2010 2nd International Asia Conference on Informatics in Control, Automation and Robotics

A Single Layer Architecture to FPGA Implementation of BP Artificial Neural Network

Liu Shoushan, Chen Yan, Xu Wenshang, Zhang Tongjun
College of Information and Electrical Engineering
Shandong University of Science and Technology
Qingdao, China
E-mail: lsscy@hotmail.com

***Abstract*—Based on discussions of the hardware implementations of artificial neural network (ANN), the single layer architecture to FPGA of the back propagation artificial neural network (BP ANN) is proposed. To construct the single layer architecture, the computing blocks of BP ANN are presented. The construction of the single layer architecture is described. According the experiments of the implementation in FPGA and the defects classification in ultrasonic non-destruction detection of carbon fibre reinforced plastic (CFRP), the single layer architecture of BP ANN performs well and can serve the applications.**

***Index Terms*—neural network, architecture, hardware implementation, FPGA**

I. INTRODUCTION

The analog or digital hardware implementations of the artificial neural networks (ANN) have always drawn more attentions, in compare with the software implementations, since the hardware implementation can perform the parallelism of the ANN. The digital hardware implementation is more popular comparing to analog hardware implementation, 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 [1]. The field programmable gate array (FPGA)-based implementation of ANN can provide a low-cost compromise between speed and area efficiency of application specific integrated chip (ASIC), and flexibility of neuro-processors on the other [2]-[6].

However, implementation of ANN in FPGA has some challenges. One of the most prominent challenges is resource consumption. Generally, the nonlinear excitation functions and the synapses (multipliers) always make major contribution to the FPGA resource consumption in a certain ANN. Aimed at decreasing the resource consumption, some researches have focused on the renew of multiplier architecture or multiplication algorithm of the ANN [4]-[7]. Lookup tables (LUTs) are used to store the excitation function in order to improve speed and reduce the resource requirement [9]. Dynamic adaptive memories are proposed to reduce memory requirement [10]. In addition, implementation of dedicated NNs with few numbers of neurons has been reported [11]. Other ways to decreasing the resource consumption have focused on the decomposition of ANN and implementing the decomposed ANN (different phases or different layers, for example) in one FPGA through reconfiguration [12]-[15]. However, reconfiguration time of the FPGA is always another bottleneck to the running ANN in situation of real-time signal processing or control [17]-[18].

For the multilayer back propagation artificial neural network (BP ANN), the bottlenecks of resource consumption and reconfiguration time for the FPGA-based implementation are significant. Upon the reconfigurable ultrasonic signal processing system [18], this paper is to propose a solution to the bottlenecks of BP ANN implementation in FPGA.

In this paper, single layer architecture of a BP ANN being implemented in FPGA is to be presented. Instead of deploying all layers in one FPGA, or decomposing the layers and phases for reconfiguration, the single layer can carry out different layers and different phases (feed-forward, back propagation, weights updating), through a reuse mechanism based on multiplexing control state machine. The rest of this paper is organized in 4 sections. In Section II, the algorithm of BP ANN is introduced, and the computing blocks of the BP ANN for multiplexing in a single layer is described. In Section III, the BP ANN architecture in a single layer is presented. Section IV proposes the experiments and results. Section V draws conclusions.

II. ALGORITHM OF BP ANN

In applications of pattern recognition, the BP ANN have always been used as the classification after the feature extraction algorithm [18]-[19]. Generally, A BP ANN with one hidden layer can virtually approximate any nonlinear function to any degree of accuracy provided sufficient number of neurons in a hidden unit is available [20]. A running BP ANN always performs two processes: the training process and classification process. When the training process completed, a final structure of the BP ANN would be confirmed for pattern recognition. The training process with different phases can be described through different blocks of computation.

