Neural Network on FPGA Midterm Report

To

Jianjian Song

From

Joseph Doherty

Robert Fendricks

Ian Kowalski

Report Date

May 23, 2014

Contents

[Introduction 3](#_Toc388562058)

[Artificial Neural Networks 3](#_Toc388562059)

[ZYBO Zinq-7000 4](#_Toc388562060)

[Neural Network on an FPGA Project 5](#_Toc388562061)

[Objectives 5](#_Toc388562062)

[Design Strategy 5](#_Toc388562063)

[Activation Function 6](#_Toc388562064)

[Layer Multiplexing 6](#_Toc388562065)

[The Building Blocks of the Network 7](#_Toc388562066)

[Design and Building Process 8](#_Toc388562067)

[Parts list 8](#_Toc388562068)

[Results 9](#_Toc388562069)

[Conclusion 10](#_Toc388562070)

[References 10](#_Toc388562071)

[Appendix A 11](#_Toc388562072)

[Appendix B 12](#_Toc388562073)

# Introduction

Artificial neural networks (ANN) are a common machine learning algorithm with many real life applications. While they are often implemented in software, their parallel nature could be better implemented in hardware where parallel computations are more easily achieved. The goal of our project is to efficiently implement an ANN in a Xilinx Zynq FPGA chip by taking advantage of its concurrent functionality. We will also be investigating the tradeoffs required to implement the ANN on the FPGA. We will be comparing these results to an established, open-source ANN library implemented in C to gauge both speed and accuracy.

# Artificial Neural Networks

Artificial neural networks are loosely modeled after the structure of how neurons communicate in a brain. Neurons communicate by receiving information through the dendrite attached to its nucleus (see Figure 1). At the nucleus, if the signals trigger a reaction, it will start a chemical chain reaction which propagates down the Axon toward the dendrites of other neurons.

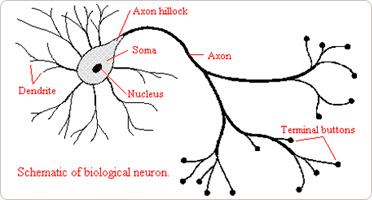


Figure : Example of a biological Neuron [6]

There are a number of different ways to model the neuron in an ANN. We will be using the feed-forward ANN (see Figure 2) to implement our network. In a feed-forward, the neuron is modeled as a hidden unit, which takes inputs from all of the layers before it, applies a predefined function to the inputs, and then outputs the results for the next layer to process. Our ANN will have four layers, the first layer will pass the inputs into the network to the next layer, the second and third layers will be hidden units which will process the data, and the last layer will be the output layer which will sum the results of the third layer and pass the sum out to the user.

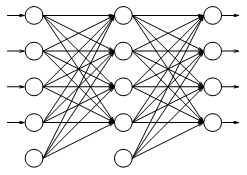


Figure : An example of a feed-forward ANN [5]

Artificial neural networks have been used in many applications such as stock market predictions, cancer diagnosis and many pattern recognition systems like facial recognition. The advantage of using an ANN for these types of problems is that they can be trained to predict the correct answer based on past data and results. In our implementation, we will use a method called back propagation to train our network. Back propagation relies on finding the partial derivatives of the equations that govern the network to adjust certain parameters inside the neurons to achieve a more accurate result in future iterations.

# ZYBO Zinq-7000

To implement the ANN we will be using the ZYBO Zinq-7000 development board. This board uses the Z-7010 chip which contains a 667 MHz dual-core Cortex-A9 processor as well as a programmable logic fabric equivalent to Xilinx’s Artix-7 FPGA [7]. Because the chip we are using has a processor, this will enable us to perform the complicated processes necessary to train the network in an embedded process instead of allocating additional hardware resources to training. Since our goal is to create an efficient network and not an efficient training algorithm, spending hardware resources on implementing the logic necessary to train the network will take up valuable space which could be allocated to additional units inside the network itself. This will simplify the logic design and increase the speed that the network can process data after it has been trained.

# Neural Network on an FPGA Project

## Objectives

The goal of this project was to implement an ANN on an FPGA and show that a concurrent implementation would be very efficient and could display improved performance over a serial, software neural network. Secondary objectives were to analyze the accuracy, speed, and cost of a system which would utilize such a network.

## Design Strategy

In designing an Artificial Neural Network on an FPGA, it is important to be conscientious of the algorithms that are being implemented. Relatively simple operations like multiplication or exponentials, which are common in software implementations of ANNs, are quite expensive when being implemented in hardware. Additionally, floating point arithmetic is complicated and would consume large amounts of resources and each operation requires many clock cycles thus increasing the time required to perform these types of calculations. To alleviate these problems, we decided to use bitwise shifts in place of the majority of our multiplication. This is equivalent to multiplying by a power of two and translates very well into hardware. We have also decided to use only integers in the programmable logic to avoid the floating point difficulties. These changes will increase the speed of our network and decrease the size of the individual units allowing for more to be packed into the FPGA fabric. These algorithmic simplifications would, in theory, decrease the accuracy of the network. However, the saved space would allow for more units to be used which would in turn increase the accuracy. Our hope is that these two factors would balance out and leave us with a competitive ANN.

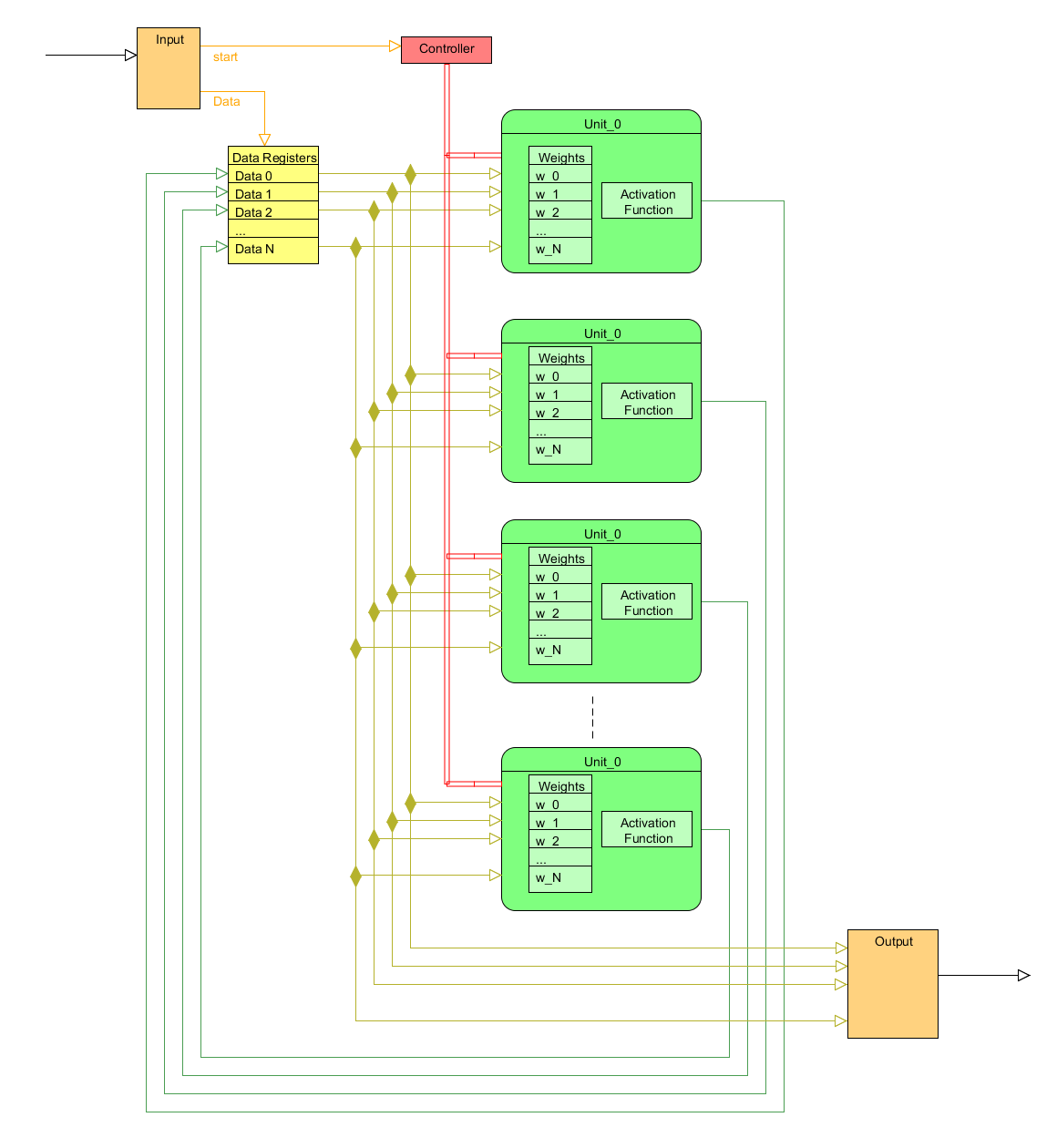


Figure : Abstract Network Design

### Activation Function

Many software implementations of activation functions use Gaussian functions or exponentials in order to compute when a hidden unit has been activated or not. Both of these approaches would be prohibitively expensive on an FPGA. Our implantation will use the Elliot Symmetric Activation Function, which has no exponents in it. The Elliot Symmetric Activation Function is represented by the following equation:

Though it doesn’t eliminate the need for division, these operations will be much less expensive that calculating exponents.

Due to our decision to use an integer network, we had to modify the activation function by scaling it to a greater range than -1 and 1. Our current function is:

### Layer Multiplexing

In a standard ANN, each layer has a dedicated unit. For instance, the input units feed into the first layer of hidden units which feed into another layer of hidden units or to the output units. These units typically have unique functionality depending on which layer they are in. Our approach uses layer multiplexing, where each unit acts as the first hidden unit on the first cycle, the second hidden unit on the second, and the output unit on the third. Our input unit is handled by a register bank into which the input data is read in from an outside source.

This approach was chosen because it allowed us to fit more units per layer inside the FPGA fabric. This did require us to use more RAM space to hold the weights used to train the network, but this was much preferred to the space lost for having specialized units for each layer.

## The Building Blocks of the Network

To put together the whole neural network, we used Xilinx’s Vivado design suite to code and test our units using Verilog. Where it was applicable, we used python scripts to generate components that would change depending on the number of units that we wanted in the network. A full diagram of our top level schematic can be found in Appendix A.

We used two separate controllers to control the timing of our network. One is named the “Network\_Controller” which controls what layer our network is currently on (either a hidden unit layer or output unit layer) and which tells the second controller what to do. The second controller is called the “RAM\_Read\_Driver”. It is in charge of selecting what RAM address to read from, which unit to write the memory from ram into, when to write the data, and when to trigger the neural units to sum up their inputs. The RAM itself holds all of the weights of each unit and loads them in when need.

The Data\_Reg\_Bank unit is used to store the values that the units must process. This will take in the initial values from the ROM/Serial input and will also take the results of the previous layer to feed into the inputs of the current layer.

The Ram\_Mux takes in data from the RAM and distributes it the neural units, leading the weights to their correct destination.

The Units in Appendix A each represent a Neural Unit, which itself is made of multiple components. The components inside include a WeightRegBank (to store the values of the weights read in from RAM), a Multisum module (To sum the inputs into the unit), an Elliot\_Activation unit (Which applies our Elliot function to the summed inputs), and a LayerMux (to determine whether we want to take the result of the Elliot function or the summed inputs as the output for the unit).

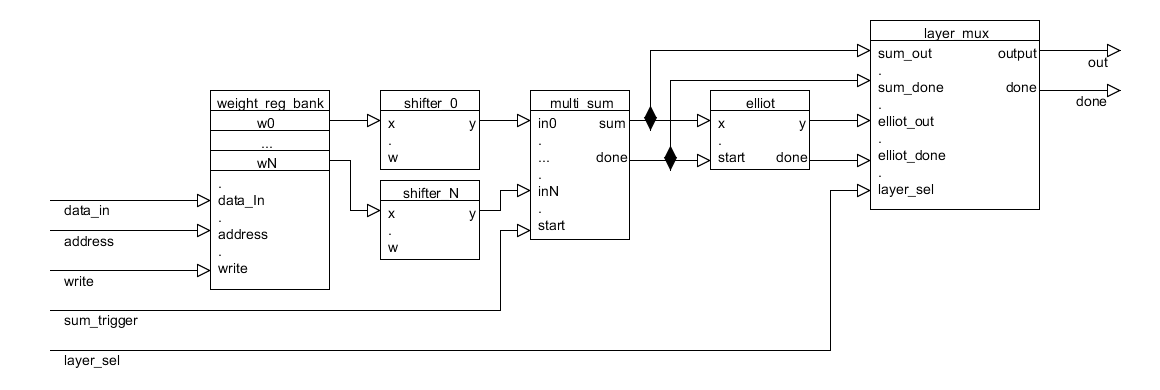


Figure : Block Diagram of Neural Unit

## Design and Building Process

We spent the first few weeks of the term researching how we should implement our network. We went through various design decisions, with our main concern being space efficiency.

# Parts list

|  |  |  |  |
| --- | --- | --- | --- |
| Description | Quantity | Price | URL |
| Zybo Zynq-7000 Development Board | 3 | $125 x 3 | <http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,719,1197&Prod=ZYBO> |
| Zynq Accessory Kit | 3 | $20 x 3 | <http://www.digilentinc.com/Products/Detail.cfm?NavPath=2,719,1197&Prod=ZYBO> |

The total cost of the project was $435 for each of us to have a board and accessory kit. Per kits, the price would only be $145.

# Results

During simulation (See Appendix B), we found that our network, with four units and 2 hidden layers, runs for about 337 clock cycles. Each hidden layer took 137 cycles to complete, and the final output layer took 82 clock cycles to complete. The reason for the discrepancy is that the output layer does not have to wait for the completion of the Elliot function, which seems to add 55 cycles to the calculations. This is mostly due to the division it must perform.

The waveform shows that our neural network seems to be functioning correctly when simulated. After the start signal is received, a one shot starts the rom controller. This controller reads from the rom and sends 4 rom\_write signals, one for each unit. At the end of this, the ROM\_Controller tells the Network\_Controller to begin. Immediately afterwards, the Network\_Controller send a signal to the RAM driver to start reading from ram. It sends four sets of four pulses to the RAM Mux; during each write a new value from RAM is stored into the four weight registers of each unit. At the end of these writes, the sum trigger goes high to tell the units to begin summing its inputs. A few cycles later, the summation is complete and each unit begins performing the Elliot Function. When the Elliot function is finished, the done signal is sent out, signifying that the value of the unitN[31:0] signals is finished. This cycle occurs three times, once for each layer of our network. On the last cycle, the layer\_sel signal goes low, which tells us that this layer is an output layer.

Unfortunately, we were not able to successfully get our network to simulate on the Zybo Zynq board. We were able to get our bitstream on to the board, however when we tried to take measurements of the output of the network, we were only able to measure noise. It would appear that there are timing issues with the network on the board. According to our simulation, the network should operate correctly, but due to time constraints we aren’t able to debug the circuit further.

# Conclusion

We were able to successfully design an artificial neural network for an FPGA, however there are inherent difficulties due to a heavy reliance on floating point arithmetic and large amounts of memory requirements with standard ANN implementations. Our method alleviates these problems but future work is required to explore the ramifications of modifications and their effects on accuracy and speed. Further, we were not able to rectify our timing issues when the network was put on the FPGA. For future work, our implementation should be redesigned to account for timing issues when implemented on a board, at which point more accurate timing and testing can be done.

# References

[1] L. Prechelt. (1994, Sep. 30). *Proben1 – A set of Neural Network Benchmark Problems and Benchmarking Rules* [Online]. Available: <http://digbib.ubka.uni-karlsruhe.de/eva/ira/1994/21>

[2] V. Salapura, *et al.* (1995, Feb.) *A Fast FPGA Implementation of a General Purpose Neuron* [Online]. Avaliable: <http://www.researchgate.net/publication/2421858_A_Fast_FPGA_Implementation_of_a_General_Purpose_Neuron>

[3] N. M. Botros and M. Abdul-Aziz, "Hardware implementation of an artificial neural network using field programmable gate arrays (FPGA's)," Industrial Electronics, IEEE Transactions on , vol.41, no.6, pp.665,667, Dec 1994

