2019 22nd Euromicro Conference on Digital System Design (DSD)

Design and Implementation of a Low-power, Embedded CNN Accelerator on a Low-end FPGA

Bahareh Khabbazan
Electrical Engineering School,
Iran University of Science and Technology
Tehran, Iran
b_khabbazan@alumni.iust.ac.ir

Sattar Mirzakuchaki
Electrical Engineering School,
Iran University of Science and Technology
Tehran, Iran
m_kuchaki@iust.ac.ir

***Abstract*— in this paper, an optimized hardware for Convolutional Neural Networks with the purpose of implementation on embedded vision systems is presented. This design method is meant to be implemented with minimum resource consumption on a low-end hardware platform. We propose an architecture on a Z-turn evaluation board featuring a Xilinx Zynq-7000 system on chip (SoC). All computations in this architecture are optimized as 8-bit. Also, the accelerator has a frequency of 160 MHz and power consumption of 1.77 watts which leads to a performance of 40.96GOP/s, using only 134 computing units and 601 KB of internal memory. So we can claim that the acceptable speed and low power and low area consumption of this architecture make it an ideal choice for portable and embedded CNN applications.**

Keywords---Convolutional Neural Networks, FPGA, low-end hardware platform, Soc, Zynq, Low power

I. INTRODUCTION

Convolutional Neural Networks (CNNs) which are the subset of artificial neural networks are a powerful tool in a variety of applications in image recognition tasks that require high accuracy. The large computational complexity of real time applications is an increasing challenge for embedded system designers who try to use less hardware resources and less power usage. A tradeoff exists in designing high performance accelerators versus employing limited hardware resources in some platforms such as FPGAs. For some embedded vision tasks, a minimum criterion for real time applications is 10 image per second [1]. Authors in [2] proposed a CNN accelerator design on embedded FPGA with data quantization method. However, they assume a fix kernel size which isn't configurable for all convolutional neural networks. In [1], the convolutional layer is accelerated with limited resources with a reduction in required band width. They have evaluated their accelerator architecture targeting a ZC706 and have reached an operating frequency of 150MHz. However their hardware and memory utilization is inefficient. Also, the power consumption is close to 10 watts which seems to be big for embedded low power tasks. Recent work shows that the data format for implementing CNNs into FPGA platforms can be compressed from 32-bit floating point into fixed point and also works in [3][4] shown that for AlexNet network 8-bit fixed point has an accuracy degradation less than 1% which is negligible in many applications. So By reducing the bit length size of computations in CNN to 8-bits, we have negligible accuracy loss but great reduction in the power consumption and area cost of the design. The main contributions of this work are:

- We implement our design on a low-end Zynq evaluation board to show the capability of minimum resource requirements compared to similar works. We use an efficient data flow to reduce the data transfer between off-chip and on-chip memory. Also by our proposed architecture and by efficient arrangement of our data stored in on-chip memory, BRAM usage is reduced as much as possible.
- We propose a parameterized architecture with the help of ARM processor on Zynq SoC which is responsible for controlling the programmable logic part of the Zynq, configuring it with different convolutional and pooling layer sizes to support various networks.

The rest of this paper is organized as follows. Section II presents the background theory of CNN models. Section III presents our architecture for optimizing acceleration methodology. Section IV describes implementation details and experimental results of FPGA implementation of AlexNet and section V concludes the paper.

II. BACKGROUND

A. Convolutional Neural Network

Convolutional Neural Networks (CNNs) contain a large number of layers which are divided in two groups depending on their function: the chain of convolutional layers, non-linearity layers and pooling layers are responsible for feature extraction stage and the chain of fully connected layers are responsible for classification stage. The main operation in CNNs, as the name suggests, is the convolutional layer which is responsible for producing various features such as lines, an edge of some orientation or blotch of some color on the first layer, etc. A 3D array of coefficients called a kernel is applied to the ID number of input feature maps. Three hyper parameters of stride (S), zero padding and the depth of the output (OD) which equals to the number of kernels, control the size of the output feature map. The weights of the 3D kernel array are the pixels which are trained with ILSVRC 2012 training dataset and in this paper, the pre-trained CNN is accelerated. As shown in Fig.1, each 3D

978-1-7281-2862-7/19/$31.00 ©2019 IEEE
DOI 10.1109/DSD.2019.00102

kernel array is applied to a KH×KW window of input feature maps. Creating one pixel of output by shifting a window with the stride of S in input feature maps and by moving in two directions of IW and IH, one output feature map with the size of OW×OH is generated. By repeating this operation for OD number of kernels, OD output feature maps are produced resulting in a 3D array output. Fig.2 presents the pseudo code of the convolutional layer function.

