FPGA Implementation of a Multilayer Artificial Neural Network using System-on-Chip Design Methodology

Ravikant G. Biradar[1,2], Abhishek Chatterjee[1], Prabhakar Mishra[1,2], Koshy George[1,2]
[1]PES Centre for Intelligent Systems, PESIT;
[2]Dept. of Telecommunication Engineering, PES Institute of Technology, Bangalore 560 085, India
{ravikant.biradar, prabhakar.mishra, kgeorge}@pes.edu, abhi0363@gmail.com

***Abstract*—Artificial Neural Networks (ANNs) find applications in engineering solutions to complex problems. The usefulness of ANNs in real-time embedded applications can be enhanced if feasible architectures for their hardware implementation can be customized to provide an attractive trade-off between area and performance. In this paper, we present a real-time embedded hardware implementation of a feed-forward neural network which employs backpropagation algorithm for training. Custom modules are designed for the activation functions, the neurons and a finite state machine that co-ordinate activities for training. The proposed ANN is implemented on a Virtex-5 Field Programmable Gate Array. Use of System-on-Chip design methodology enables design reuse while improving the performance metrics.**

Keywords—Artificial Neural Network; System-on-Chip; Field Programmable Gate Arrays

I. INTRODUCTION

Artificial Neural Networks (ANNs) provide significantly attractive features like inherent parallelism, online and offline learning and approximation of nonlinear function for solutions to complex problems [1]. These capabilities are leveraged in many complex applications that include pattern classification and recognition (e.g., [2]-[4]), and function approximation (e.g. [5]). Research in theoretical aspects of an ANN has not been sufficiently supplemented by attempts for high performance real-time embedded implementation owing to high computational complexity and dependence on floating point arithmetic. This gap has resulted in considerable limitations for use of ANN-based techniques in resource-constrained embedded applications.

Several researchers have reported implementation of ANNs in both software and hardware. Quick constructability, adaptivity and testability for a wide range of applications are some of the attractive features of the software implementation of ANNs [3]. Such implementations can be found in [6,7]. However, the latter papers do not consider hardware implementation.

ANNs for complex applications typically require an ensemble of training epochs and multiple passes in the training process. Software implementation of ANNs for such applications is ineffective due to large latency of training on the host processor. This process can be accelerated by designing custom hardware leveraging parallelism, pipelining and flexible bit width data path. Hardware implementation can accelerate the training process. The Application Specific Integrated Circuit (ASIC) approach to hardware implementation of an ANN provides high speed in real-time applications but is prohibitively costly and dedicated to a specific ANN model. The Field Programmable Gate Array (FPGA) approach on the other hand offers several advantages such as flexibility in programmable systems, parallelism which allows faster operation, high logic density, ability to reconfigure while resident on the system, inexpensive logic design due to their shorter design cycle and hardware reuse [8]. Accordingly, an efficient balance between flexibility and performance using processor and dedicated hardware on FPGA can be architected to support embedded implementation of ANNs [4]. The focus in this paper is on an area-optimized implementation of a specific ANN with real-time learning on an FPGA using System-on-chip (SoC) design methodology. To the best knowledge of the authors such an implementation is completely novel.

The main contributions of the paper are as follows: (i) A custom hardware module for efficient approximation of hyperbolic tangent activation function. (ii) Hardware abstraction of neuron along with necessary data path and communication ports as a parametrizable intellectual property (IP). (iii) Complete custom hardware implementation of backpropagation algorithm for training a candidate four-layer feed-forward neural network (FFNN).

This paper is organized as follows: Section II reviews the available literature in the relevant field. The architectural detail of the candidate neural network is presented in Section III. Embedded implementations of the hyperbolic tangent activation function, the artificial neuron and the back propagation algorithm (BPA) are discussed in Section IV. Results of implementation on Virtex-5 FPGA are detailed in the end of this section.

II. PREVIOUS WORK

An early attempt of FPGA implementation of an FFNN is reported in [9]. Owing to the use of general purpose data path elements, the area trade-off in this implementation is not

978-1-4799-7171-8/15/$31.00 ©2015 IEEE

suitable for networks that have multiple hidden layers and larger number of neurons per layer. The authors in [10] present a hardware implementation of multilayer FFNN using existing system generator library functions that are mapped on to the FPGA during synthesis. In this implementation, code from Simulink blocks has been mapped on to corresponding FPGA resources without structural optimization. Pipeline based architecture for implementation has been proposed in [11]. The speed of operation as reported in the paper is limited to 10MHz while up to 72% of available resources on Virtex XCV400 device were consumed in implementing a 2-5-2-2 network.

