Available online at www.sciencedirect.com
SciVerse ScienceDirect
Mathematics and Computers in Simulation xxx (2012) xxx–xxx
MATHEMATICS AND COMPUTERS IN SIMULATION
www.elsevier.com/locate/matcom

Original article

Optimized FPGA design, verification and implementation of a neuro-fuzzy controller for PMSM drives

Hsin-Hung Chou [a], Ying-Shieh Kung [b,*], Nguyen Vu Quynh [b], Stone Cheng [a]

[a] *Department of Mechanical Engineering, National Chiao-Tung University, 1001 University Road, East District, Hsinchu City 300, Taiwan, ROC*
[b] *Department of Electrical Engineering, Southern Taiwan University, 1 Nan-Tai Street, Yong-Kang District, Tainan City 710, Taiwan, ROC*

Received 24 October 2011; received in revised form 22 June 2012; accepted 23 July 2012

Abstract

The work presents a neural fuzzy controller (NFC) for speed loop of permanent synchronous motor (PMSM) drives based on the technology of field programmable gate array (FPGA). Firstly, a mathematic model of the PMSM drive is derived; then to increase the performance of the PMSM drive system, a fuzzy controller (FC) which its parameters are adjusted by a radial basis function neural network (RBF NN) is applied to the speed controller for coping with the effect of the system dynamic uncertainty. Secondly, very high speed IC hardware description language (VHDL) is adopted to describe the behavior of the speed controller of PMSM drives which includes the circuits of space vector pulse width modulation (SVPWM), coordinate transformation, NFC, etc. Besides, to reduce the resource usage while implementing in field programmable gate array (FPGA), a sequential execution using finite state machine (FSM) is applied. Thirdly, based on electronic design automation (EDA) simulator link, a simulation work is constructed by MATLAB/Simulink and ModelSim co-simulation mode which the PMSM, inverter and speed command are performed in Simulink as well as the speed controller of PMSM drives is executed in ModelSim. Finally, some co-simulation results validate the effectiveness of the proposed NFC-based speed controller for PMSM drives.

Keywords: PMSM; Neural fuzzy control; VHDL; FPGA; ModelSim; Finite state machine; Simulink; Co-simulation

1. Introduction

PMSM has been increasingly used in many automation control fields as actuators, due to its advantages of superior power density, high-performance motion control with fast speed and better accuracy. But in industrial applications, there are many uncertainties, such as system parameter uncertainty, external load disturbance, friction force, and unmodeled uncertainty, which always diminish the performance quality of the pre-design of the motor driving system. To cope with this problem, in recent years, many intelligent control techniques [1,5,7,11,17], such as fuzzy control, adaptive PID control, neural networks control, adaptive fuzzy control and other control method, have been developed and applied to the speed control of servo motor drives to obtain high operating performance. Although fuzzy control has been successfully applied in several industrial automation, however, it is not an easy task to obtain an optimal set of fuzzy membership functions and rules in FC. In this paper, a neural fuzzy controller (NFC) is proposed which RBF

* Corresponding author.
E-mail addresses: carin6969@itri.org.tw (H.-H. Chou), kung@mail.stut.edu.tw (Y.-S. Kung), vuquynh@lhu.edu.vn (N. Vu Quynh), stonecheng@mail.nctu.edu.tw (S. Cheng).

0378-4754/$36.00 © 2012 IMACS. Published by Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.matcom.2012.07.012

NN is firstly used to real-time identify the plant dynamic (Jacobian transformation term: $(\partial\omega_r/\partial i_q^*)$) and provided more accuracy plant information; then based on the gradient descent method and the real-time identified plant information, parameters of FC can be tuned to a near-optimal condition.

In implementation, although the execution of NFC requires many computations, FPGA can provide a solution in this issue. Especially, FPGA with programmable hard-wired feature, fast computation ability, shorter design cycle, embedding processor, low power consumption and higher density is very suitable for the implementation of the digital system [14–16]. Although the digital signal processor (DSP) is another solution to provide a flexible skill in the intelligent control technique, it suffers from a long period of development and exhausts many resources of the CPU [18]. However, FPGA implementation of a RBF NN has been developed in literatures [2,6]. Brassai et al. [2] applied the RBF NN in the area of robotics and control. The computations of neurons in hidden layer of RBF NN and weight adaption module adopt typical parallel implementation on FPGA. In addition, table look up method is used to develop the activation function. Kim et al. [6] firstly develop a floating-point processor on FPGA. Then based on this floating-point processor, a microprogram is written to implement the RBF NN with on-line learning back-propagation algorithm. The Taylor series is considered to compute the Gaussian function. However, a floating-point processor consumes FPGA resources and the executing speed might be slow. Further, in the hardware realization of an intelligent control algorithm, except the parallel processing method, the sequential execution method is alternative. The former method with continuous and simultaneous operation has the advantage of fast computation ability, but consumes much more FPGA resources. The latter method separates the overall computation with several steps and the resources with same function in each step will be common use; therefore it can greatly save much FPGA resources. In this paper, a method mixed with parallel processing and sequential execution is adopted to compute the NFC algorithm. Except that the computation of neurons in the hidden layer of RBF NN is applied by the parallel processing method, others computation, such as the Gaussian function, weight adaption module and Jacobian function in RBF NN as well as the fuzzy control algorithm are all presented by the sequential execution method. Although the sequential execution method needs to spend more computation time, it does not loss any control performance due to the fast computation power in FPGA. In this paper, finite state machine (FSM), which behavior is easy to describe by VHDL, is applied to model the computation process of sequential execution method.

Recently, a co-simulation work by electronic design automation (EDA) simulator link has been gradually applied to verify the effectiveness of the Verilog and VHDL code in the motor drive system [3,4,8–10]. The EDA simulator link [12] provides a co-simulation interface between MALTAB or Simulink and HDL simulators-ModelSim [13]. Using it you can verify a VHDL, Verilog, or mixed-language implementation against your Simulink model or MATLAB algorithm [12]. Therefore, EDA simulator link lets you use MATLAB code and Simulink models as a test bench that generates stimulus for an HDL simulation and analyzes the simulation's response [12]. In this paper, a co-simulation by EDA simulator link is applied. The PMSM, inverter and speed command are performed in Simulink and the NFC-based speed controller described by VHDL code is executed in ModelSim. Finally, some simulations results validate the effectiveness of the proposed NFC-based speed controller of PMSM drives.

2. System description of PMSM drive and speed controller design

The simulation architecture of NFC-based speed control for PMSM drive is shown in Fig. 1. The modeling of PMSM and the algorithm of the neural fuzzy controller are introduced as follows.

2.1. Mathematical model of PMSM

The typical mathematical model of a PMSM is described, in two-axis *d–q* synchronous rotating reference frame, as follows

$$\frac{di_d}{dt} = -\frac{R_s}{L_d}i_d + \omega_e\frac{L_q}{L_d}i_q + \frac{1}{L_d}v_d \tag{1}$$

$$\frac{di_q}{dt} = -\omega_e\frac{L_d}{L_q}i_d - \frac{R_s}{L_q}i_q - \omega_e\frac{\lambda_f}{L_q} + \frac{1}{L_q}v_q \tag{2}$$

