Implementation of Convolutional Neuronal Network on a Software/Hardware Co-processing Scheme to Identify Handwritten Digits

Edwin Orlando González Serrano, Walter Daniel Villamizar Luna

Trabajo de Grado para optar al título de Ingeniero Electrónico

Director

Carlos Augusto Fajardo Ariza

Ph.D en Ingeniería

Co-director

Luis Eduardo Rueda Guerrero

Msc en Electrónica

Universidad Industrial de Santander

Facultad de Ingenierías Fisicomecánicas

Escuela de Ingenierías Eléctrica, Electrónica y de Telecomunicaciones

Bucaramanga

2019

Agradecimientos

Agradecemos a nuestros padres por el apoyo incondicional que siempre nos brindaron durante el desarrollo de la carrera. A los amigos por todas las vivencias inolvidables durante estos años de universidad.

Un reconocimiento y agradecimiento a nuestro director Carlos Fajardo por motivarnos a realizar este trabajo de grado, por su disposición y entrega en la guía de nuestro proyecto.

Finalmente agradecemos a todas las personas que a lo largo de este camino contribuyeron a la formación personal y profesional en nuestras vidas.

Content

List of figures

List of tables

Resumen

Título: Implementación de una Red Neuronal Convolucional en un Esquema de Co-procesamiento Software/Hardware para la Clasificación de Digitos Manuscritos [*]

Autores: Edwin Orlando González Serrano, Walter Daniel Villamizar Luna [**]

Palabras Clave: CNN, FPGA, Hardware-Accelerator, MNIST, Zynq.

Descripción: Este trabajo hace parte de la fase inicial de un macroproyecto, el cual tiene como objetivo, el desarrollo de dispositivos portables con bajo consumo de potencia, para aplicaciones médicas utilizando redes neuronales convolucionales (CNNs). Las redes neuronales convolucionales cada vez son más populares en aplicaciones de aprendizaje profundo como por ejemplo en clasificación de imágenes, reconocimiento de voz, medicina, entre otras. Sin embargo, las CNNs son computacionalmente costosas y requieren altos recursos de memoria. Nosotros proponemos un acelerador de hardware para la red Lenet-5 (LeCun et al., 1998) en un esquema de co-procesamiento Software/Hardware. El esquema propuesto tiene como objetivo reducir los recursos de hardware y obtener un alto rendimiento en el proceso de inferencia. Hemos desarrollado una estrategia para reducir recursos de memoria y recursos computacionales. Esta estrategia nos permite reducir la cantidad de memoria requerida sobrescribiéndolas y los recursos computacionales usando un formato de datos en punto fijo. Este esquema fue diseñado alrededor de la plataforma Zynq-7000. Implementamos nuestro diseño en la tarjeta de desarrollo Digilent Arty Z7-20. Nuestros resultados evidenciaron una exactitud del 97.59 % utilizando representación en punto fijo de 12 bits. Con una frecuencia de tan solo 100MHz en el acelerador de hardware, el esquema propuesto obtiene una mejora de 17 % en el throughput cuando se compara con una implementación de software en un procesador de propósito general cuya frecuencia es de 650MHz.

---

[*] Trabajo de grado

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Carlos A. Fajardo, Ph.D en Ingeniería. Co-director: Luis E. Rueda, Msc en Electrónica..

Abstract

Title: Implementation of Convolutional Neuronal Network on a Software/Hardware Co-processing Scheme to Identify Handwritten Digits [*]

Authors: Edwin Orlando González Serrano, Walter Daniel Villamizar Luna [**]

Keywords: CNN, FPGA, Hardware-Accelerator, MNIST, Zynq.