Many implementations [12] reported in the literature either seek to implement and verify the network in Matlab or Labview, and use a cross-compiler to map the code on to an FPGA resulting in sub-optimal hardware implementation. Some implementations as reported in [13] and [14] seek to leverage pulsed neural networks for achieving desired area trade-off for on-chip implementation. However, the limitations of such approaches have been outlined in [13]. Use of multiprocessor-on-chip design for implementation of neural networks has been reported in [15]. In such approaches use of existing soft core processor IPs do not provide the required trade-off for scalability both in terms of number of neurons per layer and number of layers.

DSP processor based implementations have been reported in [16] and [17] for a variety of motor control applications. DSP processors available in the market are typically limited to 1, 2, 4 or 8 MAC cores. This limits parallelizability of neuron activations and affects the time performance of the network. Such solutions are better suited for scenarios where offline training for the application is under consideration.

To address the limitations of the implementations proposed in the literature, we propose to implement area-efficient modules for activation functions, and neurons built using these activation functions. These custom modules can be added to the IP catalogue of the FPGA tool provider and synthesis can be constrained to use these blocks where necessary. Additionally, a scalable custom module has been designed for training the network based on the BPA.

III. CANDIDATE FFNN FOR IMPLEMENTATION

The network chosen here provides sufficient complexity in terms of embedded implementation in a fixed hardware. Moreover, this network can be used for other applications such as classification with minimum modifications in the network variant. The chosen structure of the FFNN (shown in Fig. 1) consists of two hidden layers with four and five hidden neurons, three input nodes and two output neurons. The notation "w_{lij}" represents a synaptic weight connecting the i^{th} neuron of the l^{th} layer to the j^{th} neuron of the preceding layer. The nonlinear activation function of the neurons of the hidden layers is the hyperbolic tangent activation function:

$$\tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \qquad (1)$$

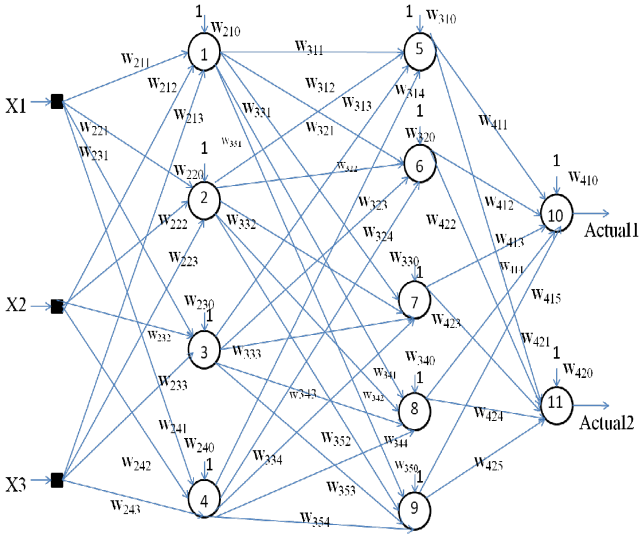

Fig. 1: A 3-4-5-2 Feed forward Neural Network

The activation functions of the neurons in the output layer are linear.

The network shown in Fig. 1 is trained using BPA. We concentrate here on this algorithm as it is the simplest to implement and can readily be used for online applications. It consists of a forward pass which computes the output of the FFNN and a backward pass to determine the updates for the synaptic weights during learning. For a neuron present in the output layer, the weight correction $\Delta w_{ji}(n)$ applied to $w_{ji}(n)$ is in accordance to (2).

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

where $\delta_j(n)$ is the local gradient defined as

$$\delta_j(n) = e_j(n) \varphi_j'(v_j(n)) \qquad (3)$$

η is the learning rate, and $y_i(n)$ is the input to j^{th} neuron from the i^{th} neuron in the previous layer. For a neuron present in the hidden layer, the weight correction is governed by (2) in which the local gradient $\delta_j(n)$ is defined by (4) as given below.

$$\delta_j(n) = \varphi_j'(v_j(n)) \sum_k \delta_k(n) w_{kj}(n) \qquad (4)$$

where, $\varphi_j'(v_j(n))$ represents the derivative of the activation function.