A. Feed-forward Computations of Training Process

The feed-forward phase includes the computations from the input layer to hidden layer and the computations from hidden layer to output layer. Let $X_k = [x_1^k, x_2^k, \ldots, x_n^k]$ ($i = 1,2,\ldots,n$, $k = 1,2,\ldots,m$) be the inputs of the BP ANN. Let $Y_k = [y_1^k, y_2^k, \ldots, y_q^k]$ ($t = 1,2,\ldots,q$) be the outputs of output layer. Let W_{ij} ($j = 1,2,\ldots,p$) be the

978-1-4244-5194-4/10/$26.00 ©2010 IEEE

neuron connection weight of input layer and hidden layer. Let V_{ij} be the neuron connection weight of hidden layer and output layer. The threshold of input layer and hidden layer is denoted as θ_j, while the threshold of hidden layer and output layer is denoted as γ_t. The input of a neuron in hidden layer is

$$s_j = \sum_{i=1}^{n} W_{ij} x_i - \theta_j \tag{1}$$

Let $x_0 = 1, W_{0j} = \theta_j$, (1) can be modified as

$$s_j = \sum_{i=0}^{n} W_{ij} x_i \tag{2}$$

Then the neuron output of hidden layer can be derived through the nonlinear activating function

$$b_j = f(s_j) \tag{3}$$

The activating function of (3) is the log sigmoid function

$$f(s) = \frac{1}{1+e^{-s}} \tag{4}$$

The input of a neuron in output layer is

$$l_t = \sum_{j=0}^{p} V_{jt} b_j \tag{5}$$

The (5) is a modified equation, just as same as the modified (2) from (1). Then the neuron output of output layer is

$$c_t = f(l_t) \tag{6}$$

B. Error Back-propagations

When the back-propagation is activated, the first step is calculating the propagating error from output layer. Let y_t^k be the desired result of output layer neuron. Let c_t^k be the actual value of output layer neuron. Let the d_t^k be the propagating error from output layer, then

$$d_t^k = (y_t^k - c_t) f'(l_t) \tag{7}$$

The second step: let e_j^k be the propagating error from hidden layer, then

$$e_j^k = \left[\sum_{t=1}^{q} d_t V_{jt} \right] f'(s) \tag{8}$$

C. Updating the Weights

The weight, from hidden layer to output layer, for next feed-forward computation is updated as

$$V_{jt}(N+1) = V_{jt}(N) + \alpha d_t^k b_j \tag{9}$$

The weight, from input layer to hidden layer, for next feed-forward computation is updated as

$$W_{ij}(N+1) = W_{ij}(N) + \beta e_j^k x_i^k \tag{10}$$

The N in (9) and (10) means the N_{th} computation of training process

III. THE SINGLE LAYER ARCHITECTURE OF BP ANN BASED SYSTOLIC ARRAY

It can be calculated from the computation blocks in Section II that the calculations of the functions include the multiplication and the addition (subtraction) except for the log sigmoid function. It means that different function, except for the log sigmoid function and its derivative, can share a unit of multiplication and addition. Inspired by a one-dimensional systolic array of the finite impulse response (FIR) filter, the single layer architecture of the BP ANN can be calculated.

A. The One-Dimentional Systolic Array of FIR Filter

A FIR filter can be implemented in VLSI with the architectures of different one-dimensional systolic arrays. The array with Broadcast Input-Move Weights-Results Stay, as shown in Fig.1, can be adopted as a basic architecture upon which the single layer architecture of BP ANN is constructed [21]. In Fig. 1, W_i *(i=0,1,2,...)* is the weight. x_i *(i=0,1,2,3...)* is the input signal sequence. When one input value is broadcasted, the calculating unit (*CU*) composed with a multiplier and an adder will perform the multiplication and addition, and the results will be saved in the register. The weight will be updated when an operation of the calculating unit is finished. When the calculations, in response to the input signal sequence, are finished, the values being saved in result registers will be the accumulations of multiplying results of all input values and corresponding weights.

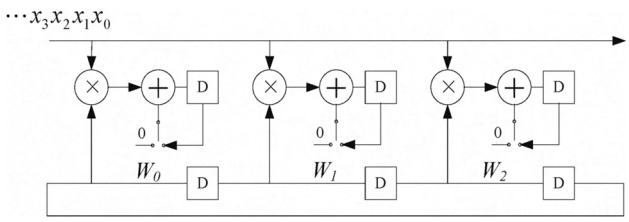


Figure 1. The one-dimensional systolic array of FIR filter

B. The Single Layer Architecture of BP ANN

Based on the one-dimensional systolic array of FIR filter in Fig. 1, the single layer architecture of BP ANN can be constructed according the three phases (feed-forward, back propagation, weights updating).

