Systolic Array Based Convolutional Neural Network Inference on FPGA

Shi Hui, Chua
Engineering Product Development
Singapore University of Technology and Design
Singapore
shihui_chua@mymail.sutd.edu.sg

T. Hui, Teo
Engineering Product Development
Science, Mathematics and Technology
Singapore University of Technology and Design
Singapore
tthui@sutd.edu.sg

Mulat Ayinet, Tiruye
Engineering Product Development
Singapore University of Technology and Design
Singapore
mulatayinet_tiruye@mymail.sutd.edu.sg

I-Chyn, Wey
Artificial Intelligence Research Center
School of Electrical and Computer Engineering, College of Engineering
Graduate Institute of Electrical Engineering, Chang Gung University
Chang Gung University, Taiwan
icwey@mail.cgu.edu.tw

***Abstract*—Convolutional Neural Networks (CNNs) possess a particular edge over its predecessor, the Multi-Layer Perceptron (MLP). This is due to its weight sharing features that allows the CNN to use less parameters for the same number of outputs as compared to the MLP. Systolic arrays capitalize on the weight sharing property of CNNs to do data reuse while performing convolutional operations, in order to reduce the power consumption from the memory accesses. A kernel fitting systolic processing element array was designed with only positive multiplication to increase the throughput and power efficiency of the CNN accelerator, while using weight stationary dataflow to achieve data reuse in the systolic array. A cost-optimized lightweight solution is implemented through low-cost FPGA hardware so as to allow for greater accessibility. The CNN accelerator consumes 0.363 W power at 100 MHz operating frequency. A peak throughput of 10.98 GOps/s was achieved with peak performance density of 0.200 GOps/s/DSP and peak power efficiency of 30.26 GOps/s/W. Even with the added support for additional functions, 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 implementations.**

***Index Terms*—convolutional neural network, FPGA, systolic array**

I. Introduction

The advancement of machine learning has created many kinds of Artificial Neural Network (ANN)s. In particular, the Convolutional Neural Network (CNN) in machine learning allows for high accuracy in image recognition, while using less parameters than its predecessors such as the Multi-Layer Perceptron (MLP). To accelerate the process of running such neural networks, many hardware have been introduced. Many of the existing hardware accelerators focus on obtaining higher performance but this often comes at a higher power cost. With global warming, power efficiency is becoming more crucial, especially when the hardware accelerator has the potential to be widely deployed, since the power used will be compounded.

The weight sharing feature of CNNs not only enables them to capture spatial features in an image, but also leads to a substantial amount of data reuse in the CNN. This data reuse can be exploited in systolic arrays to minimize data accesses in convolution. Since convolution takes up 90% of the execution time [1], data reuse during convolutions can translate to large power savings in a CNN accelerator.

This paper therefore aims to capitalize on the weight sharing property of CNNs by using systolic dataflow in the Processing Element (PE) array to achieve power efficiency. Weight stationary dataflow is used to achieve data reuse in the systolic array. The full CNN will be implemented on an Field Programmable Gate Array (FPGA), including the convolution, activation, pooling functions and the Fully Connected (FC) layer. The design will aim to accommodate as many options as possible for the activation and pooling functions.

This paper makes the following contributions:

- A weight stationary systolic array based PE array for input data reuse
- A kernel fitting PE array for full PE utilization
- Optimization of the usage of both two's complement and signed magnitude representations for PE calculations
- Additional support for Tanh, Sigmoid, Exponential, Average Pooling and Argmax functions while still being power efficient

This paper is organized as follows into four sections. Section II details the design used to achieve a power efficient weight stationary systolic array based CNN accelerator. This is followed by the Section III, where the performance of the proposed design is presented. Finally, the conclusions are drawn in Section IV.

