INTRODUCTION TO VLSI
EITF35

LAB REPORT FOR EXTRA LABS 4 and 5 – Implementation of Convolutional Neural Networks on FPGA

By

ILAYDA YAMAN

2018-11-12

Department of Electrical and Information Technology
Faculty of Engineering, LTH, Lund University
SE-221 00 Lund, Sweden

LUND UNIVERSITY

2018

Abstract

The purpose of this report is to explain the design that have been done in the extra labs of class “Introduction to VLSI”. The aim of the design is to detect images as circles or rectangles in real time with low power and high speed. As a result, the design is implemented on a FPGA board and a Convolutional Neural Network (CNN) has been used. The weights were given beforehand and the architecture of the CNN was fixed. First layer includes a convolutional layer and an activation layer (RELU) which is the main focus of lab 4. Other layers, a pooling layer, two FC layers, one sigmoid and another RELU layer, are implemented for lab 5.

Contents

1. Design specifications

The design should be able to classify 64x64 sized images into having either circle or rectangle, which means the valid output of the system will be either a one or zero. The board, Nexys 4 Artix-7 FPGA Trainer Board, is provided in the lab. For compatibility with predefined clock of the board, it is desired that the design can operate on 100 MHz clock frequency. Power should not be more than 1W in order to achieve low power. The area is limited by the resources that FPGA can provide.

2. Introduction

The main aim of the design is to detect if a 64x64 image is a circle or rectangle. As it can be seen from the Fig. 1. if the output of the CNN is a 0, the image is interpreted as a circle and if the output is 1, it is a rectangle.

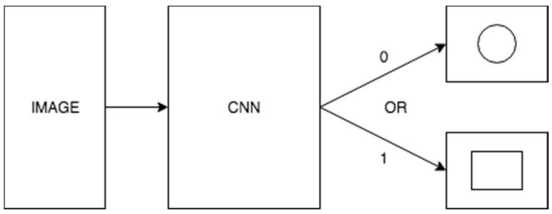

Figure 1. showing the image classification by Convolutional Neural Network

The motives to build such a design is to be able to do image detection with low power, high efficiency and in real time which can be used in many purposes such as autonomous vehicles, drones, medical equipment etc. To achieve this, the design was built as hardware efficient as possible.

3. Convolutional Neural Networks

For image detection, the most common deep learning architecture is to build a Convolutional Neural Network (CNN). The main operation of this architecture is the convolution layer as the name suggests.

In my design, the 64x64 image is convolved with a 4x4 matrix which includes the precalculated weights to obtain a 61x61 matrix. Since the images are only black and white only one dimensional convolution is needed. The basic representation of convolution in one dimension can be seen in Fig. 1.

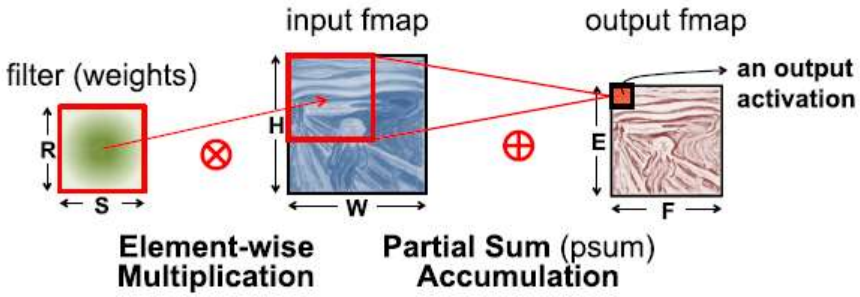

Fig. 1. Showing the Basic Concept of Convolution [1]

Convolution layer is followed by an activation layer which was chosen to be a ReLU, a Rectified Linear Unit. The function is shown in (1) and a basic graphical representation of RELU is given in Fig. 2.

