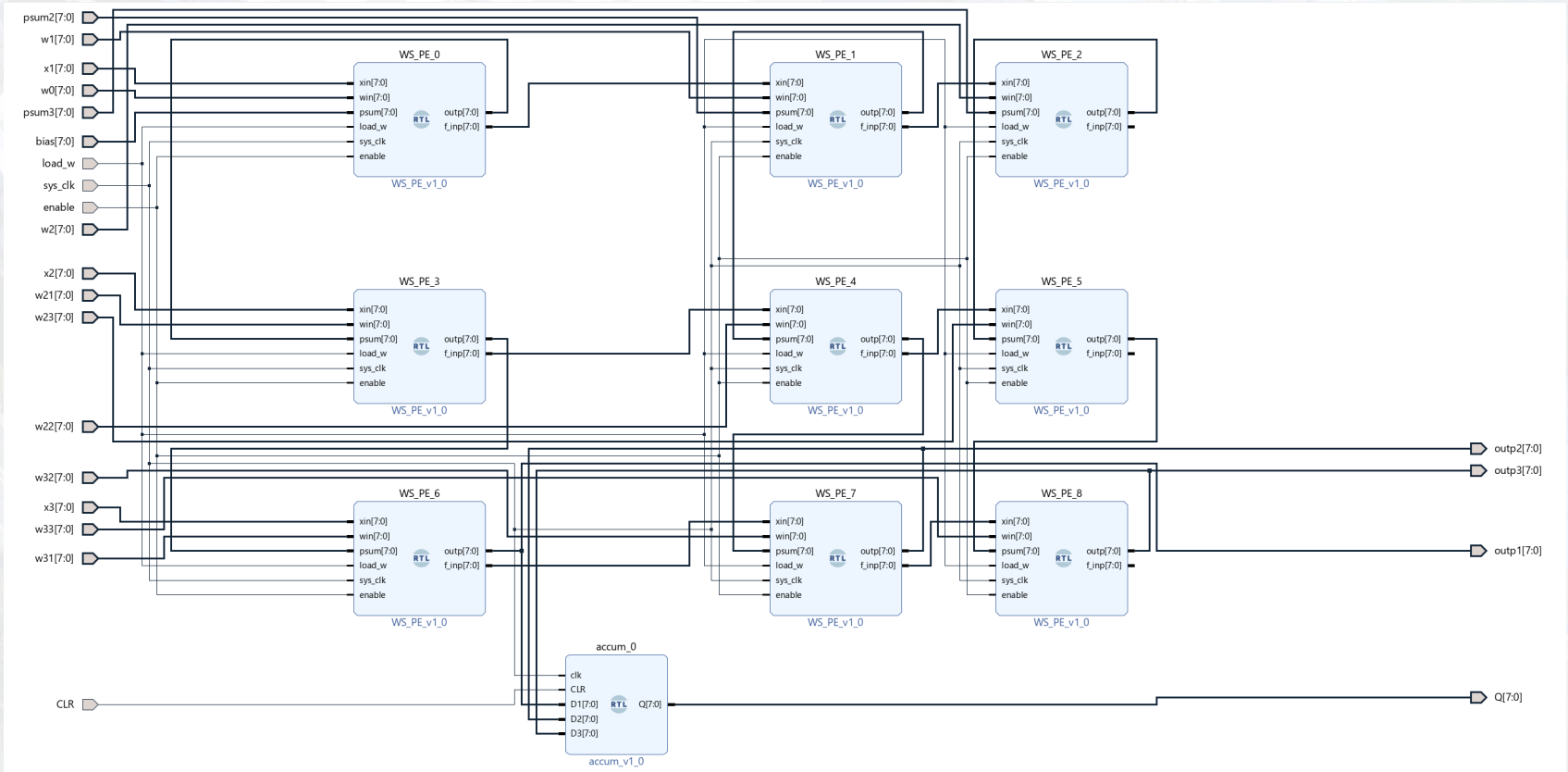

Figure 12: Block Design for 3×3 Processing Element Array

3 by 3 PE Simulated Results

- load_w signal preloads the weights
- Inputs are scheduled one clock cycle apart

