FPGA Design of High-Speed Convolutional Neural Network Hardware Accelerator

Ahmed J. Abd El-Maksoud[1], Abdallah Mohamed[1], Ahmed Tarek[1], Amr Adel[1], Amr Eid[1], Farida Khaled[1], Fatma Khaled[1], Ziad Ibrahim[2]
Eman El Mandouh[2], and Hassan Mostafa[1,3]

[1]Electronics and Electrical Communications Engineering Department, Cairo University, Giza, 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

Abstract*—Convolutional Neural Networks get increasingly importance nowadays as they enable machines to interact with the surrounding environment, which paves the way for computer vision applications. FPGA implementations of CNN architectures have higher speed and lower power consumption compared to GPUs and CPUs. This paper proposes a high-speed hardware accelerator on FPGA for SqueezeNet CNN to accelerate its processing without decreasing the classification accuracy. Several ideas are applied to solve the memory bottleneck issue such as using Ping-Pong memory and deploying several FIFOs in the design. The architecture is built as a pipelined unit to process SqueezeNet CNN layer by layer. Different parallelism techniques are applied while processing the convolution layers to speedup layers processing. Moreover, the proposed accelerator classifies 248.76 fps at a frequency of 100MHz, and 427.4 fps at a frequency of 172 MHz. The proposed accelerator is implemented on Virtex-7 FPGA, and overcomes Geforce RTX 2080Ti GPU and several SqueezeNet FPGA implementations.

Keywords*—Convolutional Neural Networks (CNNs); FPGAs; Hardware Accelerators; SqueezeNet.

I. INTRODUCTION

Artificial Intelligence plays a vital role in bridging the gap between the capabilities of humans and machines [1]. It uses the knowledge for a multitude of tasks such as image and video recognition, image classification, biomedical applications, and autonomous vehicles [2-4]. The advancements in computer vision with deep learning have been evolved with time. Convolutional Neural Network (CNN) is one of these popular algorithms which has become a state of art methodology for image recognition and is used successfully in object classification, and face detection [5-7].

CNNs typically have millions of learnable parameters which are initialized firstly. Then, they are evaluated through training iterations till getting the desired accuracy. The evaluated weights are saved for the testing or inference phase where they are used to classify the images immediately. CNN requires huge resources and billions of computations for a single frame. There are a lot of proposed architectures to accelerate the CNN processing using FPGA [8-10].

Convolutional neural networks are constructed by stacking different types of neural layers, such as convolution layers, fully connected layers, and pooling layers. CNN structure usually consists of feature extractor and classifier. Firstly, the feature extractor which extracts frame features across the CNN layers which are Convolutional layers or pooling layers. Secondly, the Classifier, which is implemented using fully connected layers. It computes every class score and decides the output class. Convolution layers are constructed with weights and biases and they are grouped as the model parameters.

Recent research on deep learning is focused primarily to achieve higher accuracy which usually increases the model size. On the other hand, smaller CNN architectures offer at least three advantages while maintaining the same accuracy. Firstly, it requires less time during the training process. Secondly, it requires lower memory storage. Finally, it is more feasible to be deployed on FPGAs.

SqueezeNet is one of the popular CNN architectures [11]. SqueezeNet uses 50x fewer parameters than AlexNet with the same accuracy level on ImageNet dataset. SqueezeNet is designed with three main strategies:

i. Replace 3x3 filters with 1x1 filters so the parameters are decreased by 9X.
ii. Decrease the number of input channels that is used as input to 3x3conv layers in Expand layer to maintain the number of parameters in the CNN model.
iii. Use down-sampling in the network so that the convolution layers have large activation maps, which leads to higher classification accuracy.

As depicted from the above three strategies. Strategy i and ii decrease the number of parameters in the CNN model while attempting to keep the accuracy. On the other hand, Strategy iii maximizes the accuracy with a limited budget of parameters.

SqueezeNet simply begins with a convolution layer (conv1), followed by 8 Fire modules (fire2–9), and ends with a final convolution layer (conv10) as shown in Fig. 1. Furthermore, Maxpooling layers with a stride of 2 are added after conv1, fire4, fire8, and conv10 layers.

SqueezeNet uses only 3x3 and 1x1 convolution kernels in the fire module. The 1x1 filters are used to shrink the input feature map size to 3x3 filters. It reduces the computation of the following 3x3 convolution layers. Consequently, this technique enables SqueezeNet to use 50x fewer parameters. In addition, it achieves the same accuracy as AlexNet. SqueezeNet uses an average pooling layer to calculate classification scores instead of the fully connected layers. Adding an average pooling layer reduces the number of computations and memory storage.

