

FPGA implementation of a wavelet neural network with particle swarm optimization learning

Cheng-Jian Lin^{a,*}, Hung-Ming Tsai^b

^a Department of Electrical Engineering, National University of Kaohsiung, Kaohsiung, Taiwan 811, ROC

^b Institute of Networking and Communication Engineering, Chaoyang University of Technology, 168 Gifong E. Rd., Wufong, Taichung County, 413, Taiwan, ROC

Received 14 October 2006; received in revised form 9 February 2007; accepted 22 May 2007

Abstract

This paper introduces implementation of a wavelet neural network (WNN) with learning ability on field programmable gate array (FPGA). A learning algorithm using gradient descent method is not easy to implement in an electronic circuit and has local minimum. A more suitable method is the particle swarm optimization (PSO) that is a population-based optimization algorithm. The PSO is similar to the GA, but it has no evolution operators such as crossover and mutation. In the approximation of a nonlinear activation function, we use a Taylor series and a look-up table (LUT) to achieve a more accurate approximation. The results of the two experiments demonstrate the successful hardware implementation of the wavelet neural networks with the PSO algorithm using FPGA. From the results of the experiment, it can be seen that the performance of the PSO is better than that of the simultaneous perturbation algorithm at sufficient particle sizes.

© 2007 Elsevier Ltd. All rights reserved.

Keywords: Wavelet neural networks (WNN); Field programmable gate array (FPGA); Particle swarm optimization (PSO); Prediction; Identification

1. Introduction

In recent years, artificial neural networks (ANN) have been widely used in many fields, such as chemistry [1], diagnosis [2], control [3], and image processing [4]. The most commonly used artificial neural network is the multi-layer perceptron (MLP) proposed by Gibson et al. [5]. The MLP is a full connection structure that uses sigmoid functions as hidden node functions. MLP equipped with global transfer functions active over a large range of input values has been widely used to perform global approximations [7].

A suitable approach to overcome the disadvantages of global approximation networks is the substitution of the global activation function with localized basis functions. In this type of local network, only a small subset of the network parameters is engaged at each point in the input space. The network adaptation at a region of the input space does not corrupt the network parameters in more distant regions. Wavelet neural networks (WNN) are examples of spatially localized networks that have been applied to many fields.

* Corresponding author.

E-mail address: cjlin@nuk.edu.tw (C.-J. Lin).

Recently, many researchers have proposed WNN for use in various applications [8–12]. Ikonopoulou and Endou [8] proposed a wavelet neural network that has the analytical ability of the discrete wavelet decomposition with the computational power of radial basis function networks. Members of a wavelet family were chosen through a statistical selection criterion that constructs the structure of the network. Ho et al. [9] used the orthogonal least squares (OLS) algorithm to purify the wavelets from their candidates, which is avoided using more wavelets than required and often resulted in an over-fitting of the data and a poor situation in [10]. Lin et al. [11] proposed a wavelet neural network to control the moving table of a linear ultrasonic motor (LUSM) drive system. They chose an initialization for the mother wavelet based on the input domains defined by training sequence examples. Huang and Huang [12] proposed an evolutionary algorithm for optimally adjusted wavelet networks.

WNNs train their parameters iteratively using a learning algorithm. The common training method used is the gradient descent (GD) method. The gradient descent algorithm including the Least Mean Squares (LMS) algorithm and back-propagation [6,13] for neural networks, are not suitable because they are likely to converge in the local minimum. Thus, this study also introduces a novel algorithm called particle swarm optimization (PSO) to achieve global optimum capability. The PSO algorithm is an evolutionary computation technique developed by Kennedy and Eberhart in 1995 [14]. The underlying motivation for the development of the PSO algorithm was the social behavior of animals, such as bird flocking, fish schooling, and the swarm theory. The PSO is similar to the GA. The PSO begins with a random population and searches for an optimum by updating the population. The advantages of the PSO are that it has no evolution operators such as crossover and mutation and it does not need to adjust many free parameters. Moreover, each potential solution is also assigned a randomized velocity. The potential solutions, called particles, are then “flown” through the problem space. Compared with the GA, the PSO has some attractive characteristics. First, the PSO has memory. A knowledge of good solutions in the PSO is retained by all particles whereas in the GA, a previous knowledge of the problem is destroyed when a population is changed. Second, the PSO has constructive cooperation between particles. The particles in a swarm share the information among themselves. There have been successful applications of the PSO for several optimization problems, such as for control problems [15–17] and feedforward neural network design [18,19].

Digital integrated circuits in the form of field programmable gate arrays (FPGA) [20–25] make the hardware designing process flexible and programmable. Recently, FPGAs have been used in many applications because the programmable logic element enables to increase the logic capability, the velocity, and the amount of memory, and the number of functions. In addition, the usage of very high speed integrated circuit hardware description language (VHDL) results in easy and fast compilation in complicated circuits. Hence VHDL has lots of benefits, such as high capacity, speedy, duplicate designs, and low cost. Many of the literature has proposed hardware implementation of neural networks, but it does not have learning ability [20–22]. Some researchers have proposed hardware implementation of neural networks with on-chip learning that uses the BP algorithm [25].

Since the wavelet function is a nonlinear activation function, it is not easy to implement using the hardware. A look-up table (LUT) has been traditionally used for implementing the nonlinear activation function in which the amount of hardware required could be large and the degree of approximation is not accurate enough. Another approach proposed the use of piecewise-linear functions, but the resulting activation function suffers from nondifferentiability at the intersections between the different linear functions. In this paper, the nonlinear activation function is approximated to accomplish a more accurate approximation using the Taylor series and LUT.

In Sections 2 and 3, the structure of a WNN and its PSO learning algorithm is described. Section 4 introduces the hardware implementation of a WNN using FPGA. In Section 5, two experimental results are presented. The conclusions and future works are given in the last section.

2. The wavelet neural network (WNN)

The structure of a WNN was already analyzed using our methodology in [26]. The structure of the WNN model is shown in Fig. 1. The input data in the input layer of the network is $x = [x_1, x_2, \dots, x_i, \dots, x_n]$, where n is the number of dimensions. A wavelet $\phi_{d,t}$ is derived from its mother wavelet $\phi(x)$ by

$$\phi_{d,t} = \phi \left(\frac{x-t}{d} \right). \quad (1)$$

It is the first derivative of the Gaussian function and is expressed as follows:

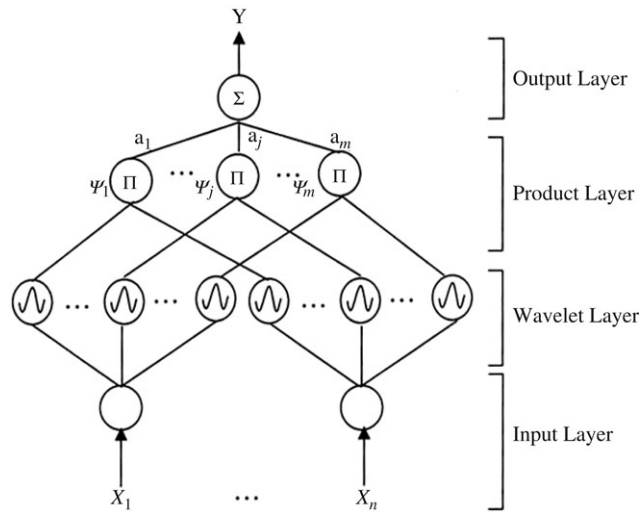


Fig. 1. The architecture of the WNN model.

