Low power CNN hardware FPGA implementation

Sherry Hareth
School of Electronics and
Comunication Engineering
Arab academy for science and
technology and maritime transport
Cairo, Egypt
Email:sherry_heshmat@yahoo.com

Hassan Mostafa
Department of Electronics
and Communications Engineering,
Cairo University, Egypt
University of Science and technology
Nanotechnology and Nanoelectronics Program
Zewail City of Science and Technology
October Gardens
6th of October, Giza 12578, Egypt
Email: hmostafa@uwaterloo.ca

Khaled Ali Shehata
School of Electronics and
Comunication Engineering
Arab academy for science and
technology and maritime transport
Cairo, Egypt
Email: Khaledshehata58@gmail.com

***Abstract*—A convolution Neural Networks (CNN) goes under the wide umbrella of Deep Neural Networks (DNN) whose applications are widely used. For example, the later are used in robotics and different applications of recognition like speech recognition and facial recognition, also nowadays in autonomous cars. Therefore the aim of implementing the CNN is to be used in real time applications. As a result of that, Graphics processing units (GPUs) are used but their worst disadvantage is it's high power consumption which can't be used in daily used equipments.**

The target of this paper is to solve the power consumption problem by using Field Programmable Array (FPGA) which has low power consumption, and flexible architecture. The implementation architecture of Alex Network, which consists of three fully connected layers and five convolution layers, on FPGA will depend on two main techniques parallelism of resources, and pipelining inside of some layers.

***Keywords*—Convolution (Conv), Fully Connected(FC), Feature maps (Fmaps), Rectified Linear Unit (ReLU), maximum pooling (maxpooling), Multiplication And accumulation (MAC) operation, Parallel Engine (PE), Local Response Normalization (LRN), Register Files (RF), Partial Sum Buffer (PSUM).**

I. Introduction

The target of modern digital systems towards eliminating direct programming and creating an intelligent system that can automatically adapt to new situations, is the job of the Artificial Intelligence (AI)[1], and Deep Learning (DL) nowadays. CNNs are one of the DL algorithms used for many applications such as image processing[2], face recognition [3],[4], autonomous cars [5], and robotics [6]. Due to the large parameters and huge amount of resources needed to be computed in CNN, the GPUs are used because they are known for their high throughput, large band width on-chip, and off-chip [7]. In addition to their ability to manage a massive parallelism, and large data reuse, they are also used to improve both training, and classification processes of CNNs [8]. On the other hand, the main disadvantage of using the GPUs is consuming large amount of power which is an important evaluation metric for modern digital systems. For example, Autonomous cars need high energy efficiency and real time performance. Therefore the direction towards using new FPGA generations instead of GPUs is highly preferred as they provide superior energy efficiency (Performance/Watt) than GPUs for DNNs [7]. Moreover, the new FPGA generations are famous with their capability of handling large hardware resources, and computing a huge number of floating points units. Implementation of CNN on FPGA is considered as a state of art of a designer for example in [7] the authors offered a case study on accelerating Ternary ResNet to compare between the usage of GPU, and new FPGA generation performance for DNNs. The later enhances the performance by 10%. Another way of implementing CNN is presented in [9] using a technique called row stationary which maximize the reuse, and accumulation in the local memory level for all types of data.

This paper is organized as follows; in Section II Background overview on CNN is presented. Section III Describes the framework used in the hardware implementation. Section IV Shows the hardware utilization results after performing the synthesis. Section V Represents pipelining. Section VI Concludes the proposed implementation .

II. Background

CNNs have two operational phases training phase and inference phase. The training phase is done on a known data set to learn the weights for minimizing the errors. The inference phase has a feature extractor and a classifier. AlexNet [10] consists of five Conv layers and three FC layers while the last FC layer is fed to a 1000-way Softmax which produces a distribution over the 1000 class labels. A non-linearity unit which is ReLU is used in each layer. Also maxpooling is applied to the outputs of layers 1, 2, and 5. To sum up, AlexNet requires 61 million weights and 724 million MACs to process one (227×227) input image. At first, the convolution layers are considered as the heart of AlexNet because of their main role which depends on MAC operations represented by a unit called PE shown in figure (1-(a)). The MAC operation are done by multiplying the input data of the image (input Fmaps) element by element with the trained weights which are called kernels filters, and accumulating operation is done as shown in figure (1-(a)) [6] . The output of the convolution layers are feed into