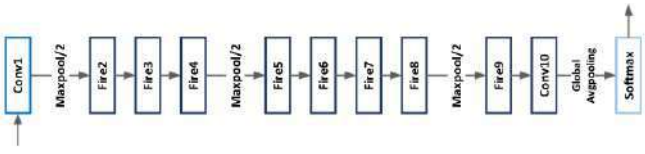

Fig. 1. SqueezeNet CNN [11]

978-1-6654-2157-7/21/$31.00 ©2021 IEEE

Hardware accelerators have become one of the popular topics in research. There are a lot of different implementations especially for FPGA implementation. For SqueezeNet accelerators implementations, SqueezeJet accelerator is a High-Level Synthesis (HLS) design for convolutional neural networks. It achieves 4.36x speedup more than Intel Core-i3 for squeezeNet at 100MHz frequency [12]. Moreover, the Edge-net accelerator processes 9 frames per second (fps) at 100MHz frequency [13]. It's implemented on DE10-Nano with an accuracy loss of 2%.

This paper is organized as follows, Section II presents the proposed architecture. Section III shows the implementation in detail. Moreover, Section IV discusses the implementation results. Finally, Section V concludes the work.

II. The Architecture

The design purpose of this architecture is to build a well-designed block using available FPGA resources and reuse the block in different fire modules. The proposed architecture is shown in Fig. 2. The architecture is pipelined architecture which processes SqueezeNet CNN layer by layer. Consequently, SqueezeNet layers such as Conv1 layer, eight Fire modules, and Maxpooling are computed iteratively. The accelerator is initially implemented in 16-bit fixed-point representation, but it's reduced to 13-bit precision to save FPGA resources while keeping the inference accuracy.

Each layer is running for a specific number of clock cycles depending on the executed fire module in this stage. The output of the squeeze layer is stored in the FIFO before loading it as input to the Expand layers. Alternately, the output of the expand layers is stored in the data memory, which is fetched again as input for the next stage and so on.

The processing of each squeeze layer is accelerated by processing 8 filters across 16 channels in parallel. Moreover, the parallelism in expand layer is done by processing 16 Filters across 8 channels at the same time. It is worth mentioning that squeeze layers always have a high number of input feature map (IFM) channels (64, 128, 256, 384, 512) and low number of filters (16, 32, 48, 64). On other hand, the expand layer has a low number of input channels (16, 32, 48, 64), but with a high number of filters (64, 128, 256, 384, 512). Therefore, choosing different parallelism methods in each of them is required.

A. Data Memory

The data memory is designed to have multiple ports equal to the number of channels fed to the adder tree in one cycle. In this case, the number of ports is 16. In order to achieve this huge number of ports with block RAMs, the data is divided across several block RAMs. Every clock cycle 16 data ports are read from the memory.

The expand stage writes 16 activation values to the memory in one cycle where the 16 values are divided equally between expand 1x1 and expand 3x3 blocks. The first 8 activation values come from expand 1x1 and the second 8 values from expand 3x3. Both expand 1x1 and expand 3x3 blocks write in different locations in the memory. Accordingly, we use dual-port block RAMs where each port is independent of the other.

B. Weights Memory

The weights memory has the same distribution as the data memory for one filter. In this design, 8 filters run in parallel, so that 8 copies of the same weight are generated, but

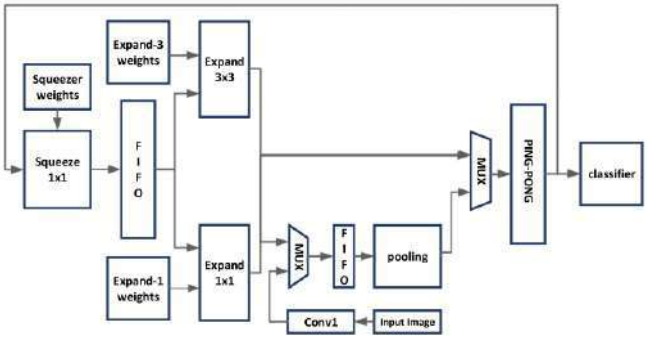

Fig. 2. The proposed Architecture

with different initialization for each filter. Weights are stored in ROMs that are implemented using either BRAMs or LUTs. The BRAM approach is chosen for layers with a lot of weights such as expand1x1 and expand3x3. The LUTs approach is better for low depth memories so LUTs are used to store the weights for the squeeze layer. On the other hand, squeeze weights are distributed into different memories which enables parallel data reading to be made every clock cycle. There are 16 Memories for each row. Hence, there are 16 values read from the squeeze module. Moreover, Expand Layer parallelism is applied by processing 16 filters across 8 channels. There are 10 expand weights required for the same channel every clock cycle. Memory is divided between Expand1x1 filters and Expand3x3 filters with two separate memory address ports.

