CNN hardware acceleration on a low-power and low-cost APSoC

Paolo Meloni, Antonio Garufi, Gianfranco Deriu, Marco Carreras and Daniela Loi
Department of Electrical and Electronic Engineering, University of Cagliari
Cagliari, Italy 09123
Corresponding author: paolo.meloni@unica.it

***Abstract*—Deep learning and Convolutional Neural Networks (CNNs) in particular, are currently one of the most promising and widely used classes of algorithms in the field of artificial intelligence, being employed in a wide range of tasks. However, their high computational complexity and storage demands limit their efficient deployment on resource-limited embedded systems and IoT devices. To address this problem, in recent years a wide landscape of customized FPGA-based hardware acceleration solutions has been presented in literature, focused on combining high performance and power efficiency. Most of them are implemented on mid- to high-range devices including different computing cores, and target intensive models such as AlexNet and VGG16.**

In this work, we implement a CNN inference accelerator on a compact and cost-optimized device, the Minized development board from Avnet, integrating a single-core Zynq 7Z007S. We measure the execution time and energy consumption of the developed accelerator, and we compare it with a CPU-based software implementation. The results show that the accelerator achieves a frame rate of 13 fps on the end-to-end execution of ALL-CNN-C model, and 4 fps on DarkNet. Compared with the software implementation, it was 5 times faster providing up to 10.62 giga operations per second (GOPS) at 80 MHz while consuming 1.08 W of on-chip power.

***Index Terms*—convolution neural networks, FPGAs, hardware accelerators**

I. Introduction

In last few years, Convolutional Neural Networks (CNNs) have been successfully applied in a variety of tasks such as image and video recognition, object identification, medical image analysis, and natural language processing, achieving excellent performances [1]. However, the high accuracy of CNNs comes at the cost of their high computational complexity and energy and storage consumption. As a result, the range of dedicated and tailored hardware platforms capable of reducing power consumption and speeding-up the most computation-intensive tasks, i.e. convolution layers, is steadily increasing.

Due to the intrinsic parallelism exhibited by convolution layers and the significant number of Multiply Accumulate (MAC) operations required, a wide community of developers is exploiting reconfigurable logic inside modern Field-Programmable Gate Arrays (FPGAs) for implementing CNN hardware accelerators [2]–[6]. Embedding a high number of parallel Digital Signal Processing (DSP) units, FPGAs allow to enable efficient implementation of MAC operations, to deliver high performance at limited clock frequencies, and to exploit spatial computation. Also leading tech giants, such as Xilinx and Intel, have started the production of specific FPGA acceleration solutions, supporting a growing ecosystem of business partners and customers who are interested in developing and deploying CNNs applications. However, to foster the use of FPGAs as accelerators for CNN applications a step further is needed. An effective accelerator architecture must be flexible and scalable to satisfy the specific needs of a target application and workload, and at the same time, must fit into FPGAs of different types and sizes providing attractive performance/cost and performance/Watt ratios. In this context, the NEURAghe architecture has proven to be a powerful accelerator on high-end Xilinx Zynq All Programmable System on Chips (APSoCs) [8]. NEURAghe exploits both the ARM-based processing core and the programmable logic in Xilinx Zynq-7000 devices to improve performance through parallelism and to widen the scope of CNN actors that can be supported, and, thus, the scope of complex algorithms that can be accelerated. The NEURAghe approach has been implemented on the Xilinx Zynq XC7Z045 APSoC costing around 2000$, and has been evaluated under the workload imposed by two state-of-the-art CNN models, ResNet-18 and VGG-16. However, not all applications require a big expense of computational power or real-time object recognition. In some use cases, implementing a similar approach on less power hungry and less expensive devices and exploiting multi-board FPGA designs can allow to achieve better performance than a single high-end FPGA.

In this paper an inference hardware accelerator is presented, which is capable of emphasizing performance over cost efficiency on light-weight CNN architectures, such as Darknet, All-CNN-C and SqueezeNet. The proposed architecture is implemented on a cost-optimized device, the Avnet MiniZed based on the Xilinx single-core Zynq XC7Z007S SoC [9]. The experiments show that the proposed design is capable of delivering up to x5 better computational throughput than CPU-based software implementation. Moreover, to the best of our knowledge, this is the first work to have addressed the execution of state-of-the art CNN on a low-power, ultra affordable development board.