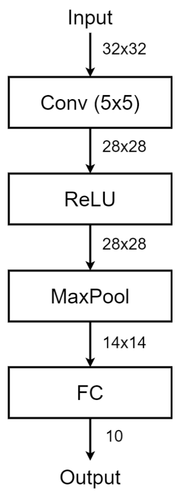

Fig. 1. Implemented Convolutional Neural Network Topology

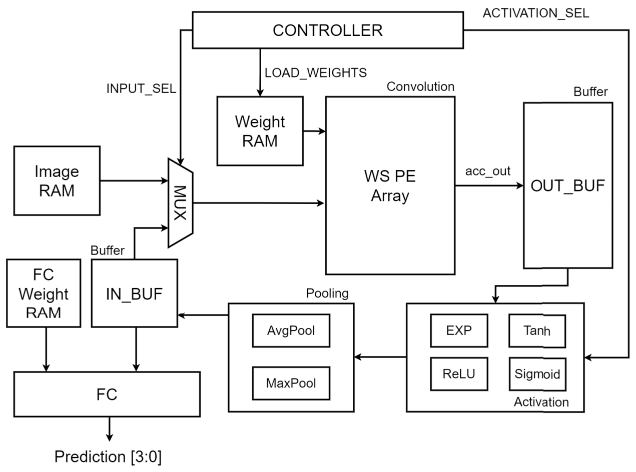

Fig. 2. System Block Diagram

II. Method

The implemented CNN topology is a lightweight model illustrated in Figure 1. The MNIST dataset was used for digits recognition, with 32×32 grayscale inpout images. The training of this model was done using Pytorch with 10 epochs and quantized to Q4.7 fixed-point format for use in the accelerator. No bit optimization was done on this training.

The proposed system block diagram is shown in Figure 2. Data will be loaded into Image Random Access Memory (RAM) and Weight RAM initially. The data is passed onto the PE array in systolic dataflow requirements. The output of the PE array is then passed into an output buffer. The output buffer data then goes into the activation block, with 4 activation modules. The controller will handle all the dataflow via select and enable signals. Every activation module contains several common activation functions that can be selected by the ACTIVATION_SEL signal. This signal can be fixed or runtime configurable. Activation function output is fed directly into the pooling functions, which has a POOL_SEL signal to choose the desired function. After the pooling functions, the data is stored in the input buffer for propagation into the next stage, which can either be the FC layer or back into the PE array for further convolutional operations. A separate FC weight RAM is used to store the weight data for the FC layer, which performs MAC operations, followed by Argmax function. The final output of the CNN accelerator is a 4-bit prediction of the output class. All the blocks are implemented with inputs and outputs all in 12-bit Q4.7 format, same as the fixed-point format for the input data.

A. Processing Element Array

The Weight Stationary (WS) PE array size was chosen to be 5×5, same as the kernel size. This is so that the number of outputs is consistent with the convolutional operation to maximize the PE utilization rate. Convolutional operations will be done such that each clock cycle will provide one output result. Individual weights for all the PEs have been configured to load in all at once in this design, to reduce the latency due to weight loading. Therefore, only 1 clock cycle is required to load all the weights from Random Access Memory (RAM) regardless of the number of PEs. This reduces the amount of data scheduling that is required for the weights, since it takes only one time step to execute. The data selection and scheduling is done in the image and weight RAMs, with an initial address given from the controller, and will be discussed in the next subsection.

A 5×5 PE array was implemented as shown in Figure 3. In the WS dataflow, the weights are preloaded into each PE's weight register before any inputs are passed in, with a total of 25 PEs. The inputs are then passed in with a stagger, with the input forwarded to the PE on the right and the result passed on to the bottom PE. The inputs x1 to x5 are passed horizontally to the other PEs in the same row but they do not have the same data since the new data takes one clock cycle to propagate to the next PE. This allows the systolic dataflow to work its way through all the PEs and sum up the corresponding values for a convolution operation. For a 5×5 array, each PE at the bottom most row will yield the partial sum of 5 results each. These 5 results at the same cycle correspond to one output feature and are then added together in an accumulator. The accumulator output, also the PE array output, is hence one output feature per clock cycle.

Furthermore, to streamline the calculations within the PE array, data arriving from the RAM is pre-processed before passing into the PE array block. A negation block is added between the RAM output and the PE array input, turning two's complement negative numbers into signed magnitude negative numbers according to Equation (1), where Y is the output and X is the input. After the negation is done, each PE will then only have to process an 11-bit unsigned multiplication for 12-bit inputs since the signed bit does not contribute to the multiplication result. The signed bits of both the input and the weight become the signed bit of the output via an Exclusive-

