FPGA Implementation of Multilayer Feed Forward Neural Network Architecture Using VHDL

S.Hariprasath
Saranathan College of Engineering
Panchapur, Trichy
E-mail hariprasadh@yahoo.com

T.N.Prabakar
Oxford Engineering College,
Pirattiyur, Trichy,
E-mail tnprabakar@gmail.com

Abstract— **This paper presents a hardware implementation of multilayer feed forward neural networks (FFNN) using Field Programmable gate arrays (FPGAs). In spite of huge improvements in FPGA densities, the number of multipliers in a NN limits the size of the network that can be implemented using a single FPGA and the NN applications are not made commercially viable. The proposed implementation is aimed at reducing the resource requirements, without much compromise on the speed, so that a larger NN can be realized on a single chip at a lower cost. The parallel processing of the layers in an NN has been exploited in this paper to implement larger NNs. An efficient and fast carry look-ahead adder and Booth multiplier are the essential building blocks of the processing elements to perform parallel computation in the neural network. The activation function is implemented using piecewise linear approximation. In this work, a 2-2-1 multilayer feed forward neural network is implemented with different fixed point representation. The hardware resources consumed and the results obtained are presented.**

Key words—Field-programmable gate array (FPGA); hard-ware implementation; VHDL; neural networks (NNs).

I. INTRODUCTION

Neural networks can be used to model and control complex nonlinear physical systems with unknown or slowly varying plant parameters. They have been successfully applied to robot arm control, chemical process control, continuous production of high-quality parts, and aerospace applications. Since most software neural networks run on sequentially operating architectures [1][8], it is not possible to simultaneously compute multiple connections or 'multiply and adds', which are typical nature of neural networks. Therefore software implementations tend to be prohibitively slow. Only by the hardware implementation of many devices that can each perform multiply and add operations and the parallel computation of each neuron in the layer these implementations are faster. Soon after the widespread revival of neural network research in the mid-eighties, it was realized that to fully profit from the massive parallelism and self learning capability in neural network models, it is essential to implement it in hardware.

The realization of neuron function and the method of integration have been focused in several papers [2][3].The connections between the neurons have been considered based on the time multiplexing of the interconnections [3][6]. A possible solution to the weight storage problem has been envisioned and is based on a "most significant weights" philosophy.

The common ways of implementing activation function are look-up table method[1][3][6] and approximation method, the first method needs a lot of memory size for look-up table entries, but the second method requires less amount of hardware components leading to good performance and less gate requirement.

Based on the survey of FPGA implementation of ANNs [7], the logical XOR ANNs occupies almost 4900 CLB (Configurable Logic Blocks) slices of the Virtex device and operated on the maximum clock rate of 10 MHz for 16-bit fixed point arithmetic operations.

In this paper, we focus the hardware realization of the neural network with modified Booth algorithm for digital multiplication [1][4], carry look-ahead adder for accumulation and the piecewise linear approximation of nonlinear log sigmoid activation function discussed in [8]. The hardware for the entire neural network is realized using VHDL. The architecture of the design is integrated on the ALTERA Stratix III FPGA chip of EP3S50F484C2 and occupies 32 DSP block elements for XOR problem.

Stratix III family devices are capable of implementing designs that require up to 270,400 Adaptive Look-Up Tables (ALUTs), 16.8MB of dedicated on-chip memory (excluding ALUTs used as memory), or 896 DSP block multipliers. Stratix III devices can be used for memory functions and complex logic functions, such as digital signal processing, wide data-path manipulation, data transformation, and microcontrollers. The high-pin-count Stratix III devices contain a two-dimensional row- and column-based architecture to implement custom logic. The advantages of the implemented design are given below:

1. Single memory to carry weights of all connection links from the previous layer to the particular neuron.

2. To improve the speed usage of high speed arithmetic operations such as modified Booth multiplication and carry look-ahead adder.
3. Design of sigmoid function based on piecewise linear approximation.
4. A single state machine, to control the activities of the processing elements in the layer and Signed fixed point data representation.

