Hardware-Accelerated ZYNQ-NET Convolutional Neural Networks on Virtex-7 FPGA

Ahmed J. Abd El-Maksoud[1], Amr Gamal[1], Aya Hesham[1], Gamal Saied[1], Mennat-Allah Ayman[1], Omnia Essam[1], Sara M. Mohamed[1], Eman El Mandouh[2], Ziad Ibrahim[2], Sara Mohamed[2], Hassan Mostafa[1,3]

[1]Electronics and Electrical Communications Department, Faculty of Engineering, Cairo University, Giza 12613, Egypt

[2]Mentor Graphics

[3]University of Science and technology, Nanotechnology and Nanoelectronics Program, Zewail City of Science and Technology, October Gardens, 6th of October, Giza 12578, Egypt.

Email: hassanmostafahassan@gmail.com

2021 International Conference on Microelectronics (ICM) | 978-1-6654-0839-4/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICM52667.2021.9664956

***Abstract*—Convolutional neural network is a class of deep neural networks that has made a great breakthrough in image recognition. CNNs are commonly used to detect and classify visual applications so that they are frequently embedded in image classification tasks. The common trend nowadays is to accelerate the processing of CNNs in order to use them in real-time applications such as image classification and object recognition. This paper presents the implementation of ZynqNet CNN architecture on FPGA. The full ZynqNet CNN layers are implemented on FPGA to reach the max acceleration and make full use of all DSP units. Several optimizations techniques are used in different design phases to improve processing speed, utilized area, and power consumption. In addition, the proposed hardware accelerator achieves 15.6 fps for ZynqNet CNN at maximum frequency. The proposed architecture runs at two different frequencies of 100MHz and 125MHz, and is implemented on Virtex-7 FPGA.**

Keywords—Accelerating CNNs; Convolutional Neural Networks (CNNs); FPGA; Hardware accelerators; ZynqNet.

I. INTRODUCTION

In recent years, the effectiveness of artificial intelligence (AI) is proved to solve a lot of complex problems. The goal is to create an intelligent system that extracts features automatically and recognizes specific patterns. AI is divided into many subfields such as deep learning, machine learning, and natural language processing [1].

Deep convolutional neural network (DCNN) is one of the popular artificial intelligence fields. During the past decade, it has emerged in many applications such as video recognition, image classification, biomedical systems, and self-driving cars [2-5]. DCNN has become a state of art methodology for computer vision problems. It consists of millions of learnable parameters, which are evaluated through the training phase.

The topology of CNN is divided into multiple learning stages that are composed of convolutional layers, non-linear rectified (ReLU) layers, pooling layers, fully connected layers, and softmax layer [6]. Every layer plays a specific role to build a complete CNN model. The training of DCNN models is commonly performed on graphics processing units (GPUs) which have thousands of cores and large external memory bandwidth. It does not require much effort to deploy existing models or train new ones on GPUs using various frameworks. While the training is carried few times during the development phase, the inference process is performed millions of times. This puts new limitations for applications that have a limited power budget and tight latency constraints, such as embedded systems or self-driving cars. As a result, there is a growing demand for dedicated hardware accelerators to get higher throughput and less power consumption.

Many researchers focus on building custom application-specific integrated circuits (ASICs) for accelerating CNNs inference workloads and getting the best performance [7]. Despite being an optimum solution, ASICs do not offer enough flexibility to accommodate the rapid evolution of CNN models and the deployment of new CNN layer types. On the other hand, FPGAs offer interesting design capabilities compared to other implementation types. FPGA-based accelerators provide high throughput, low power consumption, fast prototyping, and reconfigurability at a reasonable cost. There are different types of architectures to accelerate the CNN processing using FPGA [8-10].