IV. EMBEDDED IMPLEMENTATION

The embedded implementation of the neural network comprises of the following components.

A. Hyperbolic Tangent Activation Function

Several approaches to approximate the hyperbolic tangent activation function have been reported in literature [18]-[21]. An efficient approach to approximate the hyperbolic tangent activation function was proposed in [2]. The architecture proposed in this paper is based on piecewise linear

TABLE I. Different input range and sub-ranges

Input Range		Input Sub-Range		Output value
x_{i1}	x_{i2}	x_{s1}	x_{s2}	
4	8	-	-	0.96875
2	4	-	-	0.96875
1	2	1.875	2.0	0.96875
		1.75	1.875	0.96875
		1.625	1.75	0.9375
		1.5	1.625	0.90625
		1.375	1.5	0.90625
		1.25	1.375	0.875
		1.125	1.25	0.84375
		1.0	1.125	0.78125
0.5	1	0.9375	1.0	0.75
		0.875	0.9375	0.71875
		0.8125	0.875	0.6875
		0.75	0.8125	0.65625
		0.6875	0.75	0.625
		0.625	0.6875	0.5625
		0.5625	0.625	0.53125
		0.5	0.5625	0.5
0	0.5	-	-	Input

approximation in combination with bit-level mapping [2]. The maximum allowable error is used as the design parameter in this architecture. The advantage of such an approximation is that it can be implemented using purely combinational logic. In our work, we have used this approximation with maximum allowable error of 0.04. The architecture for the implementation of the approximation is shown in Fig. 2.

Table I gives the approximation of the hyperbolic tangent activation function as reported in [2]. In this approximation, the first quadrant of the hyperbolic tangent is divided into three regions, namely, the pass region, the processing region and the saturation region. In the pass region, the input is directly passed to the output. The approximation in the processing region is based on a parameter N_{ONE}. This parameter denotes the occurance of first '1' when the input binary sequence is scanned from left. We have modified the bit-level mapping in the range 0.5-2.0 using the correct value of N_{ONE} as compared to that reported in [2]. In the saturation region, the output is equal to highest possible value as represented digitally. In the architecture shown in Fig. 2, the input range decoder selects different region as discussed above based on the input binary stream. The bit level mapping blocks map the input to the corresponding output based on input bits.

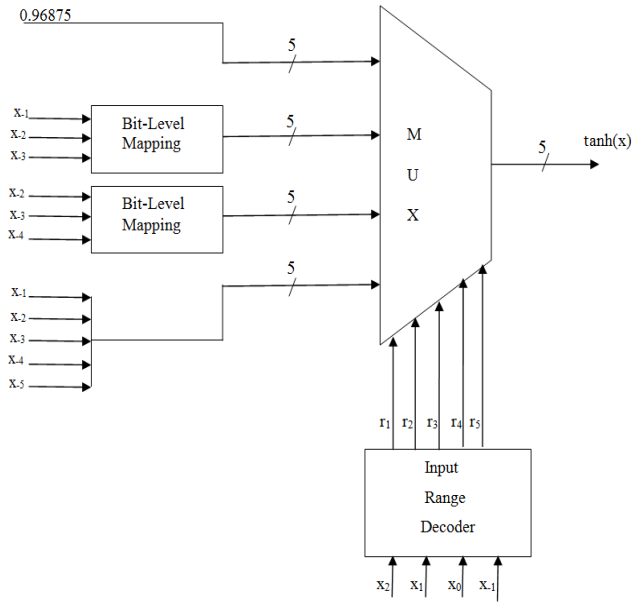

Fig. 2: Hardware Implementation of the Hyperbolic Tangent Actiation Function Approximation for ε = 0.04.

The architecture in this paper has been designed targeting FPGA implementation as against ASIC implementation reported in [2]. The implementation results are summarized in Table II.