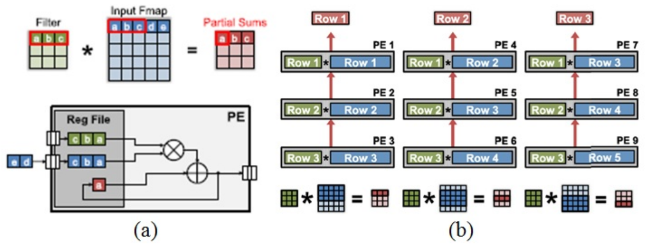

Fig. 1. (a)PE (b)Two-dimensional convolutional reuse within spatial array for row stationary dataflow[7]

output Fmaps after sliding different Kernels filters with strides given within each convolution layers. Secondly, ReLU layers are usually placed after each convolution layer. They are used as an introduction to a non-linear operation. This non-linear operation makes the network adapt with any given data sets. ReLU clamps all negative values to 0, and returns all positive values. Thirdly, LRN layers are placed after ReLU layers. They are used to normalize the response of the neurons across the depth of the same spatial location, which speed up the training process, and enhance the system accuracy. Fourthly, maxpooling layers are used to reduce the dimension of each Fmaps, and return the most important feature information.

III. Hardware architecture

The goal of the introduced architecture is to reduce the time taken for AlexNet to classify an image which means accelerating the CNN. In this section, the layers of AlexNet architecture implementation are presented in the inference phase. The technique used to handle the dataflow is row stationary and as mentioned in section (II) that the most important unit in our design is PE which acts as a neuron. Figure(1-(a)) shows the PE unit which consists of two RF to store the input as well as the current filter, and MAC operation to compute the partial sums that requires just one memory space. Since there are overlaps of input activations Fmaps between different sliding windows, the input can then be kept in the RF and get reused. With each PE processing a 1-D convolution, multiple PEs can be aggregated to complete the 2-D convolution as shown in figure (1-(b)). Therefore the main computational power in this design lies upon a matrix of PEs used in Conv layers. One of the main features of the design is using an array of 168 PEs that are mapped as a matrix of (14×12) PEs to compute the output of all layers.

A. *Convolution Layers*

1) First Convolution Layer: The input of this layer is the input image to the network, and it's size is $(227 \times 227 \times 3)$. This layer consists of 96 filters each having a size of $(11 \times 11 \times 3)$ with stride 4. Therefore to compute one row, 11 PEs should be stacked together vertically. The input is divided into 4 parts; each part consists of 63 rows, and is stored in a swapping 1 buffer. This will produce 14 rows of the output feature maps when convolved with the filters. The filters are divided into 6 groups of 16 filters. Those filters will be stored with their corresponding biases in the filter buffer. So, the matrix of the PEs should be (11×14). Each PE holds 16 rows of the same depth from 16 different filters in one of the RFs and 1 row of the input of that depth in the other RF. The outputs of the convolution of each filter row and input row are summed vertically to produce the partial sums of one row of one of the output feature maps. Due to the presence of 3 depths, we have to store these partial sums in PSUM buffer. The size of this buffer should be $(14 \times 55 \times 16)$.

2) Second Convolution Layer: The input of this layer is the output of the first LRN Layer with size $(27 \times 27 \times 96)$. It is stored in swapping buffer 2. This layer consists of 256 filters, each having a size of $(5 \times 5 \times 48)$, and with stride 1. The new feature added to this layer is that the input and the filters are divided into two groups; the first group of filers consists of the first 128 filters and they get convolved with the first half of the number of depths of the input (i.e. first $(27 \times 27 \times 48)$), the second group acts as the first one. The zero padding technique is used here to ensure that the output feature maps spatial dimensions are the same as those of the input feature maps (i.e. (27×27)). So, to compute one row, 5 PEs should be stacked together vertically. The filters of each part are divided into 9 batches of 15 filters except the ninth batch which contains 8 filters only. Those filters will be stored with their corresponding biases in the filter buffer. To accelerate the design, the matrix of the PEs in layer 2 will be considered as (10×14) where each output pixel will be the output of two depths from the same filter. In this layer, 8 depths of the filter, and input are stored in the PE each time. Size of PSUM buffer is $(27 \times 27 \times 15)$.

