ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

CNN ACCELERATOR

REAL-TIME EMBEDDED SYSTEMS

FINAL PROJECT

Zehao Chen (SCIPER: 361320)
Josef Thomas (SCIPER: 288838)

23rd May 2023

Contents

CHAPTER 1

INTRODUCTION

1.1 CONTEXT

This particular laboratory serves to implement concepts seen in class. The course itself focusses on embedded systems however this specific laboratory is centered around FPGA programming using VHDL and the Avalon suite. The laboratory expects a certain level of comfort using the VHDL language and requires a large amount of reading in order to become comfortable navigating the documentation.

For this particular project, the objective is to implement a real-time application using a multimaster system. In the case of this report this was comprised of a specialised DMA unit which takes in input data of a handwritten number (MNIST dataset), then using a Convolutional Neural Network is able to predict efficiently the value that this number represents. The design is limited by the FPGA's ressources, however it can distinguish between 3 different numbers.

1.2 SYSTEM DESIGN

This project implements several different components from the Intel IP library and notably a custom hardware accelerator for doing the CNN prediction. It uses:

- Timer (period of 100 μS)
- Performance counter
- Internal FPGA SDRAM with 64 MB
- 8 bits Parallel port for some output with the LEDs
- JTAG DEBUG Module
- CNN hardware accelerator

The full system can be seen in figure 1.1.