The paper is organized as follows. In Section II the architecture of the proposed accelerator is described in detail.

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No. 780788.

978-1-7281-4074-2/19/$31.00 ©2019 IEEE

Section III provides an overview of the software framework that enables an efficient implementation of CNNs. Section IV describes the experimental results.

II. Hardware architecture

The architecture of the proposed low-cost CNN accelerator is based on a hybrid hardware-software scheme implemented on the Zynq XC7Z007S SoC, in which an ARM Cortex-A9 single-core processor system cooperates with a Convolution-Specific Processor deployed on the programmable logic to accelerate compute-bound and memory-bound operations, as shown in Figure 1. Besides the ARM core, the Processing System incorporates a memory interface unit, providing connections to an off-chip double data rate (DDR) memory, used to store input features, weights, biases and output features, a 8-channel PL330 Data Memory Access (DMA) controller, supporting multiple data transfer types (i.e. memory-to-memory, memory-to-peripheral, peripheral-to-memory), and a rich set of peripheral connectivity interfaces.

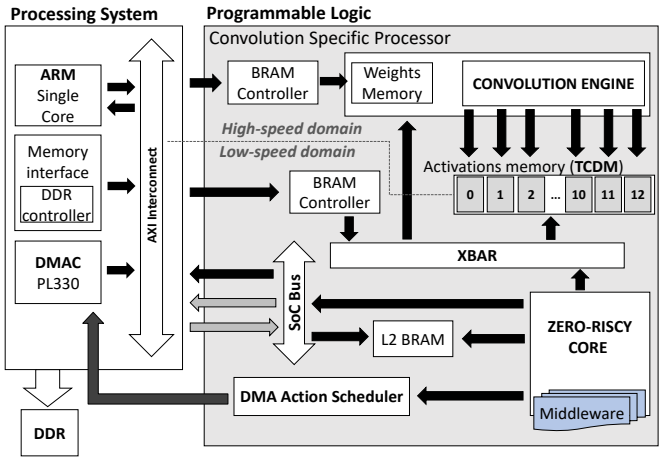

Fig. 1. Simplified block diagram of the proposed MiniZed-based architecture for CNN inference acceleration

The Convolution-Specific Processor consists of several components, all implemented using synthesizable SystemVerilog HDL:

- a Convolution Engine (CE) representing the computational core of the accelerator;
- a Weight Memory (WM) implemented with a $1024x64$ bit block BRAM;
- a local Tight-Coupled Data Memory (TCDM) implemented with 13 banks of dual-port BRAM primitives, 10 of them reserved to store output features and 3 for input features;
- a pair of AXI BRAM controllers, one for the transfer of weights to the Weight Memory and one for the transfer of activations to/from the Convolution Engine;
- a low cost RISC-V microcontroller for the synchronization of computation kernels and data transfers;
- a DMA Action Scheduler supporting and managing the communication between the Convolution Engine and the PL330 DMA device.

In the following, a detailed description of each major functional block is provided.

A. Convolution engine

The compute-bound convolution tasks are performed in the Convolution Engine by means of a Sum-of-Product (SoP) module, implemented using 54 Digital Signal Processing (DSP) slices (i.e. Xilinx DSP48E1 primitives). As schematized in Figure 2, the SoP module applies three $3x3$ weights filters to three input features x_in received by the line buffer, and computes their contribution to one output feature. The partial results of the convolutions are summed together using a dedicated Adder module.

