Design and implementation of a single layer feed forward neural network using stand-alone architecture FPGAs-based platform

Walid A .mahmoud* **Munther h.**** **Muthana h.****

Received on: 29/ 4 /2004
Accepted on: 30/6 /2005

Abstract

A single layer feed-forward neural network are proposed and implemented using the schematic editor of the Xilinx foundation series 2.1i. First the mathematical model of the data set (weights and inputs) is presented in a matrix multiplication format. Secondly the five design stages are presented and implemented without using the finite state machine, which control the processes of the forward propagation phase, error calculation, and the training algorithm. Finally the design can be optimized to decrease the total execution time and to minimize the cost, which eventually will increase the performance and improve the function density.

الخلاصة

الشبكات العصبية feed forward ذات الطبقة الواحدة تم افتراضها وتنفيذها باستخدام تقنية FPGA الاصدار i 2.1

تم اولا " اعداد الصيغة الرياضية لمجموعة البيانات (بيانات الادخال والاوزان) بطريقة ضرب المصفوفات 0 ثانيا " تم بناء مراحل التصميم وتنفيذها بدون استخدام برامج السيطرة (finit state machine) حيث تمت السيطرة على عمليات معالجة البيانات باطوارها الثلاثة forward propagation و error caiculation و training algorithm ماديا " (pure hardware) .

اخيرا " من الممكن تحسين التصميم وذلك بتقليل زمن التنفيذ الكلي وتخفيض كلفة البناء والتي حتميا " سوف تزيد الاداء وتعزز من كثافة العمليات (function density) .

Index Terms– feed forward, neural network, FPGA, schematic editor, stand-alone.

1. Introduction

Implementing neural networks in FPGAs can be quite complicated due to the large number of multiply-accumulate operations, which require a huge number of logic devices that must be employed. Although neural networks have been implemented mostly in software, hardware design is gaining some importance especially for real time applications. Software versions have the advantage of being easy to implement, but with poor performance[1,2,3]. Hardware versions are more difficult and time consuming to implement, but with better performance than software versions. There are many constraints that must be considered when the hardware design is the choice, namely

1. The application employed (digital, analog, or hybrid);

2. The data format (fixed or floating point);

The precision of the data (number of bits). Analog design is difficult to implement but it has good performance and low cost. Digital implementation using FPGAs have good performance and allows the reconfigurable of the neural networks (NNs) topology, but generally it is slower than Application Specific Integrated Circuits (ASICs). The XC4000 series is the third generation of FPGA that offers many features and flexibility over the two pervious generations of devices. The enhancements in software consist of added and improved features for schematic entry, simulation, partitioning, placement, routing, and generation of the configuration bit file. The hardware improvement includes more versatile and powerful CLBs and IOBs. Also, design rule shrinks, to the sub-micro range, have resulted in additional and faster Logic Cell Arrays (LCAs), interconnect points, and routing paths. Other features consist of boundary scan circuit, on-chip user RAM and Check Redundancy Cycle(CRC) error checking for configuration data. The most important application fields of the XC4000 are listed below:

1- Real time image processing.
2- Real time control system.
3-Customer processors.
4-Network protocols (Ethernet controller and TCP/IP).

A field programmable gate array is in a way a Complex Programmable Logic Device (CPLD) turned inside out. The logic is broken into a large number of programmable logic blocks that are individually smaller than a Programmable Logic Device (PLD). They are distributed across the entire chip in a sea of programmable interconnections, and programmable I/O blocks surround the entire array. The FPGA-based platform can be classified into two styles of architecture as either a co-processor or as a stand-alone architecture. On the other hand, when an FPGA-based platform takes on the role of a stand-alone architecture, it becomes self-contained and does not depend on any other devices to function. In relation to a co-processor, a stand-alone architecture does not depend on a host computer, and is responsible for carrying out all the tasks of a given algorithm[4,5,1]Artificial neural networks (ANNs) are a form of artificial intelligence, which have proven useful in different areas of application, such as pattern recognition [6,7] and function approximation /prediction [4]. The most popular Neural Network is the multi-layer perceptron. The multilayer perceptron (MLP) or Multilayer feedforward is divided into three layers: the input layer, the hidden layer and the output layer, where each layer in this order gives the input to the next. The extra layers gives the structure needed to recognise non-linearly separable classes. Well known multi-layer perceptrons[8,9], have proved capable of solving various problems., but because of their relatively high complexity, they are not well suited for hardware implementation also the radial basis function networks can offer high performance, but may not have sufficient generalization power for some applications[5,10].

P.Lysaght[11] produced a feed-forward ANN designed on a fine-grained AT6000 FPGA. The system has a single layer with four neurons, each with four synapses. A finite state machine controls the dynamic reconfiguration, with a 20MHz - system

clock; each layer of ANN takes 6.5µs to produce an output. Every reconfiguration has an overhead of 10 bytes for preamble and control information. To fully reconfigure the Atmel chip takes a minimum of 808µs.

Aydogan Savran [12] realized a feedforward multilayer NN using VHDL and implemented it in Spartan II FPGA. The system has three inputs and nine neurons (3 neurons in the input layer, five neurons in the hidden layer and one neuron in the output layer). The 20 multiplication and 20 summation processes takes only 10 clock cycles to calculate it's output, where each clock period can be as low as 20ns

This paper presents an FPGAs implementation of a feed-forward artificial neural network using two project libraries XC4000x and Vertix. The Xilinx foundation series 2.1i project library is a collection of libraries assigned to a given project and consisting of the project-working library and system libraries.

The Project Manager Design Tools is the top level software module in the Xilinx development system. it provides access to all the tools we need to read a design file from a design entry tool and implement it in a Xilinx device. The Design Manager performs the following functions.

1. Organizes and manages our design implementation data.
2. Creates multiple design versions for management of design changes
3. Creates multiple implementation revisions for management of implementation strategies
4. Provides access to reports
5. Export of the design for timing simulation and programming

2. Feed forward model

The neural network that will be considered, is a feed forward with two layers, the input layer and the output layer. Input layer containing 125 input value as a vector, called data set (x1, x2…x125) and the output layer containing two output neurons, neuron 1 and neuron 2 as shown in figure 1.