ZynqNet CNN is a stripped-down version of SqueezeNet CNN. It consists exclusively of convolutional layers, ReLU, and global average pooling. It has top-1 and top-5 error rates of 41.52% and 15.4%, respectively, on the ImageNet testing dataset. ZynqNet has fewer error rates than SqueezeNet. Also, the number of MAC operations reduced by 38%, and the total number of activations is reduced by 40%, with respect to SqueezeNet CNN. ZynqNet is developed by David Gschwend in 2016. It has 2.5 Million parameters and consists of 27 convolutional layers and only one average pooling layer [11].

Researchers have provided various CNN accelerator architectures on FPGA during recent years. The most similar implementations to the proposed work are investigated for further analysis. The first implementation is Gschwend's accelerator [11]. It is a hardware accelerator on FPGA that allows classifying images using ZynqNet CNN. It implements the full network based on a nested-loop approach which minimizes the number of operations and memory access. The accelerator is synthesized using High-Level Synthesis (HLS) for the Xilinx Zynq XC-7Z045. Secondly, Mousouliotis proposes a hardware accelerator that can process ZynqNet and SqueezeNet. It's implemented using HLS on Zynq-xc7z020 FPGA. It's an improved version from SqueezeJet hardware accelerator and achieves 11.54 fps for the ZynqNet CNN [12]. Finally, CNN-Grinder provides a workflow to accelerate SqueezeNet and ZynqNet using both HLS and SDSoC design methods. It achieves 14.49fps for ZynqNet on Zynq-xczu9eg [13].

This paper is organized as follows; Section II discusses Zynqnet architecture in detail. Section III shows the design improvements and hardware implementation. Moreover, Section IV shows the implementation results on FPGA and comparisons with other work. Section V concludes the work.

II. THE PROPOSED ARCHITECTURE

The proposed accelerator is built based on fully pipelined architecture. All ZynqNet CNN layers are implemented on

FPGA, which provided a massive acceleration for the whole CNN. As shown in Fig. 1, each convolution layer is implemented separately with their independent weights memory and Multiply Accumulation units (MACs) for performing the convolution operations. Two shared memories are used to store the intermediate storage between layers which are Conv1 and all Fire layers. Conv10 is interleaved with one global average pooling layer to reduce the dimension of features retaining the most dominant information. Finally, the output prediction block is implemented to get the index of the output prediction.

A. Convolution units

In the proposed design, each layer requires a specific number of MACs to achieve full parallelism. It is chosen to fully parallelize the processing of output filters. Therefore, the number of used DSPs is with respect to the number of required MACs. Also, each convolution layer is implemented in a separate block with control signals during convolution operation. Moreover, a separate weights memory is generated for each block. The convolution operation is performed using MAC units, and the output is fetched to the ReLU block.

Available DSP resources on the target FPGA is one of the main challenges during the implementation. As the required DSPs for full-layer implementation all convolution layers on the FPGAs are 4448 DSPs, which is unfeasible by Virtex-7 DSPs that are equal to 3600 DSPs only. This problem is solved by sharing a group of DSPs by the identical layers such as fire2/expand3x3 and fire3/expand3x3 layers. By this approach, the same 64 DSPs are used for the two layers. The proposed convolution unit is shown in Fig. 2. It simply consists of intermediate registers, kernels memory, counters, adders, and multipliers.

B. Kernels memory

ZynqNet CNN has around 2.5M parameters, which are either network weights or biases. The number of parameters makes it feasible to implement ZynqNet architecture on FPGAs. However, memory bottleneck requires careful attention for memory organization in the design. Usually,

