Neural Networks 79 (2016) 128–140

Contents lists available at ScienceDirect

Neural Networks

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

FPGA implementation of neuro-fuzzy system with improved PSO learning

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

[a] Bilecik Şeyh Edebali University, Faculty of Engineering, Department of Computer Engineering, Gülümbe Campus, 11210, Bilecik, Turkey
[b] Niğde University, Faculty of Engineering, Department of Electrical and Electronics Engineering, Central Campus, 51240, Niğde, Turkey
[c] Koc Information and Defence Technologies Inc., METU Technopolis, ODTÜ-Teknokent, 06800 Ankara, Turkey

HIGHLIGHTS

- A novel approach for implementation of neuro-fuzzy system on FPGA is proposed.
- Hardware implementation includes meta-heuristic learning of the system.
- A novel membership function implementation approach is also presented.
- The proposed approaches have been experimentally tested for both benchmark and practical problems.
- The obtained results show that the proposed method is effective and acceptable in handling various types of real life problems.

ARTICLE INFO

Article history:
Received 18 August 2015
Received in revised form 20 December 2015
Accepted 8 February 2016
Available online 8 April 2016

Keywords:
Neuro-fuzzy network
FPGA implementation
VHDL
System identification
Metaheuristic learning

ABSTRACT

This paper presents the first hardware implementation of neuro-fuzzy system (NFS) with its metaheuristic learning ability on field programmable gate array (FPGA). Metaheuristic learning of NFS for all of its parameters is accomplished by using the improved particle swarm optimization (iPSO). As a second novelty, a new functional approach, which does not require any memory and multiplier usage, is proposed for the Gaussian membership functions of NFS. NFS and its learning using iPSO are implemented on Xilinx Virtex5 xc5vlx110-3ff1153 and efficiency of the proposed implementation tested on two dynamic system identification problems and licence plate detection problem as a practical application. Results indicate that proposed NFS implementation and membership function approximation is as effective as the other approaches available in the literature but requires less hardware resources.

1. Introduction

Neuro-fuzzy system (NFS) is an intelligent technique for system control (Oğuz & Güney, 2010), identification/modelling and prediction (Chen, 2013). It is generally trained by algorithms based on gradient. But these algorithms have disadvantages such as needing complex gradient computations and getting stuck at a local minima.

Particle swarm optimization algorithm (PSO) (Kennedy & Eberhart, 1995), inspired by behaviour of animal swarms, has stochastic global search feature and it has been successfully used for neuro-fuzzy (Chen, 2013; Ghomsheh, Shoorehdeli, & Teshnehlab, 2007) and neural network (Çavuslu, Karakuzu, & Karakaya, 2012) training to solve nonlinear problems. Field programmable gate arrays (FPGAs), capable of parallel processing, have been extensively used in real time applications with heavy computational load (Martinez, Toledo, Fernandez, & Ferrandez, 2008). Recently, FPGAs gain more and more importance as a preferable embedded system platform in many neural, fuzzy and neuro-fuzzy applications (Abdelmoula, Rouabeh, & Masmoudi, 2012; Blake & Maguire, 1998; Campo, Echanobe, Bosque, & Tarela, 2008; Çavuslu et al., 2012; Çavuslu, Karakuzu, & Sahin, 2006; Çavuslu, Karakuzu, Şahin, & Yakut, 2010; Chou, Kung, Vu Quynh, & Cheng, 2013; Echanobe, del Campo, & Bosque, 2008; Glackin, Maguire, & McGinnity, 2004; Hima, Anitha, & Jegan, 2012; Huang, Pan, Zhou, & Chang, 2014; Juang & Hsu, 2005; Lin & Lee, 2009; Lin & Tsai, 2008; McKenna & Wilamowski, 2001; Pande, Paikrao, Chaudhari, 2013; Sanchez-Solano, Cabrera, Baturone, Moreno-Velo, & Brox, 2007; Tamas & Brassai, 2015).

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

http://dx.doi.org/10.1016/j.neunet.2016.02.004
0893-6080/

Table 1
Comparison table of this study and the related literature.

Ref.	NFS type	Training type	Training algorithm	Membership function type/approximation	Number format
Blake and Maguire (1998)	RBFN	Off-line	–	Gaussian function approximation	Fixed point
Glackin et al. (2004)	ANFIS, DFNN	Off-line	–	Functional approximation given in Jang (1993)	Fixed point
Echanobe et al. (2008)	ANFIS	Off-line	–	Triangular	Fixed point
Tamas and Brassai (2015)	NFC	Off-line	–	Functional approximation given in Jang (1993)	Integer
Huang et al. (2014)	ANFIS	Off-line	Backpropagation algorithm	Triangular	–
Pande et al. (2013)	ANFIS	Off-line	–	Triangular	Fixed point
Abdelmoula et al. (2012)	ANFIS	Off-line	–	LUT	Floating point
Chou et al. (2013)	RBF	Off-line	–	Triangular and description of exponential function using Taylor series	Signed number
Juang and Hsu (2005)	RFC	Semi-trained on FPGA	Online gradient descent learning	LUT	–
Hima et al. (2012)	ANFIS	On FPGA	Error correction	Second order approximation	Floating point
Campo et al. (2008)	ANFIS	On FPGA	LSE and gradient descent	Triangular	–
Lin and Lee (2009)	RNFN	On FPGA	Modified simultaneous perturbation	Gaussian function approximation	Fixed point
Lin and Tsai (2008)	WNN	On FPGA	PSO	LUT approximation based on Taylor series	Fixed point
This study	NFS	On FPGA	Improved PSO	Functional approximation given in Jang (1993) and new proposed function approximation	Floating point

In the literature, some studies related to implementation of neuro-fuzzy systems on FPGA were presented. These studies can be divided into two groups: implementation of neuro-fuzzy networks with offline training (Abdelmoula et al., 2012; Blake & Maguire, 1998; Chou et al., 2013; Echanobe et al., 2008; Glackin et al., 2004; Huang et al., 2014; Pande et al., 2013; Tamas & Brassai, 2015) and with online training (Campo et al., 2008; Hima et al., 2012; Juang & Hsu, 2005; Lin & Lee, 2009; Lin & Tsai, 2008).

In Blake and Maguire (1998), Blake et al., RBFN type neuro-fuzzy network's parameters have been adjusted by training in a software environment and then RBFN has been implemented on FPGA with second order nonlinear Gaussian function approximation and fixed point number format representation. Glackin et al. (2004) implemented ANFIS and DFNN networks, trained in a software environment, on FPGA. They used a functional approximation given in Jang (1993) for Gaussian membership functions (MFs) and fixed point number format for data representation. Echanobe et al. (2008) implemented offline trained ANFIS with triangular MFs on FPGA using fixed point number format. Tamas and Brassai (2015) implemented a neuro-fuzzy controller Intellectual Property (IP) core. They used a functional approximation given in Jang (1993) for Gaussian membership functions (MFs) and integer number format for data representation. Huang et al. (2014) implemented ANFIS on FPGA and they used triangular membership functions to keep stable the temperature in the chamber. Pande et al. (2013) implemented an ANFIS trained off-line beforehand in MATLAB environment. They used triangular membership function and different sizes fixed point number format. Abdelmoula et al. (2012) implemented two intelligent control approaches on FPGA. The first one was based on fuzzy logic and the other one was based on hybrid-type neuro-fuzzy ANFIS. They used look up table for member functions implementation and floating point number format for the two approaches. Chou et al. (2013) implemented RBF using triangular membership function and Taylor series for activation function approach in NN layer. They used signed data format in the implementation. Juang and Hsu (2005) designed a recurrent fuzzy controller (RFC) for temperature control system using a hybrid evolutionary method including Simplex method and PSO in a software environment, then they implemented this design adding/adapting it to online gradient descent learning for only consequent parameters on FPGA and look-up-table (LUT) method was executed for Gaussian MFs. This study can be considered as hardware implementation of semi-trained neuro-fuzzy network on FPGA. Hima et al. (2012) implemented ANFIS system and trained by error correction algorithm on FPGA. They used floating point number format and the second order approximation approach for Gaussian membership function. Campo et al. (2008) implemented piecewise multi linear ANFIS structure using triangular MFs together with an online hybrid learning algorithm (forward pass: the consequent parameters are updated by LSE, backward pass: the antecedent parameters are updated by gradient descent). Lin and Lee solved time sequence prediction and dynamic system identification problems in Lin and Lee (2009) using recurrent neural fuzzy network (RNFN) together with its "modified simultaneous perturbation method" learning on FPGA. They used fixed point number format for data representation and an approximation proposed in Blake and Maguire (1998) for implementing Gaussian MFs. In Lin and Tsai (2008), Lin and Tsai showed implementation of a wavelet neural network (WNN) and its learning using PSO on FPGA for prediction of a chaotic signal and dynamic system identification problem. They used fixed point number format for data representation and LUT approximation based on Taylor series for the network's nonlinear activation functions. For better comparison, the properties of the implementations available in the literature and the properties of the proposed one are listed in Table 1.