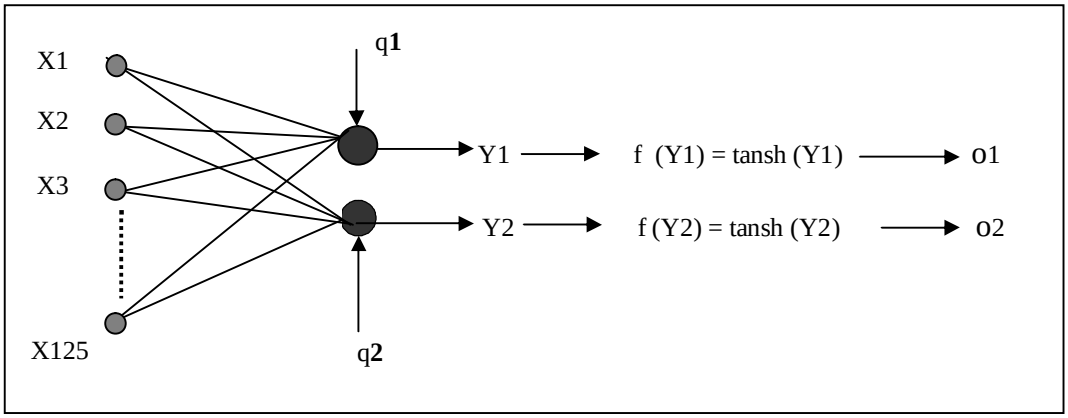

Figure 1: Artificial feed-forward neural network structure.

The results are presented at the neurons of the output layers as Y1 and Y2. The connection weights are distributed over links that joined the nodes of the two layers. Therefore,

$$Y1= \left(\sum (xi * w1i)\right)+ \theta 1 \quad \text{for i=1,2,3,.....125} \qquad (1)$$

$$Y2= \left(\sum (xi * w2i)\right)+ \theta 2 \quad \text{for i=1,2,3,.....125} \qquad (2)$$

The final sums Y1 and Y2 are the input value to the transfer function f(.). The transfer function is usually a sigmoid-shaped function having output varies between –1 and +1. This transfer function is often a hyperbolic tangent used due to it's characteristic of convergence on a correct solution with smooth effect, although it is complex to implement as a hardware. Hence,

$$o_1 = \tanh(Y1) \qquad (3)$$

$$o_2 = \tanh(Y2) \qquad (4)$$

The pervious equations can be represented as a mathematical model of two-dimension matrix multiplication, as follows:

$$\begin{pmatrix} w11 & w12 & \ldots\ldots\ldots & w1\,125 & \theta 1 \\ w21 & w22 & \ldots\ldots\ldots & w2\,125 & \theta 2 \end{pmatrix} \begin{pmatrix} X1 \\ \vdots \\ X125 \\ 1 \end{pmatrix} = \begin{pmatrix} Y1 \\ Y2 \end{pmatrix}$$

For one output neuron there is 126-multiplication process and 125-summation process, so for two output neurons we have 252-multiplication process and 250-summation process. These operations are just to find Y1 and Y2 in the forward propagation.

3. The training algorithm

The training algorithm found by[5] is quite sufficient for hardware implementation than back propagation algorithm. This algorithm depend on a set of training patterens and computes the error derivative with respect to the weight using analytical method. After calculating the error the new weights can be computed using the following formula,

$$wt+1=wt - \eta\, \partial E / \partial w \qquad (5)$$

where η is the learning rate

$\partial E / \partial w$ is calculated as follows,

$$E=1/2(\sum(o-t)2) \qquad (6)$$

and after added the small value h, (1>h>0)

using Eular formula $Eh=1/2(\sum (oh-t)2)$ (7)

$$\partial E / \partial w =(Eh-E)/h \qquad (8)$$

Introducing some assumptions and modification steps in equation (5), (6), and (8) in order to make the logical design of the neural network more optimum and take less space area, and execution time.

1. Let the value of η in equation (5) equal 1.
2. Defer the division over 2 in equation (6)to a latter stage.
3. Equation (8) will be,

$$\partial E / \partial w = (Eh-E)/2h \quad (9)$$

4. Assume that h value equal to 0.5, this will lead to

$$\partial E / \partial w = (Eh-E)/2*0.5 \quad (10)$$
$$\text{Hence } \partial E / \partial w = (Eh-E) \quad (11)$$

The benefit from steps1- 4 can be seen logically in the design procedure; i.e. step1 will cancel the multiplier device, step2-4 will cancel two dividers device. This modification and assumption will decrease the execution time up to 0.605 microsecond.

4. Block diagram of the proposed design

For simplicity the design of the feed-forward ANNs, error calculations, and it's training algorithm on pure hardware logic circuits are methodized in five stages, each stage is dedicated to provide part of the overall NNs system functionality; stage one maps the multiplication parts of equations (1) and (2) into two parallel buses (8*8)multipliers and five 256x16bits synchronize RAMs. Stage two classifies the weighted input values according to their signs (positive or negative) then sum's the two final values to find y_1 and y_2 as in equation (1) and (2). Stage three is responsible for satisfing equation (3), (4), and (6) by mapping the two actual output neuron values y_1 and y_2 in- to two look-up tables 128x8 bits RAMs. The two values o_1 and o_2 of the equations (3) and (4) are now used with the desired values t_1 and t_2 for computing the two neurons error values E_1 and E_2 as in equation (6). Stage 4 puts the final touches of the forward propagation phase, where the two error values E_1 and E_2 are summed to find out the error mean square (EMS) as in equation (6). Also in order to calculate the error derivation with respect to weight as in equation (8), the second half of memories weights (wgt+h) in stage one will take the same pervious procedures through all the predecessor stages. The final stage is stage five. This stage provides two important things, first one is to control and implement the training algorithm or in other words to update the connection weights as in equation (8). Second, it manages and synchronizes the data flow between the overall system stages operations figure 2. According to the functions types that are provided by stage five, this stage was distributed over all the four system stages as a logical circuit design or as a timer value. Explanation about the five stages operations and design will be illustrated in the following,

4.1. Stage#1

This stage is responsible for implementing the multiplication processes of 126 input values and 252 weight values in the forward phase for the two output neurons. This operation will be repeated every time when the derivative (Eh-E)/h of the stage#4 is computed. The new idea of the stage#5, is the swapping operation of the two neurons RAMs. The core of the synchronized circuit operation is the j-k flip-flop toggle properties figure 3. When the Q value of this flip-flop equal zero, two RAMs will be in read operation to compute the new weights according to equation (5), also the multiplication

processes continue multiplying the input value by the updated weight value. At the same time, the two remaining RAMs will be in write operation to record all the updated weight values of the two RAMs, which are in read operation. When the second iteration starts, and this is usually after the derivative term (EH-E) of stage#4 has been computed and ready to be used in stage#1, the Q value will be equal one. The four RAMs will exchange their roles, so the two RAMs that were in write operation will be now in read operation and vise versa. By this technique stage#1 will be in operation all the time the NNs in execution phase, except the waiting time slice to trigger from stag#4. The four zero locations in four memories are protected of their content values by embedded the control logic circuit of stage#5 with the j-k synchronized circuit. All stages take in account of their designs all the probabilities of data magnitude and signs values for multiplying, subtracting, summing, and accumulating processes, figure 4 summaries up to bit level a logic circuit design for stage#1 and the embedded logic circuit design of stage#5.

