has confirmed the ability of NN to control the dc drive in the same manner as FLC.

VI. CONCLUSION

An FLC has been designed to control the speed of a dc drive and successfully implemented in a NN with one hidden layer and few neurons. This demonstrates that implementing a FLC in a NN is an effective solution to alleviate the computation burden of the fuzzy logic while maintaining its human-like approach and control capabilities.

VII. APPENDIX

A 24 V, 2 A, 110 rad/s permanent magnet dc motor fed by a 24 V, 16 KHz four-quadrant chopper is considered in the paper. The motor parameters are $R = 4\ \Omega$, $L = 6$ mH, $K = 0.145$ Vs, $J = 63 \cdot 10^{-6}$ Kgm^2, and $B = 36 \cdot 10^{-6}$ Kgm^2/s. The current loop has a bandwidth of 1 KHz and a phase margin of 60°.

REFERENCES

[1] C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller—part I and II," *IEEE Trans. Syst., Man, and Cybern.*, vol. 20, no. 2, pp. 404–435, March/April 1990.

[2] T. Fukuda, "Theory and applications of neural networks for industrial control systems," *IEEE Trans. Ind. Electron.*, vol. 39, no. 39, pp. 472–489, Dec. 1992.

[3] G. Cybenko, "Approximation by superpositions of a sigmoidal function," *Mathematics of Cont., Signal, and Syst.*, vol. 2, pp. 303–314, 1989.

Hardware Implementation of an Artificial Neural Network Using Field Programmable Gate Arrays (FPGA's)

Nazeih M. Botros and M. Abdul-Aziz

***Abstract*—In this paper we present a hardware implementation of a fully digital multi-layer perceptron artificial neural network using Xilinx Field Programmable Gate Arrays (FPGAs). Each node is implemented with two XC3042 FPGAs and a 1 K × 8 EPROM. Training is done off-line on a PC. We have tested successfully the performance of the network.**

I. INTRODUCTION

In recent years, artificial neural networks have been widely implemented in several research areas such as image processing, speech processing and medical diagnoses. The reason of this widely implementation is their high classification power and learning ability [5]. At the present time most of these networks are simulated by software programs or fabricated using VLSI technology [4]. The software simulation needs a microprocessor and usually takes a long period of time to execute the huge number of computations involved in the operation of the network. Several researchers have adopted hardware implementations to realize such networks [1]–[4]. This realization makes the network stand alone and operate on a real-time fashion.

Manuscript received October 11, 1993; revised March 21, 1994.

The authors are with the Department of Electrical Engineering, Southern Illinois University, Carbondale, IL 62901 USA.

IEEE Log Number 9405109.

TABLE I
A COMPARISON BETWEEN CUSTOMIZED VLSI AND FPGA TECHNOLOGY

Field Programmable Gate Arrays	Custom VLSI
Standard Product	Custom product
Fast time-to-market	Manufacturing delays
Programmed by the user	Programmed in the factory
No NRE(Non-Recurring Engineering) Charges	NRE costs
In-circuit design verification	Not possible
Design changes anytime	NRE charge repeated and delay
Fully factory tested	User develops test
Can not be reasonably utilized in systems that need a very large number of gates, grater than 40k, the system will be bulky.	Can be utilized

Recently, implementation of Field Programmable Gate Arrays (FPGA's) in realizing complex hardware system has been accelerated [1]–[3]. Field programmable gate arrays are high-density digital integrated circuits that can be configured by the user; they combine the flexibility of gate arrays with desktop programmability. Their architecture consists mainly of: Configurable Logic Blocks (CLB's) where Boolean functions can be realized, Input/Output Blocks (IOB's) serve as input/output ports, and programmable interconnection between the CLB's and IOB's. The designer uses schematic capture software package such as OrCAD to design his system on the screen of a PC using digital components (gates or modules) from the library of the package. The designer downloads his design onto the FPGA's chip using software programs. These programs perform routing and placement and allow the designer to debug his design. Most of the commercially available FPGA's can be reprogrammed by simply down loading the new design; the old one is automatically erased. The relatively low cost and easiness of implementation and reprogramming of FPGA's in comparison with the custom VLSI technology offer attractive features for the designer. Table I compares between the main features of the FPGA's and that of customized VLSI technology.

