Applied Soft Computing 12 (2012) 2707–2718

Contents lists available at SciVerse ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

Neural identification of dynamic systems on FPGA with improved PSO learning

Mehmet Ali Cavuslu [a], Cihan Karakuzu [b], Fuat Karakaya [c,*]

[a] Koc Information and Defense Technologies Inc., METU Technopolis, ODTÜ-Teknokent, 06800 Ankara, Turkey
[b] Bilecik Seyh Edebali University, Engineering faculty, Department of Computer Engineering, Gülümbe Campus, 11210, Bilecik, Turkey
[c] Nigde University, Engineering Faculty, Department of Electrical and Electronics Engineering, Nigde, Turkey

ARTICLE INFO

Article history:
Received 9 June 2011
Received in revised form 31 January 2012
Accepted 4 March 2012
Available online 17 March 2012

Keywords:
Artificial neural networks (ANN)
Particle swarm optimization (PSO)
FPGA
System identification

ABSTRACT

This work introduces hardware implementation of artificial neural networks (ANNs) with learning ability on field programmable gate array (FPGA) for dynamic system identification. The learning phase is accomplished by using the improved particle swarm optimization (PSO). The improved PSO is obtained by modifying the velocity update function. Adding an extra term to the velocity update function reduced the possibility of stucking in a local minimum. The results indicates that ANN, trained using improved PSO algorithm, converges faster and produces more accurate results with a little extra hardware utilization cost.

1. Introduction

Artificial neural networks (ANNs) have been widely and successfully used in many fields to solve complex problems. ANNs are capable of successfully modeling non-linear complex correlation between input and output of a system. The most commonly used ANN is the multi layer perceptron (MLP) [1,2]. ANN is trained using a data set which is compatible with input–output relation of system to be modeled. Training is usually performed using back propagation (BP) algorithm. In recent years evolutionary algorithms are also used for ANN training as an alternative to BP algorithm.

For example population-based particle swarm optimization (PSO) [3], capable of stochastic search, has been successfully utilized in neural network training in addition to its usage in function optimization and fuzzy system control [4–7].

In recent years FPGAs, capable of parallel processing, are emerging as a predominant embedded system platform for many ANN applications [8–13]. In the literature numerous of hardware based ANN implementations on FPGAs with offline training [1,14–17] and online training [18–20] are reported. In these implementations different number formats such as fixed-point numbers with different bit width [16,18,20], floating-point numbers [1,14,17–19] and integer numbers [15] are utilized. Nonlinear logarithmic sigmoid [14–19] and nonlinear hyperbolic tangent [1,16,20] are employed as activation function. To implement nonlinear activation functions on hardware several approaches have been proposed: look-up table [15,20], piecewise-linear [1,16,18], parabolic [14] and functional [17,19].

In this work we present FPGA implementation of a system identification module based on ANN with PSO based online training. ANN and its PSO based online training module implemented in single precision floating-point number format and neural cell activation functions are implemented using functional approaches. In functional approach a divider is used in addition to adder and multiplier when compared to piecewise-linear and parabolic approach. Divider usage could be considered as the disadvantage of functional approach, but this approach has several advantages: (i) contrary to look-up table approach, no memory needed (ii) contrary to piecewise-linear approach, no control statements needed. The most important contribution of this work is the modified velocity update function of the PSO. In this paper authors proposed to add an extra term to the function which reduces the possibility of stucking in a local minimum during training. The experimental results indicate that the PSO algorithm utilizing the modified velocity update function converges faster and produces more accurate results.

The rest of the paper is organized as follows. Section 2 provides brief introduction to standard PSO and proposed improved PSO. Section 3 describes hardware implementation of ANN and its PSO based training module. Experimental results and performance of proposed approach in hardware are given in Section 4. The comparison results of the proposed improved PSO with other methods are given in Section 5 and the conclusions are given in Section 6.

* Corresponding author.
E-mail addresses: ali.cavuslu@kocsavunma.com.tr (M.A. Cavuslu), cihan.karakuzu@bilecik.edu.tr (C. Karakuzu), fkarakaya@nigde.edu.tr, karakuzu@bilecik.edu.tr (F. Karakaya).

1568-4946/$ – see front matter © 2012 Elsevier B.V. All rights reserved.
doi:10.1016/j.asoc.2012.03.022

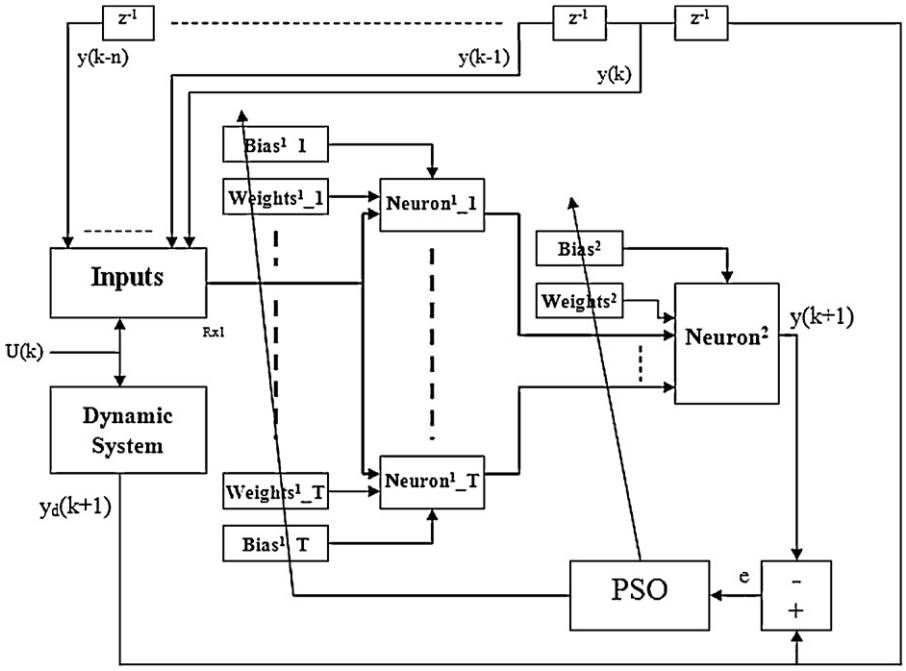

Fig. 1. PSO based ANN training and dynamic system identification architecture.

2. Standard and proposed modified PSO

PSO, inspired from social interactions among animals, is a high performance optimizer. PSO performs search using a population of particles. Each particle is directed to the most significant parts of the search space as a result of interactions between particles. The algorithm starts with randomly assigned particles. In every iteration the velocity and the location of particles are updated. Each particle's position is evaluated as a possible solution candidate. In a swarm with N particles, the position of the ith particle is defined as a vector as given in Eq. (1). D is the dimension of the search space.

$$\overrightarrow{p_i} = [p_{i1}, p_{i2}, p_{i3}, \ldots, p_{iD}] \tag{1}$$

Each particle memorizes its own best position that it has discovered, called the local best (p_b, Eq. (2)), and it also knows the best position discovered by any particle, called the global best (g_b, Eq. (3)).

$$\overrightarrow{p_b} = [p_{b1}, p_{b2}, p_{b3}, \ldots, p_{bD}] \tag{2}$$

$$\overrightarrow{g_b} = [g_{b1}, g_{b2}, g_{b3}, \ldots, g_{bD}] \tag{3}$$

The rate of change in particle position is called particle velocity and for the ith particle it is given as in Eq. (4).

$$\overrightarrow{v_i} = [v_{i1}, v_{i2}, v_{i3}, \ldots, v_{iD}] \tag{4}$$

At each iteration particle velocity is computed to determine the location of the particle at the next iteration. Particle velocity computing function, given in Eq. (5), have been proposed in [3] to compute the new velocity of each particle.

