Analog Integrated Circuits and Signal Processing (2021) 106:399–408
https://doi.org/10.1007/s10470-020-01638-5

Implementation of deep neural networks on FPGA-CPU platform using Xilinx SDSOC

Rania O. Hassan[1] · Hassan Mostafa[1,2]

Received: 5 July 2019 / Revised: 19 December 2019 / Accepted: 17 March 2020 / Published online: 24 March 2020

Abstract
Deep Convolutional Neural Networks (CNNs) are the state-of-the-art systems for image classification due to their high accuracy but on the other hand their high computational complexity is very costly. The acceleration is the target in this field nowadays for using these systems in real time applications. The Graphics Processing Units is the solution but its high-power consumption prevents its utilization in daily-used equipment moreover the Field Programmable Gate Array (FPGA) has low power consumption and flexible architecture which fits more for CNN implementations. This work discusses this problem and provides a solution that compromises between the speed of the CNN and the power consumption of the FPGA. This solution depends on two main techniques for speeding up: parallelism of layers resources and pipelining inside some layers. On the other hand, we added a new methodology to compromise the area requirements with the speed and design time by implementing CNN using Xilinx SDSOC tool (including processor and FPGA on the same board). Implementing design using HW/SW partitioning will enhance time design based on high level language(C or C++) in Vivado HLS (High Level Synthesis). It also fits for more large designs than using FPGA only and faster in design time.

Keywords Convolutional neural networks (CNNs) · Alex-Net · Accelerating CNNs · FPGA · Virtex · HW/SW co-design partitioning · SDSOC · HLS

1 Introduction

Artificial intelligence and deep learning have shown their utility and effectiveness in solving many real- world problems in the past few years. The need of direct programming and create an intelligent system is the motivation to use deep learning. This intelligent system can automatically adapt to new situations, learn and develop itself. The CNNs are the state-of-the-art today as one of deep learning algorithms, which proved their high accuracy in solving problems such as face recognition [1] and autonomous driving [2].

CNNs have significant higher accuracies than traditional algorithms but they require huge amounts of computational resources and memory access due to the large number of parameters in the convolution-operation, which represents a computational challenge for the General-Purpose Processors (CPUs) and consumes large amount of power. Recently many applications such as embedded systems in self- driving cars need high energy efficiency and real-time performance. As a result, hardware accelerators such as GPU, FPGA, and Application Specific Integrated Circuits (ASIC) have been utilized to improve the throughput of the CNN.

GPUs are the most widely used platforms to improve both training and classification processes of CNNs [3] therefore, thanks to their high throughput and memory bandwidth. However, GPUs consume a considerable amount of power which is another important performance evaluation metric in the modern digital systems.

✉ Hassan Mostafa
hmostafa@uwaterloo.ca

Rania O. Hassan
rania.osama2014@gmail.com

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

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

ASIC design, furthermore, has achieved high throughput with low power consumption [4, 5], but large development time and cost compared to other solutions. A new generation of FPGA increases the capacity of hardware resources. This provides thousands floating-point computing units and low power consumption. Hence, FPGA-based accelerators are efficient alternatives which provide low power consumption As the ASIC but high throughput and configurability at a reasonable cost.

Many approaches have been used to implement CNN on FPGA. A comparison between GPU and FPGA for DNNs has been made by the authors of [6]. They propose a detailed case study on accelerating Ternary Res-Net, the results are very promising; Stratix 10 performance is 10%, 50% and 5.4 × better in performance (TOP/sec) than Titan X Pascal GPU, FPGAs may become the platform of choice for accelerating DNNs as it has been proved by the results. The work in [4] propose an energy-efficient dataflow called row stationary to maximize the reuse and accumulation at the local memory level (RF or caches) for all types of data (weights, pixels and partial sums). The work in [7] transforms a convolution layer into a regular matrix- multiplication (MM) in the Fully Connected (FC) layer, and implements an MM-like accelerator for both layers. The authors have presented in [8] an opposite approach that transforms a regular MM into a convolution, and implements a convolution accelerator for both convolution and FC layers. Furthermore, anew implementation methodology is presented to implement design written in high-level language using Xilinx SDSOC (Software-Defined System on Chip) for HW/SW implementation with fast design time and more flexibility to change design implementation easily.

This work is an extension work to the work in [9] with adding a new fast implementation technique on Xilinx ZYNQ ultra scale board and comparison with the accelerated implementation proposed in [9]. This proposed architecture in [9] relays on parallelism for all kernels in the convolution layer which is flexible for any network size and uses extensively local memories to store all the data. These proposed techniques reduce the design time and the power dissipated in external memory access. Results are provided for hardware utilization and power consumption, comparing the results with CPU performance.

Using HLS flow with high level language will be the state of the art very soon as it gives very important advantage which is the small design time similar to what happened to ASIC flow due to FPGA as in the last few years all works targets FPGA for its fast design time and reconfigurability and sacrifice by power and delay. HLS also will defeat FPGA and will be the target to any designer to fast implement and test his design with losing some of power, delay, and area.