4.1.1. Data format

Accuracy has a great impact in error calculation and learning phase, so the precision must be as high as possible. The activation function shape of neuron output is tanh(y), like the sigmoid function shape, and the range of it's value between [1, -1]. Also this function has a linear region between y = 6 and y = -6 therefore, the system precision data format will be like this; ***SDDD.FFFF***, where S represents the sign value (0 for positive values, 1 for negative values). D represents the integer values [7, -7]. F represents the sixteen different decimal values. Table 1 shows some weights and input values representation

Table 1. Some weights and input values epresentation.

Decimal Data	Representation in Binary
X1= 5.3	01010101
X2= -5.3	11010101
W11= 3.25	00110100
W12= -7.9375	11111111
W22= 6.875	01101110
X126=1	00010000
W1126= -0.0625	10000001
W21= 0.125	00000010

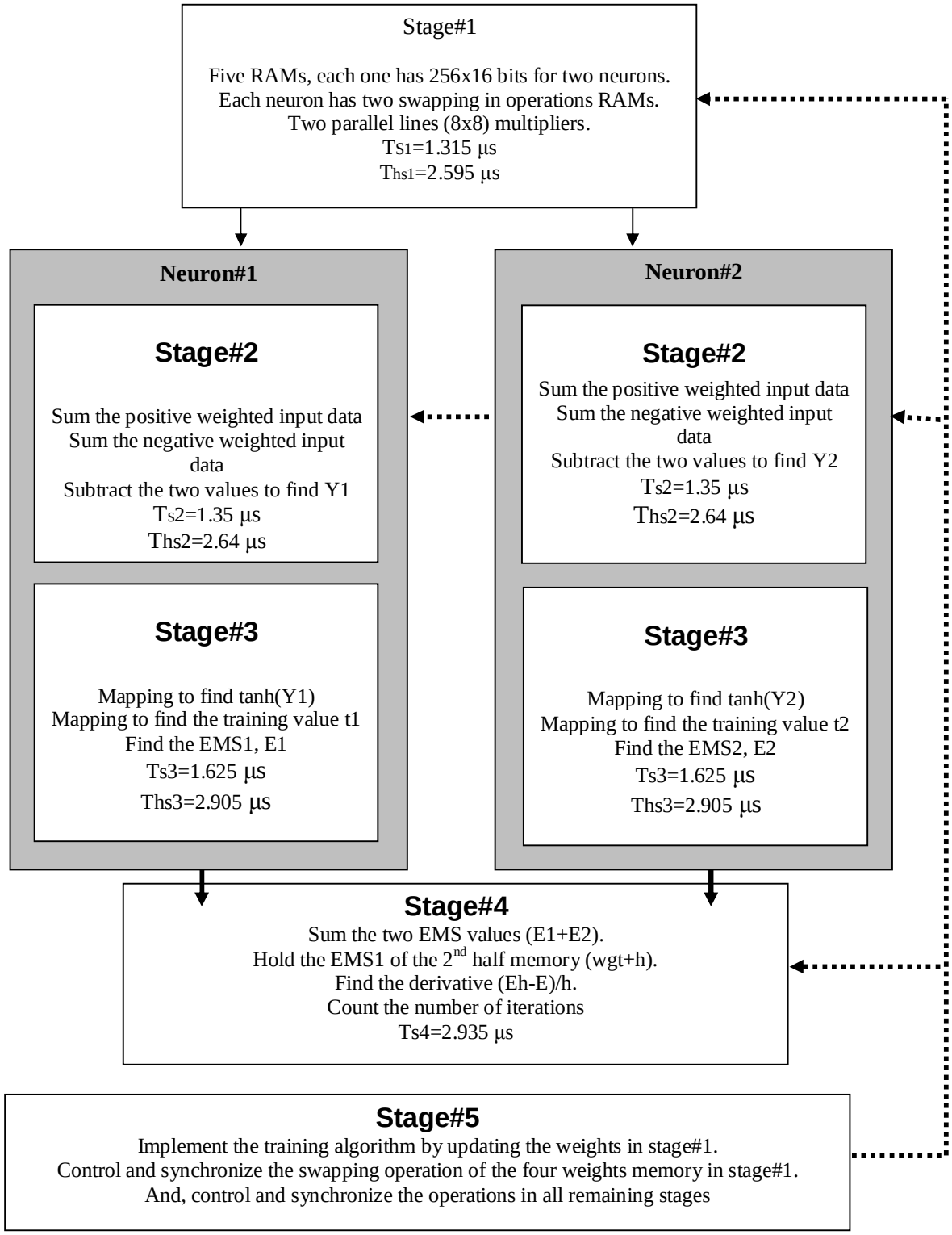

Figure 2: Block diagram of the five-stage architecture

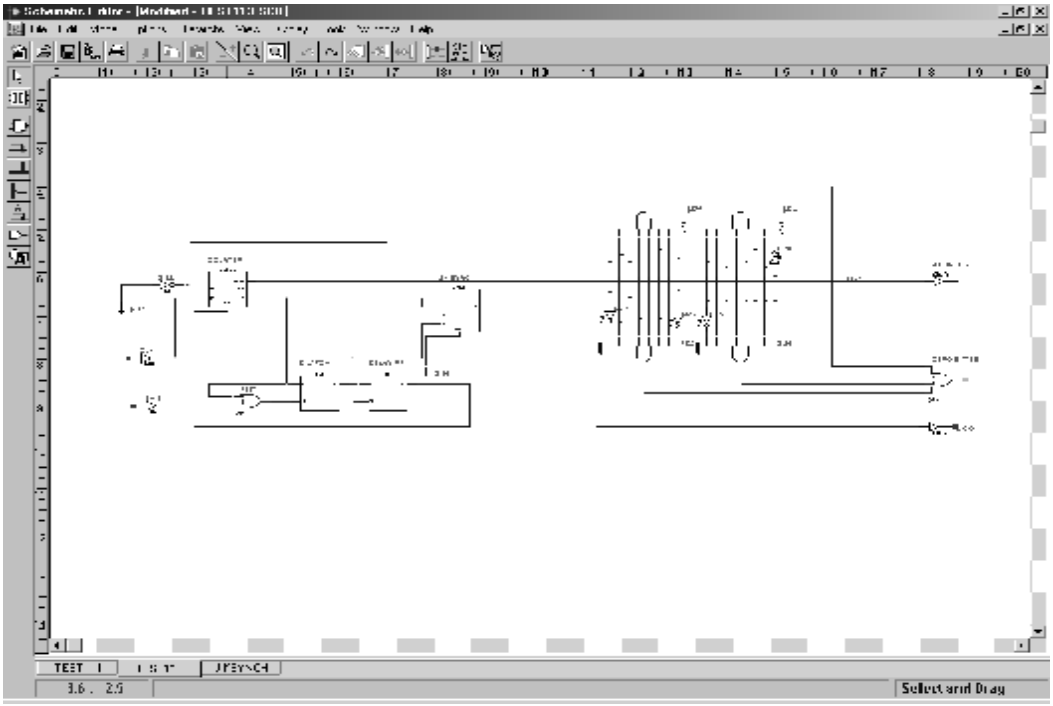

Figure 3: The j-k synchronized circuit and zero-bytes setting control circuit for stage#5.