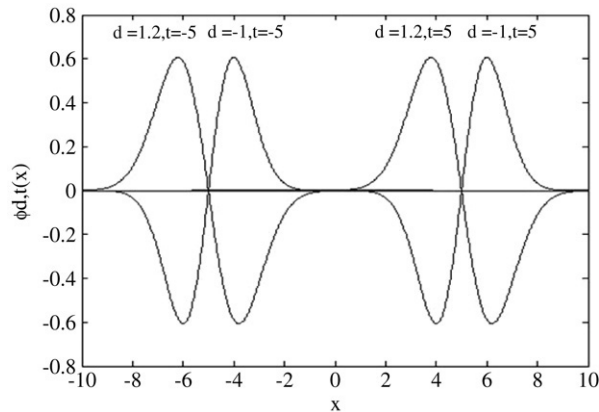


Fig. 2. Wavelet bases with different translations and dilations.

$$\phi(x) = -xe^{-\|x\|^2/2}, \tag{2}$$

where $\|x\|^2 = x^T x$. It may be regarded as a differentiable version of the Haar mother wavelet, just as the sigmoid, which is a differentiable version of a step function, and it has the universal approximation property. Therefore, the activation function of the j th wavelet node connected with the i th input data is represented as:

$$\phi_{d_{ij},t_{ij}}(x_i) = -\left(\frac{x_i - t_{ij}}{d_{ij}}\right) e^{-\| \frac{x_i - t_{ij}}{d_{ij}} \|^2/2}, \tag{3}$$

$i = 1, \dots, n \quad j = 1, \dots, m,$

where n is the number of input dimensions and m is the number of wavelets. The wavelet functions of (3) with various dilations and translations are shown in Fig. 2.

Each wavelet in the product layer is labeled as Π , i.e., the product of the j th multi-dimensional wavelet with n input dimensions $x = [x_1, x_2, \dots, x_i, \dots, x_n]$ can be defined as

$$\psi_j(x) = \prod_{i=1}^n \phi_{d_{ij},t_{ij}}(x_i). \tag{4}$$

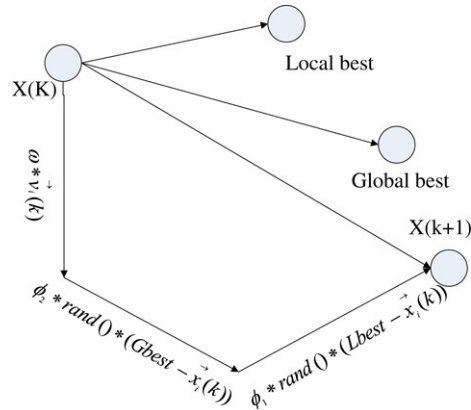


Fig. 3. The diagram of the updated velocity in the PSO.

According to the theory of multi-resolution analysis (MRA), $f \in L^2(\mathfrak{R})$ can be regarded as a linear combination of wavelets at different resolution levels. For this reason, the function f is expressed as

$$Y(x) = f(x) \approx \sum_{j=1}^m a_j \psi_j(x). \tag{5}$$

If $\psi_j = [\psi_{j1}, \psi_{j2}, \dots, \psi_{jm}]$ is used as a nonlinear transformation function of hidden nodes and weight vectors and $a_j = [a_{j1}, a_{j2}, \dots, a_{jm}]$ defines the connection weights, then Eq. (5) can be considered as the functional expression of the WNN modeling function Y .

3. The particle swarm optimization (PSO)

Particle swarm optimization (PSO) is a recently invented high performance optimizer that possesses several highly desirable attributes, including the fact that the basic algorithm is very easy to understand and to implement. It is similar to genetic algorithms and evolutionary algorithms, but requires less computational memory and fewer lines of code. The PSO conducts search using a population of particles which correspond to individuals in GA. Each particle has a velocity vector \vec{v}_i and a position vector \vec{x}_i to represent a possible solution.

Consider an optimization problem that requires the simultaneous optimization of variables. A collection or swarm of particles are defined, where each particle is assigned a random position in the N -dimensional problem space so that each particle's position corresponds to a candidate solution to the optimization problem. Then the particles fly rapidly over and search the space with the moving velocity of each particle. The PSO has a simple rule. Each particle has three choices in evolution: (1) Insist on oneself. (2) Move towards the optimum itself at present. Each particle remembers its own best personal position that it has ever found, called the local best. (3) Move towards the best the population has met. Each particle also knows the best position found by any particle in the swarm, called the global best. The PSO reaches a balance among these three choices.

At each time step, each of these particle positions is scored to obtain a fitness value based on how well it solves the problem. Using the local best position ($Lbest$) and the global best position ($Gbest$), a new velocity for each particle is updated by

$$\vec{v}_i(k+1) = \omega * \vec{v}_i(k) + \phi_1 * rand() * (Lbest - \vec{x}_i(k)) + \phi_2 * rand() * (Gbest - \vec{x}_i(k)), \tag{6}$$

where ω , ϕ_1 and ϕ_2 are called the coefficient of inertia, cognitive and society, respectively. The $rand()$ is uniformly distributed random numbers in $[0, 1]$. The term \vec{v}_i is limited to the range $\pm \vec{v}_{max}$. If the velocity violates this limit, it will be set at its proper limit. The concept of the updated velocity is illustrated in Fig. 3.

Changing velocity enables every particle to search around its individual best position and global best position. Based on the updated velocities, each particle changes its position according to the following:

$$\vec{x}_i(k+1) = \vec{x}_i(k) + \vec{v}_i(k+1). \tag{7}$$

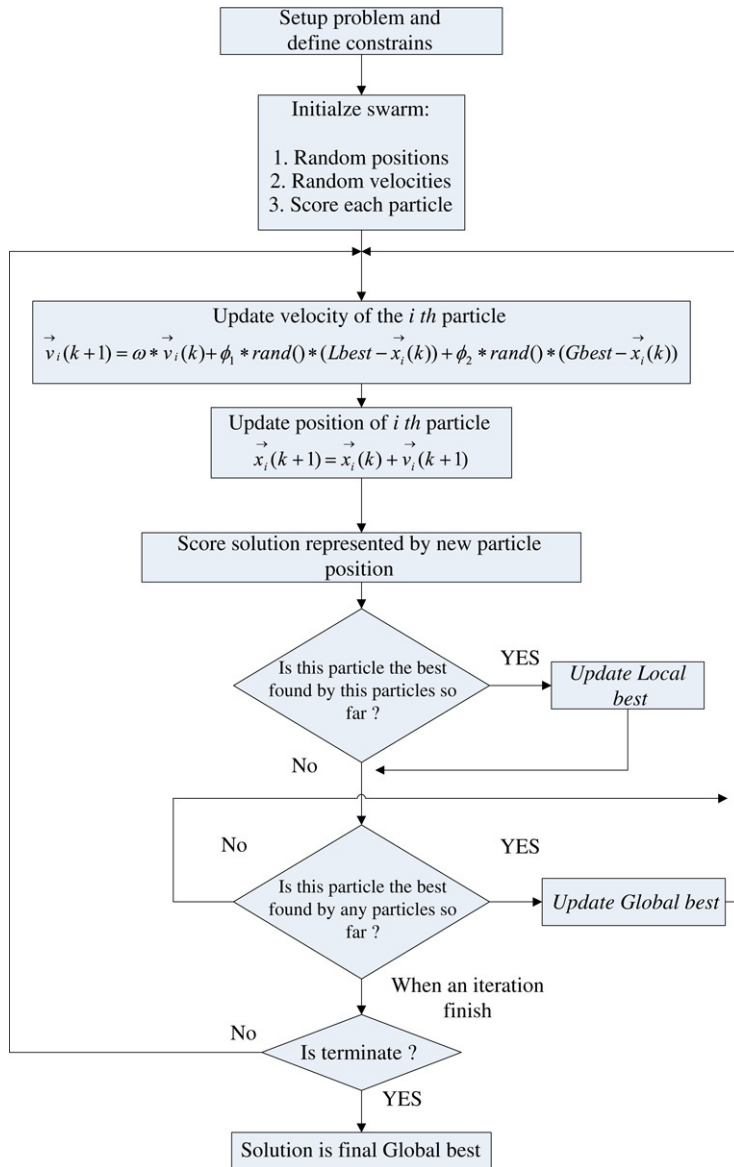


Fig. 4. PSO typical flowchart illustrates the steps and update equations.