This paper is organized as follows; Sect. 2 provides background on CNN. Section 3 discusses the hardware implementation using the two techniques. Section 4 shows the hardware utilization results for both techniques and discussion of these results with comparison. Section 5 is presenting acknowledgment for people helped to do this work. Section 6 concludes the proposed work and presents the future work.

2 Background

CNNs are computational models inspired by the way of the human brain. CNNs operation has two phases, training phase and inference phase (feed-forward path). In the training phase, the CNN is trained on a known data set to learn its weights to minimize the error. This work focuses on the feed-forward path of a pre-trained CNN.

A typical CNN feed-forward path consists of a feature extractor and a classifier. The feature extractor extracts an input features across the CNN layers which are; Convolution (Conv), Rectified Linear Unit (ReLU), Pooling (Pool) and Local Response Normalization (LRN). Then the feature extractor sends these extracted image features to the classifier which is implemented using fully connected (FC). In order to understand the proposed hardware implementation, the CNN detailed layers will be discussed in this section, also Alex-Net architecture will be discussed, which is one of the state-of-the-art CNN models will be presented.

2.1 Convolution layer

The Conv layer is the core building block of a CNN that does most of the computational heavy lifting. It's always the first layer in a CNN. The convolution operation is done by applying element-wise multiplication and accumulation between input feature maps and the weight filters (called also kernels). The Kernels are the Conv operands obtained from the training phase, then sliding these filters over the

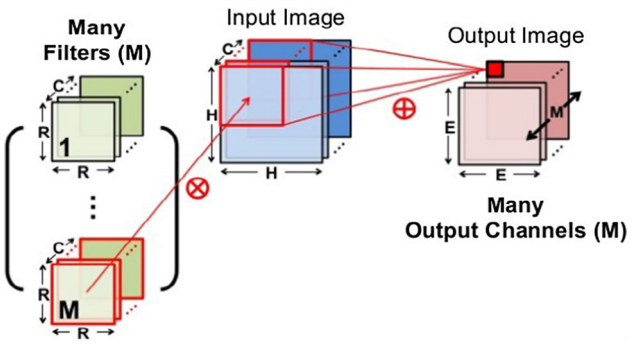

Fig. 1 Convolution operation [4]

input feature maps as shown in Fig. 1. Each of these weight filters can be thought of as feature identifiers.

The computation is given in Eq. (1), where M is the number of output feature maps (number of filters) of size E × E, C is the number of channels in Input feature maps, and R × R is the size of the Filter.

$$out[m][h_0][w_0] = b_i + \sum_{i=1}^{C}\sum_{k_h=0}^{R}\sum_{k_w=0}^{R} IN[i][h_0 + k_h]*Kernel[m][i][k_h][k_w] \quad (1)$$

2.2 Pooling layer

A key aspect of Convolutional Neural Networks is pooling layer, typically applied after the convolution layers. Pooling layers (also called sub-sampling or down sampling) reduces the dimensionality of each feature map but retains the most important information. Pooling can be of different types: Max, Average, Sum etc. In practice, Max Pooling has been shown to work better.

2.3 Rectified linear unit (ReLU)

The ReLU layer is a non-linear operation that performed after every Conv layer. Its output is given by; max (0, input). The purpose of ReLU is to introduce non- linearity in the CNN after linear operation of convolution, since the network need to learn from real world data which is non-linear and that for network to generalize or adapt with variety of data.

2.4 Local response normalization (LRN)

The LRN reduces top-1 and top-5 error rates by 1.4% and 1.2%, respectively [10]. This sort of response normalization implements a form of lateral inhibition inspired by the type found in real human neurons. The LRN layer is responsible for normalizing the local neighborhood of the excited neuron and makes it even more sensitive as compared to its neighbors to avoid the saturation in network.

The normalization functionality is given in Eq. (2).

$$out^i = in^i_{s,y} \Big/ \left(k + \alpha \sum_{j=\max\left(0,i-\frac{n}{2}\right)}^{j=min\left(N-1,i+\frac{n}{2}\right)} \left(in^i_{s,y}\right)^2\right)^{\beta} \quad (2)$$

2.5 Alex-Net

Alex-Net is one of the state-of-the-art CNN; it won the 2012 ILSVRC (Image-Net Large-Scale Visual Recognition Challenge). It is the first model to achieve top-1 and top-5 error rates of 37.5% and 17.0% respectively on the test data of Image- Net dataset [11], which is an astounding improvement compared with the other top models in the context.

The CNN developed by Krizhevsky, Sutskever, and Hinton in 2012 [12], consists of five Conv layers, Pool follows some of them, two LRN layers, and finally three fully connected layers with a final 1000-way soft-max layer as shown in Fig. 2.

There are two proposed techniques for hardware implementing of Alex-Net using RTL design for the whole system or using high-level synthesis for implementation beside HW/SW partitioning to some functions for hardware implementation.