Description: This work is part of the initial phase of a macro project, which aims, the development of portable devices with low power consumption, for medical applications using convolutional neural networks (CNNs). Convolutional neuronal networks are increasingly popular in deep learning applications such as image classification, speech recognition, medicine, among others. However, CNNs are computationally expensive and require high memory resources. We propose a hardware accelerator for the Lenet-5 (LeCun et al., 1998) network in a Software / Hardware co-processing scheme. The proposed scheme aims to reduce hardware resources and obtain a high performance in the process of inference. We have developed a strategy to reduce memory resources and computational resources. This strategy allows us to reduce the amount of memory required by overwriting data in the same memories and computational resources using a fixed-point data format. This scheme was designed around the Zynq-7000 platform. We implemented our design on the Digilent Arty Z7-20 development card. Our results achieved an accuracy of 97.59 % using 12-bit fixed-point representation. Therefore, a low utilization of resources with competitive accuracy is obtained. In addition, the proposed scheme achieves a 17 % improvement in throughput when compared to a software implementation in a general-purpose processor. The processor works at 650MHz and the FPGA is synchronized at 100MHz.

---

[*] Bachelor Thesis

[**] Facultad de Ingenierías Físico-Mecánicas. Escuela de Ingenierías Eléctrica, Electrónica y telecomunicaciones. Director: Carlos A. Fajardo, Ph.D en Ingeniería. Co-director: Luis E. Rueda, Msc en Electrónica.

Introduction

Image classification has been widely used in many areas of the industry. Convolutional neural networks (CNNs) have shown to perform high accuracy and robustness for image classification (e.g. Lenet-5 (LeCun et al., 1998), GoogLenet(Szegedy et al., 2015)). CNNs have numerous potential applications in object recognition, vision systems, medical devices, between others.

In contrast, modern models of CNNs demand high computational cost and large memory resources. This high demand is due to the multiple arithmetic operations that must be computed and the huge amount of parameters that must be saved. Thus, large computing systems and high energy dissipation are required.

Therefore, several hardware accelerators have been proposed in recent years to achieve the usage of low power and high performance. In (Shen et al., 2017) an automated design methodology was proposed to perform a different subset of CNN convolutional layers into multiple processors by partitioning available FPGA resources . The results achieved a 3.8×, 2.2× and 2.0× higher throughput than previous works in AlexNet, SqueezeNet, GoogLenet respectively.

In addition, Software/Hardware (Sw/Hw) co-design architectures have significantly allowed speedup applications. The accelerator was designed to compute intensive tasks and the software manages the control process. In (Wang et al., 2016) a CNN for a low-power embedded system was proposed. The results achieved 2× energy efficiency compared with GPU implementation. In (Gan Feng et al., 2016) methods are proposed to optimize CNN regarding energy efficient and high throughput. An FPGA-based CNN for Lenet-5 was implemented into Zynq-7000 platform. This

work achieved 37 % higher throughput and 93.7 % less energy dissipation than GPU implementation, and the same error rate of 0.99 % as software implementation .

It is important to mention that most of the works mentioned above have used a High-Level Synthesis (HLS) software. These tool allows the creation of accelerator from software directly without the need to manually create a RTL. HLS software generally causes higher hardware resource utilization.

This work aims to implement a CNN with both low hardware resource utilization and high throughput. We have designed a Sw/Hw co-processing scheme for the Lenet-5. In order to reduce hardware resources, the implementation was done using 12-bits fixed-point on the Zynq platform. The results show a not significant decrease in accuracy with low hardware resource usage.

This paper is organized as follows: Section 1 describes CNNs. Section 2 explains the scheme used. Section 3 presents details of the implementation and the results of our work. Section 4 discusses the results and some ideas for further work. Finally, Section 5 concludes this paper.

1. Convolutional Neural Network

The CNNs allow the extraction of features from input data to classify them into a set of pre-established categories. To classify data CNN must be trained. The training process aims to fit the parameters to classify the data with the desired accuracy. During the training process, many input data are presented to CNN with the respective labels. Then a gradient-based learning algorithm (LeCun et al., 1998) is executed to minimize a loss function by updating the CNN parameters (weights and biases). The loss function evaluates the inconsistency between the predicted label and the current label.

A CNN consist of a series of layers that run sequentially. The output of a specific layer is the input of the subsequent layer. The CNN typically uses three types of layers: Convolutional, sub-sampling and fully connected layers. Convolutional layers extract features from the input image. Sub-sampling layers reduce both spatial size and computational complexity. Finally, the Fully connected layers classify the data.