B. Architecure of the Neuron

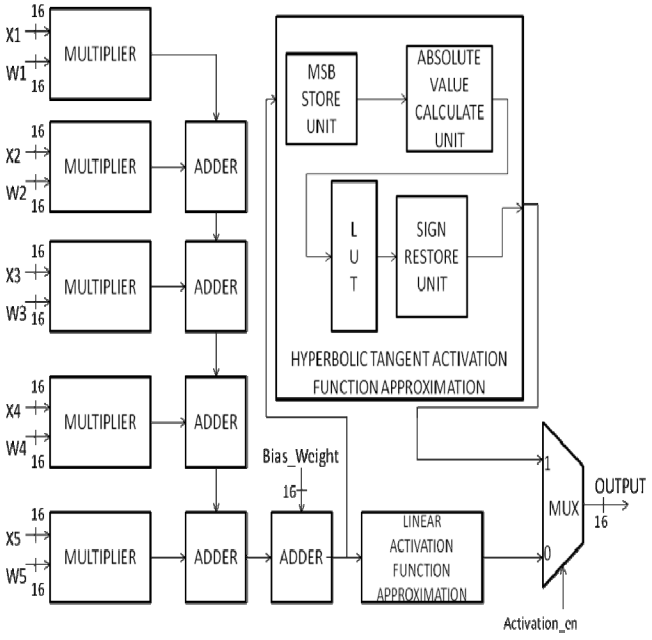

Fig. 3 : Hardware Implementation of the Artificial Neuron

The proposed architecture of a neuron is shown in Fig. 3. Each neuron suports five inputs along with the corresponding weights (x1, w1), ….., (x5, w5). A custom multiplier based on

look-up tables has been designed for each input. A group of four adders has been provided in the neuron module for accumulating weighted sum of the inputs. Another adder is provided for accomodating the bias. The net accumulated input is fed to the activation function block. In our network selection of the activation function between hyperbolic tangent and linear is provided through a multiplexer. In the hyperbolic tangent approximation block, only the values corresponding to first quadrant are stored. For the other quadrants, the property of odd fucntion is exploited using the 'absolute value calculate unit' and 'sign restore unit' . Implementation results for this module are summarized in Table III.

C. The Backpropagation Algorithm

The error in the output of the neurons in the previous layers of the network is estimated based on the calculation of the network error at the output layer. The backpropagation algorithm relies on this approximation for training the network [1]. Accordingly, the network error is estimated and propagated backwards layer-by-layer until the first hidden layer is reached. The error estimation of any given layer except the output layer depends on the error calculation of its successor [3]. Equations (1)-(4) form the basis for the proposed architecture for the training module. As shown in Fig. 4, all the weight changes of the corresponding neurons are computed sequentially. Once the outputs of the neurons present in the output layer are generated by the forward pass calculations, the errors at the neurons present in the output layer are obtained by means of multiplexers K, L, B, C and E. This error is the difference between the desired value and the actual value at the output layer neurons. Once this error is generated, all the weight changes related to the output layer neurons are generated. Since, the output layer neurons contain linear activation functions whose derivative is +1, according to (3), the local gradients of the neurons present in the output layer are calculated which are equivalent to the errors at the respective neurons. These local gradients are used in the weight change calculations in the neurons present in the hidden layer 2. Multiplexers A, B, C, K, L and E allow proper selection of the signals for the weight change calculations in the output layer. Further, multiplexers A, B, C, D, G, H, I, J, E and F provide proper signals during the weight change calculations pertaining to the neurons present in the hidden layer 2. In the process of weight change calculations of the neurons present in hidden layer 2, local gradients of the neurons are calculated and stored in the register bank as shown in Fig. 4. These local gradients are required for the calculation of weight changes related to neurons present in the hidden layer 1. Multiplexers A, B, C, D, E, F, M, N and O are used to select proper signals required during the weight change calculations related to neurons present in the hidden layer 1. The multipliers and adders perform the necessary computations required during the weight change calcuations of the neurons present in the feedforward network shown in Fig. 1. All the weight changes are calculated in accordance to (2) and are stored in the register bank shown in Fig. 4.

Emphasis has been given to efficient utilization of resources for designing this architecture. As the penalty on area or resource utilization adversely affects scalability as the network size grows, the multipliers and the adders used in this architecture are shared for several calculations efficiently.

