Feedforward Neural Network Implementation in FPGA Using Layer Multiplexing for Effective Resource Utilization

S. Himavathi, D. Anitha, and A. Muthuramalingam

***Abstract*—This paper presents a hardware implementation of multilayer feedforward neural networks (NN) using reconfigurable field-programmable gate arrays (FPGAs). Despite improvements in FPGA densities, the numerous multipliers in an NN limit the size of the network that can be implemented using a single FPGA, thus making NN applications not viable commercially. The proposed implementation is aimed at reducing resource requirement, without much compromise on the speed, so that a larger NN can be realized on a single chip at a lower cost. The sequential processing of the layers in an NN has been exploited in this paper to implement large NNs using a method of layer multiplexing. Instead of realizing a complete network, only the single largest layer is implemented. The same layer behaves as different layers with the help of a control block. The control block ensures proper functioning by assigning the appropriate inputs, weights, biases, and excitation function of the layer that is currently being computed. Multilayer networks have been implemented using Xilinx FPGA "XCV400hq240." The concept used is shown to be very effective in reducing resource requirements at the cost of a moderate overhead on speed. This implementation is proposed to make NN applications viable in terms of cost and speed for online applications. An NN-based flux estimator is implemented in FPGA and the results obtained are presented.**

***Index Terms*—Field-programmable gate array (FPGA), hardware implementation, layer multiplexing, neural networks (NNs).**

I. INTRODUCTION

THE pursuit to build intelligent human-like machines led to the birth of artificial neural networks (ANNs). Much work based on computer simulations has proved the capability of ANNs to map, model, and classify nonlinear systems. The special features of ANNs such as capability to learn from examples, adaptations, parallelism, robustness to noise, and fault tolerance have opened their application to various fields of engineering, science, economics, etc. [1]–[4]. Real-time applications are feasible only if low-cost high-speed neural computation is made viable.

Implementation of neural networks (NNs) can be accomplished using either analog or digital hardware [5]. The digital implementation is more popular as it has the advantage of higher accuracy, better repeatability, lower noise sensitivity, better testability, and higher flexibility and compatibility with other types of preprocessors. On the other hand, analog systems are more difficult to be designed and can only be feasible for large scale productions, or for very specific applications [5]. The digital NN hardware implementations are further classified as follows: 1) field-programmable gate array (FPGA)-based implementations, 2) digital signal processor (DSP)-based implementations, and 3) application specific integrated chip (ASIC)-based implementations [5], [6]. DSP-based implementation is sequential and hence does not preserve the parallel architecture of the neurons in a layer. ASIC implementations do not offer reconfigurability by the user. FPGA is the most suitable hardware for NN implementation as it preserves the parallel architecture of the neurons and can be reconfigured by the user.

However, implementation of NN in FPGA has some challenges. Generally, NN hardware implementation requires large resource because of nonlinear excitation functions and several synapses (multipliers) present in the network. Numerous works have reported new multiplication algorithms for NN [7]–[11]. To avoid the use of multipliers to reduce resource, the weights of the NN have been constrained to integer powers of two [8]. Constraint on weights leads to inferior performance of the network. Distributed arithmetic (DA) has been widely used to improve hardware resource efficiency for multiplication [9]. Lookup tables (LUTs) are used to store the excitation function in order to improve speed and reduce the resource requirement [12]. Dynamic adaptive memories are proposed to reduce memory requirement [13]. In addition, implementation of dedicated NNs with few numbers of neurons has been reported [14]. An NN with six neurons has been realized using two XC3042–50-Mhz FPGA and 1-K $\times$ 8 erasable programmable read-only memory (EPROM) [15]. An NN with eight neurons for control of inverted pendulum has been implemented using Xilinx FPGA interfaced to an 8031 microcontroller and a 16-K random access memory (RAM) [16]. A real-time controller with proportional integral derivative (PID) control algorithms implemented in FPGA and NN controller implemented in DSP board have been reported [17].

The size and complexity of an NN depends on both the total number of neurons and the number of layers. It has been reported in literature that three-layer network with sigmoid

Manuscript received April 21, 2005; revised March 6, 2006 and November 1, 2006; accepted November 15, 2006. The work was supported by the All India Council for Technical Education(AICTE), a statutory body of the Government of India under the Grant 8020/RID/TAPTEC-32/2001-02.

S. Himavathi and A. Muthuramalingam are with the Electrical and Electronics Engineering Department, Pondicherry Engineering College, Pondicherry 605014, India (e-mail: hima_pony@yahoo.co.in; amrlingam@hotmail.com).

D. Anitha was with the Electrical and Electronics Engineering Department, Pondicherry Engineering College, Pondicherry 605014, India. She is now with the SRM College of Engineering, Deemed University, Chennai 600 044, India.

Digital Object Identifier 10.1109/TNN.2007.891626

transfer function in the hidden layer and linear transfer function in the output layer can virtually approximate any nonlinear function to any degree of accuracy provided sufficient number of neurons in a hidden unit is available [18].

To realize all types of nonlinearity using three layers, large number of neurons is needed in the hidden layer resulting in a massive NN. For function approximation, multilayer networks have been found to be very useful as it is similar to a biological NN. The resurgence of research in the area of NNs took place after the advent of multilayer networks, which could overcome the drawbacks of single-layer perceptron. More layers have proved their capability in function approximation as it can handle complex decision surfaces and multiple interactions between parameters. Implementations of multilayer networks will demand huge resource and will not be a feasible solution for real-time applications such as estimators for motor control. To optimize time and resource for such applications, this paper proposes a solution.