In the Sect. 3, we will discuss the difference between the two techniques and comparison between their results.

3 Hardware implementation

The two techniques, we presented here one who target to accelerate with an optimized RTL code and the other one who targets the fast design time with very high-level language (HLS flow).

Our presented contribution is presented in two points:

First, the low power and area RTL implementation for Alex-Net CNN system is proposed. Secondly, we introduce the new approach for using HLS flow for Alex-Net with high-Level input language and controlling the functions that will be implemented on FPGA and the others will be executed on SW.

3.1 RTL implementation

The proposed architecture [9] basically depends on parallelism. The parallelism is responsible for reducing the time needed for image classification in Alex-Net architecture. In addition, some techniques as pipelining and on chip (local) memory usage are used for more acceleration which will be discussed in details showing the effect of the all techniques on the overall CNN speed.

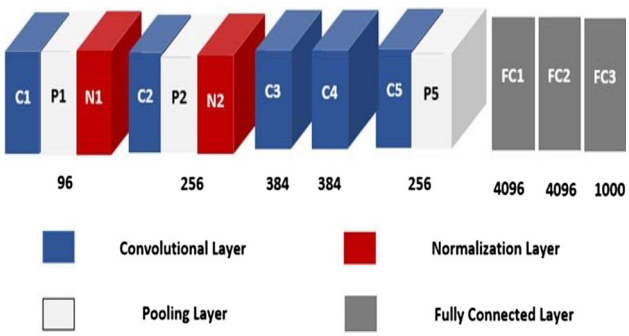

Fig. 2 Alex-Net neural network architecture

In this section, each layer hardware-implementation perspective will be discussed in details.

3.1.1 Convolution layer

MAC operations (multiplication and accumulation) is the main part in convolution layer within each filter with the input feature maps;

These MAC operations are done using the smallest basic unit which is the Parallel Engine (PE) composed of multiplier and accumulator.

The parallel execution of the weight filters is done by duplicating these PEs, Executing the Conv1 layer, 96 PEs are used which is equal to the number of the weight filters of Conv1 layer, The number of parallel PEs can be customized based on the compromise between the number of available resources on FPGA and the required CNN speed, the speed is much faster by increasing the number of PEs.

Figure 3 shows the parallel structure of PEs used for Conv1 layer, each PE is responsible for one weight filter operations as discussed before. This parallel structure speeds up the Convl execution by 96 times (number of filters). This will be further explained in the Sect. 4 showing the time reduction resulted from this parallelism.

3.1.2 Pooling layer

The Pool layer hardware is similar to the Conv layer hardware from the parallelism technique perspective; the Pool kernels across the different input feature maps are executed in parallel. The parallel structure of the Pool layer is composed of number of parallel PEs equal to the depth of the input to the Pool layer where each filter is applied on one depth. The Pool layer operations mainly depend on max-pooling operations as discussed in Sect. 2, these max-pooling operations are done using the smallest basic unit. This unit is a parallel engine composed of a comparator and a register holding the maximum number compared to its neighbors in one kernel. The Pool layer execution time is negligible compared to the Conv layer execution time. The optimization techniques are mainly concerned with reducing the execution time of the Conv layer which will be discussed in details in the later sections.

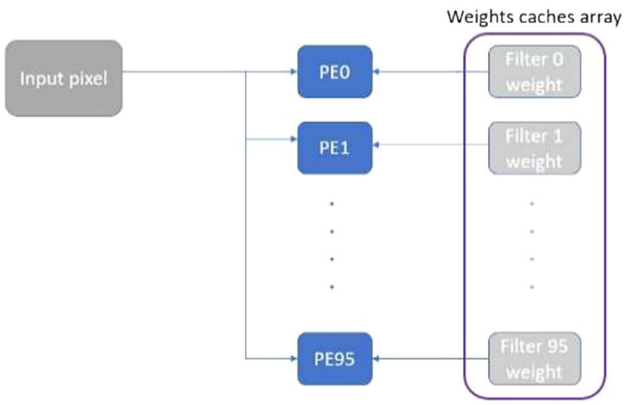

Fig. 3 Convolution layer architecture

3.1.3 Local response normalization layer

The LRN functionality is normalizing around the local neighborhood of the excited neuron as discussed in the Sect. 2. It makes it even more sensitive as compared to its neighbors.

The hardware implementation proposed as follows:

- LRN Layer needs multiple input caches to access the whole input feature maps simultaneously to be able to implement the summation process. Subsequently the number of input caches in this design is equal to the input feature maps of this LRN layer.
- A tree of adders that adds the input squares then multiply it by α and add k to the result does the summation of squares of Eq. (2).
- Implementing customized combinational fixed-point division, the proposed idea is to use two combinational integer divisions; the first one calculates the integer part and the second calculates the fraction part. In this CNN, the divider is always larger than one, so the fixed-point division is customized to work properly only when the divider is larger than one and any other values won't act properly. Sign bit is considered but actually no need for it because the Pool and ReLU output is always positive.
- Output storing technique: the output elements are stored in one cache if the following layer is group one Conv layer or two caches if the following layer is group two Conv layer.