When every particle is updated, the fitness value of each particle is calculated again. If the fitness value of the new particle is higher than those of local best, then the local best will be replaced with the new particle. If the fitness value of the new particle is higher than those of global best, then the global best will be also replaced with the new particle. The algorithm repeats the above updating process step by step, the whole population evolves toward the optimum solution. The detailed flowchart is shown in Fig. 4.

4. Hardware implementation

In this section, the study will introduce the hardware implementation of the WNN structure and its learning algorithm. The overall component of the WNN model is shown in Fig. 5. The hardware implementation of the WNN model includes two kinds of units: a wavelet unit and a learning unit.

4.1. Wavelet unit

As shown in Fig. 1, the main part of the WNN structure is the wavelet layer. The input layer can directly transmit a value to the wavelet layer. The process does not pass through any operation. The product layer will be multiplied by

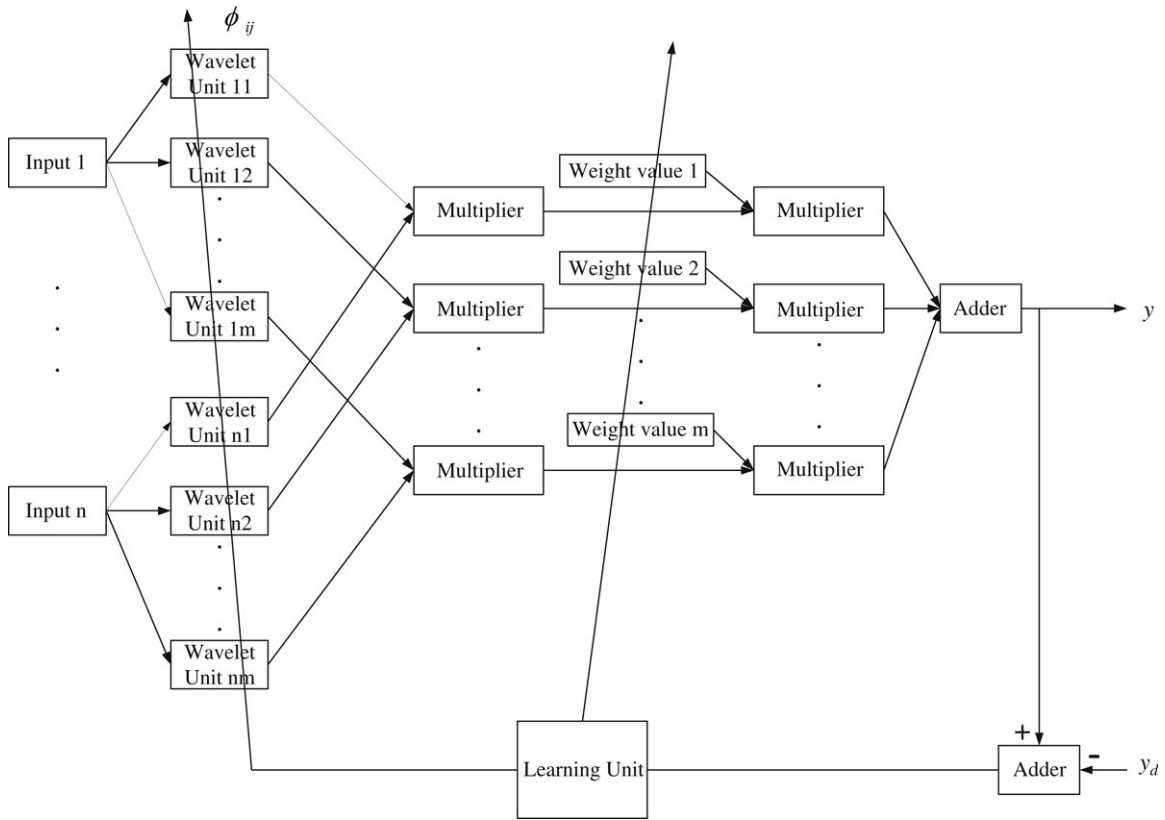


Fig. 5. The overall component of the WNN model.

the output of the wavelet layer individually. The output layer only needs to add up all the outputs of the product layer. The wavelet layer is used to perform nonlinear transformation mainly, and the wavelet function is used as the nonlinear activation function in this layer. Because the wavelet function is very complicated and cannot be easily used to perform the hardware implementation directly, the study uses the Taylor series and LUT to approximate the wavelet function.

In Eq. (8), e^{-x} is expressed by the Taylor series as follows:

$$e^{-x} = 1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \frac{x^4}{4!} - \dots + \frac{x^m}{m!} - \dots, \tag{8}$$

where m is the number of order and x is the input value. A large m leads to a more accurate approximation, but more logic gates and bits are needed. In this paper, the study choose $m = 4$ to implement the hardware for e^{-x} . Fig. 6 shows a block diagram of the hardware implementation of the Euler function e^{-x} . The implementation uses three multipliers, three dividers, an adder, an LUT, and a multiplexer. The adder adds up the outputs of every order. Some values in the Taylor series are not accurate, such as the input, which is a large negative value, so it needs to compensate for it using the LUT. Therefore, when input is a large negative value, the multiplexer chooses the values in the LUT automatically.

The overall component of the wavelet layer is shown in Fig. 7. The wavelet layer consists of a divider, an adder, four multiplier, and the Taylor series e^{-x} . Fig. 8 shows a comparison of the wavelet function and the wavelet function implemented using the Taylor series and the look-up table. The error curve of the reference function (wavelet) and the approximated function is shown in Fig. 9. From Figs. 8 and 9, the implementation can approximate the wavelet function more accurately using the Taylor series and LUT.

4.2. Learning unit

A block diagram of the hardware implementation of the PSO algorithm can be seen in Fig. 10. The design is divided into four main blocks: an evaluation fitness block, a comparator block, an update block, and a control block.

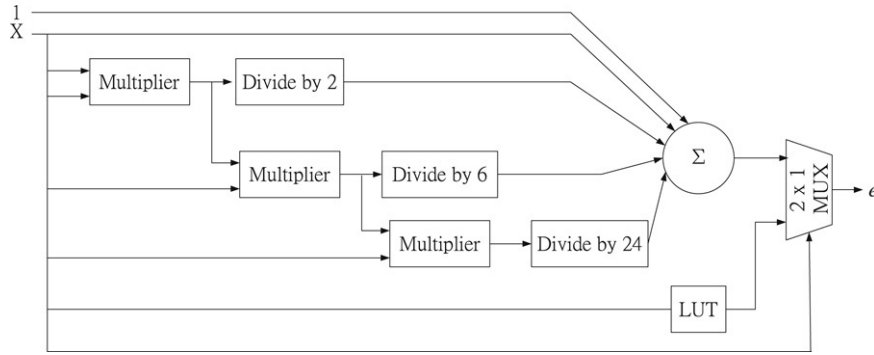


Fig. 6. The block diagram of the hardware implementations of the Euler function.