In the implementation of NN, parallel computation has the advantage of high speed but resource requirement is large which in turn increases the system cost. Sequential operation reduces resource but results in lower speed of operation. Hence, the controversial requirement of high speed and low cost is addressed in this paper.

This paper proposes a simple architecture to implement a complete NN using minimum resource regardless of the size of the network. A single largest layer (i.e., layer with maximum neurons) is implemented instead of implementing the complete network. This layer calls itself repeatedly and behaves like the different layers of the network. The control block ensures proper computation of the layer by placing the appropriate inputs, weights, biases, and excitation function of the layer that are currently being computed. The proposed multiplexed architecture implementation greatly reduces the resource requirement, thus making multilayer ANNs realizable with low-cost FPGAs. Section II describes the implementation of a single neuron. Section III introduces and details the implementation proposed in this paper. It quantifies the advantages claimed with suitable examples. Section IV validates the proposed method of NN implementation and its claims using a real-time application. Section V concludes the paper.

II. FPGA Implementation of a Single Neuron

A. Computational Blocks of a Single Neuron

The basic structure of a multi-input neuron with "n" inputs is shown in Fig. 1. Let $p_1, p_{2...}p_n$ be the inputs of the system. Let $w_1, w_{2...}w_n$ be the corresponding weights and "b" the bias. Let $f(x)$ be the nonlinear excitation function. The processing done by a neuron is described by the following:

$$y = f(x)$$

where

$$x = \sum_{i=1}^{n} p_i w_i + b \tag{1}$$

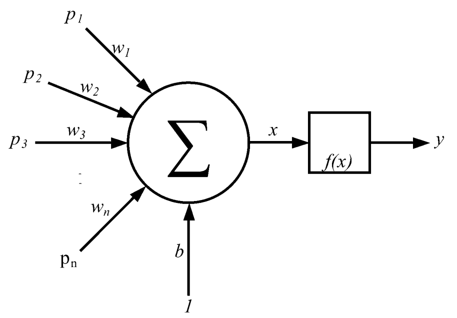


Fig. 1. Structure of a neuron.

where w_i is the weight in the ith connection and "b" is the bias. The function $f(x)$ is the nonlinear excitation function used in the neuron. The popularly used excitation functions are as listed as follows:

1) linear

$$f(x) = x; \tag{2}$$

2) log-sigmoid function

$$f(x) = \frac{1}{1+e^{-x}}; \tag{3}$$

3) tan-sigmoid function

$$f(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}. \tag{4}$$

The single-neuron implementation basically requires computational elements such as simple adder, multiplier, and complex evaluator of the nonlinear excitation function. Hence, it is clear that the computational blocks in a single-neuron implementation are the addition block, multiplication block, and excitation function block.

B. Precision of the Various Blocks in Single-Neuron Implementation in FPGA

In digital implementation of a single neuron, an important parameter is the precision of the various blocks. Word length/bit precision decides the output resolution. Higher resolution means larger resource requirement, which results in higher cost. Efficient hardware implementation can be achieved by matching suitable arithmetic resolution. As this paper aims to build multilayer NNs with minimum resource a 9-bit word length with one sign bit and eight data bits has been chosen for this implementation. In typical application hardware, matching the precision to the computational accuracy of the NN can further optimize complexity.

The input data $p_1, p_{2...}p_n$ are signed numbers in the range of -1 to $+1$ as the inputs to a neuron are normalized. This is represented using a signed 9-bit number (1 bit for sign, and 8 bit for data). The weights of the network are represented with 17 bits (1 bit for sign, 8 bit for whole part and 8 bit for fraction part). The bias is represented by a signed 25-bit number (1 bit for sign, 8 bit for whole part, and 16 bit for fractional part). The product of the weights and inputs are stored as 24-bit number

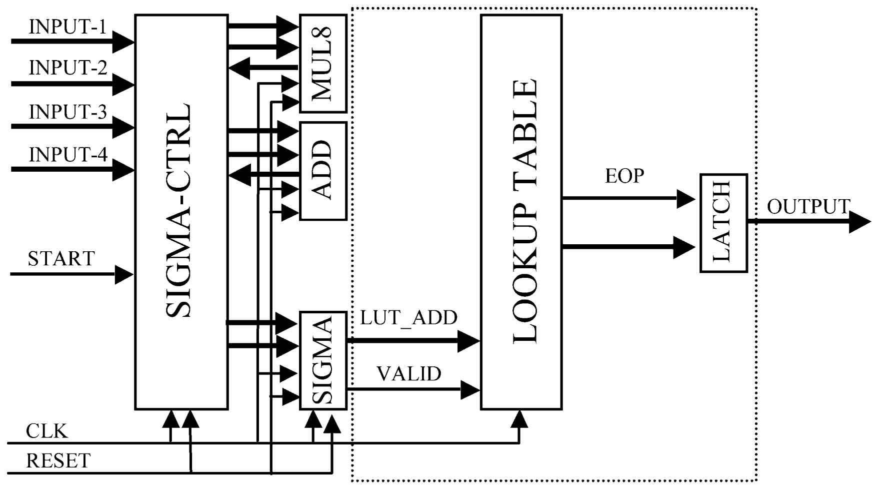


Fig. 2. Architecture of a three-input neuron for log-sigmoid excitation function with LUT.

along with a sign bit. The output $x = \sum_{i=1}^{n} p_i w_i + b$ is obtained with 29 bits (1 bit for sign + 12 bit for whole part + 16 bit for fractional part). The result of the excitation function $f(x)$ is obtained as a 9-bit signed number (1 bit for sign + 8 bit for data). The precision and word length is chosen so that a single neuron can handle a maximum of 16 inputs without the problem of overflow. However, modification to the word length could accommodate higher number of inputs to a neuron. Generally, in real-time applications (such as vector control of motor drives) of NNs the number of inputs to a neuron rarely exceeds 16.