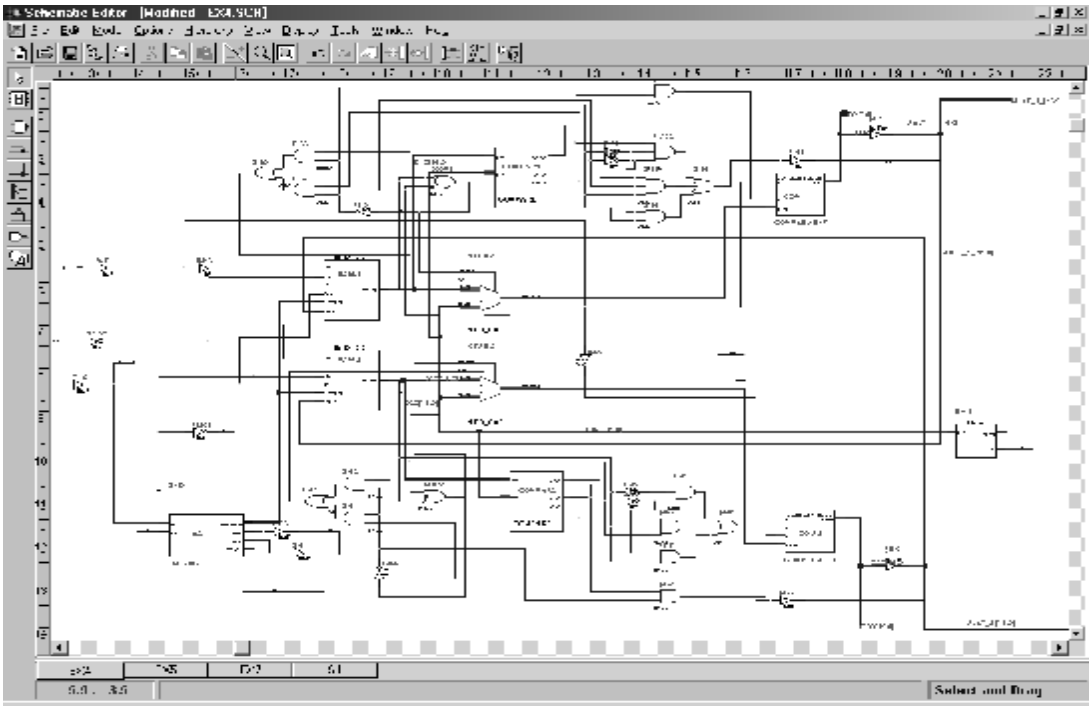

Figure 4: Stage#1 swapping circuit.

4.2. Stage#2

In order to accomplish the weighted input summation operation for the two neurons, there are four accumulators, two for each neuron, one for positive weighted input values and the other for negative weighted input values. As soon as the first zero-byte (address 126 in memory) is received and detected by the 16bits AND gate control circuit, the two opposite in signs accumulated values for each neuron will be subtract from each other. The final values of the two output Y_1 and Y_2 as in equation (1) and (2) respectively. Also the other control logic circuit in this stage is a timer circuit which resets the contents of the accumulator registers in order to repeat the same procedure of the second half memory weights (i.e., weight + h). Figure 5 shows the logic circuit design for one output neuron.

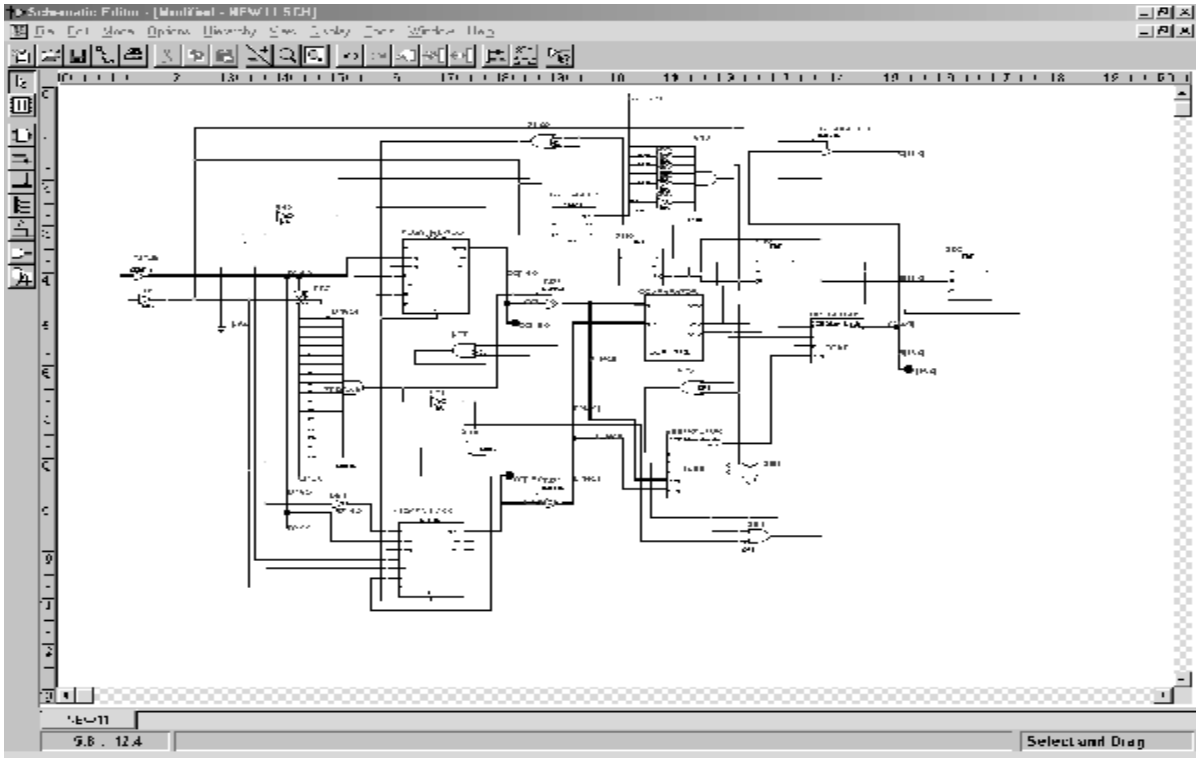

Figure 5: Weighted input accumulating logic circuit design for stage#2 and Synchronized circuit (timer) of stage#5.