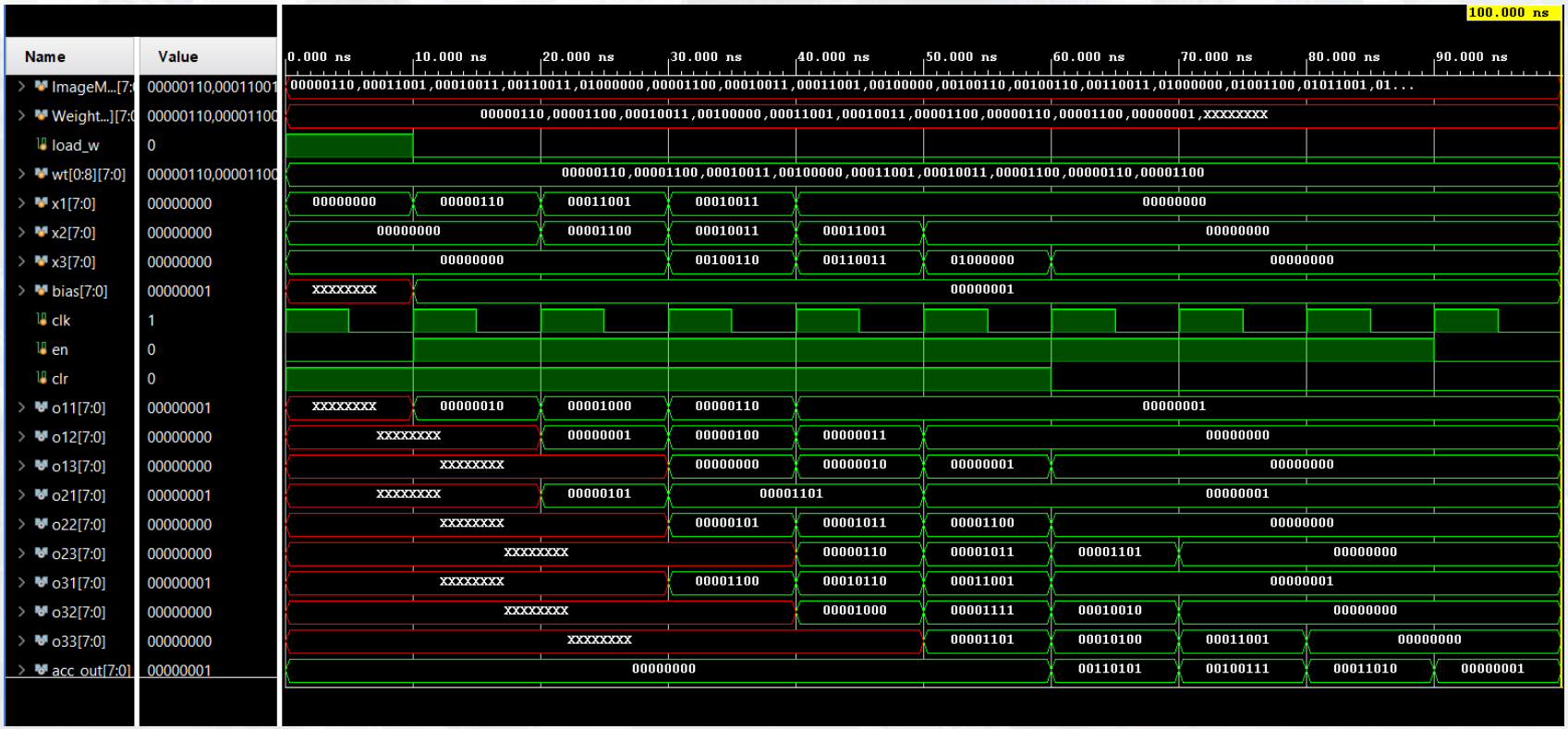

Figure 13: Simulation Results for 3×3 Processing Element Array

5 by 5 PE Simulated Results

5 inputs and 25 partial sums, accumulated into one output result

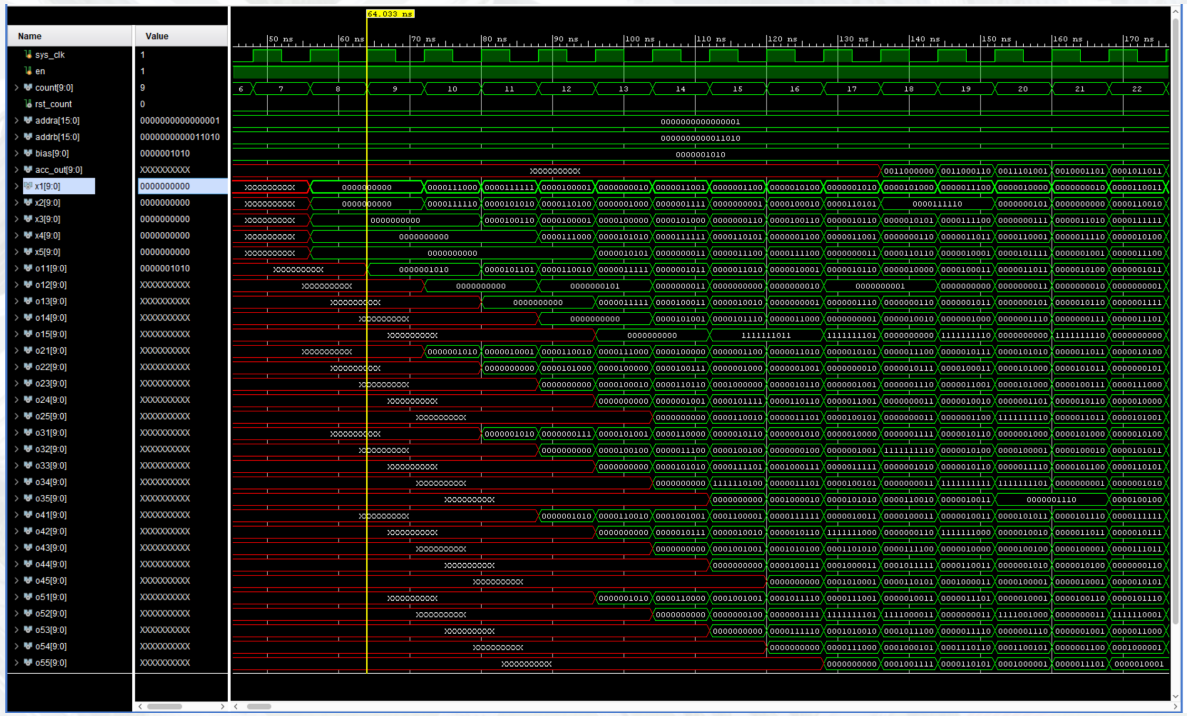

Figure 14: Simulation Results for 5×5 Processing Element Array

Reduction in Memory Access

Systolic architecture reduces the number of memory accesses as only 5 inputs need to be accessed from the RAM instead of one access per PE, which results in 25 accesses per clock cycle.

$$Reduction = 1 - \frac{k}{k \cdot k} = 1 - \frac{1}{k} \tag{6}$$

The first output from the accumulator occurs at first 2k cycles, after which the output from the PE array is continuous

Output Buffer

- Output buffer is used to collect the incoming data from PE array
- Clocked data input and address based on enable and address signals from the controller
- Asynchronous read is available with a stride of 2 both horizontally and vertically
- Four outputs are given at once to the activation block in the pooling function format
- Input feature width has to be specified in this block

Activations Block

The accumulator output from the output buffer enters the activations block next

- SEL chooses the function
- 4 supported functions:
  1. Exponential Function
  2. Approximated Tanh
  3. Approximated Sigmoid
  4. ReLU
- Output is fed into the pooling function

