ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

Implementation of a Perceptron-based Artificial Neural Network Classifier Circuit on FPGA Hardware

[1] Amit R. Chavan, [2] Ashwini Kumar Arya
[1] [2]Project Engineer (ACTS), Centre for Development of Advanced Computing (C-DAC), Pune

***Abstract* - This paper elaborates the implementation of an unsupervised Artificial Neural Network (ANN) on FPGA hardware for data classification. ANN is the best option to classify a large amount of data into several desired classes as per the characteristics and parameters of the given data samples. Implementation of an unsupervised ANN on a chip eliminates the additional stage of software simulation of the ANN for the given dataset, i.e. training of ANN using a software and then implementation of trained ANN on FPGA chip. The Unsupervised ANN is implemented on Xilinx Virtex-4 FPGA, which consumes less on-chip resources, consuming less power at optimum speed.**

Keywords— Artificial Neural Network (ANN), Data Classification, Field-Programmable Gate Array (FPGA), Heaviside Step Function, Neural Network Implementation, On-chip Neural Network Training, Perception, Unsupervised Learning.

1. INTRODUCTION

Data classification is an essential task in many of the daily processes. From biomedical to agriculture, and from statistics to space-science, data classification is widely used to analyze several parameters so that they can be "grouped" as per the desired characteristics or criteria. Generally, data classification is done by manual (e.g. classification by assigning workers), approximation (e.g. by guessing from parameters like size), and several other processes. But the methods like manual and approximation are suitable if the data or objects to be classified are smaller in number. For a large amount of data, these methods prove time-consuming – means they are taking more time to processing as the data increases. So, for these scenarios, automated data classification techniques can be the best option.

A. Problem Statement

For automated rapid data classification, use of an Artificial Neural Network (ANN) is one of the efficient options. Artificial Neural Networks (ANNs) are computing systems which mimics the biological neural networks. ANN 'learns' i.e. progressively improves performance to do tasks by considering examples without task-specific programming. ANNs are preferred in such applications, where a normal traditional algorithm with rule-based programming proves difficult for implementation. Generally, ANNs can be trained to achieve the desired functionality by several learning methods, known as Supervised and Unsupervised Learning Techniques [1], [2].

In supervised learning, there are some input variables, an output variable, and the implementation of such an algorithm is required to learn the mapping function which can map inputs and output variable [2]. That is, if 'x' is any input data and 'y' is the output variable, then an ANN should learn the mapping function y = f(x). The goal of a supervised ANN is to approximate the mapping function y = f(x), such that when a new input 'x' is given to the ANN, it should predict the output variables 'y'.

In the supervised learning process, the correct output class or value is already known for an input, so the algorithm repetitively makes predictions on the training data and is corrected by the known output values, like a supervisor. The learning process of ANN stops when the algorithm reflects an acceptable level of performance [2].

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

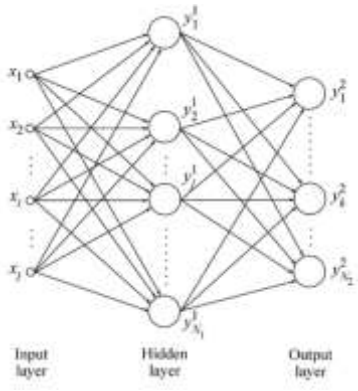

Fig. 1: General representation of an ANN [1].

In unsupervised learning, there is only input data 'x' and no corresponding output variables 'y'. The purpose of unsupervised learning is to train an ANN architecture by such a way that, it can learn more about the data without any external guidance. As opposite to supervised learning, there are neither correct answers nor any supervision. So, the algorithms fully depend on the ANN structure for data clustering or data association [2].