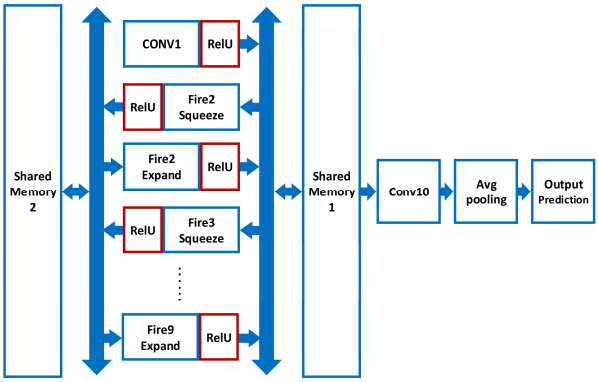

Fig. 1. The proposed architecture

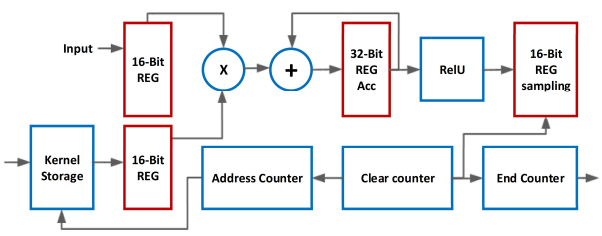

Fig. 2. Convolution unit block diagram

fixed parameters are stored into Read Only Memories (ROMs). In addition, the proposed implementation targets acceleration speed at first priority, so off-chip memories are not an option as they degrade the speed and increase the power consumption. As mentioned in convolution blocks design, each MAC block is responsible for one filter processing. Therefore, fitting all design parameters into a large ROM block increases the memory fetching latency for all MACs. For example, a 10Kb memory is mapped to 64 MACs which requires 64 cycles to fetch a new word. This method results in speed degradation. Consequently, the adopted approach is to use parallel ROMs. Each ROM is used for a group of filters in the design, which grants low latency and consistency.

C. Intermediate memory storage

Each layer has output feature maps that have to be stored in order to be passed to the next layer. Layers storage is made using different resources in FPGA such as BRAMs and register files. A shared memory of 3D BRAMs is used for all layers so that they can read and write from the same memory. The shared memory is divided into two memories to avoid conflict between reading and writing during processing. The current and next layer mutually changes between the memories by reading from memory for the current layer and writing the output in another memory.

D. Average pooling unit

During the implementation of average pooling, two optimization techniques are applied. Firstly, the Conv10 layer does not run simultaneously in the proposed design as it is split into two parts. To take the available resources into consideration, the required number of accumulators for average pooling is saved to half by exploiting the splitting of the conv10 layer. Therefore, a multiplexer is used to select between conv10 part-1 output or conv10 part-2 output, and the multiplexer output is fetched to accumulators array. Secondly, implementing divider blocks on hardware is area ad power-consuming, so they are replaced by simple shifting right operations. The accuracy of the result may be not as same as convention dividing, but the difference is so little and the result is acceptable. The schematic diagram for average pooling is shown in Fig. 3 which consists of the parallel multiplexer, accumulators array, registers, and shifting right blocks array.

E. Output prediction unit

The output prediction layer is located after the average pooling layer which is used to select the classified prediction. It compares the 1024 outputs from average pooling to get the max number which is the class index that refers to the output prediction. One comparator, counter, and two registers are needed as shown in Fig. 4. The unit processes the 1024 value serially which may take 1024 cycles, but provide a utilized area reduction more than parallel comparators.

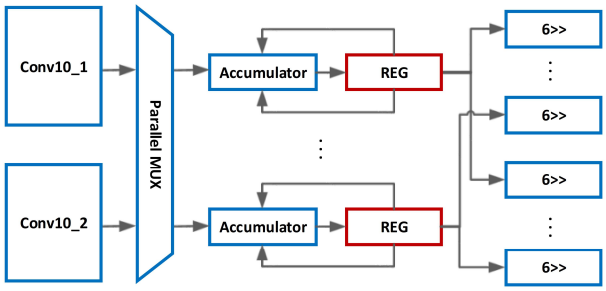

Fig. 3. Average pooling unit implementation

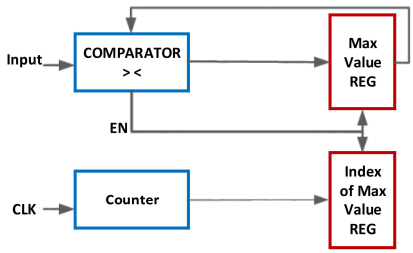

Fig. 4. Output prediction unit implementation

III. Design Improvements and Implementation

After finishing the design implementation and satisfying all processing functionality. Several optimization techniques and improvements are added to the design to improve the acceleration or power consumption. The adopted optimizations are categorized as follows.