1.1. Lenet-5 Architecture

The Lenet-5 architecture (LeCun et al., 1998) includes seven layers (Figure 1): three Convolutional layers (C1, C3, and C5), two sub-sampling layers (S2 and S4) and, two fully connected layers (F6 and OUTPUT).

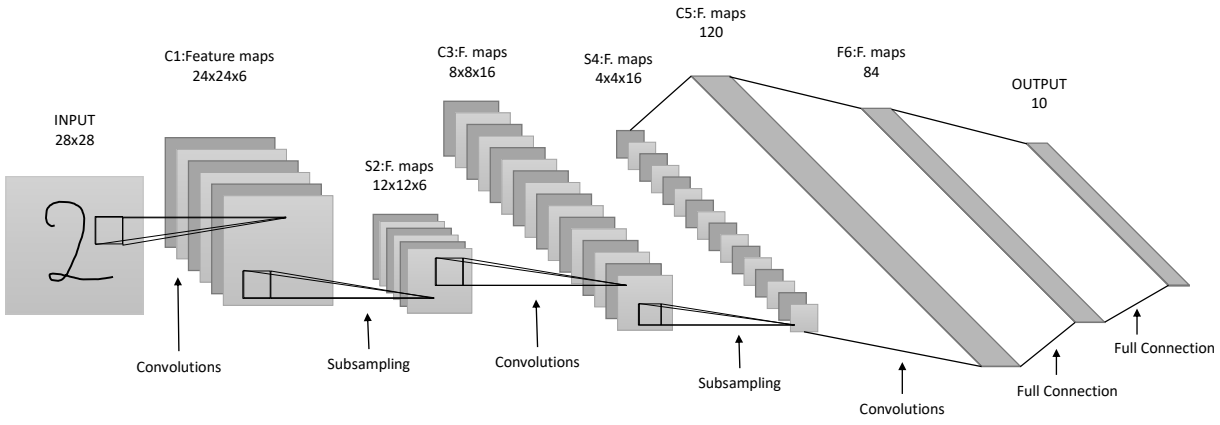


Figura 1. Lenet-5 Architecture. Convolutional neural network. Adapted from (LeCun et al., 1998)

In our implementation, the size of the input image is 28×28 pixels. Table 1 shows the dimensions of input, feature maps, weights, biases for each layer.

The convolutional layers consist of kernels (matrix of weights), biases and an activation function (Eg. ReLu, Sigmoid). The convolutional layer takes the feature maps in the previous layer with p depth and convolves them with $k \times k$ kernels with the same depth. Then, the bias is added

Table 1
Dimension of CNN layers

LAYER	Size			
	Input $n \times n \times p$	Weights $k \times k \times p \times d$	Biases [d]	Feature Maps $[m \times m \times d]$
C1	$28 \times 28 \times 1$	$5 \times 5 \times 1 \times 6$	6	$24 \times 24 \times 6$
S2	$24 \times 24 \times 6$	N.A	N.A	$12 \times 12 \times 6$
C3	$12 \times 12 \times 6$	$5 \times 5 \times 6 \times 16$	16	$8 \times 8 \times 16$
S4	$8 \times 8 \times 16$	N.A	N.A	$4 \times 4 \times 16$
C5	$4 \times 4 \times 16$	$4 \times 4 \times 16 \times 120$	120	$1 \times 1 \times 120$
F6	$1 \times 1 \times 120$	$1 \times 1 \times 120 \times 84$	84	$1 \times 1 \times 84$
Output	$1 \times 1 \times 84$	$1 \times 1 \times 84 \times 10$	10	$1 \times 1 \times 10$

to convolution output and this result passes through an activation function, in this case, ReLu. The kernels shift into the input with a stride of one to obtain an output feature map. The convolutional layers have d feature maps of $n-k+1 \times n-k+1$ pixels.

Note that weights and biases are not found in S2 and S4 layers. The output of a sub-sampling layer is obtained by taking the maximum value from a batch of the feature map in the preceding layer. A fully connected layer has d feature maps. Each feature map result of the dot product between the input vector of the layer and the weights vector. The input and weights vector have p elements.