3) Third Convolution Layer: The input of this layer is the output of the second LRN layer with size $(13 \times 13 \times 256)$. It is stored in swapping buffer 2. This layer consists of 384 filters, each having a size of $(3 \times 3 \times 256)$. Stride is 1, while the zero padding technique is also used in this layer. To compute one row, 3 PEs should be stacked together vertically. The input is divided into 4 parts; each part has 64 depths. The filters are divided into 48 groups of 8 filters each. So, the matrix of the PEs should be (3×13). Each PE holds 1 row of the same filter from 16 different depths in one of the RFs and 1 row of the input corresponding to each of those 16 depths in the other RF. The size of the PSUM buffer is $(13 \times 13 \times 16)$.

4) Fourth Convolution layer: The input of this layer is the output of the previous Conv layer, so its size is $(13 \times 13 \times 384)$. It is stored in swapping buffer 1. This layer consists of 384 filters, each having a size of $(3 \times 3 \times 192)$ with stride 1, and the zero padding. The filters are divided into two groups; the first group consists of the first 192 filters, and they get convolved with the first half of the number of depths of the input (i.e. first $(13 \times 13 \times 192)$), the second group acts as the first one. The filters of each part are divided into 12 batches of 16 filters each. Those filters will be stored with their corresponding biases in the filter buffer. So, to compute one row of the output, 3 PEs should be stacked together vertically, and their matrix should be (3×13). Each PE from every part of the 4 parts will contain 16 depths of one filter. The size of PSUM buffer is $(13 \times 13 \times 16)$.

5) Fifth Convolution Layer: The input of this layer is the output of the previous Conv layer with size $(13 \times 13 \times 384)$. It is stored in swapping buffer 2. This layer consists of 256 filters, each having a size of $(3 \times 3 \times 192)$. The stride is 1, and the zero padding technique is also used. The input, and the filters are divided into two groups; the first group of filters consists of the first 128 filters, and they get convolved with the first half of the number of depths of the input (i.e. first $(13 \times 13 \times 192)$), the second group acts as the first one. The filters of each part are divided into 8 batches of 16 filters each. Those filters will be stored with their corresponding biases in the filter buffer. So, the matrix of the PEs should be (3×13). So, to compute one row, 3 PEs should be stacked together vertically. When the first 3 rows are stored, no other PEs are activated in the same column at this time, as they will be used for the storage of another depth to keep the vertical summation valid, meaning that no extra hardware has to be added to that used in layer 1.

B. Pooling Layers

There are three maxpooling layers (pool1, pool2, and pool5) each has a window of (3×3), and a stride of 2. The whole feature map of any depth must be ready to perform the pooling successfully; this is valid in the second through fifth layers as the sizes of the feature maps are relatively small. For the first layer the convolution is divided into several iterations that will allow the maxpooling layer to receive nearly a quarter of feature map. Therefore to stride in the vertical direction in the last 2 rows of the part of the feature map present, the pooling is divided into two parts. The first part contains the maxpooling of the last two rows. This is equivalent to a (2×3) window, and the remaining (1×3) is taken from the first row from the part of the feature map ready during the next iteration. The last step is to choose from the maximum of both parts.

C. Local Response Normalization (LRN) Layer

The schematic of LRN shown in figure (2) is describing the mathematical flow of it's normalization formula. Each block in the schematic is terminated by a register to store the output of the block for duration of one clock cycle. The advantage of this architecture is to break the long 1 cycle path of the block to a multi-cycle path with a much less cycle period. As a result of that, the frequency of the used clock is increased. Another advantage is to allow the use of pipelining through this multi-cycle path, which allows boosting the throughput of the whole block by 9 times. The multi-cycle path of the block consists of 18 cycles; 2 cycles for the input register and the first adder, respectively. The remaining 16 cycles are for the division block. The adders and multipliers used are implemented automatically as LUTs and DSPs, respectively, on the FPGA.

D. Fully Connected Layers

The input to the fully connected layers needs to be a vector so reshaping is done to the output of the maxpooling layer 5 $(6 \times 6 \times 256)$ to make it a vector of (9216×1).