The fixed point data representation is described in section II and the architecture of the multilayer Perceptron neural network is described in section III. The circuit design of the booth encoder based on modified Booth algorithm, the combinational logic design of sigmoid function is also explained in section III. The simulation and synthesis results are given in section IV and finally the conclusion is given in section V.

II. DATA REPRESENTATION

The input values are represented using signed fixed point binary number format. A N-bit binary word when interpreted as a signed two's complement fixed point rational, can take on values from a subset P of the rational given by equation (1).

$$P = \left[p / 2b \,|\, -2^{N-1} \le p \le 2^{N-1} - 1, p \in Z \right] \quad (1)$$

It may be noted that P contains 2N elements and Z is the set of integers. Such representation is denoted in equation (2) which has one sign bit, 'a' integer bits and 'b' fractional bits.

$$A(a,b) = | s | ab \quad (2)$$

where a = N-b-I and s is a sign bit. If s = 1, two's complement notation is used. The value of N-bit binary number X is given by equation (3).

$$X = \frac{1}{2^b}[-2^{N-1} X_{N-1} + \sum_{0}^{N-2} 2^n X_n] \quad (3)$$

where X , represent bit n of X. The range of A (a, b) is -2N-1-b < X< 2N-1-b -2b if sign bit is present and 0 < X< 2N-1-b -2 b, if sign bit is not present. The number of bits required to represent S (a, b) is a+b+1.The example of fixed point format is given in Table 1.

TABLE I: FIXED POINT FORMAT

Number Format	Binary Representation	Decimal Representation
1-3-2	011110	+7.5
1-2-3	111111	-3.875
1-1-4	011111	1.9375

In our proposed work, the fixed point representation of (1-6-9) is used. Also the analysis of implementation using various fixed point representation is performed. The total number of bits required is 16 with one sign bit, 6 integer bits and 9 fractional bits. The neurons in the input and hidden layer receive 16-bit inputs and weights. The hidden layer sends 10-bit values with one integer bit and 8 fractional bits to the output layer. The final output from the network carries 16-bits, of 6 integer bits and 8 fractional bits.

III. NEURAL NETWORK REALIZATION

The conceptual construction of an artificial neural network is stemmed from the early understanding of the human brain. It is a collection of simple-interconnected processing elements (PEs) distributed in a parallel network. The computing abilities of neural networks come through an interaction of these simple processing units. The neurons are organized into a well-structured topology composed of individual layers. The neurons in each layer communicate with the neurons situated at the successive layers. All PEs in one layer can send signals concurrently to or from PEs of next layer. In this work, a 2-2-1 multilayer feed forward neural network as shown in figure 1 is considered.

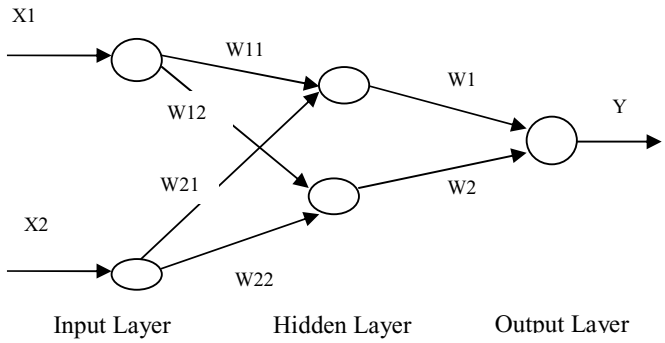

Figure.1 2-2-1 multilayer feed forward neural network

The input layer neurons pass the input signal to the hidden layer based on multiplexing. The neurons in the hidden and output layer perform the computation. Each neuron in the hidden and output layers calculates its net output by determining the product of input and weight of each connection. The final output of each neuron is determined based on the activation function. The hidden layer uses sigmoid function and the output layer uses pure linear function as activation function.

