Design and Implementation of Two Feed forward Neural Network Models Using FPGAs Schematic Editor

Dr. Walid A .mahmoud* & Dr. Munther N.** & Muthana h. **

Received on: 27 /5 /2004
Accepted on: 16/1 / 2006

Abstract

The use Artificial Neural Networks (ANN) can be a form of Artificial Intelligence (AI). The feed forward neural network has a wide application area such as pattern recognition, image compression, and classification problem. Two models of a feed forward neural network are proposed and implemented using the schematic editor of the Xilinx FPGA foundation series 2.1i. Model-1 consists of two layers and specializes in solving a linear problem. Model-2 is a modified copy from Model-1 and consists of three layers and it's responsible for classifying the non-linear problems. Each model is designed and implemented in five stages without using the finite state machine. The flexibility, low costly, and real-time operation are the main features of the proposed design take in considered. Model-1 execution time is 2.935μs and model-2 execution time is 2.96μs, while the costs of two models are 1927 CLBs and 2017 CLBs respectively. These features compare extremely well with other existing designs with good advantages.

Index Terms– Feed forward, Neural network, FPGA, Schematic Editor, Stand-alone.

الخلاصة

تعتبر الشبكات العصبية الصناعية(ANN) إحدى أنواع الذكاء الصناعي (AI) والـشبكة(Feed Forward) تمثل واحدة من تلك الشبكات العديدة. تستخدم هذه الشبكة في تطبيقـات عديـدة منهـا ضـغط الـصور (image compression) و التصنيف (classification) وتمييز الأشكال (pattern recognition).
تم افتراض وتصميم نوعين من الشبكات العصبية باستخدام تقنية FPGA الإصدار 2.1i. النموذج الأول (model-1) متخصص لحل مسائل (linear problems) ويتكون من طبقتين (طبقة الإدخال وطبقة الإخراج). حيث تتضمن طبقة الإدخال من 126 مدخل، بينما تتكون طبقة الإخراج من وحدتي إخراج عصبيتين.
النموذج الثاني (model-2) هو نسخة معدلة من النموذج الاول ومتخصص بحل مسائل (non-linear problems)، حيث تم إضافة طبقة أخرى ثالثة تتضمن وحدة إخراج عصبية واحدة لتصبح طبقة الإخراج السابقة في النموذج الأول طبقة مخفية (hidden layer) في النموذج الثاني.
سرعة التنفيذ، الكلفة القليلة، والمرونة العالية من الأمور المهمة التي تمت مراعاتها أثناء التصميم. حيث صمم النموذجان ونفذا بواسطة خمسة مراحل. لم يستخدم في مراحل التصميم الخمسة برامج السيطرة (finite state machine)، حيث تتم السيطرة والمزامنة بين مراحل التصميم من خلال الكيان المادي نفسه.
سرعة معالجة البيانات في النوذج الاول كانت 2,935 مايكرو ثانية وبكلفة 1927 CLBs ، بينما سرعة معالجة البيانات في النموذج الثاني كانت 2,96 مايكرو ثانية وبكلفة 2017 CLBs .
من خلال مقارنة التصميمان مع بعض التصاميم الحديثة المتعلقة بنفس نوعية الشبكة، وجدنا أن كلفة وسرعة ومرونة التصميم المفترض هي الأمثل دائما"، مما يرشحه لان يكون الأفضل من بين فلسفة التصاميم الأخرى.

*Department of Electrical Engineering University of Baghdad Baghdad, Iraq
**Department of Computer Engineering University of Al-Mustansiriya Baghdad, Iraq

1. Introduction

The first theories of Neural Networks (NNs) had been introduced early 20th century. The idea of neural networks was more theoretical than applicable. So early development was slowly. Later, the revolution in computer fields lead to development new types of networks. Neural networks are used in the field of AI, are massively parallel computation systems that are based on simplified models of the human brain. Some rules must exist for evaluating whether a problem is suitable for a NN implementation or not. There must be an example dataset of the problem in order to be able to train the network. There are many training algorithms depending on the type of NN and its application field where, the training algorithm which is suitable for one type of networks, it is not for other types or even other applications [1-3]

2. Feedforward Neural Network Architecture

Feedforward Neural Network (FFNN) can be classified into two types according to their functionality.