$$v_i(n+1) = \xi[v_i(n) + \alpha_1 r_1(p_{b,i} - p_i(n)) + \alpha_2 r_2(g_{b,i} - p_i(n))] \tag{5}$$

α_1, α_2 are called learning constants; r_1 and r_2 are uniformly distributed random numbers in [0–1) range; ξ is constriction factor. The most common problem with Eq. (5) is the possibility of stucking in a local minimum and as a result immature termination of the search. The authors of this paper propose an additional parameter to Eq. (5) which reduces the possibility of stucking in a local minimum. The proposed new velocity update function is given in Eq. (6).

$$v_i(n+1) = \xi[v_i(n) + \alpha_1 r_1(p_{b,i} - p_i(n)) + \alpha_2 r_2(g_{b,i} - p_i(n))] + [\alpha_3 \lambda(n)] \tag{6}$$

α_3 is also called additive learning constant; λ is a normally distributed random number vector. The value of α_3 is chosen according to the inequality given in Eq. (7). Where X is the matrix of input samples to be used for modeling or identification task.

$$\alpha_3 \ll \frac{1}{\max\{\mathrm{eig}(XX^T)\}} \tag{7}$$

After determining range of $\alpha_3\lambda$ term in modified PSO, $\alpha_3\lambda$ term is represented as power of two $[-2^{-z}\ 2^{-z}]$ (z is an integer number) to expedite the handling in hardware. As stated before, the last term in Eq. (6) prevents particles from stucking in a local minimum and it permits a more detailed search of the space. After the determination of the particle velocity, the new location of the particle is calculated using Eq. (8).

$$p_i(n+1) = p_i(n) + v_i(n+1) \tag{8}$$

3. PSO based ANN training ON FPGA

In this section, the details of PSO based ANN training on FPGA is described. The block diagram of the proposed architecture with the intention of dynamic system identification is shown in Fig. 1. PSO based ANN training on FPGA has four main stages as given in Fig. 2. Short description of these four stages is as follows.

3.1. Stage1: assignment of initial values

In FPGA implementation, the swarm (P), local best particles (P_b), global best particle (g_b), particle velocities (V_m) and fitness values (E_n) are stored in block RAMs. For P, P_b and V_m matrices a block RAM in $N \times D$ dimensions, for g_b vector a block RAM in $1 \times D$ dimension,

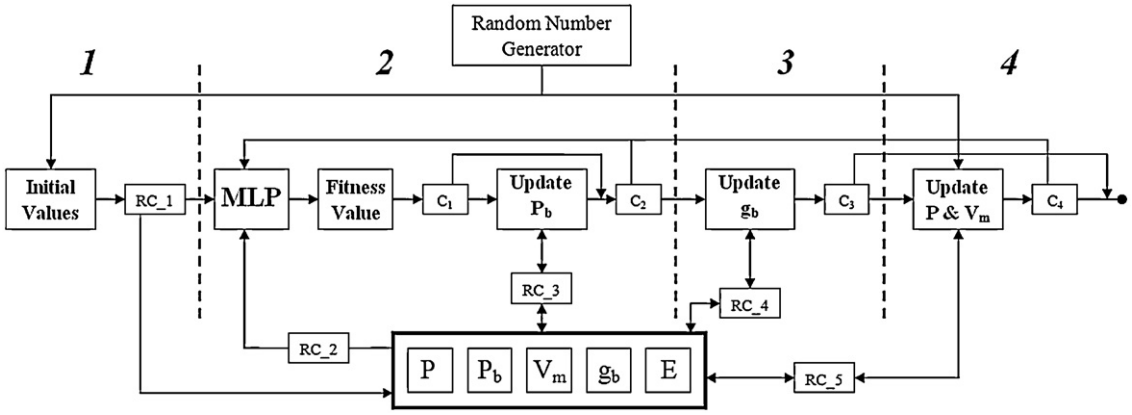

Fig. 2. Detailed architecture of PSO based ANN training on FPGA.

and for E_n vector a block RAM in $1 \times N$ dimension is used (N: swarm size, D: number of parameters optimized). The depth of the block RAMs is same as the bit width of number format utilized. In our proposed architecture parallel reading and writing transactions are possible by using separate block RAM structures for each vector. The initial values of particle positions and velocities are assigned using Eq. (9) [21] in single precision floating-point format within the range of [0,1).

$$X_{n+1} = (aX_n + b) \bmod c \tag{9}$$

Random number generation starts with X_0 seed value. a and b are constant numbers and last term corresponds to mod operation with respect to c (c is chosen as 1.0). Firstly, the initial values of P and P_b (these values are same at the beginning) are generated and written to the block RAM, next the initial values of V_m and g_b are written to the block RAM respectively. After completing the initial value generation, Stage 2 starts.

3.2. Stage 2: hardware implementation of the ANN and determination of the best local particles

Stage 2 could be summarized as follows: initial values of the particles are read from the RAM and assigned as the ANN parameters and using these parameters the error at the output of the ANN is obtained for each input. Using these error values the fitness value is calculated for each iteration. If the obtained fitness value for the particle is smaller than the previous one, the particle is assigned as the local best. This step is repeated for each particle. The most important block used in this stage is the MLP module which is the hardware implementation of the ANN. The RC_2 block reads the stored data related to ith particle from the RAM. ANN parameters (weights, biases) used for the current iteration step of the training is assigned from the swarm as in Eq. (10).

$$\overrightarrow{p_i} = [w_{11}^1, w_{12}^1, \ldots, w_{1T}^1, w_{21}^1, \ldots, w_{RT}^1, b_1^1, \ldots, b_T^1, w_1^2, \ldots, w_T^2, b^2] \tag{10}$$

In this work single-precision floating point representation (Eq. (11)) is preferred because of its dynamism and precision.

$$\text{Number} = (-1)^s(1 \cdot f)2^{e-\text{bias}} \tag{11}$$

For floating-point arithmetic on FPGA we utilized “add.lib”, “mul.lib”, “div.lib” libraries that was developed in our previous works [17,19,22].

One of the most critical components of an ANN is the activation function. Activation function generates the neuron output by processing the addition of the bias value and the multiplication result of the neuron input and weight. Most commonly used activation functions in ANN implementations contain exponential terms. Exponential functions cannot be realized directly on FPGAs. As a result several approximations have been developed. For example logarithmic sigmoid function is approximated as in Eq. (12) [23]

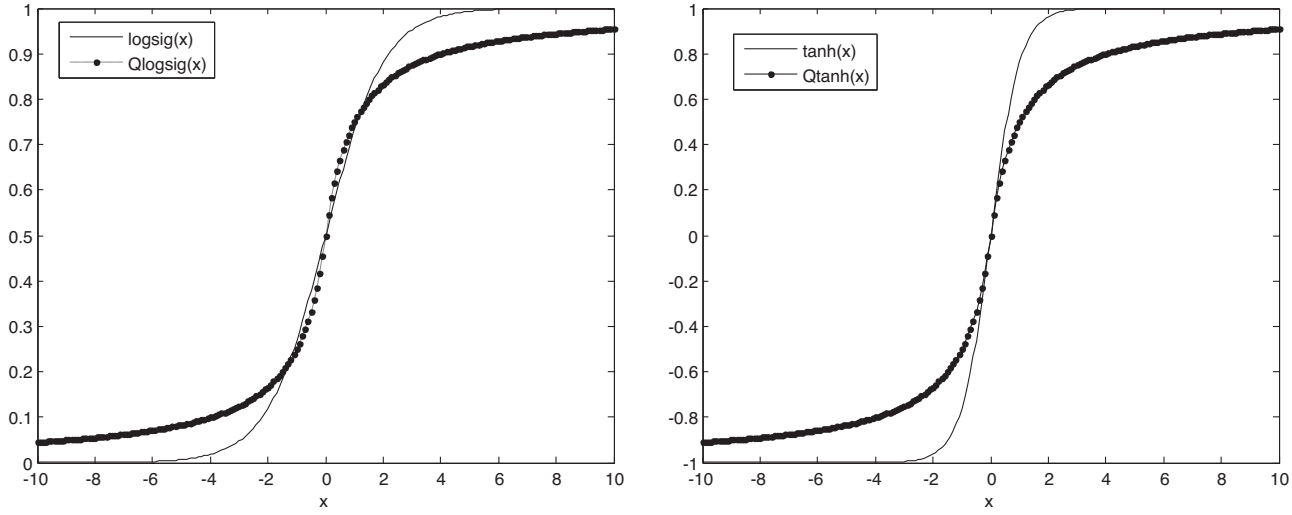

Fig. 3. Activation function approximations.