3.1.4 Fully connected layer

The Alex-Net architecture consists of 2 main parts as mentioned in the Background section: feature extractor and a classifier for improving accuracy. The feature extractor is enough for classification but adding a classifier improves the network accuracy which consists of series of FC layers. The FC layer is mainly based on matrix–vector multiplication, the matrix consists of weights. These weights are obtained from the training phase and the vector is a group of features resulting from the feature extractor part in the CNN. Each element in the output vector is a weighted sum of the input vector that's why it's called fully connected. The main idea of the proposed design is to guarantee accelerating this layer more than the software.

There are two main techniques to achieve this:

- The main idea of parallelism in this layer is using parallel engines of the building block which consists of a multiplier and accumulator the same as Conv layer corresponding to the number of rows of the weights' matrix but due to the large number of rows only part of the rows is taken in parallel and after getting the outputs corresponding to these rows, other rows are taken.
- Pipelining is used for more speed up. In the layer design there is a cache for each PE at its weight port and only one cache for all the PEs at the output. Hence, pipelining occurs in the weights cache as a weight of a certain row is used, the weight of the next row corresponding to the previous weight is being stored in the cache while computing the next weight, so that the MAC operations of the rows are done one after the other without any waste of time between each row and the preceding one.

3.1.5 Synthesis and pipelining

The synthesis on the first super layer of the design is performed. The layer consists of (Conv, Pool, and LRN layers) on Virtex-7 VC709 FPGA using Vivado 2015.2 synthesis tool, Table 1 shows the utilization of the resources. The results shown in Table 1 are stating that conv1 layers used 96 DSPs which is corresponding to the number of parallel engines equal to the depth of this layer. The Pool and LRN are used DSPs for the addressing equations.

The time and the hardware resources between layers are not utilized. Using the pipelining technique for utilize the time and reducing hardware resources between the Conv and Pool layers is very efficient. As typically in Alex-Net Pool1 parameters are (kernel size = 3 × 3, stride = 2). Therefore, the Pool layer can start its operation after the first three rows in conv1 output are completed.

The calculations of the first row of pool1 is based on the first three rows in conv1 output and then the second row will be produced after the fourth and fifth rows of conv1 have been completed and so on. Figure 4 shows the pipelining flow in time. Conv1 can overwrite its output after pool1 generates the row output that would reduce the memory storage.

Table 1 Hardware resources for the first super layer

Layer/Resources	DSPs	LUTS	BRAMs 36 k	Registers
Conv1	96	8049	48	5736
Max Pooling1	2	8022	192	3226
Norm1	7	16,897	48	248

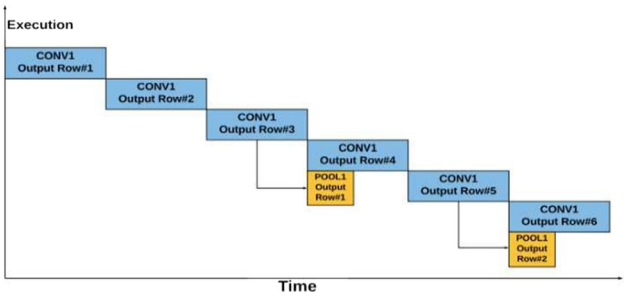

Fig. 4 Execution of convolution and pooling stages in pipelining

Table 2 Comparison between hardware resources after pipelining

Point of comparison	Original	Pipelined design
Utilization: LUTs	11%	13%
Utilization: BRAMs	22%	9%
Power	1.141 W	1.071 W

Table 2 shows that the reduction is done in resources after using pipelining compared to original design. Especially, BRAMs are decreased from 22 to 9% which is a significant reduction to save area and power.

3.2 Alex-Net implementation using SDSoC

SDSoC (Software-Defined System on Chip) environment [13] is an Eclipse-based Integrated Development Environment (IDE) for implementing heterogeneous embedded systems using the Zynq®-7000 All Programmable SoC (System on Chip) platform. The SDSoC system compilers (sdscc/sds++) transform C/C+ + programs into complete hardware/software systems based on command line options that specify target platform, and functions within the program to compile into programmable hardware.

The SDSoC system compilers generate hardware and software components that preserve program semantics and ensure synchronization between hardware and software threads, while enabling pipelined computation and communication. Each hardware function runs as an independent thread to achieve high performance with the minimum design time. The main advantage of using SDSOC is the ability to implement more large CNNs like VGG, GoogleNet, and Resnet, which has hundreds of layers much similar to AlexNet. This will give less time for design but the generated RTL is not optimized so it will take more area, power, and may be delay too. To explore the design space for Alex-net, first we choose each function for hardware acceleration and all other functions for software to get the power, area, and delay for each function separately. Then eliminate the other partitioning possibilities

based on the parameters of each function. This way guarantee that we run the possible partitions only and get the best among them.