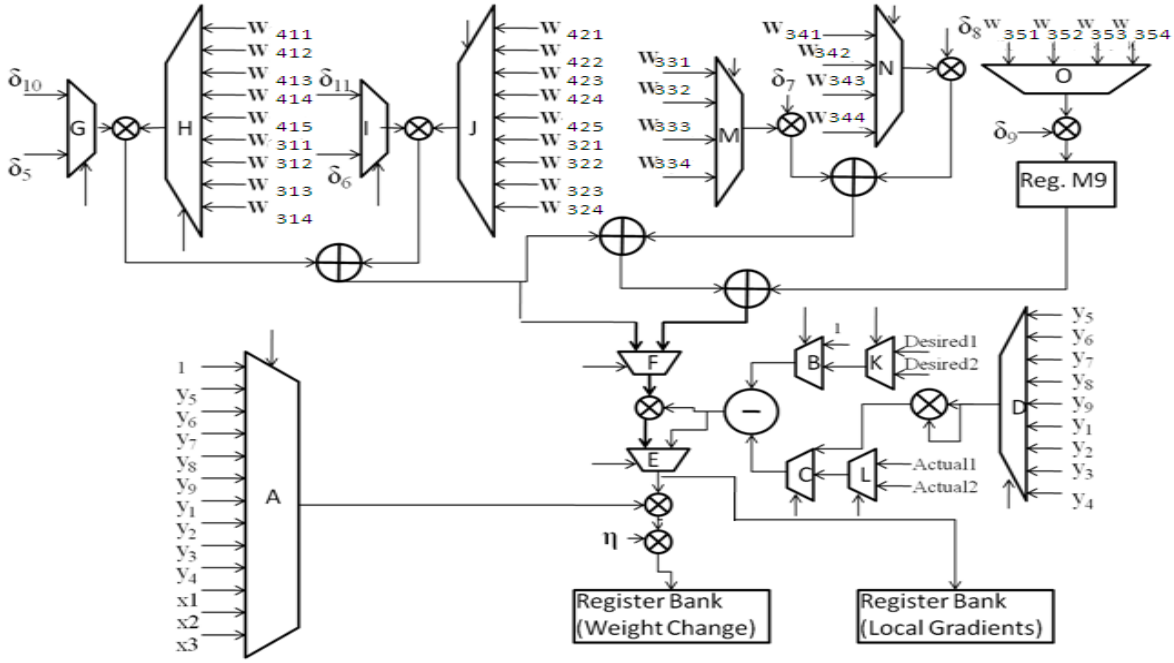

Fig. 4 : Hardware Implementation of the Backward Pass Module.

This architecture propagates the output activations backwards through the neural network using the training pattern and the change in weights at each neuron is generated. A finite state machine is used to sequence the operations on the data path. Local gradients and change in weights are stored in individual register banks. Appropriate inputs and weights are selected to estimate the error at each layer. Each neuron works with two levels of parallelism in the forward pass: synapse level and layer level. This provides faster response in the forward pass at the expense of area. Resource utilization results for the backward pass and back propagation stages are summarized in Table IV and V respectively.

D. Division Module

The back propagation algorithm requires the computation of the Mean Square Error (MSE) which needs the division operation. An efficient hardware implementation of a divider module along with its finite state machine and data path stages is shown in Fig. 5. The implementation results for the division module are given in Table VI.

To summarize, Table II depicts that the chosen method to approximate the hyperbolic tangent activation function leads to an efficient implementation on Virtex 5 XUPV5-LX110T development board. Further, the results given in Tables III, IV, V and VI indicate that the proposed modules respectively for the artificial neuron, the backward pass, the BPA and the division algorithm utilizes area efficiently when implemented on Virtex 5 XUPV5-LX110T development board.

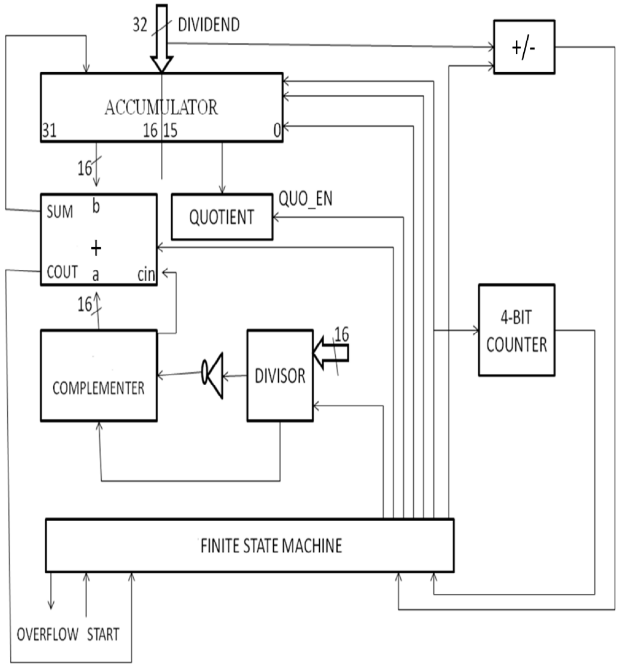

Fig. 5: Hardware implementation of division module

TABLE II. Hardware Implementation Results for Hyperbolic Tangent Activation Function Approximation with ε = 0.04

