Implementation of FPGA-Based General Purpose Artificial Neural Network

Chandrashekhar Kalbande & Anil Bavaskar

Dept of Electronics Engineering, Priyadarshini College of Nagpur, Maharashtra India
E-mail : ckshekhar333@gmail.com & anilbavaskar@gmail.com

Abstract **- The usage of the FPGA (Field Programmable Gate Array) for neural network implementation provides flexibility in programmable systems. For the neural network based instrument prototype in real time application, conventional specific VLSI neural chip design suffers the limitation in time and cost. With low precision artificial neural network design, FPGAs have higher speed and smaller size for real time application than the VLSI design. In addition, artificial neural network based on FPGAs has fairly achieved with classification application. The programmability of reconfigurable FPGAs yields the availability of fast special purpose hardware for wide applications. Its programmability could set the conditions to explore new neural network algorithms and problems of a scale that would not be feasible with conventional processor. The goal of this work is to realize the hardware implementation of neural network using FPGAs. Digital system architecture is presented using Very High Speed Integrated Circuits Hardware Description Language (VHDL) and is implemented in FPGA chip.**

Keywords**— Backpropagation, field programmable gate array (FPGA), multilayer perceptron, neural network, VHDL, Xilinx FPGA**

I. INTRODUCTION

It is a computational system inspired by the Structure, Processing Method, Learning Ability of a biological brain. Artificial Neural Networks (ANNs) can solve great variety of problems in areas of pattern recognition, image processing and medical diagnostic. The biologically inspired ANNs are parallel and distributed information processing systems. This system requires the massive parallel computation.

ANN is an information processing system that aims to simulate human brain's architecture and function. It is now a popular subject in many fields and is also a tool in many areas of problem solving. ANNs have been successfully applied to solve problems where conventional methods have been unsuccessful, such as speech recognition and synthesis, image processing and coding, pattern recognition and classification, power load forecasting, interpretation and prediction of financial trends for stock-market, manufacturing of composite structures, processing modeling, monitoring and control etc.

Characteristics of Artificial Neural Networks

A large number of very simple processing neuron-like processing elements. A large number of weighted connections between the elements Distributed representation of knowledge over the connections Knowledge is acquired by network through a learning process.

Reasons for Usage of Artificial Neural Networks

The main reasons for using an Artificial Neural Networks are as follows. It provides Massive Parallelism. A Distributed representation of any system can be developed with enhance Learning ability and Generalization ability of the system. It will also provide Fault tolerance.

Elements of Artificial Neural Networks

- Processing Units

- Topology

- Learning Algorithm

FPGAs are chosen for implementation ANNs with the following reason:

- They can be applied a wide range of logic gates starting with tens of thousands up to few millions gates.
- They can be reconfigured to change logic function while resident in the system.

- FPGAs have short design cycle that leads to fairly inexpensive logic design.
- FPGAs have parallelism in their nature. Thus, they have parallel computing environment and allows logic cycle design to work parallel.
- They have powerful design, programming and syntheses tools.

A. *Mathematical Model*

A multilayer neural network is composed of one input layer, several hidden layers for computation and one output layer. Each layer consists of a set of processing elements called neurons and the main task of each neuron is processing the following function:

y = [Cx) = [CLf=1 WiXi + b)…… (1)

where Xi stands for the ith input, and Wi is the weight in the i^{th} connection and b is the bias. The function f(x) is the nonlinear active function used in the neuron. Here we select the log-sigmoid as the active function due to its popularity, and it is described by the following:

[Cx) = l/Cl + e-X) …….(2)

Because RAM is usually used to store the weights, we did not develop the online learning in our proposed architecture, so that the hardware resource is saved.

II. COMPARISON BETWEEN SOFTWARE AND 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 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. NETWORK ARCHITECTURE

By using of the FPGA features hardware implementation of fully parallel ANN's is possible. In the fully parallel ANN's architecture number of multipliers per neuron equals to number of connections to this neuron and number of the full adders equals to number of connections to the previous layer mines one [9]. For example in 2-4-1 network output neuron have 4 multipliers and 3 adders. In this work a VHDL library were designed for floating point addition fp_add and floating point multiplication fp_mul. But most resources of FPGAs are used by multiplication and addition algorithm. So in fully parallel ANN's must be used low number precision (for example 8 bit). With the low number precision fully parallel network is not suitable for any application. With the using fp_lib (32 bit floating point number precision)in ANN's is suitable for any application. But the architecture has one multipliers and one adders per layer and is not full parallel because of area resource of FPGAs.