In this paper NFS and its iPSO (Çavuslu et al., 2012) learning module is implemented on FPGA using single precision (32 bit) floating point number format and it is tested on dynamic system identification and licence plate localization problems. Two different membership function implementation approaches are used. The first approach is taken from Jang (1993). This approach needs adder, multiplier and a divider arithmetic operations for Gaussian MF implementation on FPGA. Second approach, proposed in this paper, is one of the novel contributions of this study. This approach does not need any multiplier or memory compared to look-up table method. This study has been implemented on Xilinx Virtex 5 xc5vlx110-3ff1153 FPGA.

Rest of the paper is organized as follows: Section 2 introduces NFS architecture and its learning using iPSO. In Section 3, FPGA implementation details of NFS and its learning by means of iPSO are given. Performance of the work is tested based on three examples and results are given in Section 4. And lastly, Section 5 presents the conclusion.

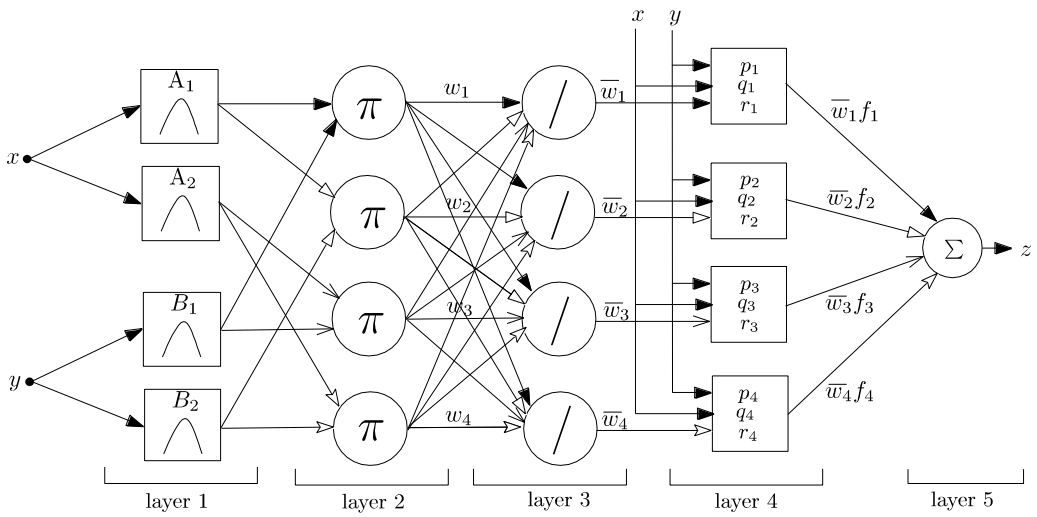

Fig. 1. NFS structure used in this study.

2. Network structure and its learning method

The system proposed in this study is a Sugeno-type fuzzy inference system in five layers neural network structure trained by iPSO algorithm. This structure is commonly known as ANFIS (Jang, 1993). It is a kind of artificial neural network that is based on Sugeno fuzzy inference system. Since it integrates both neural networks and fuzzy logic it could be called as neuro-fuzzy system (NFS) and it has the potential to gather advantages of both techniques in a single network structure. Its inference system has learning capability for modelling nonlinear systems. For efficient and optimal usage of NFS, heuristic algorithm could be used for parameter tuning. In this study we used iPSO for this purpose.

2.1. Implemented NFS architecture

Implemented NFS architecture has two inputs and four rules, which is equivalent to Sugeno type fuzzy inference system, and its block diagram is shown in Fig. 1. Definitions of the nodes in each layer are briefly given in the following paragraph.

The first layer is the layer where fuzzyfication is done using Gaussian MFs. Each node in this layer (A_i and B_i) represents a MF defined by Eq. (1) or Eq. (2) (Jang, 1993) for the relevant input. In the equations, m and σ are the centre and standard deviation of a Gaussian MF.

$$\mu_{A_i}(x) = \frac{1}{1 + \left|\frac{x - m_{A_i}}{\sigma_{A_i}}\right|^2} \tag{1}$$

$$\mu_{B_i}(y) = e^{-\left(\frac{y - m_{B_i}}{\sigma_{B_i}}\right)^2}. \tag{2}$$

In the second layer, each node labelled with π performs algebraic product T-norm operator to determine strength of a rule firing. The nodes labelled with "/" in the third layer compute weighted rule firings receiving all rule firings via full connected links between layer 2 and layer 3. The output of this layer is called normalized firing strength and it is computed by the first term of Eq. (3). Contributions of each rule to the output are computed by Eq. (3) in layer 4. In the equation, p_i, q_i and r_i are rule or consequent parameters of ith rule. Finally, weighted average defuzzyfication is performed and a crisp output is obtained in layer 5 by Eq. (3).

$$\bar{w}_i f_i = \frac{w_i}{\sum_k w_k}(p_i x + q_i y + r_i). \tag{3}$$

2.2. Particle swarm optimization (PSO) algorithm

PSO is a common heuristic optimization technique inspired from social interactive behaviour of some species of animals e.g. birds flocking to a food source (Bratton & Kennedy, 2007). Algorithm directs its individuals to the most significant area in the solution space using social interactions. Algorithm is started with individuals (particles) assigned at random. In each iteration, performance of each particle as a possible solution is determined by computing the fitness value. Algorithm keeps in its mind a local best particle and a global best particle found so far for the swarm. Based on local and global particles, position changes (particle velocities) and positions of individuals in the space are updated in the actual iteration. There are different methods for updating particle's velocity. For this purpose, we used the method (Çavuslu et al., 2012) given by Eq. (4) where p_i, $p_{lb,i}$ and p_{gb} denote respectively ith particle, its local best and the global best particles; α_1, α_2 and α_3 are learning constants; r_1 and r_2 are uniformly distributed random numbers in the interval of (0−1); ζ is restriction factor and λ is a normal distributed random number vector having the same dimension as particles. The last term in Eq. (4) allows for a more detailed search preventing the particles getting stuck in a local minimum or premature convergence. For the new generation, particles are updated by Eq. (5).

$$\begin{aligned} v_i(n+1) &= \zeta\left[v_i(n) + \alpha_1 r_1(p_{lb,i} - p_i(n)) \right. \\ &\quad \left. + \alpha_2 r_2(p_{gb} - p_i(n))\right] + [\alpha_3 \lambda(n)] \end{aligned} \tag{4}$$

$$p_i(n+1) = p_i(n) + v_i(n+1). \tag{5}$$

3. NFS learning based-PSO on FPGA

In this section, learning of ANFIS using iPSO is presented in detail. Fig. 2 shows block diagram of the learning process. As seen in the figure, iPSO is responsible for the optimization of both premise (antecedent) and rule (consequent) parameters. Training is performed in four stages as shown in Fig. 3.

3.1. Stage 1: Assigning initial parameter values

For the FPGA implementation of NFS learning; $N \times D$ dimensional matrices of swarm (P), local best particles (P_b), particle velocity (V_m) and $1 \times D$ dimensional global best particle (g_b) vector, $1 \times N$ dimensional fitness value (E_n) vector are initially created in the block RAMs of FPGA. N and D represent swarm

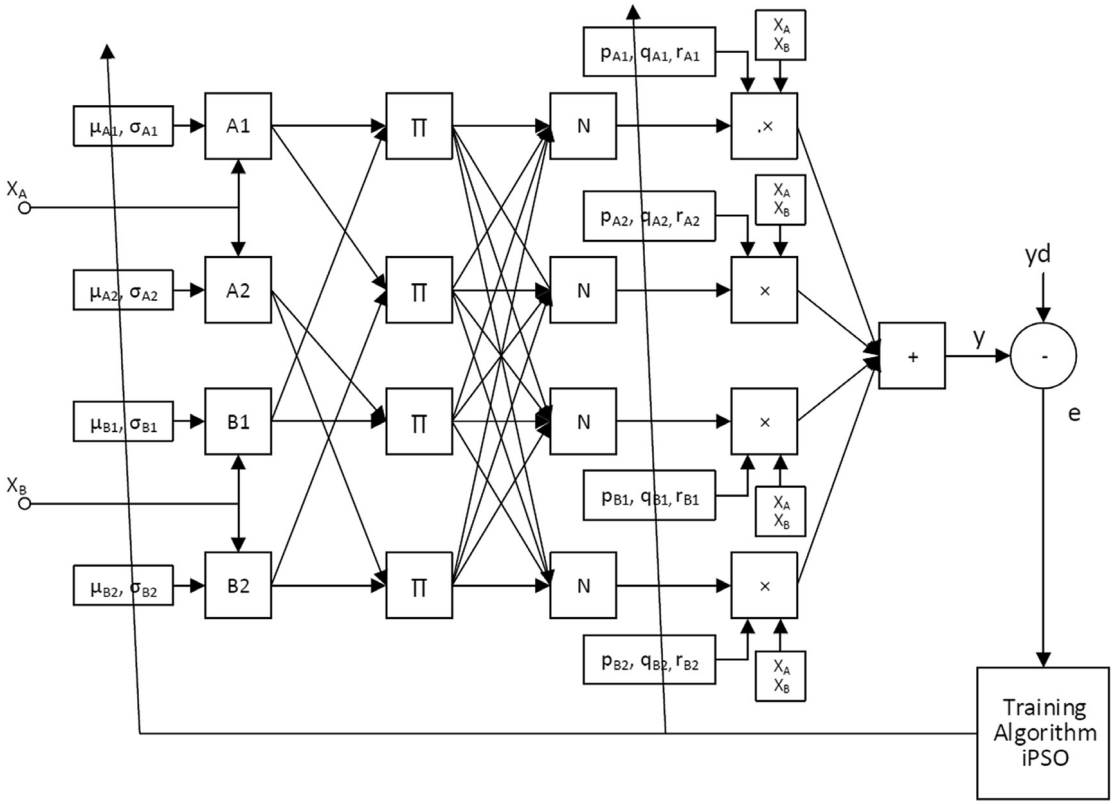

Fig. 2. Block structure of NFS' parameters training using iPSO.