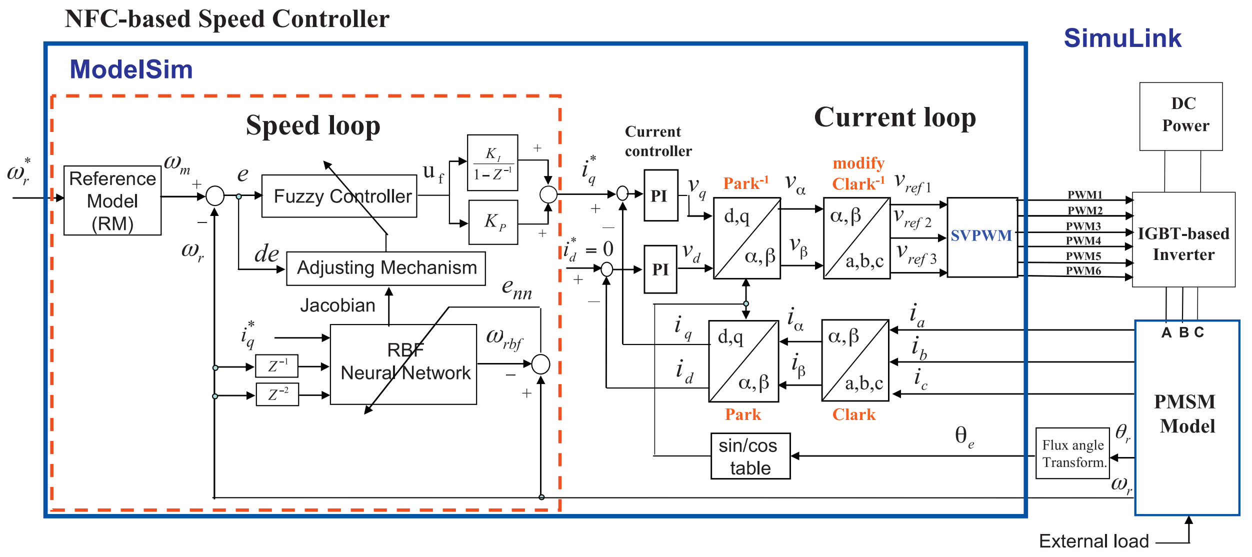


Fig. 1. The simulation architecture of NFC-based speed control for PMSM drive.

where v_d and v_q are the d and q axis voltages; i_d and i_q, are the d and q axis currents; R_s is the phase winding resistance; L_d and L_q are the d and q axis inductance; ω_e is the rotating speed of magnet flux; and λ_f is the permanent magnet flux linkage.

The current loop control of PMSM drive in Fig. 1 is based on a vector control approach. That is, if the i_d is controlled to 0 in Fig. 1, the PMSM will be decoupled and controlling a PMSM like to control a DC motor. Therefore, after decoupling, the torque of PMSM can be written as the following equation,

$$T_e = \frac{3P}{4}\lambda_f i_q \triangleq K_t i_q \tag{3}$$

with

$$K_t = \frac{3P}{4}\lambda_f \tag{4}$$

Considering the mechanical load, the overall dynamic equation of PMSM drive system is obtained by

$$J_m \frac{d}{dt}\omega_r + B_m\omega_r = T_e - T_L \tag{5}$$

where T_e is the motor torque, K_t is torque constant, J_m is the inertial value, B_m is damping ratio, T_L is the external torque, and ω_r is rotor speed.

2.2. Design of neural fuzzy controller (NFC)

The dash rectangular area in Fig. 1 presents the architecture of an NFC for the PMSM drive. It consists of a FC, a reference model and a RBF NN based parameter adjusting mechanism. Detailed description of these is as follows.

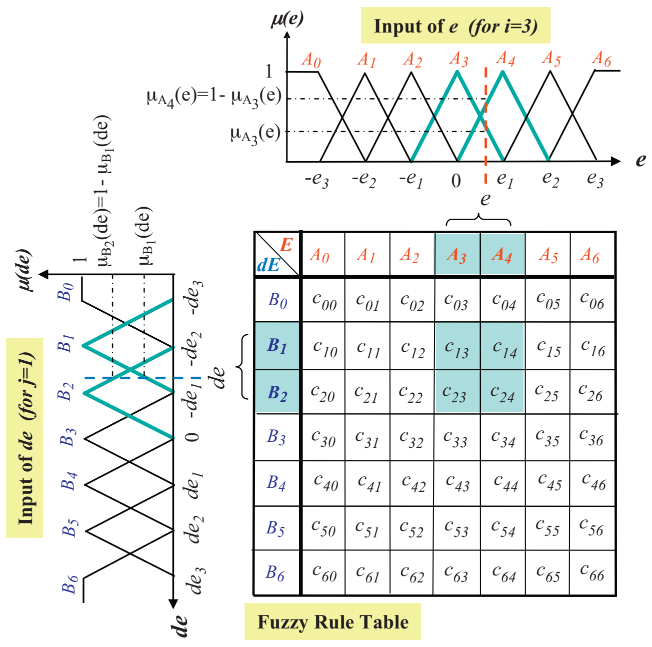
2.2.1. Fuzzy controller (FC)

The FC in this study uses singleton fuzzifier, triangular membership function, product-inference rule and central average defuzzifier method. In Fig. 1, the tracking error e and the error change de are defined by

$$e(k) = \omega_m(k) - \omega_r(k) \tag{6}$$

$$de(k) = e(k) - e(k-1) \tag{7}$$

where u_f represents the output of the FC and ω_m is the output of reference model. The design procedure of FC algorithm is as follows. Firstly, the linguist value of E and dE are $\{A_0, A_1, A_2, A_3, A_4, A_5, A_6\}$ and $\{B_0, B_1, B_2, B_3, B_4, B_5, B_6\}$,


μ(e)
Input of e (for i=3)
1
$\mu_{A_4}(e)=1-\mu_{A_3}(e)$
$\mu_{A_3}(e)$
A_0 A_1 A_2 A_3 A_4 A_5 A_6
$-e_3$ $-e_2$ $-e_1$ 0 e_1 e_2 e_3
e
e
μ(de)
Input of de (for j=1)
1
$\mu_{B_2}(de)=1-\mu_{B_1}(de)$
$\mu_{B_1}(de)$
B_0 B_1 B_2 B_3 B_4 B_5 B_6
$-de_3$ $-de_2$ $-de_1$ 0 de_1 de_2 de_3
de
de
Fuzzy Rule Table

dE \ E	A_0	A_1	A_2	A_3	A_4	A_5	A_6
B_0	c_{00}	c_{01}	c_{02}	c_{03}	c_{04}	c_{05}	c_{06}
B_1	c_{10}	c_{11}	c_{12}	c_{13}	c_{14}	c_{15}	c_{16}
B_2	c_{20}	c_{21}	c_{22}	c_{23}	c_{24}	c_{25}	c_{26}
B_3	c_{30}	c_{31}	c_{32}	c_{33}	c_{34}	c_{35}	c_{36}
B_4	c_{40}	c_{41}	c_{42}	c_{43}	c_{44}	c_{45}	c_{46}
B_5	c_{50}	c_{51}	c_{52}	c_{53}	c_{54}	c_{55}	c_{56}
B_6	c_{60}	c_{61}	c_{62}	c_{63}	c_{64}	c_{65}	c_{66}

Fig. 2. The symmetrical triangular membership function of *e* and *de* and fuzzy rule table.

respectively. Each linguist value of *E* and *dE* is based on the symmetrical triangular membership function, which is shown in Fig. 2. Secondly, the computation of the membership degree for *e* and *de* are done. Fig. 2 shows that the only two linguistic values are excited (resulting in a non-zero membership) in any input value, and the membership degree is obtained by

$$\mu_{A_i}(e)=\frac{e_{i+1}-e}{e_{i+1}-e_i} \quad \text{and} \quad \mu_{A_{i+1}}(e)=1-\mu_{A_i}(e) \tag{8}$$

Similar results can be obtained in computing the membership degree $\mu_{B_j}(de)$. Thirdly, the selection of the initial FC rules refers to the dynamic response characteristics, such as,

$$\text{IF } e \text{ is } A_i \quad \text{and} \quad \Delta e \text{ is } B_j \quad \text{THEN } u_f \quad \text{is} \quad c_{j,i}, \tag{9}$$

where *i* and *j* are from 0 to 6, A_i and B_j are fuzzy numbers, and $c_{j,i}$ is the real number. Finally, to construct the fuzzy system $u_f(e, de)$, the singleton fuzzifier, product-inference rule, and central average defuzzifier method is adopted. Although there are total 49 fuzzy rules in Fig. 2 will be inferred, actually only 4 fuzzy rules can be effectively excited to generate a non-zero output. Therefore, if an error *e* is located between e_i and e_{i+1}, and an error change *de* is located between de_j and de_{j+1}, only four linguistic values A_i, A_{i+1}, B_j, B_{j+1} and corresponding consequent values $c_{j,i}$, $c_{j+1,i}$, $c_{j,i+1}$, $c_{j+1,i+1}$ can be excited, and the output of the fuzzy system can be inferred by the following expression:

$$u_f(e,de)=\frac{\sum_{n=i}^{i+1}\sum_{m=j}^{j+1}c_{m,n}[\mu_{A_n}(e)\times\mu_{B_m}(de)]}{\sum_{n=i}^{i+1}\sum_{m=j}^{j+1}\mu_{A_n}(e)\times\mu_{B_m}(de)} \triangleq \sum_{n=i}^{i+1}\sum_{m=j}^{j+1}c_{m,n}\times d_{n,m} \tag{10}$$

where $d_{n,m}\triangleq\mu_{A_n}(e)\times\mu_{B_m}(de)$. And those $c_{m,n}$ are adjustable parameters. In addition, by using (8), it is straightforward to obtain $\sum_{n=i}^{i+1}\sum_{m=j}^{j+1}d_{n,m}=1$ in (10).

