2006 International Joint Conference on Neural Networks
Sheraton Vancouver Wall Centre Hotel, Vancouver, BC, Canada
July 16-21, 2006

Implementation of Artificial Neural Network for Real Time Applications Using Field Programmable Analog Arrays

Puxuan Dong, *Student Member, IEEE,* Griff L. Bilbro, Mo-Yuen Chow, *Senior Members, IEEE*

***Abstract* — This paper presents a method of realizing artificial neural networks (ANNs) hardware implementation using field programmable analog arrays (FPAAs). A simplified realization for neurons with piecewise linear activation functions is used to reduce the complexity of the neural network architecture. A feedforward neural network is implemented using multi-chip FPAAs. Anadigm's commercially available AN221E04 FPAA chips are adopted as the platform for simulation and experiments. The FPAA based ANN classifies two groups of data with zero error at a speed of 6.0 Million Connections Per Second (MCPS). The result is more than 1400 times faster than software implementation. The ANN architecture is also expandable to perform more complicated tasks by incorporating more FPAA chips into the implementation. The programmability of the FPAA makes rapid prototyping possible.**

Index Terms — **field programmable analog arrays, neural network hardware, rapid prototyping**

I. Introduction

Artificial neural networks (ANNs) have been playing an increasingly important role in areas such as robotics [1], process control [2-3], and motor fault detection [4-6]. Both software and hardware based approaches have been used for implementing ANNs. In general, software instructions executed serially cannot take advantage of the inherent parallelism of ANN architectures. Hardware implementations of neural networks promise higher speed operation when they can exploit this massive parallelism. Different hardware implements of neural network have been reported [7-25]. Other than the FPGA based approaches [10, 11, 18, 24], most of the hardware implementations provide no programmability. Reconfigurability of an ANN is desirable since many ANN applications, e.g., robots performing different tasks in different environments may benefit from different neural network topologies (e.g., different number of hidden nodes). The best choices for neural network implementations that achieve both high speed and rapid prototyping appear to be programmable hardware approaches like field programmable gate arrays (FPGAs) and field programmable analog arrays (FPAAs). Compared to digital hardware, FPAAs have the advantage of interacting directly with the real world because they receive, process, and transmit signals totally in the analog domain (without the need to do A/D, D/A conversions) and are suitable for real time applications. As reported in [26] on controlling a path-tracking unmanned ground vehicle, an FPAA can easily outperform the digital hardware by processing the signal 8,000 times faster. Other FPAA applications, including a voltage-to-frequency converter and a Hodgkin-Huxley neuron simulator, have been reported [27-28].

Section II of this paper proposes a simple realization of layered neural networks appropriate for FPAAs. Section III applies the neural network architecture simplification method to a multi-chip FPAA based neural network to classify the elements of a data set containing two groups of data. Section IV analyzes the speed performance of the FPAA implementing the ANN by comparing it to software implementation. Section V gives some concluding remarks.

II. Neural Network Architecture Simplification in FPAA

A. The piecewise linear activation function

In the ANN, the output of a neuron is computed by applying its activation function to a weighted sum of its inputs. Some activation functions such as hyperbolic tangent and sigmoid are expensive for digital hardware implementation. To reduce the cost for implementation, the piecewise linear activation function has been used to approximate sigmoid activation function [29]. We chose the Piecewise Linear (PL) activation function for the neurons in the hidden layer of our neural network architecture because it is naturally suited for applying FPAA hardware to the problem of interest (to be described in later sections).

A neural network must be trained to reflect or to generalize

Manuscript received on Jan. 2006.

Puxuan Dong is with Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695, USA (phone: 919-946-2866; e-mail: dongpuxuan@gmail.edu).

Griff Bilbro is with Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695, USA (e-mail: glb@ncsu.edu).

Mo-Yuen Chow is with Department of Electrical and Computer Engineering, North Carolina State University, Raleigh, NC 27695, USA (e-mail: chow@ncsu.edu).

0-7803-9490-9/06/$20.00/©2006 IEEE

a desired relationship between inputs and outputs. During the back propagation training process in a neural network, the error signal at the output of the neuron *j* at iteration *n* (i.e., presentation of the n^{th} training example) is defined by

$$e_j(n) = d_j(n) - y_j(n), \tag{1}$$

where $d_j(n)$ is the desired response of neuron *j* and is used to compute $e_j(n)$, $y_j(n)$ is to the function signal appearing at the output of neuron *j* at iteration *n*. Let $\varphi_j(\bullet)$ be the activation function; then the synaptic weight $\Delta w_{ji}(n)$ change is:

$$\Delta w_{ji}(n) = \eta \delta_j(n) y_i(n), \tag{2}$$

where

$$\delta_j(n) = e_j(n)\varphi_j'\left(v_j(n)\right), \tag{3}$$

is called the local gradient and η is the learning rate. In equation (3),

$$v_j(n) = \sum_{i=0}^{m} w_{ji}(n) y_i(n), \tag{4}$$

and w_{ji} denotes the synaptic weight connecting the output neuron *i* (there are *m* inputs) to the input of neuron *j* at iteration *n*. The PL activation function is given by