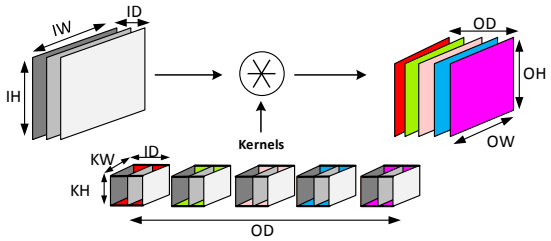

Fig. 1. A typical Convolution layer and its parameters

```
for (od =0 ; od != OD ; od ++ )                    ──► Loop 4
  for (oh =0 ; oh !=OH ; oh ++)                    ┐
    for (ow =0 ; ow !=OW ; ow ++ )                 ┘ Loop3
      for (id =0 ; id !=ID ; id++ )                ──► Loop 2
        for (kh =0 ; kh !=KH ; kh ++)              ┐
          for (kw =0 ; kw !=KW ; kw ++)            ┘ Loop1
            out(oh,ow,od)+ = in(S×oh+kh,S×ow+kw,id)
                          ×weight(kh,kw,id,od)
          end
        end
      end
      out(oh,ow,od)+ = bias(od)
    end
  end
end
```

Fig.2. Pseudo code of the typical convolutional layer

B. *Acceleration techniques of CNNs*

3D multiply-and-accumulate (MAC) operation of input feature maps and kernels for producing output feature maps, consist of six nested loops. Two loop optimization techniques of loop unrolling and loop interchange based on the acceleration strategy can be used for each Convolution layer discussed in [5]. In realizing loop unrolling techniques the number of parallel computations is proportional to computational resources on FPGA. Also the more these techniques attempt to minimize the memory access, the more effective they will be.

III. ACCELERATOR METHODOLOGY AND ARCHITECTURE

A. *Accelerator methodology*

In order to have a better data movement from on-chip memories to computation units, loop interchange technique is used. So we first serially compute loop-2, then loop-1, and finally we compute loop-4 then loop-3. To maximize the data reuse and reduce the memory access, we decided to unroll loop-4 which is related to calculating different output feature maps by using several computing units which calculate output features simultaneously as shown in Fig.3. Also we decided to unroll loop-2 to parallelize computations in depth of the layers input. As shown in Fig.4, an array of MAC units is used in each MAC cluster to realize this goal.

B. *Proposed accelerator architecture*

An accelerator architecture is divided into 3 parts, namely (1) LOAD, (2) CALC and (3) STORE. In LOAD, input feature maps and kernel weights of each layer is provided. In CALC, the process of computations, which is defined for each layer task to produce output feature map will be generated and in STORE, how to store the output and the data arrangement will be discussed.

To save power during calculations and reduce the resource consumptions, a dynamic 8-bit fixed point strategy is used. Authors in [4] and [3] show the 8-bit optimization is enough to keep an acceptable inference accuracy. Fig.3 shows the proposed accelerator for Convolution layer. The accelerator architecture is used for different types of convolution operations. In the processing system (PS) part, the ARM processor is responsible to initialize the programmable logic (PL) through the control bus to configure the convolution layer parameters in run-time cycle as illustrated in Fig.1. It also initializes the proper base addresses to start reading and writing to and from on-chip memories. After the initialization, the ARM core sends the start signal to begin the convolution operation. In this design, first the image data and kernel weights are stored as an 8-bit data in off-chip memory. Depending on the output feature range achieved for each Convolution layer which is obtained by examining the output feature range for several input features, the appropriate shifting value is selected and is adjusted by the ARM processor in PS part for Quantization unit designed in PL.