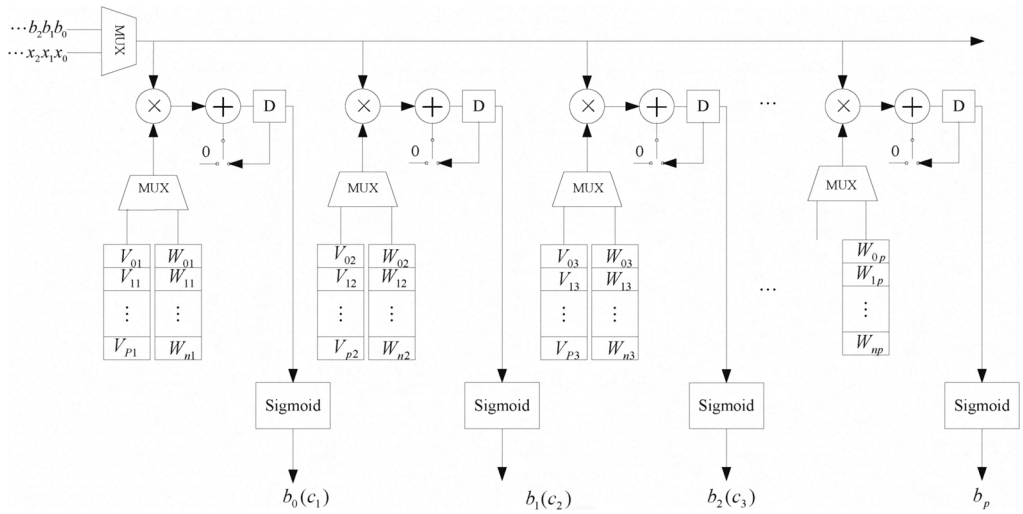


Figure 2. The one-dimensional systolic array of FIR filter

1) The single layer architecture of feed-forward computation

In this paper, a three layers network of 10-20-3 is taken as the example upon which the single layer architecture is constructed. The single layer architecture of feed-forward computation is shown in Fig. 2.

In Fig. 2, the computations from hidden layer to output layer (as the first step of feed-forward) share the calculating unit (*CU*)of multiplication and addition with computations from input layer to hidden layer (as the second step of feed-forward) through multiplexers (shown as MUX in Fig. 2). The computations of the first step are carried out by (2) and (3), and the computations of the second step are carried out by (5) and (6). When the computations, from input layer to hidden layer, are activated, the serial data (x_i *(i=0,1,2,3...)*) will be broadcasted on the units of multiplication and addition. At the same time, the connection weights (W_i *(i=0,1,2,...)*)of input layer and hidden layer will be transported to the multiplication. The multiplied results of the input data and the weights will be accumulated by the adder, till the data inputs are over. When the data inputs ended, all of the accumulated result in registers (shown as the D in Fig. 2) will be outputted to the sigmoid functions to finish the last computation of the first step. The results (shown as b_i *(i=0,1,...,p)* in Fig. 2) of the sigmoid functions will be saved for the second step. After the first step, the computations from hidden layer to output layer (the second step) will be activated, by sharing the same *CU* with the first step. There have two multiplexers: one is for the broadcasting data and the results of first step; another is for the weights of two computing steps. As the output layer has three neurons, the computations of second step only share three *CUs* with the first step. In Fig. 2, the neurons number of the single layer architecture is the same as that of the largest layer.

2) Modified architecture including the back propagation

The back propagation is carried out by (7) and (8). For three layers network of 10-20-3, the implementation of (7) only need three neurons of the architecture in Fig. 2. The implementation of (8) needs all neurons of the architecture. In order to perform the (7) and (8) in the single layer architecture, the three neurons on the left of Fig.2 are modified. One of the modified neurons including the (7) and (8) is shown in Fig. 3. In Fig. 3, different computations share the same calculation (the multiplication and addition) through multiplexers (MUX1, MUX2, MUX3). The subtraction of y_t and c_t is for (7). As the single layer architecture for the network of 10-20-3 needs 20 neurons (or calculation units), except for the three neurons including (7) and (8), the other 17 neurons do not need to perform the (7) during back propagation. One of the 17 neurons with (8) during back propagation is shown in Fig. 4.