C. Implementation of a Single Neuron in FPGA

The structure of a neuron is split into various computational subblocks and these blocks are implemented individually, and then, they are integrated to form the complete neuron. The various functional blocks are shown in Fig. 2. An LUT is used to determine the value of the excitation function. The functions of the various blocks are detailed as follows.

1) MUL8: It performs multiplication of two signed numbers. The inputs are obtained from the SIGMA_CTRL block and the product is returned back to the same block.
2) ADD: It performs the addition of two numbers. These inputs are obtained from the SIGMA_CTRL block and the sum is returned back to the same block.
3) SIGMA: It obtains the algebraic sum of two signed numbers. The inputs are obtained from the SIGMA_CTRL block. The output of this block is the address to the LUT, which is obtained by the truncation of the algebraic sum.
4) SIGMA_CTRL: This block receives the inputs of the neuron and coordinates the functions of the other blocks. It multiplexes and controls the MUL8, ADD, and SIGMA blocks. The SIGMA_CTRL block is a finite state machine that operates in a sequential manner. The sequence of operations are detailed as follows.
   a) SIGMA_CTRL block first places the correct weight and input for multiplication to the MUL8 block.
   b) This product and the appropriate summer register (depending on the sign of the product) are specified as inputs to the ADD block for addition.
   c) The partial sum thus obtained is updated in the SIGMA_CTRL block.
   d) The steps a)–c) are repeated with the appropriate values depending on the number of inputs.
   e) The bias and the summer register (depending on the sign of the bias) are placed as inputs to the ADD block for addition.
   f) The obtained partial results are updated in the SIGMA_CTRL block.
   g) The partial sums stored in the summer registers are routed to the SIGMA block to calculate the algebraic sum, and hence, obtain the value of x.
5) LUT: The valid signal from the SIGMA block activates the LUT. The LUT outputs the value for the excitation function depending on the address from the SIGMA block. The LUT also generates the end-of-process (EOP) signal, which latches the neuron output.

D. Design of LUT

The implementation of the excitation function in FPGA is done using the LUT that enables to use the inbuilt RAM available in FPGA integrated chip (IC). The use of LUTs reduces the resource requirement and improves the speed. In addition, the implementation of LUT needs no external RAM since the inbuilt memory is sufficient to implement the excitation function. As the excitation function is highly nonlinear a general procedure adopted to obtain an LUT of minimum size for a given resolution is detailed as follows.

1) Let n be the number of bits

$$y = f(x) = \frac{1}{1+e^{-x}}. \tag{5}$$

2) Determine the range of input (x) for which the range of output (y) is between 2^{-n} and 1–2^{-n}. Let x_1 and x_2 be the upper and lower limits of the input range. Substituting $\{1/(1+e^{-x1})\} = 2^{-n}$ and $\{1/(1+e^{-x2})\} = (1-2^{-n})$, it is found that

$$x_1 = -\ln(2^n - 1) \quad \text{and} \quad x_2 = +ln(2^n - 1). \tag{6}$$

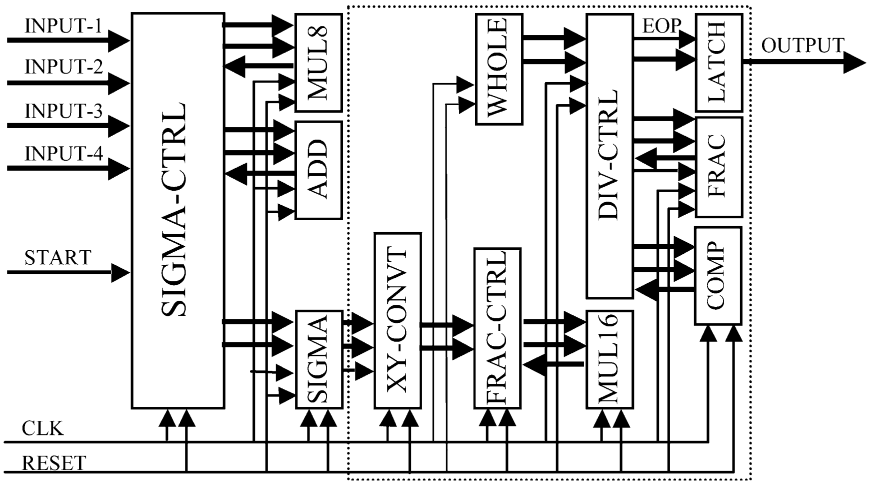


Fig. 3. Architecture of a three-input neuron for log-sigmoid excitation function without LUT.