2.1. Single Layer Feedforward (SLFF) Model

This model consists of two layers (input layer and output layer). The single layer perceptron or SLFF can only make classifications corresponding to a straight line or hyperplane in the pattern space, see figure 1 below. This means for instance that it is not possible to classify the non-linear XOR binary function. The input data for NNs are represented using feature vectors. Each element in the vector corresponds to a feature of the input [4].

2.2. Multi Layer Feedforward (MLFF) Model

Building on the algorithm of the simple Perceptron, the MLFF network model not only gives a multi layer perceptron structure for representing more than two classes, the extra layers gives the structure needed to recognise non-linearly separable classes. The MLFFs are more convenient for network classification as explained in figure 2. Such neural networks with supervised error correcting learning are used to approximate (synthesis) a non-linear input-output mapping from a set of training patterns [5]. The basic feedforward network performs a non-linear transformation of input data in order to approximate the output data. The number of input and output nodes is determined by:

1. The nature of the modeling problem being tackled.
2. The input data representation and the form of the network output required.
3. The number of hidden layer nodes is related to the complexity of the system being modeled. and thus creating an n-dimensional feature space. Feature space is easiest to visualise in the 2-dimensions. The input patterns can be drawn on the graph as or encapsulating the different pattern sets with one perceptron. It is only possible to distinguish between two pattern classes, with the visual representation of a straight separation line in pattern space [4]. A number of papers have shown that a two-layered feedforward network has the ability to approximate any non-linear continuous function to an arbitrary degree of exactness, provided that the hidden layer contains sufficient nodes. The problem of determining the network parameters (weights) is essentially a non-linear optimization task. The back-propagation method, which is a distributed gradient descent technique, is the most popular training algorithm but it is more sophisticated for hardware implementation [6-9].

4. Implementation Methods

4.1 Training Algorithms

Training or learning algorithm is a procedure that applied on the network in order to reach to the desired output with minimum error. Note that the number of connections is higher than the total number of nodes. Both numbers are chosen based on the particular application and can be arbitrarily large for complex tasks.

Although MLFF are more convenient for network classification, they introduce a new problem due to the backpropagation. The network is not guaranteed to find convergence, where the risks ending up in a situation where it is impossible to learn to produce the right output. This state is called a local minimum, also the backpropagation-training algorithm possesses another disadvantages, which deters some designers from using it. A major disadvantage of implementing the backpropagation algorithm on a FPGA is that pipelining of the algorithm on a whole cannot occur during training. This problem arises due to the weight update dependencies of backpropagation, and as a result, the utilization of hardware resources dedicated to each of the neural network's layers is wasted [12,13]. Next section a new training algorithm, which is more suitable for hardware implementation will be present and explained.

4.2 Problem Formulation

Attila [1] present new and simple training algorithm for a FFN. This simple algorithm is more convenience for hardware implementation than a regular backpropagation algorithm. Where, a large number of datasets (patterns) are presented at the input layer and the network adapts the connection weights according to these patterns. The origin of these terns may be an image. Each set of images used is called training set, so a two-dimensional image can be described as a matrix of dimension P*N (usually in pixels), whereby N is the number of original feature vectors and P is the number of patterns [5,14-17].

The neural network that will be considered, is a feed forward with two layers, the input layer and the output layer.

Let 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 3. The results are presented at the neurons of the output layers as Y_1 and Y_2. The connection weights are distributed over links that joined the nodes of the two layers. Therefore,

$$Y_1 = (\Sigma\ (\ X_i\ *\ w_{1i}\ )) + \theta_1 \quad (1)$$
for i=1,2,3, …..125

$$Y_2 = (\Sigma\ (\ X_i\ *\ w_{2i}\ )) + \theta_2 \quad (2)$$
for i=1,2,3, …..125

The final sums Y_1 and Y_2 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 [18]. Hence,

$$O_1=\tanh(Y_1) \qquad (3)$$

$$O_2=\tanh(Y_2) \qquad (4)$$

The pervious equations 1 and 2 can be represented as a ***mathematical model*** of two-dimensional matrix multiplication if one input pattern is considered as follows:

$$\begin{bmatrix} w11 \ w12 \ldots\ldots w1\,125 \ \theta 1 \\ w21 \ w22 \ldots\ldots w2\,125 \\ \theta 2 \end{bmatrix} \qquad (5)$$

$$\begin{pmatrix} X1 \\ \vdots \\ 1 \end{pmatrix} \quad \begin{pmatrix} Y_1 \\ Y_2 \end{pmatrix}^{1} \qquad (5)$$

For two input patterns there is 252-multipliction process and 250-summation process at one output neuron, so for two output neurons, we have 504-multiplication process and 500-summation process.