$$\varphi_j(x) = \begin{cases} w_+, & \mathbf{w}_l^T \mathbf{x} + w_0 \geq w_+ \\ \mathbf{w}_l^T \mathbf{x} + w_0, & w_- < \mathbf{w}_l^T \mathbf{x} + w_0 < w_+ \\ w_-, & \mathbf{w}_l^T \mathbf{x} + w_0 \leq w_- \end{cases}, \tag{5}$$

where $\mathbf{x}^T = [x_1, x_2, ...x_d] \in R^d$ is the input vector and $\mathbf{w}^T = [w_0, w_1, ...w_d, w_+, w_-] \in R^{d+3}$, with $w_+ = +1$ and $w_- = -1$, is the parameter vector that characterizes the node function. Figure 1 shows the 3D view of input-output relationship of a neuron of 2 inputs with piecewise linear activation function.

Although the PL activation function is less popular than the hyperbolic tangent activation function, the piecewise nature has attractive features such as ease of implementation and amenability to VLSI implementation [30-31]. It is also simpler to find $\varphi_j'(\bullet)$ in equation (5) since it requires only addition, multiplication and comparison operations in contrast to the trigonometric function that must be evaluated for the hyperbolic tangent function.

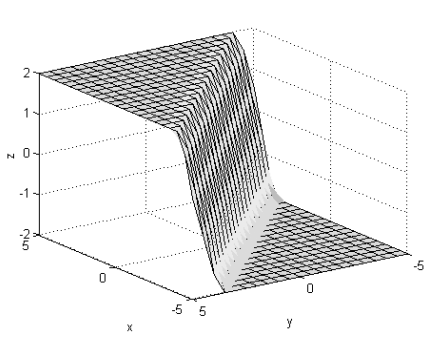

Figure 1. The piecewise linear (PL) activation function for $\mathbf{w}^T = [1,1]$ in two dimensions.

B. Implementing the PL function on FPAA

This section develops a realization of the standard PL activation function that uses two gain amplifier functional blocks. A standard PL function has the following form:

$$PL(x) = \begin{cases} -1 & x < -1 \\ x & -1 \leq x < +1 \\ +1 & x \geq +1 \end{cases}. \tag{6}$$

In a FPAA circuit which saturates symmetrically at V_+ and V_-, where $V_+ = V_0 > 0$, $V_- = -V_0 < 0$, a standard PL activation function can be obtained with two cascade gain stages G_1 and G_2 if V_0>1, where $G_1 = \frac{V_+ - V_-}{2} = V_0$ and $G_2 = \frac{2}{V_+ - V_-} = \frac{1}{V_0}$ (which will be explained in the following paragraphs). Note that the product of G_1 and G_2 is unity and $G_1 > 1 > G_2$.

Since the circuit saturates at V_+ and V_-, the relationship between input voltage *x* and output voltage $F_1(x)$ of a "through" circuit is:

$$F_1(x) = \begin{cases} -V_0 & x < -V_0 \\ x & -V_0 \leq x < V_0 \\ V_0 & x \geq V_0 \end{cases}. \tag{7}$$

A gain stage G_1 after $F_1(x)$ establishes the following relationship between the new output $F_2(x)$ and *x*:

$$F_2(x) = F_1(x) \times G_1 = \begin{cases} -V_0 & x < -1 \\ G_1 \times x & -1 \leq x < 1 \\ V_0 & x \geq 1 \end{cases}. \tag{8}$$

Adding another gain stage G_2 after $F_2(x)$ gives the following relationship between $F_3(x)$ and *x*:

$$F_3(x) = F_2(x) \times G_2 = \begin{cases} -1 & x < -1 \\ x & -1 \leq x < +1 \\ +1 & x \geq +1 \end{cases}. \tag{9}$$

Thus the standard piecewise linear activation function is obtained by inserting these two particular gain stages between the input and the output of a through circuit. Figure 2 shows

the three functions (with using $V_0 = 2.5$).

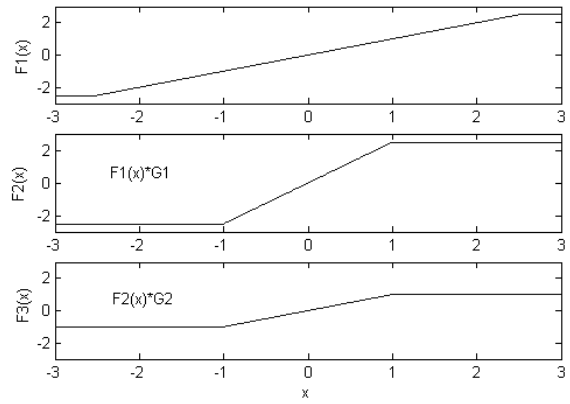

Figure 2. Obtaining the PL function using two gain stages.