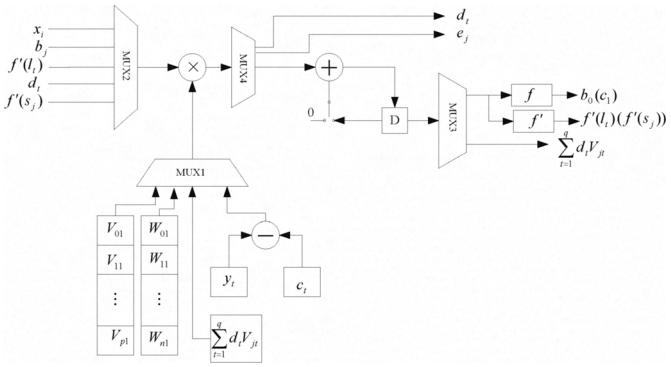


Figure 3. The modified neuron including (7) and (8)

3) The modified architecture including the weights updates

The weights updates are carried out by (9) and (10). As (9) and (10) share the same computation, architecture for the two equations can be constructed, as shown in Fig. 5. Merging the architecture in Fig. 5 to the neuron in Fig.3 and the neuron in Fig.4 separately, the neuron, as shown in Fig. 6, performing feed-forward, back propagation and weight update can be constructed, and the neuron, as shown in Fig. 7, without (7) and (9) can be constructed as well.

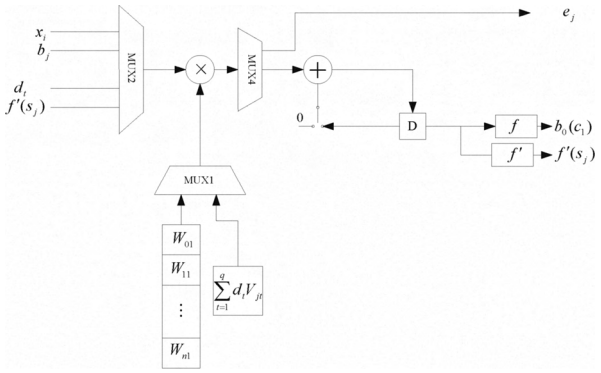


Figure 4. The modified neuron with (8)

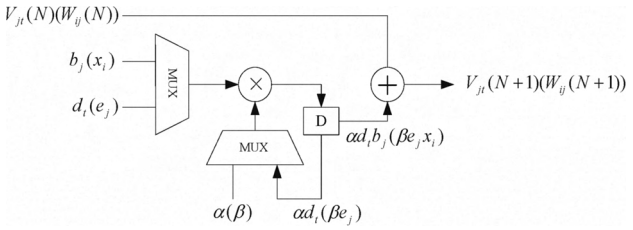


Figure 5. The multiplexing architecture of (9) and (10)

4) The single layer architecture of the neural network

Upon the neuron architectures in Fig. 6 and in Fig. 7, the single layer framwork of BP ANN, as shown in Fig. 8, can be constructed. In Fig. 8, the PE_i *(i=1, 2, 3)* is the neuron with the architecture in Fig. 6, while the PE_i *(i=4, 5, …, 20)* is the neuron with the architecture in Fig. 7. Except that the x_i and b_j are inputted in serial, the other inputs are all in parallel, as well as all the outputs

5) The memory mechanism

In the single layer architecture, the parameters (or the inputs and outputs) transportations of different *PE* or different computation blocks are all carried out through the memory units of parameters (or the inputs and outputs). Accordingly, the memory can be: first in first out (FIFO), parallel in serial out (PISO), parallel in parallel out (PIPO). The memory units for weights (W_{ij} and V_{jt}) between the *PEs* are in FIFO. The memory unit for b_j is in PISO. Other memory units for parameters are in PIPO.