For real-time data classification using ANNs, generally there are two approaches usually followed: software-based and hardware-based. In software-based approaches, the data to be classified is in digital format. Hence, the implementation and training of an ANN is done with the aid of software tools. The advantage of ANN software implementation is the related code is flexible. As the application for ANN changes, code instructions can be modified as per convenience. The main drawback of ANN software implementation is the requirement of large overhead for running in parallel with the ANN software implementation by the operating system and other software applications [3]. If the real-time data is hardware input (e.g. switch or relay) or signal input (e.g. an EEG signal or revolution per second value), taking help of software tools should be less effective. This happens due to hardware data we have to send to a host where it is evaluated by a software tool, and after processing, we have to send the processed output back to the hardware where the user observes it actually. In such a case, ANNs are first trained on a software tool. This trained ANN is preferred to implement on FPGA Hardware. So, direct implementation of ANN hardware which can be trained without any external training software tool seems to be beneficial. Implementing an ANN direct as hardware will save conversion time and transmission time from one medium (hardware) to another medium (software). Also, as the computations are directly being performed by hardware, the operating speed of ANN will be much faster. Hardware implemented ANNs fully reflects the parallel operation of the neurons, hence achieving a very high speed of information processing as compared with computer-based sequentially simulated ANNs [4].

B. Organization of the Paper

This paper is organized as follows: section II describes the basic framework required for object counting techniques using image processing. In Section III, the methodology for implementation of an unsupervised ANN on a Virtex-4 chip is presented. The results are analyzed in section IV.

II. FRAMEWORK

Generally, any Artificial Neural Network i.e. ANN is made up of planned interconnections of its basic elements called as neurons. Neurons are behaviorally similar to the neurons as in the mammalian nervous system. The basic neuron can be represented as shown in Fig. 2.

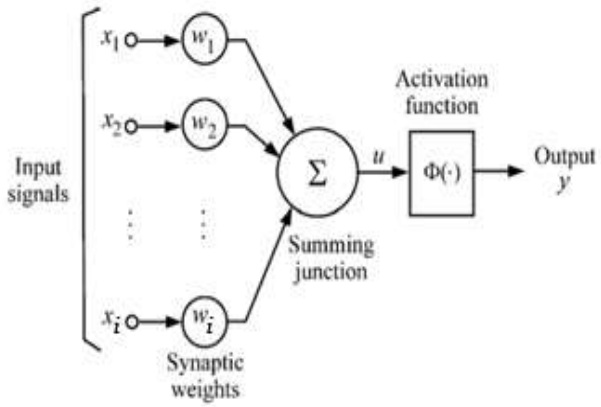

Fig. 2: Structure of a basic neuron in ANN [1].

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

A. Input Reception and processing

For any neural network, inputs are given to several building blocks (called as neurons) as per the ANN architecture. Inputs are multiplied by several parameters called as 'weight', before giving as an input to each neuron. Weights are assigned to the specific inputs. If a neuron has 'n' inputs, then it has 'n' weights that can be adjusted individually. During the training of ANN, values of the weights can be adjusted considering the error of the last test result [1].

B. Summing Junction

After processing by weights, the inputs are summed up to obtain a single value. In this step, an adjustable offset (called as bias) is also added to this sum. ANN can adjust the bias value during the ANN training. At the beginning, all the neurons have random values of weights and biases. Weights and biases are changed by a small value after every learning iteration so that the next result is a bit closer to the desired output. This way, the ANN approaches towards a state where it can be considered that it has 'learned' the desired patterns [1].

C. Activation Function

The result of the summing junction output is converted into an output signal. This is done by passing this result to an activation function. The most basic form of an activation function is a simple binary threshold function that has only two possible results: 'high' and 'low'. A similar activation function is described in Fig. 3.

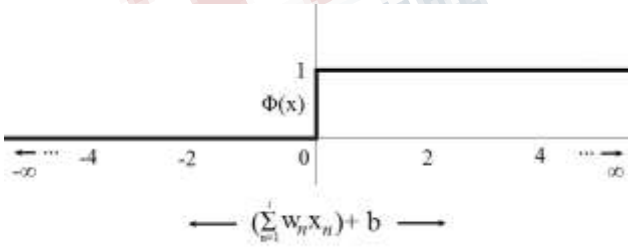

Fig. 3: The Heaviside Step function.