The artificial neural network implemented in this study is a three-layer perceptron. The network classifies selected input patterns. It consists of three layers: an input layer with 5 nodes, a hidden layer with 4 nodes and an output layer with 2 nodes. The input to the network is a continuous valued vector $x_1, x_2, \ldots x_5$. The output of the network is the class of the current input. The output of node j, y_j, is calculated as follows:

$$y_j = f\left[\left(\sum_i w_{ij}\, y_i\right) - \theta_j\right] \tag{1}$$

where θ_j is the offset (bias) of node j, W_{ij} is the weight of the connection between node j and node i, and the function f is a sigmoid function:

$$f(\alpha) = \frac{1}{1 + e^{-\alpha}}. \tag{2}$$

Training of the network is carried out using the backpropagation training procedure [5]. Our ultimate target application of such network is in speech recognition. In this application, the input to the network is acoustic features representing the unknown spoken word; the output is the recognized word.

II. GENERAL ARCHITECTURE

Fig. 1 shows a photograph of the hardware realization of the artificial neural network. We have used two wire-wrapping boards each of 40 × 15 cm to build the network. Fig. 2 shows the general architecture of the network. The input layer consists of five nodes (neurons), the hidden layer consists of four nodes and the output

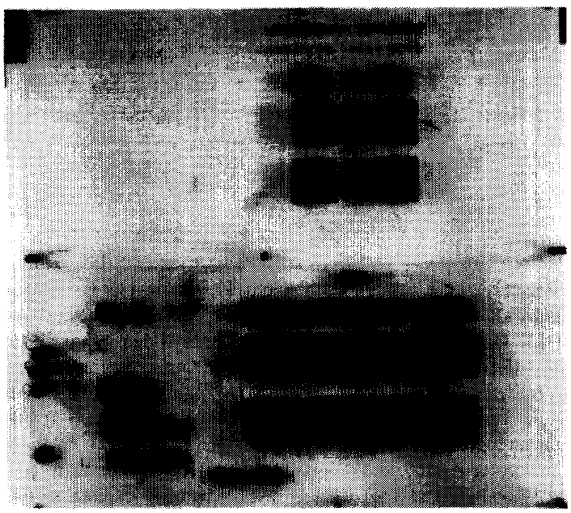

Fig. 1. A photograph of the artificial neural network.

layer consists of two nodes. These nodes are fully interconnected to each other between adjacent layers. We have trained the network off-line by simulating the network on a PC and obtain the final values of weights at the end of training session. The input layer does no processing but simply buffers the data. The nodes in the other layers form a weighted sum of their inputs. The 8-b input to the hidden and output nodes y_i is multiplied by 8-b signed weights W_{ij} to form a 16-b signed product. The products (five in the hidden and four in the output layer) and a 16-b signed bias value θ_j are accumulated into a 20-b sum. We can express the sum as: $[(\sum_i W_{ij}\, y_i) - \theta_j]$. We have scaled down the 20-b sum to a 10-b value. We have selected the 10-b scaling after running a simulated software network on a PC using the same input data of the hardware network. The 10 b are the minimum number of bits that can be retained without deteriorating the accuracy of the sum. This 10-b scaled sum serves as the address of a 1 K × 8 EPROM where a sigmoid activation function f is realized as a lookup table. The activation function produces an 8-b output. We implement two's complement representation to handle the multiplication and additions of negative numbers. We can express the output as: $f[(\sum_i W_{ij}\, y_i) - \theta_j]$ where $f(\alpha) = \frac{1}{1+e^{-\alpha}}$. Fig. 3 shows a schematic diagram of the processing flow for the hidden node. We have used two XC3042 FPGA's and a 1 K × 8 b EPROM to build each node in the network. A single XC3042 FPGA costs about $25 and contains 4200 gates with 144 CLB's and 96 I/O blocks; its physical size is 4 × 4 × .5 cm approximately. The first FPGA carries input latches and multipliers. The second carries a 20-b fast adder/accumulator circuit and a scaling logic. These two FPGA's compute the weighted sum of five inputs (four in the output layer) and the bias value and then scale the result. The EPROM holds the values of the activation functions. We use Xilinx FPGA's of 50 MHz toggle rate throughout the design since they are sufficient to keep up with the 450 ns EPROM's used in the circuit. The system clock is 4 MHz which is the maximum speed that could be achieved due to the use of slower EPROM's and two FPGA's per node. A microprogrammed controller drives the entire network. The controller generates a proper sequence of signals to control the timing for both layers. Computations for nodes of the same layer are done in parallel.