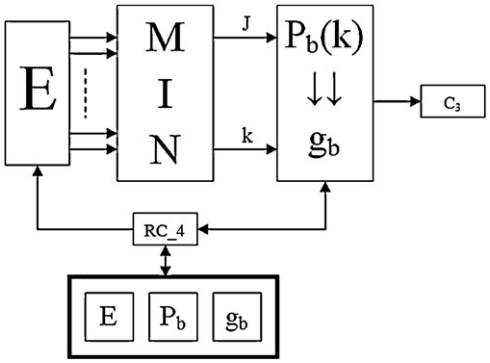

Fig. 4. Block diagram for the determination of the global best particle.

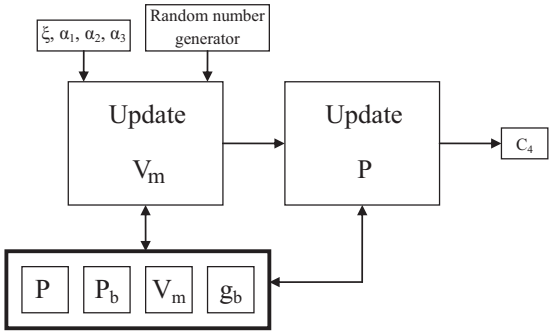

Fig. 5. Block diagram of updating procedure.

and tangent hyperbolic is approximated as in Eq. (13) [24]. Plots of original functions and its approximations are given in Fig. 3.

$$Q(x) = \frac{1}{2}\left[1 + \frac{x}{1+|x|}\right] \tag{12}$$

$$Q(x) = \frac{x}{1+|x|} \tag{13}$$

As seen from Eqs. (12) and (13) the approximations require absolute value operation. In floating-point arithmetic, absolute value operation is simply performed by changing the sign bit of the number (1–0 or 0–1). Division by two in Eq. (12) is performed by decreasing the exponent by one (e−1). Total square error criterion (Eq. (14)) is used for fitness value calculation. The error (e_j) in Eq. (14) is expressed as in Eq. (15).

$$E_n = \frac{1}{2}\sum_j e_j^2 \tag{14}$$

$$e_j = yd_j - y_j \tag{15}$$

j represents the sample index, yd_j represents the desired output and y_j is the real output of the MLP. Error calculation is performed in "fitness value" block shown in Fig. 2. Block C1 compares obtained new E_n value with previous one En_old. If the new value is less than the previous one the corresponding particle is written to the RAM space designated for p_b and value of En_old is replaced with value of

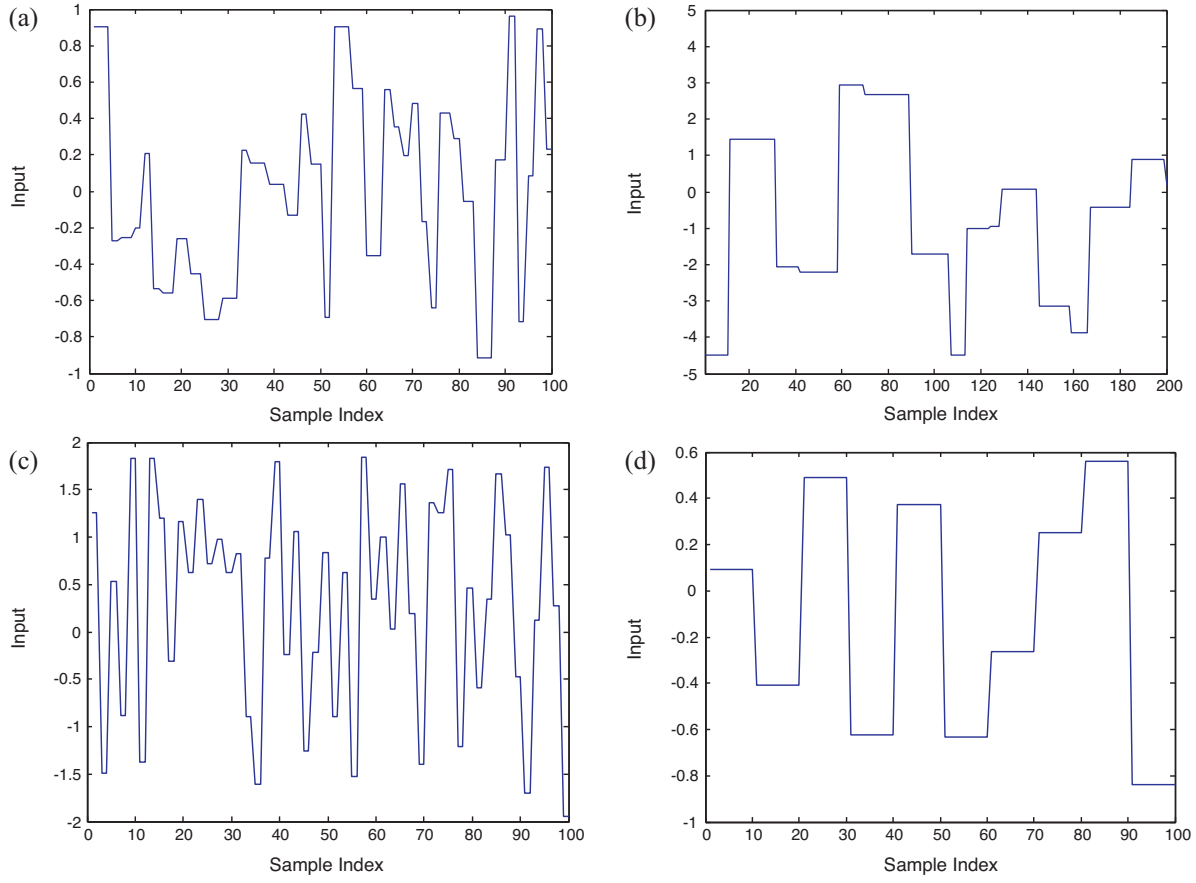

Fig. 6. Training input sequences for examples: (a) example 1: pulses with random amplitude in the range [−1 1] and with random duration between 1 and 4 sampling periods (b) example 2: a series of random continuous pulses with a sampling period in the range of [1 20] and a magnitude in the range of [−5 5] (c) example 3: pulses with random amplitude in the range [−2 2] and a with random duration between 1 and 2 sampling periods (d) example 4: series of random continuous pulses with a sampling period in the range of 10 and a magnitude in the range of [−1 1].