C. *Merging the gain stages of cascade blocks on the FPAA*

As shown in Figures 3 and 4, the neural network can be simplified further by merging the two gain blocks G_1 and G_2 into the input and output weights of the neurons. G_1 and G_2 form the standard piecewise linear transfer function for neuron *j*. The neural network architecture in Figure 2 can be simplified by multiplying every weight of neuron *j* by G_1 and multiplying w_{k1} by G_2 as shown in Figure 3. As a result, addition and multiplication are the only two operations required for a neural network implementation on the FPAA. The addition operation is performed by inverting sum amplifier blocks. The weights that a neuron uses to compute the weighted sum of its inputs are realized as the gain parameters of the inverting sum amplifiers on the FPAA. These weights are obtained from an offline training procedure using MATLAB/SIMULINK software to accurately simulate the network topology and to optimize the weights. The optimal weights are downloaded to the Anadigm FPAA chips for the corresponding real-time operation, such as controlling a mobile robot or, in this paper, classifying data points.

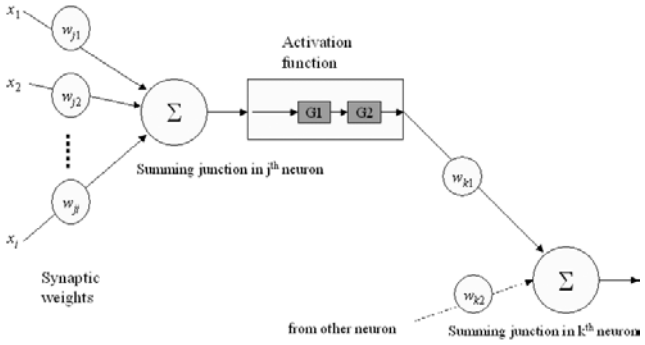

Figure 3. Neural network architecture with unmerged gain blocks.

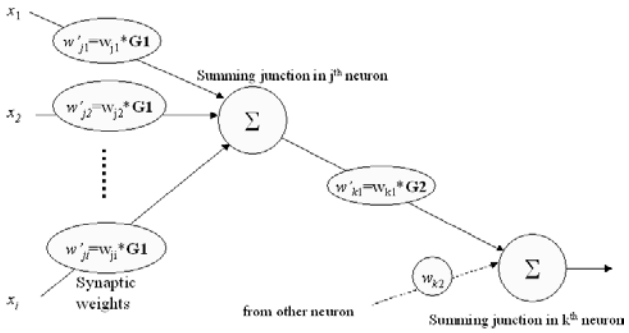

Figure 4. Neural network architecture with merged gain blocks.

Note that merging G_1 into w_{ji} will not change the input to G_2 in Figure 3. In the meantime, merging G_2 into w_{kl} will not change the input to the summing junction of k^{th} neuron. The merging procedure for G_1 implemented in FPAA is depicted in Figure 5. The circuits shown reflexes the FPAA circuits except that resistors are replaced by equivalent switched capacitors and signals are differential inside the actual FPAA. The top circuit is the one before merging and the bottom one is after merging.

This section explains why voltage saturation at the upper and lower limits of circuit does not invalidate our merging simplification. In the circuits shown in Figure 5, G_1 is equal to $R5/R4$. In all cases the output of the Op Amps saturates at $\pm v_0$ (the Op Amps are assumed to be ideal).

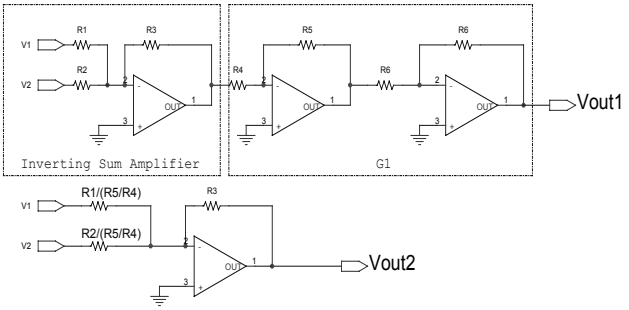

Figure 5. Simplifying the FPAA circuit by merging G_1 into the inverting sum amplifier.