1) *LOAD*

To reduce the access to the slow Off-chip memory, an on-chip data memory is selected which is responsible for storing the input feature maps and also the output feature maps. The ping-pong strategy is implemented on data memory (Dmem) by splitting the area of input features and output features with different base addresses. Also the kernel memory is an on-chip memory which is responsible for storing the kernel weights. Due to the loop interchange technique, the data arrangement in both kernel memory and data memory is so that in a fixed row and column of the 3D array of input features and kernels, the pixels in the depth are first arranged in memory respectively. Before starting the operation for the first convolution layer, the whole image data is written to the data memory through PORTB which is multiplexed between AXI CDMA and Data Memory Controller (Dmem Ctrl). Because of the wider bit-width of PORTB, both tasks of writing the image data and output features in the data memory, are done through PORTB. Another AXI CDMA is responsible to write kernel weights on each kernel memory. The feeding input features data of Computing Units (CU) is 16-bit so on each row of data memory, two 8-bit pixels are stored and will be used in each reading operation.

2) CALC

Convolution operation consists of many MAC functions which need to be parallelized. In this architecture a Computing Unit (CU) is responsible for accelerating MAC functions with two loop unrolling techniques. Considering the amount of computing resources available on our Zynq platform, and implementing multipliers by just using DSP blocks, for unrolling loop-4, four CUs are considered to be used which leads to calculation of 64 outputs in parallel. As shown in Fig.3, the CU consists of three sub modules. MAC Cluster consists of 16 MAC arrays and 16 RELU & Quantization units which are responsible for applying non-linearity function on each of the MAC arrays' outputs. Each MAC array is responsible to calculate the pixel for one output feature map. Also for unrolling loop-2 as shown in Fig.4, the MAC array consists of 2MAC units which led to the calculation of 2MAC operations in each clock cycle. For each of the 16 MAC arrays in every CU, one kernel memory is considered. So in each row of the memory, 16 series of coefficients for different kernels are concatenated. Each series contains two 8-bit pixels of kernel in depth to feed 2MAC units with kernel weights in each MAC array. The advantage of this method compared to using 16 separate kernel memories for each of the MAC arrays is that a fewer BRAM is used.

3) STORE

In this design, in order to be able to start convolution operation for a new window of data as quickly as possible, we simultaneously read input features from data memory and write the final output features produced by CU to the data memory. Therefore, the output of each CUs is stored together in the Temp Reg to provide a proper data width for the PORTB of memory which equals to 256 bits. Each of the Temp Reg units holds the output of two CUs which produces 32 output feature pixels each of them having 8 bits. At the time of storing the output of CUs in the Temp Reg, a signal is issued for the sub modules of CUs. So the convolution operation for the next window starts. Thus, there is no need to wait to write the output of the Temp Reg module in the Data Memory and the next operation begins immediately. Here too, the Data Memory Controller unit is responsible for producing the proper address for writing in data memory.

IV. EXPERIMENTAL RESULTS

As a case study, the proposed accelerator architecture for implementing CNN models on FPGA is analyzed on AlexNet. The number of MAC operations for each convolutional layer with the proper parameters which are used in Fig.1 is listed in Table I. Note that in AlexNet the width and height size of the input feature maps and kernel weights are the same. As shown in Table I, the total number of MAC operations which includes 90% of all operations performed on CNN is 667M. The designed architecture is coded in Verilog-HDL and synthesized with Vivado 2017.4. The target FPGA board for implementing the AlexNet-CNN is called "Z-turn board" manufactured by MYiR Co which has a Zynq-7020 chip on it. The implemented design is fully pipelined and by

