ICIC Express Letters
Volume **11**, Number **3**, March **2017**
 ISSN 1881-803X
pp. 565–574

HYBRID HARDWARE-SOFTWARE ARCHITECTURE FOR NEURAL NETWORKS TRAINED BY IMPROVED PSO ALGORITHM

Tuan Linh Dang[1], Thang Cao[2] and Yukinobu Hoshino[1]

[1]School of Systems Engineering
Kochi University of Technology
Tosayamada, Kami City, Kochi 782-8502, Japan
188001d@gs.kochi-tech.ac.jp; hoshino.yukinobu@kochi-tech.ac.jp

[2]Department of Information Physics and Computing
The University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
cao@hal.ipc.i.u-tokyo.ac.jp

Received September 2016; accepted December 2016

Abstract. *This paper proposes a hybrid hardware-software architecture for an evaluation of a neural network (NN) trained by particle swarm optimization with velocity control (PSO_CV) algorithm. In this design, the NN is implemented in hardware by SystemVerilog language to maintain the testing speed of the system. The PSO_CV algorithm is implemented by NIOS II processor by C language to increase the flexibility and to reduce required resources of the system. Experimental results showed the advantages of the proposed architecture concerning the speed and the required resources. In addition, the NN trained by the PSO_CV algorithm achieved better performance regarding recognition rate and learning error than the NN trained by the standard particle swarm optimization (SPSO) algorithm in our new architecture.*
Keywords: Neural network, Particle swarm optimization, Field programmable gate array, Velocity control, NIOS II, Hybrid hardware-software architecture

1. **Introduction.** An artificial neural network (NN) has recently attracted many researchers. One of the most famous algorithms for training the NN is back-propagation algorithm [1]. However, several studies in this field have mentioned the advantages of particle swarm optimization (PSO) algorithm when compared with the back-propagation algorithm concerning the speed and the accuracy [2, 3, 4]. The PSO algorithm is based on the social behaviors such as birds flocking. At any given iteration, a bird in the swarm tends to move to an optimal location based on the knowledge of this bird, and the knowledge of the whole population [5, 6, 7]. A software implementation of the NN trained by the PSO algorithm (NN-PSO) has been investigated [8, 9, 10].

Nowadays, field-programmable gate array (FPGA) has become an attractive target of research. Because of the parallelism, an FPGA program may obtain a higher operating speed than the software-only approach [11]. The NN-PSO has also been implemented in a hardware-only architecture for taking advantage of the processing in the previous studies [12, 13]. However, this hardware-only approach has the limitation of the flexibility. It is not easy to change the program in hardware because this task requires a lot of time and effort. In addition, the hardware implementation also costs many resources.

Several researchers have focused on a problem of premature convergence of the standard PSO algorithm (SPSO). One approach adds repulsion function to the SPSO algorithm [14]. The other approach uses radius r to detect whether the particle is bounded backward [15]. Several other researchers add the mutation operations to the SPSO algorithm [16, 17, 18]. The PSO algorithm may become complex because these approaches add more functions, more tasks to the SPSO algorithm. In our previous studies [19, 20], an improved

version of the SPSO algorithm called PSO with velocity control (PSO_CV) algorithm was introduced. This algorithm is not complex because only velocity update function is modified. The advantages concerning the recognition rate and the learning error of the NN trained by PSO_CV (NN-PSO_CV) were observed. However, the NN-PSO_CV was only implemented on the hardware-only architecture [19] or the software-only architecture [20].

The main contribution of this research is to create a hybrid framework for the evaluation of the NN-PSO_CV. In this architecture, the operation of the NN is accelerated by using FPGA, and the PSO training is executed by an embedded-processor. It is easy to update and modify this training module. The whole system is implemented in an FPGA chip that is compact, portable, and low power consumption.

This paper is organized as follows. Section 2 describes PSO_CV algorithm. Section 3 details the hardware-software architecture for NN-PSO_CV. Section 4 presents our experiments and our discussion. We conclude in Section 5 with several possible directions to improve our research.

2. The NN Trained by the PSO with Velocity Control.

In NN-PSO system, the number of the weights and the biases of the NN equals the D dimensions of each particle in the swarm. In each iteration of the training, the PSO algorithm tries to find the positions that minimize the learning error of the NN.