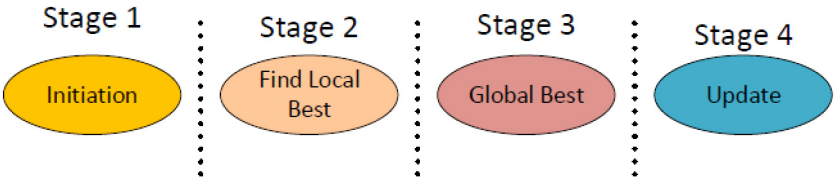

Fig. 3. Order of the implementation with 4 stages.

size (the number of particle) and the number of parameters to be optimized respectively. Depth of block RAMs is single precision (32 bits) which has the same bit length as the number format utilized.

Initial values of particles' elements and their velocities are randomly assigned in the interval of [0−1) by Eq. (6) (Brysbaert, 1991). Based on this method, random number generation is started with the initial (X_0) seed value. a and b are the constant numbers. The last term in Eq. (6) is modular operation of c constant.

$$X_{n+1} = (aX_n + b) \bmod (c). \tag{6}$$

The number produced randomly is written into the related RAM block according to state of the *out_flg* signal which checks whether the related RAM block is full or not. At the beginning, the values in P_RAM and Pbest_RAM are the same. This stage operations can be summarized with a block diagram given in Fig. 4. Memory checking transaction is done by the VHDL codes given in Fig. 5.

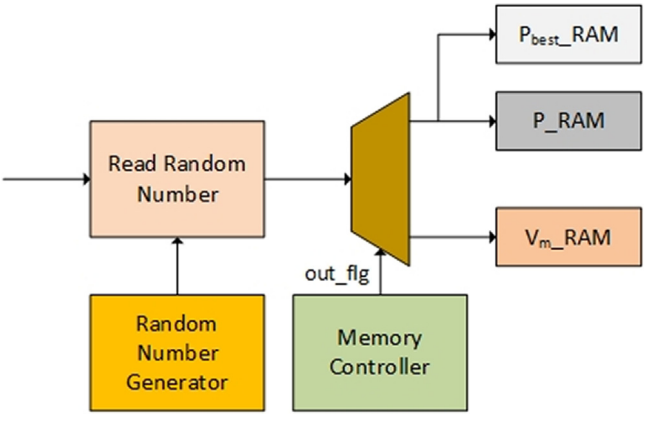

Fig. 4. Block diagram of the stage 1.

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

Particles' elements are read as NFS parameters from the P_RAM and then NFS is structured with these parameters. NFS output is calculated for each entry respectively taken from the P_RAM and error between this output and desired output is determined. Fitness value for each particle is calculated based on these errors by repeating this action for the all training data. A particle is assigned as its local best in case of its fitness value which is better than the values calculated in the previous iterations. These transactions are done for all particles in the swarm as shown in Fig. 6.

In this stage, the most important transaction is the NFS implementation. This is hardware implementation of NFS. To realize this implementation, an arithmetic operation library developed and coded in VHDL language in our previous studies

```
case Memory_Cntrl is
    when SET_P_RAM =>
            out_flg <= "01";
            if n_i = P_RAM_LENGTH - 1 then
                  n_i <= 0;
                  Memory_Cntrl <=  SET_Vm_RAM;
            else
                  n_i<= n_i + 1;
            end if;

      when SET_Vm_RAM =>
            out_flg<= "10";
            if n_i = Vm_RAM_LENGTH - 1 then
                  n_i<= 0;
                  Memory_Cntrl<=  DONE;
            else
                  n_i<= n_i + 1;
            end if;

      when DONE = > NULL;
end case;
```

Fig. 5. Memory checking VHDL codes.

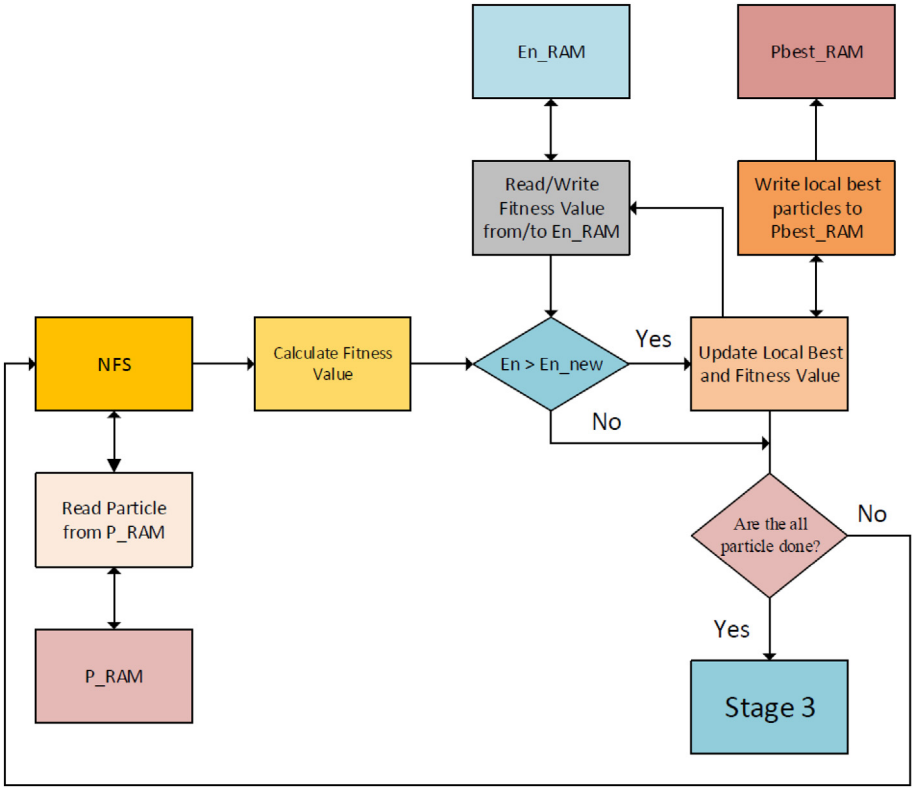

Fig. 6. Block diagram of the stage 2.

(Az, Şahin, Karakuzu, & Çavuşlu, 2006; Çavuslu, Dikmese, Şahin, Küçük, & Kavak, 2006; Çavuslu et al., 2012; Çavuşlu, Karakuzu et al., 2006; Çavuşlu et al., 2010) has been used for NFS' arithmetic operations on FPGA.

One of the important steps for hardware implementation of NFS is implementing the Gaussian MFs since it includes exponential function and exponential function cannot be directly realized on FPGA. In this study, for this purpose, functional approximations given in Eq. (1) (Jang, 1993) and our proposed one in Eq. (7) are used. The Gaussian MF approximation given in Eq. (7) requires three adders and two dividers modules. Since the time duration needed for division operation is more than multiplication, Eq. (7) can be rearranged to Eq. (8) including only one division operation where q is $1/\sigma$.

Eq. (8) can be implemented on the hardware using three adders, one multiplier and one divider modules. Eq. (8) can be rearranged

Table 2
Number of arithmetic modules used for the MF approximations on the hardware.