The output V_{out1} of the circuit with unmerged gain blocks is

$$V_{out1} = \begin{cases} V_0, & R_3(\frac{V_1}{R_1}+\frac{V_2}{R_2}) \geq 1 \\ -\frac{R_5 R_3}{R_4}(\frac{V_1}{R_1}+\frac{V_2}{R_2}), & -1 < R_3(\frac{V_1}{R_1}+\frac{V_2}{R_2}) < 1 \\ -V_0 & R_3(\frac{V_1}{R_1}+\frac{V_2}{R_2}) \leq -1 \end{cases} \quad (10)$$

The output V_{out2} of the circuit with merged gain blocks is

$$V_{out2}=\begin{cases} V_0, & \frac{R_5R_3}{R_4}(\frac{V_1}{R_1}+\frac{V_2}{R_2})\geq V_0 \\ -\frac{R_5R_3}{R_4}(\frac{V_1}{R_1}+\frac{V_2}{R_2}), & -V_0<\frac{R_5R_3}{R_4}(\frac{V_1}{R_1}+\frac{V_2}{R_2})<V_0 \\ -V_0, & \frac{R_5R_3}{R_4}(\frac{V_1}{R_1}+\frac{V_2}{R_2})\leq -V_0 \end{cases} . \quad (11)$$

Since $\frac{R_5}{R_4}=V_0$, thus $R_3(\frac{V_1}{R_1}+\frac{V_2}{R_2})\geq 1$ is equivalent to $\frac{R_5R_3}{R_4}(\frac{V_1}{R_1}+\frac{V_2}{R_2})\geq V_0$, $-1<R_3(\frac{V_1}{R_1}+\frac{V_2}{R_2})<1$ is equivalent to $-V_0<\frac{R_5R_3}{R_4}(\frac{V_1}{R_1}+\frac{V_2}{R_2})<V_0$ and $R_3(\frac{V_1}{R_1}+\frac{V_2}{R_2})\leq -1$ is equivalent to $\frac{R_5R_3}{R_4}(\frac{V_1}{R_1}+\frac{V_2}{R_2})\leq -V_0$. Thus the two circuits are equivalent.

Similar proof can be applied to the merging of G2 into the subsequent functional blocks of the FPAA.

III. Multi-chip FPAA based neural network classifying 2 groups of data

A. The two classes of data

The 8-point version of the "alternate labels" problem [32] is chosen to demonstrate the speed advantage of using FPAA implementation. The problem has two classes of data points. Let the two class be A and B. Each class has 4 data points alternating with the 4 data points of the other group in two dimensions. Each data point is represented in the usual way as an ordered pair of numbers as shown in Figure 6; we call the elements of the n^{th} pair x_n and y_n (n = 1, 2, …, 8). All 8 data points have the same the y values thus $y_1 = y_2 = \ldots = y_8$. "a" (represented by squares) and "b" (represented by circles) are two different real numbers representing the two classes. Without loss of generality, we can assume that the interval between successive x_n's is constant, say 0.4, and that the values of all y_n are the same, say 1.0, as shown in Table I. A feedforward neural network with several neurons in the hidden layer can generate the decision boundaries.

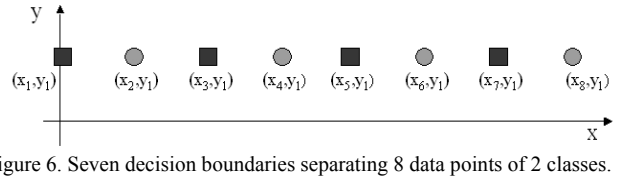

Figure 6. Seven decision boundaries separating 8 data points of 2 classes.

B. Classifying the two groups of data

Table I shows the input and output values of the 8 data points used in our simulation and experiment.

TABLE I.
TWO CLASSES OF DATA: CLASS A = 0 AND CLASS B = 1.

Input x	0	0.4	0.8	1.2	1.6	2.0	2.4	2.8
Input y	1	1	1	1	1	1	1	1
Output z	0	1	0	1	0	1	0	1

The neural network needs to implicitly generate the desired decision boundaries based the input data pairs in order to make proper classification. To classify these 8 data points, a 2-5-1 neural network is trained using the training data in Table I in MATLAB to obtain the weights. The neural network has 5 neurons in the hidden layer which has the PL activation functions and one output neuron to construct a linear combination of the outputs of the 5 hidden neurons. Before the neural network is mapped onto the FPAA, it is simulated in MATLAB/SIMULINK to verify the separation capabilities of the network. As a result, the neural network achieves 100 percent classification accuracy as shown in Figure 7 with the output (z) threshold chosen to be 0.5.

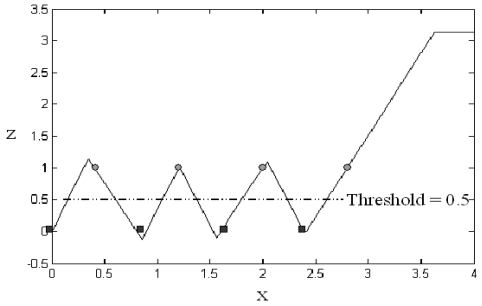

Figure 7. The output of the trained neural network view in x-z plane at y=1.
(Simulated in MATLAB/SIMULINK from the MathWorks, Natick, MA, USA)

C. Simulation and experimental results

In this section we map the trained neural network onto the FPAA devices. The FPAA used in our simulation and experiments is the AN221E04 from Anadigm Inc. The AN221E04 is a dynamically reconfigurable analog chip composed of op-amps, comparators and switched programmable capacitors. FPAA technology enables rapid-prototyping of analog circuits by programming the configurable analog modules supported by the chip, such as gain blocks, inverters, summing inverters, adders, multipliers, integrators, quadratic/linear analog filter blocks, and sine wave generators. With the aid of design software AnadigmDesigner 2, the FPAA can translate complex analog circuits into the simple set of system/block level design instead of transistor level design, and thus gives designers the analog equivalent of an FPGA. Moreover, it places analog functions under real-time software control within the system.