In SPSO algorithm [5], the new velocity of each particle depends on the current velocity, the current position of this particle, the best position found by this particle (x_Pbest), and the best position found by any particle in the swarm (x_Gbest) as can be seen in Equation (1). The new position of the particle is calculated by Equation (2). $Gbest$ and $Pbest$ which are fitness values of position x_Gbest and position x_Pbest are updated by Equation (3) and Equation (4).

$$\begin{aligned} v_p(t+1) = w \times v_p(t) + c_1 \times r_1 \left(x\_Pbest_p(t) - x\_p(t)\right) \\ + c_2 \times r_2 \left(x\_Gbest(t) - x\_p(t)\right) \end{aligned} \tag{1}$$

$$x\_p(t+1) = x\_p(t) + v_p(t+1) \tag{2}$$

$$Pbest_p(t+1) = \begin{cases} f(p(t+1)) & \text{if } f(p(t+1)) < Pbest_p(t) \\ Pbest_p(t) & \text{if } f(p(t+1)) \geqslant Pbest_p(t) \end{cases} \tag{3}$$

$$Gbest(t+1) = \underset{p}{\operatorname{argmin}}\, Pbest_p(t+1) \tag{4}$$

where w is the inertia weight, r_1 and r_2 are the random numbers, c_1 and c_2 are the cognitive coefficient and the social coefficient, and $p(t)$ is the position of particle p at time t.

The SPSO algorithm may stick to a local minimum. Our previous papers introduced the PSO_CV algorithm to overcome this limitation [19, 20]. The PSO_CV algorithm has the mechanism for the velocity control with three phases as illustrated in Figure 1.

(1) Swimming phase: particle moves toward to the middle position between x_Gbest and x_Pbest with a very high speed. The c_3 part becomes 0.
(2) Stopping phase: particle reduces the speed, and the new velocity of the particle starts to be affected by the c_3 part.
(3) Jumping phase: when the particle has a small speed, the value of the c_3 part becomes very high. The new calculated velocity will get a great value, and the particle will jump to another area for the new swimming phase as shown in Equation (5). This equation is the velocity update function of the PSOd_CV algorithm. However, several situations have significant dimensions and many local solutions. The PSOe_CV algorithm was

also introduced to search in the local area. The velocity update function of this algorithm is presented in Equation (6).

$$v_p(t+1) = w \times v_p(t) + c_1 \times (x\_Pbest_p(t) - x\_p(t)) + c_2 \times (x\_Gbest(t) - x\_p(t)) + \frac{c_3 \times r}{(v_p(t))^2} \quad (5)$$

$$v_p(t+1) = w \times v_p(t) + c_1 \times (x\_Pbest_p(t) - x\_p(t)) + c_2 \times (x\_Gbest(t) - x\_p(t)) + \frac{c_3 \times r}{e^{(v_p(t))^2}} \quad (6)$$

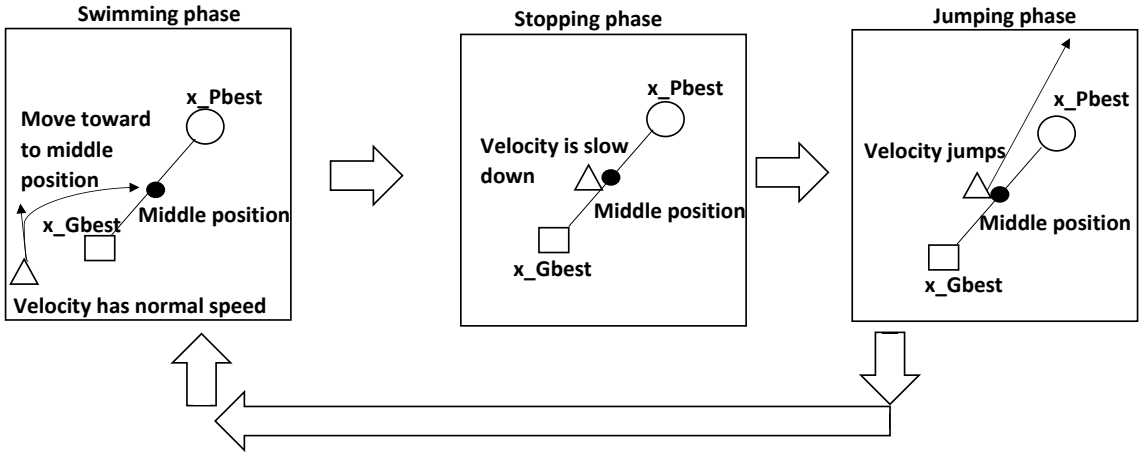

FIGURE 1. Mechanism of the PSO_CV algorithm [19, 20]

