International Journal of Computing and Digital Systems
ISSN (2210-142X)
Int. J. Com. Dig. Sys. 11, No.1 (Jan-2022)

https://dx.doi.org/10.12785/ijcds/110136

FPGA-based Acceleration for Convolutional Neural Networks on PYNQ-Z2

Thang Viet Huynh

Faculty of Electronics and Telecommunication Engineering, The University of Danang - University of Science and Technology, Danang city, Vietnam

Received 22 Sep. 2020, Revised 24 Dec. 2021, Accepted 4 Jan. 2022, Published 20 Jan. 2022

Abstract: Convolutional neural network is now widely used in computer vision and deep learning applications. The most compute-intensive layer in convolutional neural networks is the convolutional layer, which should be accelerated in hardware. This paper aims to develop an efficient hardware-software co-design framework for machine learning applications on the PYNQ-Z2 board. To achieve this goal, we develop hardware implementations of convolutional IP core and use them as Python overlays. Experiments show that the hardware implementations of the convolutional IP core outperform their software implementations by factors of up to 9 times. Furthermore, we make use of the designed convolutional IP core as hardware accelerator in the handwritten digit recognition application with MNIST dataset. Thanks to the use of the hardware accelerator for the convolutional layers, the execution performance of the convolutional neural network has been improved by a factor of 6.2 times.

Keywords: FPGA, Convolutional Neural Network, Hardware Accelerator, Python, PYNQ

1. Introduction

In recent years, convolutional neural networks (CNN) have become very popular in deep learning area. CNNs combine advantages of the convolutional filtering operations and the traditional artificial neural networks in both feature extraction and classification. It has been used in a variety of applications requiring high accuracy, such as image classification [1], [2], speech recognition [3], or self-driving cars [4]. Since the applications of CNNs is now becoming more complex, the number of layers and computational operations in the CNN architectures are rapidly increasing, thereby requiring a large amount of computing resources and memory storage.

To overcome this problem, many researchers have proposed various architectures and techniques to accelerate the inference of CNNs. With respect to hardware implementations, three hardware platforms can be used as CNN accelerators: graphic processing units (GPU) [5], application-specific integrated circuits (ASICs) [6], and field programmable gate arrays (FPGAs) [7], [8]. Among these platforms, FPGA has revealed itself as the high-performance and low-cost embedded device very suitable for the hardware prototyping of convolutional accelerators. Recently, new FPGA hardware solutions have been introduced, allowing for an efficient hardware and software co-design framework for the deep learning applications. The PYNQ [9] is an open-source project from Xilinx that makes it easier to develop FPGA based deep learning applications, in which designers can efficiently combine the benefits of programmable logic and microprocessors using the Python language and libraries.

The typical architecture of CNNs often comprises of many layers. The three common types of layers are convolutional layer, subsampling layer and fully-connected layer. The most compute-intensive layer in CNN is the convolutional layer. Therefore, to accelerate the inference of CNNs, the convolutional layer should be deployed in hardware.

In this work, we aim to develop an efficient hardware-software co-design framework for machine learning applications on the PYNQ board. To achieve this goal, we will implement the convolutional layer in the programmable logic of the FPGA hardware while keeping the other layers of the network executed in the software microprocessor for flexibility. The Xilinx ZYNQ SoC based PYNQ-Z2 device [10] is used in this work. The scientific contributions of this paper are the followings:

- We design and implement in VHDL (Very High Speed Integrated Circuits Hardware Description Language) a 2D convolutional intellectual property (IP) core fully synthesizable for FPGA.

E-mail address: thanghv@dut.udn.vn

- The designed 2D convolutional IP core is then used as a hardware accelerator to accelerate the inference of a convolutional neural network for the handwritten digit recognition application with the MNIST dataset on the PYNQ-Z2 FPGA board.

The remaining of this paper is organized as follows. Section 2 presents the background of the paper. Section 3 shows the architectual design and hardware implementation of the proposed 2D convolution IP core targeting for Xilinx PYNQ FPGA, followed by the evaluation of the designed IP core on the Xilinx PYNQ-Z2 device. The application of the designed IP core in convolutional neural network for the handwritten digit recognition is presented in detail in Section 4. In Section 5, we summarize our work and sketch out future research directions.

2. Background

A. Convolutional Neural Networks

The convolutional neural network is a commonly used deep learning model for image processing and computer vision. By combining feature extraction and classification, CNN can offer very high accuracy recognition results. A typical architecture of CNN, the LeNet network adopted from [11], is shown in Figure 1. CNN consists of three main types of layers: convolutional layers, subsampling layers and fully-connected layers.

The convolutional layer performs the two-dimensional (2D) convolution between the input data and the kernel, then an activation function is applied to the convoluted result to produce a feature map. The kernel size is normally 3x3 or 5x5 elements. A ReLu (rectified linear unit) activation function is often used in CNNs. There are often many kernels used in each convolutional layer of the CNN to produce many feature maps in order to extract different types of features from the input data.