Logic Utilization	Available	Used	Percentage Utilization
Slice LUTs	69120	1976	2 %
Slice Registers	69120	1761	2 %
Bonded IOBs	640	4	1 %
Block RAM/FIFO	148	2	1 %
DSP48Es	64	3	4 %
Memory(in KB)	5328	72	1 %

TABLE III. Hardware Implementation Results for Artificial Neuron

Logic Utilization	Available	Used	Percentage Utilization
Slice LUTs	69120	2875	4 %
Slice Registers	69120	2014	2 %
Bonded IOBs	640	4	1 %
Block RAM/FIFO	148	2	1 %
DSP48Es	64	3	4 %
Memory (in KB)	5328	72	1 %

TABLE IV. Hardware Implementation Results for Backward Pass Module.

Logic Utilization	Available	Used	Percentage Utilization
Slice LUTs	69120	4133	5 %
Slice Registers	69120	3816	5 %
Bonded IOBs	640	4	1 %
Block RAM/FIFO	148	4	2 %
DSP48Es	64	12	18 %
Memory (in KB)	5328	144	2 %

V. CONCLUSIONS

In this paper we presented the hardware implementation of a multilayer feed forward neural network based on creation of custom IPs using the system-on-chip design methodology. Architectural optimizations for both data path and finite state machine were undertaken targeting FPGA implementation. The neural network was validated on a Virtex 5 XUPV5-LX110T development board. A few applications involving functional approximation restricting the bit widths to 16 bits were performed on the network while resident on the FPGA to validate the functionality.

As seen in section IV the area trade-off for the components of the network are encouraging from the

perspective of scalability. In the present implementation, the synaptic connection were made using PLB interconnects. This led to larger area required by the interconnections.

Work is in progress on the design of parametric scalable network-on-chip as an IP module, design of reconfigurable neuron blocks to alter bit widths depending upon application and a parametric customizable module for training; these results will appear elsewhere.

TABLE V. Hardware Implementation Results for Backpropagation Algorithm.

Logic Utilization	Available	Used	Percentage Utilization
Slice LUTs	69120	12682	18 %
Slice Registers	69120	4845	7 %
Bonded IOBs	640	4	1 %
Block RAM/FIFO	148	2	1 %
DSP48Es	64	12	18 %
Memory (in KB)	5328	72	1 %

TABLE VI. Hardware Implementation Results for the Division Module.

Logic Utilization	Available	Used	Percentage Utilization
Slice LUTs	69120	2181	3 %
Slice Registers	69120	1907	2 %
Bonded IOBs	640	4	1 %
Block RAM/FIFO	148	2	1 %
DSP48Es	64	3	4 %
Memory (in KB)	5328	72	1%

References

[1] S. Haykin, *Neural Networks. A Comprehensive Foundation*, 2nd ed. New Jersey, USA: Prentice Hall, 1999.

[2] B. Zamanlooy and M. Mirhassani, "Efficient VLSI implementation of neural networks with hyperbolic tangent activation function," *IEEE Trans. VLSI Syst*., vol. 22, no. 1, pp. 39–48, January 2014.

[3] A. Gomperts, A. Ukil and F. Zurfluh, "Development and implementation of parameterized FPGA-based general purpose neural networks for online applications," *IEEE Trans. Ind. Inf.*, vol. 7, no. 1, pp. 78–89, Feb. 2011.

[4] X. Li and S. Areibi, "Hardware/Software Co-design Approach for Face Recognition," in *Proc. 16th Int. Conf. on Microelectronics*, Tunis, Tunisia, December 2004, pp. 67-70.

[5] M. Bahoura and P. Chan-Wang, "FPGA-implementation of high-speed MLP neural network," in *Proc. IEEE 18th Int. Conf. Electron. Circuits Syst*., 2011, pp. 426–429.