2. Software/Hardware co-processing scheme

Figure 2 presents the proposed SW/HW co-processing scheme to perform the Lenet-5 on Zynq-7000 SoC. It consists of two main parts: an ARM processor and an FPGA, which are connected by AMBA AXI4 bus (Xilinx, 2017). Into the ARM processor runs a C application, which is responsible for the data transfer between software and hardware. On the other hand, a hardware

accelerator is implemented into the FPGA. This accelerator consists of a custom computational architecture that performs the CNN inference process.

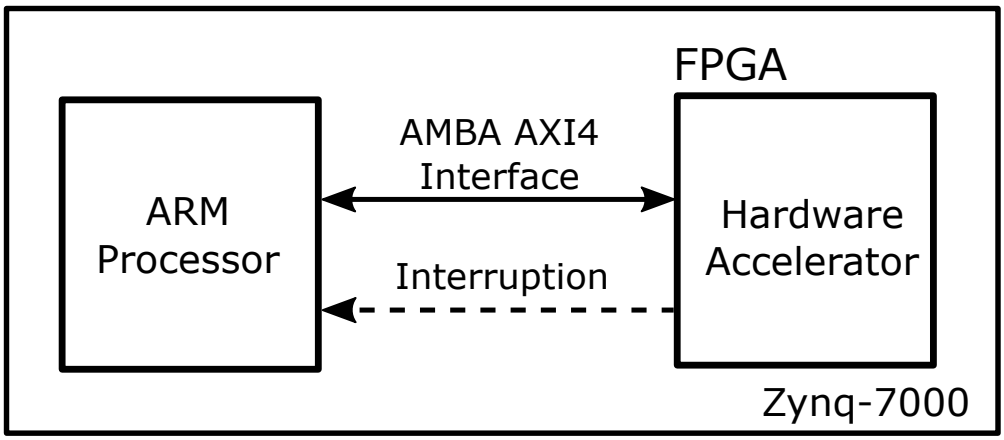


Figura 2. General Software/Hardware co-processing scheme

2.1. Hardware Accelerator

Figure 3 shows the hardware accelerator that develops the inference process of the CNN. The FSM controls the hardware resources into the accelerator. The *REGISTER_BANK* contains the dimensions of the entire architecture (Table 1). The *MEMORY_SYSTEM* consist of four Blocks RAM (BRAMs). The first one stores the *weights*, and the second one stores the *biases*. The *RAM_3* and *RAM_4* store the intermediate values of the process. These memories are overwritten in each layer. All four memories are addressed by the *ADDRESS_RAM* module.

The *MATH_ENGINE* is the mainstream module in our design because it performs all the necessary arithmetic operations in the three types of layers mentioned in Section 1. All feature maps of the layers are calculated by reusing this module. It is necessary to highlight the saving of the hardware resources with the implementation of this module.

This module allowed us:

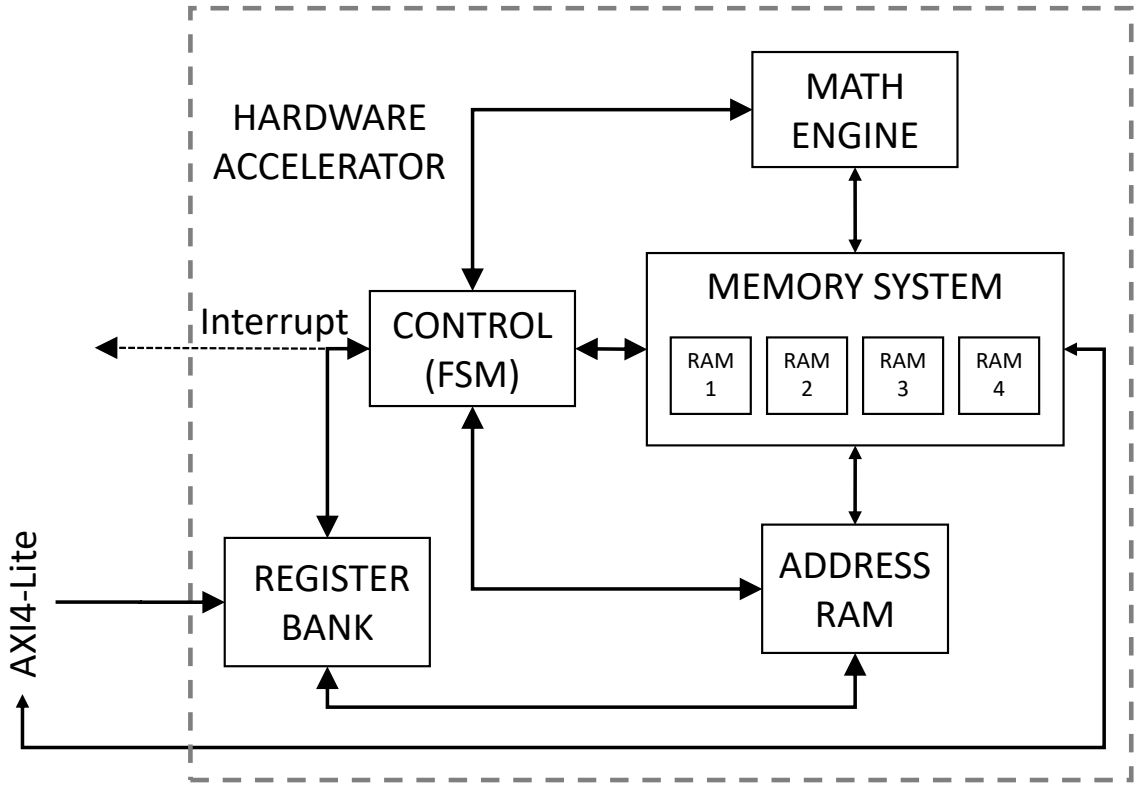


Figura 3. Hardware accelerator scheme

- To perform a convolution process in a parallel fashion.
- To calculate a dot product between vectors
- To add the bias
- To evaluate the ReLu activation function
- To perform the sub-sampling process

To perform all the processes mentioned above, the module *MATH_ENGINE* is composed of: *CONV_DOT*, *BIAS& ReLu* and *SUB-SAMPLING* sub-modules. The overall architecture of the

MATH_ENGINE is shown in Figure 4.