(If x_1, x_2, ... x_i are the inputs, w_1, w_2, ... w_i are their respective assigned weights, and b is the bias added, then the summing junction gives output, which can be represented in the equation form as

$$\sum_{n=1,2,3..i} w_n \cdot x_n + b \qquad (1)$$

If Φ(x) is the activation function of the neuron, then as per the requirement, we can define Φ(x) as

$$\phi(x) = \begin{cases} 1 & \text{if } \sum_{n=1,2,3..i} w_n \cdot x_n + b > 0 \\ 0 & \text{otherwise} \end{cases} \qquad (2)$$

These types of activation functions come under a category termed as "the Heaviside Step function" [5]. This function returns 1 if the input is positive or zero, and 0 if the input is negative. A neuron having the similar type of activation function is called as 'perceptron' [1]. In this work, these perceptrons are used with a slight modification.

By joining or cascading such a neurons in the desired way, a multilayer ANN can be constructed. Each layer can have a number of neurons, who are taking inputs from the previous layer and their outputs are cascaded to the next layer. A representation of a multilayer ANN is shown in Fig. 4.

Fig. 4: A generalized framework for a multilayer ANN.

Most of the recent works [6]-[9] have followed the same framework; however, the activation functions, HDL Language and target hardware are varying in each work.

III. METHODOLOGY

In this work, the proposed workflow is – to acquire three 1-bit input data, use the same to pass to a 3×2 Unsupervised ANN structure, performing convolution of the processed inputs, and decision making for classification of the given input combination by an activation function. The simulation of the ANN is done in Questa Sim 10.0b simulator software and its

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

implementation is done by using Xilinx ISE Design Suite 14.7 and Virtex-4 (XC4VLX25–10–FF668) FPGA Development Board Hardware.

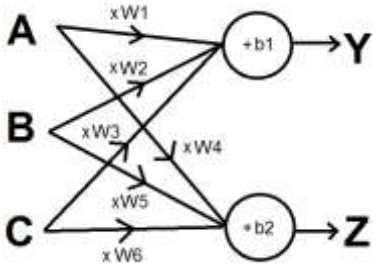

Fig. 5: Implementation scheme for the proposed unsupervised 3×2 ANN. (Note: Activation function block is not shown)

The ANN unit consists of two perceptrons, having three single bit inputs and one single-bit output. The arrangement of ANN using these perceptrons is shown in Fig. 5. Here, A, B, and C are the inputs to the ANN and Y and Z are the outputs with different activation functions. The design of the ANN is done using Verilog Hardware Descriptive Language (HDL).

A. Generation of Test Data

For the generation of a complete data set from three 1-bit inputs, generally, 23 = 8 combinations are required. It means, if A, B, and C are three inputs which are binary, then we require eight combinations of all values of these three variables – {A,B,C} = {(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)}. These data combinations are generated by using three clock signals with frequency f Hz, f/2 Hz, and f/4 Hz. The two signals with frequencies f/2 Hz and f/4 Hz are obtained from the first signal with frequency f Hz by implementing a frequency-divider cum binary counter circuit. The schematic representation is shown in Fig. 6. Here, frequency divider unit is made up of cascaded D flip-flops. The output of the frequency divider block is also given to the LEDs so that user can observe that which output is being provided to the input of the ANN block. This hardware design helps to compare the input being given to the ANN and output generated by the same, at the same time. Hence, tallying the input-output becomes easy and real-time. For Virtex-4 implementation, the value of 'f' is chosen as 1 Hz, so that step-by-step changes can be easily observed by human vision.