As shown in Fig. 5 the SDSOC environment design flow, the first step is to identify compute-intensive hot spots in the application that can be migrated to programmable logic to achieve higher performance, and to isolate them into functions that you can compile for hardware. C/C+ + code compiled for programmable logic with the SDSoC environment must conform to coding guidelines and must also conform to Vivado® High-Level Synthesis (HLS) guidelines.

The remaining flow of SDSOC is selecting the functions for hardware acceleration and running the code with choosing estimate performance and generation of SD-Card image. We then take the SD-Card image to evaluation board chosen for implementation. After placing and routing is done on board, the tool generates an estimation report for speed up for using selected function for hardware implementation. We do this flow several times until we get the best optimum solution, which gives us the hardware functions with their generated RTL code.

The proposed implementation of Alex-Net is done on the ZYNQ Ultrascale ZCU104 board. The ZYNQ UltraScale family [14] architecture enables multi-hundred gigabit-per-second levels of system performance with smart processing, while efficiently routing and processing data on-chip. UltraScale architecture-based devices address a vast spectrum of high-bandwidth, high-utilization system requirements.

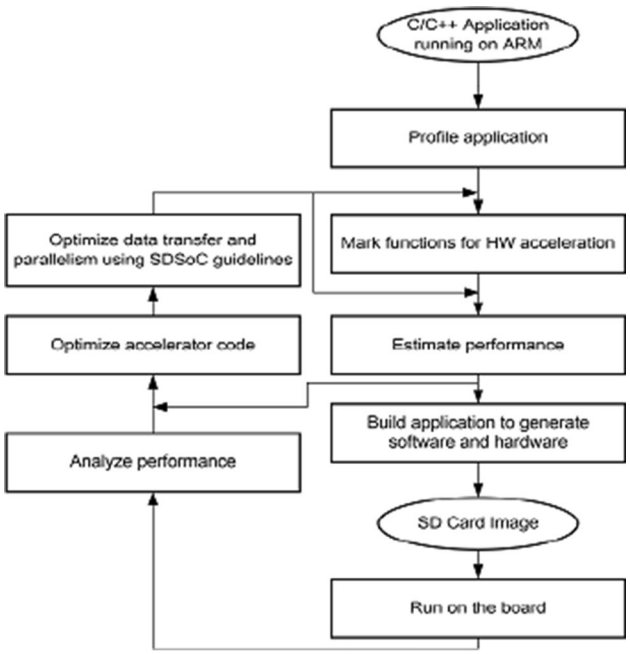

Fig. 5 SDSoC environment flow [13]

The ZCU104 evaluation board [15] provides a flexible prototyping platform with high-speed DDR4 memory interfaces, an FMC expansion port, and multi-gigabit per second serial transceivers, a variety of peripheral interfaces, and FPGA fabric for customized designs. The ZU7EV device integrates a quad core Arm®Cortex™-A53 processing system (PS) and a dual-core Arm Cortex-R5 real-time processor, which provides application developers an unprecedented level of heterogeneous multiprocessing.

The input language for SDSoC is a C/C++ code written according to high level synthesis (HLS) instructions approved by Vivado HLS tool supported with SDx environment tools [13] as shown in Fig. 5.

The Alex-Net is described using C+ + language on SDx V2018.3 with Vivado HLS 2018.3 based on the trained forward path in [16] converting their matlab code to C++ . The system is divided to 8 main C++ functions as follows: Conv Layer1(C1,P1,N1 as in Fig. 2), Conv Layer2(C2,P2,N2), Conv Layer3(C3), Conv Layer4(C4), Conv Layer5(C5,P5), Fully Connected Layer6(FC1), FC layer7(FC2), FC layer8(FC3).

The support of CPU and FPGA together on the same board makes a lot of combinations for implementing design which function will be executed by CPU and which one will be implemented on FPGA with HDL code generated to it. We rewrite Alex-Net in C++ language with the rules fit with the Vivado HLS manual guidelines. The design has been simulated on SDx 2018.3 environment using SDSoC, Vivado, and Vivado HLS. Our methodology is running the whole system by CPU only (SW solution) then takes each function to be implemented on FPGA and the other functions SW to get each function specifications (Hardware Resources, Latency, Power Estimations, Hardware accelerated cycles) separately then start to combine between the functions that can be fit on HW together and see the improvement in performance then finally decide the best combination of implementation to be done. This design has 256 different combinations of implementation as we said we tried the possible implementation with good performance as we will see the results in the Sect. 4.

