2019 7th International Engineering, Sciences and Technology Conference (IESTEC)

A 3D convolution accelerator implemented on FPGA using SDSoC

Jhon Ordoñez[+] and Guillermo Sahonero-Alvarez[*]
Centro de Investigación, Desarrollo e Innovación
en Ingeniería Mecatrónica
Universidad Católica Boliviana "San Pablo"
La Paz, Bolivia
Email: [+]jhon.ordonez@ucb.edu.bo, [*]guillermo.sahonero@ucb.edu.bo

***Abstract*—Convolutional Neural Networks (CNNs) have become a popular and useful deep learning algorithm. However, the requirements of computation have also increased. As implementation of CNNs in Embedded Systems or Cyber-Physical Systems (CPS) is required, the need of efficient technologies of computation like FPGAs arise substantially. In this paper, we present a 3D convolution accelerator implemented on a Xilinx ZCU102 FPGA board. It achieves 32.08 GOP/s of performance and an efficiency of 3.58 GOP/s per Watt. This accelerator has been developed in Xilinx SDSoC environment.**

Keywords—3D convolution, FPGAs, SDSoC

I. INTRODUCTION

In the last years, Convolutional Neural Networks have shown their potential in computer vision applications. In order to improve performance, researchers have developed many architectures of CNNs, for instance, AlexNet, VGG, LeNet, ResNet, etc. As deep neural networks increase their performance, these also demand more powerful computation. Therefore, hardware optimization for such algorithms is necessary.

Graphics Processing Unit (GPU) has become popular for inference and training of CNNs. Its architecture consists of parallel processors that accelerate the processing of tensors commonly used in CNNs. However, GPUs are limited by power dissipation requirements and high consumption, making them unsuitable for embedded systems [1].

The alternative to the usage of GPUs are Field Programmable Gate Arrays (FPGAs) which are well known for their energy efficiency. FPGAs have many advantages such as reconfigurability, flexibility, high performance, and tools that reduce development time. On the other hand, Application-Specific Integrated Circuits (ASIC), which are another type of hardware resource, lack of flexibility as these devices are not programmable or reconfigurable. In consequence, FPGAs have a shorter design time than ASICs [2].

A CNN is composed of three different types of layers: Convolutional layer (CONV), Pooling layer (POOL) and Fully Connected layer (FC). CONV layer extracts information from N feature maps and produces M outputs which are generated by M filters of dimension $k \times k$, where k is the kernel size. The filter multiplies each element of a section in the feature map. After multiplication, a bias is added. The filter moves through the feature map with a certain stride. Furthermore, in the borders a padding can be applied. That means to add zeros around the feature map in order to keep the same dimensions after convolution. Finally, an activation function is applied, typically ReLU. POOL layer reduces dimensionality to avoid data redundancy: max and average pooling are usually employed for this stage. FC layer appears at the end of the network to compute classification task. More than 90% of operations in a CNN involves convolutions [2]. Consequently, implementations should focus on parallel computation using multiple multiply-and-accumulate (MAC) units. In this work, a 3D convolution accelerator is proposed, and the design is implemented on a Xilinx ZCU102 FPGA board. Our results are encouraging as they show that our design can strongly compete with other works.

This paper is organized as follows: Section II analyzes related works, Section III describes hardware accelerator implementation, results and discussion are presented in Section IV and, finally, conclusions are made in Section V.

II. RELATED WORK

Different optimization techniques and approaches for convolution have been developed. Nevertheless, two of them are studied here: data format precision and development environment.

A. Data format precision

Depending on bit-width precision, resource utilization and computational complexity can be reduced, However, a short data format reduces accuracy: an evaluation of the trade-off between bit-width precision and resources is imperative. Figure 1 shows the quantity of implementations in different bit-width precision formats, being 16-bit fixed point the most used. Several implementations have shown that using fixed point format does not reduce significantly final accuracy [2]–[4]. Therefore, our implementation compares resource utilization between 8 and 16-bit fixed point formats.