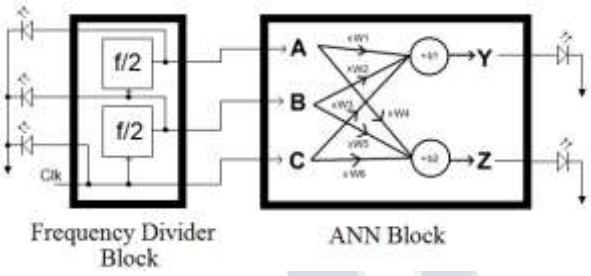

Fig. 6: Hardware scheme for generation and interfacing test data block (frequency divider block) and ANN block

B. Experimental Procedure

For implementing a perceptron, a scheme shown in fig. 2 is used. At first, a Verilog code for simple 3×1 ANN is simulated and synthesized. If the circuit of ANN is made asynchronous, it forms combinational loops within the design, which leads to failure of the hardware implementation of ANN. Combinational loops are logical structures that do not contain any synchronous feedback element like registers [10]. These loops affect the stability of output and will generate different results than expected. Since combinational loops make feedback with no any register in loops, such a design violates the synchronous principles [10]. Hence, synchronization of the design with a periodic reference signal becomes necessary. Usage of a clock signal for synchronization introduces flip-flops in the design. Also, synchronization eliminates almost all combinational loops in the digital circuit synthesized for the ANN. Apart from the clock; a synchronous reset is also introduced to the circuit for restarting the training process if needed.

At first, the three inputs A, B and C are given to the inputs of the two perceptrons. Let's call the perceptron generating output Y as P-Y, and the perceptron generating output Z as P-Z. The RTL Schematic of the implementation of the 3×2 ANN is shown in Fig. 7. One can compare Fig. 6 with Fig. 7 for understanding the hardware implementation on Virtex-4. As per the framework of a perceptron, the inputs are multiplied by some configurable weights. The bias value can also be

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

fetched to a perceptron for covering the missing classifications by linear equations of weight-input pairs [11]. Referring Fig. 5, the formula for summing junctions of P-Y and P-Z can be declared respectively as

$$SumY = (A \times W1) + (B \times W2) + (C \times W3) + b1 \quad (3)$$

$$SumZ = (A \times W4) + (B \times W5) + (C \times W6) + b2 \quad (4)$$

From the above formulas, it can be concluded that the convolution takes place for all the input values. The summation results can be cascaded to the activation function unit. Now, for activation function, the Biased Heaviside Step Function is preferred. It means that, if the output is above the pre-decided threshold, the output of the perceptron will be high. The threshold value is reconfigurable via Verilog coding. For current implementation, the biased threshold value is set to an unsigned positive 31 (9'b000011111). The activation function is same for both P-Y and P-Z, and the output is also given to the respective perceptron outputs Y and Z.

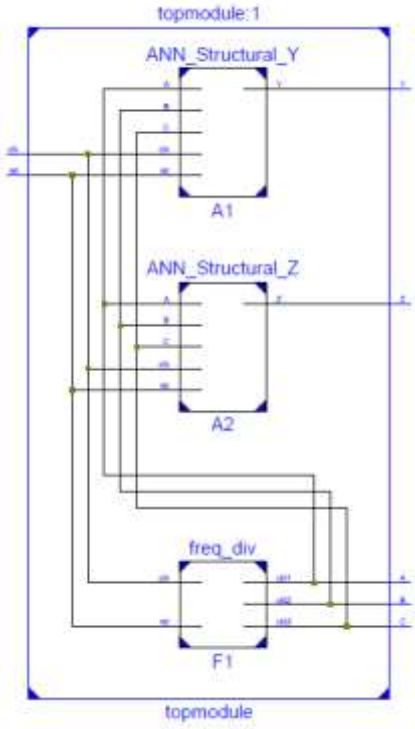

Fig. 7: RTL Schematic of 3×2 ANN.