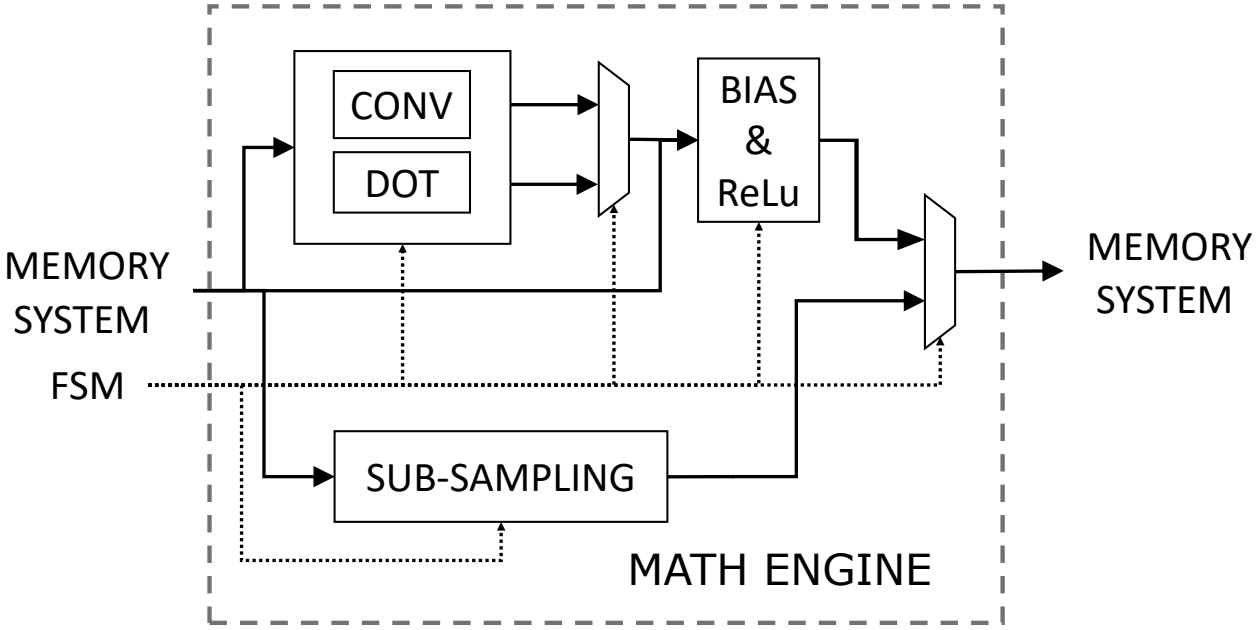


Figura 4. Math Engine

CONV_DOT consists of six blocks. Each block perform both a convolution 5×5 or a dot product between vectors width of 25. The sum of the convolution result represents the output of the *CONV_DOT* module C3, C5, F6, and OUTPUT. In C1, the result of each convolutional block is the output of *CONV_DOT*. *BIAS&ReLu* adds bias and evaluate the ReLu activation function. *SUB-SAMPLING* aims to perform the layers S2 and S4.

2.2. Description of the Sw/Hw Scheme Process

Before the accelerator classifies an image, it is necessary to store all the required parameters (weights, biases, and dimensions of CNN layers). These parameters and the input image are sent by the processor to the *MEMORY_SYSTEM* and the *BANK_REGISTER* (Initially *RAM3* stores the image). Then, the processor sets the start signal of the *FSM*, initiating the CNN inference pro-

cess. According to the data into the *BANK_REGISTER* the *FSM* configures the *MATH_ENGINE*, *ADDRESS_RAM* and *MEMORY_SYSTEM* modules to perform the pre-established layer.

For convolutional and fully connected layers, the *CONV_DOT* sub-module performs the convolution or the dot product operations respectively, while in the *Bias&ReLu* sub-module the sum of the bias and the result of the *CONV_DOT* pass through the activation function. The result of *Bias&ReLu* is stored in *RAM3* or *RAM4*. The feature maps of F6 are overwritten in *RAM3* and features maps of C3, C5 and OUTPUT are overwritten in *RAM4*. In the case of the sub-sampling layers, they are carried out by the sub-module *SUB_SAMPLING*, and the result is stored in RAM3.

The CNN inference process is complete by executing all the layers. Then the *FSM* sends an interruption to the ARM processor. The interrupt indicates to the ARM processor to extract the ten outputs from the hardware accelerator. The outputs represent digits from 0 to 9, e.g., the output 5 represents digit 4.

Finally, the ARM processor reads the first ten positions of the *RAM4*. The classification of the image is the digit represented by the output that has the maximum value.

3. Results

This section describes the analysis of the CNN performance implemented on the SW/HW co-processing scheme. During the implementation two's complement fixed-point representation was used. The implementation was carried out into a Digilent Arty Z7-20 development board. This board is design around the Zynq-7000 System on Chip (SoC) with 512MB DDR3 memory. The Soc integrates a dual-core ARM Cortex-A9 processor with Xilinx Artix-7 FPGA. The ARM processor runs at 650MHz and the FPGA is clocked at 100MHz. The project was synthesized using

Xilinx Vivado Design Suite 2018.4 software.

3.1. CNN training and validation

CNN was trained and validated on a set of handwritten digits images from the MNIST database (LeCun et al., 1998). The MNIST database contains handwritten digits from 0 to 9. The digits are centered on a 28x28 pixel grayscale image. Each pixel is represented by 8-bits, obtaining values in a range from 0 to 255. Where 0 means white background, and 255 means black foreground. The MNIST database has a training set of 60.000 images and a validation set of 10.000 images. API Keras with tensorflow backend was used to train and validate the CNN in python. Python was set up to use a floating-point representation to achieve high precision.

The validation process was iterated eight times on our scheme by changing the word length and the fractional length. CNN parameters and the validation set were quantized in two's complement fixed-point representation using MATLAB. Table 2 shows the percent of accuracy obtained in each validation.