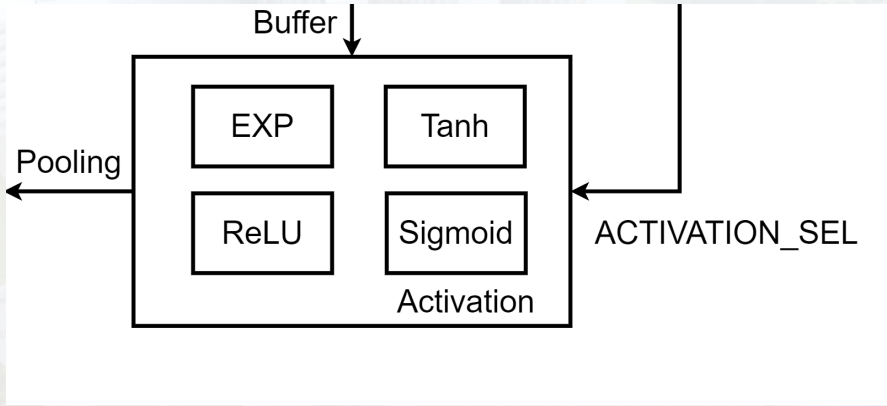

Figure 15: Activation Functions Block

Exponential Function

- Additional consideration was done to include an exponential function since many activation functions require the exponential calculation
- The e^x module will build upon the mathematical derivation in [5] for extended range
- A redefinition of input x is required as per Equation (7)

$$x = q \cdot ln2 + r \tag{7}$$

The e^x function can then be found by Equation (8).

$$e^x = 2^q \cdot e^r \tag{8}$$

Exponential Function Hardware Implementation Details

q is an integer determined by Equation (9). r is determined by Equation (10) and has a range $r < ln2$.

$$q = \lfloor x/ln2 \rfloor \tag{9}$$

$$r = x - q \cdot ln2 \tag{10}$$

- Since division is not a synthesizable operation, a division by $ln2$ will be replaced by a multiplication of $\frac{1}{ln2} = 1.442$
- 2^q is done with a bit-shifter in hardware
- The floor function is implemented by splicing the integer bits.
- The values of e^r are obtained from a 4-bit LookUp Table, with the first four fractional bits of r used to select the value, as opposed to [5] using the Xilinx CORDIC IP core

Exponential Block Design

The exponential block design is shown in Figure 16.

- The input is first used to calculate the values of q and r
- The calculations of 2^q and e^r are then done in parallel
- The results are multiplied together before giving the final output

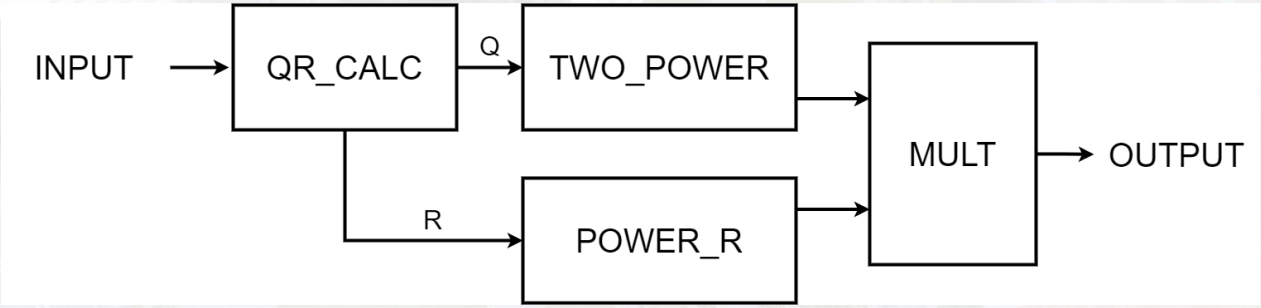

Figure 16: Exponential Block

Exponential Block Simulation Results

- Pipelined calculation from Figure 16 results in a 3 cycle internal latency
- One more clock cycle is used for the exponential block to propagate the output
- Simulation results show that the output is available at the 5th clock cycle

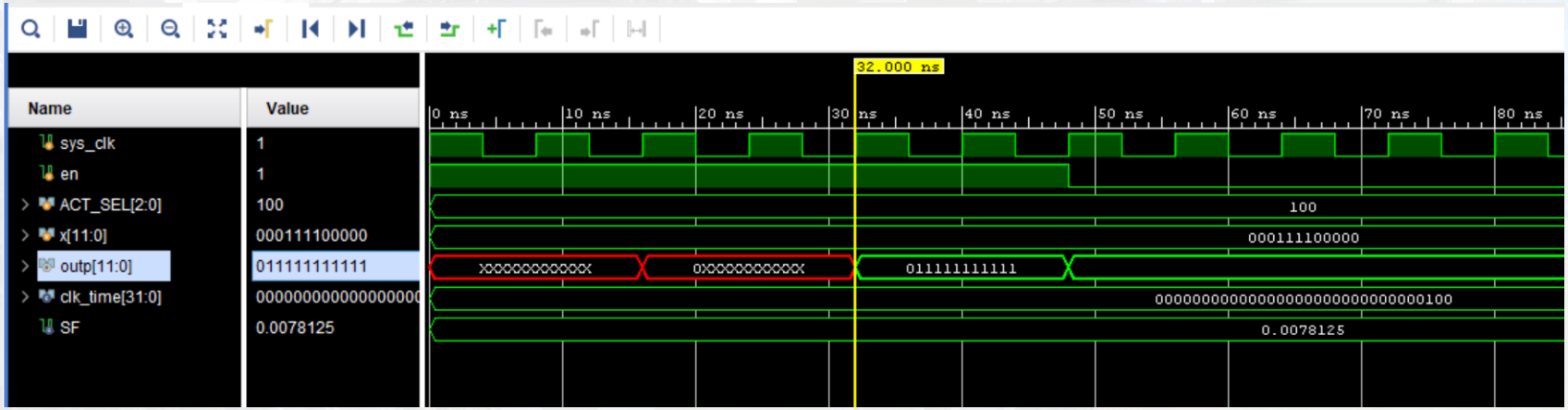

Figure 17: Simulation Results for the Exponential Block

Accuracy of the Exponential Block

- Figure 18 shows the values of the approximated and actual exponential function, with an absolute error of 0.358 at $x = 2$
- The output is saturated at 15.99 due to bit limitations
- Higher precision LUTs are required for better accuracy but the hardware resources will double for every additional bit