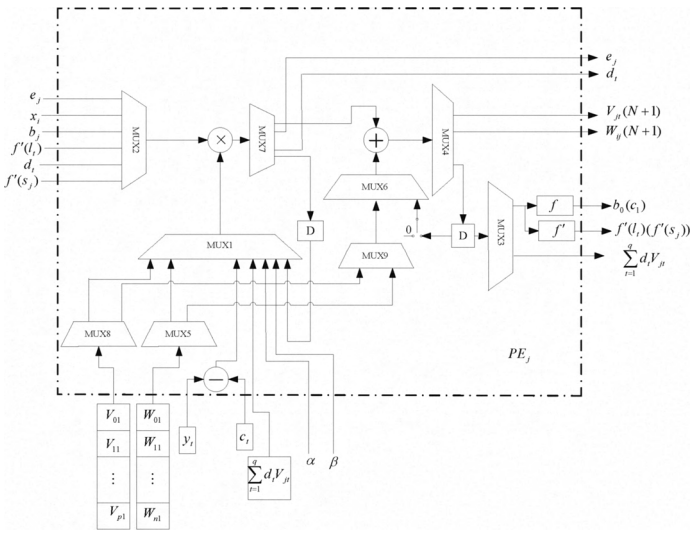


Figure 6. The neuron performing feed-forward, back propagation and weight update

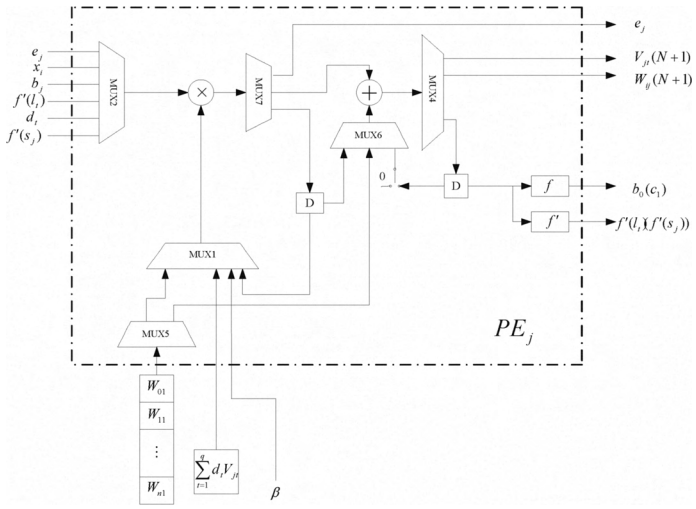


Figure 7. The neuron without (7) and (9)

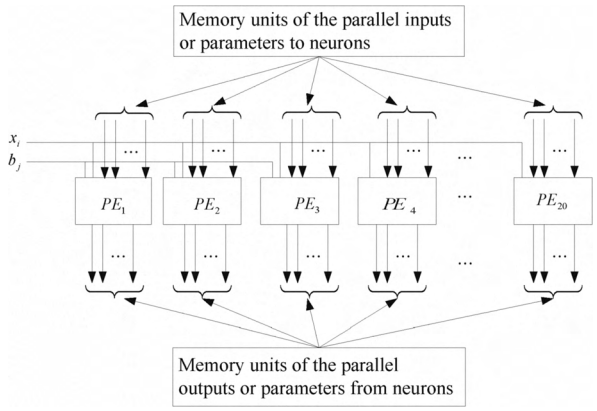


Figure 8. The single layer framework of the BP ANN

6) Timing control mechanism

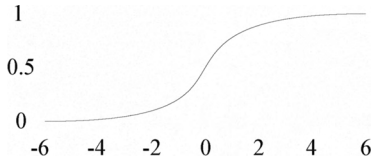
A finite state machine (FSM) is constructed to perform the timing control to different computing blocks. Basically, a computing block, or computing channel, is enabled by the multiplexers (MUX_i *(i=1, 2, … , 7)*) that are controlled by the FSM. According the input and output of the multiplexer, the control signals to computing block selection can be confirmed. The control signals numbers of the multiplexers, from MUX_1 to MUX_9, are list as: 3, 3, 1, 2, 1, 1, 2, 1, 1. The 15 control signals can be managed by a control register (as C_{MUX}) with 15 bits. The computing block selections are carried out by C_{MUX} to different multiplexers. In accordance with the order of control signals numbers listed above, the lowest three bits of C_{MUX} will be for the *MUX1*, the highest bit of C_{MUX} will be for *MUX9*. Corresponding to computing blocks, there have 11 states for the FSM. The control codes in C_{MUX} to the 11 states, with the corresponding computing blocks, are shown in Tab. I.