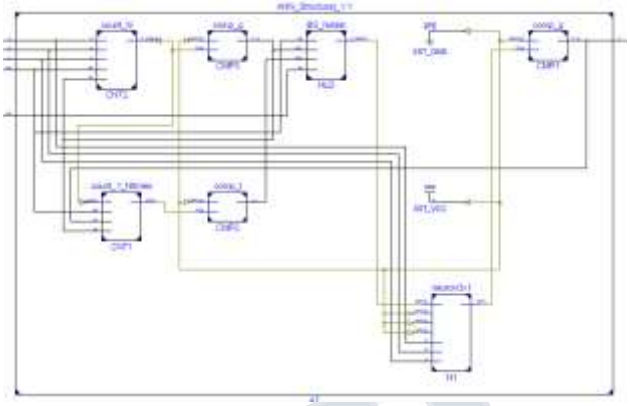

Fig. 8: RTL Schematic of a perceptron.

Now, for training the ANN, we have to reconfigure the weights and biases to get exact classification as per expectations. It is already known that inputs can have only eight possible combinations. Hence, it is considered that each perceptron must classify all the inputs in their two respective categories, according to the weights and biases applied. Each class can have half of the possible input combinations i.e. four samples. To obtain this, for each cycle of eight possible combinations, it is necessary to count a number of all 'highs' for each perceptron. If it is less than four, we have to adjust weights and biases by a specific value. The adjustment is continued for every iteration of all combinations (can be considered as learning iterations) till the circuit satisfied the said condition for classification. The resultant RTL schematic for the similar perceptron circuit is shown in Fig. 8. In the figure, the block N1 (neuron3x1) consists of only input processing unit (i.e. part of ANN where multiplication of all inputs by weights happens) and summing junction. The block CMP1 (comp_g) consists activation function for the perceptron. The output is monitored for modification in input processing parameters.

Almost all the blocks (excluding neuron3x1 and comparators) uses the clock and reset signals for synchronized operations. It increases the stability of circuit operations and formation of glitches or metastability in the outputs [12]. The same synthesized design is later implemented on Xilinx ML401 Virtex-4 FPGA Development Board hardware, which is shown in Fig. 9.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

Fig. 9: A Vitrex-4 ML401 FPGA Development Board.

IV. ANALYSIS OF RESULTS

The simulation result of the 3×2 ANN design is displayed in Fig. 10. From simulation results, it is clear that if all of the processing parameters (i.e. weights and biases) are reconfigurable, then ANN takes more time for training. (Note: Compare wave labels of Fig. 10 with diagram labels of Fig. 6 for easy understanding of simulation.)

TABLE I
Design Summary for 3×2 ANN with FDB

Logic Utilization	Used	Available	Utilization
Number of Slice Flip Flops	108	21,504	1%
Number of 4 input LUTs	79	21,504	1%
Number of occupied Slices	106	10,752	1%
Number of Slices containing only related logic	106	106	100%
Total Number of 4 input LUTs	80	21,504	1%
Number used as logic	79		
Number used as a route-thru	1		
Number of bonded IOBs	7	448	1%
Number of BUFG/BUFGCTRLs	1	32	3%
Average Fanout of Non-Clock Nets	3.45		

With a 100 MHz-to-1 Hz Frequency Divider Block (FDB), this design can run with the clock signal having minimum clock period 4.076 ns (i.e. maximum frequency 245.360 MHz) smoothly. The synthesis summary for the same is given in Table I.

For considering an application-based implementation of this 3×2 ANN on Virtex-4, the frequency divider block can be removed and the inputs can be taken from other interfaces. Without the 100 MHz-to-1Hz frequency divider block, this design can run with the clock signal having minimum clock period 3.462 ns (i.e. maximum frequency: 288.863 MHz).