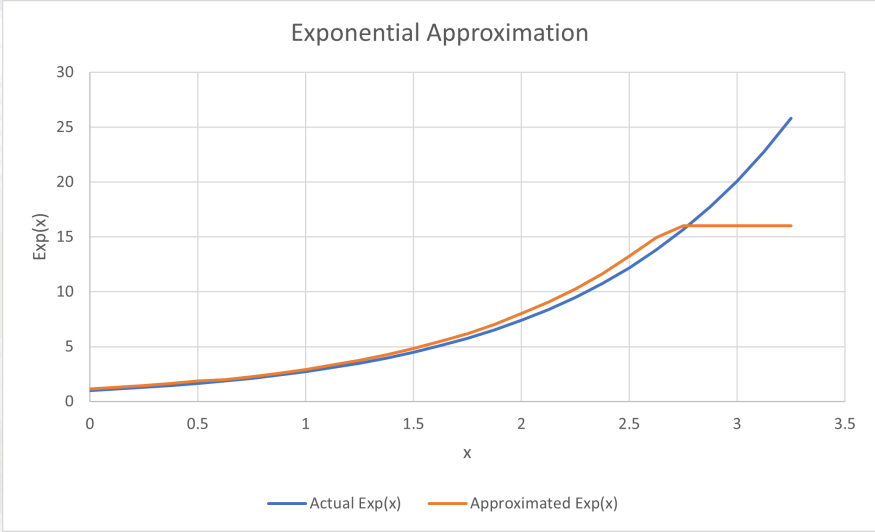

Figure 18: Exponential Block Accuracy Plot

Tanh Approximation

Linear approximation was done to implement the Tanh function

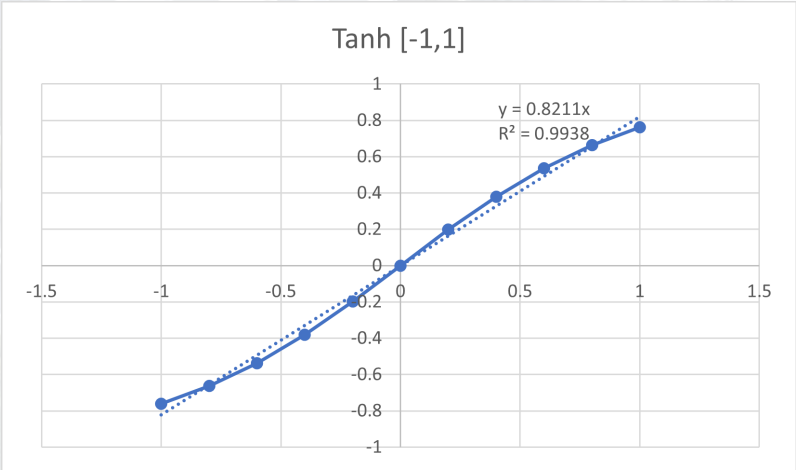

Figure 19: Tanh Best Fit Line for x = [-1,1]

Five piecewise functions are used based on the best fit lines

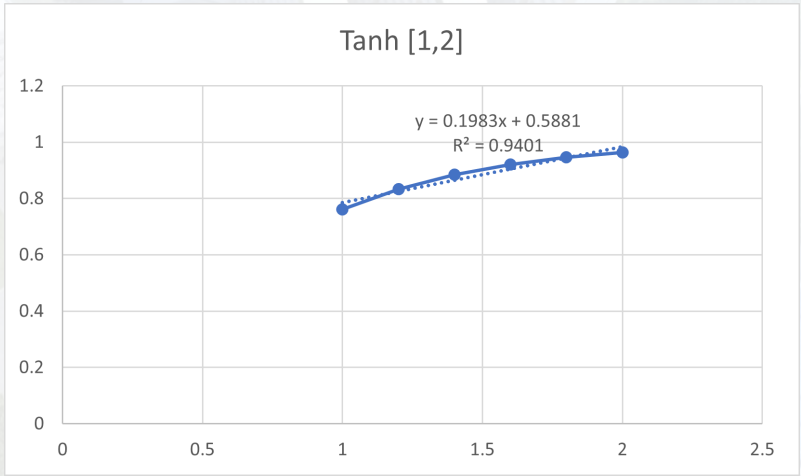

Figure 20: Tanh Best Fit Line for x = (1,2]

Accuracy of the Tanh Approximation

$$Tanh(x) = \begin{cases} \sim x + 1'b1 & \text{if } x < 0 \\ 0.8211x & \text{if } 0 \leq x < 1 \\ 0.1983x + 0.5881 & \text{if } 1 \leq x \leq 2 \\ 1 & \text{if } x > 2 \end{cases} \tag{11}$$