3. The Proposed Hardware-Software Architecture.

3.1. **Partitioning methodology.** In the NN-PSO system, the PSO module is only used during the training of the NN. This component is moved to the software side and implemented by C language to increase the flexibility. It is easy to modify the PSO parameters and even change the PSO algorithm. The NN is the main component used in the testing phase. This component is implemented by SystemVerilog language in hardware to maintain the testing speed of our architecture.

The NIOS II processor developed by Altera is chosen because it provides flexibility, high performance, low cost, and long life cycle [21]. In addition, Altera also provides the intellectual property core (IP cores) for maximizing the performance of the system [22, 23, 24].

3.2. **The operation of the proposed system.** Figure 2 demonstrates the PSO training phase. All modules except the NN module are implemented in software. The PSO weights and the input data are sent to the NN from the sending-data module. After finishing the execution, the NN sends the results to the output-data module. These data are processed by the evaluation module to calculate the new *Gbest* and the new *Pbest*. The calculation for the *Gbest* and the *Pbest* is performed by the mean squared error function as shown in Equation (7) as follows.

$$f_i = \frac{1}{T} \sum_{j=1}^{T} (target_j(k) - output_{ij}(k))^2 \quad (7)$$

where T is the number of training samples, and $target(k)$ and $output(k)$ are the kth component of the particle i in the target data and the output data of the NN.

The stopping-check module is used to test whether the stopping criteria are satisfied.