The 2-5-1 neural network with parameters obtained by the MATLAB/SIMULINK model was mapped onto the FPAA programmed using only Inverting Gain Amplifier functional blocks (represented by "Inv G" in the Figure) and Inverting Sum Amplifier functional blocks (represented by "Inv Sum" in Figure 8). The element "b" in the figure is the trained bias input for each neuron. We programmed the Inverting Sum Amplifier to accept at most 3 inputs; and several of the

Inverting Sum Amplifiers are cascaded between the hidden layer and the output layer to realize the sum operation for the output neuron. The accumulated sign-flips of Inverting Gain Amplifiers are correctly accounted for by additional inversions when necessary. For example, in Figure 8, the trained weight for the Y input to the first neuron in the hidden layer has the negative sign but there are 4 Inverting Sum Amplifiers between the input Y and the final output which provides a positive sign; thus an Inverting Gain Amplifier is needed in the signal path to generate the negative sign. The exact location of the Inverting Gain Amplifier in the signal path is chosen based on the available programming resources of each chip.

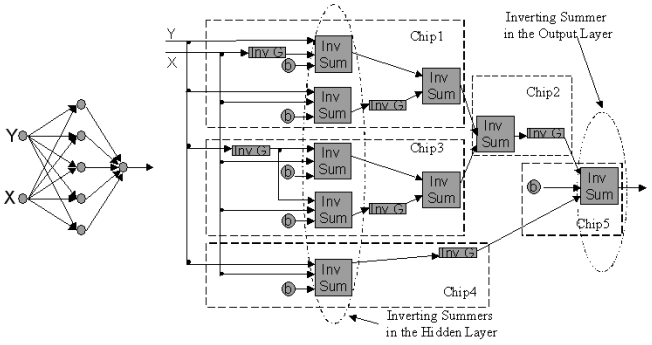

Figure 8. Constructing a 2-5-1 neural network using configurable analog modules of the FPAA.

Five AN221E04 chips are integrated together to realize the 2-5-1 neural network as shown in Figure 9. The network is decomposed into five modules as shown in Figure 8 and each module is encapsulated in one chip. The simulation result using AnadigmDesigner 2 is shown in Figure 10. Input Y is a test signal of 1*v* constant voltage and Input X is the triangular voltage input peaking at 3*v*. Setting the output threshold at 0.5*v*, the network classifies the data with 100% accuracy. The experimental result showing more details of the classification is shown in Figure 11, which is the oscilloscope screen shot of the experiment result.

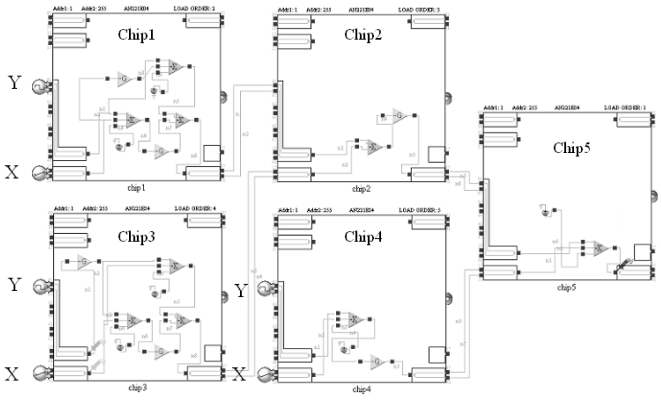

Figure 9. The multi-chip FPAA based neural network programmed using software AnadigmDesigner 2.

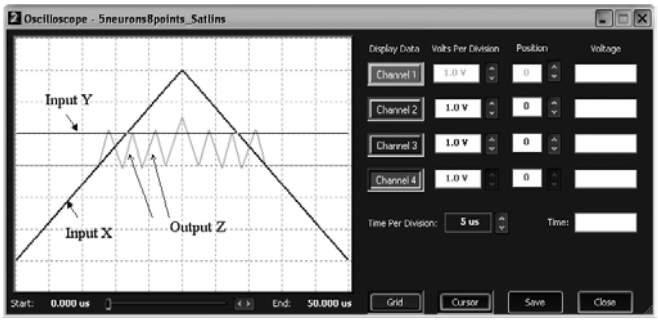

Figure 10. The simulation results of neural network classifying two classes of data.