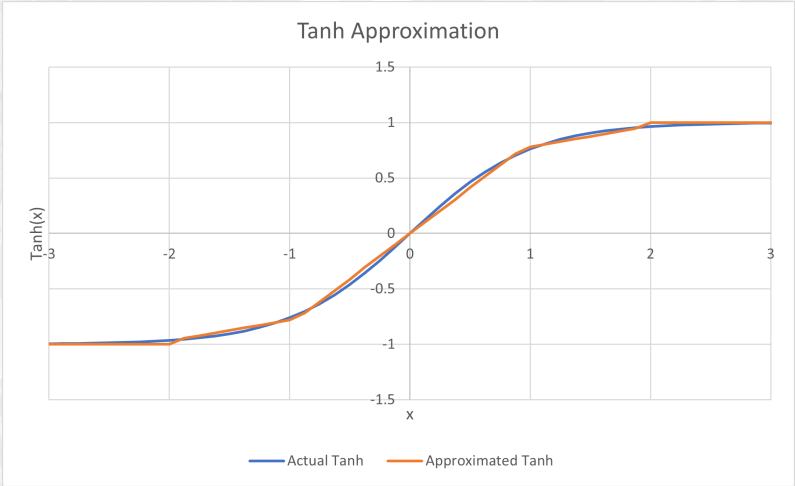

Figure 21: Tanh Approximation Accuracy Plot

Sigmoid Approximation

Linear approximation was done to implement Sigmoid function

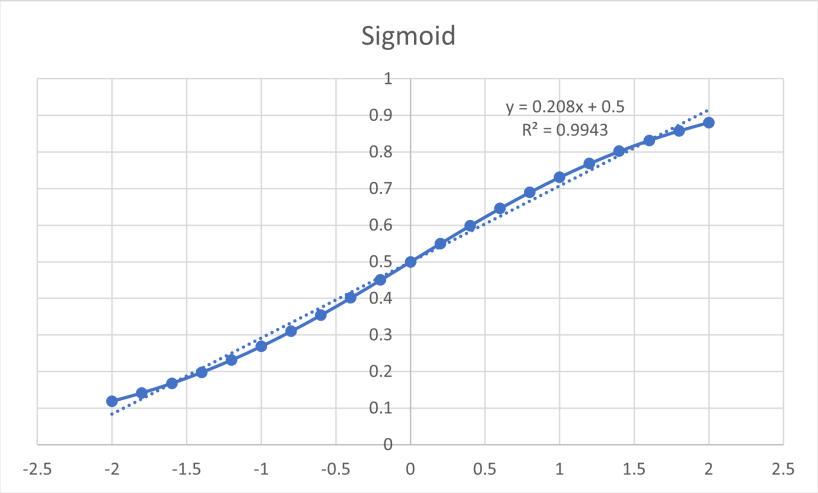

Figure 22: Sigmoid Best Fit Line for x = [0,2)

Five piecewise functions are used based on the best fit lines

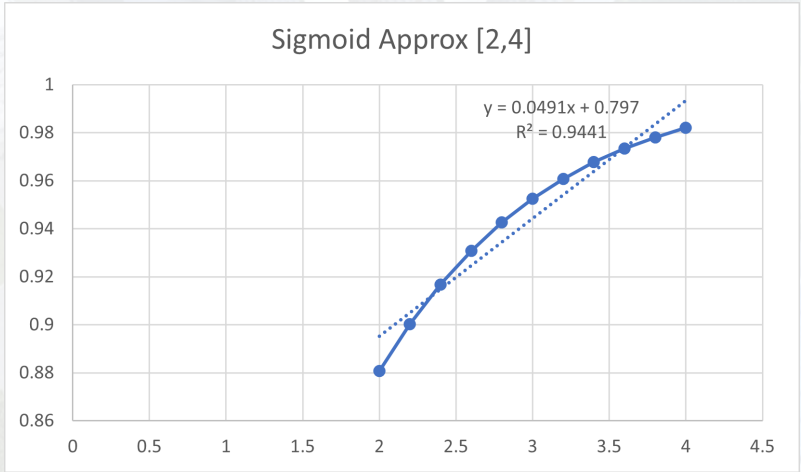

Figure 23: Sigmoid Best Fit Line for x = [2,4]

Accuracy of the Sigmoid Approximation

$$Sig(x) = \begin{cases} \sim x + 1'b1 + 1 & \text{if } x < 0 \\ 0.208x + 0.5 & \text{if } 0 \leq x < 2 \\ 0.0491x + 0.797 & \text{if } 2 \leq x \leq 4 \\ 1 & \text{if } x > 4 \end{cases} \tag{12}$$

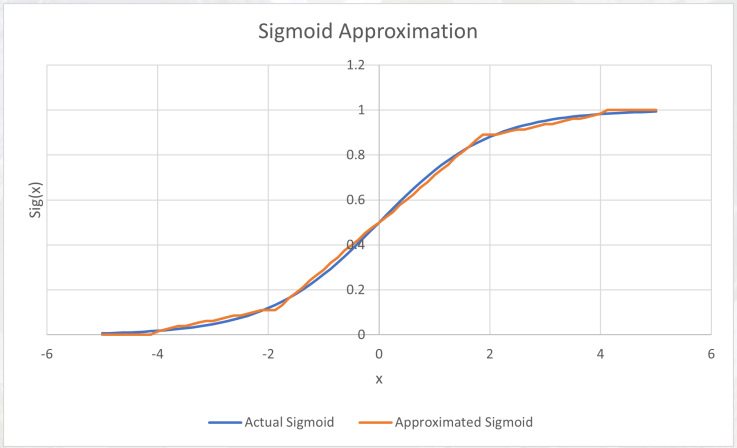

Figure 24: Sigmoid Approximation Accuracy

ReLU Function

- MSB of the input is used to determine whether the output is zero or equal to the input

$$ReLU(x) = \begin{cases} 0 & \text{if } x < 0 \\ x & \text{if } x \geq 0 \end{cases} \tag{13}$$

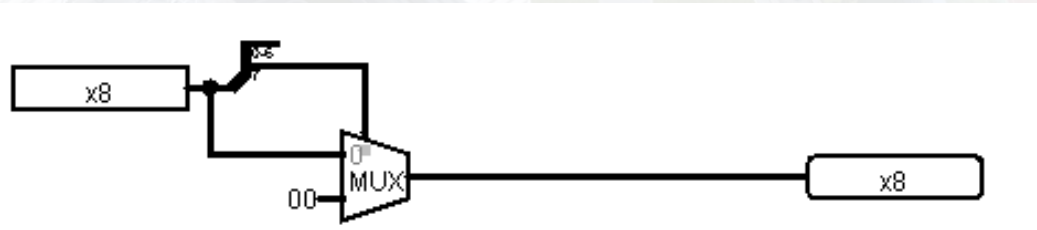

Figure 25: ReLU Circuit simulated by LogiSim [6]

