ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 1, pp: (11-18), Month: January - March 2015, Available at: www.researchpublish.com

Artificial Neural Network Implementation on FPGA Chip

Sahil Abrol[1], Mrs. Rita Mahajan[2]

ME student[1] Assistant professor[2]
[1, 2] Department of ECE, PEC University of Technology, Chandigarh, India

***Abstract*: In this review paper a hardware implementation of an artificial neural network on Field Programmable Gate Arrays (FPGA) is presented. The parallel structure of a neural network makes it potentially fast for the computation of certain tasks. The same feature makes a neural network well suited for implementation in VLSI technology. Hardware realization of a Neural Network (NN), to a large extent depends on the efficient implementation of a single neuron. FPGA-based reconfigurable computing architectures are suitable for hardware implementation of neural networks. FPGA realization of ANNs with a large number of neurons is still a challenging task. In this paper work of different researchers is presented so that it can help the young researchers in their research work.**

***Keywords:* Artificial neural network, VHDL, Back propagation Algorithm, Xilinx FPGA, Sigmoid Activation Function.**

I. INTRODUCTION

Neural networks have been used broadly in many fields, either for development or for application. They can be used to solve a great variety of problems that are difficult to be resolved by other methods. ANN has been used in many applications in science and engineering. [1] Although, neural networks have been implemented mostly in software, hardware versions are gaining importance. Software versions have the advantage of being easy to implement, but with poor performance. Hardware versions are generally more difficult and time consuming to implement, but with better performance than software versions[2].The aspiration to build intelligent systems complemented with the advances in high speed computing has proved through simulation the capability of Artificial Neural Networks (ANN) to map, model and classify nonlinear systems. Real time applications are possible only if low cost high-speed neural computation is made realizable. Towards this goal numerous works on implementation of Neural Networks (NN) have been proposed [3].Hardware-based ANNs have been implemented as both analogue and digital circuits. The analogue implementations exploit the nonlinear characteristics of CMOS (complementary metal-oxide semiconductor) devices, but they suffer from thermal drift, inexact computation results and lack of re-programmability. Digital hardware-base implementations of ANNs have been relatively scarce, representative examples of recent research can be found in. Recent advances in reprogrammable logic enable implementing large ANNs on a single field-programmable gate array (FPGA) device. The main reason for this is the miniaturization of component manufacturing technology, where the data density of electronic components doubles every 18 months [4].

ANNs are biologically inspired and require parallel computations in their nature. Microprocessors and DSPs are not suitable for parallel designs. Designing fully parallel modules can be available by ASICs and VLSIs but it is expensive and time consuming to develop such chips. In addition the design results in an ANN suited only for one target application. FPGAs not only offer parallelism but also flexible designs, savings in cost and design cycle [5].

II. COMPARISON BETWEEN SOFTWARE AN HARDWARE IMPLEMENTATION

ANN is an abstract description of human brain. As it is a mathematical model, it can be implemented by integrated circuits or simulated using computer program. Nonetheless, the inherent parallelism embedded in neural network dynamics can be only fully realized in hardware implementation. Neumann-type computers are well-known for ANN simulation. However, the

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 1, pp: (11-18), Month: January - March 2015, Available at: www.researchpublish.com

speed of this kind of simulation is constrained when the size of ANN become large. In addition, software simulation is executed sequentially. Many researchers are developing VLSI implementations using various techniques, ranging from digital to analog and even optical. Complete parallel architecture can be realized with ASIC or VLSI, but as ANN design is targeted for certain problem solving, it is a waste to use ASIC or VLSI for implementation. While the primary disadvantages of analog implementation are the inaccurate computations and low design flexibility even though they can possibly provide higher speed with low resource cost, the major problems

ANN with digital architecture are the implementation of the large quantity of multipliers and nonlinear activation function of neurons. Both of them are usually large in size.

III. ARTIFICIAL NEURON

Artificial neural networks are inspired by the biological neural systems. The transmission of signals in biological neurons through synapses is a complex chemical process in which specific transmitter substances are released from the sending side of the synapse. The effect is to raise or lower the electrical potential inside the body of the receiving cell. If this potential reaches a threshold, the neuron fires. It is this characteristic of the biological neurons that the artificial neuron model proposed by McCulloch Pitts attempts to reproduce. Following neuron model shown in Fig. 1 is widely used in artificial neural networks with some variations.

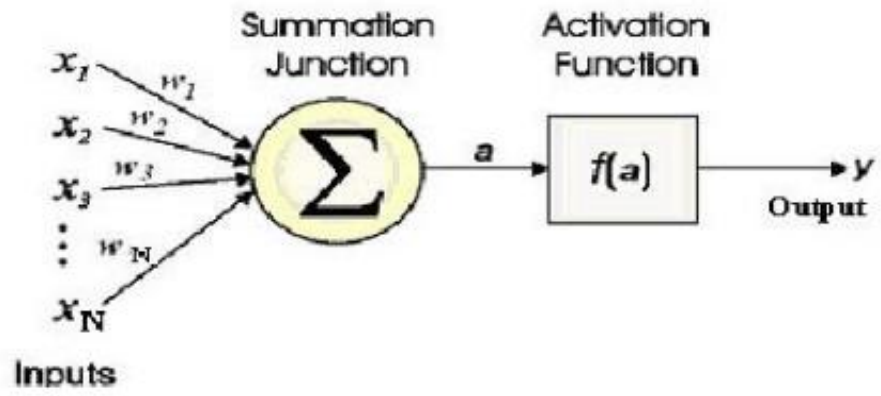

Fig: 1. Structural diagram of a Neuron

The artificial neuron given in this figure has N inputs, denoted as x1,x2,…..xN,. Each line connecting these inputs to the neuron is assigned a weight, denoted as w1,w2,…..wN respectively. The activation, a, determines whether the neuron is to be fired or not. It is given by the formula:

$$a = (\sum_{j=1}^{N} w_j x_j)$$

A negative value for a weight indicates an inhibitory connection while a positive value indicates excitatory connection.

The output, y of the neuron is given as:

y = f(a) (2)

Originally the neuron output function f(a) in McCulloch Pitts model was proposed as threshold function, however linear, ramp, and sigmoid functions are also used in different situations. The vector notation

$a = w^t x$ (3)

can be used for expressing the activation of a neuron. Here, the *jth* element of the input vector x is *xj*, the *jth* element of the weight vector of w is *wj*. Both of these vectors are of size N. A neuro-computing system is made up of a number of artificial neurons and a huge number of interconnections between them. Fig. 2 shows architecture of feed forward neural network.[5]

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 1, pp: (11-18), Month: January - March 2015, Available at: www.researchpublish.com

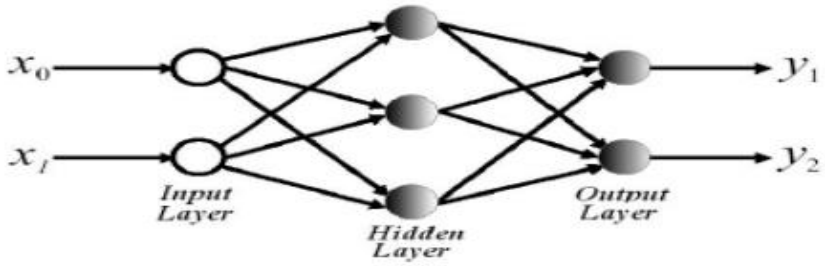

Fig: 2. Layered feed forward Neural Network

In layered neural networks, the neurons are organized in the form of layers. The neurons in a layer get inputs from the previous layer and feed their output to the next layer. These type of networks are called feedforward networks. Output connections from a neuron to the same or previous layer neurons are not permitted. The input layer is made of special input neurons, transmitting only the applied external input to their outputs. The last layer is called the output layer, and the layers other than input & output layers are called the hidden layers. In a network, if there are input and output layers only, then it is called a single layer network. Networks with one or more hidden layers are called multilayer networks.

IV. OVERVIEW OF VHDL

VHDL is a language meant for describing digital electronic systems. In its simplest form, the description of a component in VHDL consists of an interface specification and an architectural specification. The interface description begins with the ENTITY keyword and contains the input- output ports of the component. The name of the component comes after the ENTITY keyword and is followed by IS, which is also a VHDL keyword. The description of the internal implementation of an entity is called an architecture body of the entity. There may be a number of different architecture bodies of an interface to an entity corresponding to alternative implementations that perform the same function. The alternative implementations of the architecture body of the entity is termed as Behavioral Description or Structural Description.

V. NEURON ARCHITECTURE

The different types of architectures for the hardware implementation of an artificial neuron is shown in Fig3, Fig4 and Fig5.The basic functional units of a hardware neuron compute the inner product for the neuron and it is made up of the entities shown in Fig3, Fig4,Fig5. The input register was implemented with a shift register for iterative entering of the input values into the neuron. The weights register was realized using a shift register and it serves the purpose of entering the corresponding weight of the current input value into the neuron. The multiply accumulate (MAC) unit of the neuron was realized with combinational circuits for full adder and multiplier. Then the output of MAC is passed through the activation function which fires if the input to it crosses the threshold value. The activation function may be a LOGSIG,TANSIG,PURELIN or user define but the most commonly used function is the TANSIG.

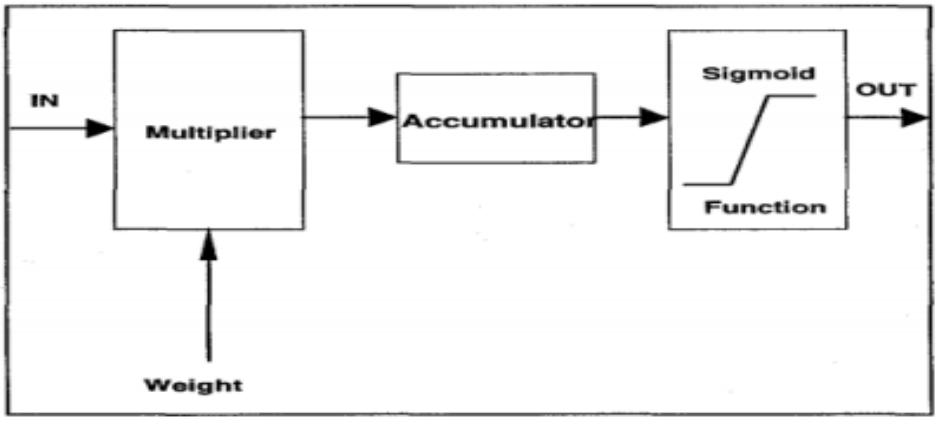

Fig 3: Hardware Architecture of an Artificial Neuron [6]

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 1, pp: (11-18), Month: January - March 2015, Available at: www.researchpublish.com

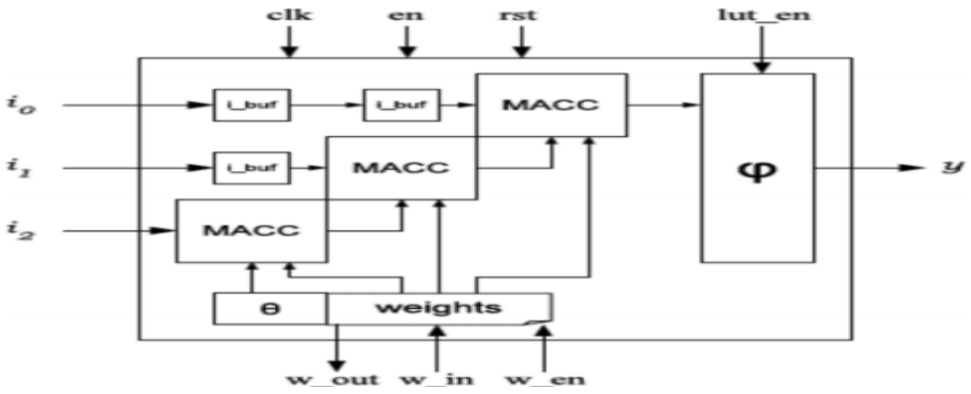

Fig 4: Functional blocks of the a Neuron [12]

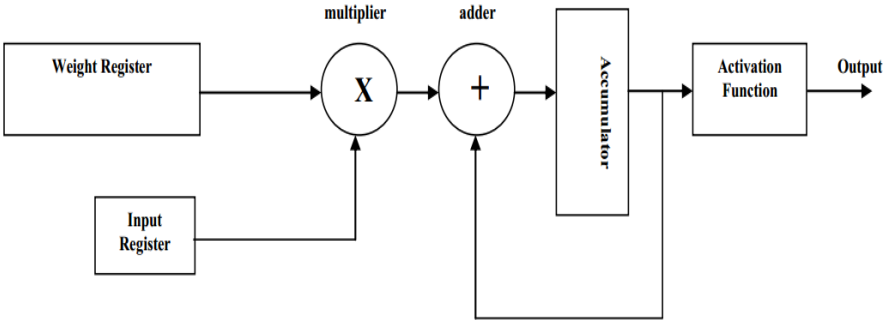

Fig 5: Simple Neuron Architecture [14]

VI. NEURON ACTIVATION FUNCTION

One of the most important parts of a neuron is its activation function. The nonlinearity of the activation function makes it possible to approximate any function. In the hardware implementation concept of neural networks, it is not so easy to realize sigmoid activation functions. Special attention must be paid to an area-efficient implementation of every computational element when implementing large ANNs on digital hardware. This holds true for the nonlinear activation function used at the output of neurons. A common activation function is the sigmoid function

$$y = \frac{1}{1 + \varepsilon^{-x}}$$

Efficient implementation of the sigmoid function on an FPGA is a difficult challenge faced by designers. It is not suitable for direct implementation because it consists of an infinite exponential series. In most cases computationally simplified alternatives of sigmoid function are used. Direct implementation for non-linear sigmoid transfer functions is very expensive. There are two practical approaches to approximate sigmoid functions with simple FPGA designs. Piece-wise linear approximation describes a combination of lines in the form of **y=ax+b** which is used to approximate the sigmoid function. Especially if the coefficients for the lines are chosen to be powers of two, the sigmoid functions can be realized by a series of shift and add operations. The second method is lookup tables, in which uniform samples takenfrom the centre of a sigmoid function can be stored in a table for look up. The regions outside the center of the sigmoid function are still approximated in a piece-wise linear fashion. This research presents an approximation approach to implement sigmoid function. A simple second order nonlinear function can be used as an approximation to a sigmoid function. This nonlinear function can be implemented directly using digital techniques. The following equation is a second order nonlinear function which has a tansig transition between the upper and lower saturation regions:

$$G_s(z) = \begin{cases} 1 \, for L \le z \\ z(\beta - \theta z) \, for 0 \le z \le L \\ z(\beta + \theta z) \, for -L \le z \le 0 \\ -1 \, for \le -L \end{cases}$$

ISSN 2348-1196 (print)
International Journal of Computer Science and Information Technology Research ISSN 2348-120X (online)
Vol. 3, Issue 1, pp: (11-18), Month: January - March 2015, Available at: www.researchpublish.com

Where β and θ represent the slope and the gain of the nonlinear function $G_s(z)$ between the saturation regions -L and L.

In another paper[10], the approximation for the sigmoid function used in this neural network is given by the function:

$$f(net) = \frac{1}{2}\left[\frac{net}{1+|net|}+1\right]$$

The circuits may be extracted, which will implement the neural network algorithm. These circuits are extracted by creating the data flow graph for the above equation. This graph is then used to configure the hardware.

VII. SINGLE LAYER PERCEPTRON IMPLEMENTATION

Definition:—A perceptron is a type of Neural Network which computes a weighted sum of its inputs, and puts this sum through a special function, called the activation, to produce the output. The activation function can be linear or nonlinear. [11]

A perceptron decides whether an input belongs to one of two classes (denoted by C1 and C2) is shown in the Fig.6.The single node computes a weighted sum of the input elements, subtracts a threshold (H) and passes the result through a hard limiting nonlinearity such that the output y is either +1 or -1. The decision rule is to respond class A if the output is +1 and class B if the output is -1. Useful technique for analyzing the behavior of nets such as the Perceptron is to plot a map of the decision regions created in the multidimensional space spanned by the input variables.

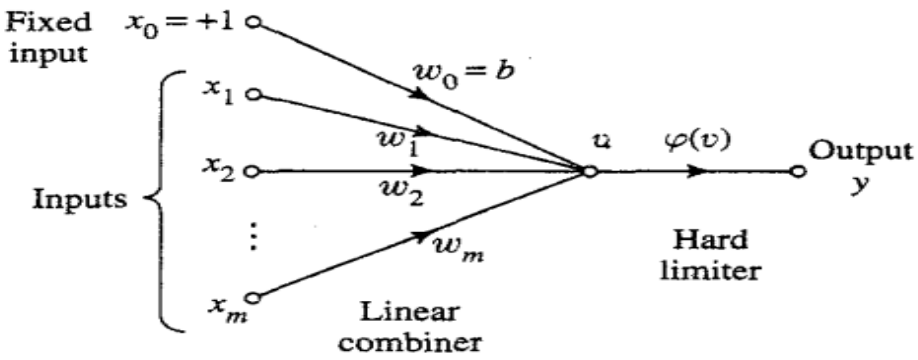

Fig.6 Single Layer Perceptron and its Signal Flow Graph[10]