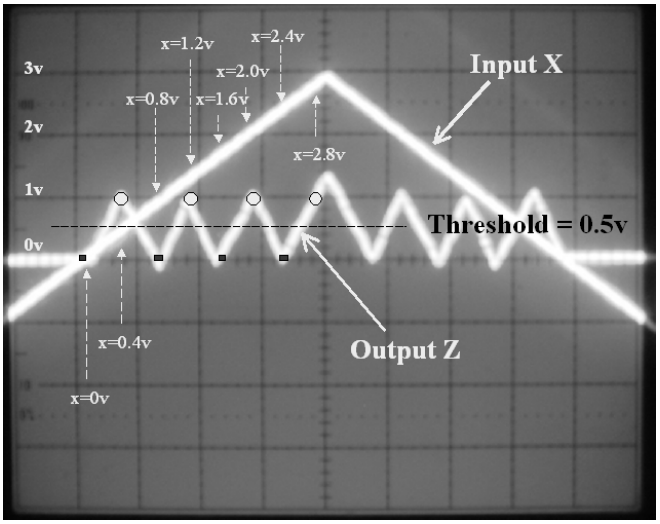

Figure 11. Experimental results of neural network classifying two classes of data.

As shown in Figure 11, the neural network trained from a 2-5-1 neural network separates the two classes of data into 2 regions and makes correct classifications of all data points with the threshold chosen to be 0.5*v*. We would also like to evaluate the speed performance for our multi-chip neural network using the standard neural network hardware measuring criteria: Millions of Connections Per Second (MCPS) [33]. The measured delay from the network input to the network output is 2.5 microseconds and there are 15 connections, yielding 6.0 MCPS in actual measured speed performance. Figure 12 shows the 5 FPAA evaluation boards for the experiments.

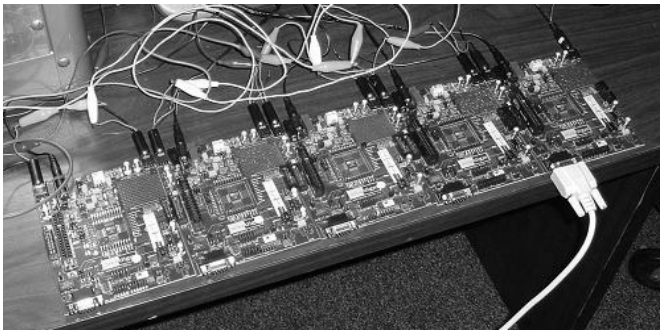

Figure 12. The five FPAA evaluation boards for the experiment

IV. Analysis of Speed Performance

To compare the speed performance of neural network implementation using FPAA to the software implementation (MATLAB on an Intel Celeron 2 GHz machine), neural networks with 4 architectures: 2-2-1, 2-3-1, 2-4-1 and 2-5-1 are implemented using both FPAA and the software. The measured implementation time of the neural network (time delay from the input to the output of the network) of all four architectures is 2.5 microseconds (error bound is below 0.5%) on the FPAA, independent of the number of neurons in the hidden layer. One the other hand, the software implementation time is more than 3.6 milliseconds. As a result, the FPAA implements the neural network more than 1400 times faster than the software implementation. Figure 13 shows the relationship between the software implementation time and the number of the neurons in the hidden layer of the network. It is shown that adding neurons into the hidden layer increases the overall software implementation time. This is because software instructions that are executed serially cannot take advantage of the inherent parallelism of ANN architectures as FPAA does. Note the experiment results are to qualitatively show how the implementation time is affected with different neuron numbers instead of showing the exact functional relationship between software implementation time and the neuron numbers. All in all, the FPAA implementation of the neural network has superior performance over software implementation.

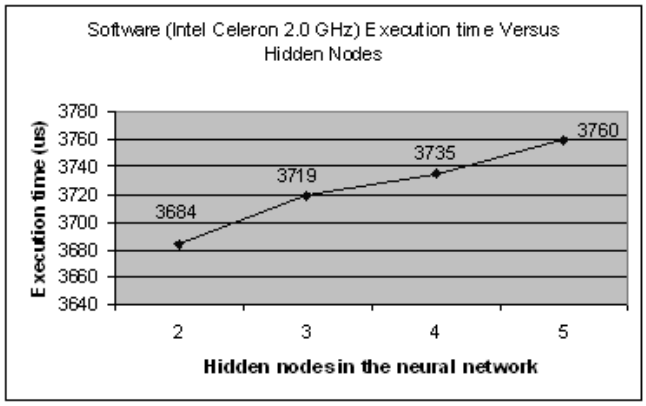

Figure 13. Neural network execution time by software versus number of hidden nodes.

V. Discussion on the Scalability of the Structure

The structure is scalable for the neural network which has same number of inputs/outputs and more neurons in the hidden layer. More summer blocks are required to obtain the final output. The positions of inverting gain blocks may need to be adjusted according the signs of the weights. An example of scaling is shown in Figure 14.