TABLE I. CONTROL CODES OF C_{MUX}

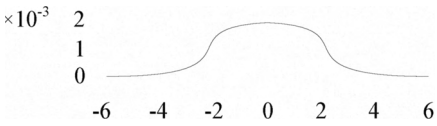
States	Computing Blocks	Control Codes of C_{MUX} ($\|a_{13}a_{12}\|\ldots\|a_5a_4a_3\|a_2a_1a_0$)
State 1	$s_j = \sum_{i=0}^{n} W_{ij} x_i$	0|X|00|0|0|00|0|000|000
State 2	$b_j = f(s_j), f'(s_j)$	
State 3	$l_t = \sum_{j=0}^{p} V_{jt} b_j$,	0|X|00|0|0|00|0|001|001
State 4	$c_t = f(l_t), f'(l_t)$	
State 5	$d_t^k = (y_t^k - c_t) f'(l_t)$	X|X|01|X|X|XX|X|010|010
State 6	$\sum_{t=1}^{q} d_t V_{jt}$	1|1|00|X|0|00|1|011|001
State 7	$e_j^k = [\sum_{t=1}^{q} d_t V_{jt}] f'(s_j)$	X|X|10|0|X|XX|X|100|011
State 8	αd_t^k	X|X|11|X|X|XX|X|011|100
State 9	$V_{jt}(N+1) = V_{jt}(N) + \alpha d_t^k b_j$	1|1|00|1|0|01|X|001|001
State 10	βe_j^k	X|X|11|X|X|XX|X|101|110
State 11	$W_{ij}(N+1) = W_{ij}(N) + \beta e_j^k x_i^k$	1|X|00|1|1|10|X|000|001

7) Implementations of Excitation Function

The excitation function in the single layer BP ANN is the log sigmoid function. To simplify the architecture and to improve the operation speed of the sigmoid function in FPGA, a look-up table (LUT) is adopted to realize the computation of sigmoid function. The log sigmoid function and its differential are shown in Fig. 9.



(a) Log sigmoid function



(b) Differential of log sigmoid function

Figure 9. Log sigmoid function and its differential

Except for the domain of [-4, 4], the amplitude curve of the differential of log sigmoid function is flat approximately. Aimed at reducing the resource costs to implementing the LUT in FPGA without sacrificing precision, a fixed amplitude value is adopted when the inputs of the differential function are out of [-4, 4]. When the inputs fall into the [-4, 4], the LUT will be performed. The implementation of log sigmoid function differential in FPGA is shown in Fig. 10. The LUT is realized through the Block RAM in FPGA. The values of the differential are saved in RAM. The inputs are as the RAM address indexes, as well as being sent to the Comparator. The comparative results control the multiplexer (MUX) to determine which look up path is selected: when the input values are less than -4, the output of the differential will be 5; when the input values are bigger than 4, the output of the differential will be 250; when the input values fall into [-4, 4], the input values will act as the address indexes.

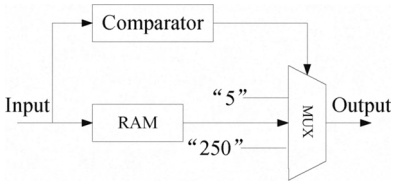


Figure 10. Implementation of log sigmoid function differential

IV. EXPERIMENTS AND RESULTS

The experiments include the architecture implementation in FPGA and the defects classification of carbon fibre reinforced plastic (CFRP).

A. Implementation of BP ANN

The single layer architecture of the BP ANN with 10-20-3 is carried out in FPGAs. The FSM takes charge of the operation states of 20 parallel calculation units by broadcasting the control signals to the calculation units and accepting the state signals from the calculation units. Controlled by the FSM, the calculation units perform the data acquisition, calculation in different computing blocks and output the results. The memory units afford the calculation results storage of different computing blocks, the weights

storage and the storage of original input and the final output results. Fig. 11 shows the single layer structure implemented in FPGA.