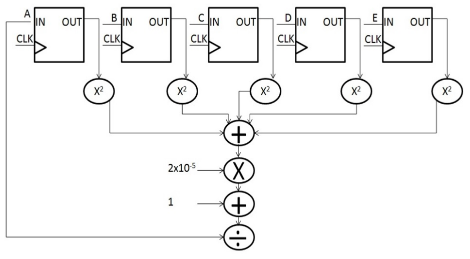

Fig. 2. LRN schematic

1) Fully Connected layer 6: The input size of this layer is 9216 pixels, which is stored in swapping buffers. There are 4096 filters each of 9216 weights, are stored in the swapping buffer with their biases. When storing in the PEs, the caches are divided into two groups. The first group stores the input pixels where each cache is filled with 256 pixels of the input pixels so the whole input needs 36 PEs to be stored. The input is stored 2 times to exploit a total of 72 PEs except for the last 2 columns of PEs which are not used. The second group stores the same way like the first one.

2) Fully Connected layer 7: The input size of this layer is 4096 pixels which is stored in the swapping buffer. There are 4096 filters each of 4096 pixels; only 3 filters are stored in the swapping buffer as well as the 4096 biases. Each feature map cache is filled with 171 pixels of input pixels, so the whole input needs 24 PEs to be stored but the last feature map is only filled with 163 pixels of input, therefore the input is stored 3 times.

3) Fully Connected layer 8: It is exactly the same in its operation as FC 7, and 8 except that the number of filters here is 1000 filters. It computes the class probability by outputting a vector of 1000 dimensions, where 1000 being the number of classes. Estimation is done to the output of this layer to determine the class of the input which is the class with the highest probability score.

E. Memory Hierarchy

Overall, there are three levels of memory hierarchy in the system as shown in figure (3) to decrease energy per access which are DRAM (SD card), GLOBAL BUFFERS (Filter, PSUM, Swapping1, and Swapping 2), and Inter-PE CACHES. DRAMs are used to store all the weights, and bais of the network. The Inter-PE CACHES are used for storing the part of the input, and weights for local reuse.

IV. Synthesis

The utilization of the resources after performing the synthesis is shown in tables (I), (II), and (III) for the discussed layers of AlexNet. The implementation is done on Zynq ZC-702 FPGA using the Vivado 2015.1 synthesis tool. Form the power consumption point of view the results show that the consumed on chip dynamic power is 59%, while the static power consumed is 41%.

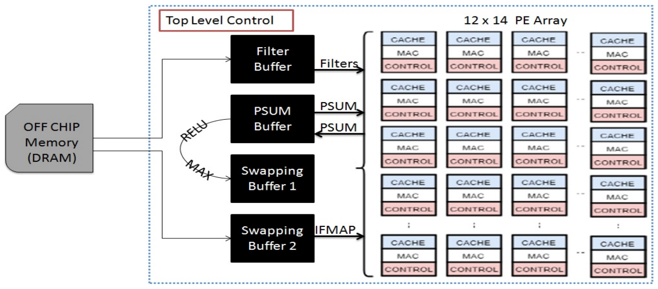

Fig. 3. Memory Hierarchy

TABLE I
FPGA UTILIZATION FOR LOGIC GATES AND LUT

Site type	Used	Fixed	Available	Util %
Slice LUTs	31943	0	53200	60.04
LUT as Logic	21159	0	53200	39.77
LUT as Memory	10784	0	17400	61.98
LUT as Distributed RAM	10752	0		
LUT as shift Register	32	0		
Slice Registers	8044	0	106400	7.56
Registers as Flip Flop	7718	0	106400	7.25
Registers as Latch	326	0	106400	0.31
F7 Muxes	5394	0	26600	20.28
F8 Muxes	2688	0	13300	20.21

TABLE II
FPGA MEMORY UTILIZATION

Site type	Used	Fixed	Available	Util %
Block RAM Tile	126	0	140	90.00
RAMB36/FIFO	120	0	140	85.71
RAMB36E1 only	120			
RAMB18	12	0	280	4.29
RAMB18E1 only	12			

TABLE III
FPGA DSP UTILIZATION

Site type	Used	Fixed	Available	Util %
DSPs	217	0	220	98.64
DSP48E1 only	217			

Moreover table (IV) shows the utilization of the PEs, latency, and number of MAC operations used in Conv layers.