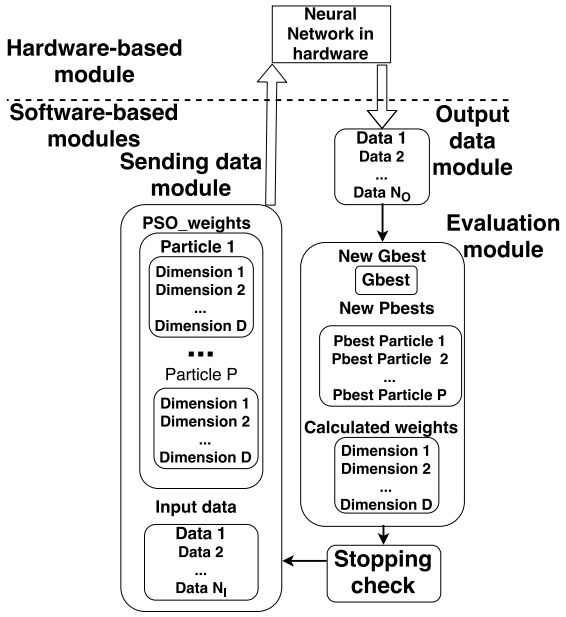

FIGURE 2. The training phase

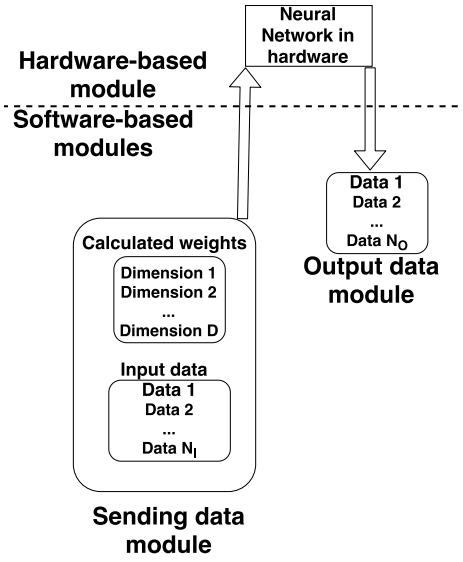

FIGURE 3. The testing phase

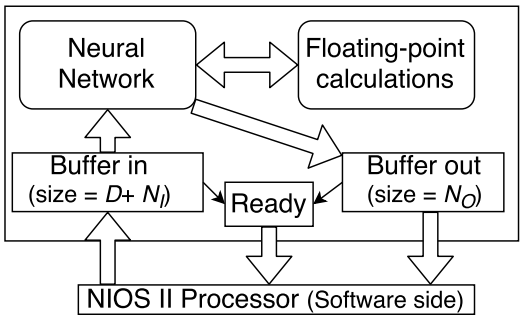

FIGURE 4. The FPGA-based component

In the testing phase, the calculated weights from the training phase and the input data are sent to the hardware-based NN from the sending-data module. The results of the NN are obtained at the output-data module (Figure 3).

3.3. **The FPGA-based component.** The FPGA-based component consists of the NN, a buffer_in, a buffer_out, a floating-point-calculation module, and a ready module (Figure 4). The connection between the FPGA-based component and the NIOS II is based on the Avalon Memory-Mapped Interface [22]. The floating-point-calculation module implements the floating-point IP cores to reduce the design time and increase the performance of our architecture [23]. The buffers are used to synchronize between the software-based modules and the hardware-based modules. The buffer_in receives the data from the software side. If the buffer_in is empty, the ready module will send the ready-to-receive signal to the processor. On the other hand, if the buffer_in is full, the input data will be sent to the FPGA-based NN. The buffer_out receives the data from the NN. If the buffer_out is full, the ready module will send the ready-to-send signal to the processor. The size of the buffer_in is $D + N_I$. In this situation, D is the number of the weights and biases of the NN or the number of the dimensions of each particle which is calculated by Equation (8).

$$D = (N_I + 1) \times N_H + (N_H + 1) \times N_H + (N_H + 1) \times N_O \tag{8}$$

where N_I, N_O, and N_H are the numbers of the nodes in the input layer, the output layer, and one hidden layer. Our NN has two hidden layers.

The operation of the FPGA-based NN is controlled by a finite state machine (FSM). The FSM of the NN is usually kept in the idle state. The FSM moves to running state when the buffer_in is full. Finishing the operation, the NN sends the output data to the buffer_out and returns to the idle state.

4. **Experiments.** All experiments were conducted with Cyclone V chip. Based on our experiments and our previous studies [19, 20], a good set of parameters for the high recognition rate of the SPSO algorithm and the PSO_CV algorithm was:

1) $w = 0.92$, $c1 = c2 = 0.3$ in the SPSO algorithm.
2) $w = 0.92$, $c1 = c2 = 0.3$, $c3 = 0.00001$ in the PSO_CV algorithm.

4.1. Experiments for evaluating of the NN-PSO_CV algorithm.

4.1.1. *Iris dataset.* This database has 150 samples of three different flowers called iris setosa, iris versicolour, and iris virginica. Each flower data has four attributes (sepal length, sepal width, petal length, and petal width) [25]. Our experiments used the 4-10-3 NN which has four input nodes (corresponding to four attributes), three output nodes (corresponding to three classes), ten nodes in one hidden layer, and two hidden layers.

Experiment 1: The database was divided randomly into two sets. The first set had a bigger number of samples (105 samples). The second set had the remaining 45 samples. The number of iterations (I) of all scenarios in this experiment was 60. When the first set was used as the training data, the second set was considered as the testing data and vice versa.

Scenario 1: This scenario focused on the significant number of the training data. In this situation, the first set was considered as the training data and the second set was the testing data. Figure 5 describes the reduction of the *Gbest*, the global minimum value of the learning error when the number of particles (P) was 40 particles. In this scenario, the *Gbest* of the PSOd_CV algorithm decreased to the lowest value (0.023) while other final *Gbest* values were 0.178 (PSOe_CV), and 0.180 (SPSO). The PSOd_CV also obtained a higher recognition rate (100%) than the PSOe_CV and the SPSO algorithms (only 66.67%).

If P was increased to 70 particles (Figure 6), *Gbest* value of SPSO was 0.034, PSOe_CV was 0.0289, and PSOd_CV was 0.0276. All three algorithms had 100% recognition rate.