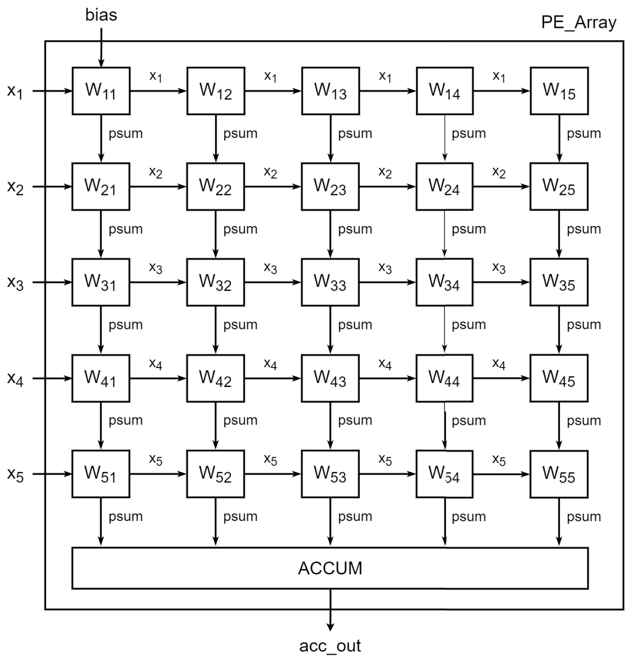

Fig. 3. Block Diagram for 5×5 Processing Element Array

OR (XOR) operation. This reduces the multiplication load by 1 bit with an increase in only one input cycle latency. Note that the bias term is not negated since it will only be involved in addition operations.

$$Y = \{X[MSB], \sim X[MSB-1:LSB]+1\} \tag{1}$$

After the multiplication, the resulting partial sum is converted back into two's complement format for further addition operations. The data and output will cascade down after the first PE receives the input. Each time step will produce one partial sum (psum) and there will be continuous dataflow after the first data finishes propagation through the array. The first accumulator output will arrive at 2k cycles, where k is the kernel shape and also the width of the PE array, in this case, at the 10^{th} cycle. This allows time for the inputs to pass through all the PEs required for the output value computation.

Systolic architecture reduces the number of memory accesses as only 5 inputs need to be accessed from the RAM instead of an access per PE, which results in 25 accesses per clock cycle. The reduction of the number of memory accesses for the data input in a systolic array can be summarized in Equation (2). For a 5×5 PE array, this translates to a 80% reduction in the input memory accesses required.

$$Reduction = 1 - \frac{k}{k \cdot k} = 1 - \frac{1}{k} \tag{2}$$

The design of each PE in the weight stationary dataflow is shown in Figure 4. A weight register is used in each PE to store the preloaded weight value, with a separate signal to trigger the loading of the weights. The input is multiplied by the weight and forwarded to the PE on the right at the next cycle. An addition is also done with the partial sum from the top, yielding a partial sum to be passed to the bottom PE.

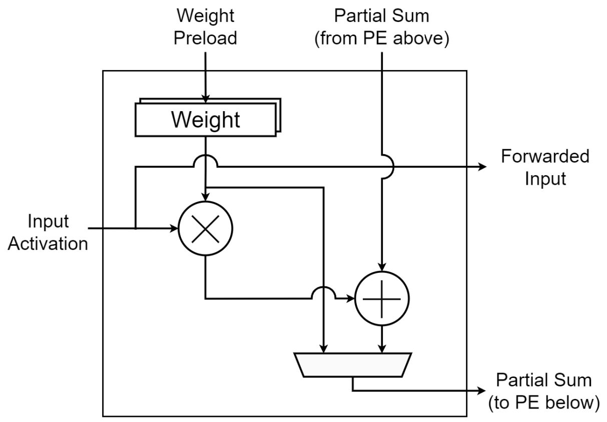

Fig. 4. Weight Stationary Processing Element Design

The improvements made on the PE array are summarized below:

1) The PE array is set at the same size as the kernel for convolution, resulting in one output per cycle and full PE utilization.
2) All the weight data are loaded in together, reducing the time taken for weights to cascade down the PE array.
3) The input weights and data are converted from two's complement to signed magnitude representation before passing into the PE array. This only has to be done once per weight value.
4) In each individual PE, two's complement representation was used for addition operations, while signed magnitude representation was used for multiplication operations.

B. Image and Weight RAM

The weight RAM is initially loaded with the weight data. It supplied both the weight and bias values to the PE array. Sparse data is stored in the LookUp Table Random Access Memory (LUTRAM) while dense data is stored in the Block Random Access Memory (BRAM). The structure of the preloaded weight file is a single dimensional array of binary bits. The weight RAM takes in an address input from the controller and starts incrementing the weight address read by 1 for every PE available. All the weights are loaded into the PE array at the same time to reduce data loading latency. As a result, the weight RAM is only in operation for a short amount of time.