3) Determine the change in input Δx that produces change in output (Δy) equal to 2^{-n} at the point of maximum slope. For a log-sigmoid excitation function, the maximum slope is at $x = 0$. The value of Δx for the output change of 2^{-n} can be obtained from

$$\Delta x = \ln\left(\frac{0.5 + 2^{-n}}{0.5 - 2^{-n}}\right). \tag{7}$$

4) The minimum number of LUT values is given by

$$(\mathrm{LUT})_{\min} = \frac{x_1 - x_2}{\Delta x}. \tag{8}$$

Appropriate number of bits that can address $(\mathrm{LUT})_{\min}$ are chosen. For the log-sigmoid excitation function with 8-bit resolution, x_1, x_2, and Δx are calculated from (6) and (7). It is found from (8) that $(\mathrm{LUT})_{\min} = 709$. In order to accommodate for $(\mathrm{LUT})_{\min}$, 10-bit address is required. Hence, 1-K RAM is used as LUT for log-sigmoid excitation function. The 1-K RAM has 1024 divisions; each of step size $\Delta x = 0.015625$ can accommodate the value of x in the range -8 to $+8$. With this method of LUT design, the nonlinearity of the excitation function is maintained for a given 8-bit resolution. Thus, a LUT of 1-K RAM for log-sigmoid excitation function with 8-bit resolution replaces the complete computation of the excitation function. However, the size of the LUT increases for higher resolution.

E. Comparison of Resource and Speed of a Single Neuron With and Without LUT

A single neuron with three inputs and various excitation functions was implemented with and without LUT in using "XCV400hq240." The architecture of a neuron without LUT is shown in Fig. 3. The blocks within the dotted lines compute the excitation function. Since more computational blocks are involved in calculation of excitation function, the resource requirement and computational time are high. It was observed that there is 70% reduction in resource requirement and 79% improvement in speed by using an LUT. The results are summarized in Table I.

TABLE I
RESOURCE AND SPEED REQUIREMENT OF A NEURON WITH AND WITHOUT LUT

Excitation Function		Implementation Type		% Saving
		LUT	COMPUT	
LOG SIGMOID	Slices Utilized	281	953	70.50
	Time Taken (μs)	5.00	24.2	79.33
TAN SIGMOID	Slices Utilized	282	1096	74.27
	Time Taken (μs)	5.00	29.8	83.22

III. HARDWARE IMPLEMENTATION OF AN NN

The implementation of a given network involves the implementation of all the neurons and every layer of the network. A multilayer network is shown in Fig. 4. Let $s^1, s^2 \ldots s^m$ be the number of neurons in layer $1, 2 \ldots m$. The total number of neurons in the given network is $S_{\text{total}} = S^1 + S^2 \ldots + S^m$. To calculate the total resource requirement (R_{total}) to implement any NN in an FPGA is obtained using

$$R_{\text{total}} = R \times S_{\text{total}} \tag{9}$$

where R is the resource requirement to implement a neuron. The resource requirement in an FPGA is given in terms of slices. An FPGA slice consists of two flip-flops and two LUTs and some associated mux, carry, and control logic. It is clear that as the number of layers increases the resource requirement and cost of the implementation increase as well. It is interesting to observe that even if the complete network is realized in hardware, computation of layers have to take place sequentially as the input to any layer is the output of the previous layer. The neurons in a layer compute in parallel but the layers compute sequentially. This sequential computation of the layers is exploited in this paper to realize a multilayer network in FPGA with minimum hardware.

A. Concept of Layer Multiplexing

In order to implement a given NN of any size with minimum hardware, the concept of multiplexing the layers of NN is pro-

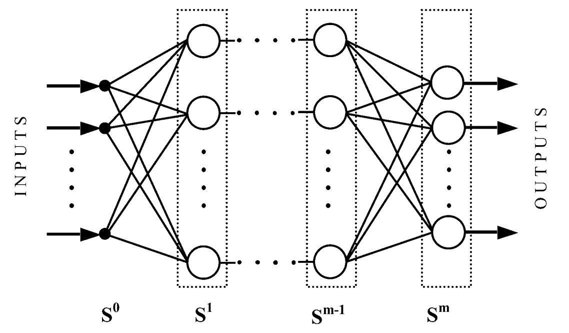


Fig. 4. General network with n layers.

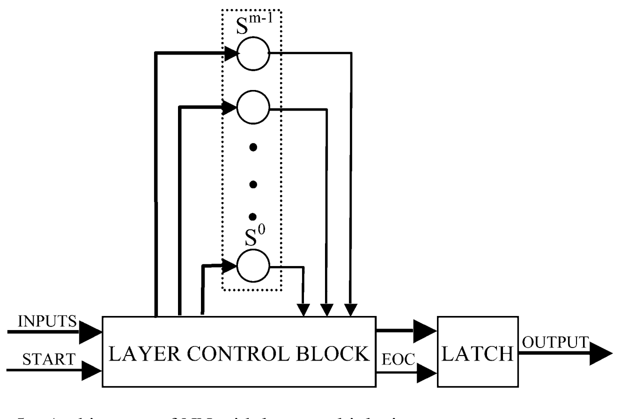


Fig. 5. Architecture of NN with layer multiplexing.

posed and introduced in this paper. Layer multiplexing is the implementation of the single largest layer (i.e., layer with maximum neurons) with each neuron having the maximum number of inputs and a control block to coordinate them. For example, consider a 3–6–8–2 network; the largest layer has eight neurons and the largest number of inputs to a neuron in the network is eight. Hence, for the considered example, eight neurons with each neuron having eight inputs is identified as the single largest layer for implementation. The implemented single layer is multiplexed to execute the functions of all layers of the network. The layer multiplexing requires an additional control block to coordinate the computation of every layer of NN. Hence, realization of complete network with a single layer implementation leads to a considerable resource reduction of FPGA.