The function of the computational neuron can be implemented by means of four main blocks such as RAM memory for updating weights, digital multiplier for multiplying the synaptic weights and input signals, adder for accumulating the outputs of multiplier and the non-linear activation function.

The most basic element of the neural network, the neuron, transforms its inputs in the feed forward stage of operation according to the steps as shown in figure.2.
1) Multiply each input by its corresponding weight
2) Sum the multiplication results.

3) “Squash” the sum using a transfer function.

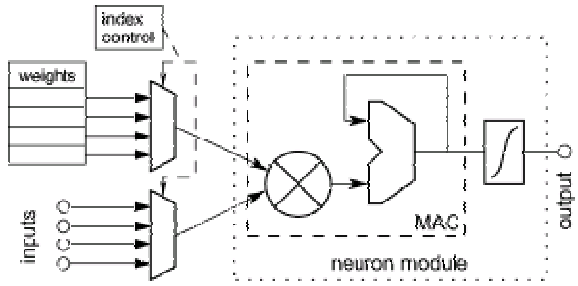

Figure .2 Neuron structure

But the feed forward processing of each neuron can be executed in many different ways. This design allows for three different processing techniques: serial, partial parallel and full parallel. In our paper partial parallel processing is implemented. As the partial parallel implementation is a tradeoff between the operational speed and resources usage, it is implemented. The basic structure is shown in figure 3.

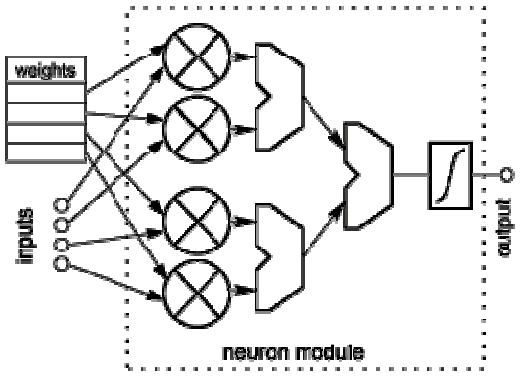

Figure.3 partial parallel processing structure of neuron

As the architecture is constructed based on timing, once the hidden layer has outputted its response, the output layer responses at every 24ns.There is flexibility to increase or decrease the number of PEs depending upon the application, since each processing element has its own memory to update its connection link weights, multiplier, accumulator and activation function.

A. Weight storage and accumulation

After training using MATLAB, the multilayer Perceptron network using back propagation algorithm, the weights on the connections between the neurons are stored in a RAM memory. The address bits, clock and enable signals needed to activate the various parts of the neural network, are generated by a state machine. A carry look-ahead adder is used to accumulate the products of input signal from the neurons of previous layer and corresponding synaptic weights.

B. Digital multiplication

Booth's Algorithm is an efficient means of multiplying signed numbers expressed in 2's complement notation. This technique recodes three consecutive bits of a multiplier (B1,) to perform operations (Y,1) as in Table 2 to generate one partial product. The booth encoder has been generalized to reduce the number of partial products in multiplication. The partial products generated are simple multiples of the multiplicand A, i.e., +A, -A, +2A, -2A and 0. Booth function basically has three basic operators, which we call 'direction', 'shift', and 'addition' operator. The direction operator determines which multiplicand is taken among normal (A) and inversed (-A) ones.

For example, if each triplet of multiplier (B) is "101", we should take the two's complement of the multiplicand A. This technique is well suited for hardware multipliers because a simple negating and/or a shifting of the multiplicand generate the partial products. Since the recoding window in a multiplier operand shifts by two bits, the modified Booth recoding leads to reduce the number of partial products by a factor of two. The multiples (Y,1) of the multiplicand are generated using simple left shift and one's complement. Based on the three consecutive bits (B1,) of the multiplier, these multiples are added using carry look-ahead adder and the output of the adder is accumulated and shifted to the register. The remaining bits of the register are filled with right shifted multiplier bits and the least significant bits of the multiplier will be lost. The sign bit of the output is obtained by adding carry out of the adder, most significant bit of multiplier and accumulator. The output of the accumulator and register generates the final product.