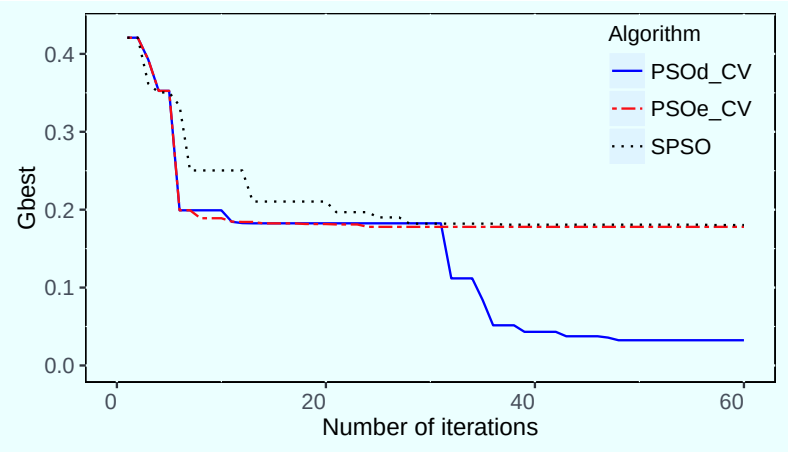

FIGURE 5. The reduction of *Gbest* when $P = 40$ particles

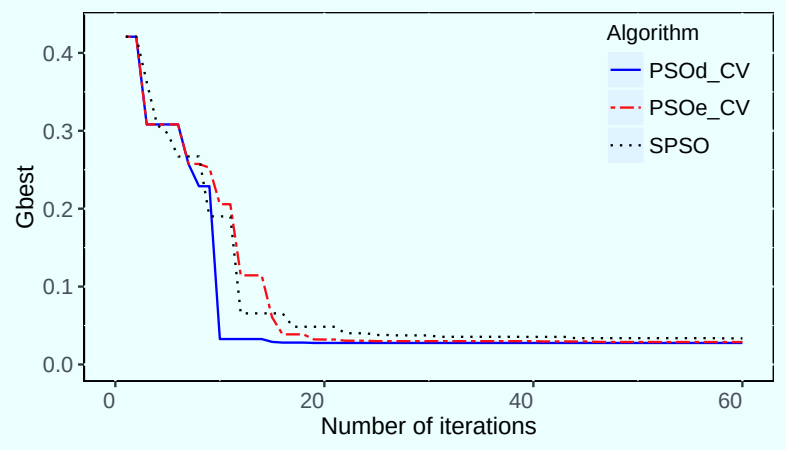

FIGURE 6. The reduction of *Gbest* when $P = 70$ particles

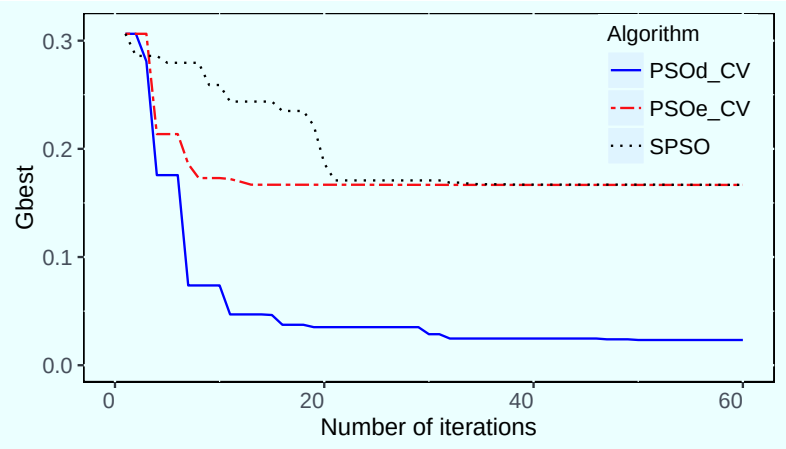

FIGURE 7. *Gbest* in scenario 2, Experiment 1 (45 training samples iris dataset)

Scenario 2: This scenario focused on the small number of the training data. The second set was used as the training data and the first set was the testing data. The number of particles was 70 particles (the recognition rate was 100% with this parameter in Scenario 1). As seen in Figure 7, PSOd_CV still produced better *Gbest* (final *Gbest* = 0.023) than PSOe_CV and SPSO. In this situation, the recognition rate of the PSOd_CV algorithm was 92.38%.