[6] K. Subramanian, S. Suresh, and N. Sundararajan, "A metacognitive neuro– fuzzy inference system (McFIS) for sequential classification problems," *IEEE Trans. Fuzzy Syst*., vol. 21, no. 6, pp. 1080–1095, Dec. 2013.

[7] G. S. Babu and S. Suresh, "Sequential projection-based metacognitive learning in a radial basis function network for classification problems," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 24, no. 2, pp. 194 –206, 2013.

[8] S. Sahin, Y. Becerikli, and S. Yazici, "Neural network implementation in hardware using FPGAs," in *Proc. Int. Conf. on Neural Information Processing*, part III, 2006, pp. 1105-1112.

[9] G. Alizadeh, J. Frounchi, M. B. Nia, M. H. Zariff, and S. Asgarifar, "An FPGA implementation of an artificial neural network for prediction of cetane number," in *Proc. Int. Conf. Comp. Comm. Eng*., 2008, pp. 605–608.

[10] A. Tisan, M. Cirstea, S. Oniga, and A. Buchman, "Artificial olfaction system with hardware on-chip learning neural networks," in *Proc. IEEE Int. Conf. Optimization of Electrical and Electronic Equipment (OPTIM 2010)*, Brasov, Romania, May 20–22, 2010, pp. 884–889.

[11] R. Gadea, J. Cerda, F. Ballester, and A. Mocholi, "Artificial neural network implementation on a single FPGA of a pipelined on-line backpropagation," in *Proc. 13th Int. Symp. Syst. Synthesis*, 2000, pp. 225–230.

[12] T. Orlowska-Kowalska and M. Kaminski, "FPGA implementation of the multilayer neural network for the speed estimation of the two-mass drive system," *IEEE Trans. Ind. Electron*., vol. 7, no. 3, pp. 436–445, Aug. 2011.

[13] H. Hikawa, "Frequency-based multilayer neural network with on-chip learning and enhanced neuron characteristics," *IEEE Trans. Neural Netw.*, vol. 10, no. 3, pp. 545–553, May 1999.

[14] T. Lehmann, "Classical conditioning with pulsed integrated neural networks: Circuits and systems," *IEEE Trans. Circuits Syst. II*, vol. 45, pp. 720–728, June 1998.

[15] R.J Aliaga, R Gadea, R.J. Colom, J.M. Monzo, C.W. Lerche, J.D. Martinez, A. Sebastia, F. Mateo, "Multiprocessor SoC implementation of neural network training on FPGA," in *Proc*. *IEEE Int. Conf. on Advances in Electronics and Micro-electronics*, pp.14, Sept. 29 2008-Oct. 4 2008.

[16] B. Singh, S. R. Arya, S. K. Dubey, A. Chandra and K. Al-Haddad, "Implementation of DSTATCOM using Neural Network Based Radial Basis Function," in *Proc. IEEE Industry Applications Society Annual Meeting,* 2013, pp. 1-8.

[17] S. K. Mondal, J. O. P. Pinto, and B. K. Bose, "A neural-network based space-vector PWM controller for a three-level voltage-fed inverter induction motor drive," *IEEE Trans. Ind. Appl*., vol. 38, no. 3, pp. 660–669, May. 2002.

[18] A. Mishra, Zaheeruddin, K. Raj, "Implementation of a digital neuron with non-linear activation function using piecewise linear approximation technique," in *Proc. Int. Conf. Microelectronics*, Dec. 2007, pp. 69-72.

[19] C.-W. Lin and J.-S. Wang, "A digital circuit design of hyperbolic tangent activation function for neural networks," in *Proc. IEEE Int. Symp. Circuits Syst. Conf*., May 2008, pp. 856-859.

[20] A. H. Namin, K. Leboeuf, R. Muscedere, H. Wu and M. Ahmadi, "Efficient hardware implementation of the hyperbolic tangent sigmoid function," in *Proc. IEEE Int. Symp. Circuits Syst. Conf*., May 2009, pp. 2117-2120.

[21] M. A. Sartin and A. C.R. da Silva, "Approximation of hyperbolic tangent activation function using hybrid methods," in *Proc. 8th IEEE International workshop on Reconfigurable and Communication-Centric Systems-on-Chip (ReCoSoC)*, July 2013, pp. 1-6.