The training algorithm of [1] is quite sufficient for hardware implementation than back propagation algorithm. This algorithm computes the error derivative with respect to the weight using analytical method. After calculating the error, new weights can be computed using the following formula,

$$w_{t+1}=w_t \ \eta \ \partial E/\partial w \qquad (6)$$

Where η is the learning rate

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

$$\partial E/\partial w=(E_h-E)/h \qquad (7)$$

and

$$E_{(p)}=1/2(\sum(o-t)^2) \qquad (8)$$

Where; E_h is the error that is created after added a small value (h) to the weights and re-process to find the two neuron outputs and hence the two neurons errors (E_h). t, is the desired or training output

5. Related Designs

Table 1 summarizes the cost and the speed for the four related designs. These results will be compared and discussed with the proposed design results. The cost (if available) is measured by the number of CLBs and IOBs while the speed represents the execution time of the design for one iteration i.e.; the delay time required to process the input and present the output.

Design Name, year	No. of Inputs	No. of neurons	No. of Layers	Cost		Speed/one iteration (µs)	notes
				CLBs	IOBs		
non-RRANN for XOR, 2003[2]	2	3	3	1239	*_	47.8	Use Virtex-E
non-RRANN for XOR, 2003[2]	2	3	3	8334.75	–	580	Use Virtex-II
RWC, 1998[19]	20	3	3	–	–	8	VLSI, 10MHz For 126 input, the speed=48 µs
Atmel AT60005 for XOR, 1996[3]	2	3	3	1216.5	–	17.6	Clock = 20MHz

*- : Unavailable

Table 1: Comparison the four similarity design

*- : Unavailable

6. Block Diagram of the Model-1 FFNN Design

Although many neural networks have been implemented in hardware, no researcher could produce the logic circuit details, which help the new designer for adding, improving and even invention in

the current design. Our designs produce the ideas over many steps. Each group of design steps are called "stage", and each stage is responsible to implement part of the over all system operation.

The first model consists of five design stages explained in figure 4. This figure gives us summary review about the main design architecture.

7. Block Diagram of the Model-2 FFNN Design

We can refer to a three-layer FFNN as Model-2. The input layer of this model is consist of 126 input values, the hidden layer include two neurons, and one neuron in the output for non-linear classification problems. The hidden layer was in fact the output layer in Model-1. The new modification in the model-1 FFNN is only the output neuron, which is added at the output layer where, the two outputs (O_1 and O_2) becomes an input to this layer. The new output formula will be,

$$Y = O_1W_{11} + O_2W_{12} + \theta \quad (9)$$

and $O = \tanh(y) \quad (10)$

Since this model has only one output neuron, stage#4 will also modify its circuit design according to one output neuron and hence one error will be calculated.Up to now the five stages of the two models are designed. Next section the two model stages will be tested, implemented, and optimized.

. Calculation of the Model-1 Execution Time and Cost

The processing time of the model-1 is the accumulated time for its individual stages as will be computed in the table 2 below.

T: processing time of the first 252 weights in the first half memories.

T_h: processing time of the second 252 weights in the second half memories

Table 2: model-1 FFNN execution time

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

T_h: processing time of the second 252 weights in the second half memories.

The design cost of this model was summarized in table 3. This table represents the report result of the Place & Route step implementation process.

Table 3: Model-1 design cost

Stage no.	CLB	IOBs	Total stages	
			CLB	IOBs
#1	1206	756	1206	756
#2	110	392	220	784
#3	201	250	402	500
#4	99	331	99	331
#5	Already computed in each stage		—	—
Total cost			1927	2371

9. Calculation of the Model-2 Execution Time and cost

Model-2 architecture match the first one with 80% where, the change was done only by embedding stage#2 model-1 in stage#2 model-2. Stage#2 for this model will add new processing time, this time is created as follows,

Time (stage#2 Model-2) = T_{S2} = T_{S1}+ (stage#2 Model-2 processing time)

stage#2 Model-2 processing time = 35ns + 25ns = 60ns

T_{S2} = 1.315μs + 60ns = 1.375 μs

Where, the 60ns are the macro device of stage#2 model-1 processing time (35ns). The timing diagram is explained in figure 6. Also, the total time required for the second halves memories (128-253)10 is,

Ths2 = Ths1+ (stage#2 Model-2 processing time)

Th_{S2} = 2.595 μs + 60ns

Th_{S2} = 2.655 μs

In the same way, the remaining stages time will be computed. Table 4 summarizes the spent time for the four stages in model-2.