In this structure there is one multiplier and one adder per layer. The inputs from previous layer enter the layer parallel and multiplier serially with their corresponding weights. The results of multiplication are stored in their neuron area in the addition Neural Network Implementation in Hardware Using FPGAs 1109 storage ROM. Multiplied value of per neuron are inputs for adder. The inputs of adder are added serially and each addition are inputs for sigmoid lookup table. The resultsof look up table are stored for next layer. This ANNs architecture is shown in Figure

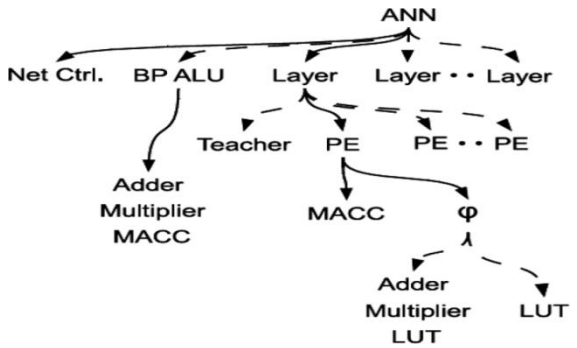


Fig.1 . Block view of the hardware architecture. Solid arrows show which components are always generated. Dashed arrows show components that may or may not be generated depending on the given parameters.

A. Recognizable Neural Networks

Among the recongurable systems currently designed in our laboratory, we present here a hardware implementation of multi-layer perceptrons. These net-works are well-suited for classification problems like hand-written character recognition. Let us briey consider this problem. The network is trained with a set of

characters written by different people. After pre-processing steps (nor-malisation, smoothing,...) we obtain a grey-level image (M x N pixels) of each character. The M x N grey-level values are stored in an input vector to which is assigned a class attribute.

Figure depicts the learning system. An input vector is presented to the neural network which determines an output. The comparison between the computed and desired output (class attribute) provides an output error. This signal is used by a learning algorithm to adapt the network parameters

We now brief describe the architecture of a multi-layer perceptron. It is composed of several layers of interconnected processing elements (PEs) or neurons: an input layer which only holds input values, one or more hidden layers of PEs and Desired

Output Network output

Learning algorithm

Comparison

B. Proposed Engine

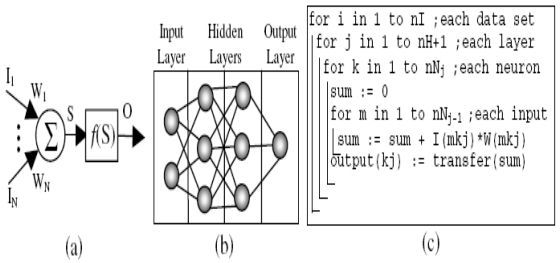


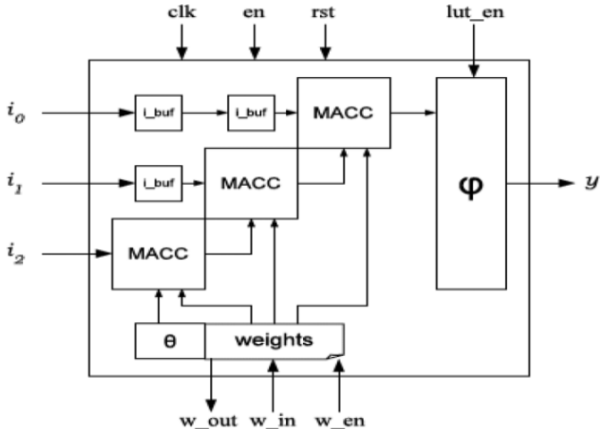
Fig. 2. a) Perceptron neuron. b) MLP topology example. c) MLP execution pseudo-code.

As presented in fig. the output of a perceptron neuron is calculated by the function *f(S)*, where f is the transfer function and *S* represents the summation of all input weight products. Analyzing fig.1a, we are able to state that there is inherent spatial parallelism in the execution of the neuron's products, called intra-neural parallelism. In fig.1b we notice that a neuron inside a layer is independent from the others within the same layer, (intra-layer parallelism). However, there is dependency among the neurons from a layer and those from the previous layer. It happens because the outputs from a layer are the inputs of the next layer. Nevertheless, the computation of different layers can be done simultaneously, since each neuron has all inputs (temporal parallelism or pipeline). It means that if the layers process different data sets, they can execute simultaneously (inter-layer parallelism).