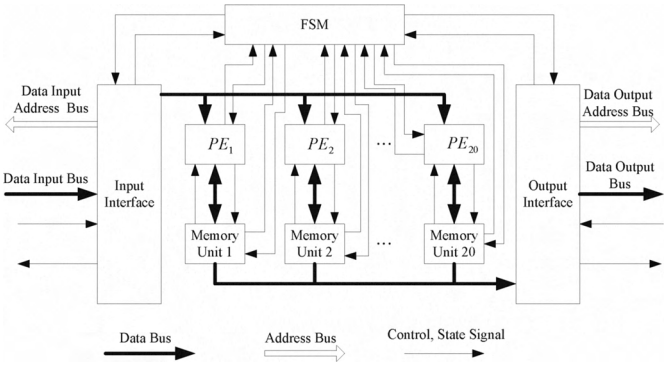


Figure 11. The single layer structure implemented in FPGA

The design and implementation are carried out using the FPGA of ALTERA Corporation with QUARTUS II. TABLE II shows the synthesis results of the single layer BP ANN architecture of 10-20-3 in different FPGAs.

Step 1: 100 samples of ultrasonic signal with three defects are obtained. The algorithms of signal decomposition by using 5 levels wavelet packet are carried out, and 10 sub-bands energies of every sample are normalized as the features for classification. 100 samples that are composed of the extracted features are divided into two groups. One group with 60 samples is used for training. The other group with 40 samples is used for classification.

Step 2: training the BP ANN of single layer architecture in FPGA with 60 samples. The relation of the iterations number of neural network training and the tolerance error within 10^{-3} is shown in Fig. 12.

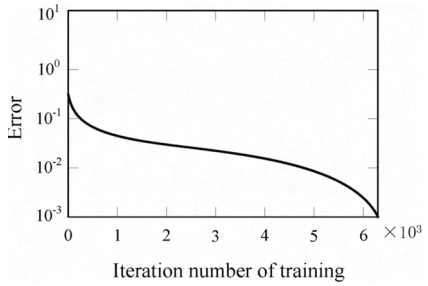


Figure 12. Relation of iterations number and tolerance error

TABLE II. SYNTHESIS RESULTS OF THE SINGLE LAYER ARCHITECTURE IN DIFFERENT FPGAS

FPGA	Element Logic	Pins	Multiplier with 9 bits	RAM (Bits)	Clock (MHz)
Cyclone	5013	70		8192	89.31
Cyclone II	1178	70	20	8192	109.06

B. *Experiment to the defects Classification of Carbon Fibre Reiforced Plastic*

BP ANN can be used as the defects classification in ultrasonic non-destruction detection of carbon fiber reinforced plastic (CFRP) in aircraft structures [20]. As the aircraft structures with CFRP are always in multilayer, such as the skin-honeycomb core structure, the defects always include the fatigue crack among the layers, the voids, lap-splice detach, adhesive-off, et al [22]. Before the classification using BP ANN, the energy feature extractions from the ultrasonic signal by using wavelet should have been carried out. The experiment objects are composite skin-honeycomb core structure with three kinds of defects: the lap-splice detaches of skin; the adhesive-off between skin and honeycomb core, the artificial void with 3 millimeters. The experiment process as follow:

Step3: classifying the defects in the 40 samples after training. The classification results of the single layer architecture in FPGA, in compare to the classification by using software implementations in personal computer, are shown in TABLE III.

V. CONCLUSION

Single layer architecture of BP ANN has been carried out in this paper. The implementation of the single layer architecture to FPGA reduces the resource consumption and time consumption due to reconfiguration. According the experiment of defects classification in ultrasonic non-destruction detection of carbon fibre reinforced plastic (CFRP), the single layer architecture BP ANN perform the classification well. Reconfiguration can only occur during