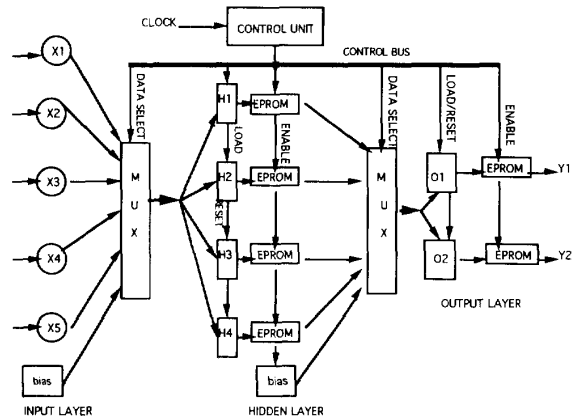

Fig. 2. General architecture of the network.

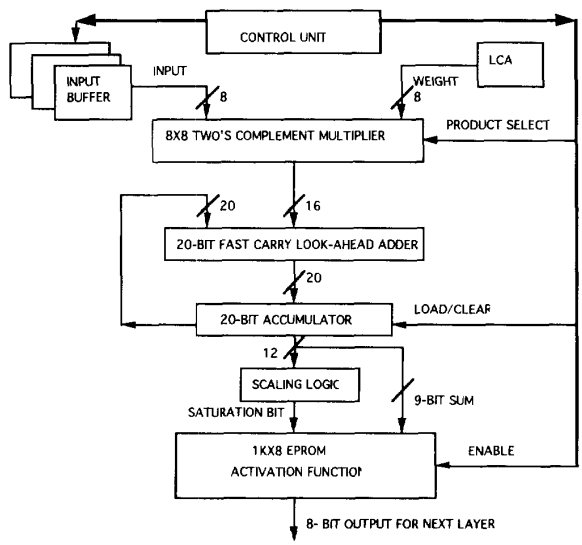

Fig. 3. A schematic diagram of the hidden node.

III. Details of the Architecture

A. Multiplication

Since the weights and biases are constants (pre-determined), multiplication of any number with them can be done in a look-up table fashion. We have programmed the CLB's to realize these look-up tables. Multiplying an 8-b number by an 8-b constant produces a sixteen bit product. We break the 8 × 8 multiplication into two 8 × 4 multiplications and one addition. We shift the most significant partial product (8 × 4) by four bits before adding it to the least significant partial product. We realize the shift by physically shifting (routing) the most significant bits. We use one CLB to generate 2 b of the product. Generation of the 12 b of the partial product requires 6 CLB's.

B. Summation

The next task performed by the nodes is to produce the sum of the partial products into a single 20-b sum. We have selected the 20 b so that no overflow can happen. We have designed a 20-b fast carry look-ahead adder to carry out the summations. Each node in

the hidden layer adds up 10 16-b partial products and a bias into a 20-b positive edge triggered accumulator. The output nodes perform the same but for eight partial product.

C. Scaling and Activation Function

The final task performed by the nodes of hidden and output layers is the scaling and the application of activation (sigmoid) function. A 20-b accumulator stores the final result of addition of all the partial products and as well as the bias value. An investigation of the behavior of the sigmoid function shows that it saturates to approximately 1.0 when $\alpha \geq +7$ and saturates to approximately 0.0 if $\alpha \leq -8$. To utilize this restricted range of $\alpha(-8 \leq \alpha \leq 7)$ we need 9 b; accordingly we scale down the 20 b to 9 b.

IV. Results

We have written a software program to simulate the network and its operation. We have applied the same input patterns to both the hardware and the simulated software network and calculated the outputs. Table II shows the outputs y_1 and y_2 of both networks. Some of the input patterns represent acoustic features (energy and zero crossings) of spoken words; the other patterns are selected randomly to observe the behavior of the network. As shown in this table the hardware network performs correctly. The hardware implementation computes 4 million interconnections (approximately 70,000 decisions) per s. This speed allows the implementation of the network in real-time applications.