2.2.2. *Radial basis function neural network (RBF NN)*

The RBF NN adopted here is a three-layer architecture which is shown in Fig. 3 and comprised of one input layer, one hidden layer and one output layer.

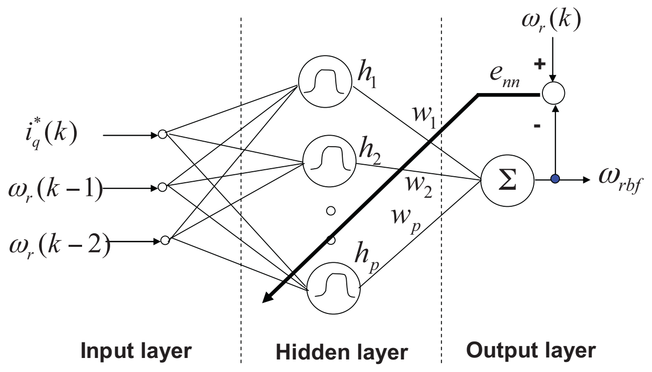


Fig. 3. The architecture of RBF NN.

The RBF NN has three inputs by $i_q^*(k)$, $\omega_r(k-1)$, $\omega_r(k-2)$ and its vector form is represented by

$$X = [i_q^*(k), \omega_r(k-1), \omega_r(k-2)]^T \tag{11}$$

Furthermore, the multivariate Gaussian function is used as the activated function in hidden layer of RBF NN, and its formulation is shown as follows.

$$h_r = \exp\left(-\frac{||X - c_r||^2}{2\sigma_r^2}\right), \quad r = 1, 2, 3, 4, \ldots p \tag{12}$$

where $c_r = [c_{r1}, c_{r2}, c_{r3}]^T$, p is the number of neuron in hidden layer, σ_r denotes the node center and node variance of rth neuron, and $||X - c_r||$ is the norm value which is measured by the inputs and the node center at each neuron. And the network output in Fig. 3 can be written as

$$\omega_{rbf} = \sum_{r=1}^{p} w_r h_r \tag{13}$$

where ω_{rbf} is the output value; w_r and h_r are the weight and output of rth neuron, respectively.

The instantaneous cost function is defined as follows.

$$J = \frac{1}{2}(\omega_{rbf} - \omega_r)^2 \triangleq \frac{1}{2} e_{nn}^2 \tag{14}$$

According to the gradient descent method, the learning algorithm of weights, node center and variance are as follows:

$$w_r(k+1) = w_r(k) + \eta e_{nn}(k) h_r(k) \tag{15}$$

$$c_{rs}(k+1) = c_{rs}(k) + \eta e_{nn}(k) w_r(k) h_r(k) \frac{X_s(k) - c_{rs}(k)}{\sigma_r^2(k)} \tag{16}$$

$$\sigma_r(k+1) = \sigma_r(k) + \eta e_{nn}(k) w_r(k) h_r(k) \frac{||X(k) - c_r(k)||^2}{\sigma_r^3(k)} \tag{17}$$

where $r = 1, 2, \ldots p$, $s = 1, 2, 3$ and η is a learning rate. Further, the $(\partial\omega_r/\partial i_q^*)$ is Jacobian transformation and can be derived from Fig. 3 and (12)

$$\frac{\partial\omega_r}{\partial i_q^*} \approx \frac{\partial\omega_{rbf}}{\partial i_q^*} = \sum_{r=1}^{p} w_r h_r \frac{c_{r1} - i_q^*(k)}{\sigma_r^2} \tag{18}$$

2.2.3. *Reference model (RM)*

Second order system as follows is usually considered as the RM in the adaptive control system

$$\frac{\omega_m(s)}{\omega_r^*(s)} = \frac{\omega_n^2}{s^2 + 2\varsigma\omega_n s + \omega_n^2} \tag{19}$$

where ω_n is natural frequency and ς is damping ratio. Furthermore, applying the bilinear transformation, (19) can be transformed to a discrete model by

$$\frac{\omega_m(z^{-1})}{\omega_r^*(z^{-1})} = \frac{\theta_0 + \theta_1 z^{-1} + \theta_2 z^{-2}}{1 + \phi_1 z^{-1} + \phi_2 z^{-2}} \tag{20}$$

and the difference equation is written as.

$$\omega_m(k) = -\phi_1\omega_m(k-1) - \phi_2\omega_m(k-2) + \theta_0\omega_r^*(k) \ + \theta_1\omega_r^*(k-1) + \theta_2\omega_r^*(k-2) \tag{21}$$

2.2.4. *Adjusting mechanism of FC*

The gradient descent method is used to derive the NFC learning law in Fig. 1. The adjusting mechanism of FC parameters is to minimize the square error between the rotor speed and the output of the reference model. The instantaneous cost function is firstly defined by

$$J_e \triangleq \frac{1}{2}e^2 = \frac{1}{2}(\omega_m - \omega_r)^2 \tag{22}$$

and the parameters of $c_{m,n}$ are adjusted according to

$$\Delta c_{m,n} \propto -\frac{\partial J_e}{\partial c_{m,n}} = -\alpha\frac{\partial J_e}{\partial c_{m,n}} \tag{23}$$

where α represents learning rate. Secondly, the chain rule is used, and the partial differential equation for J_e in (22) can be written as

$$\frac{\partial J_e}{\partial c_{m,n}} = -e\frac{\partial \omega_r}{\partial u_f}\frac{\partial u_f}{\partial c_{m,n}} \tag{24}$$

Further, from (10) and using the Jacobian formulation from (18), we can, respectively, get

$$\frac{\partial u_f(k)}{\partial c_{m,n}(k)} = d_{n,m} \tag{25}$$

and,

$$\frac{\partial \omega_r}{\partial u_f} \approx (K_P + K_i)\frac{\partial \omega_{rbf}}{\partial i_q^*} = (K_P + K_i)\sum_{r=1}^{p} w_r h_r \frac{c_{r1} - i_q^*(k)}{\sigma_r^2} \tag{26}$$

Therefore, substituting (25) and (26) into (24), the parameters $c_{m,n}$ of fuzzy controller described in (10) can be adjusted by the following expression.

$$\Delta c_{m,n}(k) = \alpha e(k)(K_p + K_i) d_{n,m}\sum_{r=1}^{p} w_r h_r \frac{c_{r1} - i_q^*(k)}{\sigma_r^2} \tag{27}$$

with $m = j, j+1$ and $n = i, i+1$.