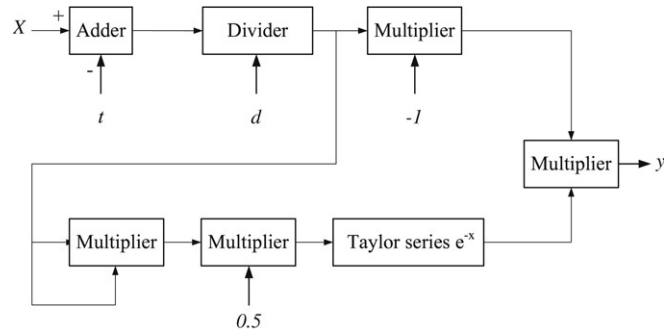


Fig. 7. The overall component of the wavelet layer using the first derivative of the Gaussian function.

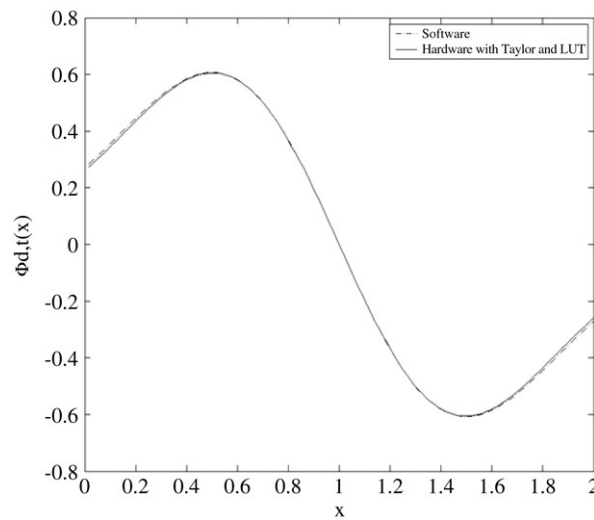


Fig. 8. The wavelet function using the first derivative of the Gaussian function and its hardware approximation with Taylor and LUT.

The evaluation fitness block calculates the mean square error (MSE) value. The comparator block compares the fitness value to find the best fitness value. The parameters of each particle in a swarm are updated through the update block. The control block manages the counter and generates the enable signal. Because the PSO algorithm needs to record the optimum solution, the memory device is necessary. The implementation uses a RAM as memory device in this study.

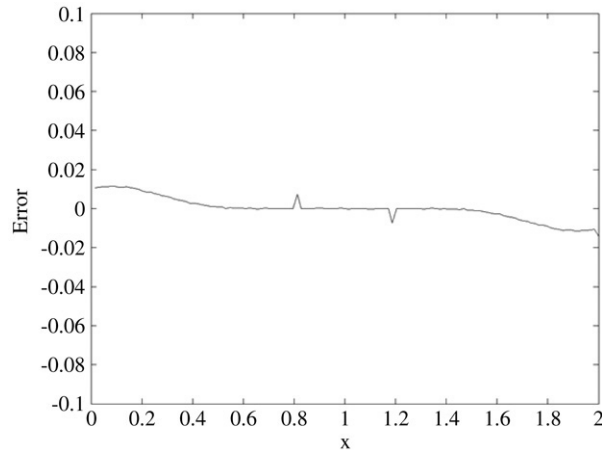


Fig. 9. The error curve of the reference function (the first derivative of the Gaussian function) and approximated function.

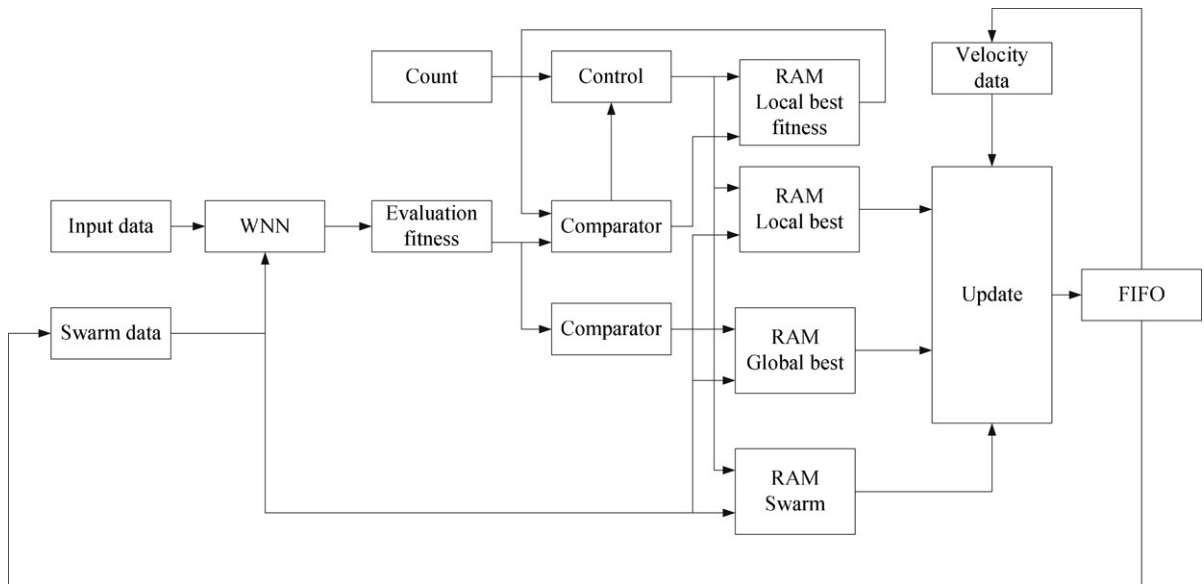


Fig. 10. The block diagram of the hardware implementation for PSO algorithm.

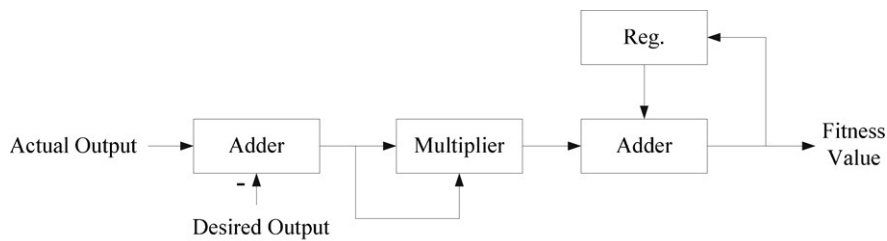


Fig. 11. The evaluation fitness block.

• *Evaluation fitness block*

The evaluation fitness block calculates the cost function in order to evaluate the performance. The block diagram is shown in Fig. 11. The error value between the actual output and the desired output is calculated by using subtraction and is followed by a square evaluator. Then, the value is accumulated until all the input patterns are calculated.

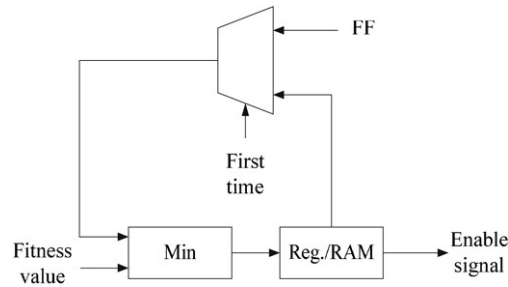


Fig. 12. The comparator block.