TABLE II
Results of Software and Hardware Networks

Input Test Patterns					Output Y1		Output Y2	
					Soft W.	Hard W.	Soft W.	Hard W.
7	20	30	48	58	.98967	.99218	.00806	.00781
3	6	19	21	59	.99620	1	.01530	.01562
14	26	39	45	50	.98967	.99218	.00806	.00781
5	19	7	29	37	.98967	.99218	.00806	.00781
10	18	29	53	61	.98967	.99218	.00806	.00781
40	35	31	27	20	.03114	.03125	.95791	.96003
75	66	50	41	19	.03114	.03125	.95791	.06093
31	20	15	8	1	.03114	.03125	.95791	.96093
62	50	55	67	70	.74267	.74218	.20795	.20312
40	32	22	44	14	.03114	.03125	.95791	.96093
20	86	54	38	97	.98967	.99218	.00806	.00781
16	8	4	2	1	.03115	.03125	.95791	.96093
1	2	4	8	16	.99499	.99218	.01238	.01562
19	27	89	34	63	.98967	.99218	.00806	.00781
19	27	89	63	34	.56162	.51562	.41332	.40625
63	34	89	27	19	.03114	.03125	.95791	.96093
34	63	89	27	19	.03114	.03125	.05791	.96093
200	180	100	80	5	.03114	.03125	.95791	.06093
200	180	100	5	80	.03114	.03125	.95791	.96093
5	80	100	180	200	.98967	.99218	.00806	.00781
8	4	1	16	4	.60804	.60937	.42076	.40625

V. Discussion and Conclusion

We have presented a successful hardware implementation of a simple artificial neural network. We can expand the implementation to realize more complex networks. Reconfigurability and adaptability are the main features of the hardware. For a new application, only the weights, biases, and scaling parameters need to be reconfigured on the CLB's without changing the basic design. We can expand the network easily by just adding more nodes with the same design. We have found Xilinx FPGA's and other alike FPGA's to be efficient tools for the design of neural networks. Their reconfigurability and desktop programmability allow design changes at the user's terminal, thereby avoiding the fabrication cycle times and non-recurring engineering charges.

Although the use of two XC3042-50MHz FPGA's and a 1 K × 8 EPROM per node makes the network bulky, we have found that its size and speed can be greatly improved by using higher density FPGA's. FPGA XC3090, which has 320 CLB's and 144 IOB's can easily accommodate the circuits in the two XC3042 used in this study. It will also significantly reduce the size as well as increase the speed by eliminating the 55-ns delay between the I/O pins of the two FPGA's. RAM's can be implemented inside the 4 or 5 K FPGA's series. These RAM's can be programmed for sigmoid lookup tables and can be downloaded with the bit stream during configuration to further increase the speed and reduce the size to one chip per node. Use of pipeline techniques in each node as well as between successive layers of the network and higher speed EPROM's and FPGA's can also greatly increase the speed. Very high density FPGA's may provide room to build more than one neuron in one chip.

References

[1] C. E. Cox and W. Ekkehard Blanz, "Ganglon—A fast hardware implementation of a connectionist classifier," *IEEE-CICC*, Phoenix, AZ, 1991.

[2] N. Botros and M. Abdul-Aziz, "Hardware implementation of an artificial neural network," in *Proc. IEEE Int. Conf. on Neural Networks*, San Francisco, CA, March 1993, vol. 2, pp. 1252–1257.

[3] B. Fehér, "Resonator based digital filters using field programmable gate array elements," *Fifth Ann. IEEE Int. ASIC Conf.*, Rochester, NY, Sept. 1992.

[4] S. Satyanarayana, Y. Tsividis, and P. Graf, "A reconfigurable VLSI neural network," *IEEE J. Solid-State Circuits*, vol. 25, pp. 849–55, June 1990.

[5] R. Lippmann, "An introduction to computing with neural nets," *IEEE-ASSP*, Mag., pp. 4–22, April 1987.