System Block Diagram

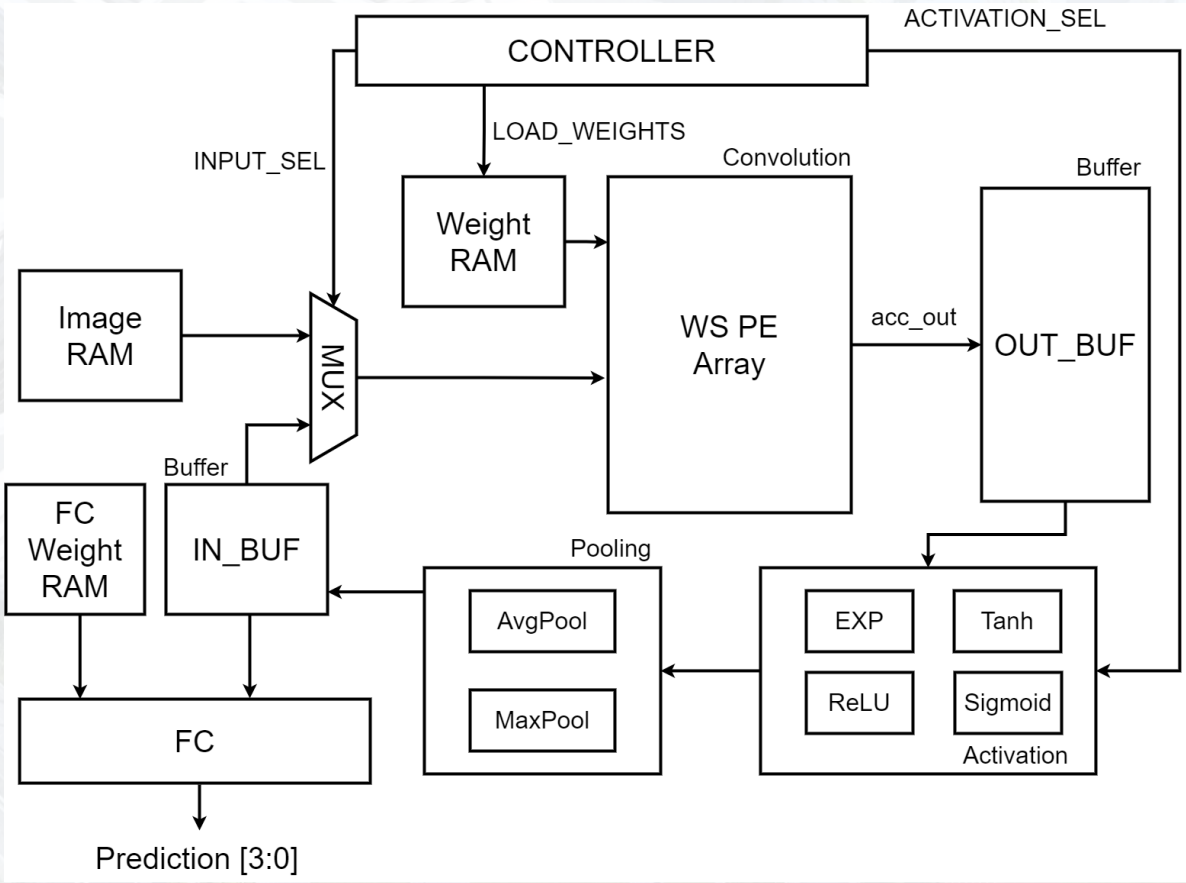

Figure 26: System Block Diagram

Pooling Functions

- Takes in POOL_SEL signal from the controller to determine the function used
- Max pooling gives the largest value out of 4 inputs, using 3 comparators to compare among the 4 input values
- Average pooling gives the average of the 4 inputs according to Equation (14)
- The output of the pooling function is a quarter of the size of the input (eg. from 28×28 to 14×14)

$$Avg\_Out = (I_1 + I_2 + I_3 + I_4) \cdot 0.25 \tag{14}$$

FC Block

- Input and weight from RAM are multiplied
- An accumulator stores the bias value and adds subsequent results to the internal register
- ReLU and negation are both done in parallel to get the positive and negative values
- Negation takes two's complement of negative numbers but retains the sign

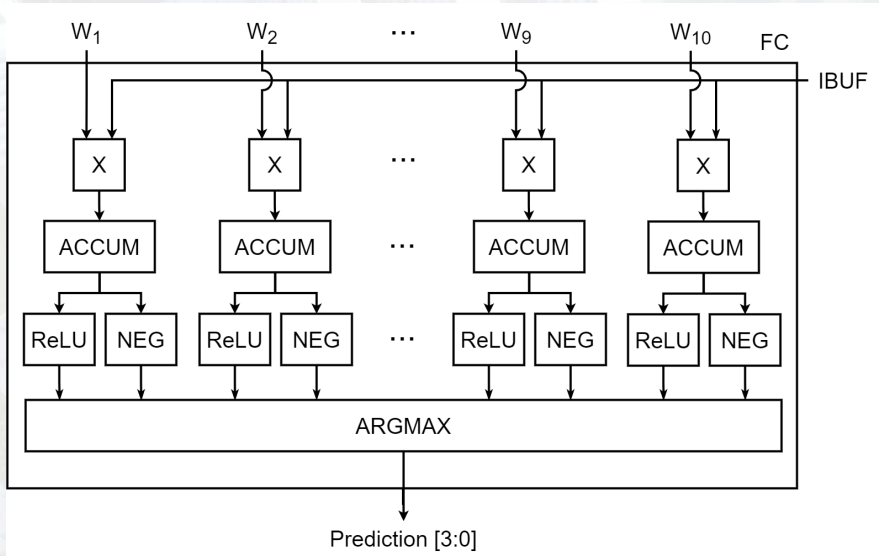

Figure 27: FC Block Diagram

Argmax

- Argmax function takes the inputs from previous step and uses a four stage comparator design is used to find the maximum of 10 arguments
- If all inputs are negative, result is the smallest absolute number
- Each comparator stores both the larger value and the position of the larger value