Table 2
Accuracy for different data representation

	Word Length [# of bits]	Accuracy %
Floating Point	32	98.85 %
Fixed-Point	≪17,10≫	98.85 %
	≪15,9≫	98.71 %
	≪16,8≫	98.70 %
	≪15,8≫	98.70 %
	≪14,8≫	98.68 %
	≪12,6≫	97.59 %
	≪11,5≫	91.46 %
	≪16,11≫	61.31 %

The $\ll$ W, F $\gg$ notation indicates the word length (W) and the fractional length (F). With the help of MATLAB, the maximum of the absolute value of the data was found in the CNN inference process. With an integer length of six bits, it is possible to represent the maximum value found, but we decided to start with a word length of seventeen and ten in the fraction length.

3.2. Execution Time

The number of clock cycles elapsed in data transfer between software and hardware is measured by a global timer. A counter was implemented on the FPGA to measures the number of the clock cycles that the hardware accelerator takes to perform the inference process. The execution time of the implementation is shown in table 3.

Table 3
Execution times per image on our implementation.

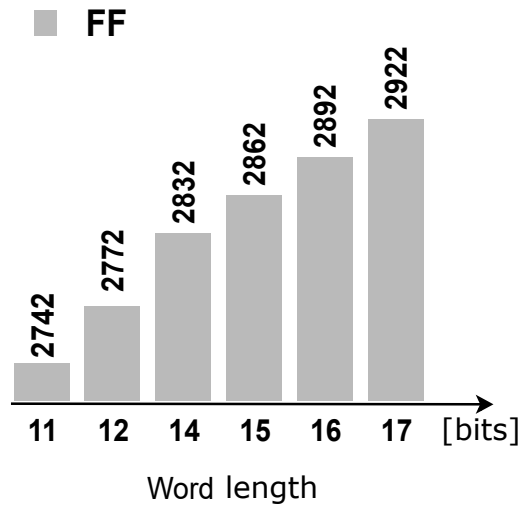
Process	Time [ms]
Load CNN parameters	7.559
Load Image Input	0.122
Inference process	2.143
Extract output data	0.002
Total Time	**2.268**

Load process of the CNN parameters is only done at once to configure the hardware accelerator. Therefore, it was not considered in the total execution time.

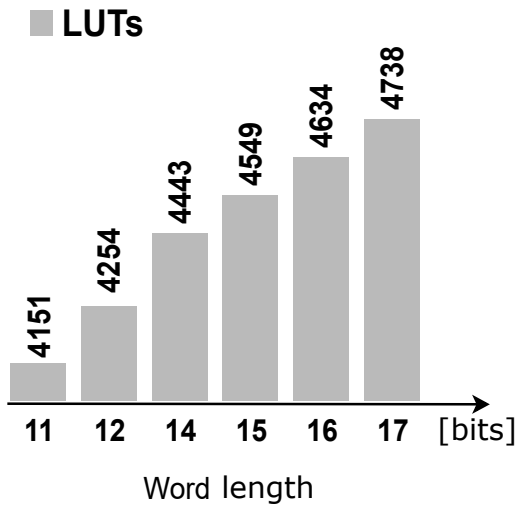
The execution time per image is 2.268 ms. Thus, it can process 440.917 images per second.

3.3. Hardware Resources Utilization

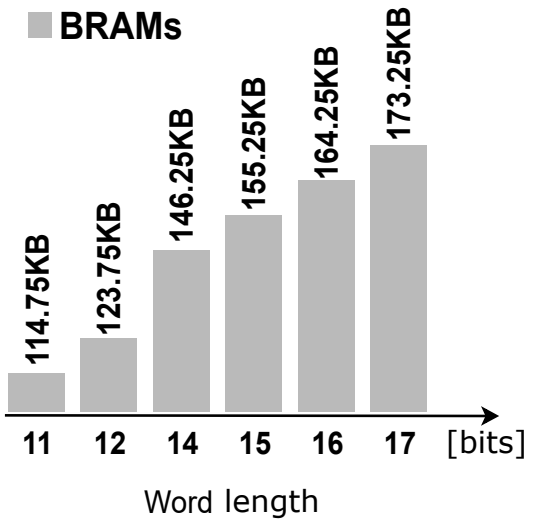
Figure 5 shows hardware resources utilization used by the hardware accelerator for different word length. The amount of DSPs used in all cases is 153. Thus, in this case the increasing of



(a)



(b)



(c)

Figura 5. Comparison of hardware resources utilization for different word length : (a) BRAMs usage; (b)LUTs usage; (c)FF usage

the word length does not increase the number of DSP used. In addition, a shorter word length significantly saves hardware resources utilization. However, it is important to consider how the accuracy will be affected (see Table 2).