TABLE II: BOOTH RECODING SCHEME

Bit Pattern	Implementation
000	0
001	+A
010	+A
011	+2A
100	-2A
101	-A
110	-A
111	0

C. Activation function Implementation

The activation function of a single neuron in the neural network is determined as a function of the net output in that neuron. In multilayer Perceptron neural network, the sigmoid function is chosen for hidden layer. It is given by equation 4.

$$Y = f(x) = \frac{1}{(1+e^{-x})} \qquad (4)$$

In this paper the log sigmoid function is realized. The hardware realization of sigmoid activation function is performed based on the piecewise linear approximation [5] as shown in equation (5). The approximation is based on selecting the integer set of break points and setting the Y axis values as power of two numbers.

$$Y = \frac{\frac{1}{2} + \frac{\hat{x}}{4}}{2^{|(x)|}} = \frac{\hat{x}+2}{2^{|(x)|+2}} \qquad (5)$$

where |I(x)| is the integer part and $\hat{X}$ = x+ |(x)| , is the fractional part of the input. It may be noted that (5) requires only addition and shift operation. The hardware implementation of the piecewise linear approximation presented in [5] consists of counter and shift register. The shift register is controlled by counter which is A activated by the absolute value of the input integer bits. x +2 is shifted I(x)1+2 times. The shifter stops the operation when the counter is zero.

The first derivative of the log sigmoid function of equation (4) is given in equation (6).

$$f'(x)_{\log sig} = f(x).(1 - f(x)) \qquad (6)$$

This function limits the output in the (0, 1) range. The non-linear squashing property of the log sig allows the linear segment to adequately correct errors with weight modifications as the output rises and falls. In this paper, the piecewise linear approximation function of log sigmoid has been attempted only with combinational logic circuits simple shifting circuits. It comprises of only combinational circuits and shift operations, as shown in equation (7).

$$f(x) = \begin{cases} 0, \text{ (region 1) if } x \le -8 \\ \frac{8-|x|}{64}, (region\ 2) \text{ if } -8 < x \le -1.6 \\ \frac{x}{4} + 0.5, (region\ 3) \text{ if } |x| < 1.6 \\ 1 - \frac{8-|x|}{64}, (region\ 4) \text{ if } 1.6 \le x < 8 \\ 1, (region\ 5) \text{ if } x > 8 \end{cases} \qquad (7)$$

Figure. 4 shows the graph of this linear approximation function. The maximum absolute error (difference) between (7) and the original log sigmoid (6) is 6.59% with a mean of 2.43% over the [-8,8] range.

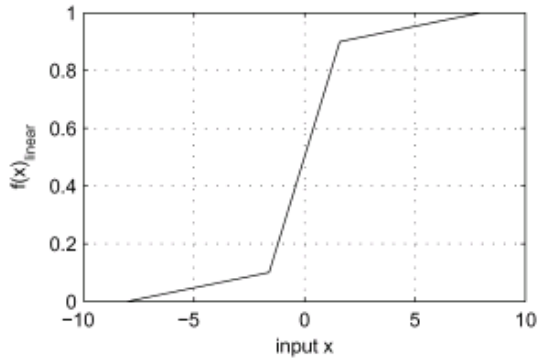

Figure.4 Implementation of linear log sigmoid function

The structural diagram of linear log sigmoid function (7) is shown in figure 5.