Stage no.	Data input time(μs)		Data output time(μs)		Processing time (ns)
	T	Th	T	Th	
#1	0	0	1.315	2.595	65
#2	1.315	2.595	1.375	2.655	60
#3	1.375	2.655	1.65	2.93	275
#4	1.65	2.93	2.96	2.96	30

Table 4: model-2 FFNN execution time

Design cost for this model can be easy conclude from the following,

Model-2 Design Cost = Model-1 Design Cost + new layer cost

The new layer cost is the output layer that designed in stage#2. Table 5 contents summarizes the cost report resulted from Place & Route implementation process. We can notice that change is appear only in stag#2 and stage#3 where, the hidden and out put layer are included in their.

*Department of Electrical Engineering University of Baghdad Baghdad, Iraq
**Department of Computer Engineering University of Al-Mustansiriya Baghdad, Iraq

Table 5: Model-2 design cost

Stage no.	CLB	IOBs	Total stages	
			CLB	IOBs
#1	1206	756	1206	756
#2	621	1043	621	1043
#3	201	250	201	250
#4	99	331	99	331
#5	Already computed in each stage		—	—
Total cost			2017	2380

According to tables 3 and 5 results, the change of CLBs cost is relatively larger than IOBs cost change. The reason of this back that RAM and multiplier consumes a CLBs much more than other devices where, stage#2 Model-2 circuit consumed pair of 8bit*8bit multipliers and pair of look-up tables RAMs.

The percent of CLBs in XC4005XL platform of the table 3 is,

1927/20736 =9.293% while,

the percent of CLBs in XC4005XL platform of the table 5 is,

2017/20736 = 9.727%

These two conclusion values are too important, when we want to develop the proposed design in the future or even for more run-time optimization.

10. Discussion and Conclusions

This paper has presented the design and implementation of the two models FFNNs by XC4005XL FPGA using the schematic editor of Xilinx 2.1i.

Two models design steps were implemented, as pure hardware; i.e. not based the finite state machine software. The main benefits obtained from stages based design idea is the ***flexibility***. This was achieved through the following points:

1. This feature was satisfied in model-1 design when, with some of

modification applied on one stage to obtain the new model-2 design.

2. The range of inputs can vary from 2-126 by justifying the zero-bytes locations and delay counter in stag#4 depending on type of application.
3. The design implementation and verification becomes more interactive and easy for debugging.
4. Requirement of adding/canceling layers or neurons are local stage dependent. The proposed design architecture in addition of being flexible, high-speed run time, and is inexpensive. These features can be discussed through comparison with a related works as shown in table 6 below

Table6: Comparison the two models with a similarity design

Design Name, year	No. of Inputs	No. of neurons	No. of Layers	Cost		Speed/one iteration (μs)	notes
				CLBs	IOBs		
Model-1 2004	126	2	2	1927	2371	2.935	Use XC4005XL
Model-2 2004	126	3	3	2017	2380	2.96	Use XC4005XL
non-RRANN for XOR, 2003	2	3	3	1239	*–	47.8	Use Virtex-E
non-RRANN for XOR, 2003	2	3	3	8334.75	–	580	Use Virtex-II
RWC, 1998	20	3	3	–	–	8	VLSI, 10MHz For 126 input, the speed=48 μs
Atmel AT60005 for XOR, 1996	2	3	3	1216.5	–	17.6	Clock = 20MHz

* - : Unavailable

The two models cost has in minimum value, the speed of them can rise up by adding more multiplier devices. These additional multipliers will decreases the execution time as follows:

The 126 input values take 1.315**μs** to finish the multiplication process for one multiplier per neuron, so for two parallel multipliers per neuron, the 126 input values will divided to two equally halves, each one contains 63 input values.

The first and second 63 input values will be multiplies in 0.6575μs can be

Concluded that stage#1 processing time will decreases to a new value for each additional multiplier per neuron, as in the table 7.