With the cross-validation, the PSOd_CV algorithm obtained the highest recognition rate and the lowest *Gbest* among three algorithms not only with the high number of training samples (105 samples) but also with the small number of the training samples (45 samples). In addition, the high recognition rate and the low *Gbest* of the PSOd_CV algorithm were also observed even with the small number of the particles ($P = 40$).

Experiment 2: A new division of iris dataset (120 training samples, and 30 testing samples) was conducted to investigate the operation of the NN-PSO system with different situations. Two main parameters affecting the training are the number of particles and the number of iterations. Experiment 1 already investigated the changing of the number of particles P. This experiment modified the number of iterations I, and kept P at $P = 50$. As seen in Table 1, the PSO_CV algorithm obtained the high recognition rate even with the small number of iterations ($I = 100$). On the other hand, the high number of iterations was required to increase the recognition rate of the SPSO algorithm.

4.1.2. *Balance-scale dataset.* The experiment with balance-scale data set was conducted to observe the operation of the NN-PSO system with another database. This dataset,

TABLE 1. Experiment 2

iterations	Algorithm	*Gbest*	Recognition
100	SPSO	0.168	66.67%
	PSOe_CV	0.008	93.33%
	PSOd_CV	0.008	93.33%
650	SPSO	0.092	80.00%
	PSOe_CV	0.008	93.33%
	PSOd_CV	0.008	93.33%
800	SPSO	0.043	90.00%
	PSOe_CV	0.008	93.33%
	PSOd_CV	0.008	93.33%

TABLE 2. Balance-scale data set

Algorithm	*Gbest*	Recognition rate
SPSO	0.119	83%
PSOe_CV	0.116	83%
PSOd_CV	0.114	85%

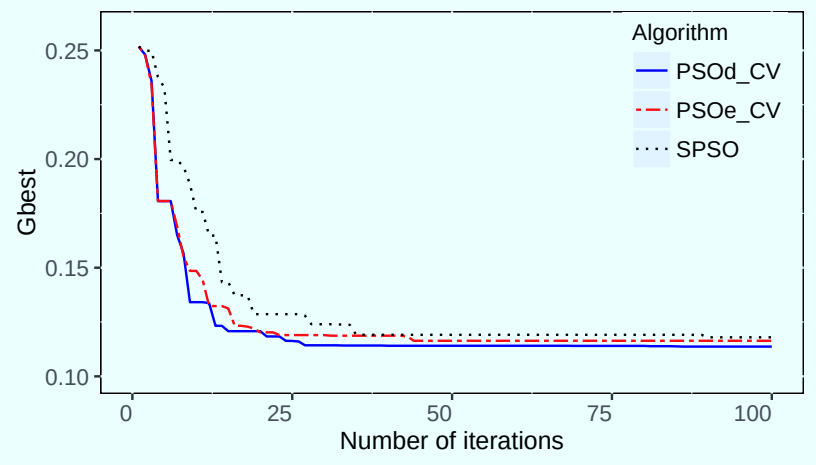

FIGURE 8. *Gbest* in the balance-scale dataset test

came from the psychology, has four attributes (left weight, left distance, right weight, and right distance) and three classes (right, left, or be balanced) [25]. The 4-10-3 NN was kept. The training data consisted of 245 samples chosen randomly. The testing data had 100 samples selected randomly. The number of particles was 60, and the number of iterations was 100. Experimental results in Table 2 and Figure 8 show that the PSOd_CV still obtained the best performance with this dataset.

Results with iris and balance-scale datasets once again confirmed that the PSOd_CV algorithm could be one solution to improve the accuracy of the SPSO algorithm.