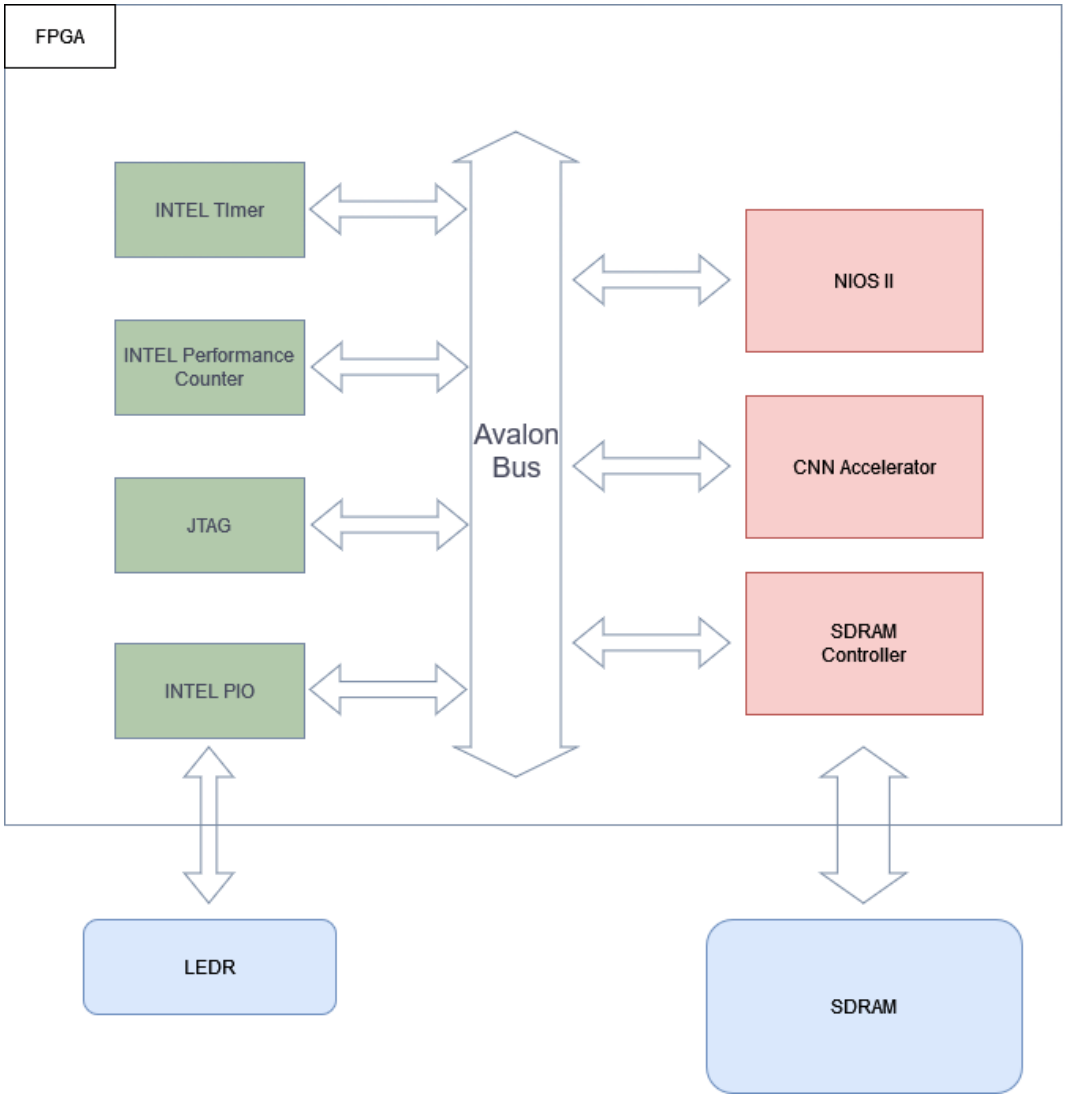

FIGURE 1.1
System(Qsys) implementation

1.3 CNN IN SOFTWARE

Machine learning and notably CNNs are most commonly built in software using the language python with the help of several libraries such as Scikit-learn, Tensorflow, PyTorch, etc. In order to efficiently build and train the CNN model, a jupyter python script in conjunction to the PyTorch package was used. While it is not the focus of the report, it was integral for its completion and allowed us to focus on the hardware implementation. It took a few iterations in order to obtain an accurate model that was also implementable within the constraints of the FPGA's resources. As the weights obtained at the end were mainly decimals and the DE1-SoC doesn't have dedicated FPU units it was necessary to convert them into fixed point notation (signed Q7.8). As such the image, the convolution weights and the neural network weights are all signed Q7.8 in this project. Once these data were converted to the fixed point representation, they were introduced either directly as constants in the vhdl system or in the SDRAM.

CHAPTER 2

CNN ACCELERATOR

2.1 OVERVIEW

As the MNIST dataset (which is what is used for this project) is a collation of handwritten numbers stored as images in conjunction with their labels, there are many ways to classify or predict their labels. Classically there is clustering, edge detection, or neural networks. However, the convolutional neural networks is often regarded as one of the best methods of classification of visual data due to its ability to reduce its dimensionality without losing information through convolution. This also for using a neural network with a low number of layers and neurons without decreasing any performance.

Due to the limitations of the DE1-SoC, the CNN accelerator can only identify the 3 different handwritten digits however as it is essentially a hardware implementation of the PyTorch model it theoretically has the same accuracy of 97.2% on the test dataset. The problem is that the FPGA does not have any dedicated FPUs and as such all data must be converted to fixed-point representations. While this is effective it can lower the precision of the numbers and as such potentially reduce the overall accuracy.

The exact implementation of the CNN is an input of 28x28 pixels (16 bits per pixel), that is convolved with a window of 7, a stride of 7 and a padding of 0. These convoluted windows are then passed through a ReLU activation function and finally passed to single neural network layer of 16 neurons to finally output 3 values. These output values represent the model's certain that the image introduced into the model is a specific number. These values are then passed through an argmax which returns the value of the index of the highest certainty.

The CNN system is represented below in figure 2.1.