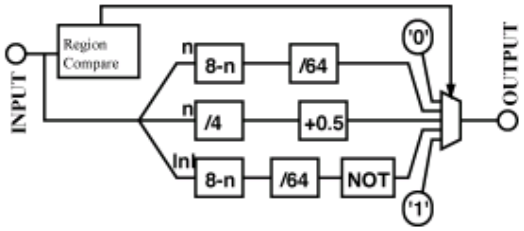

Figure.5 structural diagram of linear log sigmoid function

All the modules shown in figure 5 are implemented by manipulation of fixed point bits. The /64 module is right shift of 8 bits and /4 module is right shift of n by 2 bits. 8-n is simple inversion of n and modulus of n is done by taking the 2s complement of n. + 0.5 module is implemented by simple adder of fraction 0.5.

IV. SIMULATION AND SYNTHESIS RESULTS

The hardware design for the 2-2-1 neural network implemented in this paper is controlled by a state machine. The state transition decisions of the state machine are based on the values of the control signals. All state transitions are synchronized with the common clock. The hardware design for the 2-2-1 neural network is synthesized in VHDL using Quartus II 8.1 software. The synthesis results are taken with the help of Stratix III FPGA chip of EP3S50F484C2 which has 216 DSP block elements and 4 PLLs. The functional simulation is performed using ModelSim SE-EE 5.4E software. Timing simulation is performed using Quartus II software.

Hardware architecture presented in this paper uses four look ahead adders and four booth multipliers for implementing 2-2-1 neural network and needs totally 4935 instances. Table 5 gives the synthesis results of the Booth multiplier and carry look-ahead adder. Table.6 and Table.7 gives the synthesis results of the activation function with combinational and sequential logic circuits and the entire neural network. When

compared with sequential logic activation function, our design does not require clocked resources.

The network with 2 inputs and 3 computational neuron needs 132 ALUTs and 148 dedicated logic registers and operated at the minimum period of 34 ns. The results are compared with the actual activation function.

Figure. 6 shows the simulation results of the neuron output and Figure.7 shows the top view of the neuron implemented. Figure. 8 gives the synthesis circuit of the neuron implemented with log sigmoid function. The circuit with combinational circuit design excluding twos complementation is shown.

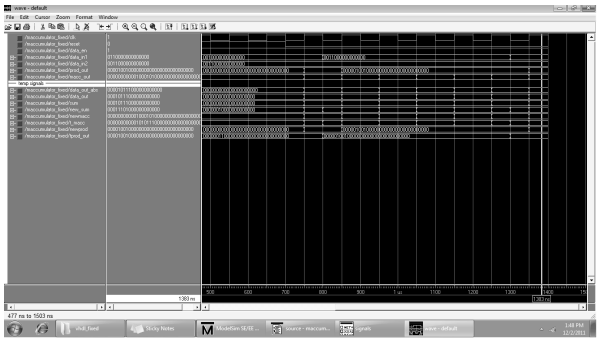

Figure.6 simulation output of a single neuron

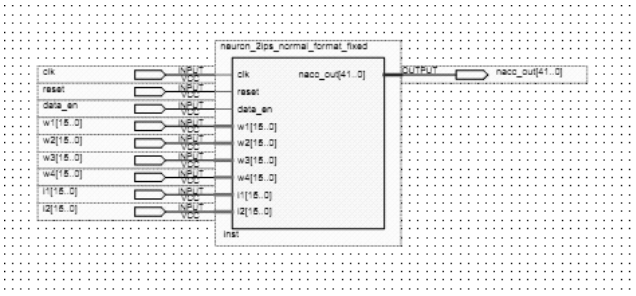

Figure. 7 Top view of neuron implementation

The hardware resources consumed for various formats of fixed point implementation is given in Table 3.