Table 7: Stage#1 Execution Time after modification

No. of Multiplier Per Neuron	**No. of dataset Groups**	**No. of Input per Group**	**Stage#1 Execution Time(μs)**
1	1	126	1.315
2	2	63	0.6575
3	3	42	0.4383
6	6	21	0.2193
7	7	18	0.185
9	9	14	0.1161

The new processing time = (stage#1 processing time)/(No. of multiplier per neuron)

Also, Total run time = total run time – (stage#1 execution time – new processing time)From table 7, it can observed the following, when the No. of multipliers exceeds 7, the new stage processing time will not be proportionate with the new stage complexity. This table can be reference for any FFNN design has the same range of input values where, the execution time and cost are the main issues.

11- References

Hidegi, A. "Implementation of Neural Network in FPGAs", A Ph.D. Thesis in Electrical Engineering La Trop Univ. Australia, 2002.

2- Kristian, R., "A reconfigurable Architecture for Artificial Neural Networks", A M.Sc. Thesis. University of Gulph, April 2003.

3- Lysaght, P., "Artificial Neural Network Implementation on a Fine- Grained FPGA", , 2002.

4-Nielsen, F., "4I Neural Networks – Algorithms and applications", **http:// www.gtfdrr.org.ygdrr.pdf, 2001**

5-Jansuz S., "A self Organization Learning Array and its Hardware-Software Co-Simulation", http://www.inf.pucrs.br/2000-ASCI.pdf, 2003

6- Donnali F., "Feed-Forward Neural Network Method", www.edgelab.sfu.ca/publications/feed_forward.pdf , 1998

7- Sajiee, Q., "On The Mapping Strategy Of A Feed-Forward Neural Network", www.lbl.gov/LBL-Publications/Proceedings/CHEP94/Hardware-Architectures/Ab13Ses2.html, 1999

8- Raaul, F., "Feed-Forward Neural Networks", laxmi.nuc.ucla.edu:8888/Teachers/sgambhir/Published_Trays/Test_avni/slide, 2001

9- Ramoo, R., "Neural Network Documentation" http://www.wolfram.com/company/terms.html, 2001

10- RICHARD L., "Field Programmable Gate Arrays", published by PIR Prentice-Hall, 1993

11- John F., "Digital Design", third edition, Prentic-Hall, Inc., 2000

12- Khurram W., " A Mixed-Mod Design for a Self Programming Chip For real-time estimation, Prediction, and Control", pro. 43rd IEEE Midwest

Symp. On Circuit and System, Lansing MI, Aug 8-11, 2000

13- Hu, Y. "Multisensor Inversion with High-Performance FPGA Computation", International Conference on Field-Programmable Technology, pp74-79, Tokyo, Japan, Dec. 15-17, 2002

14- Alan M., "Study, Implementation and Evolution of the Artificial Neural Networks Proposed by", http://www.teuschers.ch/christof, 2000

15-Yun, B., "Fault Tolerant Training of Feedforward Neural Networks", Proc. Natil Sci. Counc. ROC(A) vol. 23, No. 5, pp, 599-608, 1999

16- Malinowski, A., "Capabilities and Limitations of Feedforward NNs with Multilevel Neurons", In Proceeding of the IEEE International Symposium on Circuits and Systems, volume 1, pp 131-134, Seattle, Wash., USA, April 1995.

17- Bar, R., "Principle Components Analysis-Image Decomposition", http://www.inf.pucrs.br/~educ.pdf, 2004

18- Barratta, D., "Microelectronic Implementation of Artificial NN", http://www.umel.fee.vutbr.cz/~hub/eds/PDF/barat3.pdf, 2000

19-Aydogan, S., "Hardware Implementation of a Feedforward Neural Network Using FPGAs", http://www.eleco.emo.org.tr/ora/B.htm, 2003

20-Nielsen, F., "4I Neural Networks – algorithms and applications", **http:// www.gtfdrr.org.ygdrr.pdf, 2001**