For the image RAM, the data scheduling is much more complex. The input is a 32×32 image and the data has to be scheduled one cycle apart for every additional row of PE used. The image RAM takes in only the data enable signal from the controller, after which it will cycle through the entire array of input data automatically. The data enable signal would have to wait for the weights to be loaded before engaging. The equations to access the image addresses are shown in Equations (3) to (5).

$$addrx1 = (q) \cdot IMG\_WIDTH + p - 1 \tag{3}$$

$$addrx2 = (1 + q) \cdot IMG\_WIDTH + p - 2 \tag{4}$$

$$addrx3 = (2 + q) \cdot IMG\_WIDTH + p - 3 \tag{5}$$

The dataflow will iterate through parameters p and q per clock cycle, whereby p is the number of horizontal iterations each data has to go through and q is the number of vertical iterations the data has to pass through. The maximum p and q values can be found in Equations (6) and (7) respectively. max_q is also equal to the dimension of the output array since each output will result from one vertical step. Due to the nature of WS systolic array, the max_p is larger than the output dimension since all inputs have to go through all the PEs if it is required by any one of them in the row. Data from irrelevant kernel and input matching such as $X_{11} \cdot W_{13}$ will be discarded at the output.

$$max\_p = IMG\_WIDTH + KERNEL\_SIZE \tag{6}$$

$$max\_q = IMG\_WIDTH - KERNEL\_SIZE + 1 \tag{7}$$

C. Activations Block

The activations block consists of 4 activation modules and handle 4 outputs from the buffer in one clock cycle. Each module consists of four functions: ReLU, Tanh, Sigmoid and exponential. The ReLU function was done by using the Most Significant Bit (MSB) to determine whether the output is zero or equal to the input. The Tanh and Sigmoid functions are done with 5 piecewise linear approximation functions. Additional consideration was done to include an exponential function since many activation functions require the exponential calculation. The exponential function was done by following the work in [2] to extend the range of e^x, but using 4-bit LUTs instead of XILINX CORDIC IP core. The function selection is done by the controller.

D. Fully Connected

The FC layer performs similar operations to the convolutional layer, but since no systolic arrays was involved in the FC layer, it was implemented separately. A separate RAM was used to store the FC weights. The weights are multiplied by the same output from the input buffer, with 10 multiplication modules for the 10 output classes. The biases were stored in an accumulator, with the multiplication result summed with it. A four-staged comparator design was used to implement the Argmax function to determine the largest output for both positive and negative results. This 4-bit prediction is mapped onto the LEDs on the FPGA board.

TABLE I
CLASSIFICATION ACCURACY RESULTS.

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

III. RESULTS

The design was implemented on a PYNQ-Z2 [3] board. The classification accuracy of the CNN shown in Figure 1 on the MNIST dataset is shown in Table I. The original accuracy is the test accuracy from PYTORCH-trained CNN weights. Python quantized accuracy refers to the inference run on Python in float64 format, while truncating overflowing bits. Int3 refers to 3-bit integer being used, with any number smaller than -8 or larger than 7.9 truncated during the accumulation in Python. Likewise, Int4 refers to the 4-bit integer with range -16 to 15.9.

The classification accuracy results in Table I show that there is a 0.8% decrease in accuracy from original to int4 quantized. This loss in accuracy is due to overflowing bits since there is a limit to the output values. It is observed that there is a significant drop in accuracy from original to int3 quantized so int3 quantized is insufficient for representing the output. Therefore, 4 bits were used for the implementation in Q4.7 format. There is a total of 1.3% accuracy drop from original to implemented model. A further 0.5% accuracy loss from int4 quantized to the actual Q4.7 implemented accuracy was due to the quantization of bits, as the smallest unit in Q4.7 is 0.0078125 and any value beyond or within that scale will be rounded off. Furthermore, multiplications and additions are all done in the fixed-point format, so a lot of bits were rounded off in the calculations.

The implemented design on the PYNQ-Z2 can run at 100 MHz with pre-selected activation function of ReLU and pre-selected pooling function of maximum pooling. The other modules in the accelerator remains available during this implementation, although they were not directly called for use.

Table II presents the resource utilization report for the implemented accelerator design. LUTRAMs were used in this implementation as there were only 1996 parameters for the total weights (25+1 for convolution and 196×10+10 for FC) and 1024 values for the inputs (32×32). The Mixed-Mode Clock Manager (MMCM) is used in the XILINX [4] clocking wizard IP to alter the clock frequency.

