IMPLEMENTATION OF AN ARTIFICIAL NEURAL NETWORK EMPLOYING FIELD PROGRAMMABLE GATE ARRAY TECHNOLOGY

Yousry El- Gamal, Magdy Saeb, Nadine El- Mekky

Arab Academy for Science, Technology & Maritime Transport
School of Engineering, Computer Department,
Alexandria, Egypt

Abstract: A hardware artificial neural network implementation employing field programmable gate array technology is presented. We propose a recurrent weight-updating algorithm with variable stability controlling factor to accelerate on chip learning. The digital circuit includes a status register that holds the address of the neuron that has fired. Thus, there is no need to retrain the circuit for a given problem. While full parallelism in the adder and multiplier circuits is not achieved, yet a remarkable performance of the order of few hundreds of nanoseconds was recorded. We used one-layer module, with input multiplexers, to replace multi-layer implementation. This has led to a considerable saving of precious chip area. The simplicity of the circuit obtained, with its inherently reliable operation, makes it a worthy candidate for larger massively parallel architectures.

Keywords: Artificial Neural Networks (ANN), Multi-layer ANN (MNN), Recurrent ANN, FPGA, Parallel architecture.

1. INTRODUCTION

Recent advances in artificial neural network systems (ANNs) have led to several applications such as image recognition, speech recognition, and pattern classification [3]. Although most practical applications of ANNs are carried out using software simulators, many other potential applications require large, high-speed networks implemented in efficient custom hardware, which can fully utilize the inherent parallelism embedded in neural network dynamics. Many designers and researches are developing VLSI implementations using various techniques, ranging from digital to analog and even optical [1]. The primary disadvantages of analog implementation are the inaccuracy of analog computations and low design flexibility even though they can possibly provide higher speed with low hardware cost [2]. On the other hand, digital ANN implementation can take advantage of some of the benefits of the state-of-the-art VLSI implementation techniques. The classical ANN implementation generally requires at least five functions such as in multi-layer neural network [1]. These are:

1. Weight storage;
2. Synaptic multiplication;
3. Summation of synaptic contribution;
4. Nonlinear activation function;
5. Transmission of input and output activities among neurons.
6. Network status storage.

Two major problems are encountered in the implementation of ANN with digital architecture. These are the multipliers and the nonlinear characteristics of neurons, which require large circuits [3]. The most significant feature of neural networks is their learning ability. Size and real-time considerations show that on-chip learning is necessary for a large range of applications. In this work we propose a hardware artificial neural network implementation employing field programmable gate array technology. We employ a recurrent weight-updating algorithm with variable stability controlling factor to accelerate on chip learning [5]. The digital circuit includes a status register that holds the address of the neuron that has fired. Thus, there is no need to retrain the circuit for a given problem. While full parallelism in the adder and multiplier circuits has not been achieved, yet a remarkable performance of the order of hundreds of nanoseconds was recorded [2]. We used one-layer module, with input multiplexers, to replace multi-layer hardware implementation. This has allowed us to save precious chip area. The simplicity of the circuit obtained makes it a worthy candidate for future larger massively parallel architectures.

In the next section, we discuss the operation of ANN. In section 3 the architecture of the proposed ANN is described. Section 4 demonstrates the learning phases while training our ANN; section 5 gives a summary of the results obtained. Finally, we give an overall summary and our conclusions. An added appendix partially shows the obtained simulation results.

2. OVERVIEW OF THE ANN

ANNs are computational systems inspired by the structure, processing method, and learning ability of a biological brain [1]. Expandable digital architectures for an ANN are needed to provide an efficient real time implementation platform. This, in turn allows the construction of very large neural networks using FPGA technology [6]. A neural network has the form $N(w, x)$ where w is the weights and x is the network input. If the data samples are (x^i, y^i) then we seek w so that $N(w, x^i) = y^i$ for all i. The neural architecture is formulated as a nonlinear Least Squares of the form:

$$\min \sum (N(w, x^i) - y^i)^2 \qquad (1)$$

During the forward operation, data from neurons of a lower layer is propagated forward to neurons in the upper layer via a feed-forward connection network. Let $Ok^{(s)}$ denote the output of kth neuron of the Sth layer, then the computation performed by each neuron is

$$Hk^{(s)} = \sum_{j=1}^{N_{s-1}} W_{kj}^{(s)} O_j^{(s-1)} + \theta_k^{(s)} \qquad (2)$$

Where layers are numbered from 0 to M. $Hk^{(s)}$ is the weighted-sum of the kth neurons in the Sth layer and $Wkj^{(s)}$ is the synaptic weight [1]. The output $Ok^{(s)}$ of the neuron is obtained by computing an activation function on the weighted-sum. Usually sigmoid function is used as the nonlinear activation function [7].

3. THE IMPLEMENTATION OF ANN USING FPGA TECHNOLOGY

In this section we discuss the number representation used, the synapse unit, the neuron unit, a one-layer unit and the multi-layer unit as follows.

3.1 Number Representation

The internal data representation in this project employs Fixed-point binary. The number is divided into two parts. The first part at the left of the decimal point is represented using 8 bits. The second eight bits in the right of the decimal point represents the fraction. Therefore, a total number of bits is 16 bits is used for infix number representation.

3.2 Synapse Unit

A neural network has the form $N(w, x)$ where w is the weights and x is the network input. If the data samples are (xi, yi) then each input xj is multiplied by an adjustable constant wij before being fed to the ith processing element in the output layer, using the multiplier unit described below yielding:

$$yi = Wi * Xi \qquad \text{for all } i \qquad (3)$$

3.2.1 Add-and-Shift Multiplier

The process consists of looking at successive bits of the multiplier (B), and considering the least significant bit first. If the multiplier bit is a 1, the multiplicand (A) is copied down, else zeros are copied down. Each time the number copied down is shifted one position to the left from the previous state. Finally, the numbers are added and their sum forms the product. This unit is shown below.

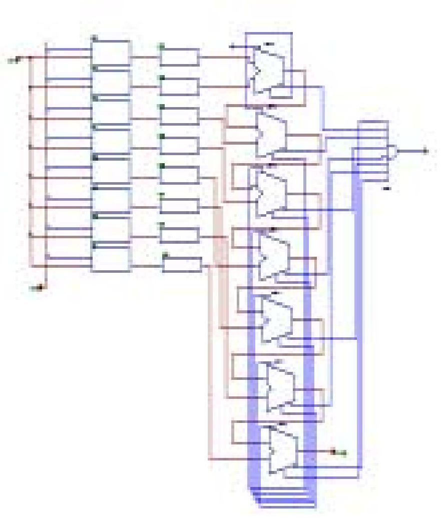


Figure 1: The ADD/SHIFT multiplier

This method takes the advantages of the fact that for binary multiplication, the partial product can only be either the top number exactly if the multiplier digit is 1 or a o if the multiplier digit is 0.

3.3 Neuron Unit

The neuron unit performs two functions:

1. Summation of synaptic contribution using the adder/subtractor described below

$$Si = \sum (Wi * Xi) \qquad (4)$$

2. Nonlinear activation function

$$Yi = f(Si + bi) \qquad (5)$$

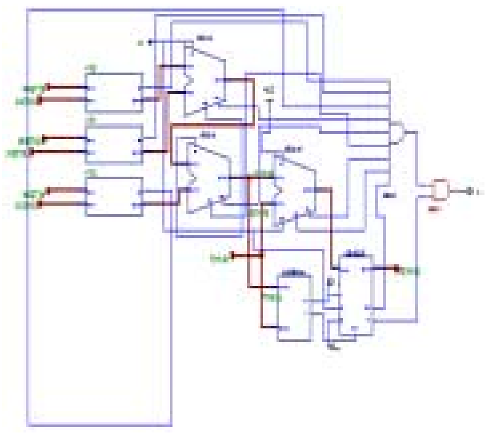


Figure2: The neuron module

3.3.1 The Adder/subtractor Module

The Adder/Subtractor module adds or subtracts two data inputs 32 bits each and Carry input. The module provides a Carry Output and an Overflow output to indicate the status of the current arithmetic operation.

3.3.2 The Activation Function

The "threshold logic" model proposed by McCulloch and Pitts [1] has two functions:

1. compare the sum with a threshold θ

   If sum >= θ then $Si = \sum Wi\ Xi - \theta i$
   Else $Si = 0$

2. A squash function f (a function that maps a large input space into small output space) is applied to the neuron's state to give its output yi.
   This function can be a sigmoid function where:

   $$yi = 1/ (1+e^{-Si}) \ldots.. (6)$$

   Or a sin function where:

   $$yi = \sin (Si) \ldots.. (7)$$

3.3.2.1 Taylor Series Expansion

We use the Taylor polynomial to find an approximation polynomial for equation (6) as shown below:

- Given that the Maclaurin series for

$1 / (1 + x) = 1 - x + x^2 + \ldots.. \ldots (8)$

- Replacing X by e^{-Si} yields

$yi = 1 - e^{-Si} + e^{-2Si} \ldots.. (9)$

- The Maclaurin Series for

$e^X = 1 + X + X^2 /2! + \ldots (10)$

- Replacing x by -Si yields

$yi = 1 - 3\ Si + 2\ Si^2 \ldots.. (11)$

3.4 One layer Unit

A one-layer unit is built by combining the simple neurons previously described

3.4.1 The status register

With each layer, there is a status register that holds the address of the neuron that has fired. This, there is no need to retrain the circuit for a given problem

3.5 Multi-layer Unit

The feed-forward network is composed of a series of two or more mutually exclusive sets of layers; the first or input layer hold site for the inputs applied to the network and distributes these values to the units in the next layer. The last, or output, layer is the stage at which the overall mapping of the network input is available. Between these two extremes, lie zero or more hidden layers where additional computing takes place. In our architecture we use a one-layer unit, with input multiplexers to replace the multi-layer hardware implementation. The size of the multiplexer depends on the number of layers in our design. If we have eight layers we will use an 8x1 multiplexer with 3 selectors [7]. The value of the selectors are used to denote the stages where we are and then the links or weights that must be stored before the execution are fed forward to provide a portion of the activation for the units in the next higher layer. Thus, from an architectural viewpoint, this design approach saves precious chip area, it also provides parallel processing within each layer. However the flow of interlayer data is necessarily serial.

4. LEARNING PHASE

Neural network computers are not programmed in the way used with digital computers. Instead of being programmed with a set of rules, the way a classic expert system is, a neural network computer learns the desired behavior by adjusting or "adapting" the weights of the interconnections until the desired output is obtained. Another input-output set is then applied, and the network is allowed to learn this set. After a few sets the network will have learned or generalized its response so that it can give the correct response to most applied input data. The scheme used to adapt the network is called the learning rule [6].

There is a specific method in training an ANN:

- Data is presented, and an output is computed.
- An error is obtained by comparing the output with a desired response, and it is used to modify the weights with a training algorithm.
- This procedure is repeated using all the data in the training set until a convergence criterion is met.

Thus in ANNs the designer does not have to specify the parameters of the system. They are automatically extracted from the input data and the desired response by means of the training algorithm.

4.1 Recurrent Learning Algorithm

Recurrent learning algorithm has been successfully applied to the training of multi-layer feed-forward neural networks in a number of practical problems. It has the advantages of:

1. Being performed totally within the neural network structure
2. Intense popularity

Criterion of the learning algorithm is to minimize the least square error between the teacher value and the actual value. At the very beginning of the learning phase, the forward path is executed to obtain the output response of the input training pattern. Then the error between the teacher value and the actual output value is used to update the weights values.
In the recurrent-trained multi-layer, the performance measure is convergence speed. This speed is affected by the choice of Learning rate λ, Momentum coefficient α. The best performance was achieved at (α, λ) = (0.8, 0.05).

$$W\ new = W\ old + \lambda\ (1-\alpha)\ \Delta w \qquad (12)$$

Suppose, that we have a single data point (X, D) consisting of an input vector X = (X*1*…Xn) and an output vector D ("desired" output) = (D*1*…Dm). For a given set w of weight values, the feed-forward network produces the output vector Y (w) = (Y*1* (w)…Ym (w)). One way to express the error Δw is the following:

$$\Delta w = \frac{1}{2}\sum_{k=1}^{m}\left(Dk - Yk(w)\right)^2 \qquad (13)$$

5. SIMULATION RESULTS

In the following few lines we provide a summary of the simulation reports obtained.

1. The Simulation Delay Summary Report:

- Average Connection Delay: 2.574 ns
- Average Connection Delay on the 10 Worst Nets: 7.095 ns.
- Total real time to completion: 13 secs.

2. Timing summary:

Constraints cover (100.0% coverage):
- 33104 paths,
- 208 nets,
- 254 connections

3. Design statistics:

- Maximum combinational path delay: 48.620ns
- Maximum net delay: 8.522ns

6. SUMMARY & CONCLUSIONS

Motivated by the relative flexibility of using FPGA technology, we have demonstrated the basic building blocks of an ANN processor implementation. The distinct features of this implementation are as follows:

- A single-layer with a multiplexer replaces a multi-layer ANN. This configuration saves precious chip area, where the chip family available to us has a relatively small gate density.
- A recurrent weight-updating algorithm with variable stability controlling factor to accelerate on-chip learning is used.
- A status register is used to store the address of the fired neuron. Thus, there is no need to retrain the neural chip for a given problem.
- In spite of a partial parallelism of the adder and multiplier circuits has been

used, yet a maximum combinational path delay of 48.620ns was achieved.