TABLE IV
SHOWS THE CONV LAYERS PE UTILIZATION, MAC OPERATIONS, AND LATENCY

Point of comparison	Utilization of PEs (%)	MAC Operations	Latency(milliseconds)
Conv layer 1	154(91.66%)	105.41 M	41.1
Conv layer 2	140(83.33%)	447.79 M	60.8
Conv layer 3	156(92.85%)	415.33 M	78.8
Conv layer 4	156(92.85%)	623.00 M	58
Conv layer5	156(92.85%)	415.33 M	43.5

V. PIPELINING

In the design 32-bits fixed point division block is used. Table(V) compares the power and time consumption with, and without pipelining which illustrates the importance of using pipelining. The time consumed without pipelining is not reasonable for any AI therefore it is important to use pipelining.

TABLE V
COMPARISON BETWEEN HARDWARE RESOURCES AFTER PIPELINING

Point of comparison	Without Pipelining	With Pipelining
Maximum frequency	4.88 MHz	59.33 MHz
Dynamic power(50% of switching)	7 mW	79 mW
Static power	120 mW	121 mW

VI. CONCLUSION

In conclusion, the paper represents an implementation of Alex Network on ZYNQ-702 FPGA operating on 50 MHz frequency. Moreover, memory hierarchy and row stationary techniques are used in the introduced architecture design to lower the power consumption of the whole system using only 60 percentage of the on chip power. In addition to this, the design can fit any network architectures not only AlexNet.

VII. ACKNOWLEDGEMENT

This work was partially funded by ONE Lab at Zewail City of Science and Technology and at Cairo University, NTRA, ITIDA, and ASRT. Also,the authors would like to thank Ibrahim Medhat, Amr Gamal, Mohamed Ayman, Mahmoud Abdel, and Mennatullah Sayed for their support in the design, and simulation.

REFERENCES

[1] Y. Alhazek, A. Ibrahim, M. Amer, A. Abubakr, and H. Mostafa, "Hardware Accelerated Epileptic Seizure Detection System Using Support Vector Machine,", IEEE International Conference on Modern Circuits and Systems Technology (MOCAST 2019), Thessaloniki, Greece, pp. 1-4, 2019.
[2] M. Adel, A. Kotb, O. Farag, S. M. Darwish, and H. Mostafa,"Breast Cancer Diagnosis Using Image Processing and Machine Learning for Elastography Images,", IEEE International Conference on Modern Circuits and Systems Technology (MOCAST 2019), Thessaloniki, Greece, pp. 1-4, 2019.
[3] M. Cokun, A. Uar, . Yildirim and Y. Demir,"Face recognition based on convolutional neural network, "Addison-Wesley, Reading, Massachusetts, 1993. in MEES, 2017.
[4] 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.
[5] M. I. Elzayat, A. M. Saad, M. M. Mostafa, M. R. Hassan, M. S. Dawrweesh, H. Abdelmunim, and H. Mostafa,"Real-Time Car Detection-Based Depth Estimation Using Mono Camera,"IEEE International Conference on Microelectronics (ICM 2018), Sousse, Tunisia, pp. 260-263, 2018.
[6] S. Levine, C. Finn, T. Darrell, and P. Abbeel,"End-to-end training of deep visuomotor policies, "J. Mach. Learn. Res., vol. 17, no. 39,pp. 140, 2016.
[7] Eriko N, Ganesh V, Jaewoong S, Debbie M, Randy H, Jason Gee Hock Ong , Yeong Tat Liew, Krishnan S , Duncan M, Suchit S,and Guy B,"Can FPGAs Beat GPUs in Accelerating Next-Generation Deep Neural Networks?,",in FPGA International Symposium on Field Programmable Gate Arrays, 2017.
[8] V. Sze et al,"Efficient Processing of Deep Neural Networks: A Tutorial and Survey,",arXiv preprint arXiv:1703.09039, 2017.
[9] Y. Chen, T. Krishna, J. Emer, and V. Sze,"Eyeriss : An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks, "in ISSCC, 2016.
[10] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton,"ImageNet Classification with Deep Convolutional Neural Networks,"in Image Net,2012.