3. Design of FPGA-based speed controller for PMSM drive

The internal architecture of the proposed FPGA-based speed controller for PMSM drive is shown in Fig. 4. The inputs of this controller are speed command ω_r^*, rotor speed ω_r, flux angle θ_e, measured three-phase currents (i_a, i_b, i_c), and the output is PWM command. The speed controller mainly includes a NFC-based speed controller, a current controller and coordinate transformation (CCCT), a SVPWM generation, frequency divider, etc. The sampling frequency of current and speed control is designed with 16 kHz and 2 kHz, respectively. The input clock is 50 MHz and the frequency divider generates 50 MHz (*Clk*) and 12.5 MHz (*Clk-step*) clock to supply all modules of the FPGA-based speed controller. All modules in Fig. 4 are described by VHDL and simulated in ModelSim. The FPGA resource usages of CCCT, SVPWM and NFC controller in Fig. 4, with the example of Altera – Cyclone EP2C70, need 647 LEs (logic elements) and 196,608 RAM bits, 1200 LEs and 0 RAM bit, 13,806 LEs and 0 RAM bit, respectively. The circuit

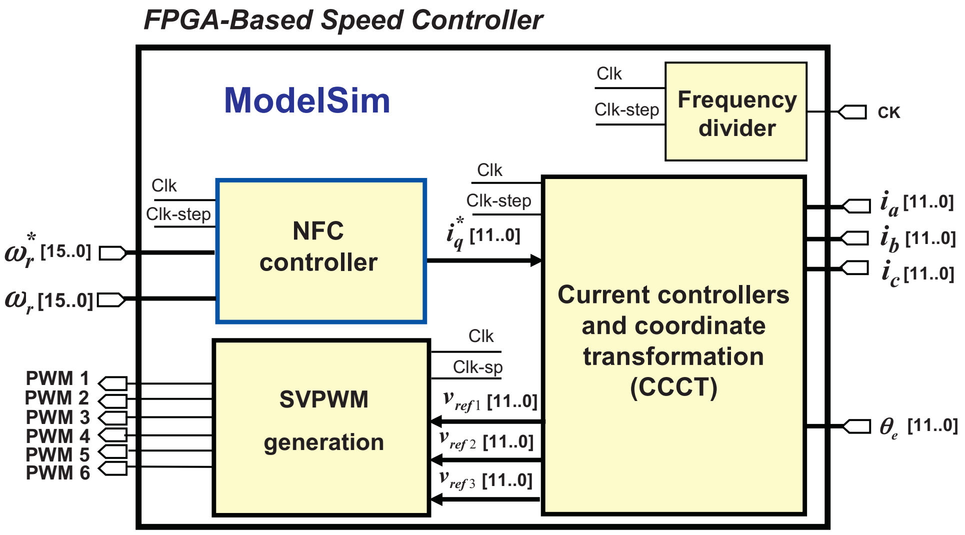


Fig. 4. Internal circuit of the proposed FPGA-based speed controller for PMSM drive.

designs of CCCT and SVPWM refer to [7]. The following paragraphs focus on the description of circuit design in NFC with detail.

3.1. Finite state machine (FSM) method

To reduce the use of the hardware resource, finite state machine (FSM) is adopted to model the computing process of algorithm. Herein, the computation of the sum of product (SOP) shown below is taken as an example to present the advantage of FSM.

$$Y = a_1 \times x_1 + a_2 \times x_2 + a_3 \times x_3 \tag{28}$$

Two kinds of design method are presented to realize the computation of SOP. There are parallel processing method and sequential execution method. Parallel processing with the designed SOP circuit is shown in Fig. 5(a), which will operate continuously and simultaneously. The SOP circuit requires 2 adders, 3 multipliers, and merely near one clock time to complete the overall computation. With the advantage of fast computation ability, the parallel processing method, however, consumes much more FPGA resources. To solve this problem, a sequential execution method using FMS to model SOP circuit is adopted and shown in Fig. 5(b). The FSM method uses one adder, one multiplier and manipulates 5 steps (or 5 clocks time) machine to carry out the overall computation of SOP. Compared to parallel processing method, the FSM method requires more operation time (if one clock time is 80 ns, 5 clocks needs 0.4 μs) in executing SOP circuit; nevertheless, it does not loss any computation power. As a result, the more complicated computation in algorithm, the more FPGA resources will be saved by applying FSM method. Besides, the state diagram in Fig. 5(a) is easy to be described by VHDL.

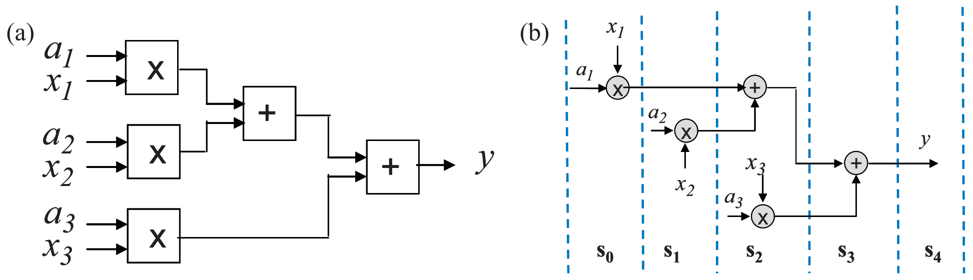


Fig. 5. Computation of SOP by using (a) parallel operation and (b) sequential execution method using FMS.

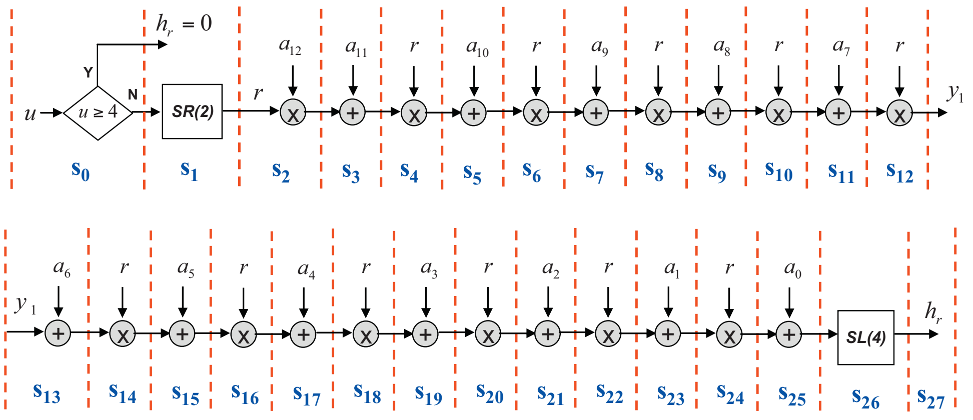

$h_r = 0$
Y
N
u
$u \geq 4$
SR(2)
r
a_{12} a_{11} r a_{10} r a_9 r a_8 r a_7 r
y_1
s_0 s_1 s_2 s_3 s_4 s_5 s_6 s_7 s_8 s_9 s_{10} s_{11} s_{12}
a_6 r a_5 r a_4 r a_3 r a_2 r a_1 r a_0
SL(4)
h_r
s_{13} s_{14} s_{15} s_{16} s_{17} s_{18} s_{19} s_{20} s_{21} s_{22} s_{23} s_{24} s_{25} s_{26} s_{27}

Fig. 6. State diagram of an FSM for describing the exponential function.

3.2. *Behavior description of exponential function*

According to the architecture of hidden layer in RBF NN in (12), it needs to compute the exponential function and is defined as follows

$$h_r = \exp\left(-\frac{||X - c_r||^2}{2\sigma_r^2}\right) \triangleq \exp(-u) \tag{29}$$

To simplify the computation, the input of exponential function is limited within 0–4 because if $u \geq 4$ the output $h_r \leq \exp(-4) = 0.0183$ will approximate to zero, otherwise if $0 \leq u < 4$, (29) can be computed by using Taylor expansion series.

$$h_r = \exp(-u) = \sum_{n=0}^{\propto} (-1)^n \frac{u^n}{n!} \approx \sum_{n=0}^{12} (-1)^n \frac{u^n}{n!} \tag{30}$$

The 12th order is selected in (30). To normalize the input value, we define ($r = u/4$) and to avoid the numerical overflow condition during computation, (30) is divided by 16. Therefore, (30) becomes

$$h_r = 16\frac{\exp(-4r)}{16} \approx 16\sum_{n=0}^{12} (-1)^n \frac{4^{n-2} r^n}{n!} \triangleq 16\sum_{n=0}^{12} a_n r^n \tag{31}$$

where $a_n \triangleq (-1)^n (4^{n-2}/n!)$ by $a_0 = 0.00625$, $a_1 = -0.25$, . . ., $a_{12} = 0.00218909$.