Approximation	Adder	Multiplier	Divider
Eq. (1)	2	1	2
Eq. (9)	3	–	1
Eq. (10)	2	2	1

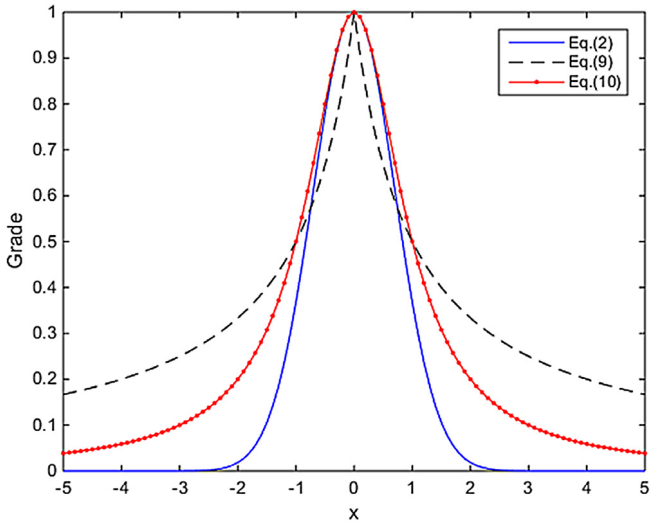

Fig. 7. Comparison of the two approximations for a Gaussian MF.

to Eq. (9) as q is always positive. Hence, our approximation of Gaussian MF has become realizable with only three adders and one divider modules as given in Table 2. Eq. (10) is another expression of Eq. (1) and it is used instead of Eq. (1) in this study. In this study, both Eqs. (9) and (10) have been used for the MF implementation on FPGA. Comparison of the MFs defined in Eqs. (2), (9) and (10) is shown in Fig. 7 for the parameters of $m = 0$ and $\sigma = 1$.

$$\mu_{A_i}(x) = \begin{cases} 1 + \dfrac{\frac{x - m_{A_i}}{\sigma_{A_i}}}{1 + \left|\frac{x - m_{A_i}}{\sigma_{A_i}}\right|}, & x < m_{A_i} \\ 1 + \dfrac{\frac{m_{A_i} - x}{\sigma_{A_i}}}{1 + \left|\frac{m_{A_i} - x}{\sigma_{A_i}}\right|}, & x \geq m_{A_i} \end{cases} \tag{7}$$

$$\mu_{A_i}(x) = \begin{cases} 1 + \dfrac{(x - m_{A_i})q}{1 + \left|(x - m_{A_i})q\right|}, & x < m_{A_i} \\ 1 + \dfrac{(m_{A_i} - x)q}{1 + \left|(m_{A_i} - x)q\right|}, & x \geq m_{A_i} \end{cases} \tag{8}$$

$$\mu_{A_i}(x) = \begin{cases} 1 + \dfrac{(x - m_{A_i})}{\sigma_{A_i} + \left|(x - m_{A_i})\right|}, & x < m_{A_i} \\ 1 + \dfrac{(m_{A_i} - x)}{\sigma_{A_i} + \left|(m_{A_i} - x)\right|}, & x \geq m_{A_i} \end{cases} \tag{9}$$

$$\mu_{A_i}(x) = \frac{\sigma_{A_i}^2}{\sigma_{A_i}^2 + (x - m_{A_i})^2}. \tag{10}$$

To calculate fitness values of particles, sum square of error (SSE) cost function given in Eq. (11) is used. In the equation, j denotes sample index and e_j represents error between jth desired output and actual output of NFS for jth training input data. This cost function is used to compute fitness value of a particle of PSO swarm. The computation of this fitness value is done using VHDL code given in Fig. 8.

$$E_n = \frac{1}{2}\sum_j e_j^2. \tag{11}$$

3.3. Stage 3: determination of the global best particle

After fitness values for all particles are calculated and local best particles are updated in P_b, the best particle is determined among them. The row vector of P_b pointed by index of minimum element of E_n is assigned as the global best particle.

3.4. Stage 4: swarm updating

Updating of the swarm is done using P, P_b, V_m and g_b taken from the RAM at the same instant. Velocity updating is executed with the parameters of $\zeta = 0.76$, $\alpha_1 = \alpha_2 = 2.1$. While updating velocity values have been restricted in the interval of $[-1\ 1]$ and values of the particles' elements have been restricted in the intervals of $[-2.5\ 2.5]$, $[0.1\ 2]$ and $[-100\ 100]$ for MFs' centres, MFs' standard deviations and rule parameters respectively. After the swarm (P) and its velocity (V_m) are updated, the new particles and their velocities are written to RAM at the same instant. These operations are repeated until iteration number is reached to maximum generation number ($G_{\max}$) as shown in Fig. 9. VHDL codes of the "*Restriction*" block in Fig. 9 are given in Fig. 10.

```
case Fitness_Value_Cont is
      when CALC_ERROR =>
            error <= add(desired_output, negative(nfs_result));
            Fitness_Value_Cont <= SQUARE_ERROR;

      when SQUARE_ERROR =>
            error_square<= mul(error, error);
            Fitness_Value_Cont<= FITNESS_VALUE;

      when FITNESS_VALUE=>
            En <= add(En, div2(error_square));
            Fitness_Value_Cont<= DONE;
      when DONE => NULL;
end case;
```

Fig. 8. VHDL codes for the computation of fitness value.

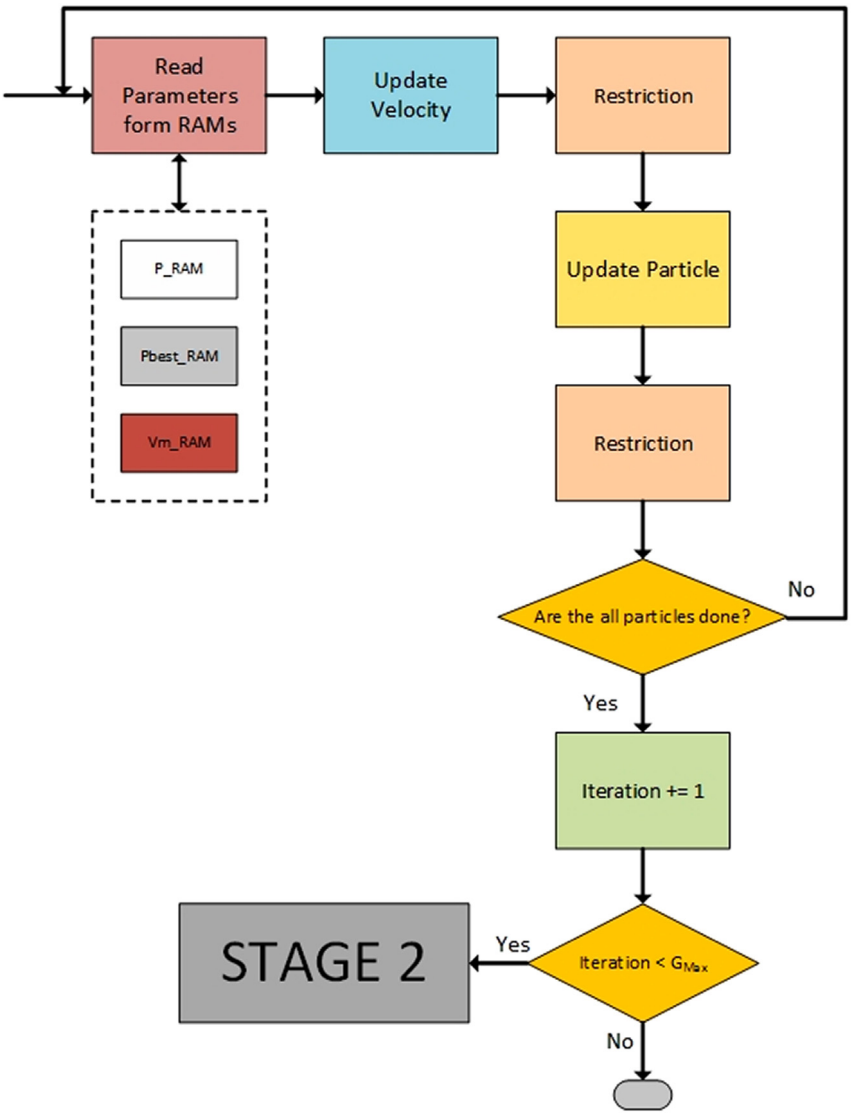

Fig. 9. Block diagram of the stage 3.

```
function restriction(Valeu, Limit : std_logic_vector(31 downto 0))
return std_logic_vector(31 downto 0) is
begin
      if Valeu(31) = '0' and Valeu > Limit then
            return(Limit);

      elsif Valeu(31) = '0' and Valeu > negative(Limit)then
            return(not(Limit(31) & Limit(30 downto 0));

      else
            return(Value)
      end if;
end restriction;
```

Fig. 10. VHDL codes of the "*Restriction*" block in Fig. 9.

4. Experimental results

Implementation of the proposed method on FPGA is experimentally tested on three examples: Two system identification problems and licence plate localization problem as a practical application. Experimental tests are done on Xilinx Virtex 5 xc5vlx110-3ff1153 using ISE Webpack 10.1 program. These problems are solved using NFS with appropriate parameters on FPGA. In this way, tuning process for NFS parameters is implied to minimize a cost function based on the error between desired output

Table 3
Synthesis results for Example 1 using Eqs. (9) and (10).