A. Area-aware optimizations

1. Control sets reduction

A control set is the grouping of control signals (set, reset, clock enable, and clock) that drives any given SRL, LUTRAM, or register. Designs with fewer control sets have more options and flexibility in terms of placement, generally resulting in improved results, especially for the utilized area. During these optimizations, the goal is to reduce the controlling as possible.

2. Quantization of the fully connected layer

The fully connected layer has the most of ZynqNet CNN model parameters. Hence, quantization of this layer from 16 to 10 bits makes an improvement in the area. Moreover, based on simulation results, quantization of fully connected layer to 10 bits makes the design run at higher frequencies too.

3. Packing of expand layer parameters

Fire modules in ZynqNet consist of a Squeeze and 2 Expand layers. Usually, the Expand layers are working simultaneously. Consequently, instead of allocating a separate memory for each one, packing both Expand parameters in one memory reduces the allocated memory.

4. Registering high fan-out nets

In the presence of tight timing constraints and high area usage, fanout is preferred to be less than 500 cells. The proposed design has all high fanout nets registered, so that replicated cells consume flip-flops and not the LUTs.

B. Power consumption optimizations

DSPs and BRAMs have optional input/output pipelining registers, which are responsible for timing and power optimization. Pushing the pipeline flops or even non-pipelined flops into big blocks improves timing paths. The movable registers are preferred to be placed prior placement with physical optimization.

C. Placement and Routing optimizations

1. Manual replication of signals

Routing congestion normally occurs due to high area usage, tight timing constraints, and high fanout nets. High fanout nets are replicated manually. This resolves the issue, but increases the utilized area. Manual replication of high fanout nets allows the placer to freely replace critical cells and rearrange block placement according to new replicas sites.

2. Removing Global Reset Signal

By default, reset signals are the second-highest fanout signals, with clock signals are in the first place. Therefore, resets should be handled carefully in the huge designs. Unlike clocks, reset signals have no dedicated buffers and global routing, such as BUFG or BUFR depends on local routing resources. FPGAs have embedded global reset that reprograms the chip into the initial state or factory reset. Depending on such utility, routing is totally improved while keeping the same function.

D. Hardware implementation

The synthesis is performed for Xilinx Virtex-7 VC709 FPGA. The synthesis flow requires selecting how the tool handles hierarchy, either full flat hierarchy, none flat, or rebuilt flattening. The none-flattening synthesis and the "AreaOptimized_high" directive lead to best results using additional options such as (None flattening, Resource sharing, Retiming, and Incremental synthesis). After design synthesis, the implementation process is made successfully after performing several physical optimizations with RTL (Register Transfer Language) changes.

IV. Discussion and Results

In this section, area utilization on Virtex-7 FPGA board and power consumption are discussed. In addition, a comparison between the proposed accelerator, Intel Core i7-4510u, and NVidia GeForce840M is presented. Finally, a comparison between the proposed work and other ZynqNet implementations is presented. Firstly, the utilization of Virtex-7 resources is shown in Table I. It is depicted from the table that the BRAMs and DSPs are nearly fully utilized to make full use of FPGA resources and use only on-chip memory for memory storage.

Power consumption is an important metric to analyze the performance of any hardware design. A lot of embedded and IoT applications require a careful design for power consumption requirements. The total power consumption in the proposed design is 10.97W and 11.5W for frequencies 100MHz and 125MHz, respectively. BRAMs and DSPs consume around 60% (6.42W) of the dissipated power. Therefore, most power reduction efforts are focused on these components. On the other hand, energy is a more accurate metric to describe the performance of the design. The energy consumption is 0.88J under a frequency of 100MHz, and 0.736J under a frequency of 125MHz.