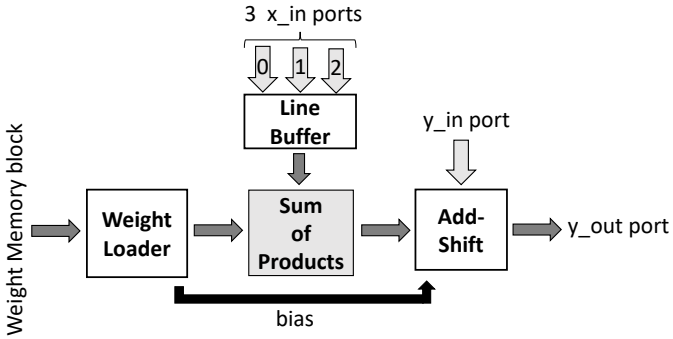

Fig. 2. Hardware architecture of the Convolution Engine

The line buffer is partitioned into three independent regions, which acquire three different streams of pixels corresponding to the 3 input features. The Convolution Engine works on 16-bit pixel data and the line buffer is fed with 2 pixels per cycle (32-bit words). The line buffer has a maximum depth of 128 32-bit words. This implies that the proposed configuration is capable of processing input features of $256x256$ pixels, which is large enough to compute all convolutional layers of CNNs such as ResNet, LeNet, SqueezeNet. The weights (27 in total) are supplied by the Weight Memory, implemented with a $1024x64$ bit block BRAM, through a Weight Loader state machine.

B. RISC-V based microcontroller

As shown in Figure 1, the proposed architecture embeds a Zero-riscy core [7]. It is a simple and low cost RISC-V processor optimized for low-power consumption and low-area occupancy (around 8080 cells). The Zero-riscy core is responsible for the synchronization of the computation kernels and data transfers in the proposed configuration.

The Zero-riscy core supports the following instructions:

- RV32I Base Integer Instruction Set;
- RV32E Base Integer Instruction Set, which is a reduced version of RV32I designed for embedded systems;
- RV32C Standard Extension for Compressed Instructions;

- RV32M Integer Multiplication and Division Instruction Set Extension.

The RV32M and RV32E can be enabled and disabled by using two parameters. The RV32C set extension reduces static and dynamic code size by adding short 16-bit instruction encodings for common operations. Typically 50%-60% of the RISC-V instructions in a program can be replaced with RVC instructions, resulting in a 25%-30% code-size reduction.

C. DMA Action Scheduler

Transferring data between the Processing System and the Programmable Logic is an essential and crucial task. The proposed architecture exploits the 32-bit General-Purpose ports for the transfer of small amount of data, and implements a DMA communication procedure for transferring large sets of data, i.e. input features, weight and output features. We configured the PL330 DMA controller [12] integrated in the Processing System of the target Zynq XC7Z007S SoC for setting up three DMA channels between the DDR memory and the Programmable Logic. Each channel is capable of supporting a single concurrent thread of DMA operation. We assigned a peripheral request interface to each DMA channel, as shown in Figure 3. Each interface consists of a peripheral request bus (p_request_bus) and a DMAC acknowledge bus (DMAC_ack_bus).

Since the Zero-riscy core cannot directly interact with these interfaces, we designed a a DMA Action Scheduler. It receives requests of data transfer from the Zero-riscy core via a dedicated bus (i.e. s_core_dmactrl_bus), and directs them to one of its 3 internal channels through a 1-to-3 decoder (see Figure 3). Channel 1 is meant for the input features loading, Channel 2 for the weights, and the third channel for the output features. Each channel is independent and consists of a FIFO and a 4-state FSM (Finite State Machine). The FIFO stores the requests from the Zero-riscy core while waiting for the previous ones to be satisfied, and the FSM manages the communication through the PL330 DMA controller's interfaces.

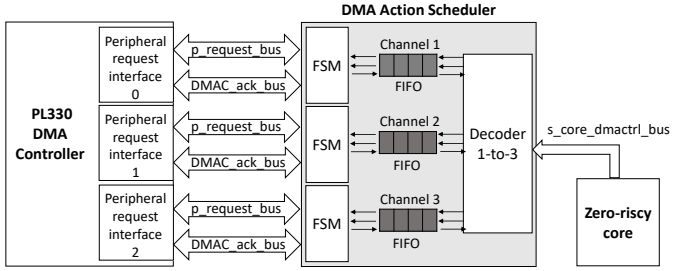

Fig. 3. Overview of the DMA communication between the PL330 DMA controller and the Zero-riscy core

III. Software Stack

The software stack is mainly composed of a front-end application, named ConvNet, which allows users to specify and program CNNs at software level, and of a back-end, also referred to as Middleware, which configures the hardware accelerator and triggers the convolution tasks.