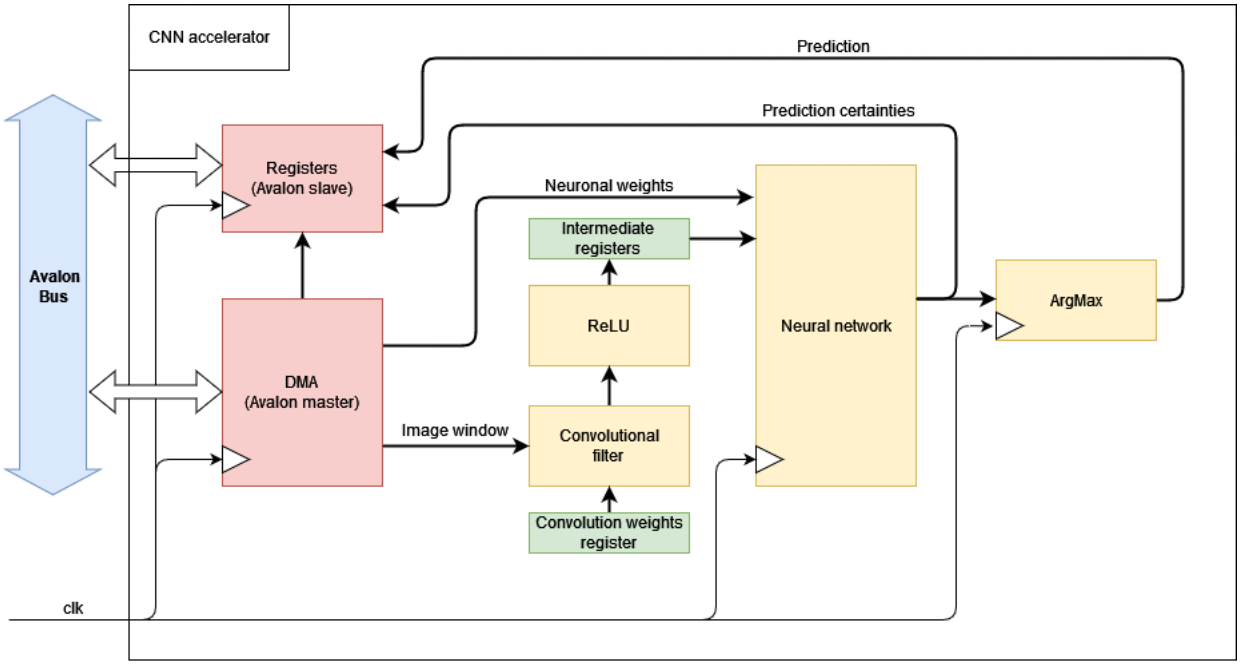

FIGURE 2.1
CNN block diagram

2.2 REGISTER MAP

Signal	Bus size	Direction	Polarity	Description
clk	1	Input	None	50MHz clock
reset_n	1	Input	Active low	Reset signal
address	32	Input	None	Address to read
read	1	Input	Active high	Read enable bit
readdata	32	Output	None	Data read from slave
write	1	Input	Active high	Write enable bit
writedata	32	Input	None	Data write to slave

Register name	Address	Size (bits)	Direction	Description
Start Image	0x00	1	Output	Set high to begin image loading and convolution
Start Weight	0x04	1	Output	Set high to begin loading neurons and predicting output
Image_ADDR	0x08	32	Input/Output	Image base address within SDRAM
Image_LEN	0x0C	32	Input/Output	Length of image in 16 bit pixels
Weight_ADDR	0x10	32	Input/Output	NN weights base address within SDRAM
Weight_LEN	0x14	32	Input/Output	Number of Neurons to load
Prediciton	0x00	32	Input	Numerical representation of prediction (i.e. = 3, etc)
Prediciton0	0x04	32	Input	Certainty that the number is 0 (Q7.8)
Prediciton1	0x14	32	Input	Certainty that the number is 1 (Q7.8)
Prediciton2	0x18	32	Input	Certainty that the number is 2 (Q7.8)

2.3 DMA

This block is controlled by register map. register map gives DMA information about how to read data and where to read data.

Also, register map sends enable signal to start it. Then DMA can give command to DRAM Memory and receive the data from it through Avalon Bus. Then data is sent indirectly to neural network. The usage of a master makes it possible to bypass the processor.

Several key points should be underscored in this context. Primarily, all mathematical operations within the neural network accelerator are executed using fixed-point arithmetic. Each value is encoded with a 16-bit representation, where the upper 8 bits constitute the integer part and the lower 8 bits form the fractional part of a number. This encoding technique suits our requirements well, with one caveat being the necessity for proper shifting of a multiplication outcome. Saturation isn't handled efficiently; values will overflow or underflow by wrapping around, however, because the weights are typically minimal, an overflow scenario is improbable.

Secondly, constraints related to resources on the FPGA posed significant challenges during the design phase. The problem was, in fact, reduced to only recognizing digits 0 to 2, deviating from the original plan of recognizing digits from 0 to 9, due to the fact that the weights of our fully connected layers couldn't be accommodated on the FPGA. Consequently, the weights of the input layer had to be streamed via the DMA in conjunction with the input data.

The accelerator's principal finite state machine remains in an idle state until a start signal is received through the Avalon slave interface. It then dispatches a start command to the DMA unit, and concurrently resets the values accumulated in the neural network. It then waits for the DMA unit to complete its reading process before reverting to the idle state.