In Table II, a comparison between the proposed implementation, Intel Core i7-4510u, and NVidia GeForce840M is presented. It is observed from the table that the proposed implementation overcomes the GPU and the CPU in terms the power consumption and processing speed. The proposed hardware accelerator classifies 15.63 frames per second (fps) at a frequency of 125 MHz, while the GPU classifies 1.067fps at a frequency of 1.029 GHz and 30W. Also, the CPU classifies 0.356fps with 15W at a frequency of 2GHz.

TABLE I. Design Utilization on Virtex-7 FPGA

Resource	Utilized	Available	Utilization
LUT	338922	433200	78.24%
LUTRAM	589	174200	0.34
FF	184723	866400	21.32%
BRAM	1413	1470	96.16%
DSP	3552	3600	98.67%

In Table III, a comparison between the proposed design and the latest implementations for ZynqNet CNN is presented [11-13]. It is obvious that this is the fastest implementation compared to other works with 15.63fps at a frequency of 125MHz. Also, the proposed implementation has the lowest energy consumed per frame and highest power efficiency (frames/Watt). Moreover, the proposed hardware accelerator is fully implemented in native RTL (Verilog), while other implementations are implemented using HLS or SDSoC. Accordingly, the power consumption is large for Gschwend work with 15.6W per frame, and it's expected to very large in Mousouliotis and CNN-Grinder implementations as they are HLS and SDSoC implementations, respectively. For FPGA utilization, the proposed implementation is designed to make full use of all FPGA resources to provide the max performance. As a result, the proposed implementation provides the best performance in terms of number of frames per second and number of frames per Watt.

V. Conclusion

This paper presented the implementation of ZynqNet CNN architecture on FPGA. The adopted approach was to implement all ZynqNet layers on the FPGA to reach the max acceleration and make full use of all DSP units. Various optimizations and approximation techniques were used in different design phases to improve processing speed, utilized area, and power consumption. The proposed accelerator achieved 15.6fps for ZynqNet CNN classification at a frequency of 125MHz. Power consumption was measured as 11.5W at 125MHz, and 10.97W at 100MHz. Moreover, the proposed accelerator run at two different frequencies of 100MHz and 125MHz. In addition, the proposed implementation provided an improvement over benchmark CPU and GPU such as Intel Core i7-4510u and Nvidia GeForce840M. Furthermore, the design overcame previous FPGA implementations in terms of power consumption and processing speed. It was synthesized and implemented on Virtex-7 FPGA. Finally, the achieved classification accuracy was 57.5% on Virtex-7 FPGA.

TABLE II. Comparison between the proposed hardware accelerator, Intel Core-i7, and NVidia GeForce840M

	Intel Core i7-4510u	NVidia GeForce840M	This Work
Frequency	2 GHz	1.029 GHz	125 MHz
Latency (Sec)	2.8125	0.9375	0.064
Power (Watt)	15	30	11.5
Performance (fps)	0.3556	1.067	15.63

Acknowledgment

This work was partially funded by ONE Lab at Zewail City of Science and Technology and Cairo University, Siemens EDA (Mentor Graphics), ASRT, NTRA, and ITAC

References

[1] S. A. Oke, "A Literature Review on Artificial Intelligence," *International Journal of Information and Management Sciences*, vol. 19, no. 4, pp. 535-570, 2008.

[2] O. Russakovsky et al., "ImageNet Large Scale Visual Recognition Challenge," *Int. J. Comput. Vis*, vol. 115, pp. 211-252, 2015.

[3] K. He, X. Zhang, S. Ren and J. Sun, "Deep Residual Learning for Image Recognition," *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, pp. 770-778, Jun. 2016.