3.4. Power Estimation

Power estimation was performed using the Xilinx Vivado software with the default parameters. Figure 6 shows the power estimation from the ARM processor and FPGA for two implementations. The first one is the proposed scheme and the second is a software implementation.

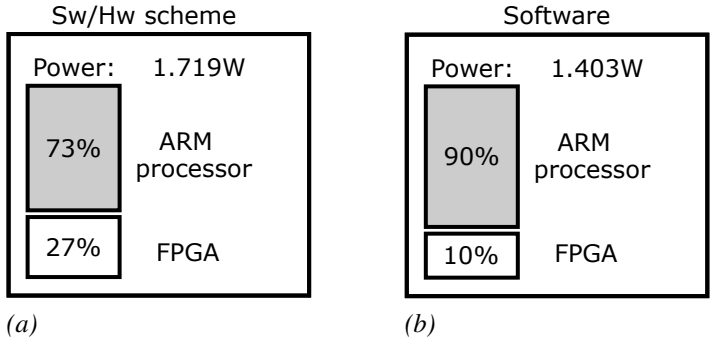


Figura 6. Power estimation of different implementations : (a) The proposed scheme using 12-bits fixed-point representation; (b)A software implementation on zynq-7000 platform.

The FPGA consumes power in both implementations, even it is nothing implemented on it due to the architecture of the Zynq-7000 platform. However, the ARM processor consumes most of the total power.

4. Discussion

We compared the implementation of our co-processing scheme with a software implementation on the Zynq-7000 platform. The results of the implementation of our scheme showed that: (i) The accuracy was 97.59 % using a 12-bit fixed-point representation. (ii) The throughput achieved was up to 441 images per second. (iii) The estimated power was $1.719W$. The values obtained from the software implementation were: (i) The accuracy of 98.85 % using a 32-bits floating-point

representation. (ii) a throughput of 365 images per second. (iii) The estimated power was $1.403W$. Although the software implementation uses more than twice the word length of our schema, surprisingly the accuracy only fell 1.27%. The implementation in software consumes less power. However, our design can classify more than 75 images per second than the implementation in software. Future work will focus on a low power version by removing the ARM processor. This will require the implementation of additional module to transfer the parameters and the input images to the hardware accelerator.

5. Conclusion

In this paper, a Sw/Hw co-processing scheme for Lenet-5 was proposed. Furthermore, our implementation on Digilent Arty Z7-20 development board achieved 97.59% of accuracy for the MNIST database by using 12-bits fixed-point format. The implementation of the proposed scheme achieved a 17% higher throughput than a software implementation on the Zynq-7000 platform.

Our results suggest that the usage of fixed-point data format allows the reduction of hardware resources without compromising the accuracy. Furthermore, the co-processing scheme makes it possible to improve the inference processing time. This encourages future advancements on energy-efficient on embedded devices for deep learning applications.

Bibliography

Gan Feng, Zuyi Hu, Song Chen, and Feng Wu (2016). Energy-efficient and high-throughput fpga-based accelerator for convolutional neural networks. In *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pages 624–626.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P., et al. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Shen, Y., Ferdman, M., and Milder, P. (2017). Maximizing cnn accelerator efficiency through resource partitioning. In *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, pages 535–547.

Szegedy, C., Wei Liu, Yangqing Jia, Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1–9.

Wang, Y., Xia, L., Tang, T., Li, B., Yao, S., Cheng, M., and Yang, H. (2016). Low power convolutional neural networks on a chip. In *2016 IEEE International Symposium on Circuits and Systems (ISCAS)*, pages 129–132.

Xilinx (2017). Axi reference guide [pdf online]. Retrieved from : https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf.