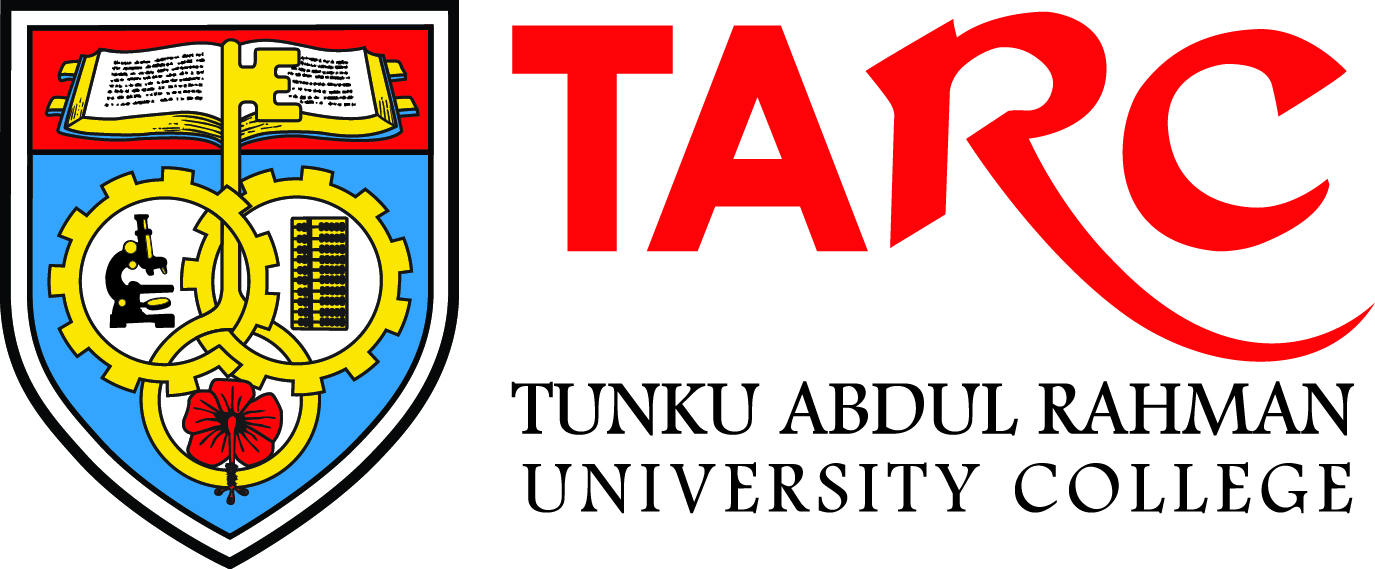
**Development of an FPGA based Iris Recognition System using Self Organizing Map for Augmented Security**

**By**

**Lee Shyan Feng**



**Faculty of Applied Sciences and Computing**

**Tunku Abdul Rahman University College**

**Kuala Lumpur**

Final Year Project

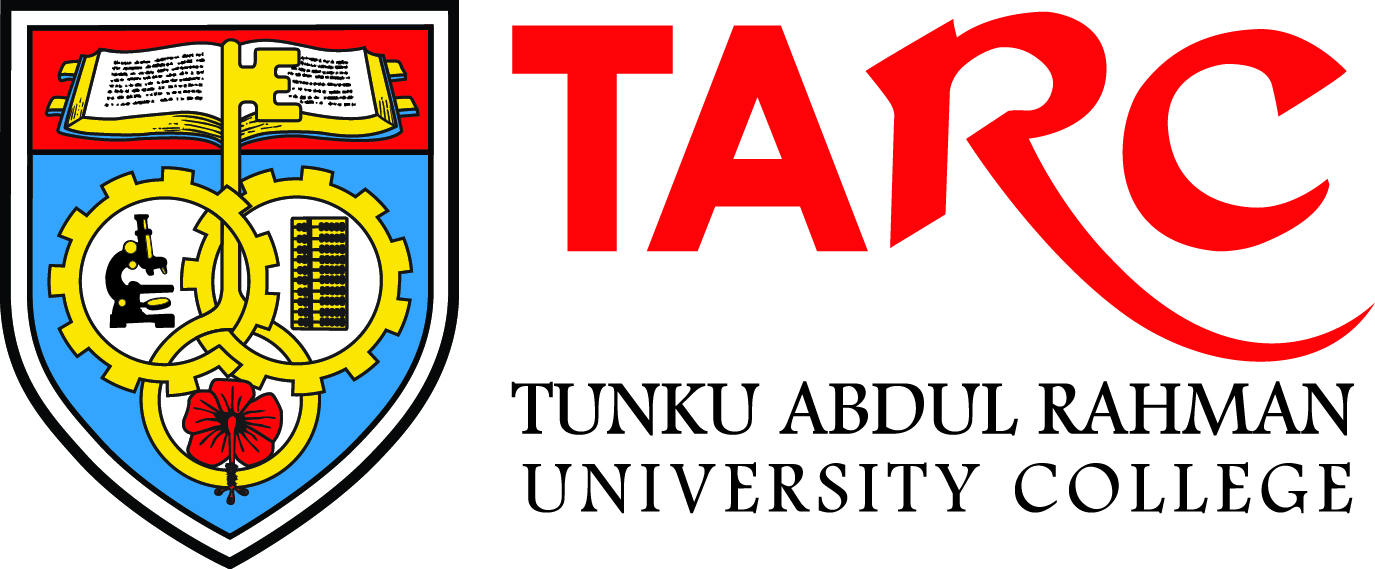
**Development of an FPGA based Iris Recognition System using Self Organizing Map for Augmented Security**

by

Lee Shyan Feng

Project supervisor

Miss Michelle Lim Sern Mi



This is a project dissertation submitted to the Faculty of Applied Sciences and Computing in partial fulfillment of the requirement for the award of Bachelor of Science Degree, Tunku Abdul Rahman University College and Campbell University, Buies Creek, U.S.A.

Department of Physical Science

Faculty of Applied Sciences and Computing

Tunku Abdul Rahman University College

Kuala Lumpur

**Acknowledgement**

I would like to express my sincere gratitude to my project supervisor, Miss MICHELLE LIM SERN MI for her guidance, encouragement and support in my project. Thanks to her insights and suggestions on technical problems during implementation. My project was able to be completed and meet the minimum requirements before the presentation date. I would like to thank her again for her patience and understanding of our difficulties and she has given me helpful comments to complete my project. I would like to express gratitude to my teammate, YAP KEN MUN and JASON CHUAH KWONG HOOI. We helped each other to go through all the difficulties and we inspire each other when we lose our direction. Last but not least, I would like to thank all my lecturers and fellow course mates for providing valuable advice and support throughout the duration of this final year project.

**Abstract**

This project focuses on an Iris Recognition System (IRS) using Artificial Neural Network (ANN) for augmented security. This work proposes an improved version of the Self-Organizing Map (SOM) algorithm which is a subset of the ANN with an additional voting system to enhance identification accuracy. This improved SOM algorithm resolves conventional issues of the recognition accuracy, speed and resource usages by using a voting system and modification in each sub-blocks. This IRS consists of three main sections, i.e. the Iris Acquisition and Segmentation, Iris Normalization and Compression and lastly iris recognition using the SOM algorithm. The preliminary steps for recognizing an iris is to detect the iris by using CMOS camera, process the captured image and uses the SOM technique and its voting system to perform recognition. Here, conventional iris recognition techniques and its voting system were reviewed. The recognition step is performed by using the SOM algorithm to analyze and arrange the iris input to obtain a better recognition rate with higher accuracy. SOM uses unsupervised training to solve the problem of classification used for low dimensional spaces and high dimensional data. The input training set arranges itself. Training is the process of matching the input vectors and the node weights which will be optimized to recognize the inputs. The matching node will be arranged by the input array. The voting system will receive 3 output arrays with 3 different starting points to start the vote, this improves recognition accuracy. The proposed IRS will be modelled, designed in Verilog code and simulation is performed using Modelsim for functional verification purposes. The complete Verilog code is integrated to perform real-time recognition operation. The iris recognition algorithm using SOM with a voting system is able to achieve a recognition accuracy of 94% and a recognition speed of 1-2 second. This system is likely to be installed in places which require augmented security to enhance safety.

Contents

[Chapter 1: Introduction 1](#_Toc440455938)

[1.1 Objectives 1](#_Toc440455939)

[1.2 Problem Statement 1](#_Toc440455940)

[1.3 Background 3](#_Toc440455941)

[Chapter 2 Literature Review 8](#_Toc440455942)

[2.1 Conventional Methods for Image/ Iris Recognition 8](#_Toc440455943)

[Chapter 3 Methodology 11](#_Toc440455944)

[3.1 Neuron Initialization Block 14](#_Toc440455945)

[3.2 Hidden Layer Block 16](#_Toc440455946)

[3.3 Weight Optimization Block 21](#_Toc440455947)

[3.4 Iteration-Check Block 27](#_Toc440455948)

[3.5 Voting System Block 31](#_Toc440455949)

[3.6 On-Chip Training Block 35](#_Toc440455950)

[3.7 Integrated SOM Module 39](#_Toc440455951)

[Chapter 4 Results and Discussions 41](#_Toc440455952)

[4.1 Neuron Initialization Block 41](#_Toc440455953)

[4.2 Hidden Layer Block 44](#_Toc440455954)

[4.3 Weight Optimization 50](#_Toc440455955)

[4.4 Iteration-Check Block 56](#_Toc440455956)

[4.5 Voting System Block 60](#_Toc440455957)

[4.6 On Chip Training Block 64](#_Toc440455958)

[4.7 The Integrated Proposed SOM Block 71](#_Toc440455959)

[4.8 Full Integration 75](#_Toc440455960)

[Chapter 5 Conclusion 77](#_Toc440455961)

[5.1 Conclusion 77](#_Toc440455962)

[5.1 Future Development 77](#_Toc440455963)

[Reference 78](#_Toc440455964)

[Appendix 79](#_Toc440455965)

## Chapter 1: INTRODUCTION

### 1.1 OBJECTIVES

1. To investigate several different iris recognition techniques and algorithm including the Self Organizing Map (SOM) and voting system techniques for the Iris Recognition System (IRS) from past literatures.

2. To model, design and verify all sub-modules of the SOM and voting system using HDL based testbench in Modelsim of Mentor Graphics environment.

3. To integrate, analyze and perform hardware implementation on all the integrated sub- modules of the SOM based IRS algorithm with voting system into the Cyclone II FPGA for hardware verification purposes.

### 1.2 Problem Statement

The aim of this project is to improve the performance, recognition accuracy and area by past researchers related to Iris Recognition. The conventional Iris Recognition Systems (IRS) is shown in Figure 1. The technique used in the Figure 1 is by means of comparing the output vector stored in the database. The output array will only be compared once with the output vector in the database. This will cause a problem if the input is actually a different user but the pattern is not much difference from the trained database, a wrong recognition outcome is likely to happen which in turns reduces recognition accuracy.

C:\Users\Feng\Desktop\Final FYP Folder\result\conventional block diagram.tif

Figure 1: Conventional IRS without a Voting System

(Omaima *et al.*, 2012; Yap and Lim, 2014)

Therefore, to improve performance of the conventional Self Organizing Map (SOM) architecture as shown in Figure 1, the proposed solution is to use a voting system before it matches with the output array in the database. The voting system will receive three output arrays from three different starting points to start the vote with the data array. If the *updatedNetwork* array after voting is higher than a 67% match (2/3 majority) with the database then the user is the correctly matched person.

### 1.3 Background

Traditionally, security identification methods used keys and passwords to secure locks but now these methods are no longer appropriate to properly protect our property because physical keys are often misplaced while virtual passwords are often forgotten. Even if passwords were well memorized and keys were well kept, the ever rampant crime rates causes easily guessed passwords or easily penetrated locks to be unsecured both virtually and in reality. Furthermore, using keys, password and fingerprints that are contact based might leave behind some bacteria due to contact and touch. For example, such contact might cause the spread of the recent Ebola virus that is rampantly spreading via body fluid or blood of an infected animal or person whose touch or contact might contain such a fluid. Therefore, a contactless recognition system such as iris recognition is most welcomed in such situations.

Iris recognition is one of the most trustworthy contactless biometric technologies today. This is due to the unique characteristics of the human iris that varies from one human to another (Kaushik *et al.* 2011). The first person who introduced the biometric technology concept by means of iris recognition was Frank Burch in the year 1936. In 1993, Daugman is the one who proposed the first algorithm which is the iris location technique. Iris location technique detects the limbic boundary followed by the pupil boundary. The iris image is converted to Cartesian form once it is located, it has also been projected onto a dimensionless pseudo-polar coordinate system by other researchers (HimanshuRai *et al.* 2014).

The Aritificial Neural Network (ANN) algorithm is generally divided into two main parts, unsupervised and supervised as shown in Figure 1.2. The SOM is under the category of an unsupervised ANN algorithm (Shyam *et al. 2005)*. The first idea of the SOM was first suggested by Christoph Von Der Malsburg in the year 1973 where it was developed and refined by Professor Teuvo Kohenen in year 1982.

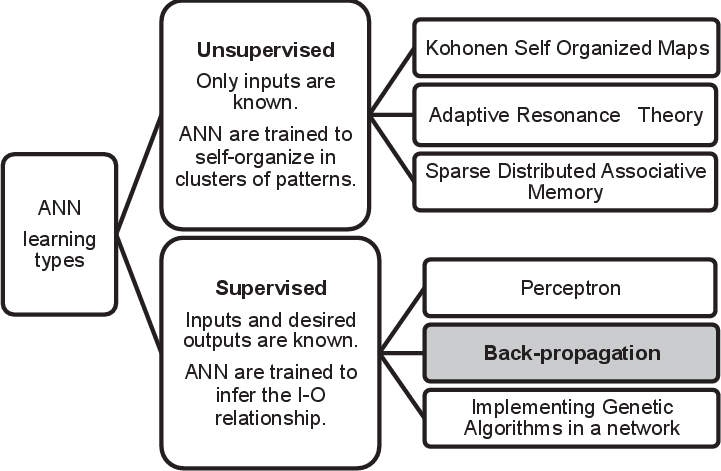
****

Figure 1.2: ANN Learning Types

Early stages of the neural networks were published by the McCulloch and Pitts in the year 1943. They introduced the several types of neural network and its logic function which had been truly influencial to people of that time. In the year 1949, based on the uniqueness of individual neurons, the idea of classical psychology is appearing everywhere including animals and this area was pursued by Donald Hebb. He developed his own synapses of neuron learning law called the Hebbian rule (N. Yadav *et al* 2015).

Later in the year 1951, the first neuro-computer which was known as The Snark was constructed by Marvin Minsky but the information of processing functions did not carry out even though it was successfully operated from a technical standpoint (N. Yadav *et al* 2015).

In the year 1957, the first ANN, Perception was developed by Frank Rosenblatt (Kendar & Shelja *et al.* 2013). After two years, a new processing element which was known as ADALINE was developed by Bernard Widrow and Marcial Hoff in the year 1959. The book entitled “Perceptrons” was published in the year 1969 by Minsky and Seymour Papert but this book had been deemed as a failed attempt because it could not solve the XOR function (N. Yadav *et al* 2015).

The development of ANN was applied in the robotics with thinking computers by engineers from two different countries i.e. the United States and Japan. They were James Anderson, Kunihiko Fukushimica, Teuvo Kohonen (creator of SOM) and John Hopfield (Kendar & Shelja *et al.* 2013).

Currently, the trend of this ANN design is fast moving towards the computer vision, processing for image or signal, recognition, medical image analysis, remote sensing for industrial inspection and scientific exploration due to the progress in the year 2013 (Kendar & Shelja *et al.* 2013). This work tends towards iris recognition. There are many different algorithms that can be used for pattern recognition, i.e. Hamming Distance, Fuzzy and SOM. All of these algorithms have their own benefits and drawbacks. Table 1 shows a comparison table on past researcher's work regarding iris recognition. From this table, we note that the Hamming Distance (HD) design by HimanshuRai achieves the highest recognition accuracy at 99.60% due to the case of 1D Gabor wavelet while the Melin *et al*. 2012 design has 99.56% recognition accuracy. Melin uses Fuzzy algorithm to complete her research due to the number of membership functions that can be adjusted and the number of inference models that can be changed.

Table 1: Past Researchers’ Recognition Work on Iris Recognition

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter  Researcher (Year) | Accuracy (%) | Algorithm | Advantages | Disadvantages |
| HimanshuRai *et al.* (2014) | 99.60 | Hamming Distance | -better accuracy on recognition | -less error correction |
| Melin *et al.* (2012) | 99.56 | Fuzzy | -better integration responsive | -the membership function is difficult to estimate |
| Yap and Lim *et al.* (2014) | 90.00 | Self- Organizing Map (SOM) | -stable accuracy on recognition  -efficient way on solving pattern classification problem | -time consuming on recognition |
| Lee *et al.*  [Author]  (2015) | 94.00 | ANN(Self- Organizing Map (SOM)) with Voting System | -high accuracy on recognition  -efficient way on solving pattern classification problem | -need to take 3 output to start the vote |

Among all the literatures that had been studied in Table 1, the method used by Yap and Lim *et al.* (2014) is quite similar to this proposed IRS. The ANN algorithm used is the same in terms of training and testing. However, the ANN used by Yap and Lim *et al.* (2014) used a comparator to compare the previous result against the new result. Once the new result is different from the previous result then it will reset. This will reduce the recognition rate.The proposed ANN uses the SOM scheme with a voting system that takes three output arrays to start the vote. It has a recognition probability higher than 66.67%. The resource usage is also lesser compared to Yap and Lim *et al.* (2014).

The ANN applied to the IRS is SOM with voting system because it is an iris recognition method with a higher accuracy and is less time consuming. The expected accuracy is between 90-94% due to the limitation of space and lighting. The application of IRS is augmented security such as for low cost home security and criminal recognition purposes.

In a nutshell, this report is a design and development of an IRS implemented on FPGA. It begins with an introduction to the background, past to current research on the topologies and strategies of iris recognition design in Chapter 1. Chapter 2 is the literature review of the SOM and Chapter 3 proposes the design methodology while the result and discussion are summarized in Chapter 4. Finally, Chapter 5 concludes this work.

## Chapter 2 Literature Review

### 2.1 Conventional Methods for Image/ Iris Recognition

#### 2.1.1 Support Vector Machine and Hamming Distance

The combined algorithm of Support Vector Machine (SVM) and Hamming Distance (HD) is used to performing training and testing for the iris recognition. The method for HD is 1D Log Gabor wavelet and HAAR Wavelet decomposition for SVM. The wavelet of 1D Log Gabor convolves the normalised iris region. The first step of this HD is to break the normalised pattern into 1D signals with a number so that the 1D Gabor wavelet is able to convolve with the 1D signals (HimanshuRai *et al.* 2014). The 1D signals are taken from each row of the 2D normalised pattern and the circular ring on the iris region will be corresponded (HimanshuRai *et al.* 2014). By referring to the Table 2.1, the advantage of this method is to improve the accuracy by means of non-false rejection and the disadvantage is the longer recognition time because it has to go through two subsidiary methods. Other than that, the accuracy of this method goes up to 99.88%.

Table 2.1: Details and Information of the SVM and HD method

|  |  |
| --- | --- |
| **Details** | **Information** |
| Advantage | - improve the accuracy by means of non-false rejection |
| Disadvantage | - longer recognition time |
| Accuracy | - 99.88% |

#### 2.1.2 Modular Neural Network with Fuzzy System

This method of Modular Neural Network (MNN) with Fuzzy System (FS) solves the pattern recognition problem with improved accuracy and efficiency. The FS is used for improved response time during integration. The architecture of this MNN and FS structure is optimized by using genetic algorithm. The first step of this algorithm is to optimize each module with two hidden layers, types of learning algorithm and the goal error (Melin *et al* 2012). There are 3 types of learning algorithm i.e. the scaled conjugate gradient, gradient descent with adaptive learning, as well as the momentum with adaptive learning and goal error (Melin *et al.* 2012). Other than that, to increase the recognition rate, the fuzzy integrator is optimized by a genetic algorithm. There are 3 phases in the development of a genetic algorithm. The first phase is to perform FS optimization, types of membership function and the parameters of membership functions. The second phase is to optimize number of rules and membership functions. The last phase is for the genetic algorithm decide on the type of fuzzy logic to be used (Melin *et al* 2012). By referring to Table 2.2, the advantage is that if unfortunately one of the module fails,it can still work with another module. The disadvantage is the resource usage might be higher because it contains more than one module in the system. The recognition accuracy for this method is 93.92%. is

Table 2.2: Details and Information of the MNN and FS Method

|  |  |
| --- | --- |
| **Details** | **Information** |
| Advantage | -will not stop working if one of more module fails |
| Disadvantage | - resource usage is high |
| Accuracy | - 93.92% |

#### 2.1.3 Self Organizing Map

Self Organizing Map (SOM) is an unsupervised learning algorithm. The SOM algorithm is divided into 6 main steps (Shyam *et al.* 2005). The first steps is to initialize the weights for each node in the network. The second step is in the set of training data which will randomly select a vector and pass it to the network. The third steps is the neuron competing with each other to become the winner (Yap and Lim *et al*. 2014) which is similar to using the Best Matching Unit (BMU) to find the winning mode which has the most similar weight as compared to the input vector. Next, the forth step is to calculate the neighborhood radius but in theory it is set to the radius of the network. The fifth step is that if the nodes were found to be within the neighborhood radius then it will be adjusted to be similar to the input vector as shown in Figure 2. It shows that the radius reduces every time it completely updates the node’s weight. It begins at a radius of 5 and slowly reduces until it reaches a radius of 1. Finally, steps 2 to 5 will be repeated until the predetermined iteration number is reached (Shyam *et al.* 2005).

C:\Users\Feng\Desktop\Final FYP Folder\som\thesis.tif

Figure 2: Neighboorhood radius update with iterations

## Chapter 3 Methodology

This project is divided into three main sections which are the Iris Acquisition and Segmentation, the Iris Post-Processing as well as the SOM recognition sections as shown in the Iris Recognition System (IRS) block diagram of Figure 3.1. Firstly, the image pre-processing block captures and segments the iris into a greyscale image and localizes the pupil/ iris by providing specific coordinates and radius values for the subsequent block. Next, the iris post-processing block normalizes and compresses the iris data into a 10 x 10 matrix array for recognition by the final SOM algorithm block.

This work focuses on the SOM section as shown in Figure 3.1 which contributes to one third of the entire IRS. There are six sub-modules in this SOM Block which are the Neuron Initialization Block, Hidden Layer Block, Weight Optimization Block, Iteration Check Block, Voting System Block and the On-Chip Training Block. Initially, the Neuron Initialization Block (Block 1) produces a sample data array which gathers the value of “1” in the middle of the array for further usage of the SOM algorithm. Secondly, The Hidden Layer Block (Block 2) is the competition section that estimates the winner neuron distances between the selected input vectors and the neurons to be fed into the Weight Optimization Block (Block 3). The Weight Optimization Block outputs the updatedNetwork array with updated neuron within the neighbourhood radius. Next, the Iteration Check Block (Block 4) compares the previous network to the current network. The Voting System Block (Block 5) performs voting on data array with three different candidate arrays and outputs the winning candidate array. Finally, the On-Chip Training Block (Block 6) can be divided into two parts, training iris data to be stored as database and database matching for iris recognition. These steps are summarized by the flow chart of Figure 3.2. The following sub-sections describe

## Chapter 3 Methodology

**C:\Users\Feng\Desktop\Final FYP Folder\Block Diagram.tif**

Figure 3.1: IRS Block Diagram

Iris Acquisition and Iris Segmentation produces coordinate

Iris Post Processing produces data array

Store data array into database

Neuron Initialization Block initialize *initNetwork* array

Select an input vector from data array

Obtain Weight from *initNetwork* and *updatedNetwork* and pass to Hidden Layer Block

Obtain Winner Weight and pass to Weight Optimization Block

Update Winner Weight, neighbours neuron and neighbourhood radius

Generate *updatedNetwork* array

Iteration check *updatedNetwork* array with *prev\_network* array in Iteration Check Block

Are both arrays the same?

**No**

Same for >10000 iterations?

**Yes**

**No**

Pass *updatedNetwork* to Voting System Block

**Yes**

All 3 arrays received by Voting System Block?

**No**

Generate a *votedNetwork* and pass to On-Chip Training Block

**Yes**

Recognition with database

Output result on LEDR

Figure 3.2: Flow Chart for the complete SOM algorithm

each of these SOM sub-blocks in detail in terms of HDL coding methodology and its corresponding hardware implementation.

### 3.1 Neuron Initialization Block

Neuron Initialization Block (Block 1) is a module used to initialize a 10x10 matrix network array with 1 bit binary data that will further be used by the Hidden Layer Block (Block 2). There are many ways to initialize the network, the three common ways are random value initialization, sample data initialization and with linear initialization. In this module, the method chosen is the sample data initialization method that gathers the value of “1” and assigns them into the middle of the array. This middle network array is represented by the coordinate range between (2, 2) to (7, 7). This module is separated into two cases which are the *initZero* case and the *assignMiddle* case as shown in the flowchart of Figure 3.3.1. The *initZero* case initializes the network to the value of “0“ so that there will not be any unknown

Case for *initZero*

Initialize network array with value "0"

Initialize network array with value "1" in the range of (2,2) to (7,7)

Case for *assignInMiddle*

Default Case: *initZero*

Figure 3.3.1: Flow Chart of the Neuron Initialization Block

value remaining in the network array. For the *assignMiddle* case, however, the value “1” is being initialized in the network array between the coordinate ranges of (2, 2) to(7,7), acting as a sample data. The partial RTL view for this network array initialization module is shown in Figure 3.3.2. On the other hand, a test bench code is designed to test this module. This module has only two inputs, i.e. the *CLOCK\_50* and the *iRST*. Therefore, the test bench is only required to instantiate the module and assign the the appropriate toggling rate to *CLOCK\_50* provide initialization values as shown in Figure 3.3.3.

C:\Users\Feng\Desktop\Final FYP Folder\result\init rtl.tif

Figure 3.3.2 Partial RTL View of the Neuron Initialization Block

C:\Users\Feng\Desktop\Final FYP Folder\result\init tb.tif

Figure 3.3.3: Partial Testbench Code for the Neuron Initialization Block

### 3.2 Hidden Layer Block

The Hidden Layer Block (Block 2) is a module to obtain the neuron position of *winner\_x* and *winner\_y* as shown in the HDL code fraction below and the RTL View of Figure 3.4.1. To obtain the position of the winner neuron as shown in the flow chart of Figure 3.4.2, the calculated *new\_distance* will be compared with the *prev\_distance*. If the *new\_distance* is smaller than the *prev\_distance*, the *prev\_distance* will be replaced by the *new\_distance* and the current *weight\_x* and current *weight\_y* will become the *winner\_x* and *winner\_y*.To calculate the *new\_distance*, the selected input for *position\_x* and *position\_y* are comparedto the*weight\_x* and *weight\_y* to find the greater *position\_x* and *position\_y*to subtract the smaller *position\_x* and *position\_y*.

The code fraction below shows the selected input for *position\_x* and *position\_y* being compared to the *weight\_x* and *weight\_y*. The smaller position value will be subtracted from the greater position value. After calculating the *position\_x* and *position\_y*, the same method is used to find the larger position value so that it can subract the smaller position value to obtain the *new\_distance*.

if(*selected\_x*>*weight\_x*)

*position\_x* = *selected\_x* - *weight\_x*;

else

*position\_x* = *weight\_x* - *selected\_x*;

if(*selected\_y*>*weight\_y*)

*position\_y* = *selected\_y* - *weight\_y*;

else

*position\_y* = *weight\_y* - *selected\_y*;

if(*position\_x*>*position\_y*)

*new\_distance* = *position\_x* - *position\_y*;

else

*new\_distance* = *position\_y* - *position\_x*;

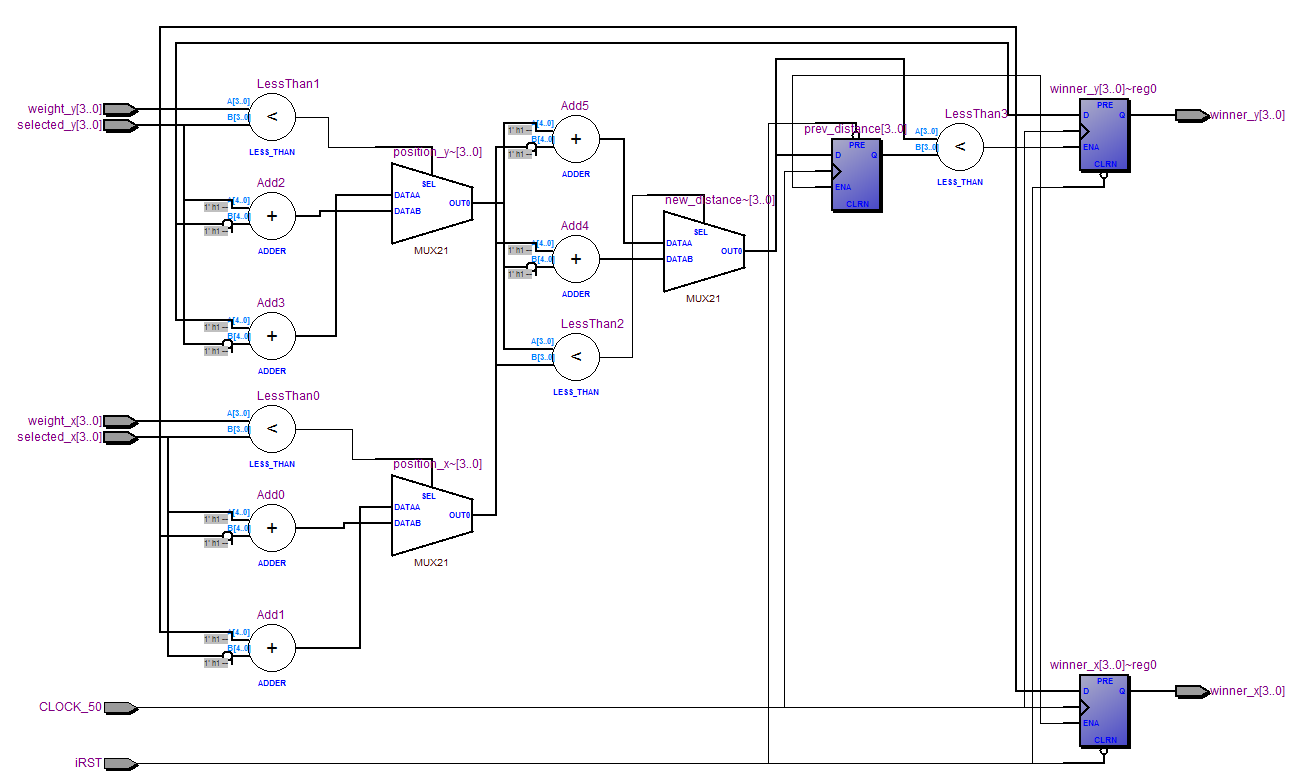


Figure 3.4.1 RTL View of the Hidden Layer Block

Is *selected\_x*>*weight\_x*?

Obtain *position\_x* value

Is *selected\_y*>*weight\_y*?

Obtain *position\_x* value

Is *position\_x*>*position\_y*?

Obtain *new\_distance* value

Are *new\_distance*<*prev\_distance*?

Update *prev\_distance*, *winner\_x* value and *winner\_y* value

Figure 3.4.2: Flow Chart of Hidden Layer Block

Once the *new\_distance* is calculated, the *prev\_distance* will be compared to the *new\_distance*. The *prev\_distance* has a default value of decimal 15 because it needs to be set as the largest distance to be compared upon. When the *new\_distance* is smaller than the *prev\_distance*, *prev\_distance* will be replaced by the *new\_distance* and the *winner\_x* and *winner\_y* will be replaced by *weight\_x* and *weight\_y*as shown in the following code fraction.

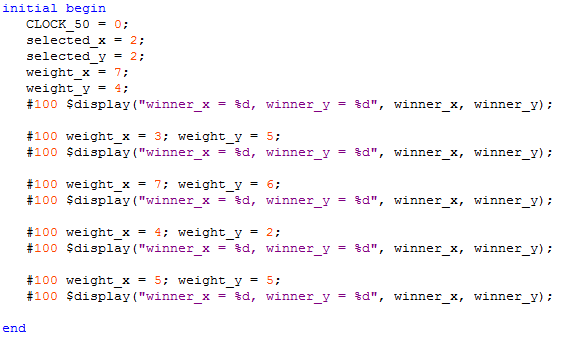
if(*new\_distanc*e<*prev\_distance*)begin

*prev\_distance* = *new\_distance*;

*winner\_x* = *weight\_x*;

*winner\_y* = *weight\_y*; end

The Hidden Layer Block testbench code is designed to ensure the Hidden Layer module and the flow chart shown in Figure 3.4.2 are functionally correct. The RTL view of this Hidden Layer Block is as illustrated in Figure 3.4.1. The test bench code is shown in Figure 3.4.3 with a default value of 2 for *selected\_x* and *selected\_y* and a default value of 7 and 4 for *weight\_x* and *weight\_y*. After one clock cycle, *weight\_x* and *weight\_y* will be assigned to values 3 and 5 and subsequently with the values of 7 and 6, 4 and 2, and lastly with values 5 and 5.

Figure 3.4.3: Hidden Layer Block Testbench Code

### 3.3 Weight Optimization Block

Weight Optimization Block (Block 3) is a module that produces an*updatedNetwork* array with the winner neuron and neighbour neuron updated as shown in the flow chart of Figure 3.5.1. There is an input signal named *update*. It is the key of this module because if this *update* signal is not triggered then this module will not be entered. This module is mainly divided into three cases namely the*calculateNewWeight* case, *updateWeight*case and lastly the *radiusForNeighbor* case.

*if(start == 1)begin*

*for(k = 0; k < 10; k = k + 1)begin*

*updatedNetwork[k][0] = network[k][0];*

*updatedNetwork[k][1] = network[k][1];*

*updatedNetwork[k][2] = network[k][2];*

*updatedNetwork[k][3] = network[k][3];*

*updatedNetwork[k][4] = network[k][4];*

*updatedNetwork[k][5] = network[k][5];*

*updatedNetwork[k][6] = network[k][6];*

*updatedNetwork[k][7] = network[k][7];*

*updatedNetwork[k][8] = network[k][8]; updatedNetwork[k][9] = network[k][9];end end*

Before going into the details for each case, there is an input to overwrite the *updatedNetwork* to the network array from the Neuron Initialization Block (Block 1) in the code fraction above. It is controlled by an input signal named *start*, if the *start* is equal to one then it will copy the network array into *updatedNetwork* array.

The *calculateNewWeight* is the first case used to calculate the value for the *new\_weight\_x* and *new\_weight\_y*.The formula used to calculate the new\_weight is given by the expression in (3.1). There are 4 array registers that is used to temporarily being used to store the value of *old\_weight\_x*, *old\_weight\_y*, *new\_weight\_x* and *new\_weight\_y*. Each of these array register is declared with 25 slots and 4 bits of data.

 (3.1)

Update *updatedNetwork*

Case for *calculateNewWeight*

Calculate *new\_weight\_x* and *new\_weight\_y*

update neuron weight and neighbor weight

Case for

*updateWeight*

Case for *radiusForNeighbor*

update *radiusForNeighborhood*

Default Case:

*calculateNewWeight*

Figure 3.5.1: Flow Chart of Weight Optimization Block

*calculateNewWeight: begin*

*if(up\_winner == 1)begin*

*tempForOldWeight\_x[num] = old\_weight\_x;*

*tempForOldWeight\_y[num] = old\_weight\_y;*

*update\_x = learning\_rate \* (selected\_x - old\_weight\_x);*

*update\_y = learning\_rate \* (selected\_y - old\_weight\_y);*

*new\_weight\_x = old\_weight\_x + update\_x;*

*new\_weight\_y = old\_weight\_y + update\_y;*

*tempForNewWeight\_x[num] = new\_weight\_x;*

*tempForNewWeight\_y[num] = new\_weight\_y;*

*num = num + 1;*

*end*

*state = updateWeight;*

*end*

The Verilog HDL code fraction as shown above implements the formula as expressed in (3.1). The *update\_x* and *update\_y* represent the backendportion of the formula and the *new\_weight\_x* and *new\_weight\_y* represent the entire formula.

The the second case updates the neuron weight and neighbor weight into the *updatedNetwork*. Whenever there is a value of "1" detected in the *updatedNetwork*, it will reuse the technique in the Hidden Layer Block (Block 2) to find the distance through all the neuron in the *updatedNetwork*. This distance will be compared to the neighbourhood and by using the formula in expressed in (3.1) to calculate the new weight for the neighbor. After calculating the new weight for the neighbor it will then store the current neuron weight and current neighbor into the temporary array register. Lastly, when all neuron are completely scanned it will then update the neuron in the *updatedNetwork*. The old weight will be replaced by the value "0" and the new weight will be replaced by value "1" as shown in the code fraction below.

*for(l = 0; l < 25; l = l + 1)begin*

*updatedNetwork [tempForOldWeight\_x[l]][tempForOldWeight\_y[l]] = 0;*

*updatedNetwork [tempForNewWeight\_x[l]][tempForNewWeight\_y[l]] = 1;*

*end*

The third case of *radiusForNeighborhood* updates the radius of neighborhood and the learning rate. The neighborhood radius is a decaying exponential function that reduces the neighborhood radius by one every time it completely updates the neurons as shown in the expression given by (3.2). Due to the difficulty and complexity of Verilog HDL to handle floating points, the *neighborhood radius*can be simplified to one-fourth of the network array, so the default value is 25 as expressed in (3.3).

After investigation with a few samples, it has been found that the *neighborhood radius*can be reduced to an optimum value of 10 because the *updatedNetwork* array will no longer be updated below the radius of 10. The clock cycle to completely update the neuron is 296, so, neighborhood radius will be reduced in every 296 clock cycles. When the clock cycle reaches 2960 then the neighborhood radius will become 0. As the learning rate is also a decaying exponential function that produces floating point values as was indicated in (3.3). The value of the learning rate ranges approximately from 0.3 to 0.1. In order to simplify the floating point values, the values will be set from 1 to 0 when the clock cycle is in hexadecimal 16'hffff. The partial RTL View of the Weight Optimization Block is as shown in Figure 3.5.2. It is part of the formula indicated by the expression given in (3.1) previously.

C:\Users\Feng\Desktop\Final FYP Folder\weight\learning rate formula.tif (3.2)

(3.3)

C:\Users\Feng\Desktop\Final FYP Folder\weight\rtl.tif

Figure 3.5.2: RTL View of the Weight Optimization Block

A HDL based testbench was designed to verify the functionality of this Weight Optimization Block. The test bench code shown in Figure 3.5.3 has a default value of "0" and "1" for *selected\_x* and *selected\_y* and a default value of 2 for *old\_weight\_x* and *old\_weight\_y*. The *update* signal is being triggered to "1" so that it can enter this block.

**C:\Users\Feng\Desktop\Final FYP Folder\weight\tb.tif**

Figure 3.5.3: Testbench Code for the Weight Optimization Block

### 3.4 Iteration-Check Block

Iteration-Check Block (Block 4) is a module used to check the *currentNetwork* array against the *previousNetwork* array in the register as indicated by the flow chart in Figure 3.6.1. An input named *resetIteration* is a signal to reset the iteration and flag to zero, so that it will restart the iteration from 0 all over again. The *previousNetwork*array will be replaced by the *currentNetwork*array after the checks between the *previousNetwork*array and the *currentNetwork* array are completed.

Is *resetIteration* equal to 1?

Compare *networkToCompare* with *previousNetwork*

Update *previousNetwork* data from *networkToCompare*

Is *flag* signal equal 1?

Update *iteration*

Is *iteration*> 10000?

Output *success* == 1

Figure 3.6.1: Flow Chart of the Iteration-Check Block

*for(i = 0; i < 10; i = i + 1)begin*

*for(j = 0; j < 10; j = j + 1)begin*

*if(currentNetwork[i][j] != previousNetwork[i][j])*

*flag = 1;*

*previousNetwork[i][j] = currentNetwork[i][j];*

*end*

*end*

The code fraction above shows that if there is a difference while checking, the *flag* signal will be triggered to 1. Also, the *currentNetwork* array will be copied into the *previousNetwork* array, so that the next *currentNetwork* array can be used to check against the old *currentNetwork* array that was copied into the *previousNetwork* array.

*if(flag == 1)begin*

*iteration = 0;*

*flag = 0;*

*end*

*else begin*

*iteration = iteration + 1;*

*if(iteration >= 10000)*

*success = 1;*

*end*

Once the *flag* is triggered to 1 in the code fraction above, it will reset the *iteration* to 0 and trigger the *flag* back to 0. Otherwise it will increase the *iteration* by adding 1 to it and if the iteration is greater than 10000,this means that the *currentNetwork*array is completely updated in the Weight Optimization Block (Block 3) which will then trigger an output *success* signal of "1". The iteration value of 10000 is chosen as a randomly large value because this module will be comparing the *currentNetwork* array from the Weight Optimization Block (Block 3) with the *previousNetwork*array in this module. The *currentNetwork*array will get updated with different number of iterations, so the iteration value has to be set as a large value.

Due to the for loop usage in Verilog code, the RTL view of this block is too large that to be seen in full. Therefore, partial RTL of the Iteration-Check Block for the *previousNetwork* array and *currentNetwork* array is as indicated in Figure 3.6.2.

C:\Users\Feng\Desktop\Final FYP Folder\result\iteration rtl.tif

Figure 3.6.2: Partial RTL View of the Iteration Check Block

To ensure the code fractions above are functionally accurate, a proper test bench code as shown in Figure 3.6.3 was written for verification purposes. The *currentNetwork* array is assigned with a default value of "1" to be filled into the array. A zero was assigned to the *currentNetwork*array at different time delays of #100, #300 and #500 respectively, so that the iteration will reset to zero.

**C:\Users\Feng\Desktop\Final FYP Folder\Flow chart\tb.tif**

Figure 3.6.3: Testbench Code for the Iteration-Check Module

### 3.5 Voting System Block

The Voting System Block (Block 5) is a module used to obtain the *votedNetwork* array from three different candidate network arrays to begin voting as shown in the flow chart of Figure 3.7.1. The voting system is being made up of three candidatesc ompeting to win a position. The data array is represented as the position, so this voting system will have to wait for all three candidate seats to be fully taken to start the vote. The code fraction below shows the *candidatesNetwork* array and the data array being copied into the *tempCandidatesNetwork* and *dataIMC* respectively using a for loop. This code fraction indicates that Candidates 1, 2 and 3 will be subsequently copied one after another. There are

C:\Users\Feng\Desktop\Final FYP Folder\som\new.tif

Figure 3.7.1: Flow Chart of Voting System Block

three signals name *slot\_1*, *slot\_2* and *slot\_3* to indicate that the *candidatesNetwork* is ready to be copied into the *tempCandidatesNetwork*.

*candidate\_1: begin*

*if(slot\_1 == 1)begin*

*for(i = 4'b0000; i < 4'b1010; i = i + 4'b0001)begin*

*for(j = 4'b0000; j < 4'b1010; j = j + 4'b0001)begin*

*tempCandidatesNetwork[0][i][j] = candidatesNetwork[i][j];*

*dataIMC[i][j] = data[i][j];*

*end*

*end*

*state = candidate\_2;*

*end*

*end*

After that, only one out of the three candidates will become the winner and take over the seat position. Every candidate array has its individual counter to keep track of the number of matched elements compared to the data array. The candidate with most matches indicated by the highest count value will become the winner of the voting system and take the *votedNetwork* array seat. A signal called *successVote* will be triggered when the voting is completed. The On Chip Training Block (Block 6) will then be informed that the *votedNetwork*is ready to be recognized with the user arraybeing stored in the database.

Figure 3.7.2 shows the partial RTL View of the Voting System Block where the *winnerCandidateNetwork* is being outputted to the On-Chip Training Block (Block 6).

C:\Users\Feng\Desktop\Final FYP Folder\result\rtl vote 1.tif

Figure 3.7.2: RTL view of Voting System Block

In the test bench code as shown as in Figure 3.7.3, three different network arrays are ready to be fed into the candidate network array to start the vote. The three candidate network arrays are similar network with some minor differences as indicated in Figure 3.7.4, so that it can be tested more precisely.

C:\Users\Feng\Desktop\Final FYP Folder\result\tb vote.tif

Figure 3.7.3: Testbench Code for the Voting System Block

C:\Users\Feng\Desktop\Final FYP Folder\voting\candidate.tif

Figure 3.7.4: Candidate Array Input Vectors for Voting System Block Testbench

### 3.6 On-Chip Training Block

The On-Chip Training Block (Block 6) is a module that is divided into two separate cases which is firstly to allow user to store user data into the iris database and secondly to perform recognition on the data that has been stored into the iris database as indicated by the flow chart in Figure 3.8.1. The iris database allows user to store up to three user's data with each user being captured three times to increase the recognition accuracy. The *startTraining* signal is the key for this module because this signal needs to be triggered to 1 to begin storing data into the iris database and to recognize users. Otherwise if the *startTraining* is at "0" then it will not be able to perform any task in this module. The first case is named as the"*save*" case, if the *trainUser* signal is triggered as 1 then it will check for the *storeControl*

**C:\Users\Feng\Desktop\Final FYP Folder\som\onchip.tif**

Figure 3.8.1 Flow Chart of On-Chip Training Block

whether it has been triggered as "1" or not. If the *storeControl* is "1" then the *storeUser* will represent the user slots in the database. Once the *storeUser* completes its selection then it will store the user data into the database. Also, while the *trainUser* is triggered as "0" then it will go to the next case which is to compare. The following shows the code fraction for the case "*save*".

*if(startTraining == 1)begin*

*case(state)*

*save: begin*

*if(trainUser == 1)begin*

*if(storeControl == 1)begin*

*case(storeUser)*

*3'b001: user\_1 = data;*

*3'b010: user\_1\_2 = data;*

*3'b011: user\_1\_3 = data;*

*3'b001: user\_2 = data;*

*3'b010: user\_2\_2 = data;*

*3'b011: user\_2\_3 = data;*

*3'b001: user\_3 = data;*

*3'b010: user\_3\_2 = data;*

*3'b011: user\_3\_3 = data;*

*endcase*

*end*

*end*

*else*

*state = compare;*

*end*

The second case compares the *new\_network* from the Weight Optimization Block (Block 3) that has already completed the iteration check at the Iteration-Check Block (Block 4).

*if(start\_compare == 1)begin*

*for (i = 0; i < 10; i = i + 1) begin*

*for(j = 0; j < 10; j = j + 1) begin*

*if(new\_network[i][j] == user\_1[i][j])*

*neuronCounter\_1 = neuronCounter\_1 + 1;*

*if(new\_network[i][j] == user\_1\_2[i][j])*

*neuronCounter\_2 = neuronCounter\_2 + 1;*

*if(new\_network[i][j] == user\_1\_3[i][j])*

*neuronCounter\_3 = neuronCounter\_3 + 1;*

*if(new\_network[i][j] == user\_2[i][j])*

*neuronCounter\_4 = neuronCounter\_4 + 1;*

*if(new\_network[i][j] == user\_2\_2[i][j])*

*neuronCounter\_5 = neuronCounter\_5 + 1;*

*if(new\_network[i][j] == user\_2\_3[i][j])*

*neuronCounter\_6 = neuronCounter\_6 + 1;*

*if(new\_network[i][j] == user\_3[i][j])*

*neuronCounter\_7 = neuronCounter\_7 + 1;*

*if(new\_network[i][j] == user\_3\_2[i][j])*

*neuronCounter\_8 = neuronCounter\_8 + 1;*

*if(new\_network[i][j] == user\_3\_3[i][j])*

*neuronCounter\_9 = neuronCounter\_9 + 1;*

*end*

*end*

*outNow = 1;*

*end*

For the code fraction above, when the *start\_compare* signalis triggered then it will compare the *new\_network* with the database and place it into nine counters. After that all counters will compare with one another to find the user that matches with the iris database. For example, if the result between user 3 and the iris database matches one of the user 3 patterns then it will be lighted as4'b0011 on the LEDR[17:14]on the DE2 Cyclone II FPGA board else it will be lighted as 4'b1000 representing that the user does not match any users in the database.

In the testbench, nine different data from three users has been stored into the database as indicated by the testbench code in Figure 3.8.2. After all user data have been completely stored in the database, the *trainUser* signal will then be triggered to "0" so that it can go into the "*compare*" case. After the matching process with the database is completed, the assigned LEDRs will light up to indicate a matched or unmatched outcome.

**C:\Users\Feng\Desktop\Final FYP Folder\result\onchip tb.tif**

Figure 3.8.2: Testbench Code for the On-Chip Training Block

### 3.7 Integrated SOM Module

The SOM module is the main module that integrates all six sub-blocks to perform the complete SOM algorithm as indicated in the block diagram shown in Figure 3.1. There is a flag named*flagForUpdateSelected* which is used to obtain the value of *selected\_x* and *selected\_y* when it is triggered as "1". When the *flagForUpdateSelected* is triggered as "1" then the *selected\_x* and *selected\_y* from the data will be updated as shown in the code fraction below. While the *flagForUpdateSelected* signal is triggered as "0", then the *weight\_x*

*if(flagForUpdateSelected == 1)begin*

*if(data[count\_i][count\_j] == 1)begin*

*selected\_x = count\_i;*

*selected\_y = count\_j;*

*end*

*flagForUpdateSelected = 0;*

*end*

and *weight\_y*values will be obtained from the *initNetwork*when the *update* signal is equal to "0", otherwise it will obtain the value from *updatedNetwork* as shown in the code fraction below.

*if(update == 1)begin*

*if(updatedNetwork[i][j] == 1)begin*

*weight\_x = i;*

*weight\_y = j;*

*end*

*end*

*else begin*

*if(initNetwork[i][j] == 1)begin*

*weight\_x = i;*

*weight\_y = j;*

*end*

*end*

Once the value of *i* is detected as 10, then the *count\_i* and *count\_j* values will be incremented in order to change the value of *selected\_x* and *selected\_y* because the *flagForUpdateSelected* will be triggered as "1" as shown in the code fraction below.

*if(i == 4'b1010)begin*

*count\_j = count\_j + 1;*

*if(count\_j == 4'b1010)begin*

*count\_i = count\_i + 1;*

*if(count\_i == 4'b1010)begin*

*count\_i = 4'b0000;*

*end*

*count\_j = 4'b0000;*

*end*

*i = 4'b0000;*

*j = 4'b0000;*

*update = 1;*

*up\_winner = 1;*

*weight\_selected\_x = selected\_x;*

*weight\_selected\_y = selected\_y;*

*old\_weight\_x = winner\_x;*

*old\_weight\_y = winner\_y;*

*flagForUpdateSelected = 1;*

*end*

Theoretically, selecting a different starting value of the SOM algorithm will produce a different output array. To fulfil the requirement of the Voting System Block, it needs to receive three *updatedNetwork* arrays to start the vote. The first *updatedNetwork* array will begin at the coordinate point of (0, 0) while the second *updatedNetwork* array will begin at (0, 9). Lastly, the third *updatedNetwork* array will start at coordinate point of (9, 9). There are two flag signal namely *secondround* and *thirdRound*. When the *secondRound* and *thirdRound* are equal to zero then it will start at point (0, 0). While the *secondRound* is triggered as "1" which means the first *updatedNetwork*has been received by the Voting System Block then it will move to the (0, 9) starting point and perform the same tasks for the second *updatedNetwork* array and subsequently the similar procedures holds for the third *updatedNetwork*array.

## Chapter 4 Results and Discussions

The result of IRS system will be presented in three main sections, firstly as a block by block basis, secondly as a fully integrated SOM subsystem and finally as a fully integrated IRS system. Several parameters will be compared upon such as the resource usage, speed and accuracy. The results of each block will mainly be shown in a Console Display and waveform. Also, the problem faced will be discussed. The following sub-sections describe in detail the results and discussions of each SOM sub-blocks and as a fully integrated IRS system.

### 4.1 Neuron Initialization Block

The result as indicated in Figure 4.1.1 shows that the value of "1" is being nicely initialized in the middle coordinate range from (2, 2) to (7, 7) of the *initNetwork* array. Table 4.1 shows that the power dissipation of this block is 121.75 mW. Also, the LUT usage had used one 4 input function and one ≤ 2 input function. Other than that, the board usage used up only 3 logic elements which is lesser than 1%.

**C:\Users\Feng\Desktop\Final FYP Folder\neuronInit\array.tif**

Figure 4.1.1: Result of *initNetwork* Array after initialization

Table 4.1: Neuron Initialization Block Power Dissipation and Resource Usage

|  |  |
| --- | --- |
| **Specifications** | **Values** |
| Power Dissipation | 121.75mW |
| LUT Usage | -4 input function (1)  -<= 2 input function (1) |
| Board Usage | 3 (< 1%) |

Figure 4.1.2 shows the setup slack between the Data Arrival Path and Data Required Path. The clock path of Data Required Path is 2.676 ns while Data Arrival Path is 3.676 ns. The result of setup slack waveform is as shown in Figure 4.1.3 indicating a data delay time of 1.055 ns.

**C:\Users\Feng\Desktop\Final FYP Folder\neuronInit\data path.tif**

Figure 4.1.2: Timing Analysis of Neuron Initialization Block's Datapath

**C:\Users\Feng\Desktop\Final FYP Folder\neuronInit\setup slack.tif**

Figure 4.1.3: Timing Waveform of the Neuron Initialization Block

### 4.2 Hidden Layer Block

The result of this method reduces complexity while maintains the accuracy of conventional method when comparing with the Euclidean distance method. Table 4.2.1shows the differences by using 3 different methods. Method 1 is the Euclidean distance formula and Method 2 is the Yap and Lim (2014) method. The result of Method 2 has lower accuracy at 13.33- 20% as compared to the conventional Euclidean Method (Method 1). The proposed method in this SOM design is capable of achieving 100% accuracy as compared to Method 1.

The waveform results as shown in Figure 4.2.1to Figure 4.2.6shows the values of *winner\_x* and *winner\_y* after calculating with the value 2 for *selected\_x* and *selected\_y*with different values of *weight\_x* and *weight\_y*. After the *new\_distance* was calculated, it was found to be lesser than the default distance which is decimal 15, so that it will update the *prev\_distance* with the value of the *new\_distance* and also output the *winner\_x* and *winner\_y*as shown in Figure 4.2.1. These steps were repeated until Figure 4.2.6. However, Figure 4.2.4 shows that there were no updates for the *winner\_x* and *winner\_y* because the *new\_distance* is not lesser than *prev\_distance*, so it will not update the *winner\_x* and *winner\_y*values. The *weight\_x* and *weight\_y* with values of 5 shown in Figure 4.2.5 has became the new updatedfor *selected\_x* and *selected\_y*. Figure 4.2.6 has no update because the *new\_distance* was the lowest among all, thus the final *winner\_x* and *winner\_y* values were 5.

Table 4.2.1: Accuracy Comparison of Three Different Techniques used in the Hidden Layer Block

C:\Users\Feng\Desktop\Final FYP Folder\result\compare table.tif

C:\Users\Feng\Desktop\Final FYP Folder\hidden\result 1.tif

Figure 4.2.1: Value of *winner\_x* and *winner\_y* when*weight\_x*=7 and *weight\_y*=4

**C:\Users\Feng\Desktop\Final FYP Folder\hidden\result 2.tif**

Figure 4.2.2: Value of *winner\_x* and *winner\_y* when*weight\_x*=3 and *weight\_y*=5

**C:\Users\Feng\Desktop\Final FYP Folder\hidden\result 3.tif**

Figure 4.2.3: Value of *winner\_x* and *winner\_y*when*weight\_x*=7 and *weight\_y*=6

**C:\Users\Feng\Desktop\Final FYP Folder\hidden\result 4.tif**

Figure 4.2.4: Value of *winner\_x* and *winner\_y*when*weight*=7 and *weight\_y*=6

**C:\Users\Feng\Desktop\Final FYP Folder\hidden\result 5.tif**

Figure 4.2.5: Value of *winner\_x* and *winner\_y*when*weight\_x*=5 and *weight\_y*=5

**C:\Users\Feng\Desktop\Final FYP Folder\hidden\result 6.tif**

Figure 4.2.6: Value of *winner\_x* and *winner\_y* when*weight\_x*=5 and *weight\_y*=5

Table 4.2.2 shows that the power dissipation for this block is 112.89mW and the LUT usages are eight units 4 input functions, 36 units 3 input functions and 6 units ≤2 input functions.

Table 4.2.2: Resource Usage and Power Dissipation of Hidden Layer Block

|  |  |
| --- | --- |
| **Specifications** | **Values** |
| Power Dissipation | 112.89mW |
| LUT Usage | -4 input function (8)  -3 input function (36)  -≤ 2 input function (6) |
| Board Usage | 59 (< 1%) |

In the Figure 4.2.7 shows the setup slack for Data Arrival Path and Data Required Path. The clock path of Data Required Path is 2.687ns and Data Arrival Path is 3.687ns. The result of the setup slack waveform in Figure 4.2.8 shows the data delay is 2.308ns.

**C:\Users\Feng\Desktop\Final FYP Folder\hidden\data path.tif**Figure 4.5.7: Timing Analysis for the Datapath of the Hidden Layer Block

**C:\Users\Feng\Desktop\Final FYP Folder\hidden\setup slack wave.tif**Figure 4.2.8: Timing Waveform of the Hidden Layer Block

### 4.3 Weight Optimization

The result in Figure 4.3.1 shows the *updatedNetwork* array before the neuron weight and neighbor weight gets updated. Before the neuron gets update, the selected neuron weight at location (0, 1) and the neighbor weight at location (0, 2) is "0". After the neuron gets updated, the winner of the weight will be at location (2, 2) because it is preset to be tested. The neuron weight at (0, 1) and neighbor weight at (0, 2) will then be updated to the value of "1" as indicated in Figure 4.3.2. Other than that, the neuron weight at (2, 2) and neighbor weight at (2, 3) becomes "0". This indicates that the neurons were successfully updated.

**C:\Users\Feng\Desktop\Final FYP Folder\weight\before.tif**

Figure 4.3.1: Result of Weight Optimization before Update

**C:\Users\Feng\Desktop\Final FYP Folder\weight\after.tif**

Figure 4.3.2: Result of Weight Optimization after Update

In the Table 4.3 shows the comparison of usage for Conventional and SOM. The improvement on the Conventional is to reduce the board usage from 8281 logic elements to 7043 logic elements that save 1238 logic elements. Other than that, the total pins used are reduced from 225 to 220. Unfortunately, the power dissipation increase 0.03mW from 135.95mW to 135.98mW. For the LUT usage, the 4 input function for Conventional is 5888 units and 3 input function is 1539 units and lastly for the ≤2 input function is 758. In SOM, the 4 input function reduced to 5435 units, 3 input function reduced to 1443 units and the ≤ 2 input function reduced to 439 units. The speed is slower compare to the Conventional because SOM used more clock than the Conventional. Also, the area of this block is lesser than the conventional method.

Table 4.3: Specification Comparison of Proposed and Conventional

Weight Optimization Method

|  |  |  |
| --- | --- | --- |
| **Specifications** | **Conventional Weight Optimization** | **Proposed SOM Weight Optimization** |
| Board Usage | 8281 | 7043 |
| Total Pins | 225 | 220 |
| Power Dissipation | 135.95mW | 135.98mW |
| LUT Usage | - 4 input function (5888)  - 3 input function (1539)  -≤ 2 input function (758) | - 4 input function (5435)  - 3 input function (1443)  -≤ 2 input function (439) |
| Speed | Fast | Slow |
| Area | More | Less |

The setup timing slack for the Data Arrival Path and the Data Required Path for the SOM are as shown in Figure 4.3.3. The clock path of Data Required Path is 2.692ns and the Data Arrival Path is 3.691ns. Other than that, the setup slack timing for Data Arrival Path and Data Required Path for the Conventional method is 2.695ns and 3.647ns respectively as shown in Figure 4.3.4. The result of SOM setup slack waveform as shown in Figure 4.3.5 shows the data delay as17.74ns whereas the Conventional method in Figure 4.3.6 indicates a slower value of 21.661ns.

C:\Users\Feng\Desktop\Final FYP Folder\weight\data path.tif

Figure 4.3.3: Timing Analysis for Datapath of Proposed Weight Optimization Block

**C:\Users\Feng\Desktop\Final FYP Folder\weight\senior data path.tif**

Figure 4.3.4: Timing Analysis for Datapath of Conventional Weight Optimization Block

**C:\Users\Feng\Desktop\Final FYP Folder\weight\wave.tif**

Figure 4.3.5: Timing Waveform of Proposed SOM Weight Optimization Block

**C:\Users\Feng\Desktop\Final FYP Folder\weight\senior wave.tif**

Figure 4.3.6: Timing Waveform of Conventional Weight Optimization Block

### 4.4 Iteration-Check Block

From the result it has been found that the *success* signal was triggered as "1" when the *currentNetwork* and *previousNetwork*were the same for 10000 iterations. The value of 10000 for the iteration was found from the result in the console because the iterations run up to 94789 as indicated in Figure 4.4.1 and it also did not receive s different *currentNetwork*value from the Weight Optimization Block(Block 3), this means that this amount of iteration is not required. Due to hardware limitations, after verifications with different types of data array, the largest value for this iteration to trigger the success signal is 7424 as shown as in Figure 4.4.2. To ensure that the iteration is completed, the iterations value was decided to be set at 10000.

C:\Users\Feng\Desktop\Final FYP Folder\iteration\console 2.tif

Figure 4.4.1: Console Display of iteration more than the value of 10000

C:\Users\Feng\Desktop\Final FYP Folder\iteration\console 1.tif

Figure 4.4.2: Console Display of iteration less than the value of 10000

Table 4.4 shows that the power dissipation in this block is 122.02mW and the usage for LUT are 77 units 4 input functions, 31 units 3 input functions and 34 units ≤2 input functions. For the board usage, it used up a total of 193 logic elements which is less than 1% of the board's total resources.

Table 4.4: Power Dissipation and Resource Usages of the Iteration Check Block

|  |  |
| --- | --- |
| **Specifications** | **Values** |
| Power Dissipation | 122.02mW |
| LUT Usage | -4 input function (77)  -3 input function (31)  -≤ 2 input function (34) |
| Board Usage | 193 (< 1%) |

The setup timing slack for Data Arrival Path and Data Required Path are as shown in Figure 4.4.3. The clock path of the Data Required Path is 2.660ns and the Data Arrival Path is 3.666ns. The result of setup slack waveform in Figure 4.4.4 shows that the data delay is 5.705ns.

C:\Users\Feng\Desktop\Final FYP Folder\iteration\data path.tifFigure 4.4.3: Timing Analysis for Datapath of the Iteration-Check Block

**C:\Users\Feng\Desktop\Final FYP Folder\iteration\wave slack.tif**

Figure 4.4.4: Timing Waveform of the Iteration-Check Block

### 4.5 Voting System Block

The result shown in Figure 4.5.1 indicates the winner network array after going through the voting system. This candidate network array had the highest percentage of similarity when compared to the data array among the other 2 candidatesand thus, became the winner. The candidate 1 had a similarity of 92%, while candidate 2 had 83% similarity and lastly candidate 3 had 80% similarity as shown in the console display and chart of Figure 4.5.2 and Figure 4.5.3 respectively. The result clearly shows that the winner is candidate 1.

C:\Users\Feng\Desktop\Final FYP Folder\voting\result.tif

Figure 4.5.1: Result of *votedNetwork* Array after a Successful Voting

C:\Users\Feng\Desktop\Final FYP Folder\voting\console.tif

Figure 4.5.2: Console Display of Candidate's Counter

Figure 4.5.3: Chart indicating Matching Percentage of Each Candidate

By referring to the Table 4.5, the power dissipation of this block is estimated as 146.08mW. In LUT usage, the highest input function units used is ≤2 input functions. The usage for ≤2 input functions is 2111. The second highest user input function is the 3 input functions that used up to 360 units. Lastly, the units for 4 input functions is 112. The board usage is 2814 logic elements that took up 8% of the total board capacity.

Table 4.5: Power Dissipation and Resource Usages of the Voting System Block

|  |  |
| --- | --- |
| **Specifications** | **Values** |
| Power Dissipation | 146.08mW |
| LUT Usage | -4 input function (112)  -3 input function (360)  -≤ 2 input function (2111) |
| Board Usage | 2814 (8%) |

The setup timing slack for the Data Arrival Path and Data Required Path are as shown in Figure 4.5.4. The clock path of the Data Required Path is 2.680ns while the Data Arrival Path is 3.680ns. The result of the setup slack waveform in Figure 4.5.5 shows that the data delay is 94.271ns.

C:\Users\Feng\Desktop\Final FYP Folder\voting\data path.tif

Figure 4.5.4: Timing Analysis for the Datapath of Voting System Block

C:\Users\Feng\Desktop\Final FYP Folder\voting\wave slack.tif

Figure 4.5.5: Timing Waveform of the Voting System Block

### 4.6 On Chip Training Block

Figure 4.6.1 shows that the *votedNetwork*from Voting System Block (Block 5) has been successfully recognized/ matchedto the database when compared to the values of the *userID*. The *userID* shows the value of 4'b0010 meaning that the User 2 has been successfully recognized. Figure 4.6.2 shows the Console Display with 81 neuronsbeing successfully matchedbetween the database and the user. Other than that, there is also a failed result to be discussed as shown in Figure 4.6.3 and Figure 4.6.4. The *userID* shows the value of 4'b1000 as shown in Figure 4.6.3 indicating that the *votedNetwork*does not match with the database. Also, in the Figure 4.6.4 shows the Console Display of an unrecognized user indicated with an X. The actual hardware implementation for the final user recognition results are displayed on the LEDs of the DE2 FGGA board.

**C:\Users\Feng\Desktop\Final FYP Folder\onchip\som result.tif**

Figure 4.6.1: Result of On-Chip Training Block with Successfully Recognition

**C:\Users\Feng\Desktop\Final FYP Folder\onchip\som console.tif**

Figure 4.6.2: Console Display of the On-Chip Training Block with Successful Recognition

**C:\Users\Feng\Desktop\Final FYP Folder\onchip\som fail resulttt.tif**

Figure 4.6.3: Result of the On-Chip Training Block with Failed Recognition

**C:\Users\Feng\Desktop\Final FYP Folder\onchip\som fail result.tif**

Figure 4.6.4: Console Display of the On-Chip Training Block with Failed Recognition

Table 4.6 shows the comparison of resource usage and power dissipation between the conventional and proposed SOM implementation for the IRS. The improvement made on the Conventional design is the reduction of board usage from 6871 logic elements to 4025 logic elements which saves up to 2846 (~30%/ one third) logic elements. Also, the total pins used reduced from 225 to 215. Other than that, the power dissipation reduced from 136.57mW to 135.71mW, reducing around 1mW of power.In terms of LUT usage, the 4 input functionsused in the Conventional design is 200 units while 3 input functions is 263 units but for the proposed SOM, this number is 1622 and 1375 respectively. This result is due to the ≤ 2 input functions has been used up to 6404 units in the conventional design and while the proposed SOM uses only 1028 units. Lastly, the database for the users from the Conventional is two. The proposed SOM database had been increased to three users with three different variations being stored in the database. Figure 4.6.7 shows the chart to summarize comparison metrics between conventional and proposed SOM implementations for the IRS.

Table 4.6: Comparison of Resource Usage and Power Dissipations for

Conventional and Proposed SOM Implementations

|  |  |  |
| --- | --- | --- |
| **Specification** | **Conventional On-Chip Training Method** | **Proposed SOM On-Chip Training Method** |
| Board Usage | 6871 | 4025 |
| Total Pins | 225 | 215 |
| Power Dissipation | 136.57mW | 135.71mW |
| LUT Usage | - 4 input function (200)  - 3 input function (263)  -≤ 2 input function (6404) | - 4 input function (1622)  - 3 input function (1375)  -≤ 2 input function (1028) |
| Database Slot | Two | Three |

Figure 4.6.7: Comparison Chart for the Conventional and Proposed SOM

Implementation for the IRS

The setup timing slack for the Data Arrival Path and Data Required Path for the SOM are as shown in Figure 4.6.8. The clock path of the Data Required Path is 2.679ns and Data Arrival Path is 3.679ns. Other than that, the setup timing slack for the Data Arrival Path and Data Required Path for the Conventional design is 2.686ns and 3.656ns respectively as shown as in Figure 4.6.9.The result of the proposed On-Chip Training Block design however has a setup slack waveform as shown in Figure 4.6.10 indicating that the data delay is 7.442ns while the Conventional implementation is as shown in Figure 4.6.11 as 120.054ns.

**C:\Users\Feng\Desktop\Final FYP Folder\onchip\data path.tif**

Figure 4.6.8: Timing Analysis for the Datapath of the Proposed On-Chip Training Block

**C:\Users\Feng\Desktop\Final FYP Folder\onchip\senior data path.tif**

Figure 4.6.9: Timing Analysis for the Datapath of the Conventional On-Chip Training Block

**C:\Users\Feng\Desktop\Final FYP Folder\onchip\waveform.tif**

Figure 4.6.10: Timing Waveform of the Proposed On-Chip Training Module

**C:\Users\Feng\Desktop\Final FYP Folder\onchip\senior waveform.tif**

Figure 4.6.11: Timing Waveform of the Conventional On-Chip Training Module

### 4.7 The Integrated Proposed SOM Block

The data array of user 2 had been used for recognition in the result as shown in Figure 4.7.1. Here, it shows that the *updateNetwork* has been successfully matched to the database of user 2. The Console Display as indicated in Figure 4.7.2 shows that 88% of the neurons has been matched to the database. The *updatedNetwork* array had been correctly updated as shown in Figure 4.7.1.

**C:\Users\Feng\Desktop\Final FYP Folder\som\result matched.tif**

Figure 4.7.1: Result of Successfully Matched User

**C:\Users\Feng\Desktop\Final FYP Folder\som\match console.tif**

Figure 4.7.2: Console Display of a Successfully Matched User

Other than the matched results, an unmatched result is as shown as in Figure 4.7.3. The value of *userID* is 4'b1000 meaning that the user cannot be matched with any array pattern within the database because this particular user has not been trained into the database.

**C:\Users\Feng\Desktop\Final FYP Folder\som\not match.tif**

Figure 4.7.3: Result of an Unmatched User

Based on the comparison table shown in Table 4.7, the power dissipation of this block is 123.98mW. In terms of LUT usage, the highest input function units being used is the 4 input functions. The usage for the 4 input functions is 6183. The second highest user input function being used is ≤2 input functions that uses up to 5776 units. Lastly, the usage for ≤2 input function is 5776. The board usage used is 14589 logic elements which take up 44% of the entire board capacity.

Table 4.7: Power Dissipation and Resource Usage of the Proposed Integrated SOM Block

|  |  |
| --- | --- |
| **Specifications** | **Values** |
| Power Dissipation | 123.98mW |
| LUT Usage | -4 input function (6183)  -3 input function (2133)  -≤ 2 input function (5776) |
| Board Usage | 14589 (44%) |

The setup timing slack for the Data Arrival Path and the Data Required Path of the SOM Block are as shown as in Figure 4.7.4. The clock path of the Data Required Path is 2.680ns and the Data Arrival Path is 3.661ns. The result of the proposed integrated SOM block setup slack waveform is as shown in Figure 4.7.5 which shows that the data delay is 92.183ns.

C:\Users\Feng\Desktop\Final FYP Folder\som\data path.tif

Figure 4.7.4: Timing Analysis for the Datapath of the Integrated SOM Block

C:\Users\Feng\Desktop\Final FYP Folder\som\waveform.tif

Figure 4.7.5: Timing Waveform of the SOM Block

### 4.8 Full Integration

Table 4.8.1 shows the iris with 3 sample for each users that has been stored in the database. These figuresclearly show that there will be differences in the captured iris positioned even if the iris images were captured from the same user. The result in the Table 4.8.2 and Figure 4.8.1 shows the recognition accuracy of three users with each being tested 10 times. Table 4.8 shows that User\_1 can be successfully recognized 40% of the time with 100% accuracy while X indicates that 60% of the time not one user can be recognized. On the other hand, User\_2 has an accuracy of 80% while being able to be recognized 50% of the time. Finally, User\_3 also has a recognition accuracy of 100% albeit being recognized only 30% of the time out of a total of 10 recognition tests conducted. Overall the IRS system can recognize a user 40% of the time with an overall accuracy of 93.33%.

Table 4.8.1: Iris Sample of 3 Users

|  |  |  |  |
| --- | --- | --- | --- |
| **Samples** | **User\_1** | **User\_2** | **User\_3** |
| First Sample | https://lh4.googleusercontent.com/jel8rlGvQyzx_fsxGNuqWIlDSk6Ot8FOyh-2l8uex6L4Oflq-78kloWvPkwbd1JJYMcw4iCtokdHdF9XalN2kCvgmH-KNoJifskG12lQjPOMrUMkAmQvZte5YDT3G8ZVaFRv_NxgBg | https://lh3.googleusercontent.com/15fLffBUPAUdg8tdQbmqZluqKfvTd7IQ-GYYcPs7TGbkPeOUX95jer43B5vdQeSgqo9a7xI4v4OuzH64atuzV5gOQrF1Bk8J9NIQWIo7oXzc6q5cMIxvHowq5-yWQ09yjSg0d6ZrLQ | https://lh4.googleusercontent.com/soL4AuTCyyvi3hmwuFmg3YskaHbaJ4ifvjKFnKqVrGOlhwuBNrhiHHF_1l-_fVou8SL1t3qKbac_YxsWxTd9aqwLLqQmoIjVaMjwSRPbhCnmR90ZG9yxpeWbmccTIhTfScTih28FxA |
| Second Sample | https://lh6.googleusercontent.com/hkiRhUeDkBh9Oy5pTwhU5avZ6eZR8SQ0rOr_hmurNtYROl5QS_X-PrsZ_WqHzmcGxtngqOVUdm2c4zTEuR32BLmlFmoPN4h0LsSpG3qNuDZdmZrDIDaZriaFa0XA59odO1yOxmW29Q | https://lh4.googleusercontent.com/K3Mo2F3FVQqZ7lLlAdyyrc9EBCaC5k7IJ0eMXnoQxMJPheAu8kR954XhKoSgQqs0RVl4IVy1TiXEGZ9UsdPK5-2CI_XVNtM459SAPXtmGJaEHEhxCgvVRIInj5DPKgx36mV09c-OQQ | https://lh4.googleusercontent.com/sOw7C_JyyJVKU7AMlq39ux8XY0EYBtAaPKms2NMxZQFlAAn_u2EUDZWjm-Z4iNUOOkAnCA8MOOYM1iXe_K6OMH-bX3V1GZf8xYEfl3rPiL5-Vv6P8nsnsG3mhQn79-OfRruphMuCHA |
| Third Sample | https://lh3.googleusercontent.com/8qJ6PDrYcEDNHLDBb8_x-Q_0EBtHxt10f4k_e0_4kTLcN137766mmeEPF2mEbWHp9laCu2G2aQGmr4eRfW_L-kOqtMgotTdxJgxSfy_TTEunh1fa1Kci5MIpWKET-913Zql2Eq55yg | https://lh5.googleusercontent.com/Uh95zVxREOSe039BdMvuw70Qs1GihglVAqsKc6UqtQphXH1sQWYFlPZhKVUZU879G2TP91EbjDqZe1n1Ew_P6FvkBScpM_w68kw2vrVifrOcPs-IkwYUwix6XxuI6S0D-z9-AaeTsA | https://lh4.googleusercontent.com/iwQX-sutls_6kdjRMxIXBjEVDWR7dzCvdFX3O5WDcLcuAhBFzQQ1dL2uyuozm425pvtapvJsr6FcB2t88QlsrlrzR7KU0kU-gfC18QeZmE6GwLXCKZx6sE_0VWWVBo2HiqlLv_lFXA |

Table 4.8.2: Result of Recognition Accuracy for 3 Users

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Trainied User** | **Matching Accuracy** | | | | | | | | | |
| User\_1 | 1 | 1 | X | X | X | X | 1 | 1 | X | X |
| User\_2 | 2 | 2 | 3 | X | X | 2 | 2 | X | X | X |
| User\_3 | 3 | X | 3 | 3 | X | X | X | X | X | X |

Figure 4.8.1: Recognition Possibility Chart of the IRS

## Chapter 5 Conclusion

### 5.1 Conclusion

In a nutshell, this work focuses on the Self Organizing Map (SOM) portion of a complete IRS system. The six sub-blocks of SOM are Neuron Initialization Block, Hidden Layer Block, Weight Optimization Block, Iteration-Check Block, Voting System Block and On-Chip Training Block have been designed , integrated and verified using Verilog HDL based test-bench in Mentor-Graphics environment and downloaded on a Altera Cyclone II FPGA DE2 board for hardware verification purposes. The main aim of this project is to improve the recognition accuracy and speed. The Voting System Block is the proposed block to increase the accuracy by receiving 3 different updatedNetwork arrays from 3 different starting points. Other than that, the iris data of each user has been captured and stored as 3 separate sets of data in the databaseto increase recognition accuracy. The proposed SOM with voting system in this work achieves a recognition accuracy of 94% and a recognition speed of 1-2 seconds. Such low cost IRS is able to be applied in common households for security augmentation purposes.

### 5.1 Future Development

In the future, there are several improvements that can be performed on the accuracy and speed of the IRS. The first suggestion is to replace the camera lens to infrared lens and auto focus features to enhance the environmental lighting issues. Other than that, before performing compression enhancement on the grayscale values can be made to increase pattern accuracy. Lastly, the recognition section can be be enhanced with 2 or more additional algorithm to start the voting system instead of using the same algorithm with different starting points.

## Reference

1. Kaushik Roy, PrabitBhattacharyamChing Y. Suen, *Towards nonideal iris recognition based on level set method, genetic algorithms and adaptive asymmetrical SVMs.* [Online] Available at: http://www.sciencedirect.com.libezp.utar.edu.my/science/article/pii/S0952197610001375. [Accessed on 15 November 2014]
2. Patricia Melin, Daniela Sanchez, Oscar Castillo, *Genetic Optimization of modular neural networks with fuzzy response integration for human recognition*. [Online] Available at: http://teknik.unitomo.ac.id/wp-content/uploads/2013/07/1-s2.0-S0020025512001430-main.pdf. [Accessed on 15 November 2014]
3. HimanshuRai, AnamikaYadav, *Iris recognition using combined support vector machine and Hamming distance approach*. [Online] Available at: http://www.sciencedirect.com.libezp.utar.edu.my/science/article/pii/S0957417413005824. [Accessed on 15 November 2014]
4. Vinícius Gonçalves Maltarollo, Káthia Maria Honório and Albérico Borges Ferreira da Silva, *Kohenen Self-Organizing Maps*. [Online] Available at : http://cdn.intechopen.com/pdfs-wm/39067.pdf. [Accessed on 7 September 2015]
5. Shyam M. Guthikonda, *Kohenen Self-Organizing Maps*. [Online] Available at : http://www.shy.am/wp-content/uploads/2009/01/kohonen-self-organizing-maps-shyam-guthikonda.pdf. [Accessed on 2December 2015]
6. Yap and Lim, *Hand Gesture RecognitionFYP Thesis*, [PDF] [Accessed on 4 November 2014]
7. Kendar Pratap & Shelja, Artifical Neural Network (ANN) inspired from Biological Nervous System. [Online] Available at : http://www.ijaiem.org/volume2Issue1/IJAIEM-2013-01-28-064.pdf. [Accessed on 2 December 2015]
8. N. Yadav, *An Introduction to Neural Network Methods*. [Online] Available at : https://www.google.com/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&uact=8&ved=0ahUKEwjf0MLf64zKAhUJjo4KHZSkBuMQFgghMAA&url=http%3A%2F%2Fwww.springer.com%2Fcda%2Fcontent%2Fdocument%2Fcda\_downloaddocument%2F9789401798150-c2.pdf%3FSGWID%3D0-0-45-1495021-p177264210&usg=AFQjCNFxoO0mRW48ycAFdIVhTxaVt\_7WvQ&sig2=fHMWwKpwZyxf5WA3712mGQ. [Accessed on 2 December 2015]

## Appendix

DE2\_D5M

module DE2\_D5M

(

//////////////////// Clock Input ////////////////////

CLOCK\_27, // 27 MHz

CLOCK\_50, // 50 MHz

EXT\_CLOCK, // External Clock

//////////////////// Push Button ////////////////////

KEY, // Pushbutton[3:0]

//////////////////// DPDT Switch ////////////////////

SW, // Toggle Switch[17:0]

//////////////////// 7-SEG Dispaly ////////////////////

HEX0, // Seven Segment Digit 0

HEX1, // Seven Segment Digit 1

HEX2, // Seven Segment Digit 2

HEX3, // Seven Segment Digit 3

HEX4, // Seven Segment Digit 4

HEX5, // Seven Segment Digit 5

HEX6, // Seven Segment Digit 6

HEX7, // Seven Segment Digit 7

//////////////////////// LED ////////////////////////

LEDG, // LED Green[8:0]

LEDR, // LED Red[17:0]

//////////////////////// UART ////////////////////////

UART\_TXD, // UART Transmitter

UART\_RXD, // UART Receiver

//////////////////////// IRDA ////////////////////////

IRDA\_TXD, // IRDA Transmitter

IRDA\_RXD, // IRDA Receiver

///////////////////// SDRAM Interface ////////////////

DRAM\_DQ, // SDRAM Data bus 16 Bits

DRAM\_ADDR, // SDRAM Address bus 12 Bits

DRAM\_LDQM, // SDRAM Low-byte Data Mask

DRAM\_UDQM, // SDRAM High-byte Data Mask

DRAM\_WE\_N, // SDRAM Write Enable

DRAM\_CAS\_N, // SDRAM Column Address Strobe

DRAM\_RAS\_N, // SDRAM Row Address Strobe

DRAM\_CS\_N, // SDRAM Chip Select

DRAM\_BA\_0, // SDRAM Bank Address 0

DRAM\_BA\_1, // SDRAM Bank Address 0

DRAM\_CLK, // SDRAM Clock

DRAM\_CKE, // SDRAM Clock Enable

//////////////////// Flash Interface ////////////////

FL\_DQ, // FLASH Data bus 8 Bits

FL\_ADDR, // FLASH Address bus 22 Bits

FL\_WE\_N, // FLASH Write Enable

FL\_RST\_N, // FLASH Reset

FL\_OE\_N, // FLASH Output Enable

FL\_CE\_N, // FLASH Chip Enable

//////////////////// SRAM Interface ////////////////

SRAM\_DQ, // SRAM Data bus 16 Bits

SRAM\_ADDR, // SRAM Address bus 18 Bits

SRAM\_UB\_N, // SRAM High-byte Data Mask

SRAM\_LB\_N, // SRAM Low-byte Data Mask

SRAM\_WE\_N, // SRAM Write Enable

SRAM\_CE\_N, // SRAM Chip Enable

SRAM\_OE\_N, // SRAM Output Enable

//////////////////// ISP1362 Interface ////////////////

OTG\_DATA, // ISP1362 Data bus 16 Bits

OTG\_ADDR, // ISP1362 Address 2 Bits

OTG\_CS\_N, // ISP1362 Chip Select

OTG\_RD\_N, // ISP1362 Write

OTG\_WR\_N, // ISP1362 Read

OTG\_RST\_N, // ISP1362 Reset

OTG\_FSPEED, // USB Full Speed, 0 = Enable, Z = Disable

OTG\_LSPEED, // USB Low Speed, 0 = Enable, Z = Disable

OTG\_INT0, // ISP1362 Interrupt 0

OTG\_INT1, // ISP1362 Interrupt 1

OTG\_DREQ0, // ISP1362 DMA Request 0

OTG\_DREQ1, // ISP1362 DMA Request 1

OTG\_DACK0\_N, // ISP1362 DMA Acknowledge 0

OTG\_DACK1\_N, // ISP1362 DMA Acknowledge 1

//////////////////// LCD Module 16X2 ////////////////

LCD\_ON, // LCD Power ON/OFF

LCD\_BLON, // LCD Back Light ON/OFF

LCD\_RW, // LCD Read/Write Select, 0 = Write, 1 = Read

LCD\_EN, // LCD Enable

LCD\_RS, // LCD Command/Data Select, 0 = Command, 1 = Data

LCD\_DATA, // LCD Data bus 8 bits

//////////////////// SD\_Card Interface ////////////////

SD\_DAT, // SD Card Data

SD\_DAT3, // SD Card Data 3

SD\_CMD, // SD Card Command Signal

SD\_CLK, // SD Card Clock

//////////////////// USB JTAG link ////////////////////

TDI, // CPLD -> FPGA (data in)

TCK, // CPLD -> FPGA (clk)

TCS, // CPLD -> FPGA (CS)

TDO, // FPGA -> CPLD (data out)

//////////////////// I2C ////////////////////////////

I2C\_SDAT, // I2C Data

I2C\_SCLK, // I2C Clock

//////////////////// PS2 ////////////////////////////

PS2\_DAT, // PS2 Data

PS2\_CLK, // PS2 Clock

//////////////////// VGA ////////////////////////////

VGA\_CLK, // VGA Clock

VGA\_HS, // VGA H\_SYNC

VGA\_VS, // VGA V\_SYNC

VGA\_BLANK, // VGA BLANK

VGA\_SYNC, // VGA SYNC

VGA\_R, // VGA Red[9:0]

VGA\_G, // VGA Green[9:0]

VGA\_B, // VGA Blue[9:0]

//////////// Ethernet Interface ////////////////////////

ENET\_DATA, // DM9000A DATA bus 16Bits

ENET\_CMD, // DM9000A Command/Data Select, 0 = Command, 1 = Data

ENET\_CS\_N, // DM9000A Chip Select

ENET\_WR\_N, // DM9000A Write

ENET\_RD\_N, // DM9000A Read

ENET\_RST\_N, // DM9000A Reset

ENET\_INT, // DM9000A Interrupt

ENET\_CLK, // DM9000A Clock 25 MHz

//////////////// Audio CODEC ////////////////////////

AUD\_ADCLRCK, // Audio CODEC ADC LR Clock

AUD\_ADCDAT, // Audio CODEC ADC Data

AUD\_DACLRCK, // Audio CODEC DAC LR Clock

AUD\_DACDAT, // Audio CODEC DAC Data

AUD\_BCLK, // Audio CODEC Bit-Stream Clock

AUD\_XCK, // Audio CODEC Chip Clock

//////////////// TV Decoder ////////////////////////

TD\_DATA, // TV Decoder Data bus 8 bits

TD\_HS, // TV Decoder H\_SYNC

TD\_VS, // TV Decoder V\_SYNC

TD\_RESET, // TV Decoder Reset

//////////////////// GPIO ////////////////////////////

GPIO\_0, // GPIO Connection 0

GPIO\_1 // GPIO Connection 1

);

//////////////////////// Clock Input ////////////////////////

input CLOCK\_27; // 27 MHz

input CLOCK\_50; // 50 MHz

input EXT\_CLOCK; // External Clock

//////////////////////// Push Button ////////////////////////

input [3:0] KEY; // Pushbutton[3:0]

//////////////////////// DPDT Switch ////////////////////////

input [17:0] SW; // Toggle Switch[17:0]

//////////////////////// 7-SEG Dispaly ////////////////////////

output [6:0] HEX0; // Seven Segment Digit 0

output [6:0] HEX1; // Seven Segment Digit 1

output [6:0] HEX2; // Seven Segment Digit 2

output [6:0] HEX3; // Seven Segment Digit 3

output [6:0] HEX4; // Seven Segment Digit 4

output [6:0] HEX5; // Seven Segment Digit 5

output [6:0] HEX6; // Seven Segment Digit 6

output [6:0] HEX7; // Seven Segment Digit 7

//////////////////////////// LED ////////////////////////////

output [8:0] LEDG; // LED Green[8:0]

output [17:0] LEDR; // LED Red[17:0]

//////////////////////////// UART ////////////////////////////

output UART\_TXD; // UART Transmitter

input UART\_RXD; // UART Receiver

//////////////////////////// IRDA ////////////////////////////

output IRDA\_TXD; // IRDA Transmitter

input IRDA\_RXD; // IRDA Receiver

/////////////////////// SDRAM Interface ////////////////////////

inout [15:0] DRAM\_DQ; // SDRAM Data bus 16 Bits

output [11:0] DRAM\_ADDR; // SDRAM Address bus 12 Bits

output DRAM\_LDQM; // SDRAM Low-byte Data Mask

output DRAM\_UDQM; // SDRAM High-byte Data Mask

output DRAM\_WE\_N; // SDRAM Write Enable

output DRAM\_CAS\_N; // SDRAM Column Address Strobe

output DRAM\_RAS\_N; // SDRAM Row Address Strobe

output DRAM\_CS\_N; // SDRAM Chip Select

output DRAM\_BA\_0; // SDRAM Bank Address 0

output DRAM\_BA\_1; // SDRAM Bank Address 0

output DRAM\_CLK; // SDRAM Clock

output DRAM\_CKE; // SDRAM Clock Enable

//////////////////////// Flash Interface ////////////////////////

inout [7:0] FL\_DQ; // FLASH Data bus 8 Bits

output [21:0] FL\_ADDR; // FLASH Address bus 22 Bits

output FL\_WE\_N; // FLASH Write Enable

output FL\_RST\_N; // FLASH Reset

output FL\_OE\_N; // FLASH Output Enable

output FL\_CE\_N; // FLASH Chip Enable

//////////////////////// SRAM Interface ////////////////////////

inout [15:0] SRAM\_DQ; // SRAM Data bus 16 Bits

output [17:0] SRAM\_ADDR; // SRAM Address bus 18 Bits

output SRAM\_UB\_N; // SRAM High-byte Data Mask

output SRAM\_LB\_N; // SRAM Low-byte Data Mask

output SRAM\_WE\_N; // SRAM Write Enable

output SRAM\_CE\_N; // SRAM Chip Enable

output SRAM\_OE\_N; // SRAM Output Enable

//////////////////// ISP1362 Interface ////////////////////////

inout [15:0] OTG\_DATA; // ISP1362 Data bus 16 Bits

output [1:0] OTG\_ADDR; // ISP1362 Address 2 Bits

output OTG\_CS\_N; // ISP1362 Chip Select

output OTG\_RD\_N; // ISP1362 Write

output OTG\_WR\_N; // ISP1362 Read

output OTG\_RST\_N; // ISP1362 Reset

output OTG\_FSPEED; // USB Full Speed, 0 = Enable, Z = Disable

output OTG\_LSPEED; // USB Low Speed, 0 = Enable, Z = Disable

input OTG\_INT0; // ISP1362 Interrupt 0

input OTG\_INT1; // ISP1362 Interrupt 1

input OTG\_DREQ0; // ISP1362 DMA Request 0

input OTG\_DREQ1; // ISP1362 DMA Request 1

output OTG\_DACK0\_N; // ISP1362 DMA Acknowledge 0

output OTG\_DACK1\_N; // ISP1362 DMA Acknowledge 1

//////////////////// LCD Module 16X2 ////////////////////////////

inout [7:0] LCD\_DATA; // LCD Data bus 8 bits

output LCD\_ON; // LCD Power ON/OFF

output LCD\_BLON; // LCD Back Light ON/OFF

output LCD\_RW; // LCD Read/Write Select, 0 = Write, 1 = Read

output LCD\_EN; // LCD Enable

output LCD\_RS; // LCD Command/Data Select, 0 = Command, 1 = Data

//////////////////// SD Card Interface ////////////////////////

inout SD\_DAT; // SD Card Data

inout SD\_DAT3; // SD Card Data 3

inout SD\_CMD; // SD Card Command Signal

output SD\_CLK; // SD Card Clock

//////////////////////// I2C ////////////////////////////////

inout I2C\_SDAT; // I2C Data

output I2C\_SCLK; // I2C Clock

//////////////////////// PS2 ////////////////////////////////

input PS2\_DAT; // PS2 Data

input PS2\_CLK; // PS2 Clock

//////////////////// USB JTAG link ////////////////////////////

input TDI; // CPLD -> FPGA (data in)

input TCK; // CPLD -> FPGA (clk)

input TCS; // CPLD -> FPGA (CS)

output TDO; // FPGA -> CPLD (data out)

//////////////////////// VGA ////////////////////////////

output VGA\_CLK; // VGA Clock

output VGA\_HS; // VGA H\_SYNC

output VGA\_VS; // VGA V\_SYNC

output VGA\_BLANK; // VGA BLANK

output VGA\_SYNC; // VGA SYNC

output [9:0] VGA\_R; // VGA Red[9:0]

output [9:0] VGA\_G; // VGA Green[9:0]

output [9:0] VGA\_B; // VGA Blue[9:0]

//////////////// Ethernet Interface ////////////////////////////

inout [15:0] ENET\_DATA; // DM9000A DATA bus 16Bits

output ENET\_CMD; // DM9000A Command/Data Select, 0 = Command, 1 = Data

output ENET\_CS\_N; // DM9000A Chip Select

output ENET\_WR\_N; // DM9000A Write

output ENET\_RD\_N; // DM9000A Read

output ENET\_RST\_N; // DM9000A Reset

input ENET\_INT; // DM9000A Interrupt

output ENET\_CLK; // DM9000A Clock 25 MHz

//////////////////// Audio CODEC ////////////////////////////

inout AUD\_ADCLRCK; // Audio CODEC ADC LR Clock

input AUD\_ADCDAT; // Audio CODEC ADC Data

inout AUD\_DACLRCK; // Audio CODEC DAC LR Clock

output AUD\_DACDAT; // Audio CODEC DAC Data

inout AUD\_BCLK; // Audio CODEC Bit-Stream Clock

output AUD\_XCK; // Audio CODEC Chip Clock

//////////////////// TV Devoder ////////////////////////////

input [7:0] TD\_DATA; // TV Decoder Data bus 8 bits

input TD\_HS; // TV Decoder H\_SYNC

input TD\_VS; // TV Decoder V\_SYNC

output TD\_RESET; // TV Decoder Reset

//////////////////////// GPIO ////////////////////////////////

inout [35:0] GPIO\_0; // GPIO Connection 0

inout [35:0] GPIO\_1; // GPIO Connection 1

///////////////////////////////////////////////////////////////////

//=============================================================================

// REG/WIRE declarations

//=============================================================================

// CCD

wire [11:0] CCD\_DATA;

wire CCD\_SDAT;

wire CCD\_SCLK;

wire CCD\_FLASH;

wire CCD\_FVAL;

wire CCD\_LVAL;

wire CCD\_PIXCLK;

wire CCD\_MCLK; // CCD Master Clock

wire [15:0] Read\_DATA1;

wire [15:0] Read\_DATA2;

wire VGA\_CTRL\_CLK;

wire [11:0] mCCD\_DATA;

wire mCCD\_DVAL;

wire mCCD\_DVAL\_d;

wire [15:0] X\_Cont;

wire [15:0] Y\_Cont;

wire [9:0] X\_ADDR;

wire [31:0] Frame\_Cont;

wire DLY\_RST\_0;

wire DLY\_RST\_1;

wire DLY\_RST\_2;

wire Read;

reg [11:0] rCCD\_DATA;

reg rCCD\_LVAL;

reg rCCD\_FVAL;

wire [11:0] sCCD\_R;

wire [11:0] sCCD\_G;

wire [11:0] sCCD\_B;

wire sCCD\_DVAL;

wire [9:0] VGA\_R; // VGA Red[9:0]

wire [9:0] VGA\_G; // VGA Green[9:0]

wire [9:0] VGA\_B; // VGA Blue[9:0]

reg [1:0] rClk;

wire sdram\_ctrl\_clk;

//=============================================================================

// Structural coding

//=============================================================================

assign CCD\_DATA[0] = GPIO\_1[13];

assign CCD\_DATA[1] = GPIO\_1[12];

assign CCD\_DATA[2] = GPIO\_1[11];

assign CCD\_DATA[3] = GPIO\_1[10];

assign CCD\_DATA[4] = GPIO\_1[9];

assign CCD\_DATA[5] = GPIO\_1[8];

assign CCD\_DATA[6] = GPIO\_1[7];

assign CCD\_DATA[7] = GPIO\_1[6];

assign CCD\_DATA[8] = GPIO\_1[5];

assign CCD\_DATA[9] = GPIO\_1[4];

assign CCD\_DATA[10]= GPIO\_1[3];

assign CCD\_DATA[11]= GPIO\_1[1];

assign GPIO\_1[16] = CCD\_MCLK;

assign CCD\_FVAL = GPIO\_1[22];

assign CCD\_LVAL = GPIO\_1[21];

assign CCD\_PIXCLK = GPIO\_1[0];

assign GPIO\_1[19] = 1'b1; // tRIGGER

assign GPIO\_1[17] = DLY\_RST\_1;

/\*

assign LEDR = SW;

assign LEDG = Y\_Cont;

\*/

assign VGA\_CTRL\_CLK= rClk[0];

assign VGA\_CLK = ~rClk[0];

always@(posedge CLOCK\_50) rClk <= rClk+1;

always@(posedge CCD\_PIXCLK)

begin

rCCD\_DATA <= CCD\_DATA;

rCCD\_LVAL <= CCD\_LVAL;

rCCD\_FVAL <= CCD\_FVAL;

end

VGA\_Controller u1 ( // Host Side

.oRequest(Read),

.iRed(wDISP\_R),

.iGreen(wDISP\_G),

.iBlue(wDISP\_B),

// VGA Side

.oVGA\_R(VGA\_R),

.oVGA\_G(VGA\_G),

.oVGA\_B(VGA\_B),

.oVGA\_H\_SYNC(VGA\_HS),

.oVGA\_V\_SYNC(VGA\_VS),

.oVGA\_SYNC(VGA\_SYNC),

.oVGA\_BLANK(VGA\_BLANK),

// Control Signal

.iCLK(VGA\_CTRL\_CLK),

.iRST\_N(DLY\_RST\_2)

);

Reset\_Delay u2 ( .iCLK(CLOCK\_50),

.iRST(KEY[0]),

.oRST\_0(DLY\_RST\_0),

.oRST\_1(DLY\_RST\_1),

.oRST\_2(DLY\_RST\_2)

);

CCD\_Capture u3 ( .oDATA(mCCD\_DATA),

.oDVAL(mCCD\_DVAL),

.oX\_Cont(X\_Cont),

.oY\_Cont(Y\_Cont),

.oFrame\_Cont(Frame\_Cont),

.iDATA(rCCD\_DATA),

.iFVAL(rCCD\_FVAL),

.iLVAL(rCCD\_LVAL),

.iSTART(!KEY[3]),

.iEND(!KEY[2]),

.iCLK(CCD\_PIXCLK),

.iRST(DLY\_RST\_2)

);

RAW2RGB u4 ( .iCLK(CCD\_PIXCLK),

.iRST(DLY\_RST\_1),

.iDATA(mCCD\_DATA),

.iDVAL(mCCD\_DVAL),

.oRed(sCCD\_R),

.oGreen(sCCD\_G),

.oBlue(sCCD\_B),

.oDVAL(sCCD\_DVAL),

.iX\_Cont(X\_Cont),

.iY\_Cont(Y\_Cont)

);

SEG7\_LUT\_8 u5 ( .oSEG0(HEX0),.oSEG1(HEX1),

.oSEG2(HEX2),.oSEG3(HEX3),

.oSEG4(HEX4),.oSEG5(HEX5),

.oSEG6(HEX6),.oSEG7(HEX7),

.iDIG(Frame\_Cont[31:0])

);

sdram\_pll u6 (

.inclk0(CLOCK\_50),

.c0(sdram\_ctrl\_clk),

.c1(DRAM\_CLK)

);

assign CCD\_MCLK = rClk[0];

Sdram\_Control\_4Port u7 ( // HOST Side

.REF\_CLK(CLOCK\_50),

.RESET\_N(1'b1),

.CLK(sdram\_ctrl\_clk),

// FIFO Write Side 1

.WR1\_DATA({1'b0,sCCD\_G[11:7],sCCD\_B[11:2]}),

.WR1(sCCD\_DVAL),

.WR1\_ADDR(0),

.WR1\_MAX\_ADDR(640\*480),

.WR1\_LENGTH(9'h100),

.WR1\_LOAD(!DLY\_RST\_0),

.WR1\_CLK(~CCD\_PIXCLK),

// FIFO Write Side 2

.WR2\_DATA( {1'b0,sCCD\_G[6:2],sCCD\_R[11:2]}),

.WR2(sCCD\_DVAL),

.WR2\_ADDR(22'h100000),

.WR2\_MAX\_ADDR(22'h100000+640\*480),

.WR2\_LENGTH(9'h100),

.WR2\_LOAD(!DLY\_RST\_0),

.WR2\_CLK(~CCD\_PIXCLK),

// FIFO Read Side 1

.RD1\_DATA(Read\_DATA1),

.RD1(Read),

.RD1\_ADDR(0),

.RD1\_MAX\_ADDR(640\*480),

.RD1\_LENGTH(9'h100),

.RD1\_LOAD(!DLY\_RST\_0),

.RD1\_CLK(~VGA\_CTRL\_CLK),

// FIFO Read Side 2

.RD2\_DATA(Read\_DATA2),

.RD2(Read),

.RD2\_ADDR(22'h100000),

.RD2\_MAX\_ADDR(22'h100000+640\*480),

.RD2\_LENGTH(9'h100),

.RD2\_LOAD(!DLY\_RST\_0),

.RD2\_CLK(~VGA\_CTRL\_CLK),

// SDRAM Side

.SA(DRAM\_ADDR),

.BA({DRAM\_BA\_1,DRAM\_BA\_0}),

.CS\_N(DRAM\_CS\_N),

.CKE(DRAM\_CKE),

.RAS\_N(DRAM\_RAS\_N),

.CAS\_N(DRAM\_CAS\_N),

.WE\_N(DRAM\_WE\_N),

.DQ(DRAM\_DQ),

.DQM({DRAM\_UDQM,DRAM\_LDQM})

);

assign UART\_TXD = UART\_RXD;

I2C\_CCD\_Config u8 ( // Host Side

.iCLK(CLOCK\_50),

.iRST\_N(DLY\_RST\_2),

.iZOOM\_MODE\_SW(SW[16]),

.iEXPOSURE\_ADJ(KEY[1]),

.iEXPOSURE\_DEC\_p(SW[0]),

// I2C Side

.I2C\_SCLK(GPIO\_1[24]),

.I2C\_SDAT(GPIO\_1[23])

);

wire [9:0] wVGA\_R = Read\_DATA2[9:0];

wire [9:0] wVGA\_G = {Read\_DATA1[14:10],Read\_DATA2[14:10]};

wire [9:0] wVGA\_B = Read\_DATA1[9:0];

//Coding

wire [9:0] gDATA;

wire gCCD\_DVAL;

wire wGFlag;

RGB2GRAY u9 (

.oDVAL(gCCD\_DVAL),

.oDATA(gDATA),

.oFlag(wGFlag),

.iRed(wVGA\_R),

.iGreen(wVGA\_G),

.iBlue(wVGA\_B),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(Read),

);

wire bDATA;

wire bCCD\_DVAL;

PupilBinaryImage u10 (

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(gCCD\_DVAL),

.iDATA(gDATA),

.oDATA(bDATA),

.oDVAL(bCCD\_DVAL),

);

wire oErosion;

wire eCCD\_DVAL;

Erosion u11 (

.iDATA(bDATA),

.iDVAL(bCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.oDATA(oErosion),

.oDVAL(eCCD\_DVAL),

);

wire oDilation;

wire dCCD\_DVAL;

Dilation u12 (

.iDATA(oErosion),

.iDVAL(eCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.oDATA(oDilation),

.oDVAL(dCCD\_DVAL),

);

wire [9:0] woC\_1;

wire [9:0] woR\_1;

wire [9:0] woC\_2;

wire [9:0] woR\_2;

wire [9:0] woC\_3;

wire [9:0] woR\_3;

wire [9:0] woC\_4;

wire [9:0] woR\_4;

wire [9:0] woC\_PC;

wire [9:0] woC\_PR;

wire [9:0] woPRadius;

wire wPFlagCoor;

wire oDetectPupil;

wire dpCCD\_DVAL;

Detect\_OuterPupil u13(

.iEND(SW[17]),

.iDATA(oDilation),

.iDVAL(dCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.oC\_1(woC\_1), //top

.oR\_1(woR\_1),

.oC\_2(woC\_2),//left

.oR\_2(woR\_2),

.oC\_3(woC\_3),//right

.oR\_3(woR\_3),

.oC\_4(woC\_4),//btm

.oR\_4(woR\_4),

.oC\_PC(woC\_PC),

.oC\_PR(woC\_PR),

.oFlag(wPFlagCoor),

.oDATA(oDetectPupil),

.oDVAL(dpCCD\_DVAL),

.oPRadius(woPRadius),

);

wire [9:0]pPupilCoor;

wire ppCCD\_DVAL;

Output\_Point u14(

.iDATA(ibDATA),

.iDVAL(ibCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iC\_1(woC\_1),

.iR\_1(woR\_1),

.iC\_2(woC\_2),

.iR\_2(woR\_2),

.iC\_3(woC\_3),

.iR\_3(woR\_3),

.iC\_4(woC\_4),

.iR\_4(woR\_4),

.iC\_PC(woC\_PC),

.iC\_PR(woC\_PR),

.iI\_R1(woI\_R1),

.iI\_C1(woI\_C1),

.iI\_R2(woI\_R2),

.iI\_C2(woI\_C2),

.oDVAL(ppCCD\_DVAL),

.oDATA(pPupilCoor),

);

wire ibData;

wire ibCCD\_DVAL;

IrisBinaryImage u15 (

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(gCCD\_DVAL),

.iDATA(gDATA),

.oDATA(ibDATA),

.oDVAL(ibCCD\_DVAL),

);

wire [9:0] woI\_R1;

wire [9:0] woI\_C1;

wire [9:0] woI\_R2;

wire [9:0] woI\_C2;

wire [9:0] woIRadius;

wire wIFlagCoor;

Detect\_InnerIris u16(

.iDATA(ibDATA),

.iDVAL(ibCCD\_DVAL),

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iC\_PC(woC\_PC),

.iC\_PR(woC\_PR),

.oFlag(wIFlagCoor),

.oI\_R1(woI\_R1), //left

.oI\_C1(woI\_C1),

.oI\_R2(woI\_R2), //right

.oI\_C2(woI\_C2),

.oIRadius(woIRadius),

);

wire [9:0] wNDATA;

wire nCCD\_DVAL;

wire nSignal;

/\*irisNormalization u17(

.iCLK(VGA\_CTRL\_CLK),

.iRST(DLY\_RST\_2),

.iDVAL(gCCD\_DVAL),

.iPupilFlag(wPFlagCoor),

.iIrisFlag(wIFlagCoor),

.iCaptured(!KEY[2]),

.iStart(!KEY[3]),

//.startGetData(wGFlag), //Receive signal from segmentation

.iDATA(gDATA),

.pupil\_radius(woPRadius),

.iris\_radius(woIRadius),

.centre\_pointX(woC\_PC),

.centre\_pointY(woC\_PR),

.oDVAL(nCCD\_DVAL),

.oSignal(nSignal), // Give signal to matching

.oDATA(wNDATA),

);\*/

assign LEDR[0] = nSignal;

assign LEDR[1] = wGFlag;

//output image

reg [9:0] pupilBinaryImage;

always@(posedge CLOCK\_50)begin

if(bDATA == 1)begin

pupilBinaryImage = 10'd1023;

end

else begin

pupilBinaryImage = 10'd0;

end

end

reg [9:0] erosionImage;

always@(posedge CLOCK\_50)begin

if(oErosion == 1)begin

erosionImage = 10'd1023;

end

else begin

erosionImage = 10'd0;

end

end

reg [9:0] dilationImage;

always@(posedge CLOCK\_50)begin

if(oDilation == 1)begin

dilationImage = 10'd1023;

end

else begin

dilationImage = 10'd0;

end

end

reg [9:0] irisBinaryImage;

always@(posedge CLOCK\_50)begin

if(ibDATA == 1)begin

irisBinaryImage = 10'd1023;

end

else begin

irisBinaryImage = 10'd0;

end

end

wire [9:0] wDISP\_R =

SW[7] ? wNDATA:

SW[6] ? pPupilCoor :

SW[5] ? irisBinaryImage:

SW[4] ? dilationImage :

SW[3] ? erosionImage :

SW[2] ? pupilBinaryImage:

SW[1] ? gDATA :

wVGA\_R;

wire [9:0] wDISP\_G =

SW[7] ? wNDATA :

SW[6] ? pPupilCoor :

SW[5] ? irisBinaryImage:

SW[4] ? dilationImage :

SW[3] ? erosionImage :

SW[2] ? pupilBinaryImage:

SW[1] ? gDATA :

wVGA\_G;

wire [9:0] wDISP\_B =

SW[7] ? wNDATA :

SW[6] ? pPupilCoor :

SW[5] ? irisBinaryImage:

SW[4] ? dilationImage :

SW[3] ? erosionImage :

SW[2] ? pupilBinaryImage:

SW[1] ? gDATA :

wVGA\_B;

endmodule

SOM.sv

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* ANN\_SOM Block

\*

\* Operation:

\* 1) To train user data into database

\* 2) To perform iris recognition by using SOM method

\*

\* Input:

\* CLOCK\_50

\* iRST

\* startTraining

\* trainUser

\* storeControl

\* storeUser

\* data[10][10]

\*

\* Output:

\* userID

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module SOM(

input CLOCK\_50, iRST, bRST,

input startRecognition,

input startTraining,

input reg trainUser,

input reg storeControl,

input reg [3:0] storeUser,

input reg data[0:9][0:9],

output reg [17:14] LED

);

//reg data[0:9][0:9];

reg initNetwork[0:9][0:9];

reg initFinish;

reg [3:0] selected\_x, selected\_y;

reg [3:0] weight\_x, weight\_y;

reg [3:0] winner\_x, winner\_y;

reg [3:0] i = 4'b0000, j = 4'b0000;

reg flagForUpdateSelected = 1;

reg [3:0] count\_i = 4'b0000;

reg [3:0] count\_j = 4'b0000;

reg up\_winner = 0;

reg update = 0;

reg [3:0] weight\_selected\_x, weight\_selected\_y;

reg [3:0] old\_weight\_x, old\_weight\_y;

reg updatedNetwork[0:9][0:9];

reg voteNetwork[0:9][0:9];

reg startCompare;

reg next = 0;

reg next2 = 0;

reg next3 = 0;

reg successVote;

reg reset = 0;

reg startWeight = 1;

reg goForNext = 0;

reg goForNext2 = 0;

reg secondRound = 0;

reg thirdRound = 0;

wire [3:0] userID;

neuronInputInitialization init (CLOCK\_50, iRST, initNetwork, initFinish);

onChipTraining train (CLOCK\_50, iRST, bRST, startTraining, trainUser, storeControl, storeUser, data, voteNetwork, successVote, userID);

hiddenLayer hidden (CLOCK\_50, iRST, selected\_x, selected\_y, weight\_x, weight\_y, winner\_x, winner\_y);

weightOptimization neuronUpdate (CLOCK\_50, iRST, up\_winner, update, startWeight, initNetwork, weight\_selected\_x, weight\_selected\_y, old\_weight\_x, old\_weight\_y, updatedNetwork);

Iteration compare (CLOCK\_50, iRST, startRecognition, reset, updatedNetwork, startCompare);

votingSystem vote (CLOCK\_50, iRST, data, updatedNetwork, next, next2, next3, successVote, voteNetwork);

always@(posedge CLOCK\_50 or negedge iRST)begin

if(!iRST)begin

selected\_x = 4'b0000;

selected\_y = 4'b0000;

weight\_x = 4'b0000;

weight\_y = 4'b0000;

i = 4'b0000;

j = 4'b0000;

flagForUpdateSelected = 1;

count\_j = 4'b0000;

count\_i = 4'b0000;

up\_winner = 0;

update = 0;

weight\_selected\_x = 4'b0000;

weight\_selected\_y = 4'b0000;

old\_weight\_x = 4'b0000;

old\_weight\_y = 4'b0000;

reset = 0;

next = 0;

next2 = 0;

next3 = 0;

startWeight = 1;

goForNext = 0;

goForNext2 = 0;

secondRound = 0;

thirdRound = 0;

LED [17:14] = 0;

//userID = 4'bzzzz;

end

else begin

if(startRecognition == 1)begin

if(initFinish == 1)begin

if(flagForUpdateSelected == 1)begin

if(data[count\_i][count\_j] == 1)begin

selected\_x = count\_i;

selected\_y = count\_j;

end

flagForUpdateSelected = 0;

end

if(flagForUpdateSelected == 0)begin

if(update == 1)begin

if(updatedNetwork[i][j] == 1)begin

weight\_x = i;

weight\_y = j;

end

end

else begin

if(initNetwork[i][j] == 1)begin

weight\_x = i;

weight\_y = j;

end

end

//////////////////////////////////////////////////////////////////////

if(secondRound == 0 && thirdRound == 0)begin

if(i < 4'b1010)begin

if(j < 4'b1010)begin

j = j + 4'b0001;

if(j == 4'b1010)begin

j = 4'b0000;

i = i + 4'b0001;

end

end

end

if(reset == 1)

reset = 0;

if(goForNext == 1)begin

update = 0;

reset = 1;

i = 0;

j = 4'b1001;

count\_i = 0;

count\_j = 4'b1001;

flagForUpdateSelected = 1;

goForNext = 0;

if(startCompare == 0)

secondRound = 1;

end

if(startCompare == 1)begin

startWeight = 1;

goForNext = 1;

next = 1;

end

else if(update == 1)

startWeight = 0;

if(i == 4'b1010)begin

count\_j = count\_j + 1;

if(count\_j == 4'b1010)begin

count\_i = count\_i + 1;

if(count\_i == 4'b1010)begin

count\_i = 4'b0000;

end

count\_j = 4'b0000;

end

i = 4'b0000;

j = 4'b0000;

update = 1;

up\_winner = 1;

weight\_selected\_x = selected\_x;

weight\_selected\_y = selected\_y;

old\_weight\_x = winner\_x;

old\_weight\_y = winner\_y;

flagForUpdateSelected = 1;

end

end

///////////////////////////////////////////////////////////////////////////////

///////////////////////////////////////////////////////////////////////////////

else if(secondRound == 1 && thirdRound == 0)begin

if(i < 4'b1010)begin

if(j >= 4'b0000 && j <= 4'b1010)begin

j = j - 4'b0001;

if(j == 4'b1111)begin //2nd compliment -1

i = i + 4'b0001;

j = 4'b1001;

end

end

end

if(reset == 1)

reset = 0;

if(goForNext2 == 1)begin

update = 0;

reset = 1;

i = 4'b1001;

j = 4'b1001;

count\_i = 4'b1001;

count\_j = 4'b1001;

flagForUpdateSelected = 1;

goForNext2 = 0;

if(startCompare == 0)

thirdRound = 1;

end

if(startCompare == 1)begin

startWeight = 1;

goForNext2 = 1;

next2 = 1;

end

else if(update == 1)

startWeight = 0;

if(i == 4'b1010)begin

count\_j = count\_j - 4'b0001;

if(count\_j == 4'b1111)begin //2nd compliment -1

count\_i = count\_i + 4'b0001;

if(count\_i == 4'b1010)begin

count\_i = 4'b0000;

end

count\_j = 4'b1001;

end

i = 4'b0000;

j = 4'b1001;

update = 1;

up\_winner = 1;

weight\_selected\_x = selected\_x;

weight\_selected\_y = selected\_y;

old\_weight\_x = winner\_x;

old\_weight\_y = winner\_y;

flagForUpdateSelected = 1;

end

end

//////////////////////////////////////////////////////////////////

else if(thirdRound == 1)begin

if(i >= 4'b0000 && i <= 4'b1010)begin

if(j >= 4'b0000 && j <= 4'b1010)begin

j = j - 4'b0001;

if(j == 4'b1111)begin //2nd compliment -1

i = i - 4'b0001;

j = 4'b1001;

end

end

end

if(reset == 1)

reset = 0;

if(startCompare == 1)begin

next3 = 1;

end

else if(update == 1)

startWeight = 0;

if(i == 4'b1111)begin

count\_j = count\_j - 1;

if(count\_j == 4'b1111)begin

count\_i = count\_i - 1;

if(count\_i == 4'b1111)begin

count\_i = 4'b1001;

end

count\_j = 4'b1001;

end

i = 4'b1001;

j = 4'b1001;

update = 1;

up\_winner = 1;

weight\_selected\_x = selected\_x;

weight\_selected\_y = selected\_y;

old\_weight\_x = winner\_x;

old\_weight\_y = winner\_y;

flagForUpdateSelected = 1;

end

end

///////////////////////////////////////////////////////////////////////

end

end

end

LED = userID;

end

end

endmodule

neuronInitialization.sv

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Neuron Initialization Block(Block 1)

\*

\* Operation:

\* To init the network to '0' and assign '1' to the

\* network in the range of (2,2) to (7,7)

\*

\* Input:

\* CLOCK\_50

\* iRST

\*

\* Output:

\* network

\* initFinish

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module neuronInputInitialization(

input CLOCK\_50, iRST,

output reg network[0:9][0:9],

output reg initFinish

);

reg initToZero = 0;

reg assignInMiddle = 1;

reg state = 0;

reg startInitialize = 1;

reg startAssign = 1;

reg [3:0] i;

always@(posedge CLOCK\_50 or negedge iRST)begin

if(!iRST)begin

startInitialize = 1;

startAssign = 1;

i = 4'b0000;

initFinish = 0;

state = 0;

end

else begin

// Here is to initialize 0 the network

case(state)

initToZero:begin

if(startInitialize == 1)begin

for(i = 4'd0; i < 4'd10; i = i + 4'd1)begin

network[i][0] = 0;

network[i][1] = 0;

network[i][2] = 0;

network[i][3] = 0;

network[i][4] = 0;

network[i][5] = 0;

network[i][6] = 0;

network[i][7] = 0;

network[i][8] = 0;

network[i][9] = 0;

end

state = assignInMiddle;

startInitialize = 0;

startAssign = 1;

end

end

// Here is to assign 1 to the range of (2,2) to (7,7) of the network

// When finished assign then initFinish will be set to 1 as finish flag

assignInMiddle:begin

if(startAssign == 1)begin

for(i = 4'd2; i < 4'd8; i = i + 4'd1)begin

network[i][2] = 1;

network[i][3] = 1;

network[i][4] = 1;

network[i][5] = 1;

network[i][6] = 1;

network[i][7] = 1;

end

startAssign = 0;

initFinish = 1;

end

end

default: state = initToZero;

endcase

end

end

endmodule

hiddenLayer.sv

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Hidden Layer Block(Block 2)

\*

\* Operation:

\* To find the winner\_x and winner\_y

\*

\* Input:

\* selected\_x

\* selected\_y

\* weight\_x

\* weight\_y

\*

\* Output:

\* winner\_x

\* winner\_y

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module hiddenLayer(

input CLOCK\_50, iRST,

input reg [3:0] selected\_x, selected\_y,

input reg [3:0] weight\_x, weight\_y,

output reg [3:0] winner\_x, winner\_y

);

reg [3:0] position\_x, position\_y;

reg [3:0] new\_distance = 4'b0000;

reg [3:0] prev\_distance = 4'hf;

always@(posedge CLOCK\_50 or negedge iRST)begin

if(!iRST)begin

position\_x = 4'b0000;

position\_y = 4'b0000;

prev\_distance = 4'hf;

new\_distance = 4'b0000;

winner\_x = 4'b0000;

winner\_y = 4'b0000;

end

else begin

// Here is to find the position\_x

if(selected\_x > weight\_x)

position\_x = selected\_x - weight\_x;

else

position\_x = weight\_x - selected\_x;

// Here is to find the position\_y

if(selected\_y > weight\_y)

position\_y = selected\_y - weight\_y;

else

position\_y = weight\_y - selected\_y;

// Here is to find the new\_distance

if(position\_x > position\_y)

new\_distance = position\_x - position\_y;

else

new\_distance = position\_y - position\_x;

// Here is to update the prev\_distance and

// output winner\_x and winner\_y if new\_distance

// is larger than prev\_distance

if(new\_distance < prev\_distance)begin

prev\_distance = new\_distance;

winner\_x = weight\_x;

winner\_y = weight\_y;

end

else begin

end

end

end

endmodule

weightOptimization.sv

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Weight Optimization Block(Block 3)

\*

\* Operation:

\* To update the neuron weight and neighbors weight

\*

\* Input:

\* CLOCK\_50

\* iRST

\* up\_winner

\* update

\* start

\* network[10][10]

\* selected\_x

\* selected\_y

\* old\_weight\_x

\* old\_weight\_y

\*

\* Output:

\* updatedNetwork[10][10]

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module weightOptimization(

input CLOCK\_50, iRST,

input reg up\_winner,

input reg update,

input reg start,

input reg network[0:9][0:9],

input reg [3:0] selected\_x, selected\_y,

input reg [3:0] old\_weight\_x, old\_weight\_y,

output reg updatedNetwork[0:9][0:9]

);

reg [1:0] state = 2'b01;

reg [1:0] calculateNewWeight = 2'b01;

reg [1:0] updateNeuronWeightAndNeighbor = 2'b10;

reg [1:0] radiousForNeighborhood = 2'b11;

reg [4:0] neighborhood = 5'd10;

reg [15:0] countdown = 16'd0;

reg learning\_rate = 1;

reg [3:0] neighbor\_x, neighbor\_y;

reg [3:0] distance;

int num = 0;

reg [4:0] l;

reg [3:0] new\_weight\_x, new\_weight\_y;

reg [3:0] update\_x, update\_y;

reg [3:0] a = 4'b0000, b = 4'b0000;

reg [3:0] i = 4'b0000, j = 4'b0000, k;

reg [3:0] tempForOldWeight\_x [0:24];

reg [3:0] tempForOldWeight\_y [0:24];

reg [3:0] tempForNewWeight\_x [0:24];

reg [3:0] tempForNewWeight\_y [0:24];

reg [3:0] x, y;

always @ (posedge CLOCK\_50 or negedge iRST)begin

if (!iRST) begin

for(x = 4'b0000; x < 4'b1010; x = x + 1)begin

for(y = 4'b0000; y < 4'b1010; y = y + 1)

updatedNetwork[x][y] = 0;

end

for(l = 5'b00000; l < 5'b11001; l = l + 1)begin

tempForOldWeight\_x[l] = 0;

tempForOldWeight\_y[l] = 0;

tempForNewWeight\_x[l] = 0;

tempForNewWeight\_y[l] = 0;

end

neighborhood = 5'd10;

countdown = 16'd0;

learning\_rate = 1;

num = 0;

i = 4'b0000;

j = 4'b0000;

neighbor\_x = 4'b0000;

neighbor\_y = 4'b0000;

new\_weight\_x = 4'b0000;

new\_weight\_y = 4'b0000;

update\_x = 4'b0000;

update\_y = 4'b0000;

a = 4'b0000;

b = 4'b0000;

distance = 4'b0000;

state = calculateNewWeight;

end

else begin

if(update == 1)begin

countdown = countdown + 1;

// While start trigger as 1 then it will copy network from

// neuronInitilization Block

if(start == 1)begin

for(k = 0; k < 10; k = k + 1)begin

updatedNetwork[k][0] = network[k][0];

updatedNetwork[k][1] = network[k][1];

updatedNetwork[k][2] = network[k][2];

updatedNetwork[k][3] = network[k][3];

updatedNetwork[k][4] = network[k][4];

updatedNetwork[k][5] = network[k][5];

updatedNetwork[k][6] = network[k][6];

updatedNetwork[k][7] = network[k][7];

updatedNetwork[k][8] = network[k][8];

updatedNetwork[k][9] = network[k][9];

end

end

case(state)

// This case is to calculate the new weight

calculateNewWeight:begin

if(up\_winner == 1)begin

tempForOldWeight\_x[num] = old\_weight\_x;

tempForOldWeight\_y[num] = old\_weight\_y;

update\_x = learning\_rate \* (selected\_x - old\_weight\_x);

update\_y = learning\_rate \* (selected\_y - old\_weight\_y);

new\_weight\_x = old\_weight\_x + update\_x;

new\_weight\_y = old\_weight\_y + update\_y;

tempForNewWeight\_x[num] = new\_weight\_x;

tempForNewWeight\_y[num] = new\_weight\_y;

num = num + 1;

end

state = updateNeuronWeightAndNeighbor;

end

// This case is to update the neuron weight and neighbor weight

updateNeuronWeightAndNeighbor:begin

if(i < 4'd10)begin

if(j < 4'd10)begin

j = j + 4'd1;

if(i == 4'd9 && j == 4'd9)begin

i = 4'd0;

j = 4'd0;

num = 0;

for(l = 0; l < 25; l = l + 1)begin

updatedNetwork [tempForOldWeight\_x[l]][tempForOldWeight\_y[l]] = 0;

updatedNetwork [tempForNewWeight\_x[l]][tempForNewWeight\_y[l]] = 1;

end

end

else if(j == 4'd10)begin

i = i + 4'd1;

j = 4'd0;

end

end

end

if(updatedNetwork[i][j] == 1)begin

if(i > old\_weight\_x)

a = i - old\_weight\_x;

else

a = old\_weight\_x - i;

if(j > old\_weight\_y)

b = j - old\_weight\_y;

else

b = old\_weight\_y - j;

if(a > b)

distance = a - b;

else

distance = b - a;

if(distance < neighborhood)begin

neighbor\_x = i + update\_x;

neighbor\_y = j + update\_y;

for(l = 0; l < 25; l = l + 1)begin

if (tempForOldWeight\_x[l] == neighbor\_x && tempForOldWeight\_y[l] == neighbor\_y)begin

tempForOldWeight\_x[num] = i;

tempForOldWeight\_y[num] = j;

tempForNewWeight\_x[num] = neighbor\_x;

tempForNewWeight\_y[num] = neighbor\_y;

num = num + 1;

break;

end

end

if(updatedNetwork[neighbor\_x][neighbor\_y] == 0)begin

tempForOldWeight\_x[num] = i;

tempForOldWeight\_y[num] = j;

tempForNewWeight\_x[num] = neighbor\_x;

tempForNewWeight\_y[num] = neighbor\_y;

num = num + 1;

end

end

end

state = radiousForNeighborhood;

end

// This case is to reduce the neighbor radius

radiousForNeighborhood:begin

if(neighborhood > 5'd0)begin

if(countdown >= 16'd296)begin

neighborhood = neighborhood - 1;

countdown = 16'd0;

end

end

if(countdown >= 16'hffff)begin

learning\_rate = 4'b0000;

end

state = calculateNewWeight;

end

default: begin state = calculateNewWeight; end

endcase

end

else begin

countdown = 0;

neighborhood = 5'd10;

learning\_rate = 1;

end

end

end

endmodule

iteration.sv

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Iteration Check Block(Block 4)

\*

\* Operation:

\* To check the currentNetwork with the previousNetwork

\* are they same for 10k times iteration

\*

\* Input:

\* CLOCK\_50

\* iRST

\* resetIteration

\* currentNetwork

\*

\* Output:

\* success

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module Iteration(

input CLOCK\_50, iRST,

input startRecognition,

input resetIteration,

input reg currentNetwork[0:9][0:9],

output reg success

);

reg previousNetwork[0:9][0:9];

int iteration = 0;

reg [3:0] i, j, x, y;

reg flag = 0;

always@(posedge CLOCK\_50 or negedge iRST)begin

if(!iRST)begin

for(x = 4'b0000; x < 4'b1010; x = x + 1)begin

for(y = 4'b0000; y < 4'b1010; y = y + 1)

previousNetwork[x][y] = 0;

end

iteration = 0;

flag = 0;

success = 0;

end

else begin

// Here is to reset the iteration, success and flag to 0

// if resetIteration is trigger as 1

if(startRecognition == 1)begin

if(resetIteration == 1)begin

iteration = 0;

success = 0;

flag = 0;

end

// Here is to compare the currentNetwork and previousNetwork

// if there is a different then the flag will trigger as 1

// Also is to copy the currentNetwork into the previousNetwork

for(i = 0; i < 10; i = i + 1)begin

for(j = 0; j < 10; j = j + 1)begin

if(currentNetwork[i][j] != previousNetwork[i][j])

flag = 1;

previousNetwork[i][j] = currentNetwork[i][j];

end

end

// If the flag is trigger as 1 then it will reset the iteration

// and flag to 0

// Otherwise it will increase the iteration by 1 and if the

// iteration is equal to 10k then the success will trigger as 1

if(flag == 1)begin

iteration = 0;

flag = 0;

end

else begin

iteration = iteration + 1;

if(iteration == 10000)

success = 1;

end

$display("iteration = %d", iteration);

end

end

end

endmodule

votingSystem.sv

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Voting System Block(Block 5)

\*

\* Operation:

\* To find the winner candidatesNetwork from 3

\* different candidatesNetwork

\*

\* Input:

\* dataFromImagePostProcessing

\* candidatesNetwork

\* firstNetwork

\* secondNetwork

\* thirdNetwork

\*

\* Output:

\* voteFinish

\* winnerCandidateNetwork

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module votingSystem(

input CLOCK\_50, iRST,

input reg dataFromImagePostProcessing[0:9][0:9],

input reg candidatesNetwork[0:9][0:9],

input reg firstNetwork,

input reg secondNetwork,

input reg thirdNetwork,

output reg voteFinish,

output reg winnerCandidateNetwork[0:9][0:9]

);

reg dataIPP[0:9][0:9];

reg tempCandidatesNetwork[0:2][0:9][0:9];

reg [1:0] state = 2'b00;

reg [1:0] candidate\_1 = 2'b00;

reg [1:0] candidate\_2 = 2'b01;

reg [1:0] candidate\_3 = 2'b10;

reg [1:0] startVote = 2'b11;

reg [3:0] i, j, x, y;

reg [6:0] counter\_1 = 0;

reg [6:0] counter\_2 = 0;

reg [6:0] counter\_3 = 0;

always@(posedge CLOCK\_50 or negedge iRST)begin

if(!iRST)begin

for(x = 0; x < 10; x = x + 1)begin

for(y = 0; y < 10; y = y + 1)begin

winnerCandidateNetwork[x][y] = 0;

dataIPP[x][y] = 0;

tempCandidatesNetwork[0][x][y] = 0;

tempCandidatesNetwork[1][x][y] = 0;

tempCandidatesNetwork[2][x][y] = 0;

end

end

counter\_1 = 0;

counter\_2 = 0;

counter\_3 = 0;

state = candidate\_1;

voteFinish = 0;

end

else begin

case(state)

// This case is to copy first candidatesNetwork into tempCandidatesNetwork[0]

// and copy dataFromImagePostProcessing into dataIPP

candidate\_1: begin

if(firstNetwork == 1)begin

for(i = 4'd0; i < 4'd10; i = i + 1)begin

tempCandidatesNetwork[0][i][0] = candidatesNetwork[i][0];

tempCandidatesNetwork[0][i][1] = candidatesNetwork[i][1];

tempCandidatesNetwork[0][i][2] = candidatesNetwork[i][2];

tempCandidatesNetwork[0][i][3] = candidatesNetwork[i][3];

tempCandidatesNetwork[0][i][4] = candidatesNetwork[i][4];

tempCandidatesNetwork[0][i][5] = candidatesNetwork[i][5];

tempCandidatesNetwork[0][i][6] = candidatesNetwork[i][6];

tempCandidatesNetwork[0][i][7] = candidatesNetwork[i][7];

tempCandidatesNetwork[0][i][8] = candidatesNetwork[i][8];

tempCandidatesNetwork[0][i][9] = candidatesNetwork[i][9];

dataIPP[i][0] = dataFromImagePostProcessing[i][0];

dataIPP[i][1] = dataFromImagePostProcessing[i][1];

dataIPP[i][2] = dataFromImagePostProcessing[i][2];

dataIPP[i][3] = dataFromImagePostProcessing[i][3];

dataIPP[i][4] = dataFromImagePostProcessing[i][4];

dataIPP[i][5] = dataFromImagePostProcessing[i][5];

dataIPP[i][6] = dataFromImagePostProcessing[i][6];

dataIPP[i][7] = dataFromImagePostProcessing[i][7];

dataIPP[i][8] = dataFromImagePostProcessing[i][8];

dataIPP[i][9] = dataFromImagePostProcessing[i][9];

end

state = candidate\_2;

end

end

// This case is to copy second candidatesNetwork into tempCandidatesNetwork[1]

candidate\_2: begin

if(secondNetwork == 1)begin

for(i = 4'd0; i < 4'd10; i = i + 1)begin

tempCandidatesNetwork[1][i][0] = candidatesNetwork[i][0];

tempCandidatesNetwork[1][i][1] = candidatesNetwork[i][1];

tempCandidatesNetwork[1][i][2] = candidatesNetwork[i][2];

tempCandidatesNetwork[1][i][3] = candidatesNetwork[i][3];

tempCandidatesNetwork[1][i][4] = candidatesNetwork[i][4];

tempCandidatesNetwork[1][i][5] = candidatesNetwork[i][5];

tempCandidatesNetwork[1][i][6] = candidatesNetwork[i][6];

tempCandidatesNetwork[1][i][7] = candidatesNetwork[i][7];

tempCandidatesNetwork[1][i][8] = candidatesNetwork[i][8];

tempCandidatesNetwork[1][i][9] = candidatesNetwork[i][9];

end

state = candidate\_3;

end

end

// This case is to copy third candidatesNetwork into tempCandidatesNetwork[2]

candidate\_3: begin

if(thirdNetwork == 1)begin

for(i = 4'd0; i < 4'd10; i = i + 1)begin

tempCandidatesNetwork[2][i][0] = candidatesNetwork[i][0];

tempCandidatesNetwork[2][i][1] = candidatesNetwork[i][1];

tempCandidatesNetwork[2][i][2] = candidatesNetwork[i][2];

tempCandidatesNetwork[2][i][3] = candidatesNetwork[i][3];

tempCandidatesNetwork[2][i][4] = candidatesNetwork[i][4];

tempCandidatesNetwork[2][i][5] = candidatesNetwork[i][5];

tempCandidatesNetwork[2][i][6] = candidatesNetwork[i][6];

tempCandidatesNetwork[2][i][7] = candidatesNetwork[i][7];

tempCandidatesNetwork[2][i][8] = candidatesNetwork[i][8];

tempCandidatesNetwork[2][i][9] = candidatesNetwork[i][9];

end

state = startVote;

end

end

// This case is to check the similarity of three tempCandidatesNetwork

// and dataIPP and put into 3 different counter

// Also compares the three counter

startVote: begin

for(i = 4'd0; i < 4'd10; i = i + 1)begin

for(j = 4'd0; j < 4'd10; j = j + 1)begin

if(tempCandidatesNetwork[0][i][j] == dataIPP[i][j])

counter\_1 = counter\_1 + 1;

if(tempCandidatesNetwork[1][i][j] == dataIPP[i][j])

counter\_2 = counter\_2 + 1;

if(tempCandidatesNetwork[2][i][j] == dataIPP[i][j])

counter\_3 = counter\_3 + 1;

end

end

if((counter\_1 > counter\_2) && (counter\_1 > counter\_3))

winnerCandidateNetwork = tempCandidatesNetwork[0];

if((counter\_2 > counter\_1) && (counter\_2 > counter\_3))

winnerCandidateNetwork = tempCandidatesNetwork[1];

if((counter\_3 > counter\_1) && (counter\_3 > counter\_2))

winnerCandidateNetwork = tempCandidatesNetwork[2];

voteFinish = 1;

end

default: state = candidate\_1;

endcase

end

end

endmodule

onChipTraining.sv

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* Voting System Block(Block 5)

\*

\* Operation:

\* To find the winner candidatesNetwork from 3

\* different candidatesNetwork

\*

\* Input:

\* dataFromImagePostProcessing

\* candidatesNetwork

\* firstNetwork

\* secondNetwork

\* thirdNetwork

\*

\* Output:

\* voteFinish

\* winnerCandidateNetwork

\*

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

module votingSystem(

input CLOCK\_50, iRST,

input reg dataFromImagePostProcessing[0:9][0:9],

input reg candidatesNetwork[0:9][0:9],

input reg firstNetwork,

input reg secondNetwork,

input reg thirdNetwork,

output reg voteFinish,

output reg winnerCandidateNetwork[0:9][0:9]

);

reg dataIPP[0:9][0:9];

reg tempCandidatesNetwork[0:2][0:9][0:9];

reg [1:0] state = 2'b00;

reg [1:0] candidate\_1 = 2'b00;

reg [1:0] candidate\_2 = 2'b01;

reg [1:0] candidate\_3 = 2'b10;

reg [1:0] startVote = 2'b11;

reg [3:0] i, j, x, y;

reg [6:0] counter\_1 = 0;

reg [6:0] counter\_2 = 0;

reg [6:0] counter\_3 = 0;

always@(posedge CLOCK\_50 or negedge iRST)begin

if(!iRST)begin

for(x = 0; x < 10; x = x + 1)begin

for(y = 0; y < 10; y = y + 1)begin

winnerCandidateNetwork[x][y] = 0;

dataIPP[x][y] = 0;

tempCandidatesNetwork[0][x][y] = 0;

tempCandidatesNetwork[1][x][y] = 0;

tempCandidatesNetwork[2][x][y] = 0;

end

end

counter\_1 = 0;

counter\_2 = 0;

counter\_3 = 0;

state = candidate\_1;

voteFinish = 0;

end

else begin

case(state)

// This case is to copy first candidatesNetwork into tempCandidatesNetwork[0]

// and copy dataFromImagePostProcessing into dataIPP

candidate\_1: begin

if(firstNetwork == 1)begin

for(i = 4'd0; i < 4'd10; i = i + 1)begin

tempCandidatesNetwork[0][i][0] = candidatesNetwork[i][0];

tempCandidatesNetwork[0][i][1] = candidatesNetwork[i][1];

tempCandidatesNetwork[0][i][2] = candidatesNetwork[i][2];

tempCandidatesNetwork[0][i][3] = candidatesNetwork[i][3];

tempCandidatesNetwork[0][i][4] = candidatesNetwork[i][4];

tempCandidatesNetwork[0][i][5] = candidatesNetwork[i][5];

tempCandidatesNetwork[0][i][6] = candidatesNetwork[i][6];

tempCandidatesNetwork[0][i][7] = candidatesNetwork[i][7];

tempCandidatesNetwork[0][i][8] = candidatesNetwork[i][8];

tempCandidatesNetwork[0][i][9] = candidatesNetwork[i][9];

dataIPP[i][0] = dataFromImagePostProcessing[i][0];

dataIPP[i][1] = dataFromImagePostProcessing[i][1];

dataIPP[i][2] = dataFromImagePostProcessing[i][2];

dataIPP[i][3] = dataFromImagePostProcessing[i][3];

dataIPP[i][4] = dataFromImagePostProcessing[i][4];

dataIPP[i][5] = dataFromImagePostProcessing[i][5];

dataIPP[i][6] = dataFromImagePostProcessing[i][6];

dataIPP[i][7] = dataFromImagePostProcessing[i][7];

dataIPP[i][8] = dataFromImagePostProcessing[i][8];

dataIPP[i][9] = dataFromImagePostProcessing[i][9];

end

state = candidate\_2;

end

end

// This case is to copy second candidatesNetwork into tempCandidatesNetwork[1]

candidate\_2: begin

if(secondNetwork == 1)begin

for(i = 4'd0; i < 4'd10; i = i + 1)begin

tempCandidatesNetwork[1][i][0] = candidatesNetwork[i][0];

tempCandidatesNetwork[1][i][1] = candidatesNetwork[i][1];

tempCandidatesNetwork[1][i][2] = candidatesNetwork[i][2];

tempCandidatesNetwork[1][i][3] = candidatesNetwork[i][3];

tempCandidatesNetwork[1][i][4] = candidatesNetwork[i][4];

tempCandidatesNetwork[1][i][5] = candidatesNetwork[i][5];

tempCandidatesNetwork[1][i][6] = candidatesNetwork[i][6];

tempCandidatesNetwork[1][i][7] = candidatesNetwork[i][7];

tempCandidatesNetwork[1][i][8] = candidatesNetwork[i][8];

tempCandidatesNetwork[1][i][9] = candidatesNetwork[i][9];

end

state = candidate\_3;

end

end

// This case is to copy third candidatesNetwork into tempCandidatesNetwork[2]

candidate\_3: begin

if(thirdNetwork == 1)begin

for(i = 4'd0; i < 4'd10; i = i + 1)begin

tempCandidatesNetwork[2][i][0] = candidatesNetwork[i][0];

tempCandidatesNetwork[2][i][1] = candidatesNetwork[i][1];

tempCandidatesNetwork[2][i][2] = candidatesNetwork[i][2];

tempCandidatesNetwork[2][i][3] = candidatesNetwork[i][3];

tempCandidatesNetwork[2][i][4] = candidatesNetwork[i][4];

tempCandidatesNetwork[2][i][5] = candidatesNetwork[i][5];

tempCandidatesNetwork[2][i][6] = candidatesNetwork[i][6];

tempCandidatesNetwork[2][i][7] = candidatesNetwork[i][7];

tempCandidatesNetwork[2][i][8] = candidatesNetwork[i][8];

tempCandidatesNetwork[2][i][9] = candidatesNetwork[i][9];

end

state = startVote;

end

end

// This case is to check the similarity of three tempCandidatesNetwork

// and dataIPP and put into 3 different counter

// Also compares the three counter

startVote: begin

for(i = 4'd0; i < 4'd10; i = i + 1)begin

for(j = 4'd0; j < 4'd10; j = j + 1)begin

if(tempCandidatesNetwork[0][i][j] == dataIPP[i][j])

counter\_1 = counter\_1 + 1;

if(tempCandidatesNetwork[1][i][j] == dataIPP[i][j])

counter\_2 = counter\_2 + 1;

if(tempCandidatesNetwork[2][i][j] == dataIPP[i][j])

counter\_3 = counter\_3 + 1;

end

end

if((counter\_1 > counter\_2) && (counter\_1 > counter\_3))

winnerCandidateNetwork = tempCandidatesNetwork[0];

if((counter\_2 > counter\_1) && (counter\_2 > counter\_3))

winnerCandidateNetwork = tempCandidatesNetwork[1];

if((counter\_3 > counter\_1) && (counter\_3 > counter\_2))

winnerCandidateNetwork = tempCandidatesNetwork[2];

voteFinish = 1;

end

default: state = candidate\_1;

endcase

end

end

endmodule