4.2. **Experiments for evaluating of the proposed architecture.**

4.2.1. *The speed.* The testing speed was investigated because the FPGA-based NN is used to maintain the testing speed. Two versions of the NN were implemented. One is the software-based NN which was implemented by the C language in the Intel Core I3 2.4 GHz. The other one is the hardware-based NN which was implemented by the SystemVerilog language. To measure the time, the FPGA-based approach used the performance counter core [24], and the Intel-based approach used the `QueryPerformanceCounter()` function [26]. In the testing phase with the FPGA-based NN, the sending of the data to the NN,

the receiving of the results from the NN, and the printing of the results to the screen were executed by the NIOS II. Thus, the speed of the NIOS still affected the experimental results. This speed can be modified by a phase-locked loop [24].

The first experiment was used with iris data. The parameters were similar to the parameters of Experiment 1 when the recognition rate of the PSOd_CV algorithm was 100% (70 particles, 60 iterations, 4-10-3 NN, 105 training samples, 45 testing samples). It is necessary to test the speed of the NN with other sizes of NN. Therefore, the 2-6-4 NN was used to solve the XOR problem. The parameters for the PSO training were not modified (70 particles, 60 iterations). The results in both scenarios suggested that our hardware-based NN obtained a higher operating speed than the conventional software-based NN as illustrated in Table 3. In these experiments, both Intel-based approach and FPGA-based approach obtained 100% recognition rate.

Table 3. The testing time in second

NN size	frequency / Approach	100 MHz	115 MHz	2.4 GHz
4-10-3	Intel processor	–	–	0.078
Iris dataset	FPGA device	0.05	0.043	–
2-6-4	Intel processor	–	–	0.036
XOR	FPGA device	0.018	0.016	–

4.2.2. *The required resources.* This experiment investigated the advantage of the proposed architecture concerning the logic utilization that is calculated from the number of the adaptive logic modules (ALMs) in our design, and the number of ALMs available in the Cyclone V. For comparison, the hardware-only architecture which was developed based on our previous research [19] was also used in this experiment. As presented in Table 4, the required resources in hardware-only approach were excessive. The 2-6-4 NN cannot be implemented in the Cyclone V device (183% of the resources). Only the smaller NN such as the 2-2-2 NN may be fitted (70% of the resources). On the other hand, the compilation of the hardware-software architecture (2-6-4 NN) required only 27% of the resources.

Table 4. The logic utilization

Approach	Logic utilization
2-6-4 NN hardware-only	183%
2-2-2 NN hardware-only	70%
2-6-4 NN hardware-software	27%

The reduction of the logic utilization may be explained by the using of the NIOS II processor. In the hardware-only architecture, the PSO module implemented in FPGA requires many resources. On the other hand, the hardware-software architecture moves the PSO module to the NIOS II processor, and the FPGA resources reserved for this PSO module are not used. Our proposed design addresses the problem of the limited resources.

5. **Conclusions.** This paper proposed the hybrid hardware-software architecture that maintains the speed in the testing phase because the NN is still implemented in the hardware. Experimental results showed that the speed of the NN in our architecture was faster than the speed of the software-based NN. In addition, the proposed architecture has the flexibility of the software-based program when the PSO training is moved to the NIOS II processor. It is easy to change the parameters of the PSO training without recompiling the FPGA part. Our experiments also showed the reduction of the logic utilization (the ALMs) in the proposed architecture when compared with the hardware-only architecture.

This paper also evaluates the operation of the NN-PSO_CV in the proposed hardware-software architecture. The experimental results demonstrated that our NN-PSO_CV obtained better performance than the NN-PSO in the proposed architecture.

Our future research will investigate the NN-PSO_CV with more complex data sets and bigger NNs. To reduce the resources, the PSO weights will be stored in the RAM. An FPGA-based module will collect the weights from the RAM, and send these weights to the NN without the need for the NIOS II processor. The connection between the NIOS II processor and the RAM will use the direct memory access (DMA) to increase the operating speed. Another possible avenue for the future research is to use the ARM processor that may work in higher clock frequency than the NIOS II.

REFERENCES

[1] S. Haykin, *Neural Networks and Learning Machines*, 3rd Edition, Prentice Hall, 2008.

[2] Z. A. Bashir and M. E. El-Hawary, Applying wavelets to short-term load forecasting using PSO-based neural networks, *IEEE Trans. Power Systems*, vol.24, no.1, pp.20-27, 2009

[3] I. Vilovic, N. Burum and D. Millic, Using particle swarm optimization in training neural network for indoor field strength prediction, *Proc. of the 51st International Symposium ELMAR*, pp.275-278, 2009.