978-1-7281-1691-4/19/$31.00 ©2019 IEEE
DOI 10.1109/IESTEC46403.2019.00126

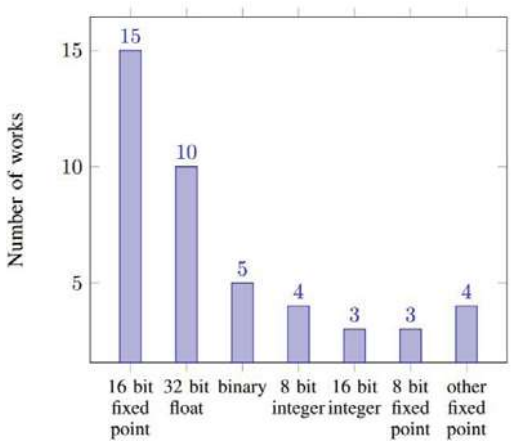

Fig. 1. Number of works with differents bit-width precisions

B. Development environment

In FPGA applications is important to select an appropriate Integrated Development Environment (IDE). There is a variety of software to synthesize and implement projects on FPGAs. However, most developers prefer Vivado HLS as shown in figure 2. This due to it offers short development time in a familiar C/C++ programming language instead of HDLs (Hardware Description Language).

Most related works were implemented on Xilinx Zynq boards, for which, SDSoC offers a suitable environment as it provides an optimized compiler and an automated software acceleration in Programmable Logic (PL). Hardware optimization is carried out by pragma directives. Moreover, SDSoC eases communication between Processing System (PS) and PL due to automatic AXI (Advanced eXtensible Interface) channels generation.

Danopoulos et al. [5] accelerated a CNN for image classification with SDSoC and Caffe integration. The accelerator was implemented on Zynq 7020 and achieved up to 98x speed-up compared with ARM CPU implementation. They constructed a highly a pipelined architecture with minimum latency that performed MAC operations efficiently. On the other hand, Amiri et al. [6] developed a binarized CNN on a Zynq 7020 using SDSoC directives and HLS pragmas like *array_partition, SDS async* and *SDS wait.* This work reported an inference speed up from 29.68 to 90.82 images/sec.

Most of the previous works were focused on PE (Processing Element) optimization. Huang et al. [4] proposed a computation module that contains several Winograd PEs and an accumulator. PEs fetch the input data from on-chip buffers and send the processed data to the accumulator buffer. The cached output results are written back to an off-chip memory. In contrast, Ahmed et al. [10] have designed a full custom MAC unit with 4 input feature maps elements. This design does not depend on using any dedicated multiplier nor embedded DSP blocks. Moini et al. [7] and Su et al. [9] also used MACs as computation elements. In [7], each MAC module is used to calculate the pixel for one output feature map and has a small kernel memory that contains coefficients. Input data is fed to the MAC modules, it is multiplied in the corresponding pixel from their kernel memory and summed the final result. Su et al. [9] suggested to map the 3D convolution to a matrix multiplication operation.

Although Vivado HLS is widely used, it is not totally adequate for Zynq and Zynq Ultrascale platforms because CPU-FPGA communication is complex. SDSoC environment should be evaluated for 3D convolution accelerator implementations on FPGAs that require CPU coprocessing.

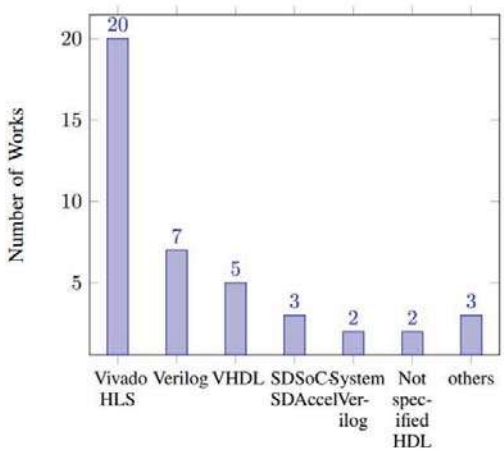

Fig. 2. Integrated Development Tools

III. PROPROSED ACCELERATOR SCHEMA

The accelerator has a pipelined architecture as shown in the pseudocode (Figure 4). It operates 3D convolution of different sizes of input feature maps, up to 256×256 and 64 channels with a kernel size of 3. Furthermore, it can operate two bit-width formats: 8-bit fixed point (4-fractional bits) and 16-bit fixed point (8-fractional bits). Fractional bits are easily configurable by using *ap_fixed* library from Vivado HLS. Figure 3 shows the general accelerator schema.