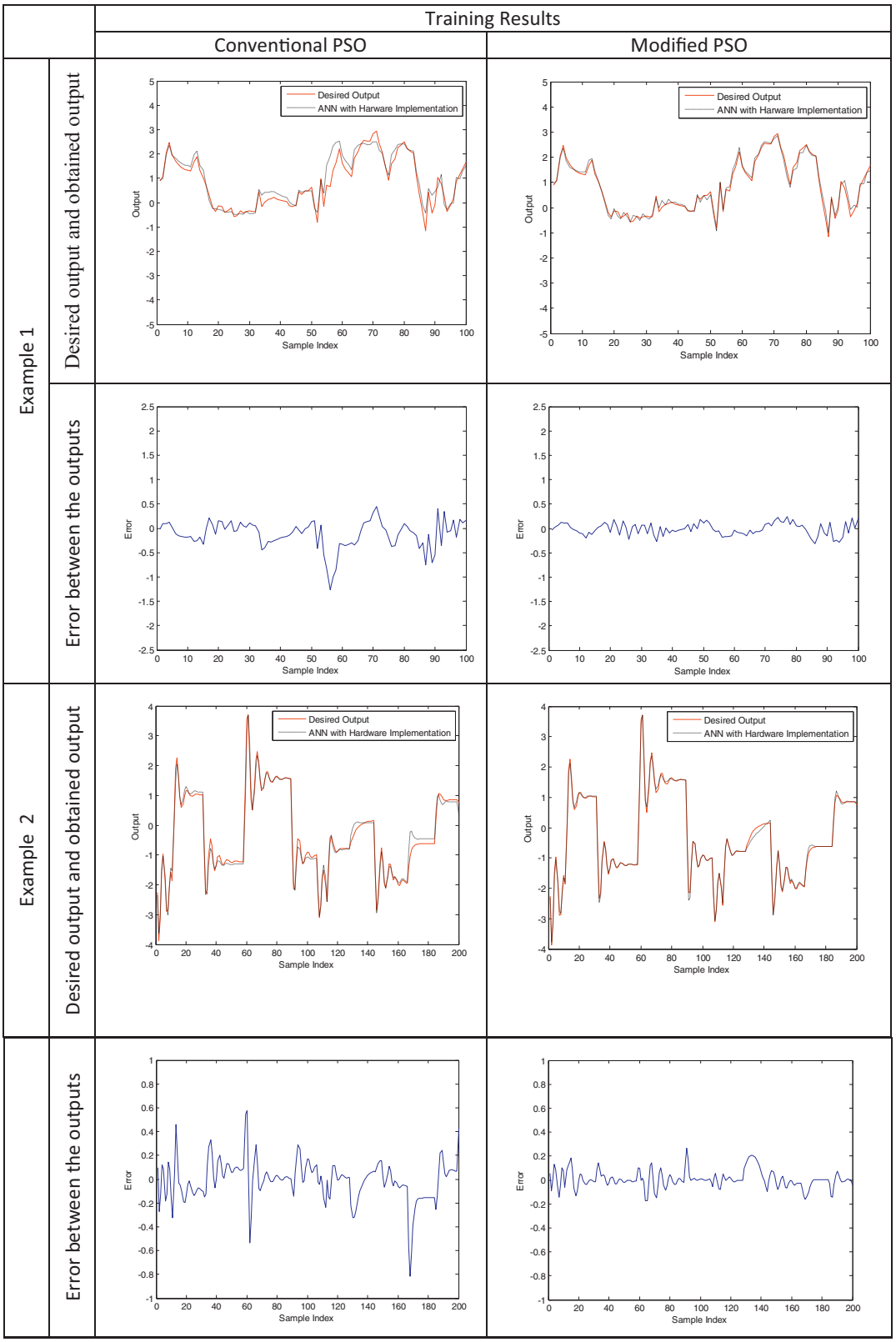

Fig. 7. Experimental results of the ANN training.

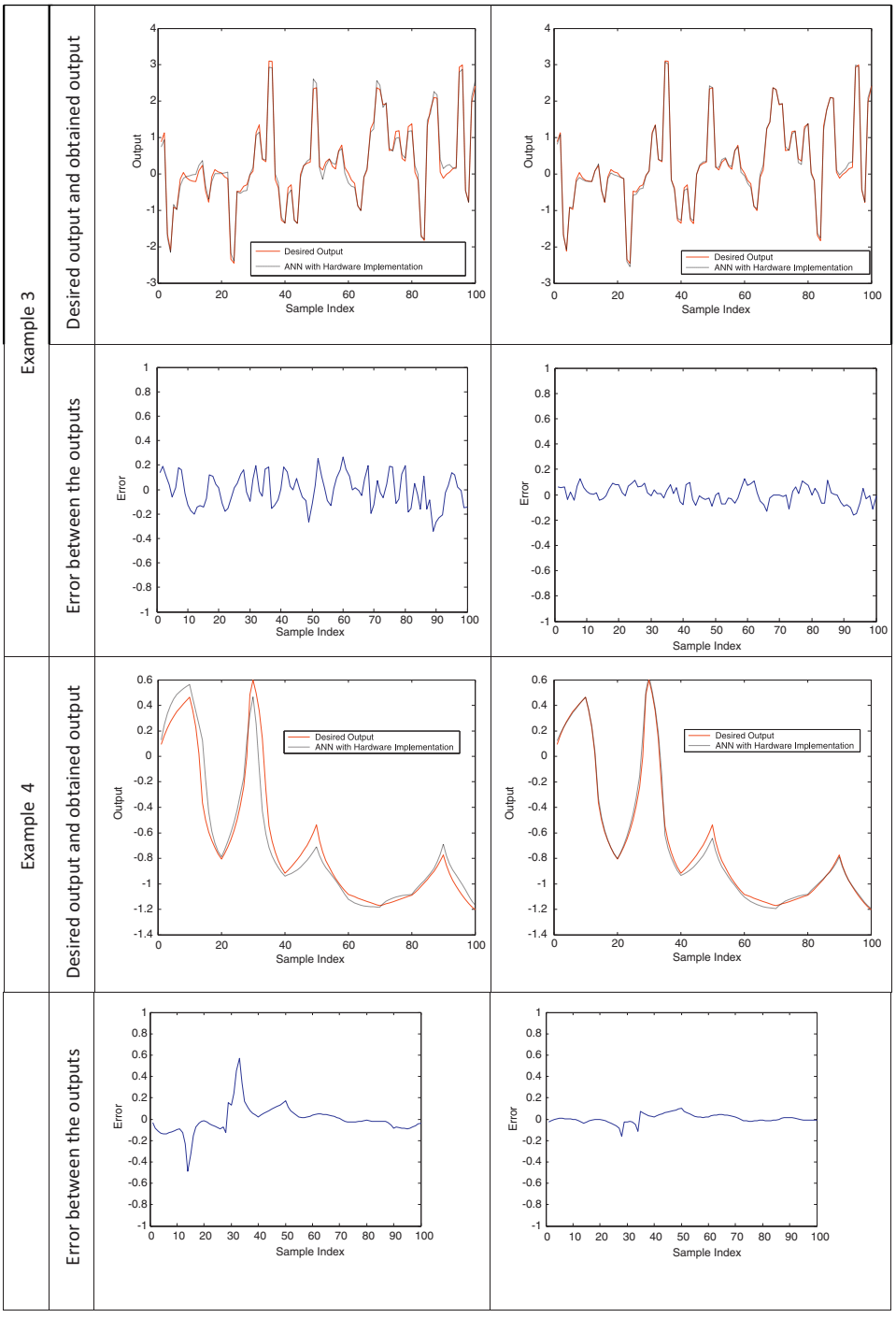

Fig. 7. *continued*

E_n. Otherwise "Update P_b" block is skipped and process continues from block C2, which makes sure that same procedures are applied to each particle. If updating is required block RC_3 reads the information of ith particle p_i of the swarm from the RAM and writes it to the RAM space designated as the local best. It also updates the fitness value of the ith particle.