Sequential execution is herein adopted to evaluate the polynomial of degree twelve in (31), and we transfer the form of (31) as follows for easy sequential computation.

$$h_r = 16(((((((((((a_{12}r + a_{11})r + a_{10})r + a_9)r + a_8)r + a_7)r + a_6)r + a_5)r + a_4)r + a_3)r + a_2)r + a_1)r + a_0) \tag{32}$$

FSM is employed to model the polynomial form in (32) and it is shown in Fig. 6, which uses one adder, one multiplier, one comparator and two shifters as well as manipulates 28 steps machine to carry out the overall computation. The *SR*(2) and *SL*(4) in Fig. 6 represent right shift with 2-bit and left shift with 4-bit, respectively. The multiplier and adder apply Altera LPM (library parameterized modules) standard. The FSM can be easily described by VHDL. Moreover, the operation of each step in Fig. 6 can be completed within 80 ns (12.5 MHz clock); therefore total 28 steps only need 2.24 μs operational times.

3.3. *Behavior description of a neuron in RBF NN*

After describing the behavior of exponential function, we further apply it in the behavior description of computing a neuron in RBF NN. In each neuron in Fig. 3, it needs to perform the function of computing the mutivariate Gaussian

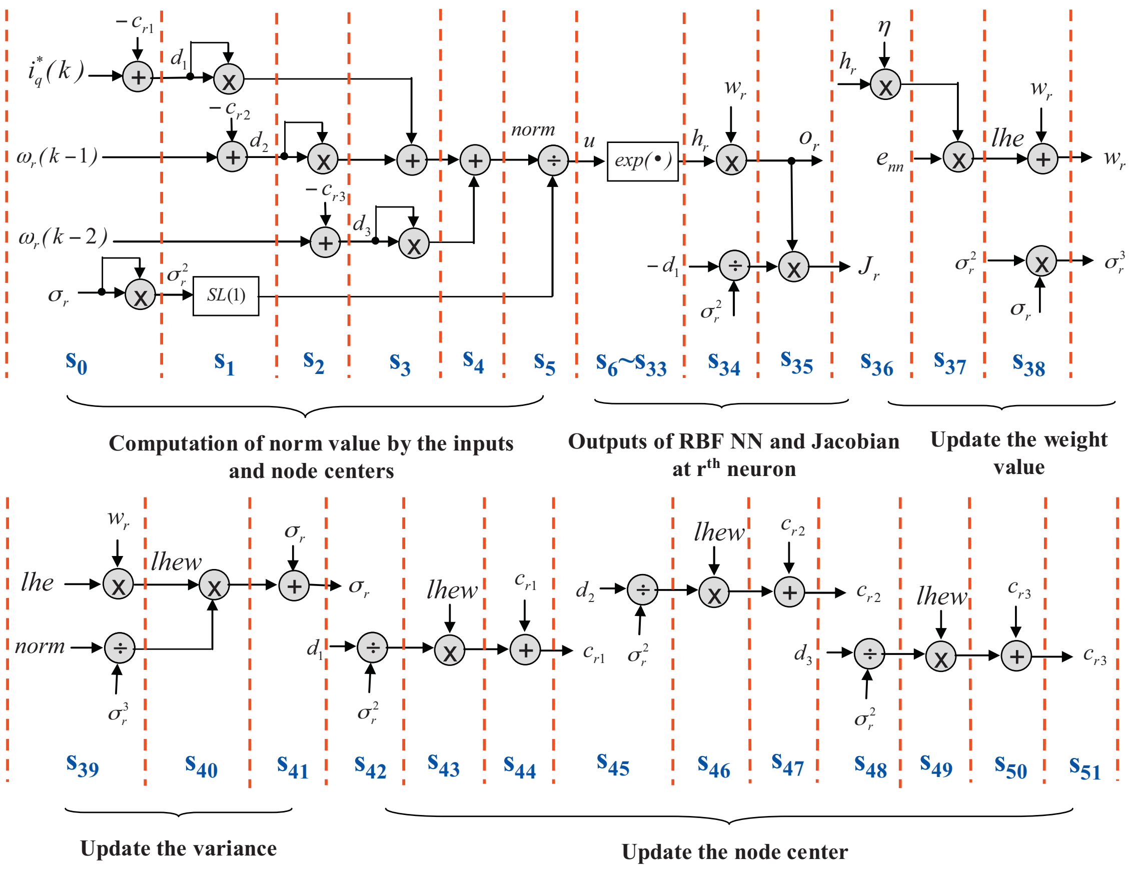


Fig. 7. State diagram of an FSM for describing rth neuron computation in RBF NN.

function in (12), individual network output in (13), individual Jacobian value in (18) and individual parameters learning in (15)–(17). According to this requirements, FSM for describing rth neuron computation in RBF NN is presented in Fig. 7 which the inputs are $i_q^*(k)$, $\omega_r(k-1)$, $\omega_r(k-2)$ and outputs are O_r (individual network output) and J_r (individual Jacobian value). Further, in Fig. 7, steps s_0–s_5 execute the computation of norm value; steps s_6–s_{35} describe the computation of exponential function and the outputs of individual network output and Jacobian value; steps s_{36}–s_{38} are the weight update; steps s_{39}–s_{41} are the variance update and s_{42}–s_{51} are the node center update. The operation of each step in Fig. 7 can be completed within 80 ns (12.5 MHz clock); therefore total 52 steps only need 4.16 μs operational times. In Fig. 7, except exponential function, the divider is also a complicated component in hardware implementation. Herein, we directly adopt Altera LPM standard to realize it. Fig. 8 shows a VHDL example to describe the divider computation of $Y=A/B$. The data format of two inputs A, B and one output Y all belong to the 16 bits, Q15 and signed number. The divider component adopts 32 bits operation with signed representation. The inputs A, B firstly need to sign-extension to 32 bits, then sent to the divider component and obtain a 32 bits output of *sat*. Finally, the divider output of Y is extracted from 16th bit down to 1st of *sat*. The resource usage of a 32 bits divider component in Fig. 8, with the example of Altera – Cyclone EP2C70, needs 980 LEs.

3.4. Behavior description of NFC

After describing the behavior of a neuron in RBF NN, we further apply it in the behavior description of computing a NFC. In the proposed system, the number of neuron in hidden layer is chosen by three. The FSM employed to model the NFC-based speed controller is shown in Fig. 9, which uses one adder, one multiplier, three neuron computational blocks (the detail for each one is shown in Fig. 7), some registers, etc. and manipulates 92 steps machine to carry out the overall computation. The data types are designed with 16-bit length, two complements and Q15 format. The multiplier

```
LIBRARY IEEE;
USE IEEE.std_logic_1164.all;
USE IEEE.std_logic_arith.all;
USE IEEE.std_logic_signed.all;
LIBRARY lpm;
USE lpm.LPM_COMPONENTS.ALL;

ENTITY Devider IS
port  ( clk,clk_D   : IN  STD_LOGIC;
        A,B         : IN  STD_LOGIC_VECTOR(15 downto 0);
        Y           : OUT STD_LOGIC_VECTOR(15 downto 0) );
END Devider;

ARCHITECTURE Devide_arch OF Devider IS
SIGNAL devideA,devideB      :STD_LOGIC_VECTOR(31 downto 0);
SIGNAL sat                  :STD_LOGIC_VECTOR(31 downto 0);
SIGNAL CNT                  :STD_LOGIC_VECTOR(7 downto 0);
BEGIN
m1: lpm_divide
GENERIC
  MAP (LPM_WIDTHN=>32, LPM_WIDTHD=>32, LPM_PIPELINE=>1,
     LPM_NREPRESENTATION=>"SIGNED", LPM_DREPRESENTATION=>"SIGNED")
  PORT MAP (numer=>devideA,denom=>devideB,clock =>clk,quotient=>sat);
```

```
GEN:BLOCK
BEGIN
PROCESS(CLK_D)
BEGIN
  IF clk_D'EVENT and clk_D='1' THEN
     CNT<=CNT+1;
     IF CNT=X"00" THEN
        devideA<=A&X"0000";
        IF B(15)='0' THEN
           devideB<=X"0000"&B;
       ELSE
           devideB<=X"FFFF"&B;
       END IF;
     ELSIF CNT=X"01" THEN
        Y<=sat(16 downto 1);
        CNT <=X"00";
     END IF;
  END IF;
END PROCESS;
END BLOCK GEN;
END Devide_arch;
```