The ConvNet application is run on the ARM core and takes care of all necessary tasks for the execution of a convolution layer:

- initialization of the Zynq SoC, which includes the loading of the Middleware into the instruction memory of the accelerator (i.e L2 BRAM, see Figure 1);
- configuration of the convolution layer's parameters (i.e. kernel size, layer size);
- loading of input features, biases and weights in the DDR;
- setup of DMA transfers to be performed by the DMAC PL330 by means of APIs;
- ordering of Middleware execution on the Zero-riscy core.

ConvNet waits until the Middleware has completed the convolution layer, then terminates.

The Middleware is run by the Zero-riscy core. It is composed of two parts: the first is responsible for setting up the convolution parameters into the programming registers of the Convolution Engine and for triggering the start of a new convolution, and the second part is responsible for scheduling computational kernels and data transfers between the TCDM, the Weight Memory and the DDR.

A. DMA transfer strategy

The scheduling strategy applied in the Middleware is optimized to improve efficiency under limited bandwidth availability. It exploits double-buffering method to optimize the overlapping of the computation tasks with data transfers by splitting both the TCDM and the Weight Memory into two partitions (ping and pong). Figure 4 schematizes the synchronization mechanism. The contribution of an Input Group (IG), composed of 3 input features, is being computed to an Output Group (1 output feature) while the input features and weights of the next IG are transferred from the Processing System to the Programmable Logic (see blue and yellow time segments in Figure 4). Convolution tasks cannot start until all the input features and weights have been completely transferred, therefore two barriers are used to stop the ConvNet application from ordering Middleware execution. The barriers are implemented as while-loops that force the ARM core to wait until the value '0' (data transfer completed) is read from the proper channel's address. When the calculation of an Output Group is completed, the result is transferred to the DDR.

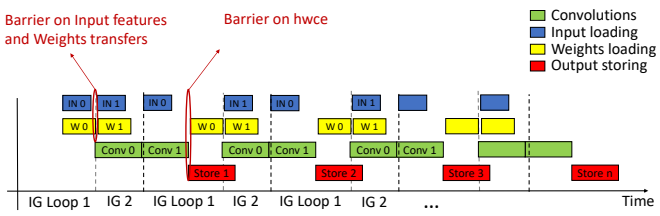

Fig. 4. Synchronization mechanism

B. Programming and performing DMA transfers

Figure 5 summarizes the three steps required for programming and performing DMA transfers.

In the first step, the ARM processor generates a program code, called micro-code, and stores it in a region of DDR memory accessible by the DMA controller. The micro-code contains all DMA transfers involved in a convolutional layer. It defines each DMA transfer as a sequence of instructions to be performed by the PL330 DMA controller's instruction execution engine.

In the second step, the ARM processor programs the DMA controller's internal registers to initialize operating mode and micro-code address. In the third step, after initialization, the PL330 DMA controller's instruction execution engine waits to be triggered by the Programmable Logic, through the Peripheral Request Interfaces, to execute microcode instructions.

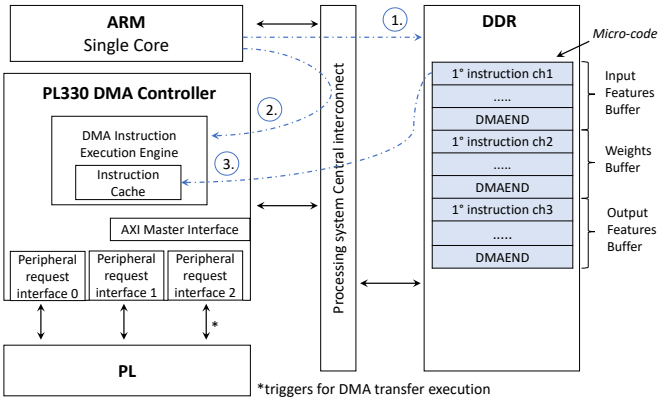

Fig. 5. Overview of the steps required for programming DMA transfers

C. DMA micro-code

The micro-code executed by the PL330 DMA controller defines the DMA transactions to be done, as a sequence of instructions.