The control block ensures the complete computation of NN using layer multiplexing by sequencing and placing the appropriate inputs, weights, biases, and value of excitation function (from LUT) of each layer. A generalized NN with n number of layers is shown in Fig. 4. Let S^0 be the number of inputs to the network and $S^1, S^{2\cdots}S^m$ be the number of neurons in layer $1, 2 \ldots m$. Let any layer say $(m-1)$th layer have the maximum number of neurons S^{m-1}. As explained earlier, only $(m-1)$th layer is implemented in FPGA with each neuron having maximum number of inputs. This single layer with the control block is used to realize the complete network.

B. Architecture for NN With Layer Multiplexing

A simple architecture suitable for FPGA implementation with layer multiplexing has been proposed as shown in Fig. 5. It is seen that the largest layer with each neuron having the maximum number of inputs is implemented. A start signal is given to the control block to initiate the operation of the network. The control block places simultaneously the correct set of inputs, weights, and biases to every neuron of the layer. The results of the neurons are computed parallel and sent to the layer control block to be provided as inputs to the next layer. Once the complete network is computed, the end of computation (EOC) signal is issued to latch the output of the NN, as shown in Fig. 5.

C. Implementation of Layer-Multiplexed NN

Implementation details of a layer-multiplexed NN are presented in this section. Four different network architectures are chosen such that for all the networks the single largest layer contains five neurons and each neuron has eight inputs. The architectures are as follows:

1) 8–5–3;
2) 8–5–5–3;
3) 8–5–5–5–3;
4) 8–5–5–5–5–3.

All those NNs have eight inputs and three outputs. The log-sigmoid excitation function is used for the hidden layers and the linear excitation function is used for the output layer. The choice of networks is to illustrate the saving in resource, which increases as the number of layers increases. The schematic for the implementation of those networks with layer multiplexing is shown in Fig. 6. The implementation includes the single largest layer and a control block. The internal blocks of the neuron have been detailed in Section II. The layer control block coordinates the computation of the different layers by placing the appropriate inputs, weights, biases, and value of excitation function (from LUT) for each layer. The sequential operation of the layer control block is detailed as follows.

1) The layer control block waits for the start signal to go high in order to initiate the operation.
2) It issues a start signal to all the neurons along with their respective inputs, weights, and biases. Each neuron block calculates the addresses for the LUT.
3) The addresses for the LUT from the neuron blocks are stored in the layer control block.
4) The addresses for the LUT are given from the layer control block to get the result of the excitation function.
5) The result of the excitation function is stored in the layer control block.
6) Steps 2)–5) are repeated with appropriate values of inputs to compute all the hidden layers of the network.
7) To implement the output layer (with linear excitation function), steps 2) and 3) are processed to get the final result. At the end of the computation of NN, the EOC signal is issued to latch the outputs.

The various steps involved in FPGA design flow are as follows: 1) design entry, 2) synthesis, 3) simulation, 4) implementation, and 5) device programming. The Verilog hardware description language (VHDL) design entry (coding), synthesis, implementation, and device programming have been done using Xilinx6.1i, and simulated with ModelSim XE II 5.7c. The programming file generated during device programming was downloaded and tested in the IC “XCV400hq240” using