The resource utilization was kept to a minimum by design, keeping in mind the limitations of a cost-efficient solution. Two methods were used to reduce the resource utilization. First, a lightweight model was used with a small number of parameters. Secondly, this design was quantized into 12 bits with no pre-processing of the weight values. The weight values trained from Pytorch was quantized directly for use in the accelerator.

The peak throughput was calculated by finding the peak number of operations that can be handled by the accelerator in one second. The peak number of operations was obtained by calculating the total number of multiplication and addition

TABLE II
CNN ACCELERATOR UTILIZATION REPORT.

Resource	Utilization	Total Available	Utilization %
LUT	7137	53200	13.42
LUTRAM	2032	17400	11.68
FF	2373	106400	2.23
DSP	55	220	25.00
IO	7	125	5.60
MMCM	1	4	25.00

TABLE III
CNN ACCELERATOR IMPLEMENTATION RESULTS.

Metric	**Proposed Accelerator**
Clock Frequency	100 MHz
Clock Period (ns)	10
Setup Slack (ns)	0.349
Hold Slack (ns)	0.029
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

operations in the design. The theoretical maximum speed of the design was obtained by subtracting the setup time slack from the clock period. This theoretical maximum speed is used to calculate the peak throughput of the design. The actual clock speed would depend on the clock dividers available on the FPGA. The implemented results for the proposed design is shown in Table III.

Performance comparison of the proposed design is done against other similar designs in Table IV. The proposed design is first compared with other systolic array based FPGA accelerators in the first two references before comparing with non-systolic array based accelerators with different tiers of FPGA boards used.

The proposed design is used for a small network and hence has the least number of DSPs used. The throughput is hence not indicative of performance since the network size of each design varies largely. Both the performance density and power efficiency of the proposed accelerator are the second best among all the comparisons. A good balance is hence achieved between performance and power. The number of DSP slices available on an FPGA can be used as a rough gauge of FPGA tier. The proposed design is implemented on the most cost-efficient FPGA in this performance comparison table. 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 implementations.

When compared to [5], the PYNQ-Z2 and Zedboard are both based on the ZYNQ 7000 XC7Z020 chipset. Both implementations also use the Weight Stationary systolic array dataflow with 5×5 kernel size for convolutional operations. Therefore, these two designs are good comparisons. The array size used in [5] is 8×4 while the array size used in this design is 5×5. According to [10], the array shape has an impact on the execution time and hence the timing performance and throughput of the accelerator. In this aspect, the proposed design outperforms [5] since the 5×5 array fits the convolutional kernel size exactly. [5] also expends a significant amount of resources for the First In, First Out (FIFO) buffers used, which is almost 3× the total number of PEs. Combining these reasons, the proposed accelerator performs 6.1× better in performance density when compared to [5]. The power data was unavailable from [5] but since the chipset used is the same as the PYNQ-Z2, a minimum power was assumed by taking the static device power of the proposed design, which is 0.110W. Given this assumed best case performance, the proposed accelerator still performed 1.59× better in power efficiency compared to [5].

[6] uses the Output Stationary dataflow in their design, as opposed to the Weight Stationary dataflow in the proposed design. 8 bits were used in [6] as opposed to 12 bits used in the proposed design. The proposed design used 1.5× the number of bits compared to [6]. This is one of the main factors resulting in lower power efficiency than [6]. [6] used quantized weights during CNN training to optimize the trained weights obtained. This was not done in the proposed design as proposed design was aimed at easy accessibility for the accelerator so any weights trained from other software can be directly used in the accelerator. Furthermore, the accelerator used in [6] is the XC7Z035, which is a better performing FPGA than used by the proposed design. Although both FPGAs belong to the ZYNQ 7000 family, the XC7Z035 has better peak DSP performance than the XC7Z020. Combining all of the above factors, [6] performed 1.2× better in power efficiency than the proposed design. However, the proposed design performed 1.2× better in performance density than [6].

When compared to non-systolic implementations, proposed design performs 6.17× and 5.77× better in power efficiency than compared to [7] and [8] respectively, while still performing better in performance density. The data on the number of DSPs used in [7] was unavailable so it was estimated based on the number of PEs used, with a minimum of 8 DSPs used in the design. This number is then used to calculate the estimated performance density of [7].