4.3. Stage#3

Two look-up tables' 128x8-RAMs are used to map the actual value Y1 and Y2 to their correspondence tanh( ) function, also the two desired values t_1 and t_2 are selected from two look-up tables 16x8-RAMs. In order to reduce the design space area, one look-up table is used for each output neuron, where the signs of Y1 and Y2 are transferred to their mapped values (O1, O2), this of course due to the behavior shape of the activation function. The 16bits AND gate of the zero-byte synchronize circuit also used here with nine delays D-type flip-flops. The main function of this circuit is to permit the training value t_1 and t_2 to flow from the look-up table out put and then to calculate the errors as in equation (6). Three equations are implemented in this stage, where the final data is the Error Mean Square value. Optimization of the logic circuit design will be discussed in the next sections. The divider in figure 6 will be removed, where the most execution time of this stage is spent .

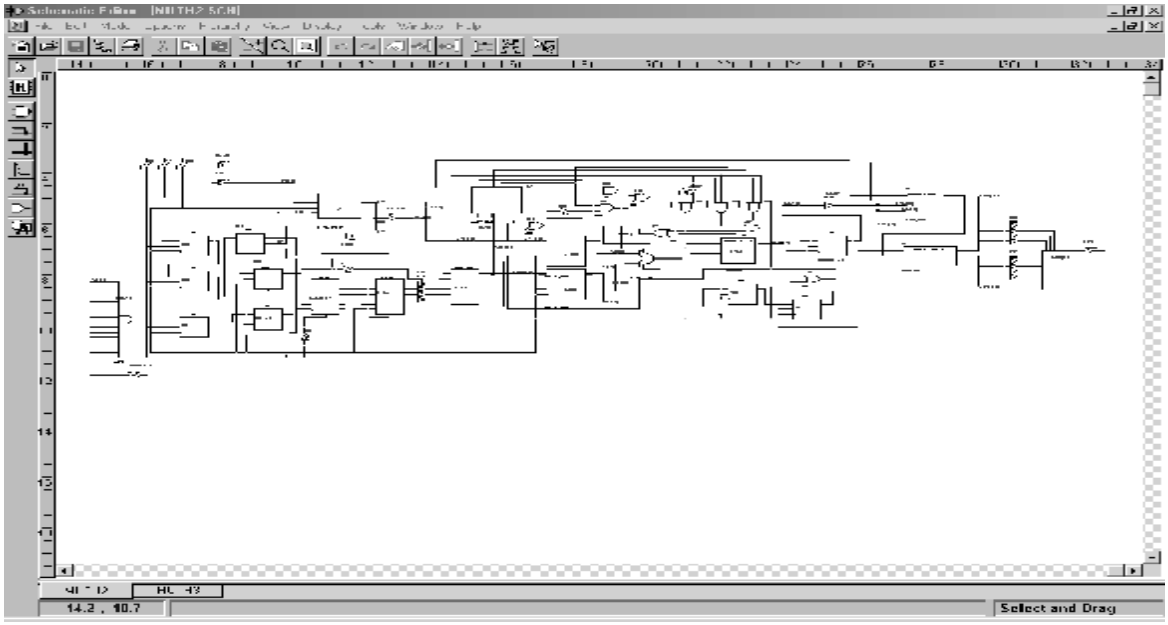

Figure 6: Stage#3 logic circuit design.

4.4. Stage#4

In this stage the two neurons EMS are added and the derivation of equation (8) is computed in order to estimate the new update weights value in stage#1. The two-control circuits of stage#5 are to synchronize the arrival time occurrence of the error value from stage#3. This stage put the final touches of the forward phase and starts the learning phase as shown in figure 7.

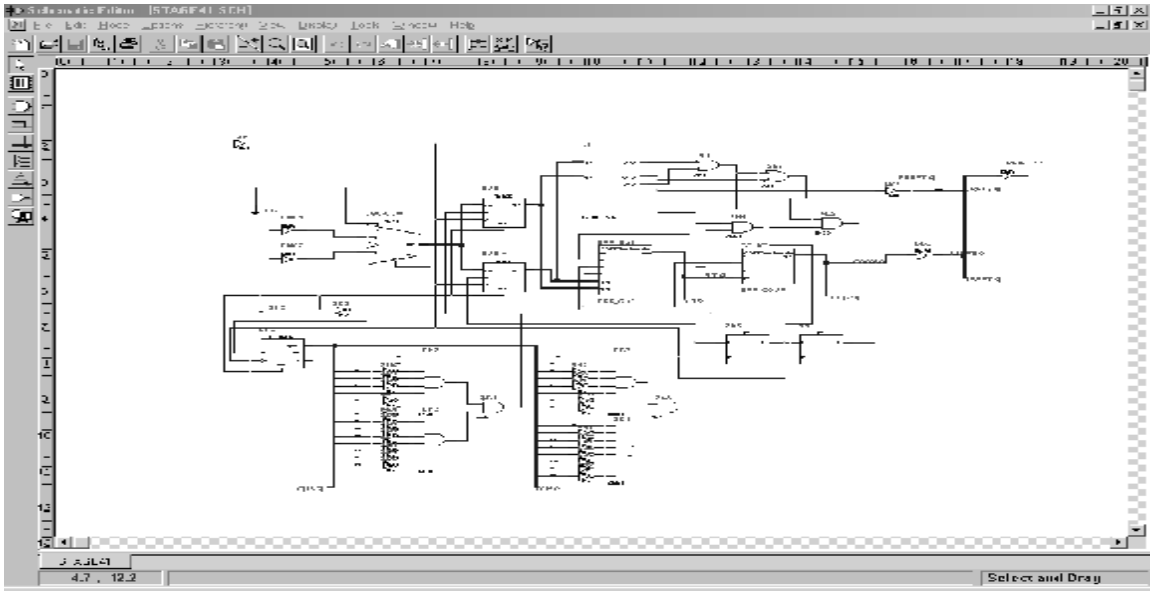

Figure 7: Stage#4 logic circuit design and the two embedded synchronize circuit

4.5. Stage#5

As mentioned previously, stage#5 functionality can be divided into two main functions, first it is responsible to implement and manage the learning phase algorithm. Secondly it synchronizes the data flow among all system stages; therefore it must be distributed. The second function operation doesn't conflict with the zero-bytes synchronized circuit which are founded in stage#2 and stage#3. The zero-bytes synchronized circuits design are related to input system parameters and hence to an input system architecture, so any change in order or number of the memories contents will come into effect in the synchronized operation of the zero-bytes circuits. Stage#5 adds more flexible fashion in the re-design system, this can be noticed when optimizing the design to decrease the execution time and minimizes the logic space area. The only changes are to modify the synchronizing time of the data flow between stages are:

1- Add/remove the delay time units (D flip-flops).