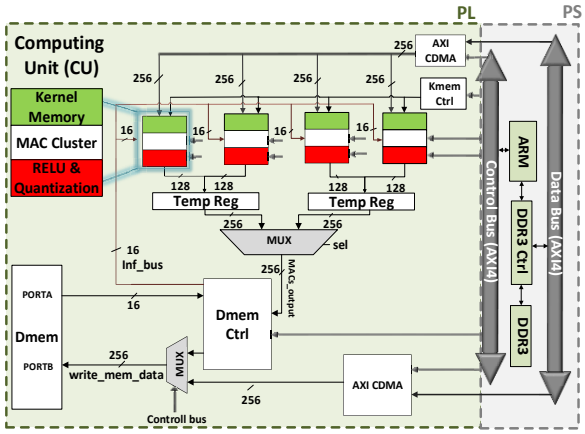

Fig.3.The proposed convolutional accelerator architecture

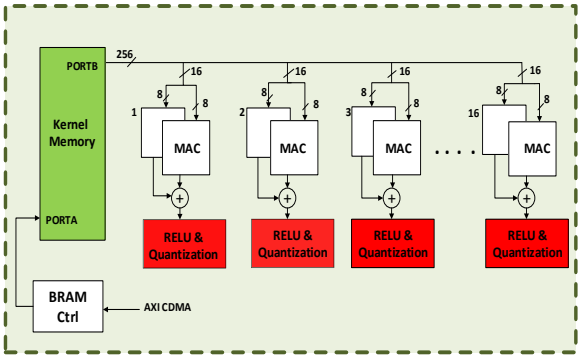

Fig.4. Computing Unit (CU) architecture

using register slices on high latency paths to increase the operating frequency, it has successfully achieved a clock frequency of 160MHz. The architecture contains four Computing Units (CUs) each of which has 16 MAC arrays. Each MAC array contains 2 MAC units and each of MAC units is implemented with one DSP48E1 block. At each clock cycle, one output pixel is generated by each of these MAC units. So each CU can produce 16 output pixels in one run cycle. The limiting factors for the number of MAC operations is BRAM units and the number of DSP blocks which leads to the maximum performance of 20.48 Giga multiply accumulation operations per second. Before each run cycle, kernel weights in kernel memory will be updated by AXI CDMA and the ARM processor initiates the control registers with proper parameters in PL part. After the whole run cycle is finished, 64 output features are calculated. So layers 2,3,4,5 need 4 run cycles to completely generate their output feature maps. After finishing the whole operation of one convolution layer, the output will be stored in data memory and is used as an input feature map by pooling layer just by changing the base reading address of the data memory. The off-chip memory is not being used during the run cycle procedure. Table II shows the resource utilization of each layer of our implementation and Table III shows the computation time of each layer. The total time required for accelerator to execute all 8 layers of convolution and pooling is equal to 36.53 milliseconds. This should be added to the loading

time of the kernels and data and storage time of the output results which makes the network performance to take 40 milliseconds. So our accelerator has the ability to process 25 frames per second which is appropriate considering the processing time of 10 frames per second as a suitable criterion for real-time embedded vision applications [1].

TABLE I. NUMBER OF MAC OPERATIONS IN EACH CONV LAYER

Layer	OH,OW	OD	KH,KW	ID	#MAC
1	54	64	11	3	68M
2	27	256	5	64	299M
3	13	256	3	256	100M
4	13	256	3	256	100M
5	13	256	3	256	100M
Total					667M

TABLE II. RESOURCE UTILIZATION

BRAM(36Kb)	FF	LUT	DSP48	Resource
133.5	31470	34179	134	**Utilization**
95.36%	29.92%	64.24%	60.9%	**Percent**

TABLE III. COMPUTATION TIME

Layer	1	2	3	4	5	Total
Convolution(ms)	4.43	14.59	4.86	4.86	4.86	33.6
Pooling(ms)	1.38	1.28	0.27	-	-	2.93

The important point to be taken into account here is that the results obtained in Table III are achieved only by using a very small number of DSP48s, BRAMs and LUTs resources in comparison with other reported implementations. Reference [6] consumes the number of DSP48 blocks 16 times more than our work. If we had used the resources of Virtex-7 platform where on-chip memory resources are much higher than our platform, we definitely could have achieved a higher performance.