Fig. 8. Example of divider computation using VHDL.

and adder apply Altera LPM standard. Although the algorithm of the NFC is high complexity, the FSM can give a very adequate modeling and easily be described by VHDL. In Fig. 9, steps s_0–s_5 execute the computation of reference model output; steps s_6–s_7 are for the computation of speed error and error change; steps s_8–s_{12} execute the fuzzification and look-up fuzzy table; s_{13}–s_{21} are for the defuzzification; s_{22}–s_{25} are the computation of current command; s_{26}–s_{81} describe the computation of RBF NN and Jacobian value by using three parallel neuron computational block; finally s_{82}–s_{91} execute the tuning of fuzzy rule parameters. The operation of each step in Fig. 9 can be completed within 80 ns (12.5 MHz clock); therefore total 92 steps only need 7.36 μs operational times. It does not loss any control performance for the overall system because the operation time with 7.36 μs is less than the sampling interval, 500 μs (2 kHz), of the speed control loop in Fig. 1. Finally, the execution time of NFC in software by using Nios II processor is evaluated and it is 1190.8 μs. It shows that the computational power in hardware of FPGA is about 160 times faster than in software by using Nios II processor.

4. Simulation results

The NFC-based speed control block diagram for PMSM drive is shown in Fig. 1 and its Simulink/ModelSim co-simulation architecture is presented in Fig. 10. The SimPowerSystem blockset in the Simulink executes the PMSM and the inverter. The EDA simulator link for ModelSim executes the co-simulation using VHDL code running in ModelSim program with two works. The work-1 of ModelSim in Fig. 10 performs the function of speed loop neural fuzzy controller (NFC) and the work-2 executes the function of current controller and coordinate transformation (CCCT) and SVPWM. All works in ModelSim are described by VHDL. The sampling frequency of current and speed control is designed with 16 kHz and 2 kHz, respectively. The clocks of 50 MHz and 12.5 MHz will supply all works of ModelSim. The designed PMSM parameters used in simulation are that pole pairs is 4, stator phase resistance is 1.3 Ω, stator inductance is 6.3 mH, inertia is $J = 0.000108\ \mathrm{kg\,m^2}$ and friction factor is $F = 0.0013\ \mathrm{N\,m\,s}$.

To evaluate the effectiveness of the proposed control algorithm, three tested cases with various PMSM parameters are conducted, in which

Case 1: (normal-load condition)

$$J = 0.000108, \quad F = 0.0013 \tag{33}$$

Case II: (light-load condition)

$$J = 0.000108/3, \quad F = 0.0013/3 \tag{34}$$

Case III: (heavy-load condition)

$$J = 0.000108 \times 3, \quad F = 0.0013 \times 3 \tag{35}$$

The co-simulation is carried out in Fig. 10. The control objective is to control the rotor speed of PMSM to track the output of the reference model. In the case of the FC design, the membership function and the fuzzy rule table are

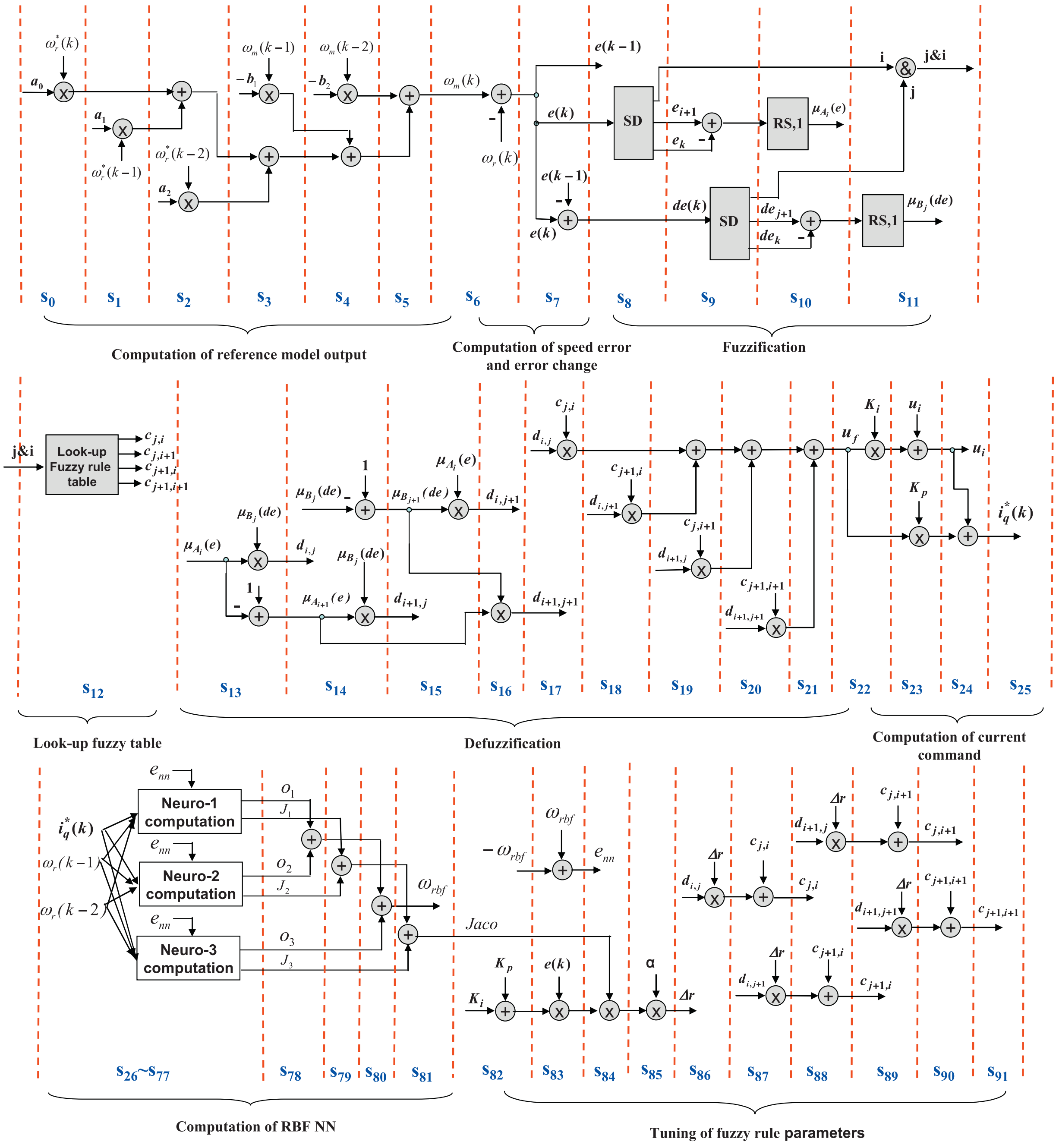


Fig. 9. State diagram of an FSM for describing the NFC in speed loop controller of PMSM drive.

designed in Fig. 11. Besides, the parameters of PI controller in Fig. 1 are selected as $K_p = 1$ and $K_i = 0.025$. Square waves with period of 0.16 s and magnitude variation from 0 to 500 rpm up to 1000 to 1500 rpm is adopted as a tested input command. To evaluate the tracking performance of FC at various system conditions, the system parameters are initially designed at the normal-load condition (Case I), and the simulation result is shown in Fig. 12. It presents a good speed following response in Fig. 12(a) and a complete current decoupled effect in Fig. 12(b). Therefore, a desired rotor speed response with the characteristics of no overshoot, 0.017 s rising time and zero steady-state value, which approximates the speed response curve in Fig. 12(a), is considered as a comparator curve while PMSM runs at different condition. However, when the system parameters change to the light-load (Case II) and heavy-load (Case