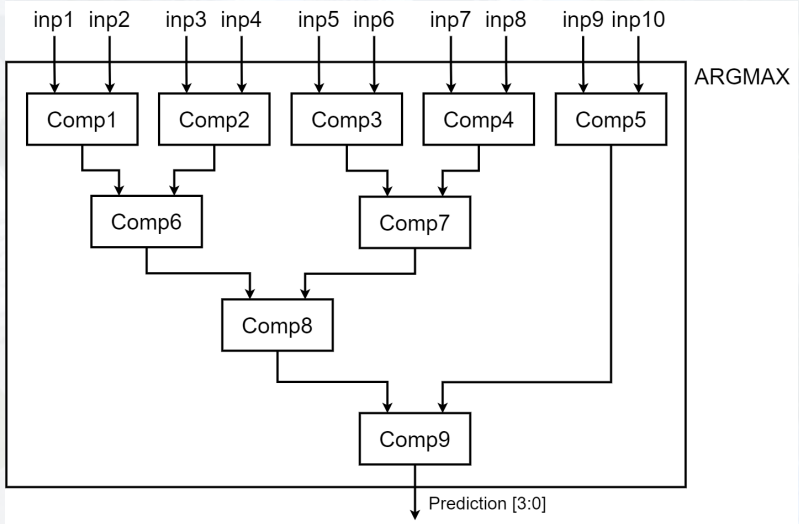

Figure 28: Argmax Block Diagram

Prediction is output based on the position of the largest value and mapped to the FPGA LEDs

Section 4

Results

FPGA Used

- The FPGA used is the TUL PYNQ-Z2 based on the ZYNQ XC7Z020 chipset
- Low cost model, readily available
- 125 MHz external reference clock on the PL
- 13,300 logic slices available, each with four 6-input LUTs and 8 FFs
- 220 DSP slices (minimum 10-bit for utilization)

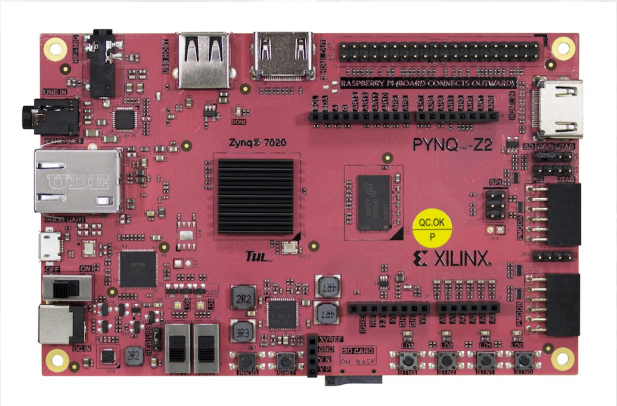

Figure 29: PYNQ-Z2 FPGA as illustrated in [7]

Accuracy

- Pytorch trained weights were quantized after training was completed
- 12-bit fixed-point Q4.7 format was used in the implementation
- Classification accuracy on MNIST dataset is shown in Table 1
- Implemented accuracy is 90.9%, with a 1.3% drop in accuracy from the original model
- 0.8% drop to Int4 quantized accuracy is due to bit overflow issues
- Further 0.5% drop to implemented accuracy is due to fixed-point precision loss

Table 1: Classification Accuracy Results.

Original	Int3 Quantized	Int4 Quantized	Implemented
92.2%	66.35%	91.4%	90.9%

Number of Operations per Clock Cycle

Table 2: Number of Operations per Module per Clock Cycle.

Module Name	Multiply	Add	Number of Modules	Total Ops
PE Array	25	25	1	50
Tanh	1	1	4	8
Sigmoid	1	1	4	8
Exponential	3	1	4	16
AvgPool	1	3	1	4
FC	10	10	1	20

Resource Utilization Report

LUTRAM was used instead of BRAM due to sparse data. A Xilinx clocking wizard IP is used to alter the clock frequency and accounts for the use of the Mixed-Mode Clock Manager (MMCM).

Table 3: Resource Utilization Report.

Resource	Utilization	Total Available
LUT	7,137	53,200
LUTRAM	2,032	17,400
FF	2,373	106,400
DSP	55	220
IO	7	125
MMCM	1	4