TABLE III CONSUMPTION OF HARDWARE RESOURCES

Number format	ALUTS	Dedicated logic registers	DSP blocks
1-6-9	132	148	32
1-3-12	132	155	32
1-5-10	125	162	32

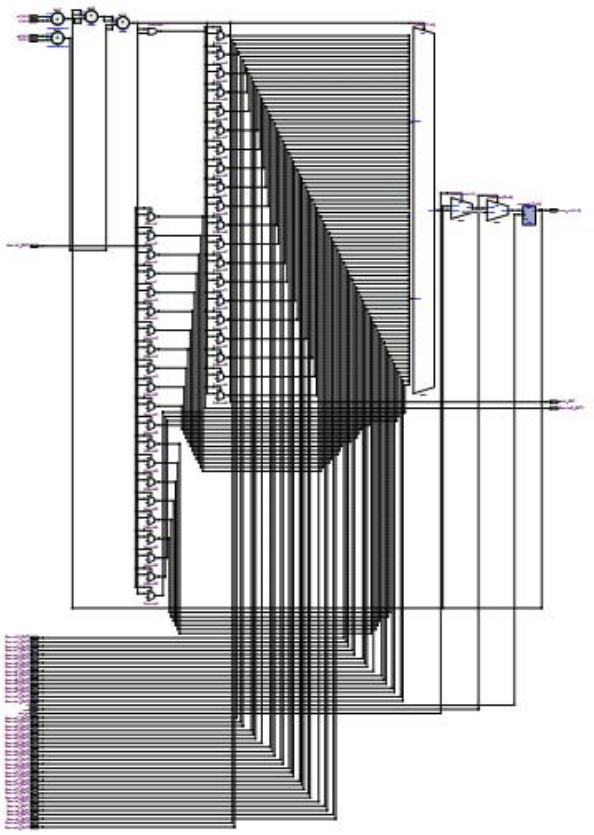

Figure.8 synthesis result of a neuron

V. CONCLUSION

Multilayer feed forward NNs have been implemented using FPGA. Hardware realization of the feed forward neural network is presented in this paper reduces the resource for implementation and time for execution. The architecture presented in this paper is fully parallel and fast activated arithmetic blocks and piecewise linear approximated activation function leads to the speed improvement. Different data formats are implemented for a 2-2-1 feed forward network.

The data distribution from previous layer to each neuron of next layer is performed simultaneously using multiplexer and the data processing in all neurons of the layer is also performed in parallel. The activation function used utilizes less number of resources compared to sequential logic design. Several additional benefits of implementing neural networks in hardware have arisen due to the reconfigurable feature of Field Programmable Gate Arrays (FPGA) as required. Flexibility in the architectural design leads this implementation as suitable for many applications.

REFERENCES

[1] S.Coric, I.Latinovic and A.Pavasovic, "A Neural Network FPGA Implementation" 5th seminar on Neural Network. Applications in Electrical Engineering, IEEE,Neurel-2000

[2] Gilberto Contreras, Patricia Nava, "Design, Implementation and Testing of an FPGA-based Neuro-Coprocessor", Proceedings of the IEEE International Conference on Electrical Engineering, May 2003

[3] Acosta Nelson & Tosini Marcelo, "Custom Architectures for Fuzzy and Neural Networks Controllers", JCS&T, Vol. 2, No. 7, October 2002

[4] A. D. Booth, "A signed binary multiplication technique,"Quarter. J. Mech. Appl. Math., vol. 4, part 2, pp. 236-240,1951.

[5] C.Alippi and G. Storti Gajani, "Simple Approximation of Sigmoidal Functions: Realistic Design of Digital Neural Networks capable of learning". IEEE, 1991

[6] Henriette Ossoinig, Erwin Reisinger, Christian Steger,Reinhold Weiss, "Design and FPGA-Implementation of a Neural Network", Technical Report TR 96/05, Institute for technical Informatics, Graz University of Technology, May 1996

[7] Jihong Liu, Deqin Liang, "A survey of FPGA based hardware implementation of ANNs", International Conference on Neural Networks and Brain, Volume 2, Page(s):915 - 918, Oct. 2005

[8] The Impact of Arithmetic Representation on Implementing MLP-BP on FPGAs: A Study Antony W. Savich, Medhat Moussa, Member and Shawki Areibi, Ieee Transactions On Neural Networks, VOL. 18, NO. 1, JANUARY 2007