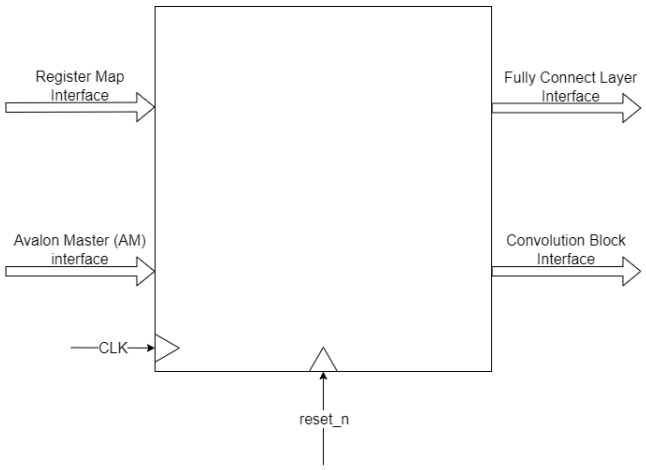

FIGURE 2.2
DMA

2.4 CONVOLUTIONAL BLOCK

The convolutional block is entirely combinatorial as the sequencing is determined by the DMA. It is comprised of two parts:

1. The convolutional filter: which takes in a array of 49 fixed point (16 bit) numbers then convolves them with a constant array of 49 fixed point (16 bit) numbers. This constant array represents the

convolutional weights which were trained using pytorch. The convolutional bias is also added to the final convolved output.

2. The ReLU activation: This takes the convolved result and passes it through the ReLU function as defined above in **??**

These two components are wrapped by the convolutional block which simply connects them together. It also stores the result of each convolution in a dedicated register which is select by the DMA based on which part of the image is being read

2.5 NEURAL NETWORK

The neural network is structured with a single layer of neurons. The specific features of the network in place are outlined in the provided details. Although the different components (such as the layer and neuron) are designed generically, the overall network entity is customized for this particular neural net.

The input layer takes in the image data on a pixel-by-pixel basis and the weight values column by column. It also receives an accumulate signal, instructing it to add the current inputs to its accumulated state. In practical terms, an accumulate pulse will be triggered each time a pixel and its associated weight column are loaded. After all pixel values have been accumulated, the output vector holds the comprehensive information corresponding to the input image.

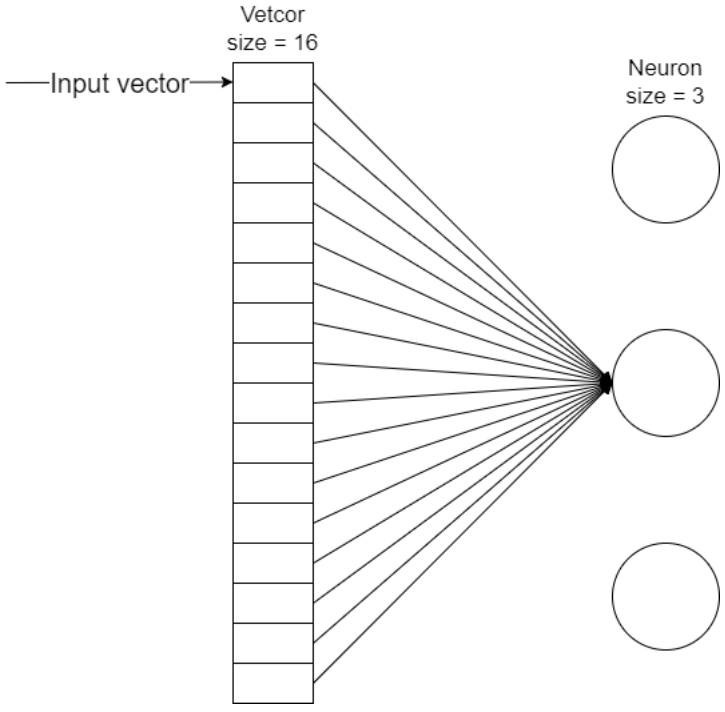

FIGURE 2.3
Fully Connected Layer

2.6 ARGMAX

The argmax block is used to selected the index of max value of fully connected layer. Also, the argmax block is driven by the clock signal.

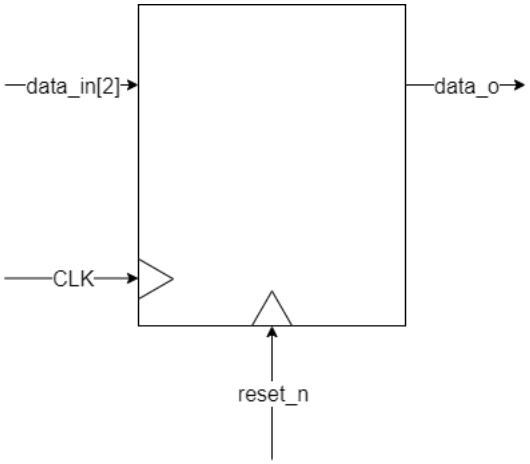

FIGURE 2.4
Argmax

2.7 TEST BENCHES

For the testbench, we use a set of real image data and weights as inputs. The data in the testbench represent the image of digital 2, and in the hardware simulation, the result is 2, which means the result is correct.