Given the total amount of bytes to be transferred, the micro-code calculates how many loops are required to complete a transaction. Each loop can transfer 64, 32 or single bytes. The first instruction of the micro-code is Wait For a Peripheral Request. As soon as the PL330 DMA controller receives a request from the Zero-riscy core, all loops of 64 byte burst are executed. Loops are handled by setting a counter. The PL330 DMA controller is provided with 2 counters and each of them is able to count up to 256. If a loop contains more than 256 iterations, the second counter is enabled to perform a nested loop. A basic loop is composed of 4 instructions:

- DMALP specifies the number of loops to perform and sets the counter to use;
- DMALD loads the content of the source address to the DMA buffer;
- DMAST store the content to the location that the Destination Address Registers specifies;
- DMALPEND indicates the last instruction in the program loop.

Once the 64 byte bursts have been completed, the 32-byte loops are executed (where each DMALD/DMAST sequence transfers 32 bytes), and finally the spare words loops which move single bytes for each iteration. The last byte of a transfer is always sent as a single byte outside of any loop, implementing an acknowledge signal for the DMA Action Scheduler. At last, a DMAEND instruction states the ending of the micro-code.

IV. Experimental results

The proposed architecture has been implemented on the Avnet MiniZed board, based on Xilinx Zynq XC7Z007S SoC and costing around 100$. The ARM Cortex-A9 single-core in the Processing Systems is clocked at 667 MHz, while the Convolution Specific Processor operates at 80 MHZ. In this section, we first report the hardware resource utilization. Then, we evaluated the implementation of two state-of-the art CNN models on the proposed architecture, and we measured the speed up factor our approach yields compared to a software implementation using the ARM Compute Library [11]. Finally, we compared the results achieved in the Avnet Minized implementation with those achievable on a high-end device, to investigate the potential benefits of multi-board designs.

Table I shows the hardware resources consumed by the proposed accelerator, after optimization and full implementation on Xilinx Vivado 2017.1, using the Post-Route Physical Optimization strategy. The proposed configuration uses the 82% of the processing power available in the target SoC.

TABLE I
FPGA resources utilization of the accelerator when implemented on Zynq Z-7007s SoC

	DSP	BRAMs	LUTs	Flip-Flops
Available	66	50	14400	28800
Used	54	44	12610	12501
Utilization	82%	88%	87%	43%

Considering the limitations regarding the computational power and the size of the memory banks, we evaluated the performance and energy efficiency of the proposed architecture under the computational load imposed by two small and well-known CNN models: DarkNet Reference [13] and All-CNN-C [14]. These models both deal with typical computer vision tasks. The first network is designed to be small but powerful. It uses mostly convolutional layers and max pooling layers, without the large fully connected layers at the end. The All-CNN-C model is suitable for small dataset problems. In this network, max-pooling layers are replaced by regular convolutional layers with a stride of 2.

Table II and Table III report performance and efficiency results achievable by the proposed MiniZed-based architecture on the DarkNet and ALL-CNN-C models, when using 16- and 8-bit data precision. Performance is evaluated in terms of execution time (ms) and speed (GOps/s).

Considering 8-bit data precision results, our accelerator achieves a frame rate of 13 fps on the end-to-end execution of ALL-CNN-C model, and 4 fps on DarkNet. In Figure

TABLE II

EXECUTION TIME, PERFORMANCE AND EFFICIENCY RESULTS FOR THE IMPLEMENTATION OF THE DARKNET REFERENCE MODEL ON MINIZED, WHEN USING 16- AND 8-BIT DATA PRECISION.