The authors in [1] defined a $Perf / DSP$ as the mean share of each DSP48 block in the total performance. The higher value of this ratio shows the more efficient use of available DSP48 resources to achieve a higher performance. $Perf / DSP$ in our work is equal to 0.293 while in reference [1] is only 0.098. This means our architecture achieved significant performance improvement by using fewer DSP48 blocks which makes it possible to be implemented on low-end platforms with limited resources. our design could achieve higher performance with almost one third of DSP48 and BRAM resource and one half of the LUTs compared to [1].

V. CONCLUSION

This paper presented a CNN accelerator with focus on minimizing power and resource utilization. The design was implemented on Z-turn evaluation board with a frequency of 160MHz and a power consumption of 1.77watts. The performance reached 40.96 GOPs with power efficiency of 23.14GOPs/W which proves that this accelerator is a good candidate in embedded applications with low-end platforms.

TABLE IV. COMPARISON WITH PREVIOUS CNN IMPLEMENTATIONS

Architecture	ISCA2010[7]	ISFPGA2015[5]	FPL2016[8]	FPL2016[6]	CSII2017[1]	Our Work
Precision	48bit fixed	32bit float	8-16-bit fixed	16-bit fixed	Q15	8-bit fixed
Frequency	200MHz	100MHz	100 MHz	156MHz	150MHz	160MHz
FPGA Chip	SC240T	VX485T	Stratix V GXA7	VX690T	XC7Z045	XC7Z20
CNN Size	0.26 GMAC	1.33 GOP	1.46 GOP	1.45 GOP	1.33 GOP	1.334GOP
Performance	16 GOPs	61.62 GFLOPs	134.1 GOPs	565.9 GOPs	38.4 GOPs	40.96GOPs
DSP Util.	N/A	2240	256	2144	391	134
Logic Util. [a]	N/A	186251	121K	273805	77442	34179
On-chip BRAM [b]	N/A	1086	1,552	956.5	545	133.5
Energy Efficiency	1.14 GOPS/W	3.31 GOPS/W	6.87 GOPS/W	22.15 GOPS/W	3.84GOPS/W	23.14 GOPS/W

[a]. Xilinx FPGAs in LUTs and Altera FPGAs in ALMs
[b]. Xilinx FPGAs in BRAMs (36 Kb) and Altera FPGAs in M20K RAMs (20 Kb)

VI. REFERENCES

[1] S. Moini, B. Alizadeh, M. Emad, and R. Ebrahimpour, "A Resource-Limited Hardware Accelerator for Convolutional Neural Networks in Embedded Vision Applications," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 64, no. 10, pp. 1217–1221, 2017.

[2] J. Qiu *et al.*, "Going Deeper with Embedded FPGA Platform for Convolutional Neural Network," in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '16*, 2016, pp. 26–35.

[3] K. Guo *et al.*, "Angel-Eye: A complete design flow for mapping CNN onto embedded FPGA," *IEEE Trans. Comput. Des. Integr. Circuits Syst.*, vol. 37, no. 1, pp. 35–47, 2018.

[4] B. Y. Fu, E. Wu, A. Sirasao, S. Attia, K. Khan, and R. Wittig, "Deep Learning with INT8 Optimization on Xilinx Devices INT8 for Deep Learning INT8 Deep Learning on Xilinx DSP Slices," 2017.

[5] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays - FPGA '15*, 2015, pp.161–170.

[6] H. Li, X. Fan, L. Jiao, W. Cao, X. Zhou, and L. Wang, "A high performance FPGA-based accelerator for large-scale convolutional neural networks," in *FPL 2016 - 26th International Conference on Field-Programmable Logic and Applications*, 2016, pp. 1–9.

[7] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, "A dynamically configurable coprocessor for convolutional neural networks," *ACM SIGARCH Comput. Archit. News*, vol. 38, no. 3, p. 247, 2010.

[8] Y. Ma, N. Suda, Y. Cao, J. S. Seo, and S. Vrudhula, "Scalable and modularized RTL compilation of Convolutional Neural Networks onto FPGA," in *FPL 2016 - 26th International Conference on Field-Programmable Logic and Applications*, 2016.