[4] V. G. Gudise and G. K. Venayagamoorthy, Comparison of particle swarm optimization and backpropagation as training algorithms for neural networks, *Proc. of IEEE Swarm Intelligence Symposium*, pp.110-117, 2003.

[5] J. Kennedy and R. Eberhart, Particle swarm optimization, *Proc. of the IEEE International Conference on Neural Networks*, vol.4, pp.1942-1948, 1995.

[6] R. Poli, J. Kennedy and T. Blackwell, Particle swarm optimization, *Swarm Intelligence*, vol.1, no.1, pp.33-57, 2007.

[7] I. C. Trelea, The particle swarm optimization algorithm: Convergence analysis and parameter selection, *Information Processing Letters*, vol.85, pp.317-325, 2003.

[8] M. T. Das and L. C. Dulger, Signature verification (SV) toolbox: Application of PSO-NN, *Engineering Applications of Artificial Intelligence*, vol.22, nos.4-5, pp.688-694, 2009.

[9] R. Mendes et al., Particle swarms for feedforward neural network training, *Proc. of the IEEE International Joint Conference on Neural Networks*, vol.2, pp.1895-1899, 2002.

[10] K. W. Chau, Application of a PSO-based neural network in analysis of outcomes of construction claims, *Automation in Construction*, vol.16, no.5, pp.642-646, 2007.

[11] E. Monmasson and N. C. M. Cirstea, FPGA design methodology for industrial control systems – A review, *IEEE Trans. Industrial Electronics*, vol.54, no.4, pp.1824-1842, 2007.

[12] M. A. Cavuslu, C. Karakuzu and F. Karakaya, Neural identification of dynamic systems on FPGA with improved PSO learning, *Applied Soft Computing*, vol.12, no.9, pp.2707-2718, 2012.

[13] C. J. Lin and H. M. Tsai, FPGA implementation of a wavelet neural network with particle swarm optimization learning, *Mathematical and Computer Modelling*, vol.47, pp.982-996, 2008.

[14] J. Riget and J. Vestertrom, *A Diversity-Guided Particle Swarm Optimizer*, Technical Report, University of Aarhus, 2002.

[15] T. Krink, J. S. Vesterstrom and J. Riget, Particle swarm optimisation with spatial particle extension, *Proc. of the IEEE International Conference on Evolutionary Computation*, pp.1474-1479, 2002.

[16] A. Ratnaweera, S. K. Halgamuge and H. C. Watson, Self-organizing hierarchical particle swarm optimizer with time-varying acceleration coefficients, *IEEE Trans. Evolutionary Computation*, vol.8, no.3, pp.240-255, 2004.

[17] X. Cai et al., Dispersed particle swarm optimization, *Information Processing Letters*, vol.105, no.6, pp.231-235, 2008.

[18] Y. C. Lu, J. C. Jan and G. H. Hung, Enhancing particle swarm optimization algorithm using two new strategies for optimizing design of truss structures, *Engineering Optimization*, vol.45, no.10, pp.1251-1271, 2012.

[19] T. L. Dang and Y. Hoshino, A hardware implementation of particle swarm optimization with a control of velocity for training neural network, *Proc. of the IEEE International Conference on Systems, Man, and Cybernetics*, pp.1980-1985, 2015.

[20] Y. Hoshino and H. Takimoto, PSO training of the neural network application for a controller of the line tracing car, *Proc. of the IEEE International Conference on Fuzzy Systems*, pp.1-8, 2012.

[21] Altera, *NIOS II Classic Processor Reference Guide*, https://www.altera.com, 2016.

[22] Altera, *Avalon Interface Specifications*, https://www.altera.com, 2016.

[23] Altera, *Floating-Point IP Cores User Guide*, https://www.altera.com, 2016.
[24] Altera, *Embedded Peripherals IP User Guide*, https://www.altera.com, 2016.
[25] M. Lichman, *UCI Machine Learning Repository*, http://archive.ics.uci.edu/ml, 2016.
[26] *Microsoft Developer Resource*, https://www.msdn.microsoft.com, 2016.