3.3. Stage 3: determination of the global best particles

After the calculation of fitness values and determination of the local best particles (P_b), the global best particle is identified. The minimum value among the values kept in E_n, the minimum error which corresponds to the best fitness value, is determined and corresponding row vector of the local best matrix (P_b), which is marked with the same index as the minimum element of E_n, is assigned as the global best (g_b). Detailed block diagram of this step is given in Fig. 4. The C3 block in Fig. 4 checks the value of minimum criteria value against the termination criteria (J_d). If the minimum criteria value is less than termination criteria the training is stopped. If that is not the case stage 4 is activated. RC_4 block is in charge of reading the fitness values, the data related to the local best particle and writing the global best values.

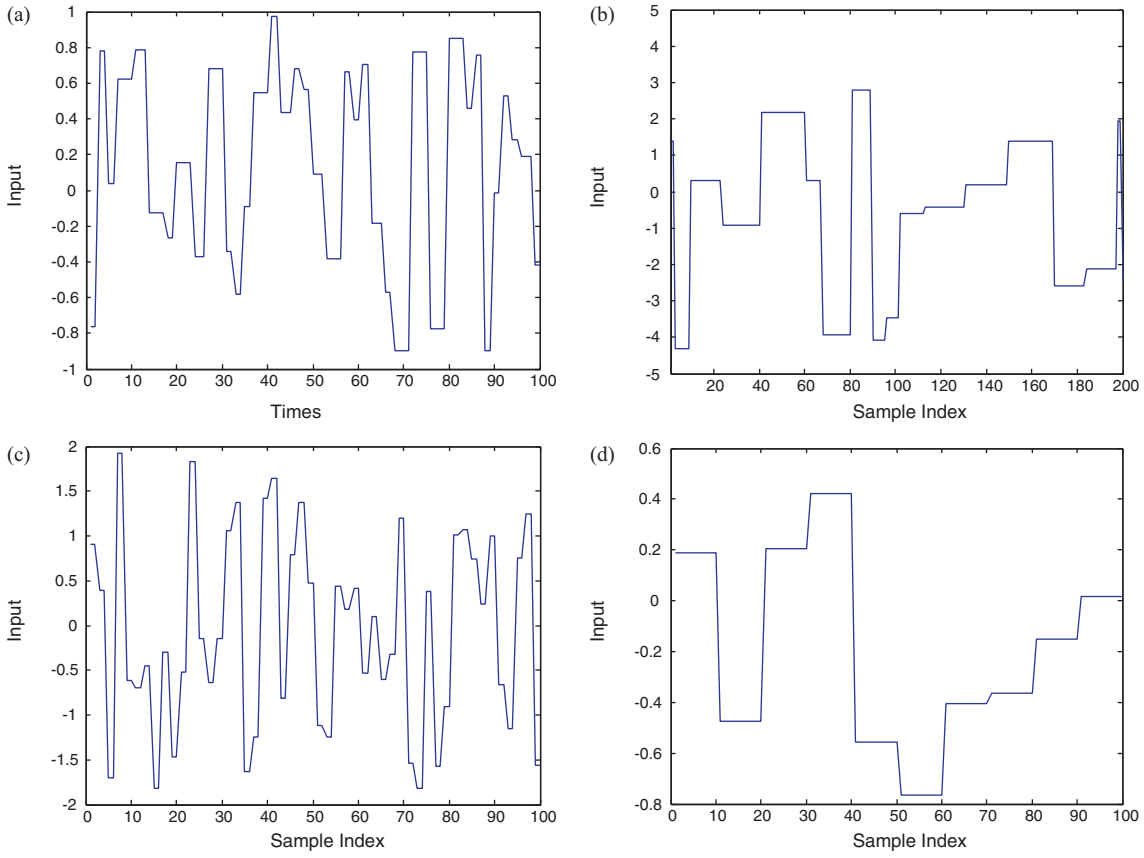

Fig. 8. Input sequences for the testing phase for (a) example 1 (b) example 2 (c) example 3 (d) example 4.

3.4. Stage 4: updating

The values of P, P_b, V_m and g_b are read and updated simultaneously from the RAM. Two different methods are employed for this purpose. In the first method V_m matrix is updated using standard PSO equation (Eq. (5)). In the second method V_m matrix is updated using modified PSO equation (Eq. (6)) ($\xi = 0.76$, $\alpha_1 = \alpha_2 = 2.1$). During the updating calculations velocity values are kept within $[-1\ 1]$ range, particles component values are limited within the range of $[-100\ 100]$. In modified PSO, the values of $\alpha_3\lambda$ elements are limited within the range of $[-2^{-z}\ 2^{-z}]$.

The updated values of P and V_m are simultaneously written back to the same location of the RAM. After updating the parameters of the each particle, the iteration number is increased by one and flow moves back to stage 2. If the iteration number is greater than the maximum generation number ($G_{\max}$), the process is terminated. Else, Stage 2 is activated. Block C4, shown in Fig. 5, is in charge of this procedure.

4. Experimental results

The details of PSO based ANN training implementation on FPGA (Altera Cyclone II EP2C70F672C6), is given in previous section. The hardware implementation is experimentally tested with four dynamic system identification problems. Learning phase for the dynamic system identification problem is operated as illustrated in Fig. 1. ANN identification model, whose parameters are determined at the end of the learning phase, is tested for the data set unseen in the learning phase. This testing phase is operated using the architecture given in Fig. 1 by neglecting PSO block and its tuning operands. For testing, the ANN parameters (weights, biases) are assigned from the global best (g_b) values which are found at the end of learning phase as in Eq. (16).

$$\vec{g_b} = [w_{11}^1, w_{12}^1, \ldots, w_{1T}^1, w_{21}^1, \ldots, w_{RT}^1, b_1^1, \ldots, b_T^1, w_1^2, \ldots, w_T^2, b^2] \quad (16)$$

Dynamic system identification is the procedure of properly tuning the parameters representing the system. This tuning procedure is performed by optimizing the performance based on the error between the real output of the system to be identified and the output of the model selected to identify.

For system identification a feed-forward ANN architecture is used. For the learning phase, a PSO algorithm with 100 particles is used and as velocity update function both Eqs. (5) and (6) are used and results are tabulated to show the effectiveness of the modified update function.

To evaluate contribution of the modified velocity update function, performance of the both velocity update functions, given in Eqs. (5) and (6), are compared using results of 10 separate runs operated on the hardware (FPGA) implementation for the selected examples listed in Table 1. Training and testing phases have been

Table 1
Selected examples to evaluate the improved PSO.

Examples		
1	$y(k+1) = \frac{y(k)y(k-1)[y(k)+2.5]}{1+y^2(k)+y^2(k-1)} + u(k)$	[25]
2	$y(k+1) = \frac{24+y(k)}{30} y(k) - 0.8\frac{u(k)^2}{1+u(k)^2} y(k-1) + 0.5u(k)$	[26]
3	$y(k+1) = 0.5\left[\frac{y(k)}{1+y^2(k)} + (1+u(k))u(k)(1-u(k))\right]$	[27]
4	$y(k+1) = y(k) + u(k)e^{-3\lvert y(k)\rvert}$	[28]

Table 2
The ANN architectures and activation functions used for each example.