DarkNet Reference architecture (Layer name; Layer description)	Input image size	Input Features	Output Features	Workload [GOPs]	Time [ms] (16bit; 8bit)	Speed [GOps/s] (16bit; 8bit)	Efficiency (16bit; 8bit)
Conv_1; (3x3) Conv.	256x256	3	16	0,0566	25,52; 11,76	2,21; 4,82	26%; 56%
Conv_2; (3x3) Conv.	128x128	16	32	0,1509	27,36; 12,48	6,21; 13,61	72%; 79%
Conv_3; (3x3) Conv. stride 2	64x64	32	64	0,1509	21,41; 10.31	7,27; 15,10	84%; 87%
Conv_4; (3x3) Conv.	32x32	64	128	0,1509	19,57; 10,12	7,95; 15,38	92%; 21%
Conv_5; (3x3) Conv.	16x16	128	256	0,1509	21,31; 11,98	7,14; 12,70	83%; 74%
Conv_6; (3x3) Conv. stride 2	8x8	258	512	0,1509	33,05; 20,83	4,60; 7,31	53%; 42%
Conv_7; (3x3) Conv.	4x4	512	1024	0,1509	97,32; 79,21	1,55; 1,91	18%; 11%
Conv_8; (1x1) Conv.	4x4	1024	1000	0,0327	155; 97,93	0,21; 0,33	2%; 2%

TABLE III

EXECUTION TIME, PERFORMANCE AND EFFICIENCY RESULTS FOR THE IMPLEMENTATION OF THE ALL-CNN-C ALGORITHM ON MINIZED, WHEN USING 16- AND 8-BIT DATA PRECISION.

All-CNN-C architecture (Layer name; Layer description)	Input image size	Input Features	Output Features	Workload [GOPs]	Time [ms] (16bit; 8bit)	Speed [GOps/s] (16bit; 8bit)	Efficiency (16bit; 8bit)
Conv_1; (3x3) Conv.	32x32	3	96	0,0053	2; 1,09	2,65; 4,86	31%; 28%
Conv_2; (3x3) Conv.	32x32	96	96	0,1699	22,8; 11	7,45; 15,44	86%; 89%
Conv_3; (3x3) Conv. stride 2	32x32	96	96	0,1699	26,8; 14,9	6,34; 11,40	73%; 66%
Conv_4; (3x3) Conv.	16x16	96	192	0,0849	12; 6,8	7,08; 12,49	82%; 72%
Conv_5; (3x3) Conv.	16x16	192	192	0,1699	23,7; 13,3	7,17; 12,77	83%; 74%
Conv_6; (3x3) Conv. stride 2	16x16	192	192	0,1699	26,4; 16	6,43; 10,62	74%; 61%
Conv_7; (3x3) Conv.	8x8	192	192	0,0425	9,9; 6,6	4,29; 6,43	50%; 37%
Conv_8; (1x1) Conv.	6x6	192	192	0,0027	6,8; 6,15	3,51; 3,88	41%; 22%
Conv_9; (1x1) Conv.	6x6	192	10	0,0001	0,74; 0,9	1,68; 1,38	19%; 8%

6, we compared these results with those obtained with a pure software implementation, executed on a dual-core ARM Cortex-A9 at 800MHz using the ARM Compute Library and QASYMM8 data. The proposed hardware accelerator is 5 times faster providing up to 10.62 giga operations per second (GOPS) at 80 MHz while consuming 1.08 W of on-chip power. We used the Xilinx Power Estimator (XPE) tool to calculate the power consumption of the SoC system, which amounts to around 2 W when considering the overall board consumption.

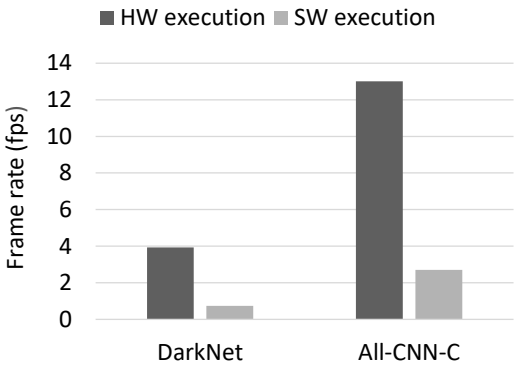

Fig. 6. Comparison of frame rates achieved by DarkNet and All-CNN-C in the MiniZed-based implementation (HW execution) with those achieved in pure software execution on ARM dual-core processor.