	Eq. (9)			Eq. (10)		
	Used	Available	Utilization%	Used	Available	Utilization%
Number of slice registers	5 798	69 120	8%	5 712	69 120	8%
Number of slice LUTs	39 172	69 120	56%	38 975	69 120	57%
Number of block RAM/FIFO	9	128	6%	9	128	6%
Number of BUFG/BUFGCTRLs	1	32	3%	1	32	3%
Number of DSP48Es	53	64	82%	61	64	95%

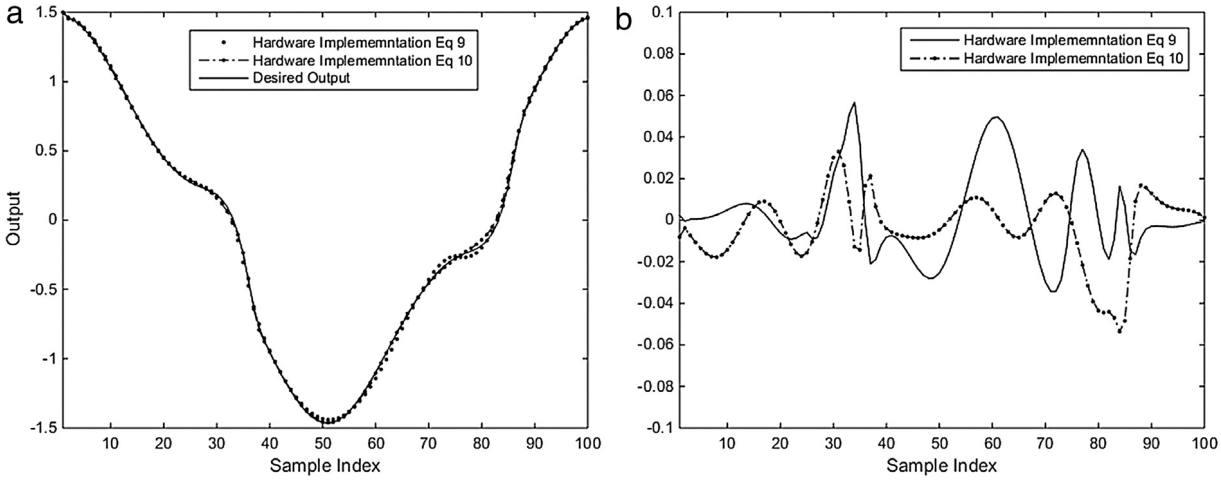

Fig. 11. Experimental results for Example 1 for the training data: (a) comparative training performance (b) identification errors.

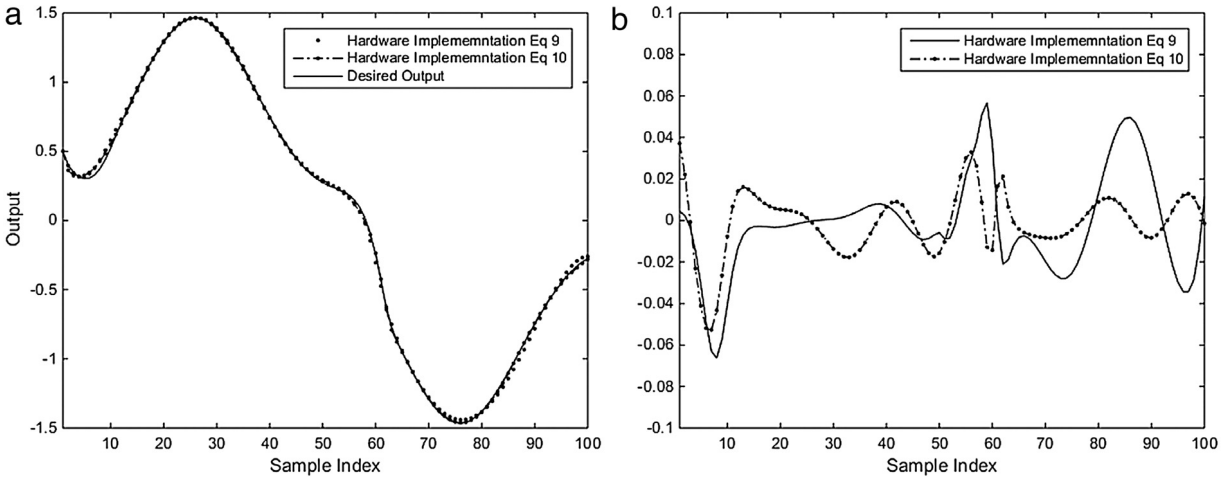

Fig. 12. Test results of the NFS trained for Example 1 (a) comparative test performance (b) identification errors.

and actual output of the NFS. For the tests, iPSO is run 1000 generation to tune NFS parameters. The iPSO algorithm in the proposed approach is run with a constriction factor of $\xi = 0.76$ and learning factors $\alpha_1 = \alpha_2 = 2.1$. α_3 value is different for each example since it depends on the training input data set as defined in Çavuslu et al. (2012). Its value is given separately for the each example.

4.1. Example 1: system identification 1

The system to be identified is defined by Eqs. (12) and (13) (Narendra & Parthasarathy, 1990). This system has been identified with an NFS with inputs of $u(k)$ and $y(k)$. Each input has been fuzzyficated with two MFs using (11), hence NFS has four rules. iPSO swarm size (N) is chosen as 100, λ vector is restricted within the interval of $[-2^{-12}\ 2^{-12}]$ and $\alpha_3 = 2^{-12}$.

$$y(k+1) = \frac{y(k)}{1+y^2(k)} + u^3(k) \tag{12}$$

$$u(k) = \cos\left(\frac{2\pi k}{100}\right) \quad k = 1, 2, 3, \ldots, 100. \tag{13}$$

Synthesis results of training for this example are listed in Table 3. Fig. 11 shows performance of trained NFS and error for the hardware identification of the system.

NFS, which was trained with the input sequence given in Eq. (13), is tested with another input sequence given in Eq. (14). Results of this test are shown in Fig. 12. Results indicate that ANFIS is well trained to make similar generalizations using both

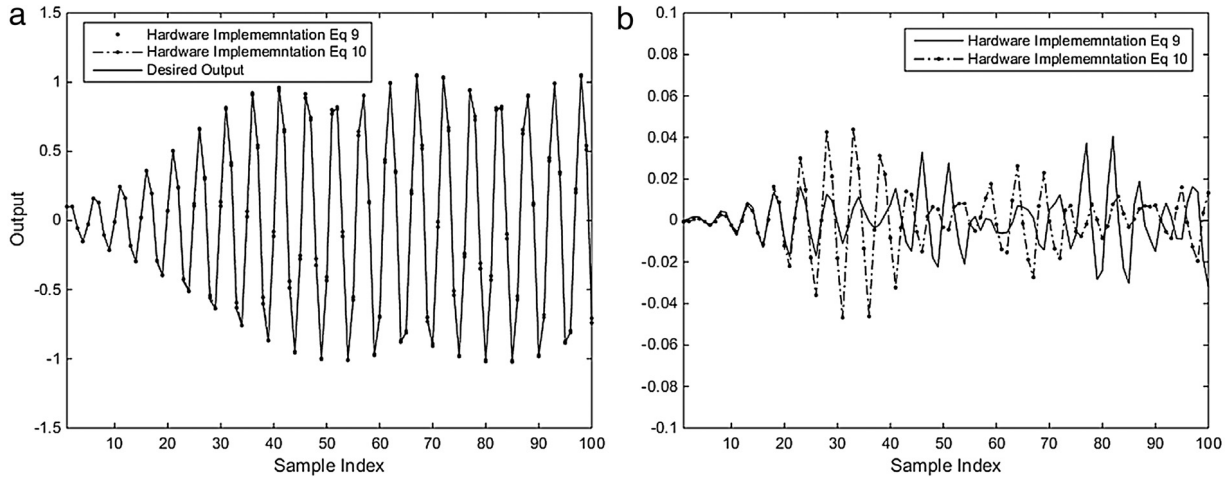

Fig. 13. Experimental results of Example 2 for the training data: (a) comparative training performance (b) identification error.

Table 4
Obtained fitness values of the global best particle using Eqs. (9) and (10) for training and test.

	Eq. (9)	Eq. (10)
Training	0.0158	0.0152
Testing	0.0285	0.0218

Eqs. (9) and (10). In Table 4, obtained fitness values of the global best particle at the end of tuning process are given.

$$u(k) = \sin\left(\frac{2\pi k}{100}\right) \quad k = 1, 2, 3, \ldots, 100. \tag{14}$$

Results given in Figs. 11, 12, Tables 1 and 4 indicate that obtained performance with proposed Gaussian MF implementation approach is satisfactory even though it is the farthest to the original Gaussian MF as seen in Fig. 7. In Table 5 the processing time for each stage of the proposed algorithm is given in terms of number of clock cycles and seconds for 100 MHz clock frequency.