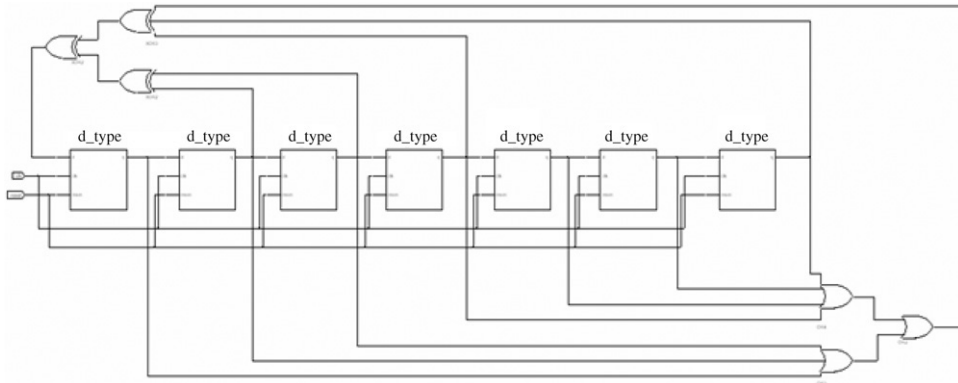


Fig. 13. Linear feedback shifts register.

- *Comparator block*

The comparator block diagram is shown in Fig. 12. The block begins with a multiplexer to account for the initial state, where the fitness value is fixed to FFh. Next, the evaluation of a value with a previous best value is compared: If the current fitness value $<$ *best* and *best* = current fitness value than store the *best* in register or RAM. Then, the comparator block delivers an enable signal to the RAM and stores the current position in D -dimensional hyperspace.

- *Update block*

Because the *rand()* function is uniformly distributed random numbers in $[0, 1]$, it is not implemented with hardware circuit directly. The random number generator uses a linear feedback shift register (LFSR). The linear feedback shift register counts through $2^n - 1$ states, where n is the number of bits. The linear feedback shift register state with all bits equal to 1 is an illegal state which never occurs in normal operations. It takes a few clock impulses and watches the output values of the register, which should appear more or less randomly. Depending on the logic used in the feedback path, the register follows a predefined sequence of states. Fig. 13 shows the linear feedback shift register as a random number generator.

The update block diagram is shown in Fig. 14. The velocity data is changed by using Eq. (6), and the swarm data is moved by using Eq. (7). As can be seen in Fig. 14, the update can use some simple components to implement such as an adder and a multiplier. The linear feedback shift register with 12 bits is used as a random number generator.

The hardware implementation of the PSO uses asynchronous updates, where the swarm and velocity data are updated after each particle with D variables have been calculated according to Eqs. (6) and (7), rather than waiting until all the particles are updated. This seems to decrease the cost over synchronous updates, although asynchronous updates are not as beneficial to parallel processing as synchronous updates.

- *Control block*

The control block has the following function:

- Manages the counter, providing the necessary address for the RAM.
- Determines the RAM to read or write.
- Generates the necessary enabling signal to make the RAM work.

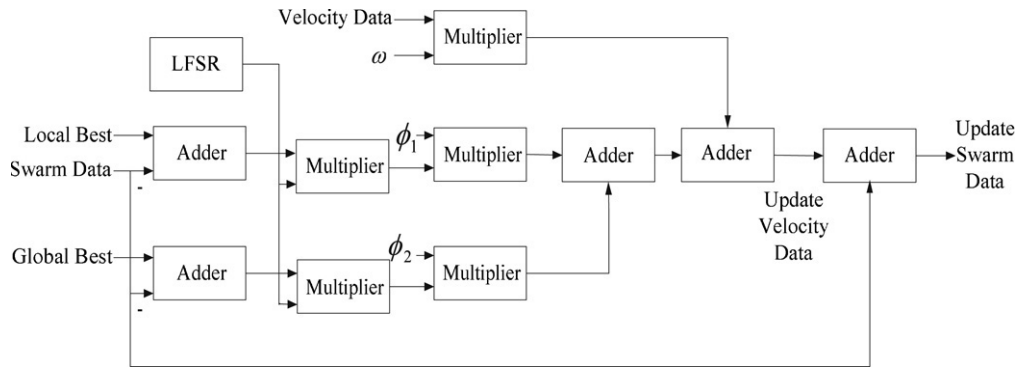


Fig. 14. The update block.

Table 1
Translation and dilation values after SCA method for prediction problem

Cluster	t_{1j}	d_{1j}
1	0.266758	0.133941
2	0.667020	0.120197
3	0.867452	0.123739
4	0.551097	0.056612

5. Simulation results

In this section, the study discusses the implementation of the WNN using PSO to solve two problems. The first problem is the problem of predicting a chaotic signal, and the second problem is the identification of a dynamic system. The hardware implementation uses the CLB (Configurable Logic Block) and the connecting wires to design the function in FPGA. The software utilized ISE 6.2i with Xilinx Virtex II-xc2v8000 chip and MATLAB 6.1 for these examples. Each problem was simulated on a Pentium IV 3.2 GHz desktop computer with 1 GB RAM.

Example 1: Prediction of a chaotic signal

In this example, the implemented WNN model was used to predict a chaotic signal. The classical time series prediction problem is a one-step-ahead prediction problem that is described in [5]. The following equations describe the logistic function:

$$x(k + 1) = ax(k)(1 - x(k)). \tag{9}$$

The behavior of the time series generated by this equation depends critically upon the value of the parameter a . If $a < 1$, the system has a single fixed point at the origin, and from a random initial value between $[0, 1]$, the time series collapses to a constant value. For $a > 3$, the system generates a periodic attractor. Beyond the value $a = 3.6$, the system becomes chaotic. In this study, it set the parameter value a to 3.8. The first 60 pairs (from $x(1)$ to $x(60)$), with the initial value $x(1) = 0.001$, were the training data set, while the remaining 100 pairs (from $x(1)$ to $x(100)$), with initial value $x(1) = 0.9$, were the testing data set used for validating the proposed method.

The number of nodes in the wavelet layer was determined in advance by using the self-clustering algorithm (SCA) [27]. The number of nodes in each layer is two, four, four, and one. Table 1 shows that the initial translations and dilations of the wavelet layer before learning was determined by the SCA method, where t_{ij} and d_{ij} represent translation and dilation with the i th input dimension and j th cluster. The initial weights of the output layer were random in the interval $[0, 1]$. The coefficient ω was set to 0.5. The coefficient of cognitive ϕ_1 and society study ϕ_2 were set to 2. The different particle sizes were five, ten, twenty, forty, and fifty. The wavelet function used the first derivative of the Gaussian function. The average error with different particle size is shown in Fig. 15. The large particle performed better, but required more cost. Thus, the study selected a suitable particle size at ten to implement the hardware after trade off. Learning was carried out 1000 times. Fig. 16(a) shows the prediction results of the desired output and the WNN model output of the software implementation with the different learning algorithms. The “o” represents the desired output of the time series, the notation “*” represents the WNN model output of the software

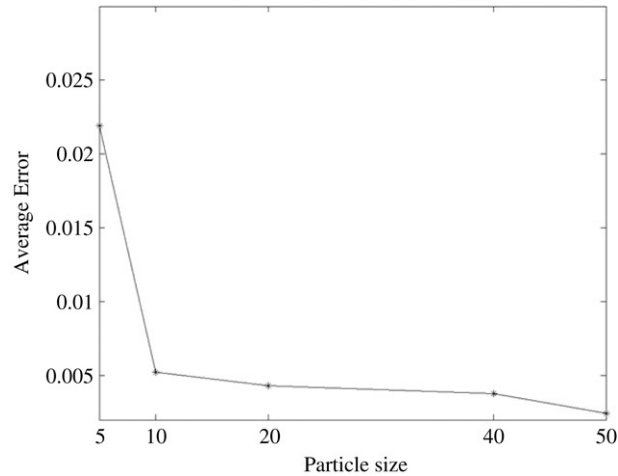


Fig. 15. The average error with the different particle size.