Implemented Design

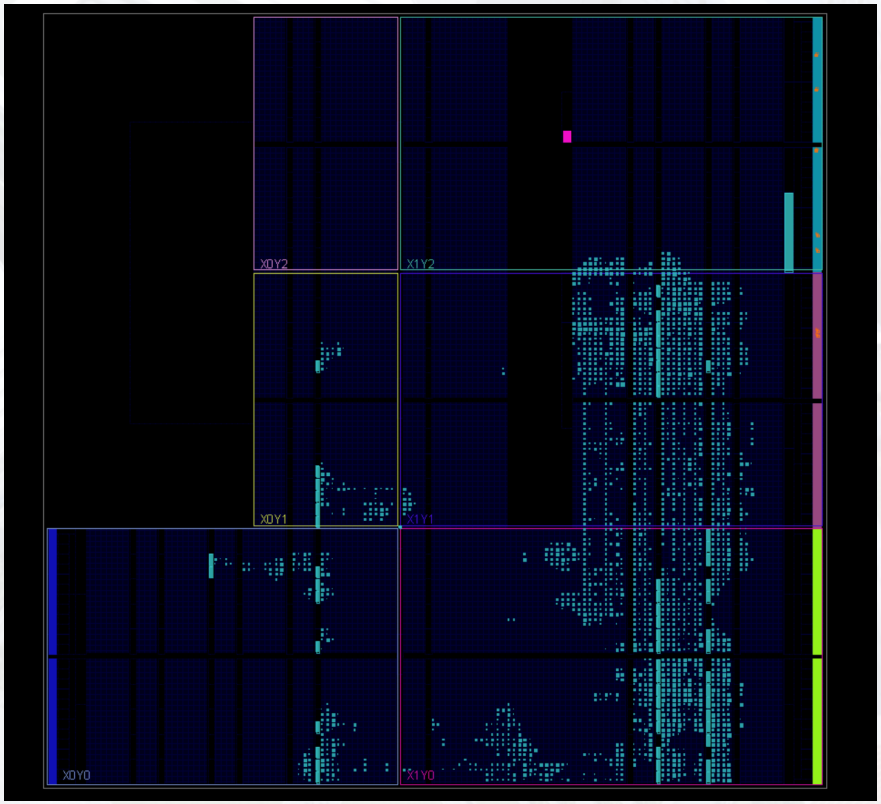

Figure 30: Implemented Device Usage

Performance Metrics

- The accelerator throughput can be calculated by finding the number of operations that can be handled by the accelerator in one second
- Measured in Giga Operations per second, can be normalized against each DSP slice for performance density and against each Watt for power efficiency
- The theoretical maximum speed of the FPGA is when the setup time slack is reduced to zero
- Actual clock speed will depend on the clock dividers available on each FPGA

$$GOps = \frac{Num\_Operations}{Clock\_Period - Setup\_Slack} \tag{15}$$

Accelerator Performance

Table 4: Implementation Results.

Metric	**Proposed Accelerator**
Clock Frequency	100 MHz
Clock Period (ns)	10
Setup Slack (ns)	0.349
Power Consumption (W)	0.363
Number of operations	106
Throughput (GOps)	10.98
Performance Density (GOPs/DSP)	0.200
Power Efficiency (GOPs/W)	30.26

Performance Comparison

Proposed design achieved up to 1.59× better power efficiency compared to other systolic implementations and up to 6.17× better power efficiency compared to non-systolic dataflow.

Table 5: Performance Comparison.

	Proposed	[3]	[4]	[8]	[9]	[10]
Board	PYNQ-Z2	ZedBoard	Zynq-XC7Z035	Artix-7	Zynq-XC7Z100	Virtex 690t
DSPs Used	55	64	758	8**	364	2833
Max DSP Slices	220	220	900	240	2020	3600
Throughput (GOps)	10.98	2.09	125	1.28	19.81	636
Performance Density (GOps/DSP)	0.2	0.0328	0.165	0.16**	0.054	0.224
Power Efficiency (GOps/W)	30.26	19*	36.3	4.9	5.24	24.46
Improvement	1x	1.59x	0.83x	6.17x	5.77x	1.24x

*Data unavailable, estimated using static device power of 0.110W
**Data unavailable, estimated based on 8 PEs = 8 DSPs used

Section 5

Conclusion

Conclusion

- Systolic array is a more energy efficient architecture than regular MAC array for convolutional operations
- Input data access was reduced by 80% on a 5×5 array
- Weight stationary dataflow is implemented on a cost efficient FPGA using a lightweight model
- Up to 6.17× better power efficiency achieved compared to non-systolic array implementations
- Proposed accelerator can also compute ReLU, Tanh, Sigmoid and exponential functions, unlike most other accelerators with only ReLU function
- Depending on accuracy and range requirements of the exponential function, LUT implementation can be considered instead of the Xilinx CORDIC IP block to reduce pipeline delay

Future Work

- Custom Quantization Engine can be developed to have an application aware quantized CNN training for better performance and optimization
- Bit variable configuration can be introduced to allow for different range of inputs for different networks
- Block reconfigurable PE arrays can be used to for different array shapes and explore the most optimized configuration

References

[1] N. P. Jouppi, C. Young, N. Patil, et al. "In-datacenter performance analysis of a tensor processing unit". In: *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*. 2017, pp. 1–12. DOI: 10.1145/3079856.3080246.

[2] Y.-H. Chen, T. Krishna, J. S. Emer, et al. "Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks". In: *IEEE Journal of Solid-State Circuits* 52.1 (2017), pp. 127–138. DOI: 10.1109/JSSC.2016.2616357.

[3] Y. Cao, X. Wei, T. Qiao, et al. "FPGA-based accelerator for convolution operations". In: *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*. 2019, pp. 1–5. DOI: 10.1109/ICSIDP47821.2019.9172934.

[4] Y. Li, S. Lu, J. Luo, et al. "High-performance Convolutional Neural Network Accelerator Based on Systolic Arrays and Quantization". In: *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*. 2019, pp. 335–339. DOI: 10.1109/SIPROCESS.2019.8868327.

[5] R Rekha and K. P. Menon. "FPGA implementation of exponential function using cordic IP core for extended input range". In: *2018 3rd IEEE International Conference on Recent Trends in Electronics, Information Communication Technology (RTEICT)*. 2018, pp. 597–600. DOI: 10.1109/RTEICT42901.2018.9012611.

[6] C. Burch. "Logisim: A graphical system for logic circuit design and simulation". In: *Journal of Educational Resources in Computing* (Mar. 2002), pp. 5–16.

[7] TUL. *TUL*. URL: https://www.tulembedded.com/FPGA/ProductsPYNQ-Z2.html (visited on 02/27/2022).

[8] A. N. Mazumder, H. Ren, H.-A. Rashid, et al. "Automatic Detection of Respiratory Symptoms Using a Low Power Multi-Input CNN Processor". In: *IEEE Design Test* (2021), pp. 1–1. DOI: 10.1109/MDAT.2021.3079318.

[9] Y. Yao, Q. Duan, Z. Zhang, et al. "A FPGA-based Hardware Accelerator for Multiple Convolutional Neural Networks". In: *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*. 2018, pp. 1–3. DOI: 10.1109/ICSICT.2018.8565657.

[10] C. Zhang, Z. Fang, P. Zhou, et al. "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks". In: *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. 2016, pp. 1–8. DOI: 10.1145/2966986.2967011.