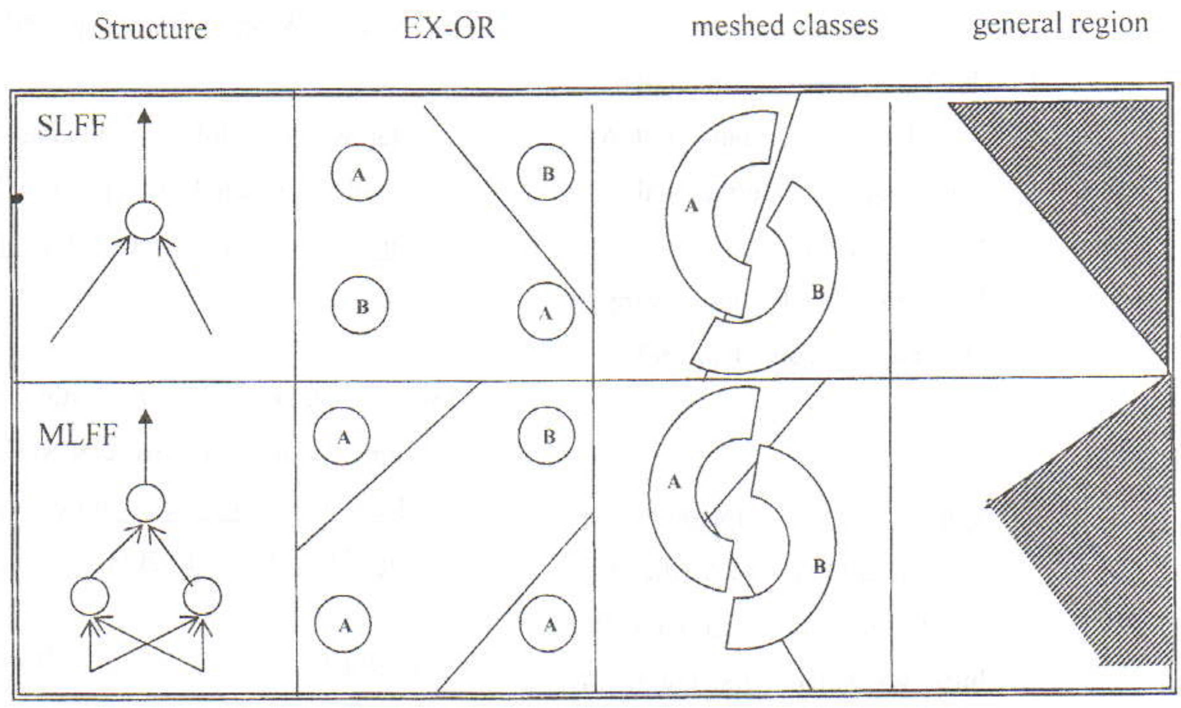


Figure 2: Decision regions (IEEE SSP April 1987)