[4] H. Elhosary, M. H. Zakhari, M. A. Elgammal, K. A. H. Kelany, M. A. Abd El Ghany, Khaled N. Salama, and H, Mostafa, "Hardware Acceleration of High Sensitivity Power-Aware Epileptic Seizure Detection System Using Dynamic Partial Reconfiguration," *IEEE Access*, vol. 9, pp. 75071-75081, 2021.

[5] M. Abdou, R. Mohammed, Z. Hosny, M. Essam, M. Zaki, M. Hassan, M. Eid, and H. Mostafa, "End-to-End Crash Avoidance DeepIoT-based Solution," *IEEE International Conference on Microelectronics* (ICM 2019), Cairo, Egypt, pp. 103-107, 2019.

[6] S. Khan, H. Rahmani, S. A. A. Shah and M. Bennamoun, "A guide to convolutional neural networks for computer vision," *Synth. Lectures Comput. Vis*., vol. 8, no. 1, pp. 1-207, 2018.

[7] S. Han, X. Liu, H. Mao, J. Pu, A. Pedram, M. A. Horowitz, et al., "EIE: Efficient Inference Engine on Compressed Deep Neural Network," arXiv:1602.01528, 2016.

[8] A. J. Abd El-Maksoud et al., "FPGA Design of High-Speed Convolutional Neural Network Hardware Accelerator, " *IEEE Novel Intelligent and Leading Emerging Sciences conference (NILES 2021)*, Cairo, Egypt, In Press.

[9] E. Adel, R. Magdy, S. Mohamed, M. Mamdouh, and H. Mostafa, "Accelerating Deep Neural Networks Using FPGA," *IEEE International Conference on Microelectronics (ICM 2018)*, Sousse, Tunisia, pp. 180-183, 2018.

[10] A. J. Abd El-Maksoud, M. Ebbed, A. H. Khalil and H. Mostafa, "Power Efficient Design of High-Performance Convolutional Neural Networks Hardware Accelerator on FPGA: A Case Study with GoogLeNet," in *IEEE Access*, doi: 10.1109/ACCESS.2021.3126838, Nov. 2021.

[11] D. Gschwend, "ZynqNet: An FPGA-Accelerated Embedded Convolutional Neural Network," vol. Master ETH-Zurich:Swiss Federal Institute of Technology Zurich, 2016.

[12] P. G. Mousouliotis and L. P. Petrou, "Software-defined FPGA accelerator design for mobile deep learning applications," arXiv:190203192, 2019.

[13] P. G. Mousouliotis and L. P. Petrou, "CNN-Grinder: From algorithmic to high-level synthesis descriptions of CNNs for low-end-low-cost FPGA SoCs," *Microprocessors Microsyst.*, vol. 73, Mar. 2020.

TABLE III. Comparison between the proposed hardware accelerator with other ZYNQNET FPGA implementations

		Gschwend [11]	Mousouliotis [12]	CNN-Grinder [13]	This work v1	This work v2
FPGA		Zynq XC-7Z045	Zynq-xc7z020	Zynq-xczu9eg	Virtex-7	
Design Type		HLS	HLS	SDSoC	Full RTL design	
Memory		On/Off-chip	On/Off-chip	On/Off-chip	On-chip	
Precision		16-bit	Floating point 8-bit	Floating point 8-bit	15-bit	16-bit
FC layer precision		16-bit	Floating point 8-bit	Floating point 8-bit	15-bit	10-bit
Frequency (MHz)		200	100	100	100	125
Power (W)		7.8	-	-	10.97	11.5
Performance (fps)		0.5	11.54	14.49	12.5	15.63
Inference time (Sec)		2	0.0866	0.069	0.08	0.064
Power Efficiency (frames/W)		0.0641	-	-	1.139	1.359
Energy (frame/J)		15.6	-	-	0.88	0.736
	LUTs	154K	36.2K	34.5K	339K	283K
	BRAMS	1090	96.5	97.5	2130	2130
	DSPs	739	172	164	3552	3552