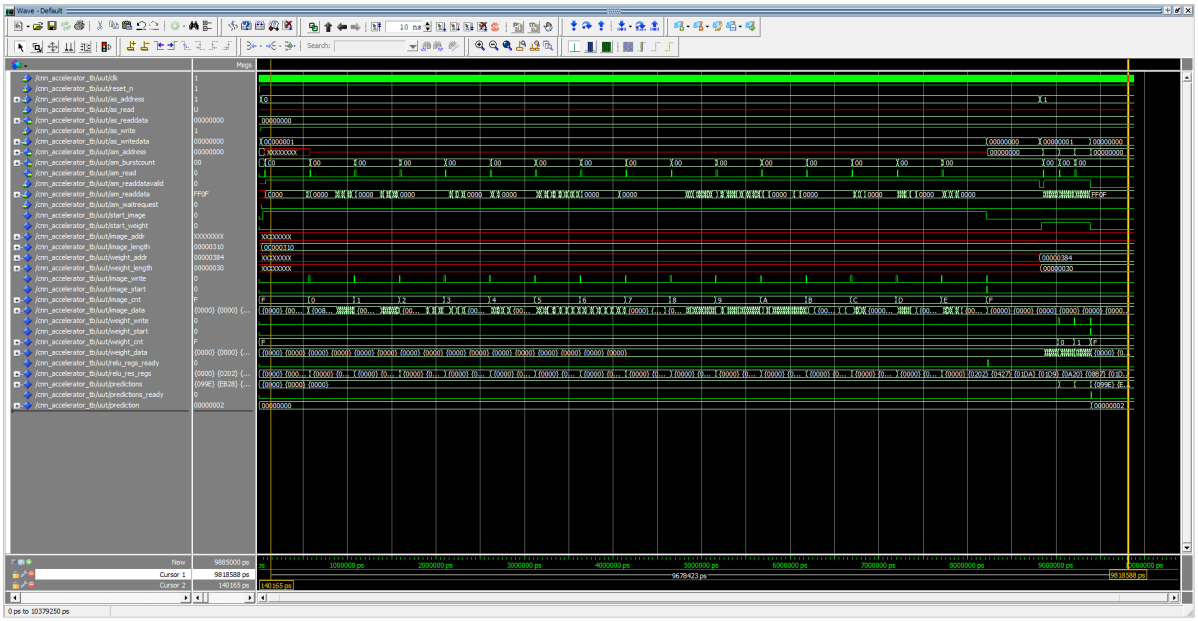

FIGURE 2.5
Testbench of Top level

2.8 OPERATION

In order to use this IP, the following steps need to be taken:

1. First the address of the neural network weights in the SDRAM must be written to the Weight_ADDR register.
2. Then the amount of neural network weights must be written to the register Weight_LEN

3. Now the address of the image to be classified must be written to the register Image_ADDR
4. The length of this image must be written to Image_LEN
5. Once this is done, the system is ready to calculate the convolution. To commence this calculation, set the register Start Image to high.
6. After the convolution is complete, the neural network can be calculated. Set the register Start Weight high.
7. The prediction certainties are now available in the the prediction registers: prediction0, prediction1, prediction2. The index of the predicted value can also be read in the register prediction.

CHAPTER 3

PROFILING

3.1 PROFILING SETUP

The software profiling was done using python because as stated before machine learning is often in done in python. As such it is a fairer test the hardware accelerator against what is would replace. The software was profiled using the decorator %timeit.

The hardware was profiled using the performance counter(using an Intel timer at 50MHz with a period of 100 μs, measuring the time it takes for the accelerator to load the image from memory, pass it through the CNN and then have its prediction ready.

Both systems were done for a single value and then for 1000 values.

3.2 PROFILING RESULTS

Type	Executions	Time(ms)	Time/op(μs)
Software	1	0.487	487
Software	1000	707	707
Hardware	1	0.12	120
Hardware	1000	103.7	103.7

3.3 RESULTS AND DISCUSSION

The CNN accelerator is able to correctly predict numbers with a greater speed in general than in software. Due to the approximative nature of fixed point representation, the accuracy is not quite as good as the model. However using a more powerful FPGA would allow us to store the values over more bits and as such increase the precision. Not only this, but more outputs would be able to be predicted.

From the profiling results we can see than the hardware accelerator is roughly 7 times faster. This represents a significant increase given how machine learning models are often feed large amounts of data.

The benefit of this particular system is that is independent of other components. It simply requires the start address of the image in the memory and the length of image as well as the address of the weights in memory and their length.