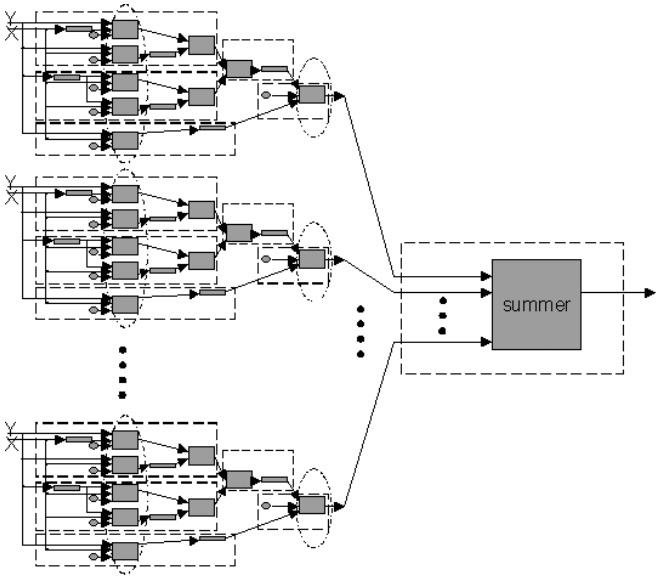

Figure 14. Scalability of the FPAA based ANN.

VI. Conclusion

This paper demonstrates the hardware implementation a feedforward artificial neural network using low-cost commercially available FPAA chips. We proposed a simplified realization for neurons with piecewise linear activation functions and thereby reduced the complexity of the neural network architecture correspondingly. Our final ANN requires only two types of analog function blocks: the Inverting Gain Amplifier and the Inverting Sum Amplifier. In this effort, we did not require the many other functional blocks available on the Anadigm FPAA chip, but these additional resources can be combined with ANNs for conventional signal processing at the input or output of an ANN. The hardware neural network correctly performs a classification task at the speed of 6.0 MCPS. We used 5 chips to realize a 2-5-1 ANN, but more complicated network architectures can be realized by integrating more Anadigm's AN221E04 chips. We found that FPAA-based ANNs are convenient to implement, fast to operate and scalable. We conclude that the proposed approach to realizing ANNs is suitable for real time applications.

References

[1] F. L.Lewis, "Neural-network control of robot manipulators," *IEEE Expert*, pp. 64-75, June 1996.

[2] J. Teeter and M.-Y. Chow, "Application of Functional Link Neural Network to HVAC Thermal Dynamic System Identification," *IEEE Transactions on Industrial Electronics*, vol. 45, no. 1, pp. 170-176, 1998.

[3] M.-Y. Chow and J. Teeter, "A Knowledge-Based Approach for Improved Neural Network Control of a Servomotor System with Nonlinear Friction Characteristics," *Mechatronics*, vol. 5, no. 8, pp. 949-962, 1995.

[4] B. Ayhan, M.-Y. Chow, and M.-H. Song, "Monolith and Partition Schemes with LDA and Neural Networks as Detector Units for Induction Motor Broken Rotor Bar Fault Detection," *KIEE International Transactions on Electrical Machinery and Energy Conversion Systems*, June 1, 2005 (invited).