Example	ANN Architecture	Activation functions	
		Hidden layer	Output layer
1	1 hidden layer w/ 4 neurons + output layer w/ 1 neuron	Logarithmic sigmoid	Linear
2	1 hidden layer w/ 4 neurons + output layer w/ 1 neuron	Tangent hyperbolic	Linear
3	1 hidden layer w/ 5 neurons + output layer w/ 1 neuron	Logarithmic sigmoid	Linear
4	1 hidden layer w/ 5 neurons + output layer w/ 1 neuron	Tangent hyperbolic	Linear

Table 3
Memory usage for storing PSO parameter matrices and vectors.

	Bit
P	67,200
P_{best}	67,200
V_m	67,200
g_{best}	672
E_n	3200
Utilized	205,472
Total	1,152,000
Percentage	17.8

Table 4
Ranges of $\alpha_3\lambda$ term in improved PSO for each Example.

Example	1	2	3	4
$\alpha_3\lambda$	$[-2^{-12}\ 2^{-12}]$	$[-2^{-15}\ 2^{-15}]$	$[-2^{-11}\ 2^{-11}]$	$[-2^{-11}\ 2^{-11}]$

operated for the training and test data sets respectively for each separate run. Average performance criteria (APC) computed from performance criteria based on Eq. (14) is used as comparison cost. The ANN architectures and activation functions used in the experiments are listed in Table 2 and the input data set sequences are shown in Fig. 6 for the training phase of each example.

RAM space needed for the PSO parameters is given in Table 3 and ranges of $\alpha_3\lambda$ term in modified PSO are listed in Table 4.

Total number of arithmetic modules (adder, multiplier, divider) used in the proposed system identification architecture is summarized in Table 5 and the hardware resource utilization results are given in Table 6.

Fig. 7 shows the desired values of the output and the estimation of the hardware (FPGA) implementation using Eqs. (5) and (6) and the error of the hardware implementation of the dynamic system identification for the training phase.

After training, ANNs are tested using the input sequences given in Fig. 8. In Fig. 9, the results produced by the ANNs and the error (between the desired output and ANN output) is given for comparison of improved PSO against the conventional PSO.

To evaluate the contribution of the modified velocity update function in a general framework, performances of the both conventional PSO and improved PSO are compared using results of 10 separate runs of the examples on the hardware (FPGA) implementation. Training and testing phases are operated for the training and test data sets respectively for each separate run. APC, computed from performance criteria based on Eq. (14), is used as comparison cost. Table 7 summarizes results of this comparison for both training and testing data sets. As clearly seen from Fig. 9 and Table 7, training and testing phases performed using improved PSO, converges faster and produces smaller performance cost value. This shows the effectiveness of proposed velocity update function, due to its ability in skipping the local minimums.

5. Comparison of improved PSO with other methods

To give more general idea about the performance of the proposed improved PSO, it has been compared with BP, artificial bee colony (ABC) [29] and conventional PSO [3] algorithms on software environment. Comparisons are carried out for ANN training for the identification examples given in the previous section. Codes for ABC algorithm has been taken from [30] and adopted to the identification examples. ANN architectures are the same as ones given in the previous section for each example. Learning rate has been used as 0.05 for BP, swarm size is 100 and bees/particles component values are limited within the range of [−100 100] for ABC and PSO algorithms in comparisons. Comparisons have been done using results of 10 separate runs operated on the software implementation. Training and testing phases operated for the training and test data sets respectively for each separate run. Table 8 shows results of this comparison in terms of measures of central tendency for both

Table 5
Number of arithmetic modules used in Examples.

Example	Modules	Conventional PSO			Improved PSO		
		Sum	Mul	Div	Sum	Mul	Div
1	MLP	24	16	4	24	16	4
	Fitness function	2	1	–	2	1	–
	Updating	5	4	–	6	4	–
	Total	31	21	4	32	21	4
2	MLP	20	16	4	20	16	4
	Fitness function	2	1	–	2	1	–
	Updating	5	4	–	6	4	–
	Total	27	21	4	28	21	4
3	MLP	25	15	5	25	15	5
	Fitness function	2	1	–	2	1	–
	Updating	5	4	–	6	4	–
	Total	32	20	5	33	20	5
4	MLP	20	15	5	20	15	5
	Fitness function	2	1	–	2	1	–
	Updating	5	4	–	6	4	–
	Total	27	20	5	27	20	5

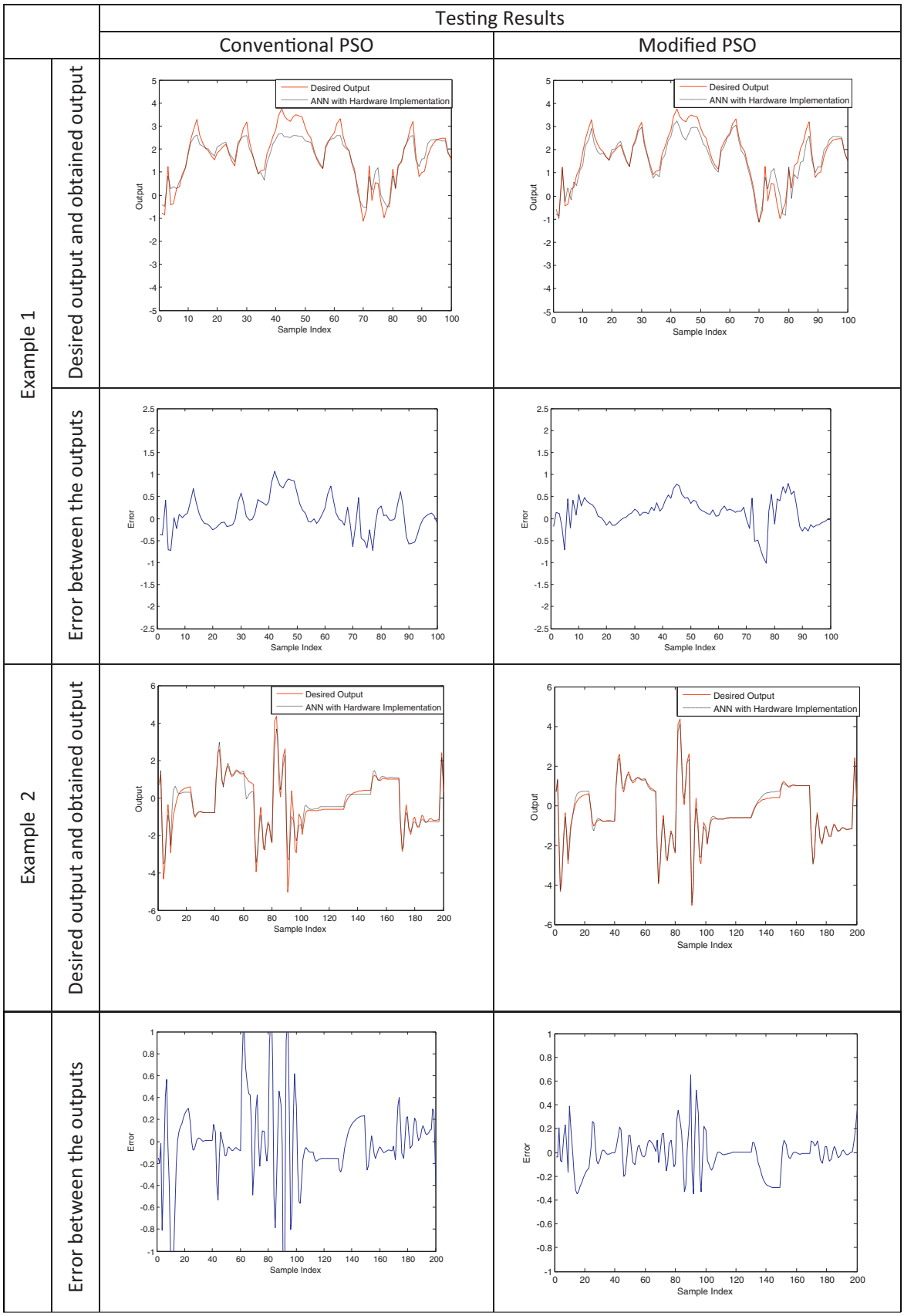

Fig. 9. Experimental results of the testing phase.