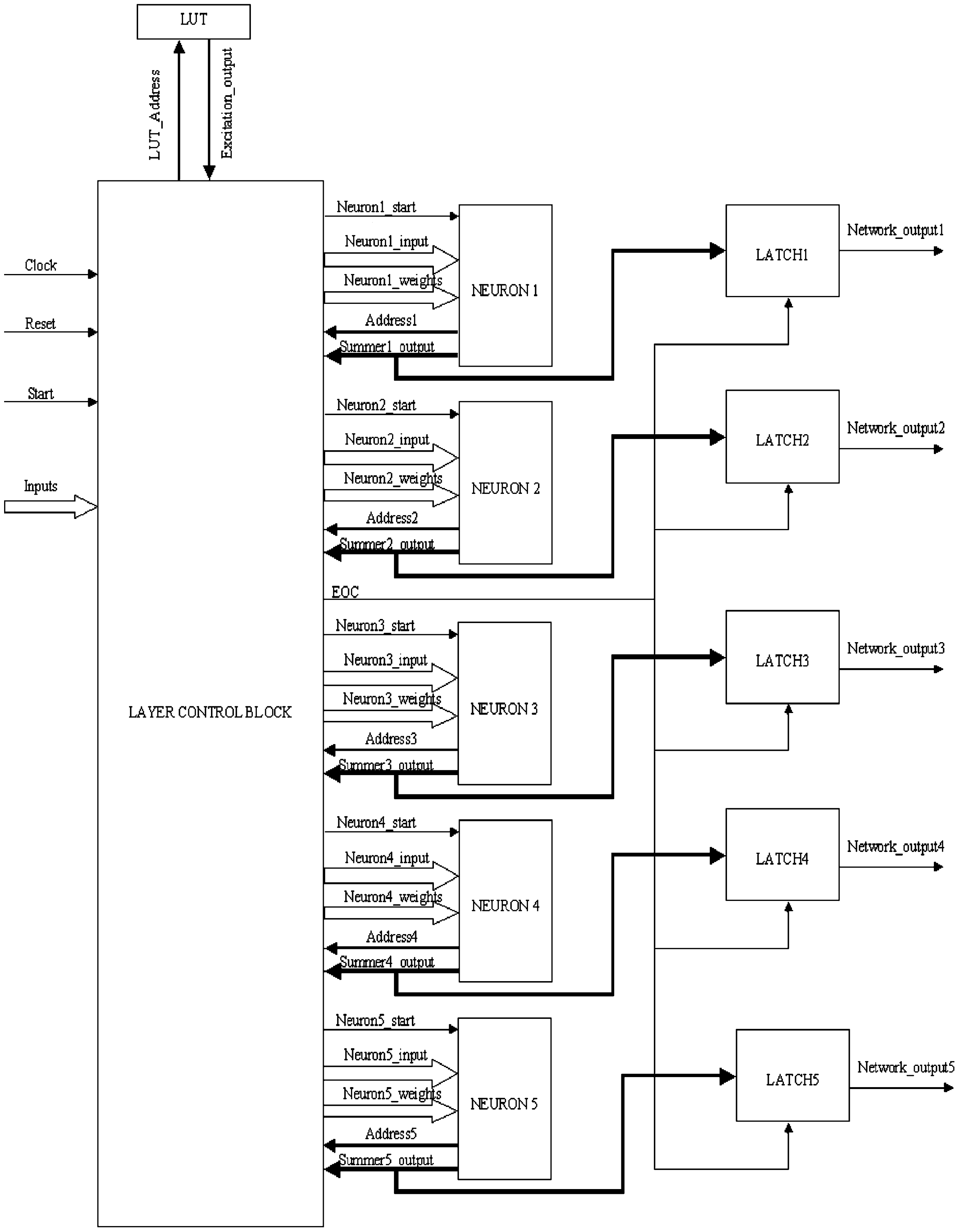


Fig. 6. Schematic for implementation of NN with layer multiplexing.

mechatronics test equipment—model MXUK-SMD-001. The highest speed achievable for the implementation is 73 MHz. The sample synthesis report for the network 8–5–5–3 is shown in Table II. Different multilayer NN architectures are implemented with and without layer multiplexing and various results are presented in Tables III and IV.

TABLE II
SYNTHESIS REPORT FOR AN NN OF 8–5–5–3

Device Selected	XCV400hq 240		
FEATURES	Present	Utilized	%
Slices	4800	2993	62
Flip Flops	9600	2558	26
LUTs with 4 input	9600	5460	56
Bonded IOBs	170	19	11
BRAMs	10	2	20
GCLKs	4	1	25

TABLE III
RESOURCE REQUIREMENT FOR VARIOUS NN ARCHITECTURE WITH CONVENTIONAL AND PROPOSED IMPLEMENTATION

Network Architecture	Resource Requirement in Slices			
	Implementation Type		Saving %	Control Block
	Con-ventional	**Proposed**		
8-5-3	3736	2863	23.36	329
8-5-5-3	6071	2993	50.70	403
8-5-5-5-3	8406	3035	63.89	450
8-5-5-5-5-3	10741	3055	71.55	472

TABLE IV
SPEED FOR VARIOUS NN ARCHITECTURE WITH CONVENTIONAL AND PROPOSED IMPLEMENTATIONS

Network Architecture	Execution in Clock Cycles		% Overheads
	Implementation Type		
	Con-ventional	**Proposed**	
8-5-3	111	120	8.1
8-5-5-3	158	186	17.7
8-5-5-5-3	205	351	22.9
8-5-5-5-5-3	252	317	25.79

The percent reduction in resource is shown in Table III. It can be seen that as the number of layers of NN increases, the percent saving in resource increases as well. The overhead in speed is shown in Table IV. The speed index is represented in clock cycles. It can be seen that as the number of layers increases, the slice requirement increases in both cases. In the conventional implementation, a complete layer has to be implemented; hence, the increase in slices is large. When using layer multiplexing, only the weights of the new layer and a little overhead on controller are required. Hence, the increase in slice is much lower compared to conventional implementation. The slices required for the control block are also shown in Table III. From Table III, it is proved that the proposed layer-multiplexed FPGA implementation derives maximum percent saving in resource with increased number of layers. However, the marginal decrease in speed may not be a limitation for many applications, as the clock frequency of FPGA devices is high. The resource saving has a good impact on the cost and enhances the viability of using NNs for any given application.

TABLE V
COMPARISON OF RESOURCE AND SPEED FOR FLUX ESTIMATOR

Network Architecture **(8-5-5-2)**	*Con-ventional Method*	*Proposed Method*	*Saving %*	*Overhead %*
Resource in Slices	5791	2863	50.56	-
Execution in cycles	158	186	-	17.7

IV. FPGA IMPLEMENTATION OF NEURAL-BASED FLUX ESTIMATOR

This paper proposes a method for implementation of NNs using FPGA with minimum resource without much loss in speed. This work is intended for real-time applications. The resource reduction technique is demonstrated by implementing a neural-based flux estimator using FPGA. The results validate the aforementioned technique.

Estimation of flux is essential for sensorless drives. Neural-based estimators have the advantage that they are robust to parameter variations, model independent, and may generalize well over a wide range [19]. Different NN architectures were considered based on performance and implementation aspects. The mean square error was chosen as the performance index and the maximum number of neurons in a layer was kept as small as possible. Considering all those issues, an 8–5–5–2 architecture was chosen.

The inputs to the ANN are the direct and quadrature axis stator voltages and currents measured at the k and $k-1$ sample: $v_{ds}(k), v_{ds}(k-1), v_{qs}(k), v_{qs}(k-1), i_{ds}(k), i_{ds}(k-1), i_{qs}(k), i_{qs}(k-1)$ and the output is the direct and quadrature axis flux $\psi_{ds}(k), \psi_{qs}(k)$. Log-sigmoid activation function was used for the hidden layers of the network and the linear activation function is chosen for the output layer. The network was trained using Matlab NN toolbox for 1040 training data for a mean square error of 3×10^{-10}.