[5] M.Y. Chow, “Methodologies of Using Artificial Neural Network and Fuzzy Logic Technologies for Motor Incipient Fault Detection,” *World Scientific Publishing Co*. Pte. Ltd., 1998.
[6] M.-Y. Chow, G. Bilbro, and S. O. Yee, “Application of Learning Theory to a Single Phase Induction Motor Incipient Fault Detection Artificial Neural Network,” *International Journal of Neural Systems*, vol. 2, no. 1&2, pp. 91-100, 1991.
[7] M. Holler, S. Tam, H. Castro and R. Benson, “An electrically trainable artificial neural network (ETANN) with 10240 `floating gate' synapses ,” *Neural Networks, 1989. IJCNN, International Joint Conference on*, pp. 191 - 196 vol.2, 18-22 June 1989.
[8] S. Tam, B. Gupta, H. Castro and M. Holler, “Learning on an Analog VLSI Neural Network Chip,” *Proceedings of the IEEE International Conference on Systems, Man & Cybernetics*, 1990.
[9] Y. Maeda, H. Hirano and Y. Kanata, “AN Analog Neural Network Circuit with a Learning Rule via Simutaneous Perturbation,” *Proceedings of the IJCNN-93-Nagoya*, pp. 853-856, 1993.
[10] S. S. Kim and S. Jung, "Hardware implementation of a real time neural network controller with a DSP and an FPGA," *presented at Robotics and Automation, 2004. Proceedings. ICRA '04. 2004 IEEE International Conference* on, 2004.
[11] W. Qinruo, Y. Bo, X. Yun, and L. Bingru, "The hardware structure design of perceptron with FPGA implementation," *presented at Systems, Man and Cybernetics, 2003. IEEE International Conference on*, 2003.
[12] S. B. Yun, Y. J. Kim, S. S. Dong, and C. H. Lee, "Hardware implementation of neural network with expansible and reconfigurable architecture," *presented at Neural Information Processing, 2002. ICONIP '02. Proceedings of the 9th International Conference on*, 2002.
[13] H. Withagen, “Implementing Backpropagation with Analog Hardware,” *Proceedings of the IEEE ICNN-94-Orlando Florida*, pp. 2015-2017, 1994.
[14] T. Szabo, L. Antoni, G. Horvath, and B. Feher, "A full-parallel digital implementation for pre-trained NNs," presented at Neural Networks, 2000. IJCNN 2000, *Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, 2000.
[15] S. Popescu, "Hardware implementation of fast neural networks using CPLD," *presented at Neural Network Applications in Electrical Engineering, Proceedings of the 5th Seminar on*, 2000.
[16] B. Girau, "Digital hardware implementation of 2D compatible neural networks," *presented at Neural Networks, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, 2000.
[17] H. Abdelbaki, E. Gelenbe, and S. E. EL-Khamy, "Analog hardware implementation of the random neural network model," *presented at Neural Networks, Proceedings of the IEEE-INNS-ENNS International Joint Conference on*, 2000.
[18] J. Zhu, G. J. Milne, and B. K. Gunther, "Towards an FPGA based reconfigurable computing environment for neural network implementations," *presented at Artificial Neural Networks, Ninth International Conference on*, 1999.
[19] J. Liu and M. Brooke, "A fully parallel learning neural network chip for real-time control," *presented at Neural Networks, International Joint Conference on*, 1999.
[20] J. Liu and M. Brooke, "Fully parallel on-chip learning hardware neural network for real-time control," *presented at Circuits and Systems, Proceedings of the IEEE International Symposium on*, 1999.
[21] E. J. Brauer, J. J. Abbas, B. Callaway, J. Colvin, and J. Farris, "Hardware implementation of a neural network pattern shaper algorithm," *presented at Neural Networks, International Joint Conference on*, 1999.
[22] P. M. Engel and R. F. Molz, "A new proposal for implementation of competitive neural networks in analog hardware," *presented at Neural Networks, Proceedings. 5th Brazilian Symposium on*, 1998.
[23] J. Tang, M. R. Varley, and M. S. Peak, "Hardware implementations of multi-layer feedforward neural networks and error backpropagation using 8-bit PIC microcontrollers," *presented at Neural and Fuzzy Systems: Design, Hardware and Applications, IEE Colloquium on*, 1997.
[24] D. S. Reay, T. C. Green, and B. W. Williams, "Field programmable gate array implementation of a neural network accelerator," *presented at Hardware Implementation of Neural Networks and Fuzzy Logic, IEE Colloquium on*, 1994.
[25] A. Achyuthan and M. I. Elmasry, "Mixed analog/digital hardware synthesis of artificial neural networks," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, vol. 13, pp. 1073-1087, 1994.
[26] P. Dong, G. Bilbro, and M.-Y. Chow, “Controlling a Path-tracking Unmanned Ground Vehicle with a Field-Programmable Analog Array,” *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Monterey, CA, 24-28 July, 2005.
[27] P. I. Yakimov, E. D. Manolov, and M. H. Hristov, "Design and implementation of a V-f converter using FPAA," *presented at Electronics Technology: Meeting the Challenges of Electronics Technology Progress, 2004. 27th International Spring Seminar on*, 2004.
[28] M. Sekerli and R. J. Butera, "An implementation of a simple neuron model in field programmable analog arrays," *presented at Engineering in Medicine and Biology Society, 2004. EMBC 2004. Conference Proceedings. 26th Annual International Conference of the*, 2004.
[29] K. Basterretxea, J. M. Tarela, and I. del Campo, "Approximation of sigmoid function and the derivative for hardware implementation of artificial neurons," *Circuits, Devices and Systems, IEE Proceedings [see also IEE Proceedings G- Circuits, Devices and Systems]*, vol. 151, pp. 18-24, 2004.
[30] A. Atiya, E. Gad, S. Shaheen, and A. El-Dessouky, "On training piecewise linear networks," *presented at Neural Networks for Signal Processing IX, Proceedings of the IEEE Signal Processing Society Workshop*, 1999.
[31] A. Bermak and A. Bouzerdoum, "VLSI implementation of a neural network classifier based on the saturating linear activation function," *presented at Neural Information Processing, Proceedings of the 9th International Conference on*, 2002.
[32] S. Ridella, S. Rovetta, and R. Zunino, "Circular backpropagation networks for classification," *Neural Networks, IEEE Transactions on*, vol. 8, pp. 84-97, 1997.
[33] E. van Keulen, S. Colak, H. Withagen, and H. Hegt, "Neural network hardware performance criteria," *presented at Neural Networks, IEEE World Congress on Computational Intelligence*, 1994 IEEE International Conference on, 1994.