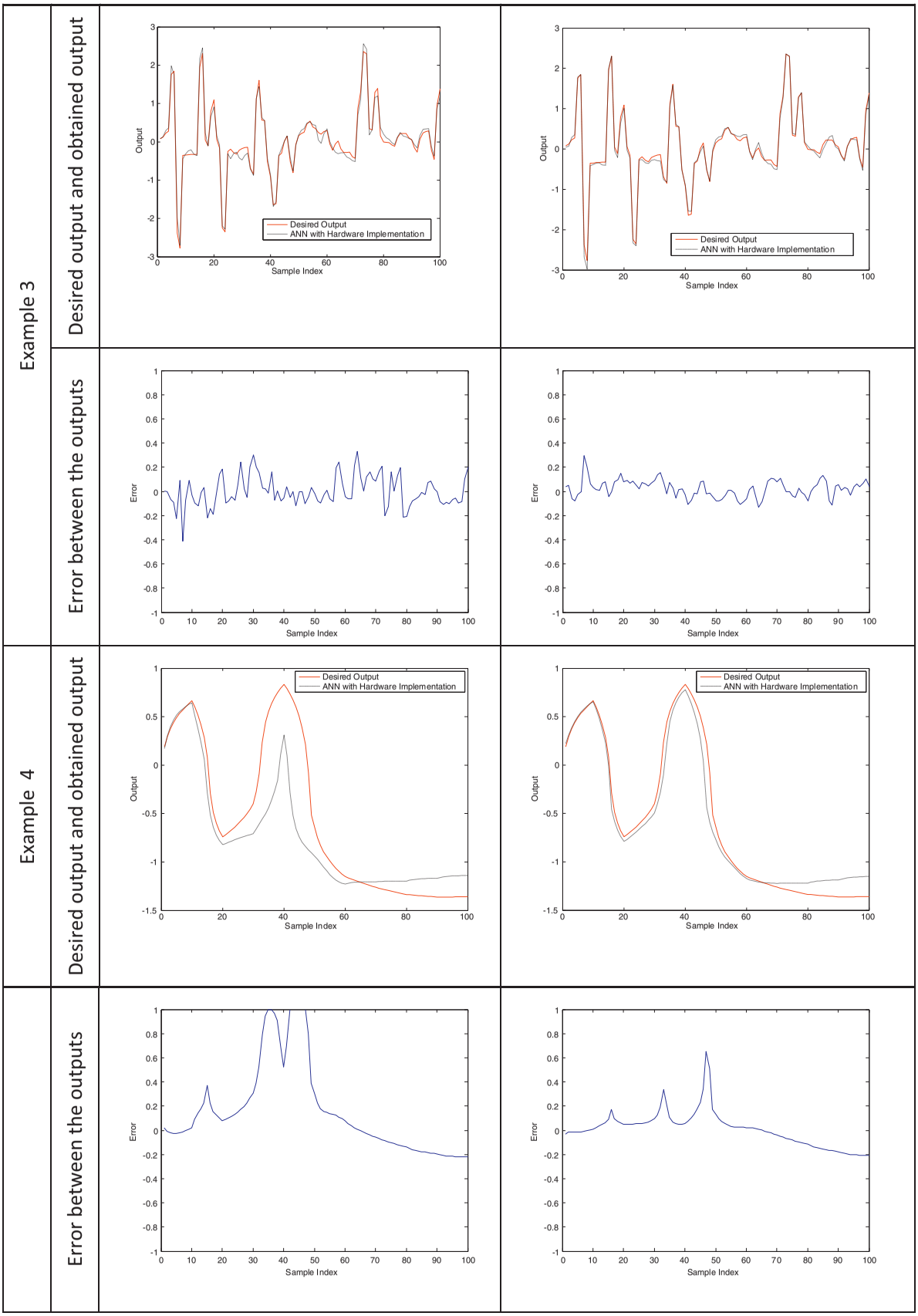

Fig. 9. (*continued*)

Table 6
Hardware resource utilization for each Example.

Example	Conventional PSO			Improved PSO		
	Utilized LEs	Total LEs	Percentage	Utilized LEs	Total LEs	Percentage
1	52,001	68,416	76	53,245	68,416	78
2	48,163	68,416	70	49,737	68,416	72
3	54,073	68,416	79	55,381	68,416	81
4	50,174	68,416	73	49,737	51,229	74

Table 7
Average performance criteria (APC) values for 10 runs for the both conventional and improved PSO.

Example		Average performance criteria		Maximum iteration
		Training	Testing	
1	Conventional PSO	4.0212	8.6418	1500
	Improved PSO	1.8992	6.0184	1500
2	Conventional PSO	4.3759	12.2492	4000
	Improved PSO	0.55109	2.3481	4000
3	Conventional PSO	0.6303	0.8149	1000
	Improved PSO	0.1892	0.2856	1000
4	Conventional PSO	0.6195	7.1068	1500
	Improved PSO	0. 0964	2.6024	1500

Table 8
Performance comparisons of the four algorithms based on ANN training and testing.

Algorithm	Measures	Example 1 (iteration = 1500)		Example 2 (iteration = 4000)		Example 3 (iteration = 1000)		Example 4 (iteration = 1500)	
		Training	Testing	Training	Testing	Training	Testing	Training	Testing
BP	Mean	4,240336	11,391400	1,244220	14,405780	0,488310	0,673310	1,133990	14,258010
	Std	1,619980	3,159723	0,457640	2,920739	0,187704	0,243582	0,415601	4,346798
	Worst	7,160700	13,992100	2,216900	20,790500	0,742200	1,023000	1,745600	19,659600
	Best	2,055200	5,827700	0,776300	10,409900	0,181800	0,302600	0,616600	3,781200
ABC	Mean	4,743880	9,113430	1,636280	14,279380	2,130490	2,212380	1,782770	17,151750
	Std	**0,563837**	1,852862	0,341141	2,975066	0,604044	0,623781	1,405144	7,783955
	Worst	5,622800	12,587700	2,136100	19,954700	3,143400	3,069800	5,168700	24,056900
	Best	3,974700	6,961000	1,240500	10,445300	1,180800	1,144200	0,401800	2,674300
PSO	Mean	4,509840	7,782630	3,666150	11,504090	0,614710	0,984320	0,779730	10,439520
	Std	1,968178	1,971067	0,806233	3,420604	0,180456	0,380693	0,327521	3,067629
	Worst	8,601400	10,751800	4,728000	17,999000	0,838100	1,856600	1,217000	15,868400
	Best	2,141900	4,970900	2,650600	8,291500	0,283200	0,585300	0,317300	6,989700
Improved PSO	Mean	**1,502510**	**5,713820**	**0,604320**	**2,463310**	**0,236720**	**0,377050**	**0,126833**	**2,899920**
	Std	0,599760	**0,829939**	**0,237170**	**0,866204**	**0,070984**	**0,118777**	**0,111158**	**1,002375**
	Worst	**2,161900**	**6,339600**	**0,906200**	**3,875300**	**0,339000**	**0,552400**	**0,383600**	**4,565700**
	Best	**0,339900**	**3,647500**	**0,322600**	**1,518200**	**0,125800**	**0,188700**	**0,026300**	**1,555300**

training and testing data sets. In the table, "*Mean*", "*Std*", "*Worst*" *and* "*Best*" represent respectively arithmetic mean, standard deviation, maximum and minimum of the cost values obtained at the end of 10 separate runs. The best values are marked with bold fonts for each category in the table. As can be seen from the table, for all categories except one, the exceptional superiority of the improved PSO is clear.