Perceptron Convergence Algorithm*[10]:*

The updating of weights and bias process follows throughout a n algorithm which is known as PerceptronConvergence Algorithm. It is explained as below.

Variables and Parameters:

X(n)=(m+1)-by-1 input vector

=[+1,x1(n), x2(n), ….. xm(n)]^T

W(n)= (m+1)-by-1 weight vector

= [b(n), w1(n), w2(n), ….. wm(n)]^T

b(n) = bias

y(n) = actual response (quantized)

d(n) =desired response

η = learning rate parameter, a positive constant less than unity

STEP 1: Initialization.

Set w(0)=0. Then perform the following computation for time step n=1,2,…

STEP 2 : Activation.

At time step n, activate the perceptron by applying continuous valued input vector x(n) and desired response d(n).

STEP 3: Computation of Actual Response of Perceptron.

$y(n)= sgn[(w^T(n)x(n)]$.

Where sgn(.) is the Signum function.

STEP 4. Adaptation of Weight Vector.

Update the weight vector of Perceptron:

$W(n+1) = w(n) + \eta.[d(n) – y(n)].x(n)$

Where, d(n)= +1, if x(n) belongs to class C1

= -1, if x(n) belongs to class C2

STEP 5: Continuation

Increment time step n by one and go back to step 2.Single Layer Perceptron can learn or be trained by using this Perceptron Convergence algorithm.

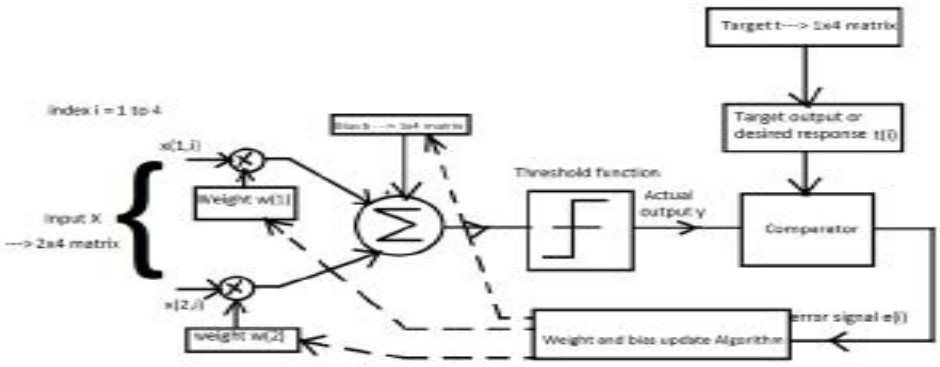

Fig.7 Single layer Single Neuron Perceptron for AND, OR, NAND,NOR function.

Fig.*7* shows block diagram of Single layer Single Neuron Perceptron which can be implemented for AND, OR, NAND and NOR function. Input of Perceptron used here for all functions is 2x4 matrix is: X= [1 1 -1 -1; 1 -1 1 -1]; Target matrix is different for all functions as all functions are different. But it is 1x4 always matrix for 2x4 input matrix.

Target matrix for AND function: T=[1 -1 -1 -1];

Target matrix for NAND function: T=[-1 1 1 1];

Target matrix for OR function: T=[1 1 1 -1];

Target matrix for NOR function: T=[-1 -1 -1 1];

Perceptron gets input from input matrix and maps actual output (which is initially taken as zero) with target output.

The goal is to minimize the generated the error signal which is nothing but the difference between actual output and target output.. In this paper, the error signal becomes zero within only two iterations for learning rate =1 and threshold=0. For varying learning rate and threshold, the Algorithm takes different number of epochs to converge as shown in fig.5. It has been inferred from the fig that for larger the learning rate, smaller number of epochs (iterations) required.

To implement the Perceptron on Hardware, it is essential that all the values are in the fixed point format. This is because floating point numbers are never synthesizable.