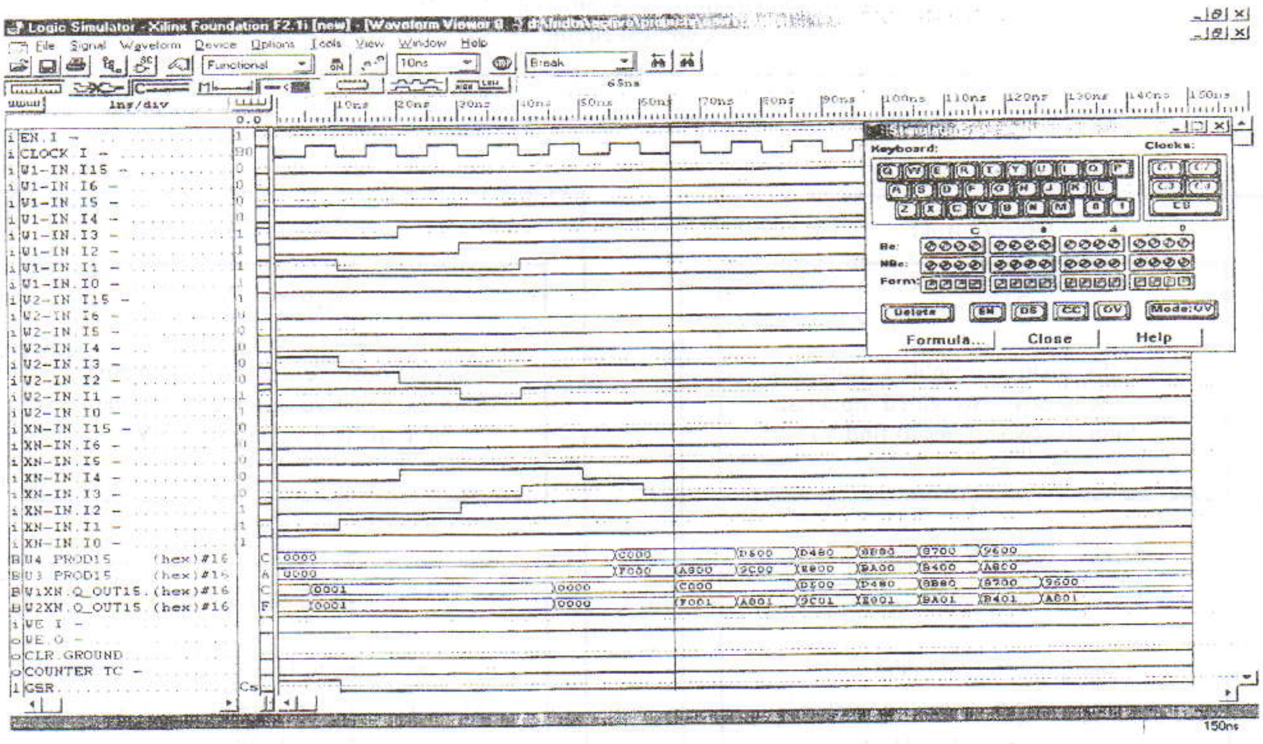


Figure 5: Waveform viewer for Model-1 input signals.

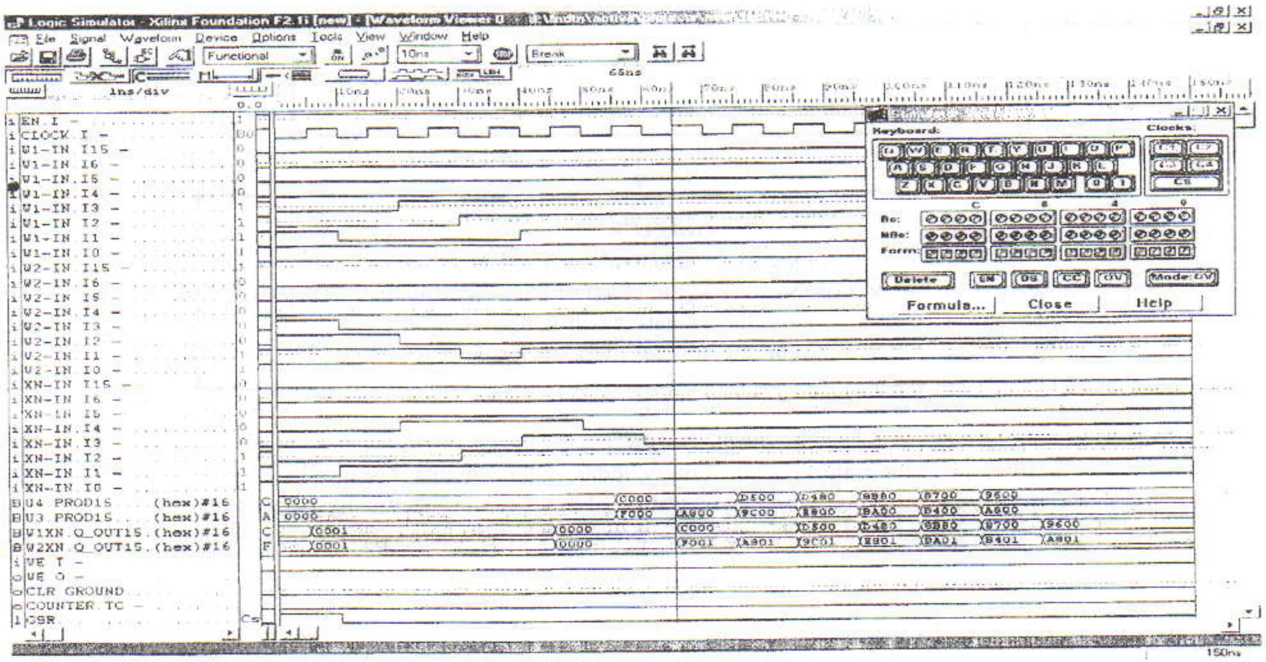
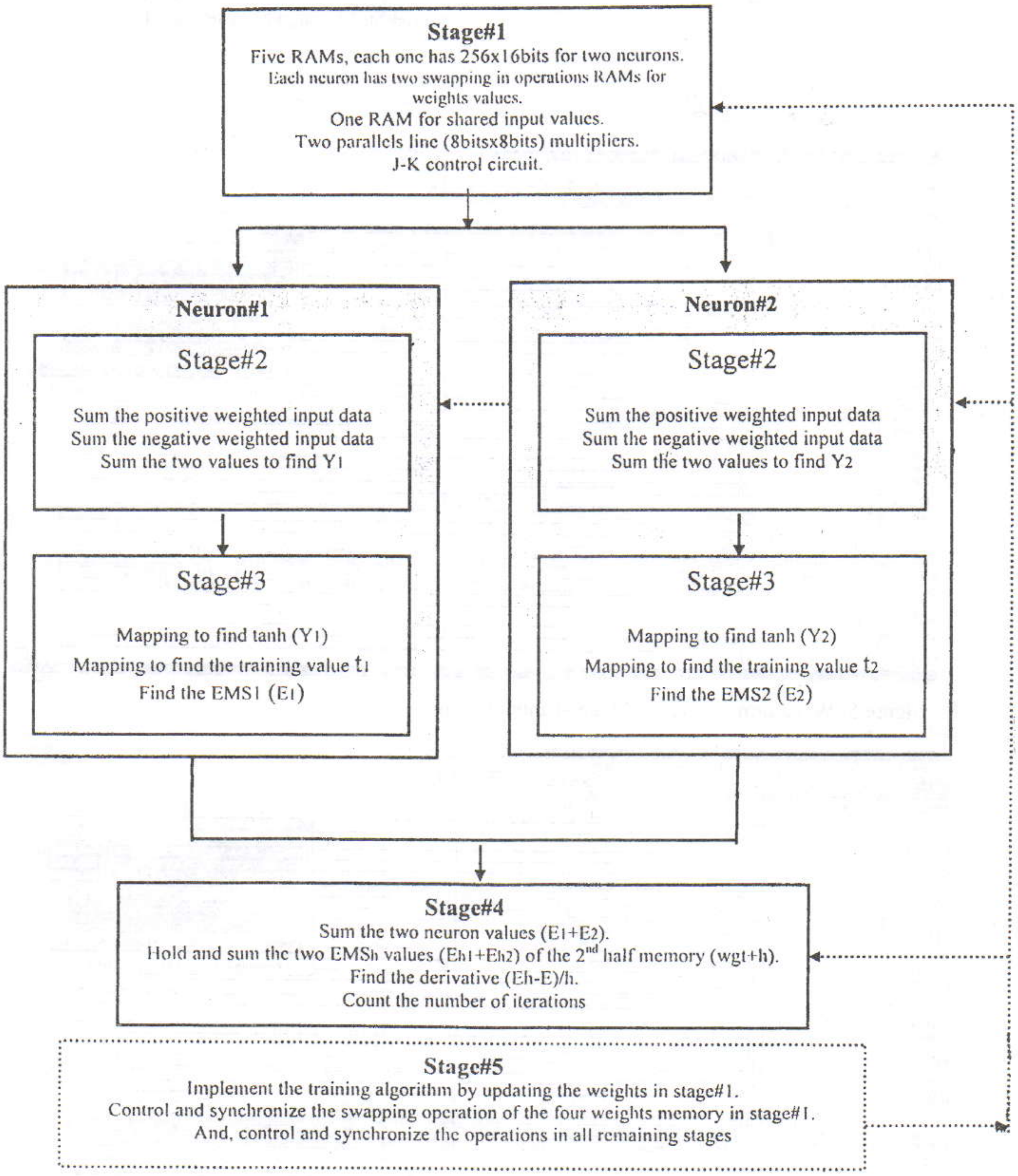


Figure 6: Timing diagram of stage#2 model-2



Stage#1
Five RAMs, each one has 256x16bits for two neurons.
Each neuron has two swapping in operations RAMs for weights values.
One RAM for shared input values.
Two parallels line (8bitsx8bits) multipliers.
J-K control circuit.
Neuron#1
Stage#2
Sum the positive weighted input data
Sum the negative weighted input data
Sum the two values to find Y1
Stage#3
Mapping to find tanh (Y1)
Mapping to find the training value t1
Find the EMS1 (E1)
Neuron#2
Stage#2
Sum the positive weighted input data
Sum the negative weighted input data
Sum the two values to find Y2
Stage#3
Mapping to find tanh (Y2)
Mapping to find the training value t2
Find the EMS2 (E2)
Stage#4
Sum the two neuron values (E1+E2).
Hold and sum the two EMSh values (Eh1+Eh2) of the 2nd half memory (wgt+h).
Find the derivative (Eh-E)/h.
Count the number of iterations
Stage#5
Implement the training algorithm by updating the weights in stage#1.
Control and synchronize the swapping operation of the four weights memory in stage#1.
And, control and synchronize the operations in all remaining stages

——— Data flow
.... Control flow

Figure 4: Block diagram of the model-1 five-stage architecture