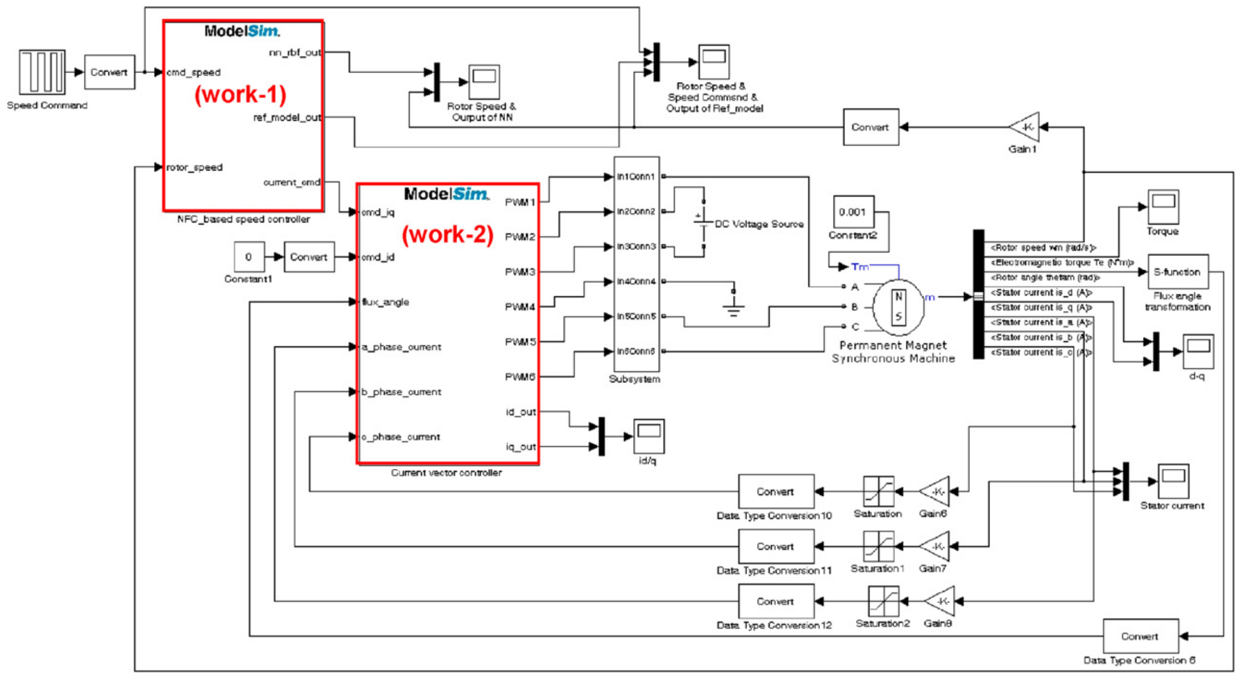
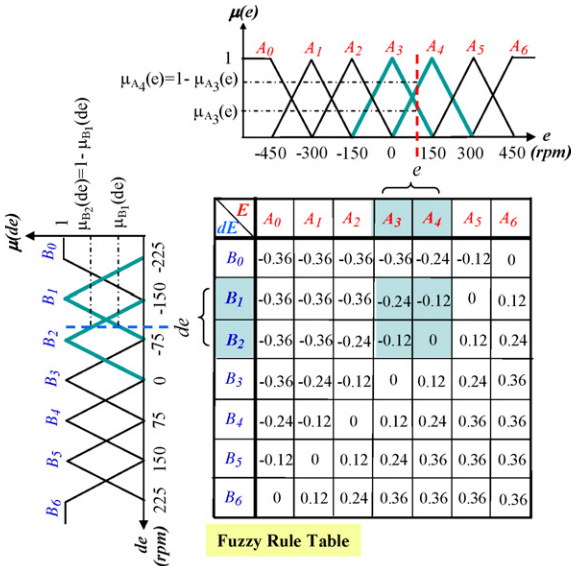


Fig. 10. Simulink and ModelSim co-simulation architecture for NFC-based speed control of PMSM drive.

III) condition, the speed and current responses are shown in Figs. 13 and 14. The rotor speed response in Fig. 13 lags behind the desired rotor speed response with a large overshoot condition and in Fig. 14 is ahead of the desired rotor speed response with a small overshoot condition. It shows that the rotor speed response is greatly affected by system parameters variation if the speed controller uses FC only.

To cope with the system uncertainty problem, a NFC is adopted in Fig. 1. The NFC consists of a FC, a RM and a RBF NN based adjusting mechanism. The RBF NN is applied to real-time identify the plant dynamic for providing an exact plant information to the learning algorithm of FC. The desired rotor speed response with the characteristics of no


μ(e)
1
$\mu_{A_4}(e)=1-\mu_{A_3}(e)$
$\mu_{A_3}(e)$
A_0 A_1 A_2 A_3 A_4 A_5 A_6
-450 -300 -150 0 150 300 450 (rpm)
e
μ(de)
1
$\mu_{B_2}(de)=1-\mu_{B_1}(de)$
$\mu_{B_1}(de)$
B_0 B_1 B_2 B_3 B_4 B_5 B_6
-225 -150 -75 0 75 150 225
de (rpm)
de
e

dE \ E	A_0	A_1	A_2	A_3	A_4	A_5	A_6
B_0	-0.36	-0.36	-0.36	-0.36	-0.24	-0.12	0
B_1	-0.36	-0.36	-0.36	-0.24	-0.12	0	0.12
B_2	-0.36	-0.36	-0.24	-0.12	0	0.12	0.24
B_3	-0.36	-0.24	-0.12	0	0.12	0.24	0.36
B_4	-0.24	-0.12	0	0.12	0.24	0.36	0.36
B_5	-0.12	0	0.12	0.24	0.36	0.36	0.36
B_6	0	0.12	0.24	0.36	0.36	0.36	0.36

Fuzzy Rule Table

Fig. 11. Fuzzy membership function and the initial fuzzy-rule table.

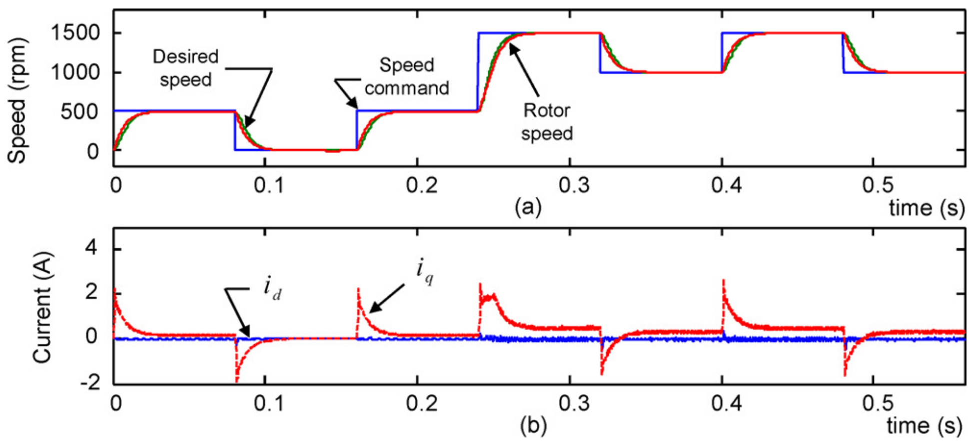


Fig. 12. Simulation results when FC is used and PMSM is operated at normal-load condition.