6. Conclusion

In this paper, the hardware implementation of ANN with PSO online training for system identification is proposed. Contribution of this work is twofold. First, it introduces the modified velocity update function for PSO (Eq. (6)). Authors proposed to add an extra term ($\alpha_3\lambda(n)$) to the function which reduces the possibility of stucking in a local minimum. Second, it proposes an effective system identification architecture based on ANN with PSO learning on FPGA. To prove the effectiveness of the proposed improved PSO and system identification architecture, four system identification problems from the literature are selected. Proposed architecture is trained and tested for those selected identification problems on FPGA. The results indicate that ANN on FPGA is trained efficiently well to make an acceptable generalization and PSO algorithm utilizing the modified velocity update function converges faster and produces more accurate results with a little extra hardware utilization cost. The improved PSO is also compared with other methods (BP and ABC) and the results show that improved PSO produced much better results than the aforementioned methods.

References

[1] P. Ferreira, P. Ribeiro, A. Antunes, F.M. Dias, A high bit resolution FPGA implementation of a FNN with a new algorithm for the activation function, Neurocomputing 71 (1–3) (2006) 71–77.
[2] S.K. Mandal, S. Sural, A. Patra, ANN- and PSO-based synthesis of on-chip spiral inductors for RF ICs, IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems 27 (1) (2008) 188–192.
[3] M. Clerc, J. Kennedy, The particle swarm-explosion, stability and convergence in a multidimensional complex space, IEEE Transactions on Evolutionary Computation 6 (1) (2002) 58–73.
[4] F.A. Guerra, L. dos, S. Coelho, Multi-step ahead nonlinear identification of Lorenz's chaotic system using radial basis neural network with learning by clustering and particle swarm optimization, Chaos, Solitons & Fractals 35 (2008) 967–979.

[5] J. Zhou, Z. Duan, Y. Li, J. Deng, D. Yu, PSO-based neural network optimization and its utilization in a boring machine, Journal of Materials Processing Technology 178 (1–3) (2006) 19–23.
[6] C.D. Bocaniala, J.S. da Costa, Application of a novel fuzzy classifier to fault detection and isolation of the DAMADICS benchmark problem, Control Engineering Practice 14 (6) (2006) 653–669.
[7] S.P. Ghoshal, Optimizations of PID gains by particle swarm optimizations in fuzzy based automatic generation control, Electric Power Systems Research 72 (3) (2004) 203–212.
[8] J. Martinez, F.J. Toledo, E. Fernandez, J.M. Ferrandez, A retinomorphic architecture based on discrete-time cellular neural networks using reconfigurable computing, Neurocomputing 71 (4–6) (2008) 766–775.
[9] M. Krips, T. Lammert, A. Kummert, FPGA implementation of a neural network for a real time hand tracking system, in: Proceedings of the First IEEE International Workshop on Electronic Design, Test and Applications, 2002, pp. 313–317.
[10] H. Ossoinig, E. Reisinger, C. Steger, R. Weiss, Design and FPGA-implementation of a neural network, in: Proceedings of the 7th International Conference on signal Processing Applications & Technology, Boston, USA, 1996, pp. 939–943.
[11] S. Sahin, Y. Becerikli, S. Yazici, Neural network implementation in hardware using FPGAs, Lecture Notes in Computer Science 4234 (2006) 1105–1112.
[12] J. Zhu, G.J. Milne, B.K. Gunther, Towards an FPGA based reconfigurable computing environment for neural network implementations, in: Proceedings of the Ninth International Conference on Artificial Neural Networks (ICANN'99) IEE Conference Proceedings 470, Edinburgh, UK, 1999, pp. 661–666.
[13] M. Mousa, S. Areibi, K. Nichols, On the arithmetic precision for implementing back-propagation networks on FPGA: a case study, in: R. Amos, Omondi, C. Jagath, Rajapakse (Eds.), FPGA Implementations of Neural Networks, Springer, USA, 2006, pp. 37–61.
[14] N. Nedjah, R.M.D. Silva, L.M.M. Mourelle, M.V.C.D. Silva, Dynamic MAC-based architecture of artificial neural networks suitable for hardware implementation on FPGAs, Neurocomputing 72 (10–12) (2009) 2171–2179.
[15] E. Won, A hardware implementation of artificial neural networks using field programmable gate arrays, Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 581 (3) (2007) 816–820.
[16] D. Ferrer, R. Gonzalez, R. Fleitas, J.P. Acle, R. Canetti, NeuroFPGA-implementing artificial neural networks on programmable logic devices, in: Proceedings of Design, Automation and Test in Europe Conference and Exhibition, vol. 3, 2004, pp. 218–223.
[17] M.A. Çavuşlu, C. Karakuzu, S. Şahin, Neural network hardware implementation using FPGA, in: Proceedngs of ISEECE 2006 3rd International Symposium on Electrical, Electronic and Computer Engineering Symposium, Nicosia, TRNC, 2006, pp. 287–290.
[18] A.W. Savich, M. Moussa, S. Areibi, The impact of arithmetic representation on implementing MLP-BP on FPGAs: a study, IEEE Transactions on Neural Networks 18 (1) (2007) 240–252.
[19] M.A. Çavuşlu, C. Karakuzu, S. Şahin, M. Yakut, Neural network training based on FPGA with floating point number format and its performance, Neural Computing & Application 20 (2) (2011) 195–202.
[20] A. Farmahini-Farahani, S.M. Fakhraie, S. Safari, Scalable architecture for on-chip neural network training using swarm intelligence, in: Proceedings of Design, Automation and Test in Europe Conference (DATE'08), Munich, Germany, 2008, pp. 1340–1345.
[21] M. Brysbaert, Algorithms for randomness in the behavioral sciences: a tutorial, Behavior Research Methods, Instruments & Computers 23 (1991) 45–60.
[22] I. Az, S. Şahin, C. Karakuzu, M.A. Çavuşlu, Implementation of FFT and IFFT algorithms in FPGA, in: Proceedings of ISEECE 2006 3rd International Symposium on Electrical, Electronic and Computer Engineering Symposium, Nicosia, TRNC, 2006, pp. 7–10.
[23] D.R. Tveter, Backpropagator's review, 1998, from http://www.dontveter.com/bpr/activate.html (accessed 28.05.10).
[24] D.L. Elliot, A better activation function for artificial neural networks, Technical Research Report T.R. 93-8, Institute for Systems Research, University of Maryland, 1993.
[25] K.S. Narendra, K. Parthasarathy, Identification and control of dynamical systems using neural networks, IEEE Transactions on Neural Networks 1 (1) (1990) 4–27.
[26] Y. Oussar, I. Rivals, L. Personnaz, G. Dreyfus, Training wawelet networks for nonlinear dynamic input–output modeling, Neurocomputing 20 (1998) 173–188.
[27] P.S. Sastry, G. Santharam, K.P. Unnikrishnan, Memory neuron networks for identification and control of dynamical systems, IEEE Transactions on Neural Network 5 (2) (1994) 306–319.
[28] R. Babuŝka, Fuzzy Systems, Modeling and Identification, from http://www.dcsc.tudelft.nl/∼babuska/transp/fuzzmod.pdf (accessed 15.12.11).
[29] D. Karaboga, B. Akay, A survey: algorithms simulating bee swarm intelligence, Artificial Intelligence Review 31 (1) (2009) 68–85.
[30] Artificial Bee Colony (ABC) Algorithm Homepage, http://mf.erciyes.edu.tr/abc/software.htm (accessed 15.12.11).