TABLE III. RESULTS OF DEFECTS CLASSIFICATION OF CFRP THROUGH DIFFERENT IMPLEMENTATIONS

Defects	Rate of Correct Classification (%)		Rate of Misclassified to Other Defects (%)		Rate of Misclassified to none-defects (%)	
	Implementation in FPGA	Implementation in software	Implementation in FPGA	Implementation in software	Implementation in FPGA	Implementation in software
lap-splice detach of skin	90.27	90.31	4.61	4.23	5.12	5.46
the adhesive-off of between skin and honeycomb core	89.47	89.46	6.21	6.19	4.32	4.35
artificial void with 3 millimeters	92.13	92.16	3.75	3.73	4.12	4.17

training process, if adding or removing neurons is necessary. The control mechanism of the architecture will not change, if the neurons are added or removed, but the storages of memory units will be altered. The architecture of BP ANN in this paper can give a support to further researches on the other ANNs.

ACKNOWLEDGMENT

This work was supported by the Research Project of "SDUST Spring Bud" under contract number 2008BW2046 from Shandong University of Science and Technology, China.

REFERENCES

[1] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization," IEEE Transactions on neural networks, Vol. 18, NO. 3, May 2007, pp: 880-888.

[2] B. Noory and V. Groza, "A reconfigurable approach to hardware implementation of neural networks," Canadian Conference on Electrical and Computer Engineering, vol.3, 2003, pp: 1861-1864.

[3] J. G. Eldredge and B. L. Hutchings, "Density enhancement of a neural network using fpgas and run-time reconfiguration," IEEE Workshop on FPGAs for CustomComputing Machines, April 1994, pp. 180 -188.

[4] Y. J. Chen and D. Plessis, "Neural network implementation on a FPGA," in Proc. IEEE Africon Conf., 2002, vol. 1, pp. 337–342.

[5] M. Marchesi, G. Orlandi, F. Piazza, and A. Uncini, "Fast neural networks without multipliers," IEEE Trans. Neural Netw., vol. 4, no. 1, Jan. 1993, pp. 53–62.

[6] 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, Sep. 1999, pp. 661–666.

[7] 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, Oct. 2004, pp. 1113–1117.

[8] 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.

[9] 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.

[10] Y. Taright and M. Hubin, "FPGA implementation of a multilayer preceptron neural network using VHDL," Proc. Int. Conf. Signal Process. (ICSP), vol. 2, 1998, pp. 1306–1310.

[11] 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.

[12] J.G.Elredge and B.L. Hutchings, "RRANN: A Hardware Implementation of the Backpropagation Algorithm Using Reconfigurable FPGAs", Proc. IEEE Int. Conf. on Neural Networks", June 1994.

[13] C.E. Cox, W.E. Blanz, "GANGLION- A Fast Field- Programmable Gate Array Implementation of a Connectionist Classifier", Journal of Solid State Circuits, 1992, 27 (3) : 288-299.

[14] V. Jean, B. Patrice, R. Didier, S. Mark, T. Hervé, B. Philippe. "Programmable Active Memories: Reconfigurable Systems Come of Age", IEEE Transactions on VLSI Systems, 1996, 4(1): 56-69.

[15] Z Haiyan, L Xin, Hardware implementation of multilayer feed forward network with intelligent neuron, Journal of Harbin Engineering University, 2006, 27: 40-45.

[16] C. Huang, F. Vahid, "Dynamic coprocessor management for FPGA-enhanced compute platforms", Proceedings of the 2008 international conference on Compilers, architectures and synthesis for embedded systems, ACM, 2008, Pages 71-78

[17] Scott Hauck, Andre Dehon, " Reconfigurable Computing: The Theory and Practice of FPGA-Based Computation", San Fransisco, CA: Morgan Kaufmann, November 02, 2007.

[18] L. Shoushan, "A study on reconfigurable architecture of ultrasonic signal processing system", Ph. D Thesis, Zhejiang University, 2007.

[19] Y. Keji, Study on denoising techniques for ultrasonic signals in wavelet domain based on neural networks, Journal of Zhejiang University (Engineering Science), 2005,39 (6) : 775-779.

[20] 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.

[21] K K Parhi. VLSI Digital Signal Processing Systems: Design and Implementation, Beijing: China machine press, 2004.

[22] L. Jizhong A study on Nondestructive Evaluation of CFRP Porosity and the Development of the System, Ph. D Thesis, Zhejiang University, 2005.