4 Results and discussion

4.1 RTL implementation results

The main purpose is to accelerate Alex-Net architecture for image classification, the functionality is verified on the RTL with reasonable accuracy and the timing results are obtained and compared to the MATLAB R2014a execution-time to show how much the CNN speed is enhanced. Table 3 shows the execution time of some layers of the MATLAB R2014a (CPU) and the one of the accelerated

Table 3 Simulation time of SW and RTL

Layer	MATLAB Simulation time(s)	FPGA Virtual simulation time
Conv1	7.8	11 ms
Pool1	0.37	65.6 µs
Norm1	1.83	700 µs
Conv2	7.9	8.7 ms
Pool2	0.3	1690 ns
Norm2	1	432.64 µs

CNN on FPGA. It's clear that the bottleneck of the CNN is the Conv layer; therefore it is the first design consideration to be accelerated as discussed in the previous Sect. 3.2.

Comparative study to the GPU is summarized in Table 4. The mentioned results show that the FPGA is faster than the CPU but slower than GPU, but it is still preferred over the GPU due to the FPGA lower power consumption which is proved by estimating the dynamic power consumption of the first three layers (Conv-Pool-LRN)using Virtex7 FPGA which is 0.785 W and that estimation is reasonable compared to the power consumed in [17]which is 7.2 W for the first ten layers which is lower than the power consumed by the GPU as mentioned in [18] that Titan X GPU throughput of 3.23 TOP/s comes at a relatively high-power cost which is 250 W.

4.2 AlexNet implementation using SDSoC

4.2.1 Hardware resources utilization

After performing Debug build for design with choosing only one hardware function and the other functions are executed by CPU, we get the detailed reports for synthesis and implementation of this function to get hardware utilizations (LUT, BRAMs, DSPs, and FFs). The Zynq Ultra scale 104 contains 1728 DSPs, 624 BRAMs 18 kb, 230,400 LUTs, and 460,800 FFs. Table 5 shows the hardware resources for some functions of design (The used resources and the utilization (U)). The Maximum Resources are used in the first 2 super layers as including convolution, max-pooling, and LRN. Some of these functions do not fit due to its large parameters and it needs more BRAMs. As we are using single precision floating point for all parameters.

Table 4 Simulation time of Alex-Net on different GPUs

Hardware accelerator	Execution time (ms)(forward path)
Proposed architecture	40.94
Pascal Titan X	5.32
GTX 1080	7.00
Maxwell Titan X	7.09

The synthesis of design using SDSOC gives over estimate to utilization resources, which is actually much smaller after implementation. The BRAMs do not fit for some layers due to the large parallel computational in convolution operation but it is decreased after the implementation phase. The pipelining is very important here for implementing any of conv super layers to decrease time of computations and resources. We used some of HLS pragmas for pipelining in convolution operations.

Vivado HLS supports some pragmas for those functions will be implemented on FPGA. We used ap_fifo pragma supported by Vivado HLS for streaming the reading of inputs and the writing operations of outputs for all functions with using HLS_pipeline pragma inside loops. These individual results for each function will guide to decrease the implementation combinations among all 8 functions. The implementations here is SW or HW for 8 functions will result in 256 implementation combinations so we will try some of these combinations depend on these results later.

4.2.2 Power estimations

Table 6 lists the estimated power (Watt) consumptions for the hardware partitioned functions including on chip power consists of dynamic power, programmable logic (PL), Processing system (PS), and static power (PL and PS). These numbers based on the automatic generated RTL from the tool. We get the numbers using Vivado power estimator for the generated RTL Project. We can see the benefits of locating conv. layers for hardware acceleration with less power and less delay.

4.2.3 Hardware accelerated cycles

Hardware acceleration is metric defined by SDSoC tool. Hardware acceleration is the number of clock cycles improvement in execution of system if the function is implemented as hardware function on the programmable logic.

Table 7 shows the hardware acceleration for the generated platforms of the synthesized hardware as estimated by debugging compilers of tool. The used performance estimation assumes worst-case latency of hardware functions, it also assumes worst-case data transfer size for arrays so it could be the hardware function latency and data transfer size at run time is smaller than such assumptions). As expected, the big super layers that including

Table 5 Hardware resources

HW Func	LUTs		BRAMs 18 k		DSPs		FFs	
	Used	U%	Used	U%	Used	U%	Used	U%
Conv layer1	19,260	8	953	152	95	5	11,037	2
Conv layer2	20,696	8	1298	208	93	5	11,232	2
Conv layer3	3232	1	1881	301	5	~ 0	1550	< 1
Conv layer4	2964	1	1580	253	5	~ 0	1368	< 1
Conv layer5	3841	1	1169	187	5	~ 0	1701	< 1
FC layer6	691	~ 0	0	0	5	0	663	~ 0
FC layer7	690	~ 0	0	0	5	~ 0	661	~ 0
FC layer8	687	~ 0	0	0	5	~ 0	652	~ 0

Table 6 Power estimations

HW function	On-chip power		Dynamic power	
	Total	PS	PL	PS
Convlayer1	3.592	2.777	0.221	2.678
Convlayer2	3.605	2.781	0.23	2.682
Convlayer3	3.592	2.781	0.217	2.682
Convlayer4	3.593	2.781	0.218	2.682
Convlayer5	3.598	2.781	0.223	2.682
FClayer6	3.607	2.781	0.231	2.682
FClayer7	3.616	2.781	0.241	2.682
FClayer8	3.611	2.781	0.236	2.682

Table 7 Hardware accelerated clock cycles

HW function	Accelerated clock cycles $\times 10^9$
Convlayer3	2.147483647
FClayer6	2.147483647
FClayer7	1.649306326
FClayer8	0.403445609

convolution or FC6 that contains many Parallel computations compared to the other layers has the large enhancements.