2- Increases/decreases the timer duration time.

Point 2 refers to the counters address bus, which are used as a timer unit in all stages of the neural network system design. Figure 8 explains the swapping technique for two memories out of four memories.

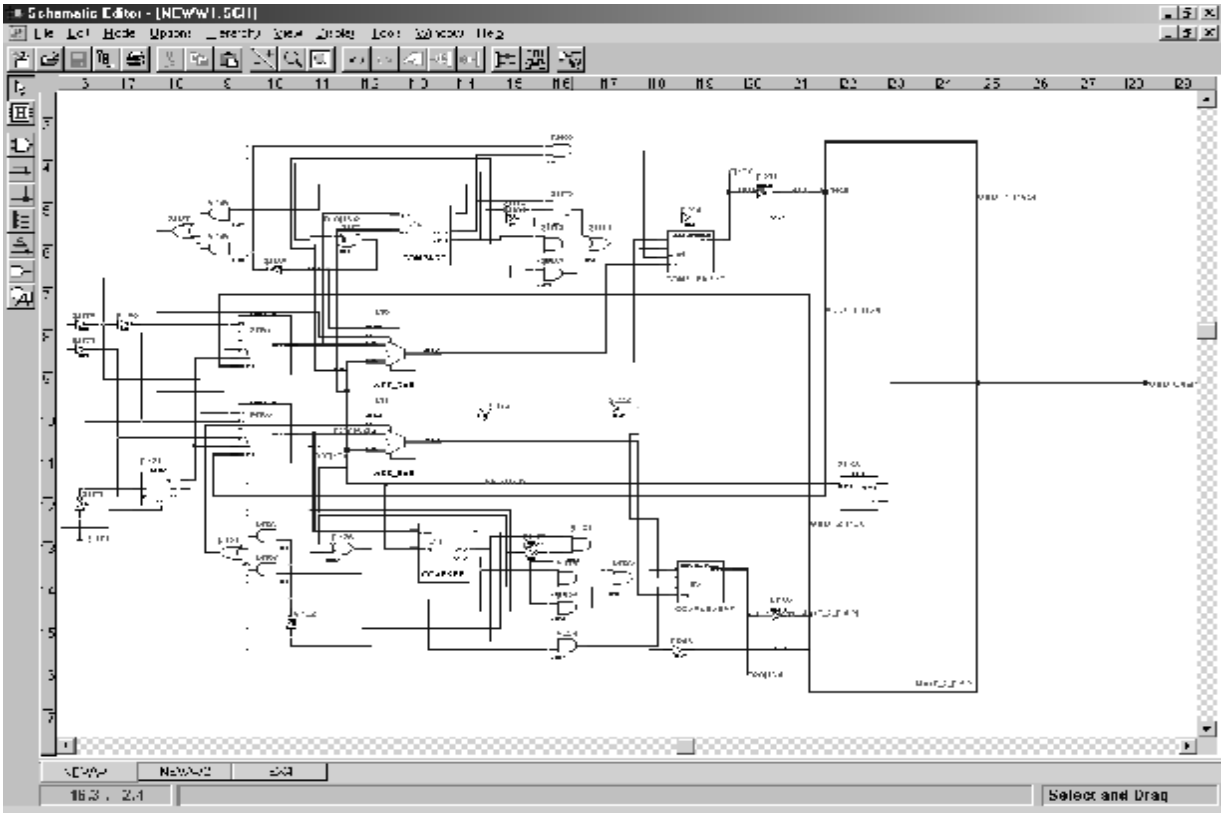

Figure 8: Swapping circuit of stage#5 applied on two memories in stage#1.

5. System stages timing

The time required for the five stages to complete their functions is illustrated in table 2. The processing time in any stage can be defined as the time spent for this stage to receive data, manipulating data, and to output data in appropriate time. Therefore,

Processing time for stage#n = time of data output (tm) – time of data input(to)

This is true for one stage and for all system stages. The control signals may be exist or may be not if the data bus taken this role. Table 2 shows that one iteration of the feed-forward system required 2.935 µs to complete the following operations:

1. Multiply and sum-up 126 input values with the first 252 weight values.
2. Multiply and sum-up 126 input values with the second 252 weight values.
3. Calculate the two neurons output values Y1, Y2, Yh1, and Yh2.
4. Mapping to find 01, 02, 0h1, and 0h2.
5. Compute the EMS values.
6. Compute the error derivative with respect to weight.
7. Updating the weights.

Table 2: the feed-forward system execution time.

Stage no.	Data input time(µs)		Data output time(µs)		Processing time
	T	T_h	T	T_h	
1	0	0	1.315	2.595	65 ns
2	1.315	2.595	1.35	2.63	35 ns
3	1.35	2.63	1.625	2.905	275 ns
4	1.625	2.905	2.935	2.935	30 ns

* T: is processing time of the first 252 weights in the first half of memories.
** T_h: is processing time of the second 252 weights in the second half of memories.

The processing time (65ns) for stage#1 refers to the time consumed by the first word addresses in two memory, so the two time values 1.315 µs and 2.959 µs are specifying the total processing time of 126 words in memories. From the table, it can be concluded that: -

If the network is learned in some few iterations i.e., when the error value is acceptable, then it is fine. But if the error value is still undesirable, the number of iterations will be increased to reach up the designer value. This is limited to a maximum value of 50, this means after fifty iterations the total execution time of the system will be 146.75µs. although this result is almost acceptable, it can be improved by applying the modification of training algorithm that mention in section 4.3.

Figures 9, 10, 11, and 12 show samples of the processing time diagram for each stage. It can be observed that the divider in figures 12, 13 takes 180ns to finish its operation. When applying the modification procedure and remove divider from stage#3 circuit, the new timing states of this stage are 1.445µs and 2.725µs instead of 1.625µs and 2.905µs respectively.

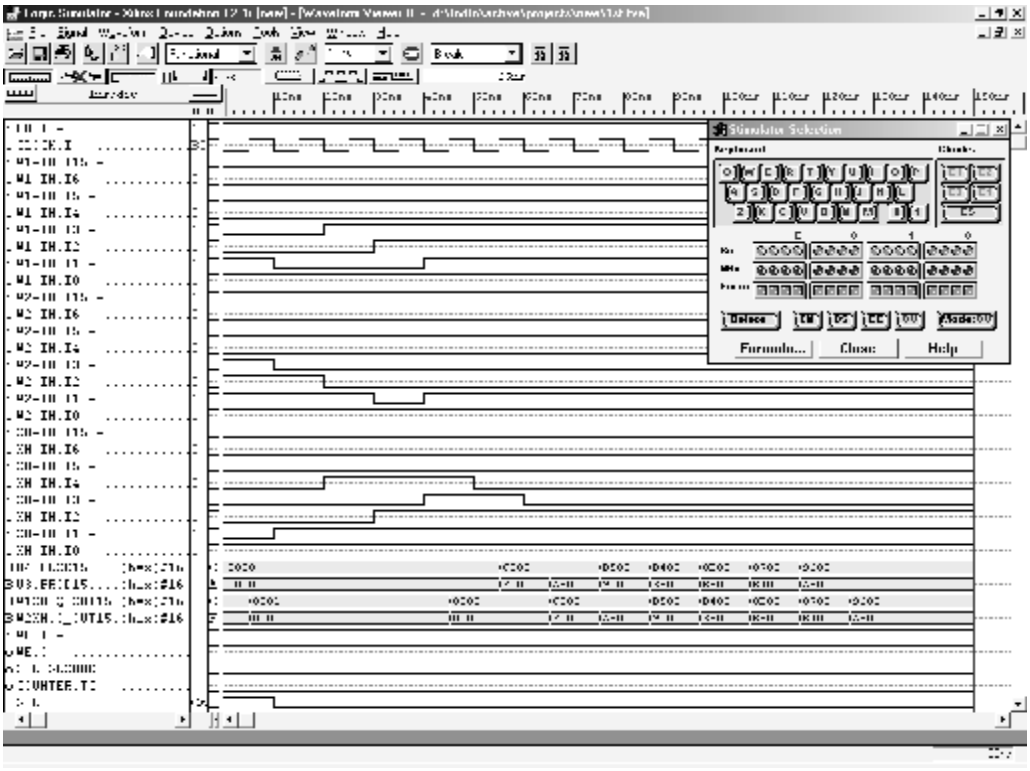

Figure 9: Stage#1 timing diagram

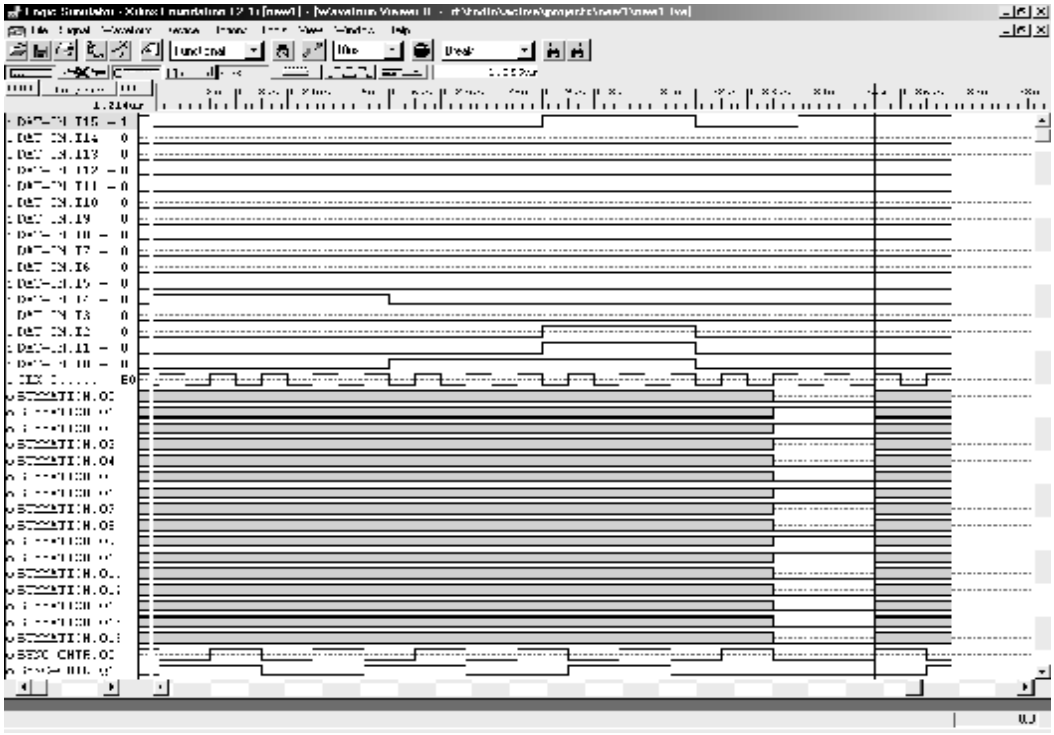

Figure 10: Stage#2 timing diagram.

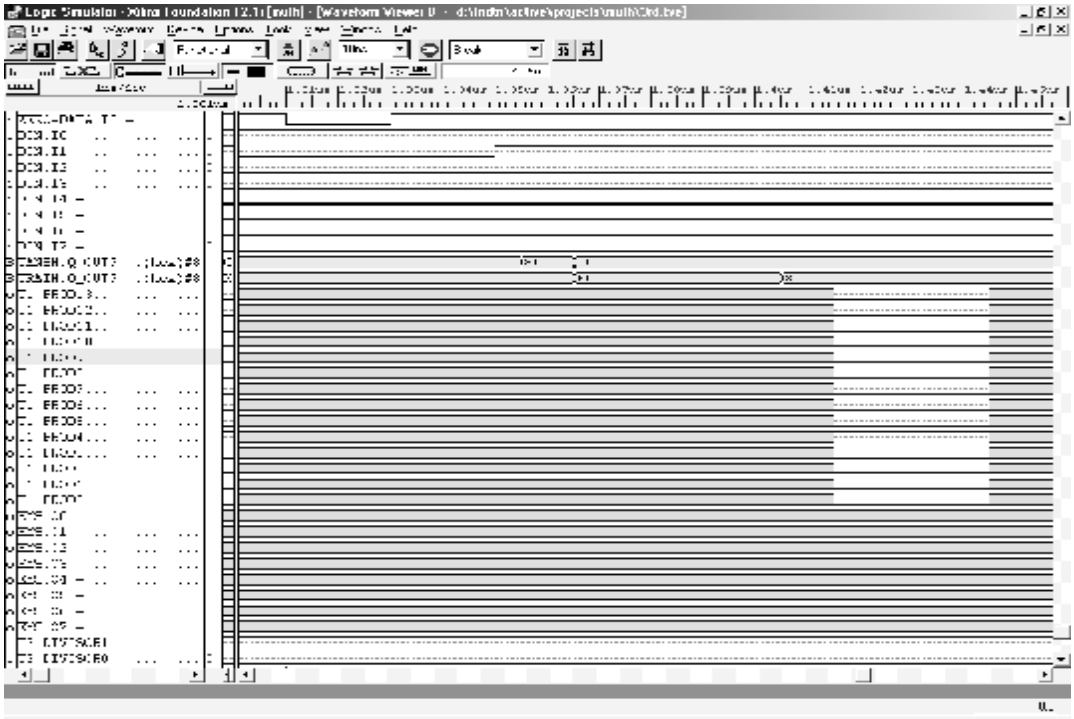

Figure 11: Stage#3 timing diagram-error square.

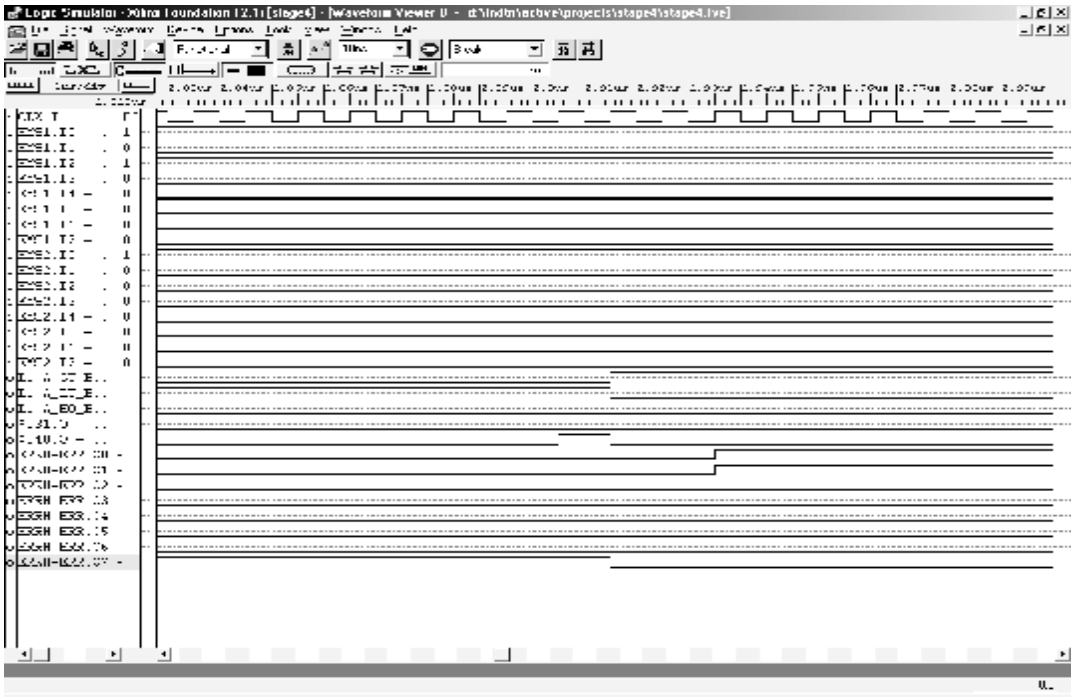

Figure 12: Stage#4 timing diagram.

6. Conclusion

This paper has presented the design and implementation of a single layer feed-forward NN by XC4000 FPGA using the schematic editor of Xilinx 2.1i. The proposed design architecture is simple, flexible, and inexpensive. All design steps are implemented, as pure hardware; i.e. not upon the finite state machine software. New control methods are used in the design procedure depending on the memory data itself or in a separate timers distributed over the five system stages. The memories swapping technique make the following:

- Update the weights,
- forward the data,
- and record the updated weights in the same time.

Comparison between the P.Lysaght [11] design and the proposed design appears, the execution time of [11] to full reconfigure is minimum of 808µs, while the execution time of the proposed design after 50 iterations is 136.25µ. this difference candidate to be larger if only four inputs is consider to the proposed design.

Aydogan Savran [12] system design takes only 10 clock cycles to calculate it's output for 20 multiplication and summations operations. This means for 252-multiplication process and 250-summation process the system approximately takes 2.52µs, where the propagation and gate delays are neglected. When comparing this result with stage#2 execution time which is 2.63µs, the [12] execution time is faster by 0.11µs, but if take the cost in consider, [12] is too more costly although the limited number of its inputs. A fully parallel network is fast but inflexible. Because; in a fully parallel network the number of multipliers per neuron must be equal to the number of connections to this neuron (imaging our system which have 252 connections). Since all products must be summed, the number of full adders equals to the number of connections to the pervious layer minus one. For example in a 3-5-1 network the output neuron must have 5 multipliers and 4 full adders, while the neurons in the hidden layer must have 3 multipliers and 2 full adders. Therefore our system design have 252 connections for two neurons are implemented in two parallel line multipliers and four accumulators. So the costly and complexity of our design are too low comparing with the [12]. The proposed design is candidate to represents the best and the fast among other designs philosophy.

7. References

1- David Sikter,"Benchmarking and feasibility study of the a J100 TM Java Processor in a real time image processing application", http://www.paricle.kth.sel~/indsey/Java

2- Kristian R" ,.A reconfigurable Architecture for Artificial Neural Networks", Thesis April 2003.

3- P.Lysaght" ,Artificial Neural Network Implementation on a Fine-Grained FPGA", http://Oak.eee.strath.ac.uk/ papers/PL- fp149.pdf

4- Felix S" ,.Towards an Artificial Neural Network Framework", http://www.kip.uni-heidelberg.de/vision

5- Attila Hidegi" ,Implementation of Neural Network in FPGAs",phd.thesis 2002.

6- Richard M" ,.Designing and Training an RBF" , lecture p1 RJM 1/08/03, http://www.kip.uni-cybernetics.de/vision

7- PHILIPP F" ,.Parallel Neural Network Training on Multi-Spert", http://citeseer.nj.nec.com/190273.htmp

8- Rolf F" ,.Codesign of Fully Parallel Neural Network for a Classification Problem", http://www.inf.pucrs.br/2000-SCI.pdf

9- G. Orr and K.-R. Miuller , "Solving the III- Conditioning in Neural Network Learning", In: Neural Networks: Tricks of the Trade, (eds), Lecture Notes in Computer Science 1524, Springer-Verlag, pp.193-206, 1998

10- A Feed Forward Neural Network for Determining a User's Location, www.edgelab.sfu.ca/publications/feed forward. pdf

11- Creating a feed forward / backpropagation neural network, mathforum.com/epigone/comp.soft-sys. matlab/prouherthol

12- On The Mapping Strategy Of A Feed-Forward Neural Network, www.lbl.gov/LBL-Publications/ Proceedings/ CHEP94 / Hardware - Architectures/Ab13Ses2. html