A. Architecture

This architecture has the following components:

1) *FIFO IN/FIFO OUT:* These FIFOs are responsible for storing momentaneously data from PS to accelerator (PL) and viceversa. Both have been declared as HLS Streams with depth of 32 and consume BRAM (Block RAM) resources. Data from PS were allocated through SDSoC function *sds_alloc()* and the directive: *#pragma SDS data zero_copy.*

2) *Buffers:* Each buffer stores $(K-1)$ rows of the input feature map, where K is the kernel size. In our implementation $K = 3$ was selected. The directive *#pragma HLS array partition complete* $dim = 1$ implements each buffer as a BRAM partitioned in 2 independent memories.

3) *Register Bank:* All kernel coefficients are stored as registers in order to access them immediately. This configuration is carried out by *#pragma HLS array partition complete* $dim = 0$ directive.

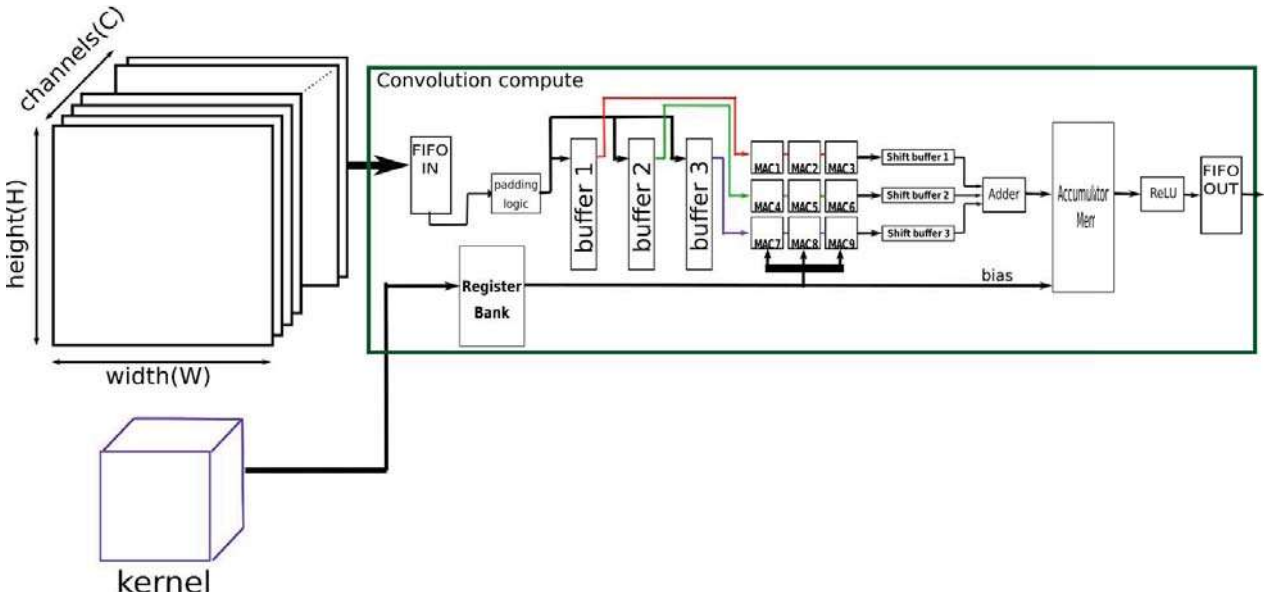

Fig. 3. Hardware accelerator schema for 3x3 kernel

4) *MAC units:* Each unit multiplies and accumulates data from buffers (input feature map) and register bank (kernel). It executes 3 vector convolutions simultaneously. *#pragma HLS dependence* inter false directive provides extra information to the compiler in order to avoid conflicts in the for loop.

5) *Shift buffers and adder:* In order to manage each vector convolution result, shift registers control dataflow from MAC units. Finally, an adder sums all independent results. Shift registers were also partitioned like the register bank.

6) *Accumulator Mem:* This memory adds the current output of convolution and the previous one. Once all channels are operated, bias is added. Accumulator Mem is the largest memory in the architecture, and it was partitioned using *#pragma HLS array partition complete* $dim = 1$ directive.

7) *ReLU Module:* It executes ReLU activation function.

SDSoC provides functions that are translated to hardware. The complete process of compilation includes synthesis, implementation, bitstream generation and integration of binary file for PL and executable file for PS. The main functions and directives used in our implementation are:

- **sds alloc():** This function allows that PL can access a certain section of the DDR memory located in PS.
- **#pragma HLS pipeline:** It allows the concurrent execution of operations.
- **#pragma HLS dataflow:** This directive is used to compute 8 kernels simultaneously for convolution.
- **#pragma HLS array partition:** Implements variables as storage elements either BRAM or register. Moreover, it splits an array into smaller arrays or individual elements.

```
1: sds_alloc(Input,Output,kernel)
2: read_input(Input,inStream)
3: for k ∈ {0,...,CHANNELS} do
4:    initialize(kernel, buffers,
      shift_registers,accumulator_mem)
5:    #pragma HLS array partition kernel / shift_registers /
      buffers / accumulator_mem
6:    for i ∈ {0,...,Height + 2 * padding} do
7:       for j ∈ {0,...,Width + 2 * padding} do
8:          #pragma HLS DEPENDENCE buffers / accumu-
            lator_mem inter false
9:          #pragma HLS PIPELINE
10:         if padding-condition then
11:            read_stream_data(inStream)
12:         end if
13:         store_buffers()
14:         window_convolution()
15:         store_shiftregisters()
16:         sum_shiftregisters()
17:         store_accumulate_accumulator_mem()
18:         if k =CHANNELS then
19:            ReLU_write_data(output)
20:         end if
21:      end for
22:   end for
23: end for
```

Fig. 4. Pseudocode for the SDSoC algorithm

- **#pragma HLS dependence:** Provides additional information when different addresses of a memory are accessed.

B. Experiment setup

The proposed accelerator was implemented on a Xilinx Zynq ZCU102 Ultrascale. This evaluation board provides designers a rapid prototyping platform and many useful resources. The maximum frequency is 1.5 GHZ for PS and 300 MHz for PL.

All binary and executable files are transferred to the target through a SD card. The board runs Linux operating system and its terminal is accessed by serial communication.

IV. Results and Discussion

The accelerator was designed by using SDSoC and implemented on a ZCU102 board. Table I shows the comparison of the resource utilization of using 8 and 16-bit fixed point format. Moreover, 8 kernels are computed in parallel. There is a considerable difference in BRAM and DSP consumption. Both implementations consume a little percentage (4.29%) of DSPs, only 108 of 2520 available. The block RAM usage is equal to 58.33 %. This is because our design has made use of many memories for input data, kernel coefficients and accumulators.

TABLE I. RESOURCE UTILIZATION

Resource	*8-bit*		*16-bit*	
	used	*% utilization*	*used*	*% utilization*
LUT	122 106	44.55	122 722	44.78
FF	135 894	24.79	181 186	33.05
BRAM	624	34.21	1064	58.33
DSP	36	1.43	108	4.29

Our hardware accelerator executes 0.464 Giga operations in 14.46 µs for 16-bit width precision where execution time was estimated by measuring the number of cycles of the external clock of ZCU102 board which runs on 1200 MHz. Additionally, this implementation only dissipates 8.97 watts and achieves a performance of 32.08 GOP/s and an efficiency of 3.58 GOP/s per watt. Thus, our implementation is suitable for embedded systems.

In Table II, a comparison with related works is presented. Our performance is similar to [7] and [9], but, by recalling Hu et al. [8] which implementation consumes more than 2000 DSPs, our implementation requires of less DSPs. As many DSPs in usage increase power consumption, our design needs less energy to achieve similar performance. On the other hand, regarding the work presented by Ahmed et al. [10], the shorter format (8-bit fixed point) and zero utilization of DSPs is overcame by our design due to the few DSP's resource employment. In terms of efficiency, [7] reported a low power consumption (less than 10) and a performance of 38.4 resulting 3.84 GOP/s per watt in efficiency, which is similar to ours (3.58 GOP/s per watt).

From the programming perspective, [7] and [10] used hardware description languages while we propose a high-level design to simplify the acceleration process. On the other hand, Xilinx SDAccel/openCL is proposed for parallel programing across heterogeneous platforms like KCU1500 [8].

V. Conclusion

This paper proposes a FPGA-based accelerator for large scale 3D convolution in order to accelerate the inference task in CNNs. It computes 8 kernels in parallel, reducing latency. The optimization has been developed by means of SDSoC functions and pragma directives which allow us to focus on efficient hardware. The accelerator is implemented on Xilinx Zynq Ultrascale ZCU102 platform. It achieves a performance of 32.08 GOP/s and an efficiency of 3.58 GOP/s per watt, which is suitable for power limited embedded applications. Additionally, implementation for 16-bit fixed point consumes more BRAMs than 8 bits, but in both implementations DSP utilization is low (less than 5%). Finally, it should be remarked that SDSoC should be explored because it reduces the development time for Zynq applications.

References

[1] H. Yonekawa y H. Nakahara, «On-chip memory based binarized convolutional deep neural network applying batch normalization free technique on an FPGA,» *Proceedings - 2017 IEEE 31st International*

TABLE II. COMPARISON WITH PREVIOUS WORKS

	Moini et al. [7]	*Hu et al. [8]*	*Sun et al. [9]*	*Ahmed et al. [10]*	*Our work*
Performance (GOP/s)	38.4	54.6	49.31	4.17	32.08
Frequency (MHz)	150	Not reported	100	834	50
DSP utilization	391	2184	198	0	108
Precision	16 fixed	16 fixed	16 fixed	8 fixed	16 fixed
Device	Xilinx XC7Z045	Xilinx KCU1500	Xilinx XC7Z020	Intel Arria 10	Xilinx ZCU102
IDE	Vivado-Verilog	SDAccel	Vivado HLS	VHDL	SDSoC

Parallel and Distributed Processing Symposium Workshops, IPDPSW 2017, pp. 98-105, 2017.

[2] F. Sun, C. Wang, L. Gong, Y. Zhang, C. Xu, Y. Lu, X. Li y X. Zhou, «UniCNN: A Pipelined Accelerator Towards Uniformed Computing for CNNs,» *International Journal of Parallel Programming,* vol. 46, pp. 776-787, 2018.

[3] C. N. Networks, N. Cui, D. Zhang, J. Liu y X. Zhou, «A High-efficiency FPGA-based Accelerator for Convolutional Neural Networks using Winograd Algorithm A High-efficiency FPGA-based Accelerator for Convolutional Neural Networks using Winograd Algorithm,» 2018.

[4] S. Moini, B. Alizadeh, M. Emad y R. Ebrahimpour, «A Resource-Limited Hardware Accelerator for Convolutional Neural Networks in Embedded Vision Applications,» *IEEE Transactions on Circuits and Systems II: Express Briefs,* vol. 64, pp. 1217-1221, 2017.

[5] Y. Ma, Y. Cao, S. Vrudhula y J. Seo, «Optimizing the Convolution Operation to Accelerate Deep Neural Networks on FPGA,» pp. 1-14, 2018.

[6] L. Hu, Recent Developments in Intelligent Computing, Communication and Devices, vol. 555, Springer Singapore, 2017, pp. 651-657.

[7] L. Gong, C. Wang, C. Li, H. Chen y X. Zhou, «MALOC : A Fully Pipelined FPGA Accelerator for Convolutional Neural Networks with All Layers,» vol. XX, pp. 1-12, 2018.

[8] D. Danopoulos, C. Kachris y D. Soudris, «Acceleration of image classification with Caffe framework using FPGA,» *2018 7th International Conference on Modern Circuits and Systems Technologies, MOCAST 2018,* pp. 1-4, 2018.

[9] M. Amiri, M. Hosseinabady, S. Mcintosh-smith y J. Nunez-yanez, «Multi-Precision Convolutional Neural Networks on Heterogeneous Hardware,» pp. 425-430, 2018.

[10] H. O. Ahmed, M. Ghoneima y M. Dessouky, «Concurrent MAC Unit Design using VHDL for Deep Learning Networks on FPGA,» *2018 IEEE Symposium on Computer Applications & Industrial Electronics (ISCAIE),* pp. 31-36, 2018.