4.2.4 Implementation combinations

There are 256 implementation combinations for this design as we said. We will choose some of them depends on the results of area of each functions to decide which functions will implemented on HW and the others on SW. Table 8 shows the utilization of hardware resources for different implementations focus on fully connected layers combinations. We can see that the large hardware resources used in the case of FC6 and FC8 implemented in HW (2^{nd} case)

Table 8 Hardware resources utilization

HW function	LUT U%	BRAMs U%	FFs U%	DSPs U%
FC6,FC7	9.73	12.82	6.36	0.58
FC6,FC8	13.38	17.95	8.79	0.58
FC7,FC8	9.73	12.82	6.36	0.58
FC6-8	11.85	15.38	7.72	0.87

because layers 6 save and sends data to layer 7 which is on CPU then CPU send output of layer7 to HW for layer8. It is much better if all the 3layers on HW as in the last case with the less one in resources if we choose to implement two consecutives layers on HW together.

As shown in Table 9 the implemented combinations for HW focus on fully connected layers and show the accelerated clock cycles that will improve speed of system. Furthermore, the dynamic power is the dominant in on-Chip power so the different solutions have a small effect on PL power, as it is much smaller than PS power. Hence, the PS power is the dominant term in dynamic power too. These results are very useful guide to choose and eliminate the combinations for implementation.

4.3 Comparison between RTL and SDSoC HW implementation

Here is a simple comparison based on the previous results of synthesis phase for both RTL and SDSOC of the first super layer implementation shown in Table 10. The optimized RTL has better results as SDSOC generates un-optimized RTL code RTL takes a lot of LUTs but with minimum use of BRAMs and FFs but SDSOC goes to use BRAMs more than LUTs and DSPs. On the other hand, the SDSOC flow is giving less time for design, more flexible for any change, the optimized RTL can replace for the generated RTL from Tool, which will be better comparable to optimized RTL regular flow, and finally doing part of

Table 9 Implementation performance parameters

HW function	On-chip power	Dynamic power	Accelerated clock cycles $\times 10^9$
FC6,FC7	3.67 W	2.98 W	2.15
FC6,FC8	3.7 W	3.01 W	2.55
FC7,FC8	3.68 W	2.98 W	2.04
FC6-8	3.72 W	3.03 W	2.14

Table 10 Comparison between RTL and SDSOC implementation

Conv Layer1	LUTs	BRAMs 36 k	DSPs	FFs
Optimized RTL	32,968	288	105	9210
SDSOC	19,260	477	95	11,037

function on software gives more ability to implement large CNNs on the same evaluation board. This is the main part of using SDSOC flow for fast design time and fitting more large designs.

4.4 Design guidelines

The guidelines for implementation using SDSOC.

- Write the input C++ code with optimized techniques supported by SDSOC tool.
- Divide design into sub-functions considering the functions that will target hardware will be written in a good way that suits HW implementation.
- Using the useful pragmas that helps to decrease time and resources.
- After getting the implementation of design try to replace RTL generated code with another optimized one that will enhance performance.
- Finally, one of possible work in this area is to using partial dynamic reconfiguration (PDR).

5 Conclusion and future work

In this paper, the acceleration of the forward path of a pre-trained Alex-Net on FPGA is demonstrated, introducing the parallelism and pipeline techniques used to accelerate the CNN. The proposed architecture is discussed in details illustrating the achieved high speed and low power performance. Future work concerns deeper analysis of particular mechanisms, new proposals to try different methods. The next step is to compromise between the time required for the image prediction and the number of resources. In addition, some techniques can be used to reduce the power consumption as pruning and PDR. Furthermore, a new approach for implementation is proposed using SDSoC tool for hardware/software partitioning which improve design performance and increase the facility to implement more large designs especially for CNNs systems. Future work for Alex-net on SDSoC will focus on eliminates number of trials to get best implementation for design which functions will be implemented on HW. It is recommended for future work to try partial dynamic reconfiguration with SDSoC to reduce power consumption and enhance performance.

Acknowledgements This work was partially funded by Mentor Graphics and ONE Lab at Zewail City of Science and Technology, Egypt and Cairo University, Egypt. Also we acknowledge the previous team who did the Accelerating Deep Neural Networks Using FPGA and engineer mahmoud Kishky for his support in SDSOC tool and coding according to tool guidelines.