Fig.1c presents a MLP pseudo-code. The first (outer) loop executes the entire network for all data sets. The second loop executes all hidden and output layers. The third loop executes all neurons of the layer specified in the second loop. The fourth loop executes the products and sums of each neuron, where weights and inputs are determined by previous loops. After that, the transfer function is applied to the total sum of a neuron, generating the neuron's output. Serial code implemented like this and executed in general purpose processors (GPPs) fails to explore the several different levels of inherent parallelisms inside an MLP, as previously indicated Some works implement ANN in parallel computers [1], e.g., clusters and multiprocessors [5], which yield great speedup over the sequential monoprocessed one.

However, since MLP network present fine-grained parallelism, their implementation in parallel computers not always is efficient, due to speedup, scalability and cost.

Our solution hypothesis is to design and implement MLP networks using hierarchical parallel and parameterized dedicated hardware architectures, to improve the computational performance.



Functional blocks of the PE component.

The main features of our architecture are its spatial and temporal parallelisms in different hierarchical levels, and their parameterizations. The parameters are divided in two groups, named network and architecture parameters. The first group determines the main features of the network, such as: number of inputs, number of neurons, number of layers, type of transfer function and so on. The second group determines the main features of the architecture, such as: parallelism degree among layers, neurons and modules, implementation of the sub-operations, word length (to represent input, weight and output values), and so on.

The proposed architecture is hierarchically composed of layer, neurons and modules (fig.2b). Observing fig.2, we notice that there are four possible

parallelism hierarchical levels in our architecture: (1) H1 is the network, composed of layers (temporal parallelism); (2) H2 is the layer, composed of several neurons (spatial parallelism); (3) H3 is the neuron, with operation modules pipelined execution (temporal parallelism); (4) finally H4 is neuron module with parallel implementation (temporal and spatial parallelism) of each module (fig.2a). Fig2.c is a possible implementation of a neuron with parallelism in H4 in multiplication and addition modules.

Although there are parallelism levels in our architecture, they can be used or not.

Thus, the designer must analyze the tradeoffs between performance and cost. Total parallelism implies in high performance, but higher relative cost. For example, it is possible to design an engine without H1 parallelism. In this case, only one layer would be executed at a time, which does not affect other parallelism levels or their execution.

C. The Backpropagation Algorithm

The Backpropagation algorithm is widely used for training multi-layer percep-trons [9]. It iteratively computes the values of weights using a gradient descent algorithm.

The Backpropagation algorithm consists of the following stages:

1. Network initialization.

All weights are initialized to small random numbers.

2. Forward propagation.

An input vector is presented and propagated layerwise through the network.

3. Output error computation.

4. Backward propagation.

The output error signal is back-propagated through the network. This pro-cess allows to assign errors to hidden neurons.

5. Weight update.

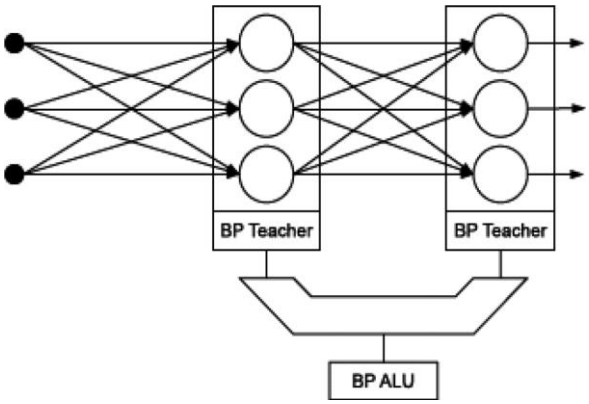
Previously computed errors (stages 3 and 4) and neuron activations deter-mine the weight changes.

Steps 2 to 5 are carried out for all vectors in the database.

This training process is repeated until the output error signal falls below a predetermined threshold.

When we train a system by example, it is usually impossible to provide every possible input signal. Therefore, an important issue of training is the capability of the network to generalize to previously unseen patterns. However, the generalization capability depends on the network topology. A rule of thumb for obtaining a good generalization is to use the smallest system that can learn the training vectors.

Unfortunately, the Back propagation algorithm does not give any information about the topology of the network (number of hidden layers, interconnections, number of neurons). Pruning (or growing) algorithms allow to _nd an optimal topology during training by removing (or adding) neurons and connections.Stands for Symmetric MultiProcessing.



III. SOFTWARE AND HARDWARE REQUIREMENT

For Software simulation I will prefer MODELSIM and for synthesis I will be prefer XILINX. Hardware requirement is SPARTAN-3.