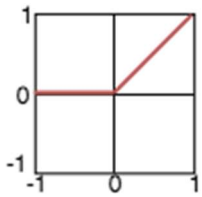

Fig. 2. Showing the graphical representation of ReLU

Another key layer is the max pool layer where the number of elements in the matrix is reduced considering their magnitude which is also called down sampling. This allows the network to be less prone to noise and not to be affected by small changes. Normally, the stride size is equal to one side of the max pool matrix [1]. A simplified version of max pool is provided in Fig. 3.

Fig. 3. Showing a Simplified Version of Max Pool Layer [1]

Fully Connected (FC) layer includes matrix multiplication of the outcome of max pool layer and accumulation of the matrix elements. Weights are given for this layer specifically. It is similar to convolution layer, but more weights are presented.

3.1. Equations

$$f(x) = x^{+} = \max(0, x) \tag{1}$$

4. Hardware architecture

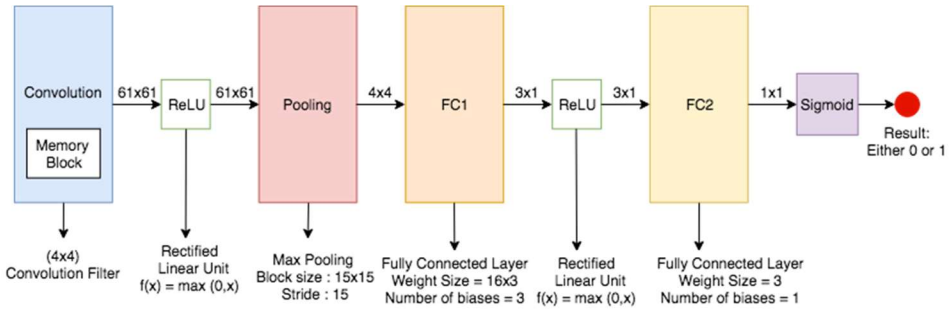

Fig. 1. General Architecture of the Convolutional Neural Network

The weights were initialized from a different top module which was provided in the lab files. There are 67 weights in total; 16 conv, 48 FC1 & 3 FC2. In order to store them, they were placed in registers when the data signal is valid.

4.1. Convolution Layer

Convolution layer is divided into 5 different modules: Controller, memory unit, PE, Accumulator and ReLU. A general view of these modules is given in Fig. 2.

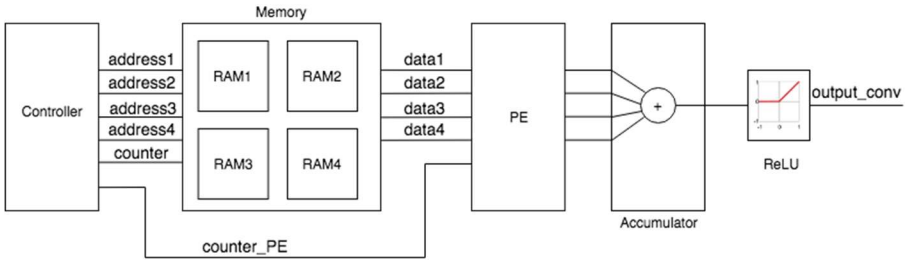

Fig. 2. Overall Architecture of Convolution Layer

Controller mainly synchronizes modules with each other by control signals and make sure memory module is in the right address when read. The 64x64 image data was stored in 4 distributed memories as .COE files. By having 4 different RAMs, we achieve both area and speed efficient design, the basic idea behind the memory unit can be seen from Fig.3.