C. Ping-Pong Memory

The structure of Ping-Pong memory is shown in Fig. 3. Every clock cycle, 16 data ports are read from the memory by the squeeze unit. The memory is used for FIRE-2, FIRE-4, FIRE-6, FIRE-7, and FIRE-8 layers in addition to the pooling layer. When a fire stage starts, the squeeze layer reads data from memory A and processes it. After that, the expand layer processes the output of the squeeze layer and stores the data into memory B. In the next fire stage, the squeeze layer reads the data from memory B while the expand layer stores it in memory A. This alternating process is achieved using Ping-Pong memories where the control signal decides which of the two memories will receive the input data along with the required signals such as the address and write enable.

D. The Intermediate FIFO

The intermediate FIFO holds the output data of the squeeze stage and passes it to the expand stage. Once the squeeze finishes processing a pixel, a new pixel gets stored in the buffer. After the buffer gets filled with *W+2* entries the expand stage starts processing the data in the buffer. FIFO is chosen to store the intermediate results between the modules because it is easily implemented by a chain of registers and required data is accessed easily (No addresses). Since the FIFO size is 2W+3, so all pixels of a 3x3 window are fetched at the same time after filling the whole FIFO (2w+3 cycles) as shown in Fig. 4.

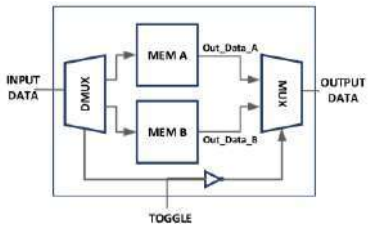

Fig. 3. Ping-Pong Memory

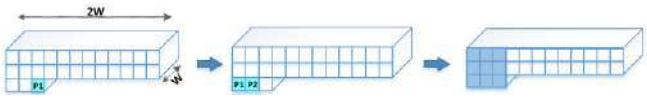

Fig. 4. FIFO Dataflow

III. Implementation Insights

In this section, several implementation insights are presented such as CNN software model training, design improvements, and validation methodology.

A. CNN Model Training

The SqueezeNet model is trained for internal building security application. Therefore, the model is trained on 10 classes from ImageNet dataset. These classes are different types of weapons. The top-1 classification accuracy of the software model is around 72%.

B. Implementation Improvments

The accuracy of the hardware model is usually a little lower than the software model. However, the actual difference between the calculated accuracy and the software model accuracy is very large. After analyzing the issue, it is found that the hardware, which uses fixed-point representation, overflows causing the numbers to wrap around. Accordingly, a saturation limit is used on the numbers to prevent wrapping which improved the hardware accuracy. The second modification is to use dynamic fixed-point representation to change the fraction part width across layers. After several experiments, it is found that earlier layers need fewer bits for the integer part and more for the fraction part and vice versa for the last layers. This improves the hardware accuracy, but it is still away from the software accuracy. The third modification is made by retraining the model on clipped ReLU, then it's applied on the hardware level. This makes the hardware and software models have almost the same accuracy. Finally, the inference accuracy is tested versus the precision to check if the later one can be reduced. It's found that the inference accuracy started to reduce from 12-bit, so a precision of 13-bit is selected while keeping the inference accuracy and reducing the FPGA utilizations by 3% LUTs, 3% REGs, and 7% BRAMs.

Firstly, the HDL design is implemented on Virtex-7 series FPGA on 100 MHz with minimum positive slack. The goal is to increase the frequency to 200 MHz, but several issues are faced during the design such as fanout, wire delay, and floor planning. Firstly, the fanout problem is created by increasing design parallelism. Parallelism leads to high fanout which causes high cell and net delays. High fanout leads to large wire delay, so buffers are inserted to reduce the delay. The tool is constrained for buffers insertion to reach the required frequency. Secondly, low utilization makes the tool spread the logic cells over the FPGA to avoid creating congestion regions. This makes the blocks to be allocated far away from each other which creates long routes and high wire delays. Consequently, custom FPGA floor planning is used to place the blocks near to each other, which decreases the wire delay and improves the frequency.