To compare the results achieved on Minized with those achievable on a high-end device, we implemented the 16-bit version of the accelerator on a Xilinx ZC706 board that contains a Xilinx XC7Z045 chip with 900 DSP slices and costs around 2800$. The clock frequency of such accelerator implementation is 140 MHz. Darknet and SqueezeNet [**?**] models were used to test the performance and energy efficiency of the proposed accelerator on the two boards. The results are shown respectively in Figure 7 and Figure 8.

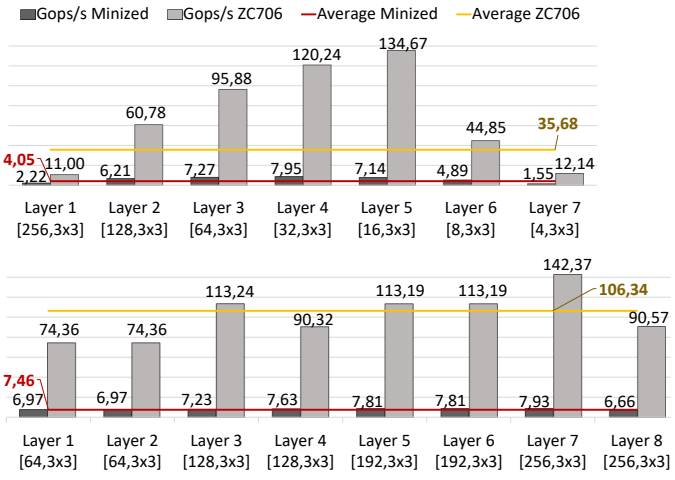

Fig. 7. Comparison of the performance achieved across all the convolutional layers by the Minized and the ZC706 on DarkNet (top) and SqueezeNet (bottom)

Both implementations perform better on SqueezeNet. This because performances are strictly dependent on the layer's parameters, such as image size and number of input and output features. In DarkNet, the first and the last layer are responsible

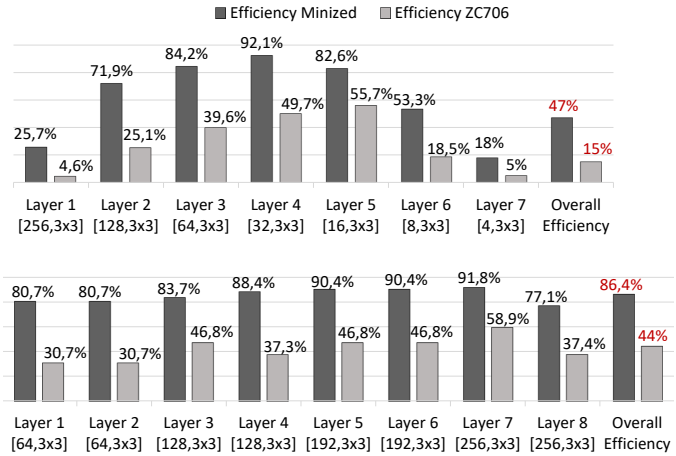

Fig. 8. Comparison of the efficiency achieved across all the convolutional layers by the Minized and the ZC706 on DarkNet (top) and SqueezeNet (bottom)

for a performance drop. As shown in Table II, the first layer takes as input 3 features of 256x256 pixels each and provides 16 output features of the same size as output. This means that tiling is required. The last layer takes 512 input features of size 4x4 and gives 1024 output features back. In SqueezeNet, the lowest efficiency is obtained on the last layer (77%), where the image size is the smallest (14x14). The highest efficiency (91,81%) is reached with a 28x28 image size, so tiling is not necessary.

On both networks the single Minized implementation achieves higher energy efficiency levels but reports lower average throughput, compared with the ZC706. As summarized in Table IV for SqueezeNet, taking into account the power consumption of the two boards, the XC7Z007S implementation shows around three times less GOps/s per each Watt consumed.

TABLE IV
PERFORMANCE SUMMARY ON SQUEEZENET: GOPS/S PER WATT ACHIEVED BY THE TWO APSOCS

Board used	Price ($)	Clock (MHz)	Power (W)	Performance/Power (GOps/s per Watt)
MiniZed	100	80	2.5	2.98
ZC706	2800	140	10	10.63

The implementation of multi-board designs can provide some benefits. For example, replacing a single XC7Z045 with 14 XC7Z007S to execute a task based on SqueezeNet, allows to achieve same good performance (106.34 GOps/s) saving around 50% of costs.

V. CONCLUSION

In this paper we have presented the design, implementation and evaluation of a CNN inference accelerator on a low-power and low-cost device. We have shown that implemented on the Xilinx single-core Zynq XC7Z007S SoC, our 80 MHz accelerator is 5 times faster than a CPU-based software implementation in processing light-weight CNN architectures, such as Darknet and All-CNN-C, while consuming 1.08 W of on-chip power. We have demonstrated that it can actually run on low power hardware, providing very good performance over cost efficiency.

REFERENCES

[1] Y. LeCun, Y. Bengio and G. Hinton, "Deep learning," Nature, vol. 521(7553), pp. 436-444, 2015.

[2] A. Shawahna, S.M Sait, and A.H. El-Maleh, "FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review," IEEE Access, vol. 7, pp. 7823–7859, 2019.

[3] K. Abdelouahab, M. Pelcat, J. Srotand F. Berry, "Accelerating CNN inference on FPGAs: A Survey," CoRR, abs/1806.01683, 2018.

[4] C. Wang, L. Gong, Q. Yu, X. Li, Y. Xie and X. Zhou, "DLAU: A Scalable Deep Learning Accelerator Unit on FPGA," IEEE Trans. Comput.-Aided Design Integr. Circuits Syst., vol. 36, pp. 513–517, 2017.

[5] S. Mittal, "A survey of FPGA-based accelerators for convolutional neural networks," Neural Comput. Appl. pp. 1–31, 2018.

[6] D. Pani, P. Meloni, G. Tuveri, F. Palumbo, P. Massobrio, and L. Raffo, "An FPGA Platform for Real-Time Simulation of Spiking Neuronal Networks,". Front. Neurosci. 2017.

[7] P. D. Schiavone, F. Conti, D. Rossi, M. Gautschi, A. Pullini, E. Flamand and L. Benini, "Slow and steady wins the race? A comparison of ultra-low-power RISC-V cores for Internet-of-Things applications," 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS), pp. 1–8, 2017.

[8] P. Meloni, A. Capotondi, G. Deriu, M. Brian, F. Conti, D. Rossi, L. Raffo and L. Benini, "NEURAghe: Exploiting CPU-FPGA Synergies for Efficient and Flexible CNN Inference Acceleration on Zynq SoCs," ACM TRETS, vol. 11, pp. 1–24, 2018.

[9] Xilinx Inc. (2019, Apr.) Avnet MiniZed. [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-odbhjd.html.

[10] Xilinx Inc. (2019, Jan.) Zynq-7000 all programmable SoC technical reference manual. [Online]. Available: http://www.xilinx.com/support/documentation/user_guides/ug585-Zynq-7000-TRM.pdf.

[11] ARM (2019, Apr.) ARM Compute Library. [Online]. Available: https://developer.arm.com/compute-library.

[12] ARM (2019, Apr.) AMBA DMA Controller DMA-330 Technical Reference Manual. [Online]. Available: https://static.docs.arm.com/ddi0424/b/DDI0424B_dma330_r1p0_trm.pdf

[13] J. Redmon. (2019, Febr.) Darknet: Open Source Neural Networks in C. [Online]. Available: https://pjreddie.com/darknet/.

[14] J. T. Springenberg, A. Dosovitskiy, T. Brox and M. A. Riedmiller, "Striving for Simplicity: The All Convolutional Net," CoRR abs/1412.6806, 2014.

[15] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and 1MB model size," CoRR abs/1602.07360, 2016.