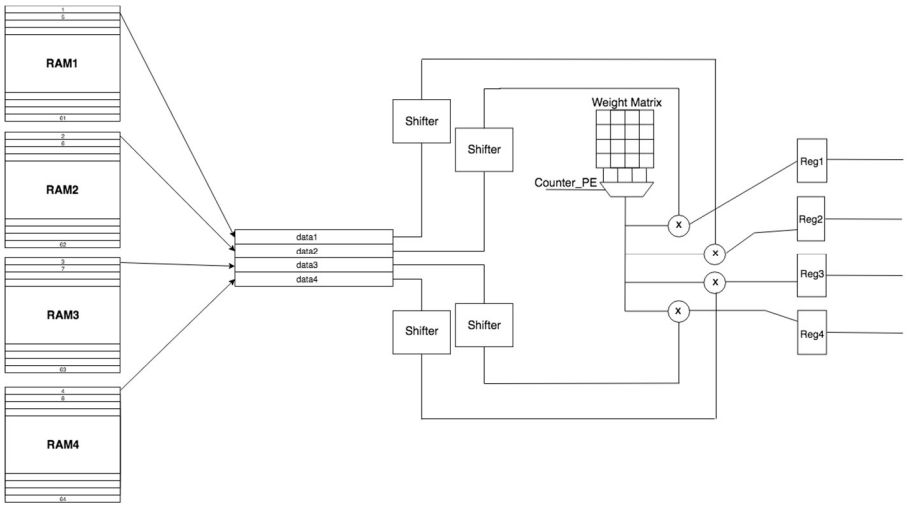

Fig. 3. Showing How Memory and PE Module are Connected

By having 4 different memories, convolution can start without waiting for the first complete 16 data which saves a lot of area and time which can be again seen from Fig. 3.

4.2. Pooling layer

The second part of the design starts with a max pool layer where the number of values carried to the next layer is reduced. A 15x15 filter is applied to find the maximum value in each case and passed to the next stage. Since the output of convolution in this case is a 61x61 matrix the last row and column of the matrix is discarded. At the end a 4x4 matrix is obtained which includes the highest number in each section. Fig. 4. shows the general idea of max pool layer in my design.

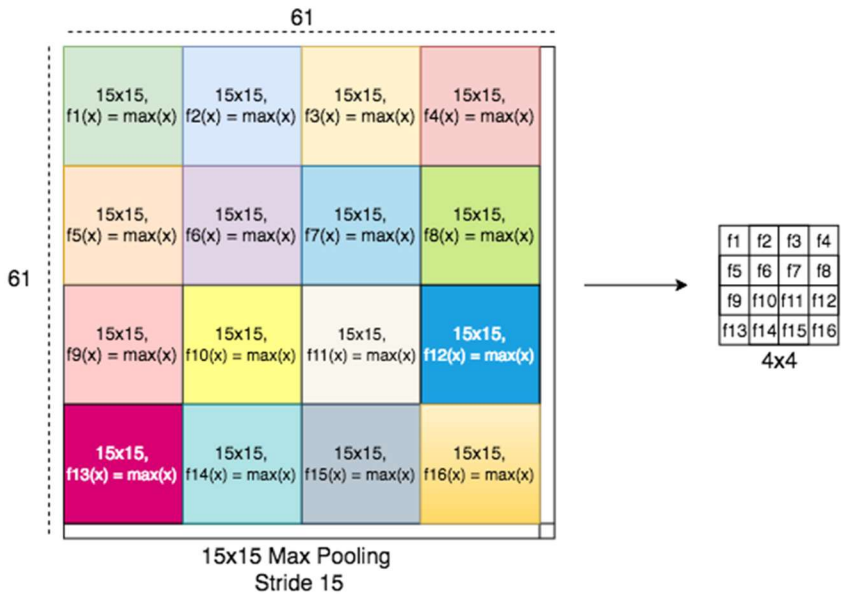

Fig. 4. Showing the Max Pool Layer

4.3. Fully Connected Layer

Fully Connected (FC) layer includes matrix multiplication of the outcome of max pool layer and 48 weights that were given for this layer specifically. In Fig. 4, it is shown that the output of max pool layer (4x4 matrix) is multiplied with 3 different sets of weights 3x16, which are represented as Weight1, Weight2, Weight3. Each set includes 16 unique weights.