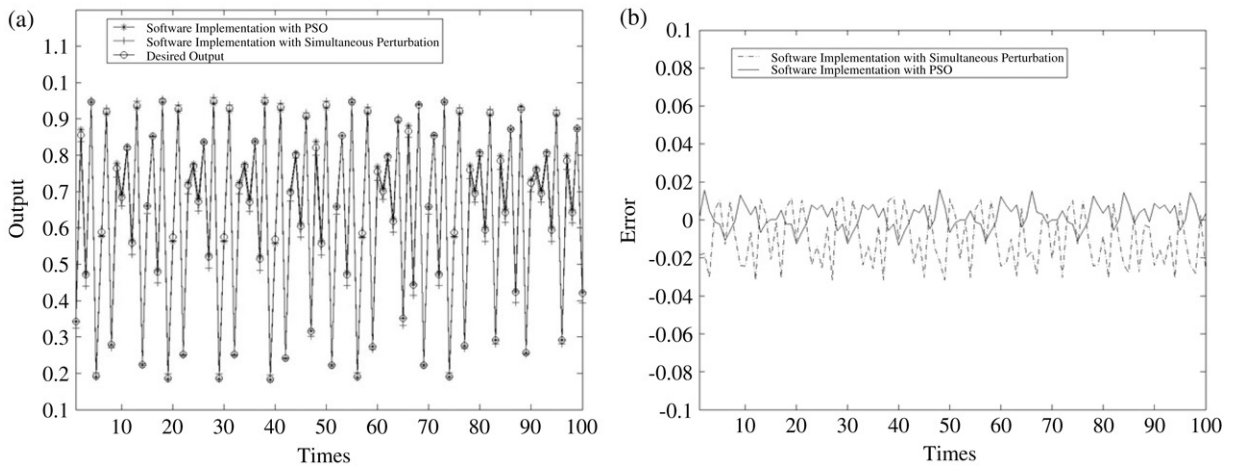


Fig. 16. (a). The prediction results of the desired output and the WNN model output of hardware implementation with the different learning algorithm. (b). The prediction errors of the WNN model output of hardware implementation with the different learning algorithm.

implementation with PSO, and the notation “+” represents the WNN model output of the software implementation with simultaneous perturbation. The prediction errors of the WNN model output of the software implementation with the different learning algorithms are shown in Fig. 16(b). When the sufficient particle size is used, the performance of the PSO was better than that of the simultaneous perturbation algorithm. Fig. 17(a) shows the prediction results of the desired output and the WNN model output of the hardware implementation and (b) displays the prediction errors of the WNN model output of the hardware implementation. From the experimental results, the PSO algorithm can solve prediction problems and the performance of the hardware implementation approximates that of the software implementation. Fig. 18 shows the hardware implementation of the WNN with the PSO algorithm.

Example 2: Identification of the dynamic system

In this example, the proposed WNN model was used to identify a dynamic system. The identification model has the form

$$\hat{y}(k + 1) = \hat{f}[u(k), u(k + 1), \dots, u(k - p + 1)y(k), y(k + 1), \dots, y(k - q + 1)]. \tag{10}$$

Since both the unknown plant and WNN are driven by the same input, the WNN adjusts itself with the goal of causing the output of the identification model to match that of the unknown plant. Upon convergence, their input–output

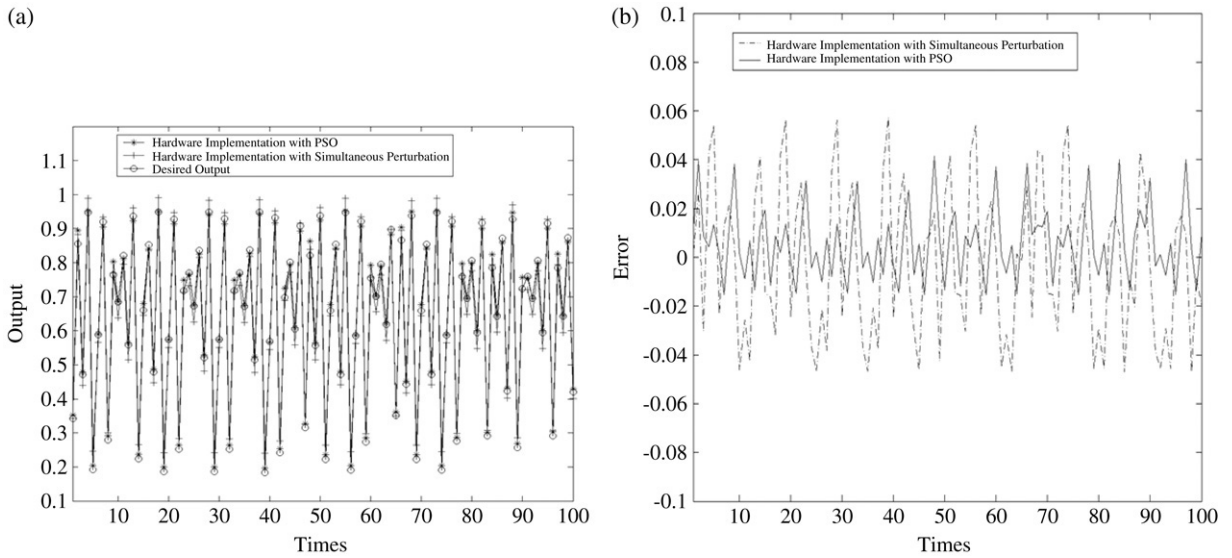


Fig. 17. (a). The prediction results of the desired output and the WNN model output of hardware implementation. (b). The prediction errors of the WNN model output of hardware implementation.

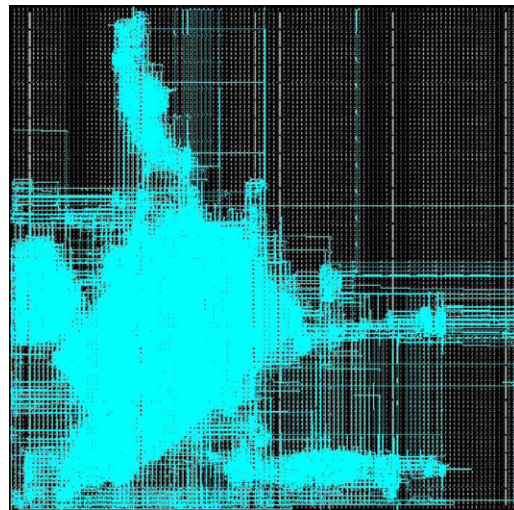


Fig. 18. The hardware implementation of the WNN with the PSO algorithm for prediction problem.