4.2. Example 2: system identification 2

As the second example the system to be identified is defined by Eq. (15) (Chen & Billings, 1992). This system has been identified with an NFS which has the same structure as the one used for Example 1. $y(k)$ and $y(k-1)$ are the inputs. iPSO swarm size (N) is chosen as 100. λ vector was restricted within the interval of $[-2^{-12}\ 2^{-12}]$ and $\alpha_3 = 2^{-12}$. In the training phase, as initial condition the inputs are set to $y(0) = 0.1$ and $y(-1) = 0.1$.

$$\begin{aligned} y(k+1) = {} & -1.17059y(k-1) + 0.606861y(k) \\ & + 0.679190y^2(k)y(k-1) \\ & - 0.136235y^4(k)y(k-1) + 0.165646y^3(k)y(k-1) \\ & - 0.00711966y^6(k-1) - 0.114719y^5(k)y(k-1) \\ & - 0.0314354y(k)y(k-1) + 0.0134682y^3(k). \end{aligned} \tag{15}$$

Figs. 13 and 14 show performance of trained NFS and error for the hardware identification of the system.

NFS trained with data obtained under $y(0) = 0.1$ and $y(-1) = 0.1$ initial conditions is tested experimentally with data obtained for $y(0) = y(-1) = 0.5$, $y(0) = y(-1) = -0.1$ and $y(0) = 0.25$, $y(-1) = -0.25$ initial conditions respectively. Obtained results are given in Figs. 15–17. Performances seen in the figures show that NFS is trained to make superior generalization. For purpose of comparing, Table 6 shows the obtained fitness values of the global best particle at the end of tuning process for NFS implementation using Eqs. (9) and (10). Once again, results in Table 6 show that obtained performance with our proposed Gaussian MF implementation approach is satisfactory.

4.3. Example 3: licence plate detection

In this example, PSO-based NFS learning is implemented to detect the place of licence plate from an automobile image. Here, NFS is operated to make a decision on whether the candidate selected region of an image includes the licence plate or not. Block diagram of the decision system is shown in Fig. 18. As shown in the figure, the candidate grey image is firstly filtered by a vertical Sobel filter to highlight the edges. Then, the edges are subjected to thresholding. After thresholding, the candidate region is converted to binary. Three features are extracted from this binary image

Table 5
Processing time both in terms of clock cycles and seconds for each stage in the proposed implementation on FPGA.

	Block	Number of clock cycles required for one iteration of iPSO training (with 100 particles and 100 training samples)	Execution time for 100 MHz clock frequency (ms)
Stage 2	Read particle	2 000	0.020
	NFS	510 000	5.1000
	Fitness value	20 000	0.2
	Update locale best and fitness value	2 300	0.023
Stage 3	Determine global best	7	0.00007
Stage 4	Read parameters from RAMs	2 000	0.020
	Update parameter	700	0.007
	Write parameters to RAMs	2 000	0.020
	Total	**539 007**	**5.39007**

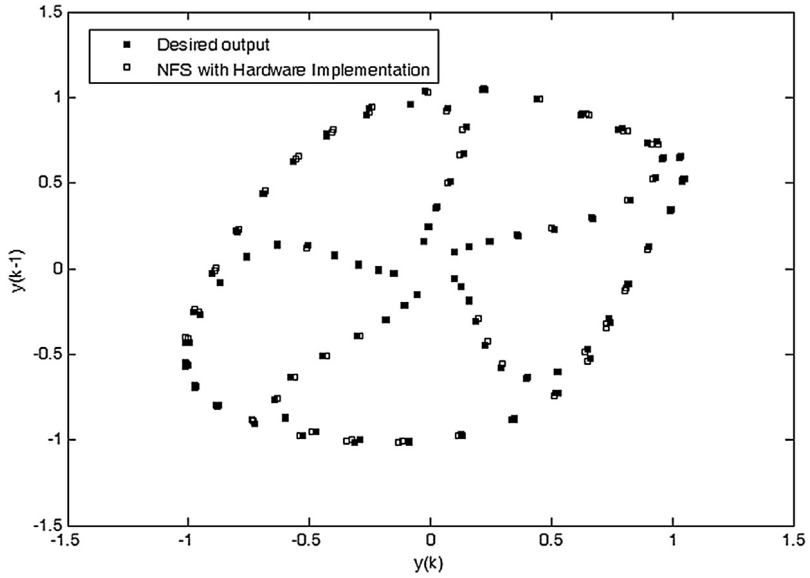

Fig. 14. Comparative experimental results in terms of phase portrait for Example 2.

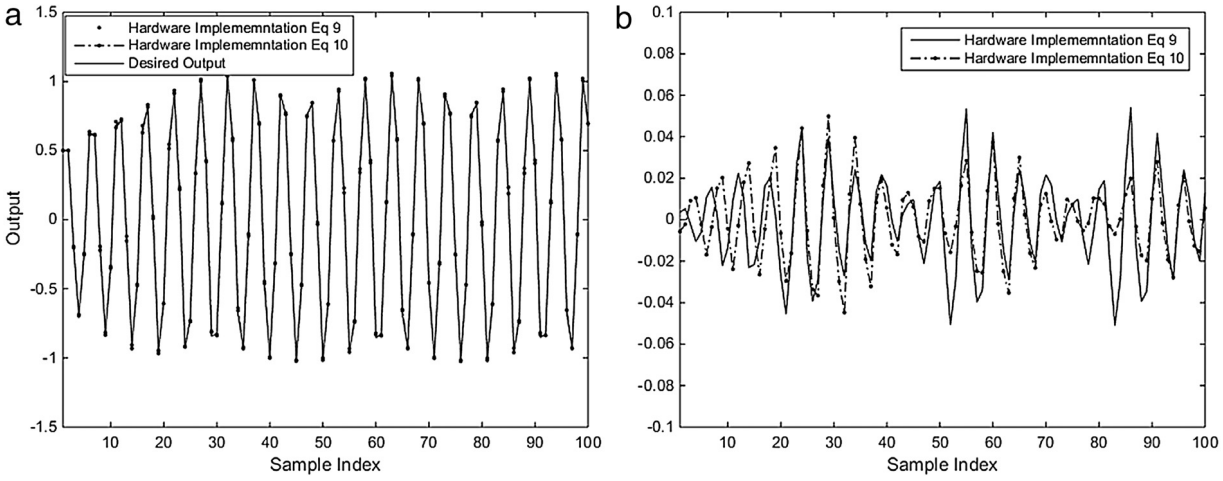

Fig. 15. Testing results for [0.5 0.5] initials: (a) comparative testing performance (b) identification errors.

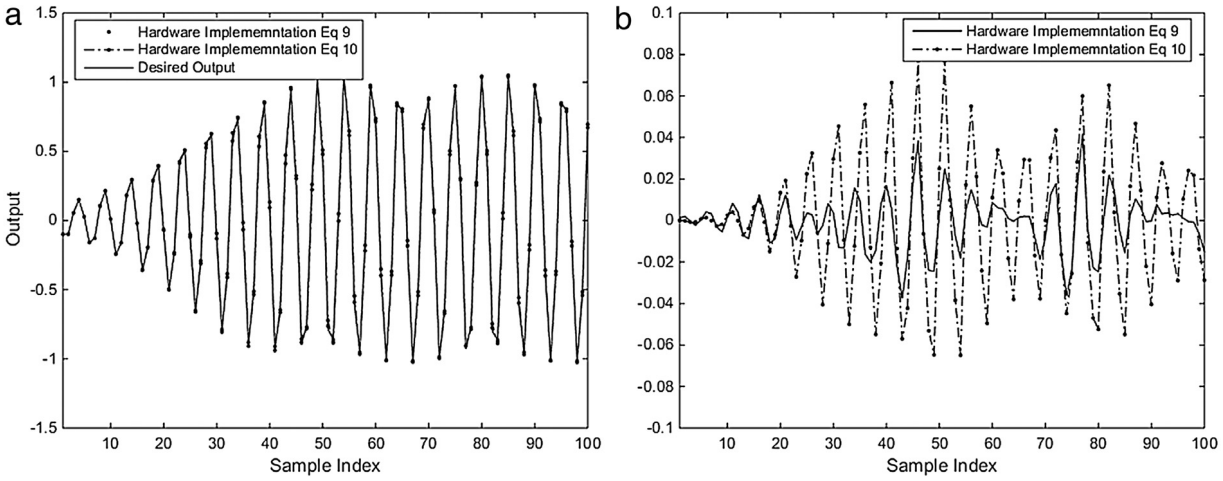

Fig. 16. Testing results for [−0.1 − 0.1] initials: (a) comparative testing performance (b) identification errors.

and they are used as inputs of NFS. These features are the mean, variance, and the statistical feature which is extracted using the method given in Çavuslu, Karakaya, and Altun (2008).