C. Verification Methodology

Comparing the implemented design on FPGA with the software model is important to make sure of the hardware results. MATLAB model is implemented to test the HDL design. Firstly, it is used to compare the intermediate results of HDL design with the software model to check the equivalence of the results. Also, test benches are created and executed for each module. Finally, a top-level test bench is used after the final integration between all accelerator modules to validate the results.

IV. Discussion and Results

In this section, design synthesis, the processing time for each layer, and power consumption are discussed. In addition, a comparison between the proposed accelerator and GPU of Geforce RTX 2080Ti is presented. Finally, a comparison between SqueezeNet implementations and the proposed work is investigated.

The synthesis is performed on Xilinx Virtex-7 VC709 FPGA. It is noted that the FPGA resources are mostly utilized by the main FIFO, pooling block, and the Fire blocks. Fire blocks have most of the BRAMs and DSPs due to large weights storage and multipliers count.

Power consumption is an important metric to analyze the performance of any hardware design. A lot of embedded applications require a careful design with specific power consumption requirements. The total on-chip power in the proposed design is 8.9W. BRAMs and DSPs consume around 60% (5W) of dissipated power. Therefore, most power reduction efforts are focused on these components. These results are obtained at a frequency equals to 100 MHz. on the other hand, energy is a more accurate metric to describe the performance of the design. The energy consumption is 35.78mJ under an operating frequency of 100 MHz. Expand block is the most power-consuming unit. As the expand block has the most DSPs and weights ROMs which have the most computations.

In Table I, the processing time is recorded for each layer at two different frequencies of 100MHz and 172MHz. It is noticed that Conv1 layer is a bottleneck layer. The high-speed performance of the proposed accelerator is attributed to the applied parallelism in addition to overlapping between layers. The expand layer is processed while the squeeze layer still running and the pooling layer begins while squeeze and Expand are running. Moreover, the FIFO module enables the start of Expand or pooling layers even if the complete output of squeeze module is not loaded yet, as explained before.

In Table II, a comparison between the proposed implementation and GeForce RTX 2080Ti GPU is presented. The proposed hardware accelerator is compared with two versions using a 13-bit fixed-point representation with a dynamic clipped ReLU activation function at two different operating frequencies. It is observed from the table that the proposed implementation has better power consumption results for both frequencies. The proposed design classifies

TABLE I. Processing time of each layer in SqueezeNet

Layer	**This work** (100 MHz)	**This work v2** (172 MHz)
	Processing Time (µs)	
Conv1 & pooling1	1021.725	595.328
Fire2	255.805	149.05
Fire3 & pooling2	511.325	297.933
Fire4	260.765	151.939
Fire5 & pooling3	521.245	303.713
Fire6	199.955	116.5
Fire7	294.045	171.33
Fire8	397.075	231.36
Fire9	522.525	304.46
Conv10	32.035	18.67
Total	**4.016 ms**	**2.34 ms**

427.4 frames per second (fps) at a frequency of 172 MHz while the GPU classifies 331.1 fps at frequency of 1.545 GHz. Also, the classification accuracy is less than GPU by 0.03% only which is a good hardware result.

Another comparison is presented between the proposed work and the latest implementations for SqueezeNet CNN on different FPGAs. As shown in Table III, it is clear that this implementation is the fastest compared to [13] and [14] work with 248.76 fps with 100 MHz frequency. Although the processor consumes more power, it has the best energy consumption results. Moreover, the classification accuracy is higher than other works with 69.6% top-1 accuracy. Finally, the number of classes is 10 classes in the proposed work and 1000 classes for other implementations. However, the SqueezeNet model will not change except for Conv10 layer and Softmax layer. The Softmax processing time and power consumption are negligible compared to other SqueezeNet layers. In addition, the Conv10 layer processing time is 32.035µs and 18.67µs at a frequency speed of 100MHz and 172MHz, respectively. If the number of classes is increased to 1000, the processing time for Conv10 is estimated to be 7.19ms with 138.99fps at a frequency of 100MHz, and 4.2ms with 237.73fps at frequency 172MHz. Consequently, the proposed work still overcomes [13] and [14]. It is worthy to mention that parallelism can be increased to speed up the processing. Moreover, the proposed work is implemented especially for a building security application with 10 classes, and there are no FPGA implementations for SqueezeNet with only 10 classes.