Table 1: Device Utilization Summary for Perceptron for AND, NAND, OR and NOR function

Logic Utilization	Used	Available	%age
Number of occupied Slices 1..AND2. NAND 3. OR 4. NOR	**38**	**7680**	**1**
	40	**7680**	**1**
	10	**7680**	**0**
	22	**7680**	**0**
Number of 4 input LUTs 1..AND 2. NAND 3. OR 4. NOR	**71**	**15360**	**1**
	71	**15360**	**1**
	17	**15360**	**0**
	39	**15360**	**0**
Number of bonded IOBs 1..AND 2. NAND 3. OR 4. NOR	**37**	**221**	**16**
	37	**221**	**16**
	21	**221**	**9**
	29	**221**	**13**

VIII. CONCLUSION

This paper has presented the design and implementation of a neuron that will be used in any neural network, the activation function that designed inside the neuron is a sigmoid function. This design was performed using schematic editor tools in a reconfigurable device (FPGA) program. The design of the neuron will be as a micro neuron. Therefore, the user can use any numbers of this neuron by just make drag to the neuron component from the library of a schematic editor.Finally , it can be say that FPGAs technology and their low cost , and re-programmability make this approach a very powerful option for implementing ANNS as an alternative to the development of custom hardware.

REFERENCES

[1] Medhat Moussa and Shawki Areibi and Kristian Nichols, "On the Arithmetic Precision for Implementing Back-Propagation Networks on FPGA", University of Guelph, school of engineering, Guelph, Ontario, Canada, 2003.

[2] Rolf F. Molz, Paulo M. Engel, Fernando G. Moraes ," Codesign to Fully Parallel Neural Network for a Classification Problem" , University Montpellier II, France, 2000.

[3] .A. Muthuramalingam ,S. Himavathi, and E. Srinivasan, "Neural network implementation using fpga: Issues and Application," The International Journal of Information Technology , vol. 4, no.2, pp.86-92, 2008.

[4] M.T.Tommiska, "Efficient digital implementation of the sigmoid function for reprogrammable logic," IEEE Proceedings, Computers and Digital Techniques, vol. 150, no. 6, pp. 403- 411, 2003.

[5] .A.Savran and S.Ünsal, "Hardware implementation of a feed forward neural network using FPGAs'," Ege, Department of Electrical and Electronics Engineering, 2003.

[6] Esraa Zeki Mohammed and Haitham Kareem Ali, "Hardware Implementation of Artificial Neural Network Using Field Programmable Gate Array" International Journal of Computer Theory and Engineering, Vol. 5, No. 5, October 2013.

[7] Haitham Kareem Ali and Esraa Zeki Mohammed, "Design Artificial Neural Network Using FPGA" IJCSNS VOL.10 No.8, August 2010.

[8] Hardik H. Makwana, Dharmesh J. Shah, Priyesh P. Gandhi," FPGA Implementation of Artificial Neural Network" International Journal of Emerging Technology and Advanced Engineering Volume 3, Issue 1, January 2013.

[9] Rafid Ahmed Khalil and Sa'ad Ahmed Al-Kazzaz, "Digital Hardware Implementation of Artificial Neurons Models Using FPGA" Al-Rafidain Engineering Vol.17 No.2 April 2009.

[10] Simon Haykin ―Neural Networks A comprehensive Foundation second edition, Prentice Hall, 2007, pp. 158-161.

[11] Mu-Song Chen, Ph.D. Thesis ―Analysis And Design of The Multilayer Perception Using Polynomial Basis Functions the University of Texas At Arlington December 1991.

[12] Chandrashekhar Kalbande & Anil Bavaskar, "Implementation of FPGA-Based General Purpose Artificial Neural Network" ITSI-TEEE 2320 – 8945, Volume -1, Issue -3, 2013.

[13] Rafael Gadea, Joaquín Cerdá, Franciso Ballester, Antonio Mocholí, "Artificial Neural Network Implementation on a single FPGA of a Pipelined On-Line Backpropagation".

[14] Emmanuel Adetiba, F.A. Ibikunle, S.A. Daramola and A.T. Olajide, "Implementation of Efficient Multilayer Perceptron ANN Neurons on Field Programmable Gate Array Chip" International Journal of Engineering & Technology IJET-IJENS Vol14 No:01.