relationship should match. The plant to be identified is guided by the difference equation

$$y(k + 1) = \frac{y(k)}{1 + y^2(k)} + u^3(k). \tag{11}$$

The output of the plant depends nonlinearly on both its past values and inputs, but the effects of the input and output values are additive. When the WNN was applied to this identification problem, the coefficients $\omega = 0.5$, $\phi_1 = 2$, $\phi_2 = 2$ were chosen, The 100 training input patterns were generated from $u(k) = \cos(2\pi k/100)$, and the training process was continued for 1000 time steps. The SCA algorithm was used to decide the structure of the nodes and its parameters value. The number of nodes in each layer was one, four, four, and one. The translation and dilation values are shown in Table 2. The initial weights were the same method as in Example 1. Fig. 19(a) shows the outputs of the plant and the identification of the software implementation. In this figure, the output of the WNN model are presented as *'s while desired output values are presented as o's. The error of the plant and the identification of the

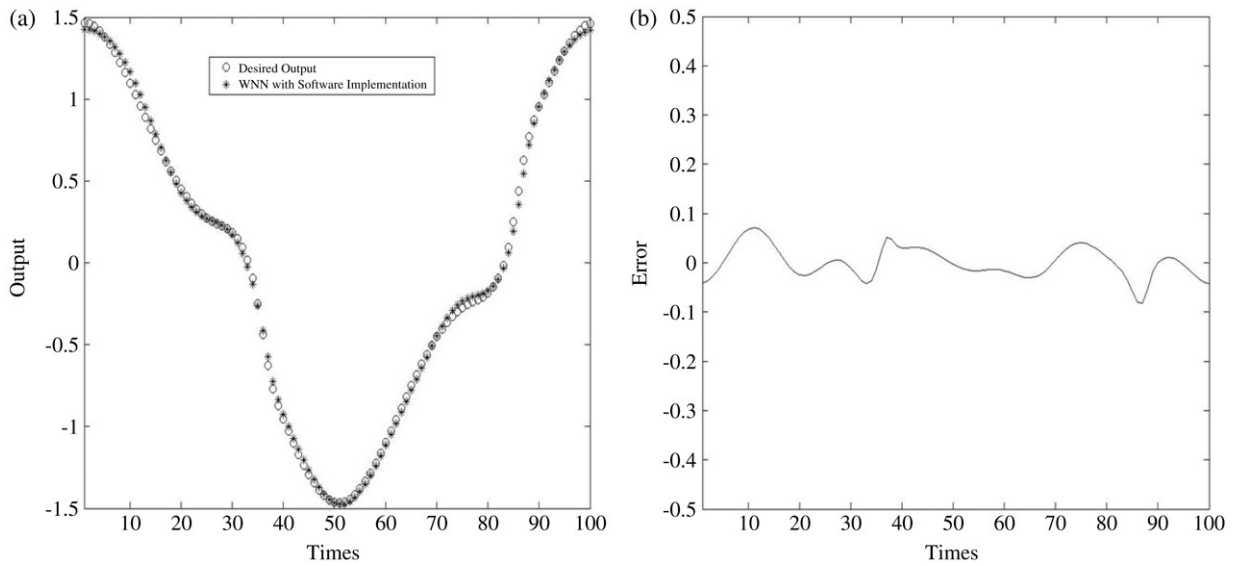


Fig. 19. (a). The outputs of the plant and the identification of software implementation. (b). The error of the plant and the identification of software implementation.

Table 2
The translations and dilations of wavelet layer by SCA method for identification problem

Cluster	t_{1j}	t_{2j}	d_{1j}	d_{2j}
1	0.259127	0.522199	0.097602	0.034716
2	0.366056	0.371130	0.087997	0.057849
3	0.767271	0.505619	0.064074	0.049479
4	0.997669	0.981260	0.040452	0.066314

Table 3
The RMS errors of the prediction and identification examples using the software and hardware implementations

	Software implementation	Hardware implementation
RMS error of prediction example	0.000673	0.00161
RMS error of identification example	0.003155	0.00654

software implementation are shown in Fig. 19(b). The results show that the identification capability of the WNN model is good. Fig. 20(a) shows the outputs of the plant and the identification of hardware implementation and the error between the plant and the identification of hardware implementation is shown in Fig. 20(b). The performance of the WNN model output of the hardware implementation compared with the performance of the WNN model output of the software implementation is discussed here. From Figs. 19(b) and 20(b), the difference between the WNN model output of the hardware implementation and the WNN model output of the software implementation is small. The experimental results demonstrate the identification capability of the hardware implementation of the WNN model. Fig. 21 shows the hardware implementation of the WNN with the PSO algorithm. In order to give a better view of the performance, the root-mean-square (RMS) errors of the prediction and identification examples using the software and hardware implementations are shown in Table 3.

6. Conclusion

In this paper, implementation of wavelet neural networks with learning ability using FPGA was proposed. PSO is used to find an optimum solution and only some simple operations are required to adjust parameters. The results of two experiments demonstrate the successful hardware implementation of the wavelet neural networks with the

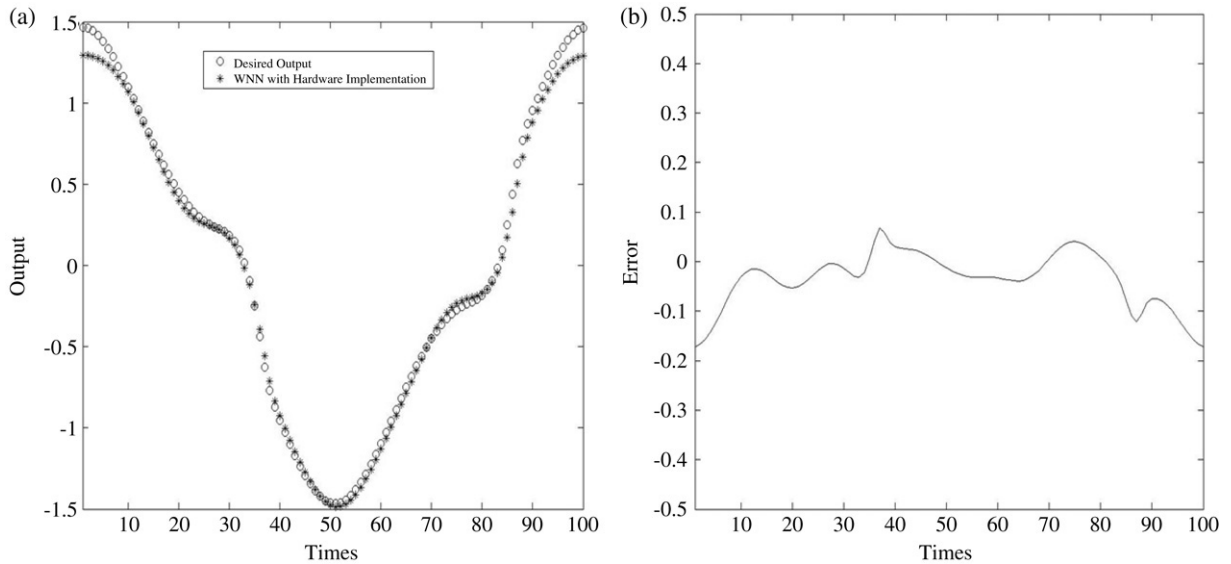


Fig. 20. (a). The outputs of the plant and the identification of hardware implementation. (b). The error of the plant and the identification of hardware implementation.