Comparisons with [9] showed that the proposed design has a much smaller gain of only 1.24× power efficiency even though [9] did not use systolic arrays. The performance density of [9] also outperforms the proposed design by 1.12×. This can be attributed mainly to the type of FPGA used as the Virtex 690t is a very high end model of FPGA and

TABLE IV
PERFORMANCE COMPARISON.

	Proposed	[5]	[6]	[7]	[8]	[9]
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

hence has better performance than the proposed design on the PYNQ-Z2. Additionally, [9] used the same hardware for both convolutional and FC layers, which also contributes to power efficiency.

Overall, the proposed design is cost and power efficient with very little resource utilization. This is done mainly through the systolic array design, with the signed magnitude representation of fixed-point numbers leading to only positive multiplication used. An optimized array shape also helped in improving the throughput of the system. Furthermore, the proposed design has support for additional functions not commonly found in other hardware accelerators.

IV. Conclusion

While systolic arrays have been around for a long time, they are still very efficient in reducing the number of memory accesses. In this paper, the number of data accesses per clock cycle has been reduced by 80% with the design of a 5×5 PE array with weight stationary dataflow. The problem of high power consumption by convolution operations is addressed using systolic arrays to increase data reuse. Systolic dataflow passes the inputs automatically through the array to compute all the required results for convolutional operations.

The PEs in this design handles only positive multiplication in the signed magnitude format, while using two's complement for addition operations. This utilizes the best of both representations by using unsigned operations for both multiplication and addition, making it much more efficient.

Most existing accelerators also do not allow for choice in the selection of different activation and pooling functions and primarily use only ReLU and maximum pooling functions. A system design for a systolic array based CNN accelerator that takes in 32×32 input images was presented, with added support for the ReLU, Tanh, Sigmoid and exponential functions, as well as average and maximum pooling functions. Argmax was used in the FC layer with 10 output classes.

A cost-efficient accelerator was designed on the PYNQ-Z2 FPGA using 12-bits Q4.7 fixed-point representation, reducing the amount of resources required to store the data as compared to floating-point representations. With high levels of reuse using the systolic architecture, 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, despite having additional functionalities to support other common activation and pooling functions. Moving forward the design could be extended to Binarized Neural Network (BNN), [11], for further optimization of hardware resource and power consumption.

References

[1] C. Zhu, K. Huang, S. Yang, Z. Zhu, H. Zhang, and H. Shen, "An efficient hardware accelerator for structured sparse convolutional neural networks on fpgas," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, pp. 1953–1965, 09 2020.

[2] 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)*, pp. 597–600, 2018.

[3] PYNQ, "Pynq - python productivity for zynq - home."

[4] AMD, "Trademark information — AMD."

[5] Y. Cao, X. Wei, T. Qiao, and H. Chen, "Fpga-based accelerator for convolution operations," in *2019 IEEE International Conference on Signal, Information and Data Processing (ICSIDP)*, pp. 1–5, 2019.

[6] Y. Li, S. Lu, J. Luo, W. Pang, and H. Liu, "High-performance convolutional neural network accelerator based on systolic arrays and quantization," in *2019 IEEE 4th International Conference on Signal and Image Processing (ICSIP)*, pp. 335–339, 2019.

[7] A. N. Mazumder, H. Ren, H.-A. Rashid, M. Hosseini, V. Chandrareddy, H. Homayoun, and T. Mohsenin, "Automatic detection of respiratory symptoms using a low power multi-input cnn processor," *IEEE Design Test*, pp. 1–1, 2021.

[8] Y. Yao, Q. Duan, Z. Zhang, J. Gao, J. Wang, M. Yang, X. Tao, and J. Lai, "A fpga-based hardware accelerator for multiple convolutional neural networks," in *2018 14th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 1–3, 2018.

[9] C. Zhang, Z. Fang, P. Zhou, P. Pan, and J. Cong, "Caffeine: Towards uniformed representation and acceleration for deep convolutional neural networks," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 1–8, 2016.

[10] E. Yago, P. Castelló, S. Petit, M. E. Gómez, and J. Sahuquillo, "Impact of the array shape and memory bandwidth on the execution time of cnn systolic arrays," in *2020 23rd Euromicro Conference on Digital System Design (DSD)*, pp. 510–517, 2020.

[11] M. Xiang and T. H. Teo, "Implementation of binarized neural networks in all-programmable system-on-chip platforms," *Electronics*, vol. 11, p. 663, Feb 2022.