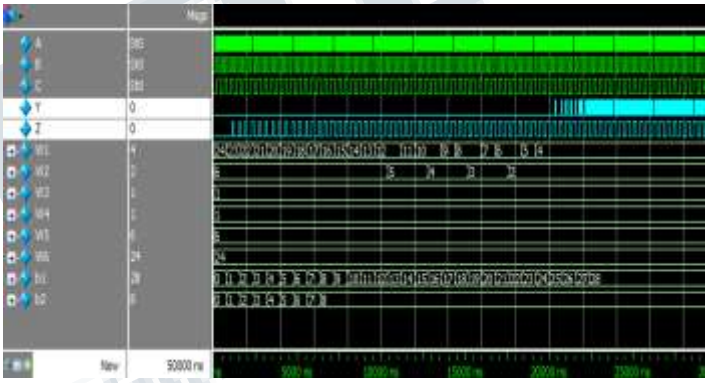

Fig. 10: Simulation Results for 3×2 ANN.

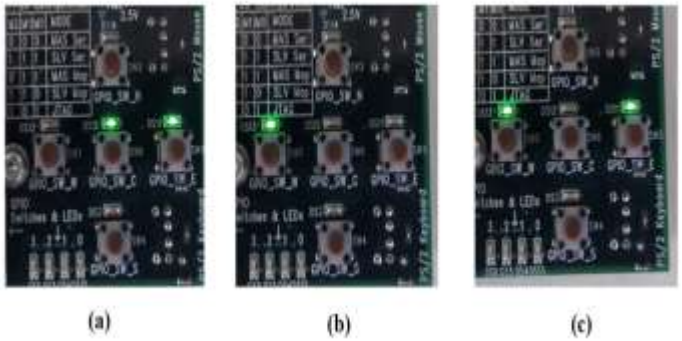

Fig. 11: Implementation results during the training of the 3×2 ANN.

(a) For input combination 011. (b) For input combination 100. (c) For input combination 101.

West, Center and East LEDs are reflecting inputs C, B and A respectively.

ISSN (Online) 2394-2320

International Journal of Engineering Research in Computer Science and Engineering (IJERCSE)
Vol 5, Issue 2, February 2018

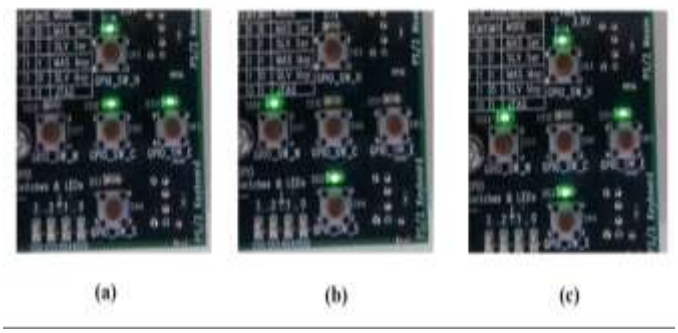

Fig. 12: Implementation results after training of the 3×2 ANN.

(a) For input combination 011. (b) For input combination 100. (c) For input combination 101.
North and South LEDs are displaying outputs Y and Z respectively.

This is the maximum possible speed of the proposed on-chip trainable unsupervised ANN for a Virtex-4 FPGA hardware. The synthesis summary for the same is given in Table II. Here, the number of bounded IOBs is increased from 7 to 10, due to the removal of frequency divider block (FDB) and taking actual inputs from hardware for classification.

TABLE II
Design Summary for 3×2 ANN without FDB

Logic Utilization	Used	Available	Utilization
Number of Slices	46	10752	0%
Number of Slice Flip Flops	52	21504	0%
Number of 4 input LUTs	81	21504	0%
Number of bonded IOBs	10	448	2%
Number of GCLKs	1	32	3%

The implementation results of the 3×2 ANN on Virtex-4 Development Board are shown in Fig. 11 and Fig. 12 respectively. The inputs A, B, and C can be observed on the East, Center and West LEDs respectively. The output Y and Z are displayed on the North and South LEDs. After training, the input-output combinations for the final values of input processing parameters are displayed in Table III. From results, we can conclude that – if we consider {A,B,C} as a three-bit input binary number, the ANN classifies the input by two ways. At output Y, if the number lies in the upper half, Y goes high. At output Z, if the number is odd, Z goes high. By designing activation function algorithm circuit in different ways, we can obtain different results.