V. Conclusion

This work proposed a design of a high-speed hardware accelerator on FPGA. The accelerator was designed to process SqueezeNet CNN. Several ideas were applied to solve the memory bottleneck problem such as using Ping-Pong memory and deploying several FIFOs in the design. The architecture was built as a pipelined unit using available FPGA resources. After that, this block was used to process different fire modules/layers of SqueezeNet in sequence. Moreover, different parallelism techniques were applied while processing the convolution layers to speedup layers processing. SqueezeNet was firstly trained for the application of internal building security with 10 classes. Fixed-point representation overflow was solved by using clipped RelU and dynamic fixed-point representation to keep the classification accuracy. Processing speed was the main goal without increasing the utilized area or decreasing classification accuracy. In addition, the proposed accelerator consumed 8.9W and classified 248.76 fps at 100 MHz, and 427.4 fps at 172 MHz. The proposed accelerator gave a higher processing speed (fps) over Geforce RTX 2080Ti at 172 MHz. Moreover, it overcame previous SqueezeNet FPGA implementations in the performance (fps) and classification accuracy.

TABLE II. Comparison between two versions of the proposed implementation and GeForce RTX 2080Ti GPU

	This work	This work v2	GeForce RTX 2080TI
Frequency	100 MHz	172 MHz	1.545 GHz
Latency (ms)	4.02	2.34	3.02
Power (Watt)	8.9	17.4	55
Performance (fps)	248.76	427.4	331.1
Top-1 Accuracy	69.6%	69.6%	69.9%

TABLE III. Comparison between different implementations for SqueezeNet on FPGA

		[13]	[14]	This work
FPGA		De10 board	Zynq7020	Virtex-7 V709
No. of Classes		1000	1000	10
Frequency (MHz)		100	-	100
Power (Watt)		2	7.95	8.9
Time (ms)		110	1030	4.02
Energy (mJ)		220	8,188	35.78
Top-1 Accuracy		55%	57.5%	69.6%
Performance (fps)		9.1	1	248.76
Utilizatio	**BRAMS**	-	80%	61.77%
	DSPs	-	95%	73.8%
	FF	-	48%	15.47%
	LUTs	-	102%	19.32%

Acknowledgment

This work was partially funded by Mentor Graphics and ONE Lab at Zewail City of Science and Technology, Egypt and Cairo University, Egypt.

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] SR. Ke et al., "A Review on Video-Based Human Activity Recognition," *Computers*, vol. 2, no. 2, pp. 88-131, 2013.

[3] 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.

[4] 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.

[5] K. Annapurani and D. Ravilla, "CNN based Image Classification Model," *IJITEE*, vol. 8, no. 11, pp. 1106-1114, 2019.

[6] A. Krizhevsky, I. Sutskever and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, vol. 60, no. 2, pp. 84-90, Jun. 2012.

[7] A. Khan, A. Sohail, U. Zahoora and A. S. Qureshi, "A Survey of the Recent Architectures of Deep Convolutional Neural Networks," *arXiv:1901.06032*, 2019.

[8] 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.

[9] M. Motamedi, P. Gysel, V. Akella and S. Ghiasi, "Design Space Exploration of FPGA-based Deep Convolutional Neural Networks," *Proc. IEEE Asia South Pacific Design Auto. Conf. (ASP-DAC)*, pp. 575-580, Jan. 2016.

[10] R. Osama and H. Mostafa, "Implementation of Deep Neural Networks on FPGA-CPU platform Using Xilinx SDSOC," *Springer Analog Integrated Circuits and Signal Processing*, vol. 106, pp. 399-408, 2021.

[11] F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally and K. Keutzer, "SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and < 0.5 MB Model Size," *arXiv:1602.07360*, 2016.

[12] P. G. Mousouliotis and L. P. Petrou, "SqueezeJet: High-level Synthesis Accelerator Design for Deep Convolutional Neural Networks," *14th International Symposium ARC 2018*, pp. 55-66, May 2018.

[13] K. Pradeep, K. Kamalavasan, R. Natheesan and A. Pasqual, "Edgenet: Squeezenet Like Convolution Neural Network on Embedded FPGA," *25th IEEE International Conference on Electronics Circuits and Systems (ICECS)*, pp. 81-84, 2018.

[14] M. Arora and S. Lanka, "Accelerating SqueezeNet on FPGA," [Online] Available: https://lankas.github.io/15-618Project/