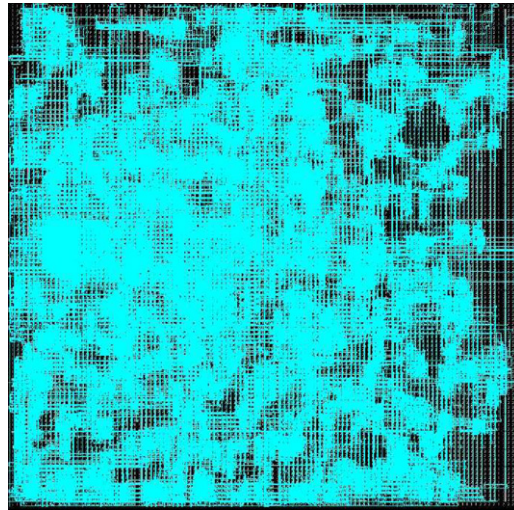


Fig. 21. The hardware implementation of the WNN with the PSO algorithm for identification problem.

PSO algorithm using FPGA. Some of the features of the wavelet neural networks with the PSO algorithm can be summarized as follows: (1) an analog wavelet neural network is realized based on digital circuits; (2) hardware implementation of the PSO learning rule is relatively easy; and (3) hardware implementation can take advantage of parallelism. From the results of the experiment with the prediction problem, it can be seen that the performance of the PSO is better than that of the simultaneous perturbation algorithm at sufficient particle sizes.

A major drawback of the existing wavelet neural networks is that their application domain is limited to static problems due to their inherent feedforward network structure. In the future, a recurrent wavelet neural network will be proposed for solving identification and control problems.

Acknowledgement

This work was supported in part by the National Science Council, Taiwan, ROC, under Grant NSC 95-2221-E-324-028-MY2.

References

- [1] G. Mealing, M. Bani-Yaghoub, R. Tremblay, R. Monette, J. Mielke, R. Voicu, C. Py, R. Barjovanu, K. Faid, Application of polymer microstructures with controlled surface chemistries as a platform for creating and interfacing with synthetic neural networks, in: *Proceedings. IEEE International Joint Conference on Neural Networks*, vol. 5, 31 July–4 Aug. 2005, pp. 3116–3120.
- [2] S.R. Bhatikar, R.L. Mahajan, Artificial neural-network-based diagnosis of CVD barrel reactor, *IEEE Transactions on Semiconductor Manufacturing* 15 (1) (2002) 71–78.
- [3] M.C. Choy, D. Srinivasan, R.L. Cheu, Neural networks for continuous online learning and control, *IEEE Transactions on Neural Networks* 17 (6) (2006) 1511–1531.
- [4] M. Markou, S. Singh, A neural network-based novelty detector for image sequence analysis, *IEEE Transactions on Pattern Analysis and Machine Intelligence* 28 (10) (2006) 1664–1677.
- [5] G.J. Gibson, S. Siu, C.F.N. Cowan, Application of multilayer perceptrons as adaptive channel equalizers, in: *Proc. IEEE Conf. Acoust. Speech Signal Process.*, 1989, pp. 1183–1186.
- [6] D.E. Rumelhart, G.E. Hinton, R.J. Williams, Learning internal representation by error propagation, in: D.E. Rumelhart, J.L. McClelland (Eds.), *Parallel/Distributed Processing Exploration in yth Microstructure of Cognition*, MIT Press, Cambridge, MA, 1986, pp. 318–362.
- [7] K. Hornik, Multilayer feedforward networks are universal approximators, *Neural Networks* 2 (1989) 359–366.
- [8] A. Ikonopoulou, A. Endou, Wavelet decomposition and radial basis function networks for system monitoring, *IEEE Transactions on Nuclear Science* 45 (5) (1998) 2293–2301.
- [9] D.W.C. Ho, P.A. Zhang, J. Xu, Fuzzy wavelet networks for function learning, *IEEE Transactions on Fuzzy Systems* 9 (1) (2001) 200–211.
- [10] Q. Zhang, Using wavelet networks in nonparametric estimation, *IEEE Transactions on Neural Networks* 8 (1998) 227–236.
- [11] F.J. Lin, R.J. Wai, M.P. Chen, Wavelet neural network control for linear ultrasonic motor drive via adaptive sliding-mode technique, *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control* 50 (6) (2003) 686–698.
- [12] Y.C. Huang, C.M. Huang, Evolving wavelet networks for power transformer condition monitoring, *IEEE Transactions on Power Delivery* 17 (2) (2002) 412–416.
- [13] F.J. Lin, C.H. Lin, P.H. Shen, Self-constructing fuzzy neural network speed controller for permanent-magnet synchronous motor drive, *IEEE Transactions on Fuzzy Systems* 9 (5) (2001) 751–759.
- [14] J. Kennedy, R. Eberhart, Particle swarm optimization, in: *Proc. IEEE Int. Conf. on Neural Networks*, 1995, pp. 1942–1948.
- [15] Z.L. Gaing, A particle swarm optimization approach for optimum design of PID controller in AVR system, *IEEE Transactions on Energy Conversion* 19 (2) (2004) 384–391.
- [16] H. Yoshida, K. Kawata, Y. Fukuyama, S. Takayama, Y. Nakanishi, A particle swarm optimization for reactive power and voltage control considering voltage security assessment, *IEEE Transactions on Power Systems* 15 (4) (2000) 1232–1239.
- [17] M.A. Abido, Optimal design of power-system stabilizers using particle swarm optimization, *IEEE Transactions on Energy Conversion* 17 (3) (2002) 406–413.
- [18] C.F. Juang, A hybrid of genetic algorithm and particle swarm optimization for recurrent network design, *IEEE Transactions on Systems, Man and Cybernetics, Part B* 34 (2) (2004) 997–1006.
- [19] R. Mendes, P. Cortez, M. Rocha, J. Neves, Particle swarms for feedforward neural network training, in: *The 2002 International Joint Conference on Neural Networks*, 2002, pp. 1895–1899.
- [20] N.M. Botros, M. Abdul-Aziz, Hardware implementation of an artificial neural network using field programmable gate arrays (FPGA's), *IEEE Transactions on Industrial Electronics* 41 (6) (1994) 665–667.
- [21] J.J. Blake, L.P. Maguire, T.M. McGinnity, B. Roche, L.J. McDaid, The implementation of fuzzy systems, neural networks and fuzzy neural networks using FPGAs, *Information Sciences* 112 (1–4) (1998) 151–168.
- [22] H. Faiedh, Z. Gafsi, K. Torki, K. Besbes, Digital hardware implementation of a neural network used for classification, in: *The 16th International Conference on Microelectronics*, Dec. 2004, pp. 551–554.
- [23] C.T. Yen, W.D. Wang, Y.T. Lin, FPGA realization of a neural-network-based nonlinear channel equalizer, *IEEE Transactions on Industrial Electronics* 51 (2) (2004) 472–479.
- [24] Y. Maeda, T. Tada, FPGA implementation of a pulse density neural network with learning ability using simultaneous perturbation, *IEEE Transaction on Neural Networks* 14 (3) (2003) 688–695.
- [25] H. Hikawa, A digital hardware pulse-mode neuron with piecewise linear activation function, *IEEE Transactions on Neural Networks* 14 (5) (2003) 1028–1037.
- [26] C.J. Lin, C.C. Shih, P.Y. Chen, A nonlinear time-varying channel equalizer using self-organizing wavelet neural networks, in: *IEEE Int'l Joint Conf. on Neural Networks*, Budapest, 25–29 July 2004, pp. 2089–2094.
- [27] C.J. Lin, Ch.Y. Lee, C.H. Chen, A self-adaptive quantum radial basis function network for classification applications, in: *International Joint Conference on Neural Networks*, Budapest, July 2004.