TABLE III
Implementation results after training of 3×2 unsupervised ANN as per final values of weights and biases

A	B	C	Y	Z
0	0	0	0	0
0	0	1	0	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	1
1	1	0	1	0
1	1	1	1	1

CONCLUSION AND FUTURE SCOPE

This work introduces a prototype hardware implementation of a 3×2 perceptron-based unsupervised ANN on Virtex-4 FPGA chip for data classification purpose. These types of designs are much faster than processor-based implementations, since they utilize the comparatively less on-chip area. These designs can also handle real-time data well as compared with software simulated ANNs due to natural parallelism in the ANN hardware. If the Unsupervised ANN has more reconfigurable input processing parameters, the circuit of activation function algorithm becomes more complex. Hence, it is recommended to implement a trained ANN on the chip rather than training the same on the hardware. If area optimization is not important, then the similar reconfigurable ANN hardware can be used smoothly in application-based classification and regression circuits.
Such a neural network can be further developed for the regression-based hardware where prediction of future values can be done. Also, it can be used for automated unsupervised classification of any fixed sized data. Since

on-chip training is enabled, the ANNs can still train themselves after hardware implementation, leading to auto-reconfiguration of ANN parameters. This is equivalent to self-upgrading the hardware, which is the rare phenomenon in hardware. These types of self-learning hardware components can be further developed as a neuroprocessor.

ACKNOWLEDGEMENTS

Authors are grateful to Mrs. Mita Karajagi (Associate Director and Head of the Department), Mr. Aditya Kumar Sinha (Joint Director), Mrs. Risha P. (Joint Director), and all the staff of Adavanced Computer Training School (ACTS), C-DAC Innovation Park, Pune (India) for their support in the research activities.

REFERENCES

[1] Omondi, Rajapakshe, FPGA Implementations of Neural Networks, Springer.

[2] Brownlee, Master Machine Learning Algorithms.

[3] Ayman Youssef, Mohammed, Nasar, "Two Novel Generic, Reconfigurable Neural Network FPGA Architectures", IEEE 4th International Conference on Artificial Intelligence with Applications in Engineering and Technology, 2014.

[4] Dinu, Cirstea, "A Digital Neural Network FPGA Direct Hardware Implementation Algorithm", IEEE International Symposium on Industrial Electronics (ISIE), 2007.

[5] Kanwal, Generalized Functions: Theory and Technique, 2nd ed. Boston, MA: Birkhäuser, 1998.

[6] Chaitra, "Hardware Implementation of Artificial Neural Networks using Back Propagation Algorithm on FPGA", International Journal of Research in Engineering and Technology, Vol. 05 Sp. Issue 04, pp. 211-214, May 2016.

[7] Elmisery, Khalil, Salama, Algeldawy, "Adaptation of ANN For FPGA Implementation and its Application for Speaker Identification", IEEE International Conference on Electrical, Electronic and Computer Engineering, pp 317-320, 2004.

[8] Shah, Vishwakarma, "FPGA implementation of ANN for reactive routing protocols in MANET", IEEE International Conference on Communication, Networks and Satellite, pp. 11-14, July 2012.

[9] Langer, Bhat, Agarwal, "Neural-network-based space-vector pulse-width modulation for capacitor voltage balancing of threephase three-level improved power quality converter", IET Power Electronics, vol. 7, issue 4, pp. 973-983, April 2014.

[10] Fayyazi, Kirsch, "Efficient Simulation of Oscillatory Combinational Loops", 47th ACM/IEEE Design Automation Conference (DAC), June 2010.

[11] Smola, Vishwanathan, Introduction to Machine Learning, Cambridge University Press.

[12] Wakerly, Digital Design: Principle and Practices, 4th Edition, Pearson.