In this example, the licence plate recognition application is implemented in two stages (Fig. 19). Firstly, we carried out training stage on FPGA and saved the best particle. Secondly, we tested

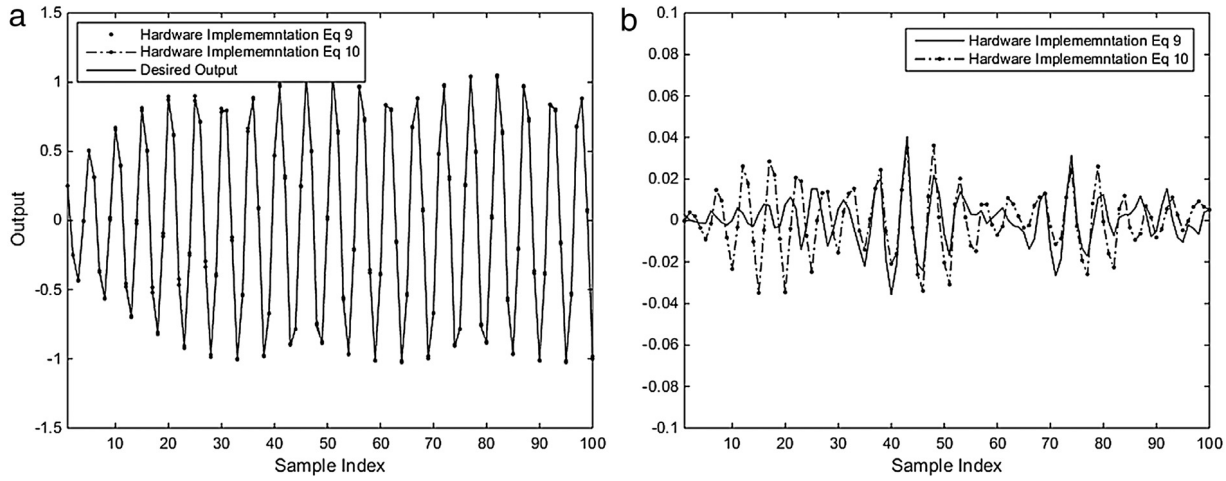

Fig. 17. Testing results for [0.25 −0.25] initials: (a) comparative testing performance (b) identification errors.

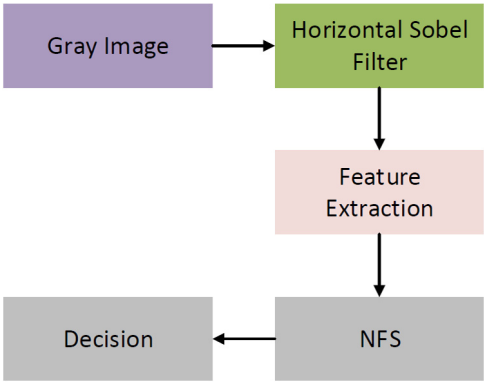

Fig. 18. Block diagram of the licence plate detection process.

Table 6
Fitness values using Eqs. (9) and (10) for training and tests.

	Fitness values	
	Eq. (9)	Eq. (10)
Training	0.0085	0.0114
Test_1	0.0250	0.0175
Test_2	0.0088	0.0515
Test_3	0.007	0.0111

licence plate image data sets on FPGA using NFS constructed with the best particle. Process step can be explained step by step as below:

- Images of the candidate plate regions on PC are transmitted to "*Data Parse*" module via Ethernet using IEEE 802.3 protocol.
- Data and commands, received by "*Data Parse*" module, are dissociated.
- If the received data is an image, it is written into an 8-bit RAM block.
- After image frame is transferred, a start signal is sent to "*Feature Extractor*" block.
- This block reads image data from RAM, applies the related transaction for feature extraction and writes the features into "*RAM*" block in 32 bit floating point number format.
- After the completion of the feature extraction "*NFS*" block is activated. "*NFS*" block configures itself using parameter values taken from the current individual of iPSO.
- Then, it reads the features from the RAM block as its training inputs, computes its output for these inputs, and sends error related to input and output data set to "*Training Algorithm*" block.
- This block activates iPSO and sends the new NFS parameters determined by iPSO to both NFS and RAM blocks.

For the each input, NFS have two membership functions as given in Eqs. (9) and (10) in its first layer. It has been designed to have six rules and trained with total of 100 image samples in which half of the images does not include complete licence plate. Some examples of training images are given in Fig. 20. Synthesis results for the training phase are summarized in Table 7. After the training, NFS model is tested with total 2500 images approximately half of them includes complete licence plate. The test results are summarized in Table 8.

5. Conclusions

Low cost, high-speed embedded implementation is crucial for real time applications. Under this point of view, FPGA-based implementation is well suited for implementing NFS which is commonly used one. In this paper, the implementation of NFS together with its iPSO learning on FPGA is presented. The proposed FPGA implementation method has four stages as mentioned in the previous sections. In addition to the NFS implementation, a novel mathematical approximation for Gaussian MFs is also proposed and presented efficient utilization of it.

The proposed method has been experimentally tested for two highly nonlinear dynamic system identification benchmark problems and one practical application example on Xilinx Virtex 5

Table 7
Synthesis results both of training and testing for Example 3.

Logic utilization	NFS with training algorithm			Full system		
	Used	Available	Utilization	Used	Available	Utilization
Number of slice registers	10 399	69 120	15%	15 048	69 120	21%
Number of slice LUTs	58 602	69 120	84%	65 736	69 120	94%
Number of block RAM/FIFO	17	128	13%	25	128	20%
Number of BUFG/BUFGCTRLs	1	32	3%	1	32	3%
Number of DSP48Es	64	64	100%	64	64	100%

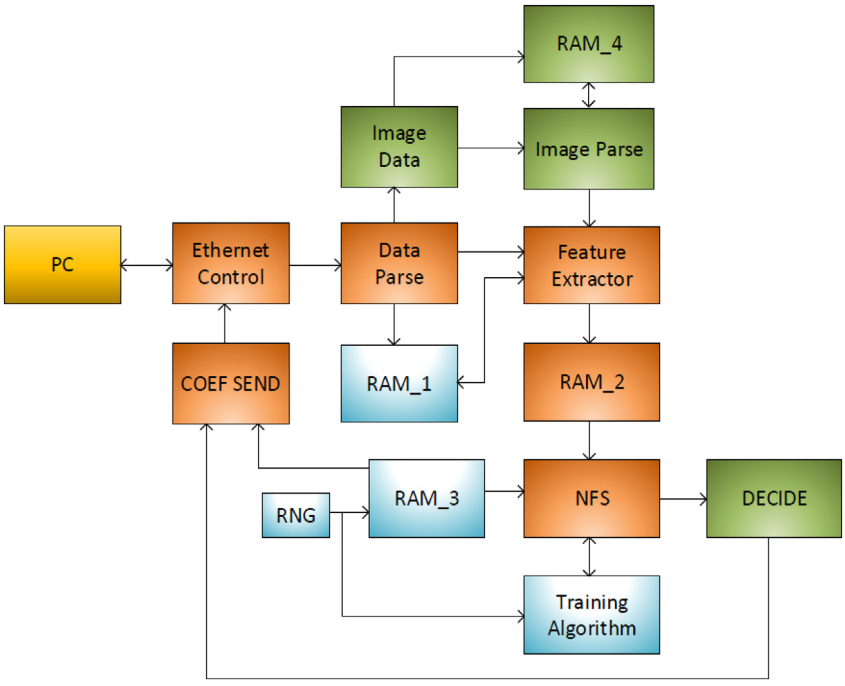

Fig. 19. Detailed block diagram of the FPGA implementation of Example 3. (Green blocks are used at test phase, Blue blocks are used at training phase and Orange blocks are used at both test and training phases.) (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

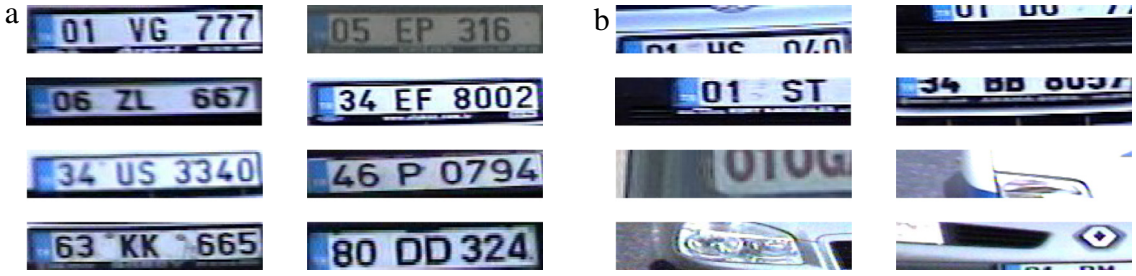

Fig. 20. Some training images: (a) including full plate (b) not including full plate.

Table 8
Test results of trained NFS using Eqs. (9) and (10) for Example 3.