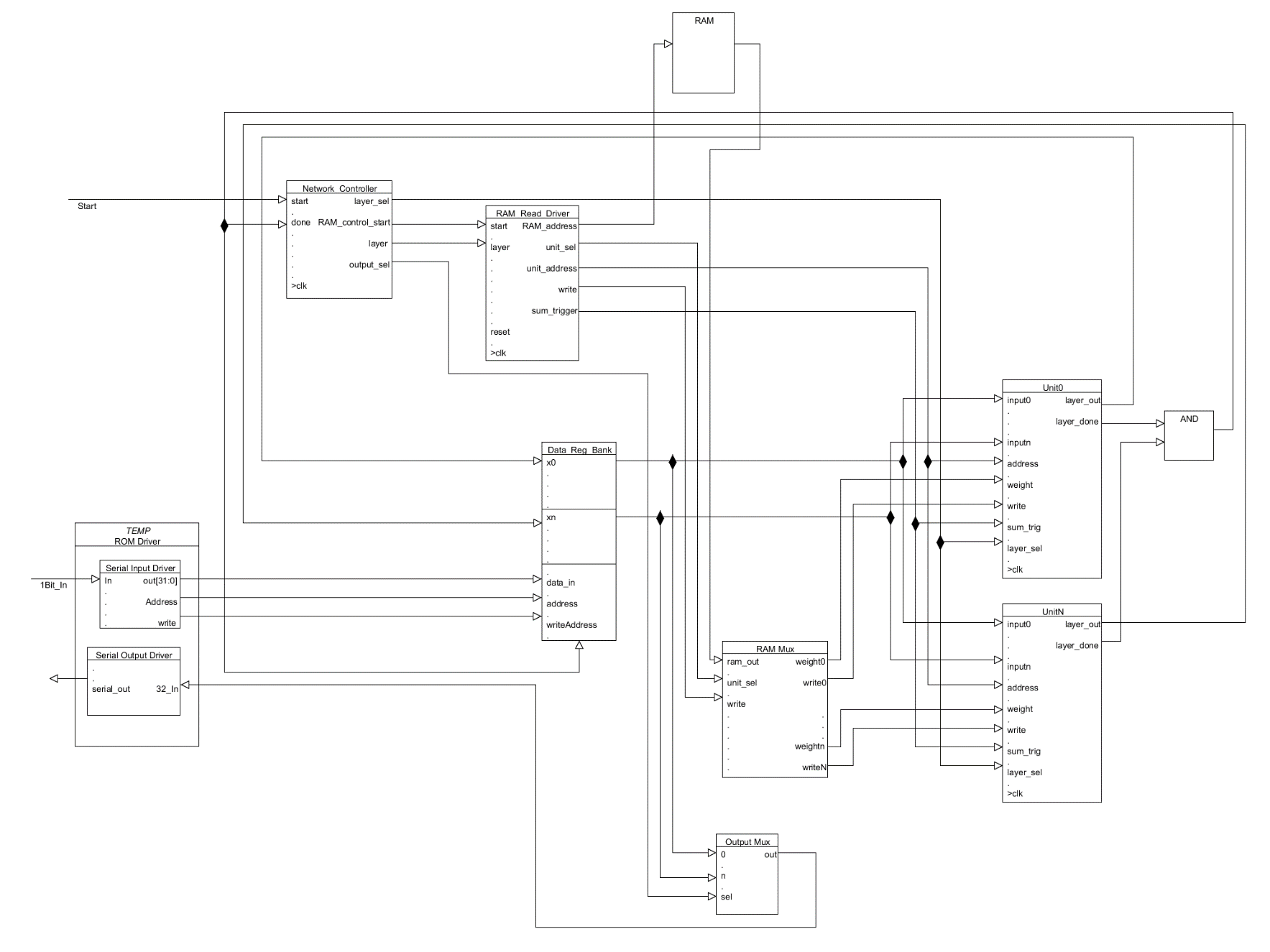
[4] P. Sibi, *et al.* (2013, Jan. 31) *Analysis of Different Activation Functions using Back Propagation Neural Networks* [Online]. Avaliable: <http://www.jatit.org/volumes/Vol47No3/61Vol47No3.pdf>

[5] S. Nissen (2003, Oct. 31) *Implementation of a Fast Artificial Neural network Library (FANN)* [Online] Available: <http://download2.polytechnic.edu.na/pub4/sourceforge/f/project/fa/fann/fann_doc/1.0/fann_doc_complete_1.0.pdf>

[6] B Minasny, *et al.* (2004) *Soil Hydraulic Properties* [Online] Avaliable: <http://hopmans.lawr.ucdavis.edu/1_soil_hydraulic_properties.htm>

[7] Xilinx [Online] Avaliable: <http://www.xilinx.com/products/boards-and-kits/1-4AZFTE.htm>

# Appendix A



# Appendix B