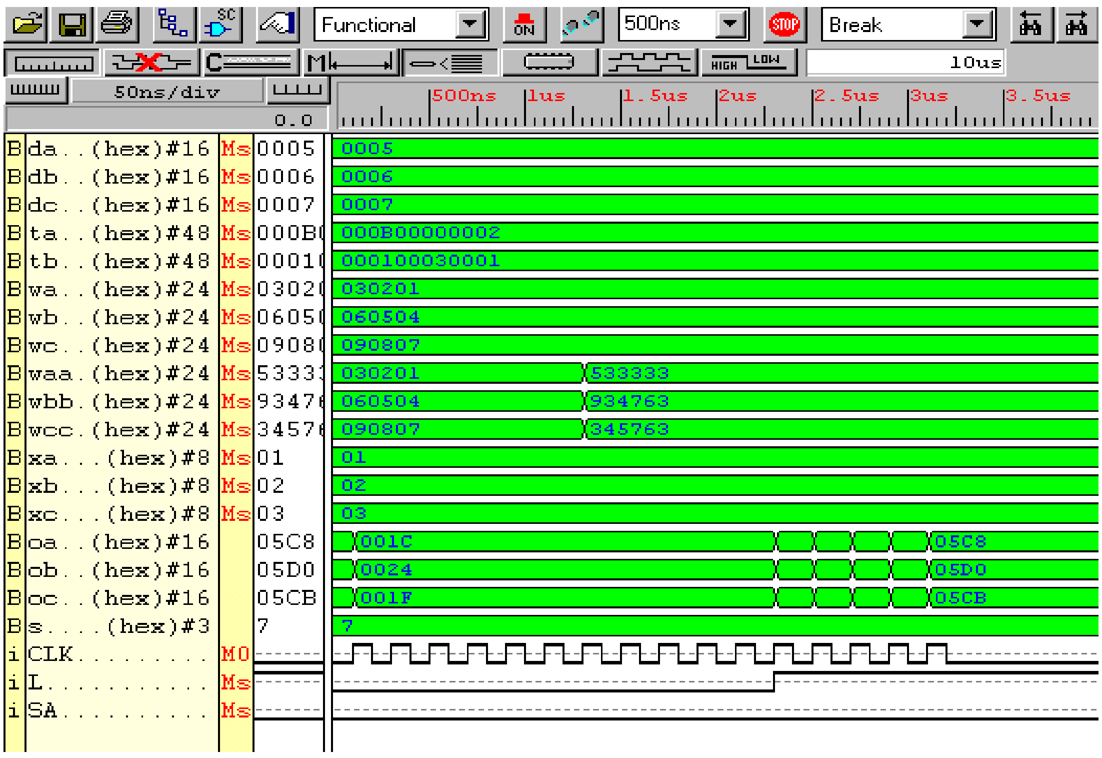
- Most of our design optimization efforts were concentrated on the minimization of the utilized area rather than minimizing time delays. However, in future implementation with advanced chip FPGA families, better performance can be easily realized.

REFERENCES

1. George Cybenko, “Neural Networks in Computational Science and Engineering,” Computational Science & Engineering, Spring 1996, pp.36-43.
2. Simon Y. Foo, Lisa R. Anderson and Yoshiyasu Takefuji, “Analog Componets for the VLSI of Neural Networks, ” Circuits & Devices, Vol. 6, No. 4, July 1990, pp. 18-26.
3. Haykin, Simon, “Neural networks: a comprehensive foundation,” 2nd ed , 1999, Prentice-Hall, Inc.
4. Smith Murray, “Neural networks for statistical modeling,”1993, Van Nostrand Reinhold.
5. H. P. Graf, L. D. Jackel, “Analog electronic neural network circuits,” IEEE Circhits Devices Mag., Vol. 5, pp. 44-49, July 1989.
6. M. Yasunga et al, “A self-learning dogital neural network using wafer-scale LSI”, IEEE journal of Solid-State Circuits, Vol. 28, pp. 106-114, 1993.
7. F. Sargeni, V. Bonaiuto, “ A fully digitally programmable CNN chip,” IEEE Trans. Circuits-II, Vol. 42, pp. 741-745, November 1995.

APPENDIX

Partial Simulation results



```
B da..(hex)#16  Ms 0005   0005
B db..(hex)#16  Ms 0006   0006
B dc..(hex)#16  Ms 0007   0007
B ta..(hex)#48  Ms 000B(  000B00000002
B tb..(hex)#48  Ms 0001(  000100030001
B wa..(hex)#24  Ms 0302(  030201
B wb..(hex)#24  Ms 0605(  060504
B wc..(hex)#24  Ms 0908(  090807
B waa.(hex)#24  Ms 5333:  030201  533333
B wbb.(hex)#24  Ms 9347(  060504  934763
B wcc.(hex)#24  Ms 3457(  090807  345763
B xa...(hex)#8  Ms 01     01
B xb...(hex)#8  Ms 02     02
B xc...(hex)#8  Ms 03     03
B oa..(hex)#16     05C8   001C  05C8
B ob..(hex)#16     05D0   0024  05D0
B oc..(hex)#16     05CB   001F  05CB
B s....(hex)#3     7      7
i CLK.........  M0
i L...........  Ms
i SA..........  Ms
```