Test image feature		True	False	Total	Performance%
Eq. (9)	Complete licence plate	1262	25	1287	98.06%
	Not complete licence plate	1203	10	1213	99.18%
	Total	2465	35	2500	98.60%
Eq. (10)	Complete licence plate	1248	39	1287	96.96%
	Not complete licence plate	1207	6	1213	99.50%
	Total	2455	45	2500	98.20%

Table 9
Performance comparison of iPSO.

	Fitness value (average of 10 runs)	
	Training	Test
ABC	0.6285	0.8013
PSO	0.0751	0.0898
iPSO	0.0158	0.0285

xc5vlx110-3ff1153 FPGA. Experimental results demonstrated that the proposed NFS implementation together with its iPSO learning and proposed approximation for Gaussian MFs are effective in handling various types of problems with high precision and accuracy. In order to compare the performance of iPSO in NFS training, NFS used in Example 1 is also trained using PSO and ABC (Artificial, 0000). The performance results are tabulated in Table 9 in terms of fitness value for ABC, PSO and iPSO. Results clearly indicate the superiority of the iPSO compared to ABC and PSO (for algorithms swarm size: 100, iteration: 2000, number of runs: 10).

As a summary, contributions of this paper are twofold. First, it introduces the first hardware implementation of NFS with online training using iPSO. Second, a new approach on hardware implementation of Gaussian MFs is proposed, which requires less hardware components. Test results indicate that the proposed NFS implementation is as effective as the other approaches presented in the literature.

Appendix A. Supplementary data

Supplementary material related to this article can be found online at http://dx.doi.org/10.1016/j.neunet.2016.02.004.

References

Abdelmoula, C., Rouabeh, H., & Masmoudi, M. (2012). Behavior control of a new designed mobile robot based on fuzzy logic and neuro fuzzy approaches for monitoringwall. In *2012 7th international conference on design & technology of integrated systems in nanoscale era (DTIS)* (pp. 1–6).

Artificial Bee Colony (ABC) Algorithm. http://mf.erciyes.edu.tr/abc/.

Az, I., Şahin, S., Karakuzu, C., & Çavuşlu, M. A. (2006). Implementation of FFT and IFFT algorithms in FPGA. In *Proc. ISEECE 2006 international symposium on electrical, electronic and computer engineering symposium proceedings, Nicosia, TRNC* (pp. 7–10).

Blake, J. J., & Maguire, L. P. (1998). The implementation of fuzzy systems, neural networks and fuzzy neural networks using FPGAs. *Information Sciences*, *112*, 151–168.

Bratton, D., & Kennedy, J. (2007). Defining a standard for particle swarm optimization. In *Proc. the 2007 IEEE swarm intelligence symposium* (p. 120).

Brysbaert, M. (1991). Algorithms for randomness in the behavioral sciences: A tutorial. *Behavior Research Methods, Instruments & Computers*, *23*(1), 45–60.

Campo, I., Echanobe, J., Bosque, G., & Tarela, J. M. (2008). Efficient hardware/software implementation of an adaptive neuro-fuzzy system. *IEEE Transactions on Fuzzy Systems*, *16*(3), 761–778.

Çavuşlu, M.A., Dikmese, S., Şahin, S., Küçük, K., & Kavak, A. (2006). Akıllı anten algoritmalarının IEEE 754 kayan sayı format ile FPGA tabanlı gerçeklenmesi ve performans analizi. In *Proc. URSI-TÜRKİYE' 2006 3* (pp. 610–612) (in Turkish).

Çavuşlu, Mehmet Ali, Karakaya, Fuat, & Altun, Halis (2008). Plaka Yeri Tespiti için Kenar Bulma, Bit Tabanlı Öznitelik Çıkartma ve YSA Sınıflandırıcısının FPGA Üzerine Uyarlanması, GömSis 2008 Gömülü Sistemler ve Uygulamaları Sempozyumu Bildiri Özetleri Kitabı, s. 27 (in Turkish).

Çavuslu, M. A., Karakuzu, C., & Karakaya, F. (2012). Neural identification of dynamic systems on FPGA with improved PSO learning. *Applied Soft Computing*, *12*(2012), 2707–2718.

Çavuşlu, M. A., Karakuzu, C., & Sahin, S. (2006). Neural network hardware implementation using FPGA. In *Proc. ISEECE 2006 3rd international symposium on electrical, electronic and computer engineering symposium proceedings, Nicosia TRNC* (pp. 287–290).

Çavuşlu, M. A., Karakuzu, C., Şahin, S., & Yakut, M. (2010). Neural network training based on FPGA with floating point number format and it's performance. *Neural Computing & Applications*, *20*, 195–202.

Chen, Mu-Yen (2013). A hybrid ANFIS model for business failure prediction utilizing particle swarm optimization and subtractive clustering. *Information Sciences*, *220*, 180–195.

Chen, S., & Billings, S. A. (1992). Neural networks for nonlinear dynamic system modelling and identification. *International Journal of Control*, *56*(2), 319–346.

Chou, Hsin-Hung, Kung, Ying-Shieh, Vu Quynh, Nguyen, & Cheng, Stone (2013). Optimized FPGA design, verification and implementation of a neuro-fuzzy controller for PMSM drives. *Mathematics and Computers in Simulation*, *90*, 28–44.

Echanobe, J., del Campo, I., & Bosque, G. (2008). An adaptive neuro-fuzzy system for efficient implementations. *Information Sciences*, *178*(9), 2150–2162.

Ghomsheh, V.S., Shoorehdeli, M.A., & Teshnehlab, M. (2007). Training ANFIS structure with modified PSO Algorithm. In *Proc. 15th midetaerianconfernce on control and automation*. Paper T14-003.

Glackin, B., Maguire, L. P., & McGinnity, T. M. (2004). Intrinsic and extrinsic implementation of a bio-inspired hardware system. *Information Sciences*, *161*(1–2), 1–19.

Hima, Bindu S., Anitha, Mary J., & Jegan, R. (2012). U-Tube manometer calibration using ANFIS. *International Journal of Computer Applications (0975–8887)*, *49*(20).

Huang, C., Pan, S., Zhou, J., & Chang, C. (2014). Enhanced temperature control method using ANFIS with FPGA, Hindawi Publishing Corporation The Scientific World Journal.

Jang, R. J.-S. (1993). ANFIS: adaptive-network-based fuzzy inference system. *IEEE Transactions on Systems, Man and Cybernetics*, *23*(3), 665–684.

Juang, C.-F., & Hsu, C.-H. (2005). Temperature control by chip-implemented adaptive recurrent fuzzy controller designed by evolutionary algorithm. *IEEE Transactions on Circuits and Systems. I. Regular Papers*, *52*(11), 2376–2384.

Kennedy, J., & Eberhart, R.C. (1995). Particle swarm optimization. In *Proc. IEEE int. conf. neural network IV, Vol. 4* (pp. 1942–1948) http://dx.doi.org/10.1109/ICNN.1995.488968.

Lin, C.-J., & Lee, C.-Y. (2009). FPGA implementation of a recurrent neural fuzzy network with on-chip learning for prediction and identification applications. *Journal of Information Science and Engineering*, *25*(2), 575–589.

Lin, C.-J., & Tsai, H.-M. (2008). FPGA implementation of a wavelet neural network with particle swarm optimization learning. *Mathematical and Computer Modelling*, *47*, 982–996.

Martinez, J. J., Toledo, F. J., Fernandez, E., & Ferrandez, J. M. (2008). A retinomorphic architecture based on discrete-time cellular neural networks using reconfigurable computing. *Neurocomputing*, *71*(4–6), 766–775.

McKenna, M., & Wilamowski, B.M. (2001). Implementing a fuzzy system on a field programmable gate array. In *Proc. intl. joint conf. neural networks* (pp. 189–194).

Narendra, K. S., & Parthasarathy, K. (1990). Identification and control of dynamical systems using neural networks. *IEEE Transactions on Neural Networks*, *1*(1), 4–27.

Oğuz, Y., & Güney, İ. (2010). Adaptive neuro-fuzzy inference system to improve the power quality of variable-speed wind power generation system. *Turkish Journal of the Electrical Engineering & Computer Sciences*, *18*(4), 625–645.

Pande, P. R., Paikrao, P. L., & Chaudhari, D. S. (2013). Digital ANFIS model design. *International Journal of Soft Computing and Engineering (IJSCE)*, [ISSN: 2231-2307] *3*(1).

Sanchez-Solano, S., Cabrera, A. J., Baturone, I., Moreno-Velo, F. J., & Brox, M. (2007). FPGA implementation of embedded fuzzy controllers for robotic applications. *IEEE Transactions on Industrial Electronics*, *54*(4), 1937–1945.

Tamas, T., & Brassai, S.T. (2015). Hardware implementation of a neuro-fuzzy controller using high level synthesis tool. In *5th international conference on recent achievements in mechatronics, automation, computer science and robotics.*