# Design Artificial Neural Network Using FPGA

**Haitham Kareem Ali [1] and Esraa Zeki Mohammed [2]**

[1]Communication Engineering Department, Sulaimani Technical College, Sulaimani, Iraq
[2]State Company for Internet Services, Ministry of Communication, Kirkuk, Iraq

**Abstract**
In the last few years, the electronic devices production field has witness a great revolution by having the new birth of the extraordinary FPGA (Field Programmable Gate Array) family platforms. These platforms are the optimum and best choice for the modern digital systems now a day.
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 a hardware design of an artificial neural network on Field Programmable Gate Arrays (FPGA) is presented. A digital system architecture is designed to realize a feedforward multilayer neural network. The designed architecture is described using Very High Speed Integrated Circuits Hardware Description Language (VHDL).

***Key words:***
*Artificial Neural Network, Hardware Description Language, Field Programmable Gate Arrays (FPGAs), Sigmoid Activation Function.*

## 1. Introduction

Artificial neural networks (ANN) have found widespread deployment in a broad spectrum of classification, perception, association and control applications [1].
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-peed neural computation is made realizable. Towards this goal numerous works on implementation of Neural Networks (NN) have been proposed [2].
Artificial neural networks (ANNs) have been mostly implemented in software. This has benefits, since the

designer does not need to know the inner workings of neural network elements, but can concentrate on the application of the neural network. However, a disadvantage in real-time applications of software-based ANNs is slower execution compared with hardware-based ANNs.
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 reprogrammability.
Digital hardware-based implementations of ANNs have been relatively scarce, representive 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 miniaturisation of component manufacturing technology, where the data density of electronic components doubles every 18 months [3].
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 [4].

## 2. 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 figure (1) is widely used in artificial neural networks with some variations.

Manuscript received August 5, 2010
Manuscript revised August 20, 2010

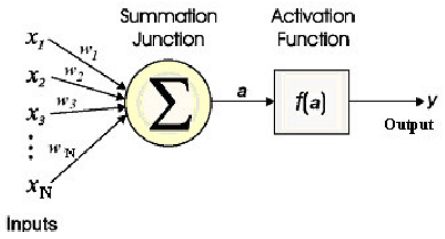


Figure (1) Structural diagram of a neuron

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:

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

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:

$$y = f(a) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, (2)$$

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

$$a = \mathbf{w}^T \mathbf{x} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots, (3)$$

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. Figure (2) shows architecture of feedforward neural network [5].

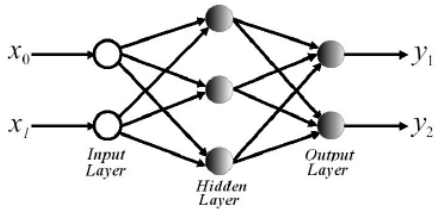


Figure (2) Layered Feedforward Neural Network

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.

(Multilayer perceptron is well-known feedforward layered network, on which the Backpropagation learning algorithm is widely implemented). The structure, where connections to the neurons are to the same layer or to the previous layers are called recurrent networks. Hopfield and Boltzmann Machine are examples of widely used recurrent networks [6].

## 3. 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. After describing a digital system in VHDL, simulation of the VHDL code has to be carried out for two reasons. First, we need to verify whether the VHDL code correctly implements the intended design. Second, we need to verify that the design meets its specifications. The simulation is used to test the VHDL code by writing test bench models. A test bench is a model that is employed to exercise and verify the correctness of a hardware model and it can be described in the same language.

Some synthesis tools are capable of implementing the digital system described by the VHDL code using a PGA (Programmable gate array) or CPLD (Complex programmable logic devices). The PLDs are capable of implementing a sequential network but not a complete digital system. Programmable gate arrays and complex programmable logic devices are more flexible and more versatile and can be used to implement a complete digital system on a single chip. A typical PGA is an IC that contains an array of identical logic cells with programmable interconnections. The user can program the functions realized by each logic cell and the connections between the cells. Such PGAs are often called FPGAs since they are field programmable [7, 8].

## 4. The Proposed Design

The proposed design consists of neuron architecture design, activation function problem solving and artificial neural network design which consist of two layers.

### 4.1 Neuron Architecture

The processing element of an ANN is the Neuron. A Neuron can be viewed as processing data in three steps; the weighting of its input values, the summation of them all and their filtering by sigmoid function. The summation can be calculated by a serial accumulation. For the weighted inputs to be calculated in parallel using conventional design techniques, a large number of multiplier units would be required. To avoid this, Multiplier/Accumulator architecture has been selected. It takes the input serially, multiplies them with the corresponding weight and accumulates their sum in a register. The processes are synchronized to clock signal. The number of clock cycles for a neuron to finish its work, equals to the number of connections from the previous layer. The accumulator has a load signal, so that the bias values are loaded to all neurons at start-up. Figure (3) shows the proposed neuron design.

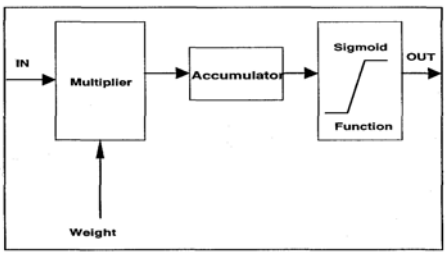


Figure (3) Neuron Architecture

### 4.2 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 [9].

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

$$y = \frac{1}{1+\varepsilon^{-x}} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots, (4)$$

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 taken from the centre of a sigmoid function can be stored in a table for look up. The regions outside the centre 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:

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

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

The structural diagram of the approximated sigmoid function implemented using this process Gs(z) is shown in Figure (4).

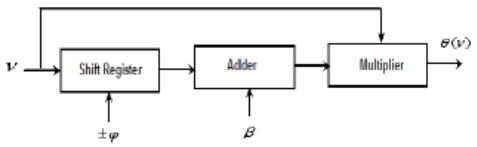


Figure (4) Structural Diagram for Sigmoid Function

The VHDL code for the approximated sigmoid activation function shown in code (1).

Code (1)

```
Library IEEE;
Use IEEE.std_logic_1164.all;
Use IEEE.std_logic_arith.all;
Use IEEE.std_logic_unsigned.all;

Entity Sigmoid is
    Port(
          Z:in INTEGER range 0 to 511;
         S:out INTEGER range 0 to 255
         );
End Sigmoid;
Architecture Sigmoidarch of Sigmoid is
Signal TEMP:integer range  0 to 255;
Signal B:integer range o to 511;
Signal A:integer range 0 to 511;
Signal Z:integer range 0 to 65535;
Signal ZZ: integer range 0 to 262144;
Costant L: integer range 0 to 255:=255;
Constant M: integer range 0 to 1023:=512;
Begin
A<=Z;
B<=M-A;
ZZ<=B*A;
Z<=ZZ/265;
TEMp<=Z;
S<=TEMP when A<L else L;
End Sigmoidarch;
```

### 4.3 Layer Architecture

Implementation of a fully parallel neural network is possible in FPGAs. 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. Since all of the products must be summed, the number of full adders equals to the number of connections to the previous layer minus one. For example in a 3-1 network the output neuron must have 3 multipliers and 2 full adders. So different neuron architectures have to be designed for each layer. Because multipliers are the most resource using elements in a neuron structure, a second drawback of a fully parallel network is gate resource usage. Figure (5) show single layer artificial neural network with three input nodes and one output node.

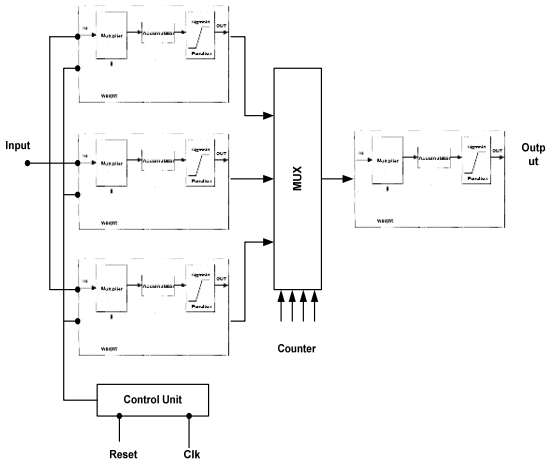


Figure (5) Artificial Neural Network Design

## 5. Conclusions

This paper presents design solution to eliminate the FPGA design for a Artificial Neural Network. The motivation for this study stems form the fact that an FPGA coprocessor with limited logic density and capabilities can used in building Artificial Neural Network which is widely used in solving different problems.

Future work involves estimating the maximum size of ANNs in modern FPGAs. The main points are the size and parameterisability of multipliers and the number of interlayer interconnections. The first defines mainly the required area resources and the second defines the required routing

## References

[1] Benard Widrow, David E. Rumelhart, and Michael A. Lehr, “Neural Networks: Applications in Industry, Business and Science”, Communications of the ACM, vol. 37, no. 3, pp. 93–105, 1994.

[2] A. Muthuramalingam, S. Himavathi, E. Srinivasan, “Neural Network Implementation Using FPGA: Issues and Application”, The International Journal of Information Technology, vol. 4, no. 2, pp.86-92, 2008.

[3] 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.

[4] Aydoğan Savran, Serkan Ünsal, “Hardware Implementation of a Feedforward Neural Network Using FPGAs’, Ege University, Department of Electrical and Electronics Engineering, 2003.

[5] Haykin S., “Neural Networks-A Comprehensive Foundations”, Prentice-Hall International, New Jersey, 1999.

[6] C. S. Rai, Amit Prakash Singh, "A Review of Implementation Techniques For Artificial Neural Networks", University School of Information Technology, GGS Indraprastha University, Delhi, 2006.

[7] S.P.Joy Vasantha Rani P.Kanagasabapathy, "Design of Neural Network on FPGA", International Conference on VLS, USA, 2004.
[8] Hamblen, J. and Furman, M., "Rapid prototyping of digital systems", Boston, Kluwer Academic Publisher 2nd Edition, 2001.
[9] Mutlu Avcý, Tulay Yýldýrým, "Generation of Tangent Hyperbolic Sigmoid Function for Microcontroller Based Digital Implementation of Neural Networks", International XII. Turkish Symposium on Artificial Intelligence and Neural Networks, 2003.

**Haitham Kareem** received the B.S. Electrical and Electronic(AVIONICS) Engineering from Al-Rashied College of Engineering & Science in lraq (1992), M.S. degrees in Communication and Radar Engineering from Al-Rashied college of Engineering & Science in lraq (1997) and Ph.D. in Electronic Engineering from university of technology, Al- Rashied college of engineering & science in lraq (2006).
From 1992-1997 working as Ass.Lecturer in University of Technology-College of Engineering-(lraq). From 1998-1999 Ass.Lecturer-Academic Cararey for Technicians-(Khartoum-Sudan). From 2000-2004 Ass.Lecturer-University of Technology-College of Engineering-(lraq). From 2004-2006 Lecturer-University of Diyala -College of Engineering-(lraq). From 2006 until now Lecturer-Foundation of Technical Education-Sulaimany Technical College- Communication Engineering Department-(Iraq).

**Esraa Zeki** received the B.S. in computer science from Mosul University in 2000-2001(Iraq), and M.S. degrees in computer science from Sulaimani university in 2008-2009 (Iraq).
From 2001-2002 working as external lecturer in Kirkuk technical institute, software department in Kirkuk technical Education (Iraq), from 2002 until now working in state company for internet services in Kirkuk (Iraq).