The flux estimator developed was implemented in FPGA using the method proposed in the paper. The proposed method requires the implementation of only five neurons with each neuron having eight inputs and a control block, whereas implementation of the complete network would require 13 neurons.

The flux estimator was implemented using the conventional method as well as the method proposed in this paper. The results obtained are shown in Table V. It can be seen from Table V that the saving in resource is 50% and the overhead in speed is 17.7%. As the operating clock frequency of FPGA is high, the extra overhead in cycles does not affect overall operation of the flux estimator.

The actual values and the estimated values from the FPGA-based flux estimator with 8-bit precision as well as the error at each sample are shown in Fig. 7. For clarity only, one cycle is shown. Fig. 7(a) shows the direct axis flux and Fig. 7(b) shows the quadrature axis. The solid line shows the actual value of flux, the dashed line shows the estimated value of flux, and the dotted line shows the error. It can be seen that, though in simulation the mean square error obtained was 3×10^{-10}, in FPGA the mean square error has increased to $.6.497 \times 10^{-5}$ because of a bit precision. Higher bit precision increases the accuracy

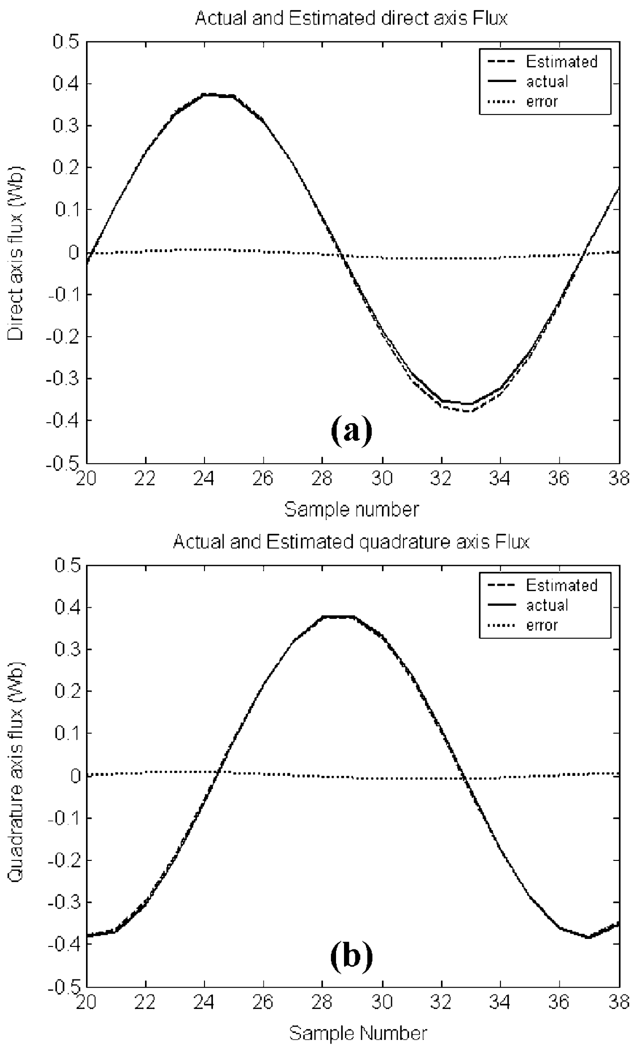


Fig. 7. Actual and estimated flux. (a) Direct axis flux. (b) Quadrature axis flux.

of estimator but the implementation demands increase slice/resource, and hence, the system cost. This is true irrespective of the method of implementation. However, for a given bit precision, the proposed method of FPGA-based NN implementation uses reduced resource and system cost.

V. Conclusion

Multilayer feedforward NNs have been implemented using FPGA. The implementation of a single neuron using LUT reduces the resource for implementation and time for execution. Further, to realize multilayer networks with minimum resource, the proposed method of implementing only the single largest layer along with a control block is detailed. The control block ensures proper functioning, by assigning simultaneously the appropriate inputs, weights, biases, and excitation function for every neuron of the layer that is currently being computed in parallel. Different network configurations have been implemented with and without layer multiplexing. Layer multiplexing reduces the resource requirement for any given application. The percent saving in resource increases with the number of layers of the network. With increase in the number of layers, the control block complexity increases resulting in a moderate overhead on speed. As the operating clock frequency of FPGA devices is high the actual increase in computational time per sample may not be a constraint for many applications. The large saving in resource leads to reduced cost making NN applications more feasible and promising. To substantiate the claim, a flux estimator for sensorless drives has been developed and tested. The results presented indicate the efficacy of the proposed technique.