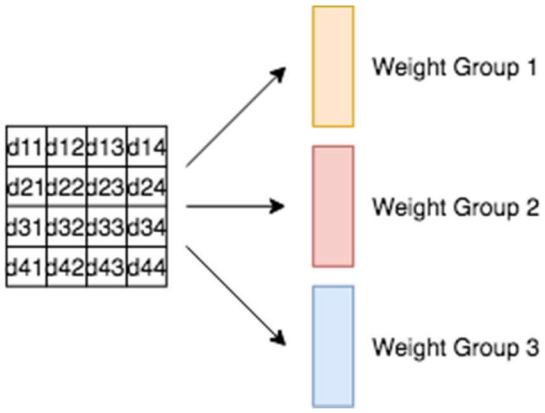

Fig. 5. Showing the First Fully Connected Layer

First FC layer is followed by another ReLU and second FC layer which includes 3 weights. These weights are multiplied and accumulated in one final value which is sent to last layer. If this value is bigger than zero, the image is classified as a rectangle and the design gives a value of 1 or if the value is smaller than zero the image is classified as a circle.

5. Implementation results and analysis

5.1. FPGA implementation

This session presents the FPGA implementation results, including the resource utilization and timing performance.

First of all, Table 1. shows the area utilization of the design in the FPGA and available resources. Also, it can be seen what percentage of the FPGA is used by the design

Table 1. Utilization of Resources

Resource	Utilization	Available	Utilization %
LUT	1677	63400	2.65
LUTRAM	256	19000	1.35
IO	5	210	2.38
FF	777	126800	0.61
BUFG	1	32	3.13

All user specified timing constraints are met according to timing reports of Vivado. Also, it can be seen from the reports that “Worst Negative Slack” is 1.193 ns, “Worst Hold Slack” is 0.110 ns and “Worst Pulse Width Slack” is 3.750 ns. From these values, we can conclude that, the design will perform as it was designed and there won’t be unknown states. The power analysis of the design can be seen from Fig. 1.

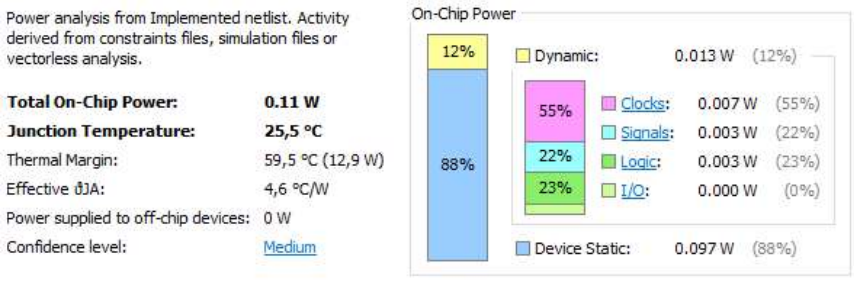

Fig. 1. Showing the Power Analysis of the Design

5.2. Result analysis

Considering the timing, it can be seen from the timing reports that all the specifications are met. However, there are some remarks that should be noted. In the first version of my design, I didn't properly pipeline my fully connected layer, so it was acting like a bottleneck of my design. I didn't meet my time requirements since Fully Connected layers requires a high number of multiplications and additions and most of the time this part includes the longest critical path. As a result, I changed my design to be properly pipelined by adding registers in proper places.

Furthermore, some compression algorithms can be applied to increase the number of arithmetical operations which will highly increase the performance. Also, more research should be done considering the requirements of the convolutional neural networks and look if there are more hardware friendly algorithms to be applied.

6. References

[1] V. Sze, Y. Chen, T. Yang and J. S. Emer, "Efficient Processing of Deep Neural Networks: A Tutorial and Survey," in Proceedings of the IEEE, vol. 105, no. 12, pp. 2295-2329, Dec. 2017.