The subsampling layer performs the reduction of the spatial size of feature maps from its previous convolutional layer. It is useful to extract the dominant features which are rotational and positional invariant, thereby maintaining the effectiveness of the training process of the model. The subsampling layer is also useful to reduce the computational complexity of the network. There are two types of subsampling operations: max-pooling and average-pooling, for which the max-pooling is preferable as it performs better than the average-pooling. The commonly used subsampling operation is the 2x2 max-pooling.

There are multiple pairs of convolutional and subsampling layers concatenating in a convolutional neural network. For example, the network in Figure 1 has two convolutional layers and two subsampling layers to perform the feature extraction for the input data.

Once the feature extraction is done, the output of the subsampling layer is flattened into a single vector of values and fed into the fully-connected layer. The fully-connected layer performs the classification task to produce a label indicating the correct category of the input image. In the example in Figure 1, three fully-connected layers are used.

Among all the layers of the network, the most compute-intensive layer is the convolutional layer. In this work, the convolutional layer will be implemented on the programmable logic of the FPGA so as to accelerate the inference performance of the whole convolutional neural network. The description of the 2D convolution operation is presented in the next subsection.

B. The 2D convolution operation

The convolution operation is, by far, known as the most commonly used and high compute-intensive operation in both image processing [12], [13] and artificial intelligence applications like convolutional neural networks [6], [11]. Given an M×N input image I and an S×S kernel W, the 2D convolution output image F of size M×N is computed by Equation (1), as follows:

$$F(m,n) = \sum_{i=0}^{S-1} \sum_{j=0}^{S-1} W[i,j] \cdot I[m-i, n-j] \qquad (1)$$

Figure 2 shows an illustrated view of the 2D convolution computation, in which the image size is 5x5 pixels and the kernel size is 3x3 elements. To compute the convolution for each pixel, a sliding window of size SxS is utilized for extracting the right neighboring pixels necessary for the convolution computation of the computed pixel at hand. In general, a 2D convolution with an S×S kernel requires S×S multiply-accumulate (MAC) operations for each sample; thereby, the number of MAC operations is M×N×S×S for the whole image.

C. The PYNQ-Z2 FPGA

The PYNQ-Z2 board is a Xilinx ZYNQ SoC device based on a dual-core ARM Cortex-A9 processor integrated with a FPGA fabric [14]. The functional block diagram of the Xilinx ZYNQ SoC is shown in Figure 3. The dual-core ARM Cortex-A9 processor is referred to as the Processing System (PS), and the FPGA fabric is referred to as Programmable Logic (PL). The PS subsystem includes a number of dedicated peripherals (including memory controllers and other peripheral interfaces) and can be extended with additional customized hardware IP cores in the PL overlay.

Overlays, or hardware libraries, are programmable FPGA designs that extend the user applications from the PS subsytem of the ZYNQ device into the PL subsystem. Overlays can be used to accelerate a software application, or to customize the hardware platform for a particular application. In addition, the most advantage feature of the PYNQ-Z2 board is that it provides a Python interface to allow overlays in the PL to be controlled from Python programs running in the PS, making FPGAs easier to use

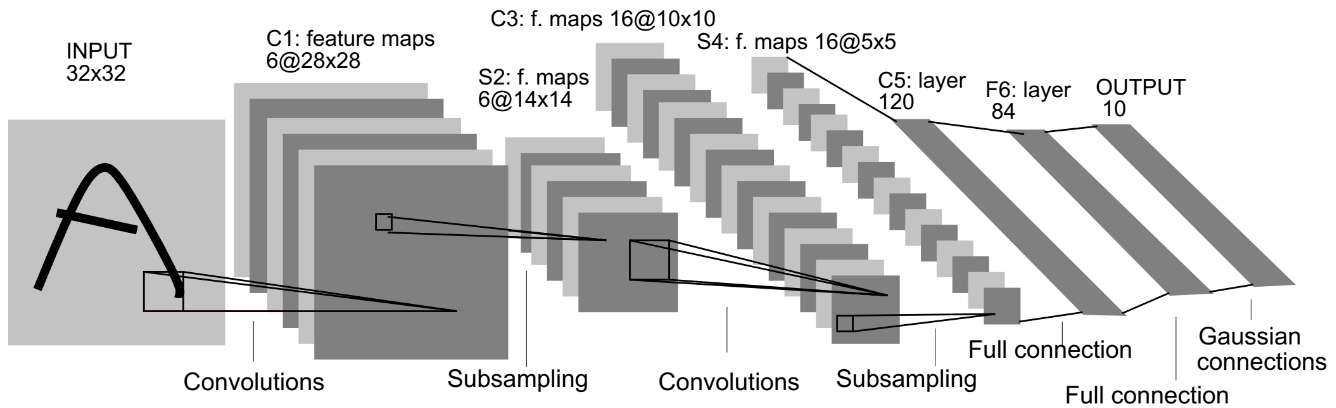

Figure 1. The Lenet convolutional neural network architecture

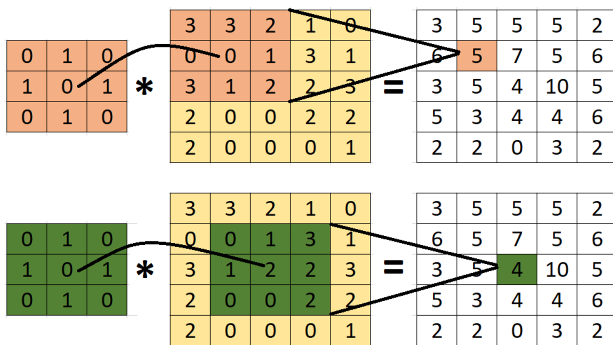

Figure 2. An illustration of the 2D convolution operation

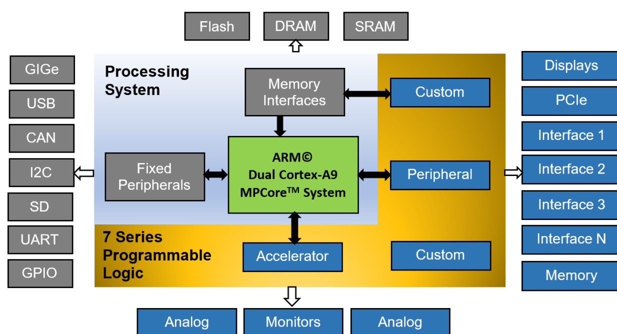

Figure 3. Functional block diagram of Xilinx ZYNQ SoC

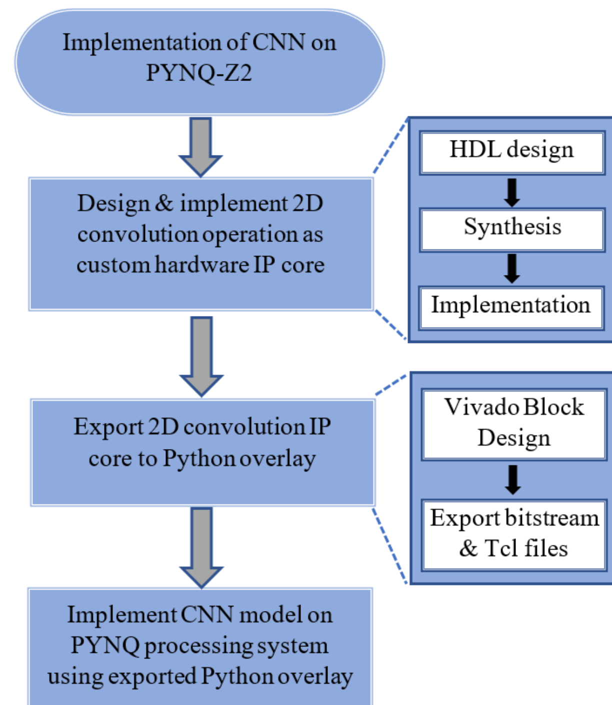

Figure 4. The design flow for CNN on PYNQ-Z2

with most of the computer vision and machine learning applications. Figure 4 presents the design flow which is applied in this work for the implementation of CNN models on the PYNQ-Z2 board.

D. Data representation for FPGA implementation

One challenge for efficient hardware implementations of image processing and machine learning applications on FPGA is to choose the suitable data format for real number operands and operations. Both fixed-point and floating-point number formats, as well as the logarithmic number system, can be used for the hardware realization on FPGA, as reported in [15], [16], [17], [18], [19], [20]. The logarithmic number system, in which a real number is represented as a fixed-point logarithm, was developed [15] and applied for adaptive signal processing algorithms [16]. Floating-point number format gives more accurate computed results [17], [18], [19], while fixed-point number format brings much better computation performance with respect to execution time and hardware resources.

Recently, it is shown that the low-precision fixed-point number format is sufficient for the training and computation of deep learning neural network models [20], with little to

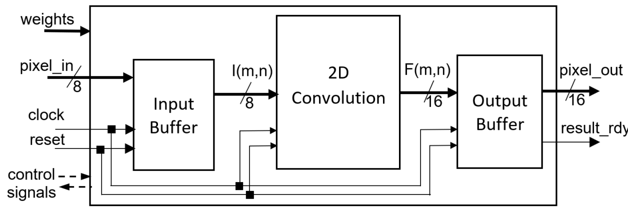

Figure 5. Functional block diagram of the 2D convolution IP core

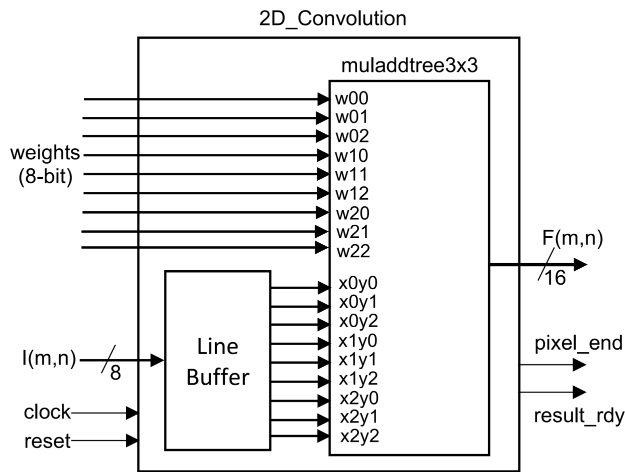

Figure 6. Block diagram of the 2D convolution module

no degradation in the classification accuracy. To perform the arithmetic computations in the designed 2D convolution core, we employ the VHDL fixed-point package designed by David Bishop [21] and use the fixed-point signed number format Q1.7 with 8-bit operands in this work.

3. Design and Implementation of 2D Convolution Core on FPGA

In this section, we present the design and implementation of 2D convolution IP core targeting for the Xilinx FPGAs. For the design and hardware implementation of the targeted IP core, we utilize the Vivado Design Suite 2018.3 WebPack Edition from Xilinx and we implement the design on the PYNQ-Z2 FPGA board.

In this work, we will investigate three different hardware implementations of the 2D convolution core that correspond to the image sizes of 32x32, 64x64 and 128x128 pixels. The kernel size is fixed at 3x3 elements. We then generate three different Python overlays corresponding to the three chosen hardware implementations of the designed IP core.

Figure 5 briefly presents the general block diagram of the designed IP core. The IP core consists of three modules: an input buffer module, a 2D convolution module and an output buffer module. The three modules are fully pipelined to increase the execution performance and data throughput.

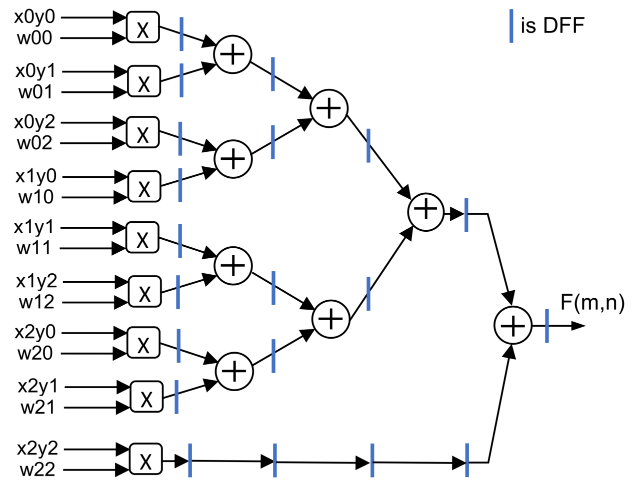

Figure 7. Block diagram of the muladdtree3x3 submodule

A. Input buffer and Output buffer modules

The input- and output-buffer modules are designed to support the communication between the IP core and the host processing system. The input buffer module is used to store all the incoming image pixels before sending them to the convolution module. Similarly, the output buffer module will store all the computed results and notify the host processing system about the readiness of the convolution computation via the `result_rdy` signal.

B. Convolution module

The 2D convolution module – the heart of the IP core – includes two submodules: a line buffer and a muladdtree3x3 submodule, as shown in Figure 6.

The line buffer reads the input image line I(m, n) to extract the nine neighboring image pixels (including the convolved pixel at hand) necessary for the convolutional computation of each input image pixel. The nine output pixels of the line buffer are denoted as $x_m y_n$, where $m, n = 0, 1, 2$. After some overhead clock cycles, the line buffer continuously provides the nine image pixels for the convolutional computation at every clock cycle.

The muladdtree3x3 submodule performs the convolution between the kernel and the nine neighboring image pixels; this computation is a dot-product between the weight vector and the output vector of the line buffer. For implementing this dot-product, we use a mul-add tree architecture to increase the computing speed. Figure 7 gives details about the architecture of the muladdtree3x3 submodule. The mul-add tree architecture is fully pipelined using a series of registers (DFF), thereby providing the convolved result for each input pixel at every clock cycle.

C. Synthesis result of the 2D convolution core

The IP core architecture is implemented in VHDL. Table I presents the synthesis results of the 2D convolutional IP core for three different implementations corresponding

TABLE I. Synthesis result of 2D convolutional IP core

Resource	Available	Resource Utilization		
		32x32	64x64	128x128
LUT	53200	1463	3517	11766
LUTRAM	17400	570	2218	8786
FF	106400	486	559	795

LUT: Look-Up Table; FF: Flip-Flop

to three input image sizes of 32x32 pixels, 64x64 pixels and 128x128 pixels. As shown in Table I, all the three implementations are successfully fitted on the chosen Xilinx PYNQ-Z2 FPGA board; the hardware resources increase with the size of the input image.

D. Packaging convolution IP core as a Python overlay on the PYNQ-Z2 board

Once the 2D convolution core has successfully been synthesized and verified, we export the design into an user IP core using the Xilinx Vivado software tool [22] (the free WebPack edition). For simplifying the software control, we employ an Advanced eXtensible Interface (AXI) Lite to carry out the data communication between the IP core and the ZYNQ-7 host processing system.

Figure 8 shows the block design view of the whole system, in which the 2D convolution core (`conv2D_0`) is connected with ZYNQ-7 processing system via the AXI interconnect and is under the common reset control of the processor system reset block.

We then run the bit stream generation and export the system to a Python overlay that can be loaded and executed on the PYNQ-Z2 development board. The exported overlay consists of two main parts: the bitstream file (.bit) that contains the hardware design and the project block diagram Tcl file (.tcl). The Tcl is used by PYNQ to automatically identify the ZYNQ system configuration, IP including versions, interrupts, resets, and other control signals [14]. As we investigate three hardware implementations of the convolution IP core, we then generate three different Python overlays corresponding to the three hardware implementations of IP cores with input image sizes of 32x32 pixels, 64x64 pixels and 128x128 pixels. The kernel size for all three implementations is fixed at the size of 3-by-3.

E. Evaluation of the 2D convolution IP core

In this subsection, we present the performance evaluation of the designed IP core. We evaluate both the theoretical peak performance and the practical sustained performance of the designed IP core. The peak performance can be determined via simulations with the assumption that the data transfers between the IP core and external memory will cause no delay. On the other hand, the sustained performance provides a more realistic figure of merit for the whole system since it takes into account the data transfers between the IP core and memory.

TABLE II. Peak performance of 2D convolution IP core

Specification	Implementation		
	32x32	64x64	128x128
Number of pixels	1024	4096	16384
Execution cycles	1034	4106	16394
Overhead cycles	10	10	10
Execution time [μs]	10.34	41.06	163.94
Frames per second	96759	24358	6100

Note: Measurements are carried out at a clock frequency of 100MHz.

TABLE III. Performance evaluation of 2D convolution IP core

Measurement	Implementation		
	32x32	64x64	128x128
HW execution time (s)	0.033	0.124	0.487
SW execution time (s)	0.260	1.061	4.364
Speed-up (times)	**7.8**	**8.6**	**9.0**

Table II reports the peak performance of the designed IP core. Since all the computing modules are fully pipelined, the IP core is expected to provide the computed result at every clock cycle. The execution cycles for the three implementations are 1034, 4106 and 16394 clock cycles, respectively, with the same overhead latency of 10 clock cycles each. We configure a working clock frequency of 100MHz for the IP core. The corresponding execution times measured in μs are then reported. The maximal frame rates at a clock frequency of 100MHz of the three implementations are 96759, 24358 and 6100 frames per second for the image sizes of 32x32, 64x64 and 128x128, respectively.

Table III presents the performance comparison between the hardware implementations of the 2D convolutional IP core and their pure software implementations in Python running on the same PYNQ-Z2 board. Figure 9 illustrates the performance speedups of the hardware implementations over the software ones. The sustained performances of the hardware implementations are worse than their corresponding peak performances; the performance degradation is due to the data transfer between the IP core and the external memory. However, the hardware implementations outperform their software counterparts by the factors of 7.8, 8.6 and 9.0 times, respectively.

4. Convolutional Neural Network Application for Handwritten Digit Recognition on PYNQ-Z2

We make use of the designed convolution IP core in a practical application, that is: the handwritten digit recognition with the MNIST dataset [11], [23]. In this application, we train a convolutional neural network to carry out the classification problem on the SoC PYNQ-Z2 device. A hardware-software co-design approach is exploited in this work. Specifically, the forward inference of the trained convolutional neural network model is executed on the

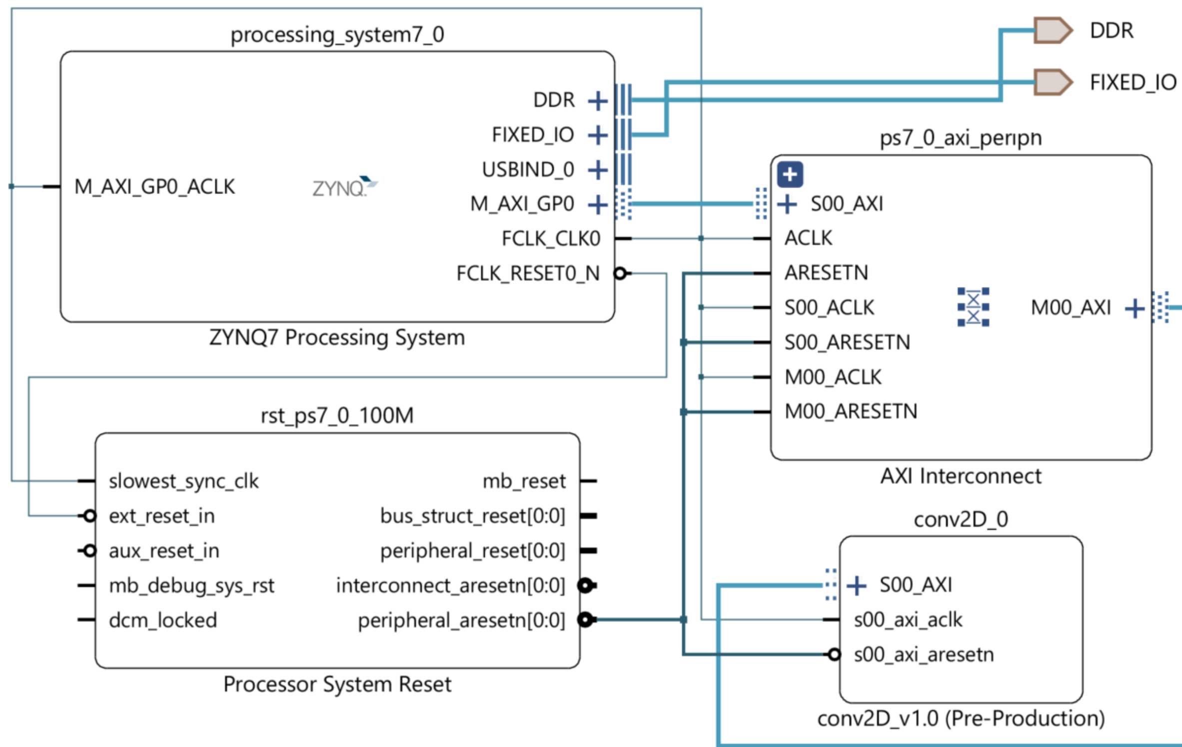

Figure 8. Block design view of the ZYNQ7 system with 2D convolution IP core via AXI

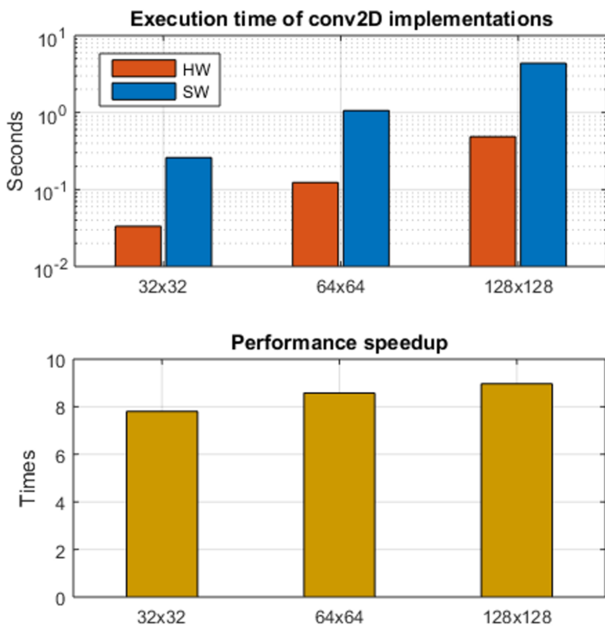

Figure 9. Performance comparison among various implementations of the 2D convolution IP core

chosen FPGA device as follows:

- The convolutional operation will be executed on the Programmable Logic of the PYNQ-Z2 device; the designed 2D convolutional IP core is loaded as a Python overlay and is executed as a Python function.
- The other operations of the network (i.e., ReLu activation function, max-pooling, flattening, fully-connected layers) will be executed on the Processing System of the PYNQ-Z2 device based on the ARM Cortex-A9 processor.

The MNIST dataset is used for training and testing the network. The dataset has a training set of 60.000 samples and a test set of 10.000 samples. There are 10 different handwritten digits ranging from 0 to 9 in the dataset. Each digit is normalized and centered in a gray-level image with size 28x28. For the sake of convenience, the samples are extended to 32x32 with background pixels.

A. *CNN configuration and model training*

The chosen architecture of the CNN for handwritten digit recognition is shown in Table IV. There are eight layers in the network. The first layer is a convolutional layer having 16 kernel maps, followed by a max-pooling layer for dimensional reduction. The third and fourth layers are another pair of convolutional and max-pooling layers having 36 kernel maps. All convolutional layers use the

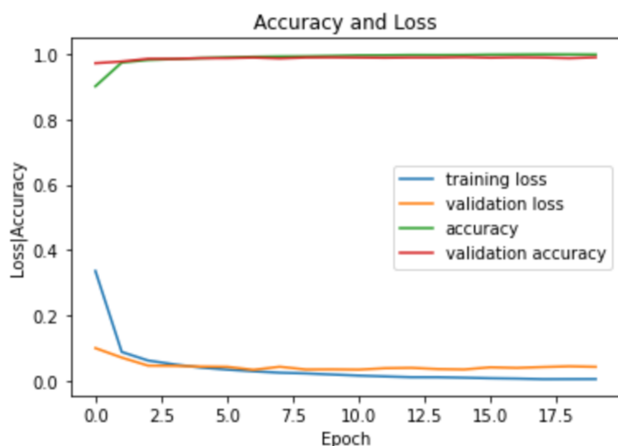

Figure 10. Convolutional neural network model training after 20 epochs, with a test accuracy of 99.04%

kernel size of 3x3 and the ReLu activation function, while all max-pooling layers use the window size of 2x2.

After the convolution and max-pooling operations, all the extracted features are flattened into a vector by the flattening layer with a resulted feature vector of 1764 elements. These feature vector then become the inputs of three consecutive fully-connected layers having 120, 84 and 10 neurons, respectively for performing the classification task. The final fully-connected layer has the softmax activation function while the other fully-connected layers use the ReLu activation functions.

For training the convolutional neural network model, we exploit the Google Colab [24]. The chosen convolutional neural network is described with Python and the model training is executed in Colab framework. Figure 10 illustrates the result of the training process. After 20 epochs, the convolutional neural network gives the accuracies of 99.77% for training set and 99.04% for test set. These accuracies are comparable with the accuracies of other related models reported in [25] for handwritten digit recognition applications. In [19], a deep neural network for handwritten digit recognition with MNIST dataset was used, resulting in an accuracy of 97.14% for the test set. Compared with previous work in [19], this work offers a much higher accuracy.

Once the training is done, all the parameters of the trained network are saved to use in the execution of the forward inference of the network on PYNQ-Z2 device.

B. Performance evaluation on PYNQ-Z2

The performance evaluation of the handwritten digit recognition application is carried out on the PYNQ-Z2 device. Two implementations are performed on the device: *i)* a pure software implementation that runs on the Processing System of the device fully based on the ARM Cortex-A9 processor; and *ii)* a hardware-software co-design implementation with the 2D convolution IP core executed on the Programmable Logic of the FPGA device while others operations executed by the ARM Cortex-A9 processor. For a better understanding of the performance of the entire network, the execution time of each layer is also measured.

Table V reports the total execution time of two implementations of the network. Thanks to the use of the convolution IP core on FPGA fabric, the hardware-software co-design implementation outperforms the pure software implementation by a factor of 6.2 times.

Table V also reports the execution times of all layers, which allow for an efficient performance profiling. The convolution layers account for most of the computational loads of the network: 98% and 95% of the total execution times of the two implementations, respectively. Obviously, the performance of the whole network can be further improved when the execution of the convolutional layers are speeduped.

5. Conclusions

In this paper, we have presented the design, implementation and evaluation of a 2D convolutional IP core synthesizable for FPGAs. We have developed and generated hardware implementations of the IP core as Python overlays, and carried out the performance evaluation on the PYNQ-Z2 device. It has been shown that the hardware implementations of the IP core outperform their software implementations. Furthermore, we make use of the designed convolutional IP core as hardware accelerator in the handwritten digit recognition application, in which a hardware-software co-design framework is deployed. Thanks to the use of hardware accelerator for the convolutional layers, the execution performance of the convolutional neural network has been improved by a factor of 6.2 times. We believe that the framework presented in this work will help to accelerate the FPGA-based hardware implementations of the image processing and deep learning applications.

To further increase the performance of the convolutional neural network implementations, we will improve the communication interface between the IP core and the ARM-based processing system by employing an AXI-Stream interface with direct memory access (DMA) control mechanism. This will be our future work.

Acknowledgment

This research is funded by Funds for Science and Technology Development of the University of Danang under project number B2019-DN02-61.

References

[1] Y. Wei, W. Xia, M. Lin, J. Huang, B. Ni, J. Dong, Y. Zhao, and S. Yan, "Hcp: A flexible cnn framework for multi-label image classification," *IEEE transactions on pattern analysis and machine intelligence*, vol. 38, no. 9, pp. 1901–1907, 2015.

[2] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, pp. 1097–1105, 2012.

TABLE IV. Convolution neural network architecture for handwritten digit recognition with MNIST dataset

No	Layer	Number of feature maps	Size	Kernel size	Stride	Activation function
0	Input Image	1	32x32	-	-	-
1	Convolution	16	30x30	3x 3	1	ReLu
2	MaxPooling	16	15x15	2x2	2	-
3	Convolution	36	13x13	3x3	1	ReLu
4	MaxPooling	36	7x7	2x2	2	-
5	Flattening	-	1764	-	-	-
6	Fully-Connected	-	120	-	-	ReLu
7	Fully-Connected	-	84	-	-	ReLu
8	Fully-Connected	-	10	-	-	Softmax

TABLE V. Execution time [second] of convolution neural network implementations with MNIST dataset on PYNQ-Z2 FPGA

Layer	Pure software implementation		Implementation with 2D Convolution IP core on hardware	
	(s)	(%)	(s)	(%)
1st Convolution	**4.017**	**69.30**	**0.634**	**68.22**
1st ReLu + MaxPooling	0.005	0.08	0.002	0.24
2nd Convolution	**1.735**	**29.94**	**0.253**	**27.28**
2nd ReLu + MaxPooling	0.004	0.06	0.004	0.38
Flattening	0.001	0.02	0.001	0.11
1st Fully-Connected	0.028	0.47	0.028	2.96
2nd Fully-Connected	0.005	0.08	0.005	0.51
3rd Fully-Connected	0.003	0.05	0.003	0.31
Total execution time	**5.797**	100.00	**0.929**	100.00
Speedup	**1X**		**6.2X**	

[3] A. B. Nassif, I. Shahin, I. Attili, M. Azzeh, and K. Shaalan, "Speech recognition using deep neural networks: A systematic review," *IEEE access*, vol. 7, pp. 19 143–19 165, 2019.

[4] M. Bojarski, D. Del Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang *et al.*, "End to end learning for self-driving cars," *arXiv preprint arXiv:1604.07316*, 2016.

[5] D. Strigl, K. Kofler, and S. Podlipnig, "Performance and scalability of gpu-based convolutional neural networks," in *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*. IEEE, 2010, pp. 317–324.

[6] Y.-H. Chen, T. Krishna, J. S. Emer, and V. Sze, "Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks," *IEEE journal of solid-state circuits*, vol. 52, no. 1, pp. 127–138, 2016.

[7] 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*, 2015, pp. 161–170.

[8] M. Sit, R. Kazami, and H. Amano, "Fpga-based accelerator for losslessly quantized convolutional neural networks," in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 295–298.

[9] "PYNQ Homepage," accessed: 2020-10-18. [Online]. Available: pynq.io/home.html

[10] TUL Technology Unlimited, "TUL PYNQ-Z2 board," accessed: 2020-10-18. [Online]. Available: tul.com.tw/ProductsPYNQ-Z2.html

[11] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[12] B. Cope *et al.*, "Implementation of 2d convolution on fpga, gpu and cpu," *Imperial College Report*, pp. 2–5, 2006.

[13] S. Perri, M. Lanuzza, P. Corsonello, and G. Cocorullo, "A high-performance fully reconfigurable fpga-based 2d convolution processor," *Microprocessors and Microsystems*, vol. 29, no. 8-9, pp. 381–391, 2005.

[14] "PYNQ Overlay Tutorials," accessed: 2020-10-18. [Online]. Available: pynq.readthedocs.io/en/v2.5.1/pynq_overlays.html

[15] J. N. Coleman, E. Chester, C. I. Softley, and J. Kadlec, "Arithmetic on the european logarithmic microprocessor," *IEEE Transactions on Computers*, vol. 49, no. 7, pp. 702–715, 2000.

[16] F. Albu, J. Kadlec, N. Coleman, and A. Fagan, "Pipelined implementations of the a priori error-feedback lsl algorithm using logarithmic arithmetic," in *2002 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3. IEEE, 2002, pp. III–2681.

[17] T. V. Huynh, "Design space exploration for a single-fpga handwritten digit recognition system," in *2014 IEEE Fifth International*

Conference on Communications and Electronics (ICCE). IEEE, 2014, pp. 291–296.

[18] T. V. Huynh, “Evaluation of artificial neural network architectures for pattern recognition on fpga,” *International Journal of Computing and Digital Systems*, vol. 6, no. 03, pp. 133–138, 2017.

[19] T. V. Huynh, “Deep neural network accelerator based on fpga,” in *2017 4th NAFOSTED Conference on Information and Computer Science*. IEEE, 2017, pp. 254–257.

[20] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, “Deep learning with limited numerical precision,” in *International conference on machine learning*. PMLR, 2015, pp. 1737–1746.

[21] Bishop, David W., “VHDL-2008 support library,” 2011, accessed: 2020-10-18. [Online]. Available: github.com/FPHDL/fphdl

[22] Xilinx, “Vivado Design Suite Evaluation and WebPACK,” accessed: 2020-10-18. [Online]. Available: xilinx.com/products/design-tools/vivado/vivado-webpack.html

[23] “MNIST database,” accessed: 2020-10-18. [Online]. Available: yann.lecun.com/exdb/mnist/

[24] Google, “Google Colaboratory (Colab) Introduction,” accessed: 2020-10-18. [Online]. Available: colab.research.google.com/notebooks/intro.ipynb

[25] A. Baldominos, Y. Saez, and P. Isasi, “A survey of handwritten character recognition with mnist and emnist,” *Applied Sciences*, vol. 9, no. 15, p. 3169, 2019.

Thang Viet Huynh Thang Viet Huynh received his PhD degree in Electrical and Electronic Engineering from Graz University of Technology (TUGraz), Austria in 2012. He is currently working as a senior lecturer at the Faculty of Electronics and Telecommunication Engineering, Danang University of Science and Technology (DUT), The University of Danang (UDN), in Danang City, Vietnam. His research interests include embedded reconfigurable computing (FPGA), hardware implementations of deep learning models, edge computing, and respective applications.