References

[1] B. Hassinbi, D. G. Stork, and G. J. Wolff, "Optimal brain surgeon and general network pruning," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 1992, vol. 2, pp. 441–444.
[2] B. Widrow and R. Winter, "Neural nets for adaptive filtering and adaptive pattern recognition," *Computer*, vol. 21, no. 3, pp. 25–39, Mar. 1988.
[3] K. Fukushima, S. Miyake, and T. Ito, "Neocognitron: A neural network model for a mechanism of visual pattern recognition," *IEEE Trans. Syst., Man, Cybern.* , vol. SMC-13, no. 5, pp. 826–834, 1983.
[4] S. Grossberg, E. Mingolla, and D. Todorovic, "A neural network architecture for preattentive vision," *IEEE Trans. Biomed. Eng.*, vol. 36, no. 1, pp. 65–84, Jan. 1989.
[5] L. M. Reyneri, "Implementation issues of neuro-fuzzy hardware: Going towards HW/SW codesign," *IEEE Trans. Neural Netw.*, vol. 14, no. 1, pp. 176–194, Jan. 2003.
[6] M. Cristea and A. Dinu, "A new neural network approach to induction motor speed control," in *Proc. IEEE Power Electron. Specialist Conf.*, 2001, vol. 2, pp. 784–788.
[7] Y. J. Chen and D. Plessis, "Neural network implementation on a FPGA," in *Proc. IEEE Africon Conf.*, 2002, vol. 1, pp. 337–342.
[8] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," *IEEE Trans. Neural Netw.*, vol. 4, no. 1, pp. 53–62, Jan. 1993.
[9] B. Noory and V. Groza, "A reconfigurable approach to hardware implementation of neural networks," in *Can. Conf. Electr. Comput. Eng.*, 2003, pp. 1861–1863.
[10] J. Zhu, G. J. Milne, and B. K. Gunther, "Towards an FPGA based reconfigurable computing environment for neural network implementations," *Inst. Elect. Eng. Proc. Artif. Neural Netw.*, vol. 2, no. 470, pp. 661–666, Sep. 1999.
[11] R. H. Turner and R. F. Woods, "Highly efficient limited range multipliers for LUT-based FPGA architectures," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 15, no. 10, pp. 1113–1117, Oct. 2004.
[12] X. Yu and D. Deut, "Implementing neural networks in FPGA," in *Proc. Hardware Implemen. Neural Netw. Fuzzy Logic Inst. Elect. Eng. Colloq.*, Mar. 9, 1994, pp. 1–5.
[13] Y. Taright and M. Hubin, "FPGA implementation of a multilayer preceptron neural network using VHDL," *Proc. Int. Conf. Signal Process. (ICSP)*, vol. 2, pp. 1306–1310, 1998.
[14] M. M. Syiam, H. M. Klash, I. I. Mahmoud, and S. S. Haggag, "Hardware implementation of neural network on FPGA for accidents diagnosis of the multi-purpose reactor of Egypt," in *Proc. 15th Int. Conf. Microelectron. (ICM)*, Dec. 2003, pp. 326–329.
[15] N. M. Botors and M. Abdul-Aziz, "Hardware implementation of artificial neural network using field programmable gate arrays," *IEEE Trans. Ind. Electron.*, vol. 41, no. 6, pp. 665–667, Dec. 1994.
[16] E. Pasero and M. Perri, "Hw-Sw Co design of a flexible neural controller through a FPGA-based neural network programmed in VHDL," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, Jul. 2004, pp. 4639–4644.
[17] S. S. Kim and S. Jung, "Hardware implementation of real time neural network controller with a DSP and an FPGA," in *Proc. IEEE Int. Conf. Robot. Autom.*, Apr. 2004, vol. 5, pp. 3161–3165.
[18] K. M. Hornick, M. Stinchcombe, and H. white, "Multilayer feedforward neural networks are universal approximators," *Neural Netw.*, vol. 2, no. 5, pp. 141–154, 1985.
[19] P. Vas, *Sensorless Vector and Direct Torque Control.* Oxford, U.K.: Oxford Univ. Press, 1998.

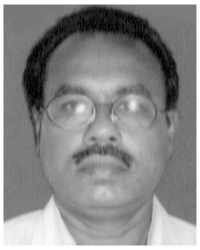
S. Himavathi received the B.E. degree in electrical and electronics engineering from College of Engineering, Guindy, Chennai, India, in 1984, the M.E. degree in instrumentation technology from Madras Institute of Technology, Chennai, India, in 1987, and the Ph.D. degree in the area of fuzzy modeling from Anna University, Chennai, India, in 2003.

Currently, she is an Assistant Professor in the Department of Electrical and Electronics Engineering, Pondicherry Engineering College, Pondicherry, India. She has approximately 30 publications to her credit. Her research interests are fuzzy systems, NNs, and hybrid systems and their applications.

Dr. Himavathi is a Reviewer of the *AMSE Journal of Modeling*, IEEE Industrial Electronics Society, and Asian Neural Networks Society.

D. Anitha received the B.Tech. degree in electrical and electronics engineering and the M.S. degree in drives and control from Pondicherry Engineering College, Pondicherry, India, in 2003 and 2005, respectively.

She has executed a project on FPGA implementation of estimators. Currently, she is on a Faculty of SRM College of Engineering, Deemed University, Chennai, India. Her areas of interest are FPGA, NNs, and drives.



A. Muthuramalingam received the B.E. degree (with distinction and Gold Medal) in electrical and electronics engineering from Annamalai University, Tamil Nadu, India, in 1982, the M.Tech. degree (with distinction) in industrial electronics from KREC, Mangalore University, Karnataka, India, in 1985, and the Ph.D. degree in power electronics and drives from Indian Institute of Technology, Madras, India, in 2002.

Currently, he is an Assistant Professor at the Pondicherry Engineering College, Pondicherry, India, where he has been harnessing the developmental activities of the Electrical and Electronics Engineering Department. He has taken an assignment in which students learn technological advancements in depth through design-oriented courses using computer-aided design (CAD) tools. His current research interest is in application of soft-switching techniques for medium-power conversion systems and AI techniques to PE systems and solid-state drives.