IV. RESULT V ERIFICATION AND ANALYSIS

Observe the required result like arithmetic, logical, branching and shifting.

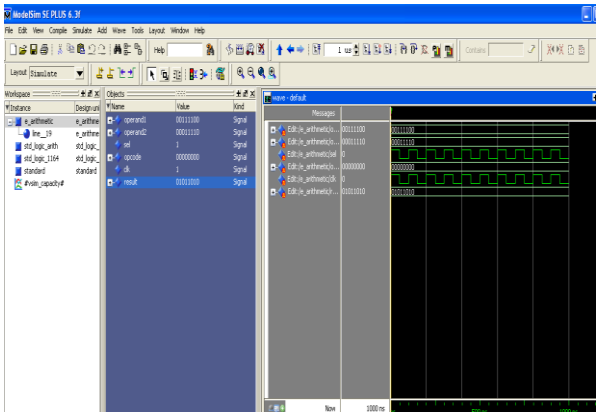


Figure 8.Simulation Result of Arithmetic Processor for addition

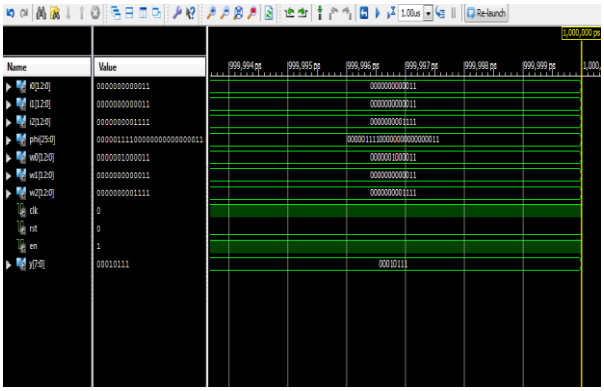


Figure 8.Simulation Result of Processing Element

V. ACKNOWLEDGEMENTS

Authors wish to remark the great task carried out by the Xilinx and Modelsim user guide; and the authors wish to thank Anil Bavaskar for his contribution in the design process.

VI. DISCUSSION AND CONCLUSION

The goal of our work is to implement a General purpose neural based instrument for automatic Waveform detection.

In general, it is shown that implementation of neural networks using FPGAs. The resultant neural networks are modular, compact, and efficient and the number of neurons, number of hidden layers and number of inputs are easily changed.

VII. REFERENCES

[1] I. A. Basheer and M. Hajmeer, "Artificial neural networks: Fundamentals, computing, design, and application," *J. Microbio. Methods*, *vol.*43, pp. 3–31, Dec. 2000.

[2] Xu Wang, Hong Wang and Wenhui. Wang, Artificial Neural Network Theory and Application, Shenyang: Northeastern University Press,2000. I]M. Ananda Ro and J. Srinivas, Neural Networks Algorithms

[3] A. R. Omondi, "Neurocomputers: a dead end?," *International Journal of Neural Systems*, vol. 10, no. 6, pp.

[4] J. *C.* Rajapakse and W. Lu, "Unified approach to independent component neural networks", *Neural Computation*,2000.475-481,2000.

[5] Applications, Alpha Science International Ltd., Part Ill, pp.157, 2003

[6] Maeda, Y.; Tada, T.: "FPGA Implementation of a Pulse Density Neural Network with Learning Ability Using Simultaneous Perturbation", IEEE Transactions on Neural Networks, vol. 14, no. 3,2003, pp. 688-695.

[7] M. Paliwal and U. A. Kumar, "Neural networks and statistical techniques:A review of applications," *Expert Systems With Applications*,vol. 36, pp. 2–17, 2009

[8] Alexander Gomperts, Abhisek Ukil,, *and Franz Zurfluh "*Development and Implementation of FPGA-Based General Purpose Neural Networks for Online Applications*"* VOL. 7, NO. 1, FEBRUARY 2011.

[9] M. Ananda Ro and J. Srinivas, Neural Networks Algorithms and Applications, Alpha Science International Ltd., Part Ill, pp.157, 2003

[10] U. Ruckert, A. Funke and C. Pintaske, "Acceleratorboard for Neural Associative Memories," Neurocomputing, Vo1.5, No.1, pp 39-49, 1993

[11] **M.** Stevenson, **R.** Weinter, and **B.** Widow, "Sensitivity of Feedforward Neural Networks **to** Weight Errors," **IEEE** Transactions on Neural Networks, Vol. 1, No. 2, pp 71-80,1990.

◈◈◈