References

1. Coşkun, M., Uçar, A., Yildirim, Ö., & Demir, Y. (2017). Face recognition based on convolutional neural network. In *MEES*.
2. Chen, Z., & Huang, X. (2017). End-to-end learning for lane keeping of selfdriving cars. In *IVS*.
3. Sze, V., et al. (2017). Efficient processing of deep neural networks: A tutorial and survey. arXiv:1703.09039
4. Chen, Y., Krishna, T., Emer, J., & Sze, V. (2016). Eyeriss : An energy-efficient reconfigurable accelerator for deep convolutional neural networks. In *ISSCC*
5. Elnabawy, A., Abdelmohsen, H., Moustafa, M., Elbediwy, M., Helmy, A., & Mostafa, H. (2018). A low power CORDIC-based hardware implementation of Izhikevich neuron model. In *IEEE international NEWCAS*.
6. Nurvitadhi, E., et al. (2017). Can FPGAs beat GPUs in accelerating NextGeneration deep neural networks. In *ISFPGA*.
7. Suda, N., et al. (2016). Throughput-optimized opencl-based fpga accelerator for largescale convolutional neural networks. In *ACM*.
8. Qiu, K. J., et al. (2016). Going deeper with embedded fpga platform for convolutional neural network. In *ACM*.
9. Adel, E., Magdy, R., Mohamed, S., Mamdouh, M., El Mandouh, E., & Mostafa, H. (2018). Accelerating deep neural networks using FPGA. In *30th international conference on microelectronics (ICM)*.
10. Beam, A. (2017). Deep learning 101—part 1: History and background. In *Online*.
11. Krizhevsky, A. (2015). ImageNet classification with deep convolutional neural networks. In *Online*.
12. Krizhevsky, A., Sutskever, I., & Hinton, G. (2012) ImageNet classification with deep convolutional neural networks. In *NPIS*.
13. "Xilnix," [Online]. https://www.xilinx.com/support/documentation/sw_manuals/xilinx2015_2/ug1027-intro-to-sdsoc.pdf. Retrieved 20 July 2015.

14. [Online]. https://www.xilinx.com/support/documentation/user_guides/ug1085-zynq-ultrasc-ale-trm.pdf. Retrieved 17 Jan 2019.
15. "www.Xilnix.com," [Online]. https://www.xilinx.com/support/documentation/boards_and_kits/zcu104/ug1267-zcu104-eval-bd.pdf. Retrieved 9 Oct 2018.
16. Motamedi, M., Gysel, P., Akella, V., & Ghiasi, S. (2016). Design space exploration of FPGA-based deep convolutional neural networks. In *21st Asia and South Pacific design automation conference (ASP-DAC)* (pp. 575–580).
17. Kang, Y., Kim, S., Shin, T., & Chung, J. Running convolutional layers of AlexNet in neuromorphic computing system. In *NRF-2014R1A1A2A16055253*.
18. Gysel, P. M. (2012). Ristretto: Hardware-oriented approximation of convolutional neural networks. In *B.S. thesis, Bern University of Applied Sciences*, Switzerland

Rania Osama received the B.Sc. and M.Sc. degrees (with honors) in Electronics Engineering from Cairo University, Cairo, Egypt, and currently she is a PhD candidate at the Electronics Engineering from Cairo University, Cairo, Egypt.

Dr. Hassan Mostafa (S'01, M'11, SM'15) received the B.Sc. and M.Sc. degrees (with honors) in Electronics Engineering from Cairo University, Cairo, Egypt, in 2001 and 2005, respectively, and the PhD. degree in Electrical and Computer Engineering in the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON, Canada in 2011. He is currently an Associate Professor at the Nanotechnology and Nanoelectronics Program at Zewail City of Science and Technology, Egypt. Dr. Mostafa has worked as an NSERC postdoctoral fellow in the Department of Electrical and Computer Engineering, University of Toronto, Toronto, Ontario, Canada. His postdoctoral work includes the design of the next generation FPGA in collaboration with Fujitsu research labs in Japan/USA. He has authored/coauthored over 200 papers in international journals and conferences and is the author/co-author of 5 published books. His research interests include Neuromorphic computing, IoT hardware security, Software defined radio, reconfigurable low power systems, Analog-to-Digital Converters (ADCs), low-power circuits, Subthreshold logic, variation-tolerant design, soft error tolerant design, statistical design methodologies, next generation FPGA, spintronics, Memristors, energy harvesting, MEMS/NEMS, Power management, and Optoelectronics. Dr. Mostafa is a member of the IEEE Technical Committee of VLSI Systems and Applications (TC-VSA) since 2017 and an IEEE Senior Member since 2015. He was the recipient of University of Toronto Research Associate Scholarship in 2012, Natural Sciences and Engineering Research Council of Canada (NSERC) Prestigious PostDoctoral Fellowship in 2011, Waterloo Institute of Nano-technology (WIN) nanofellowship research excellence award in 2010, Ontario Graduate Scholarship (OGS) in 2009, University of Waterloo SandFord Fleming TA Excellence Award in 2008.