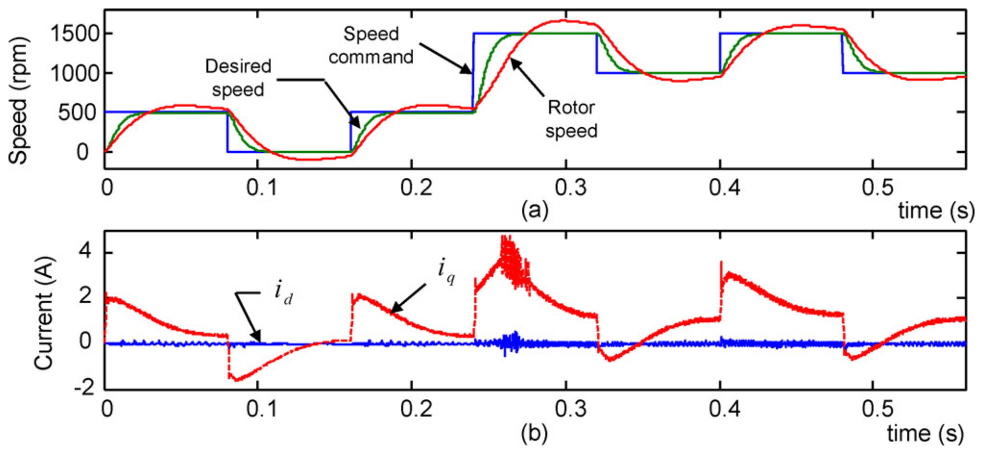


Fig. 13. Simulation results when FC is used and PMSM is operated at heavy-load condition.

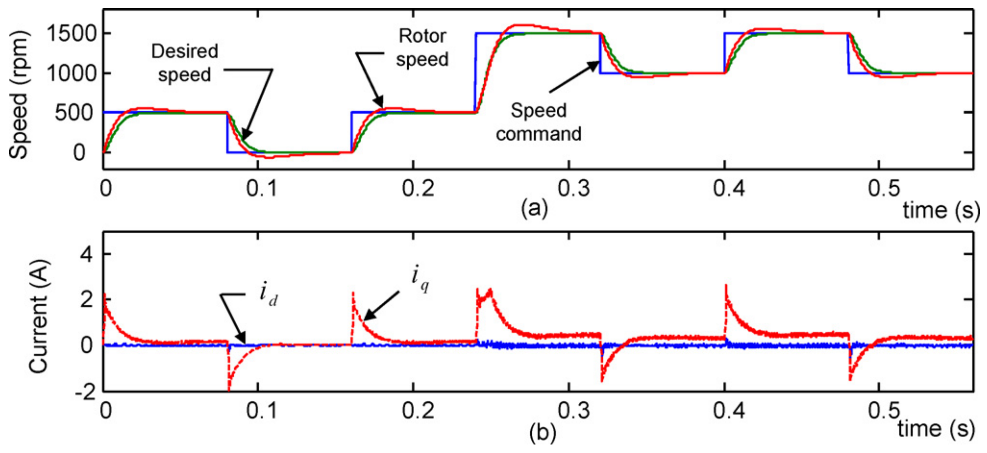


Fig. 14. Simulation results when FC is used and PMSM is operated at light-load condition.

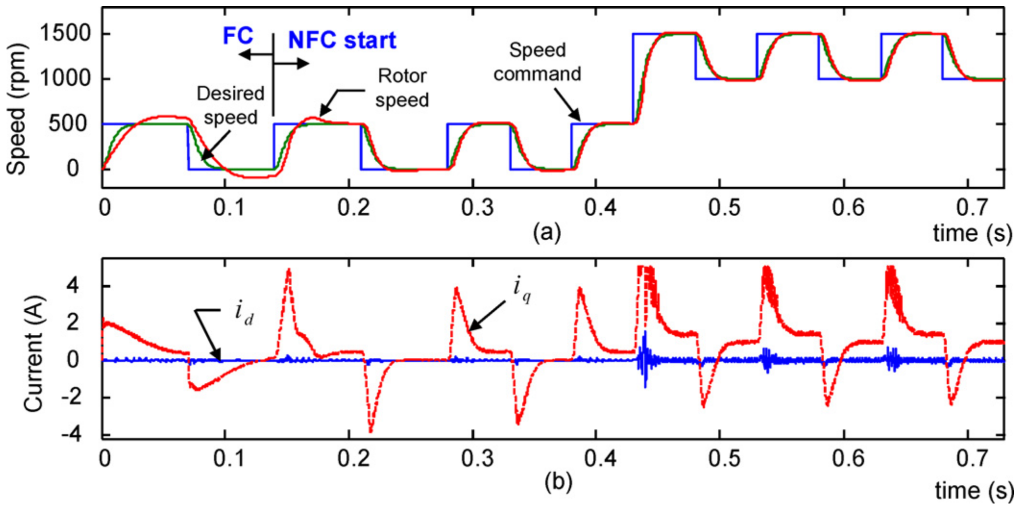


Fig. 15. Simulation results when NFC is used and PMSM is operated at heavy-load condition.

overshoot, 0.017 s rising time and zero steady-state value in Fig. 12 is considered to design the transfer function of the RM. According to the required specifications, a second order system with the natural frequency of 230 rad/s and the damping ratio of 1 is chosen. Then, after applying the bilinear transformation with sampling frequency of 2 kHz, the parameters of the difference equation in (21) are obtained by $\theta_0 = 0.00295$, $\theta_1 = 0.0059$, $\theta_2 = 0.00295$, $\phi_1 = -1.7825$, and $\phi_2 = 0.7943$. In NFC design, the initial fuzzy parameters in Fig. 11 is the same as the FC, but the fuzzy parameters of the $c_{m,n}$ can be tuned using (27) if the output of rotor speed cannot follow the output of RM. The learning rate α is set as 0.3. The initial parameters in RBF NN are chosen by $w_r = 10$, $\sigma_r = 250$, $c_{r1} = c_{r2} = c_{r3} = 250$, where $r = 1, 2, 3$. The learning rate η in RBF NN is set as 0.15. In simulation, square waves with magnitude variation from 0 to 500 rpm up to 1000 to 1500 rpm is adopted as a tested input command and its simulation results under heavy-load and light-load condition are presented in Figs. 15 and 16, respectively. In Fig. 15, in the beginning time, the FC is applied to the speed loop of PMSM drive system and the rotor speed shows a lag and an overshoot response. After 0.14 s, the NFC is adopted. Meanwhile, the $c_{m,n}$ parameters are tuned to an adequate value for reducing the error between the rotor speed and the output of RM. Finally, the rotor speed can accurately track well after one cycle learning. In Fig. 15(b), it exhibits that the current i_q need to generate a larger current value to force the motor running speed to fast track the output of RM. Similar results appear in the light-load condition in Fig. 16. Additionally, two another transition conditions, which

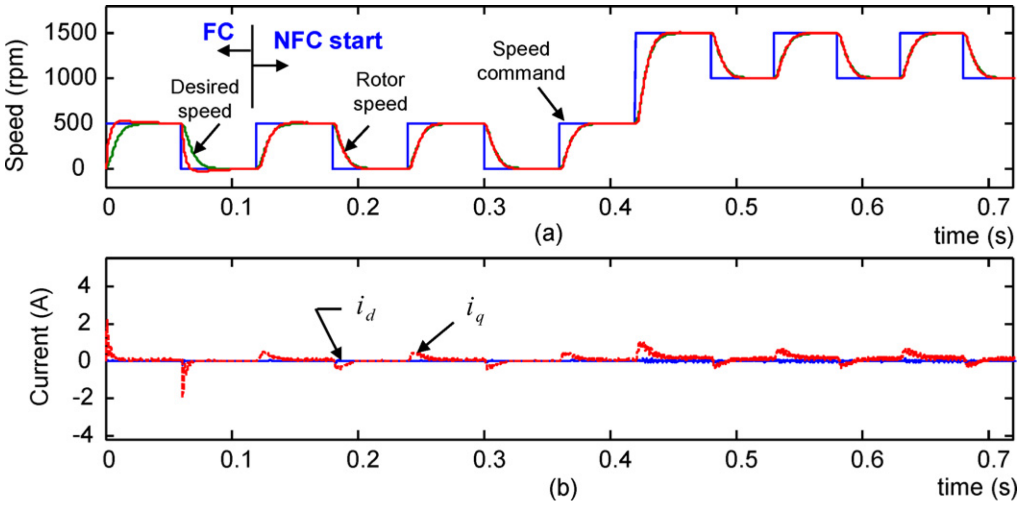


Fig. 16. Simulation results when NFC is used and PMSM is operated at light-load condition.

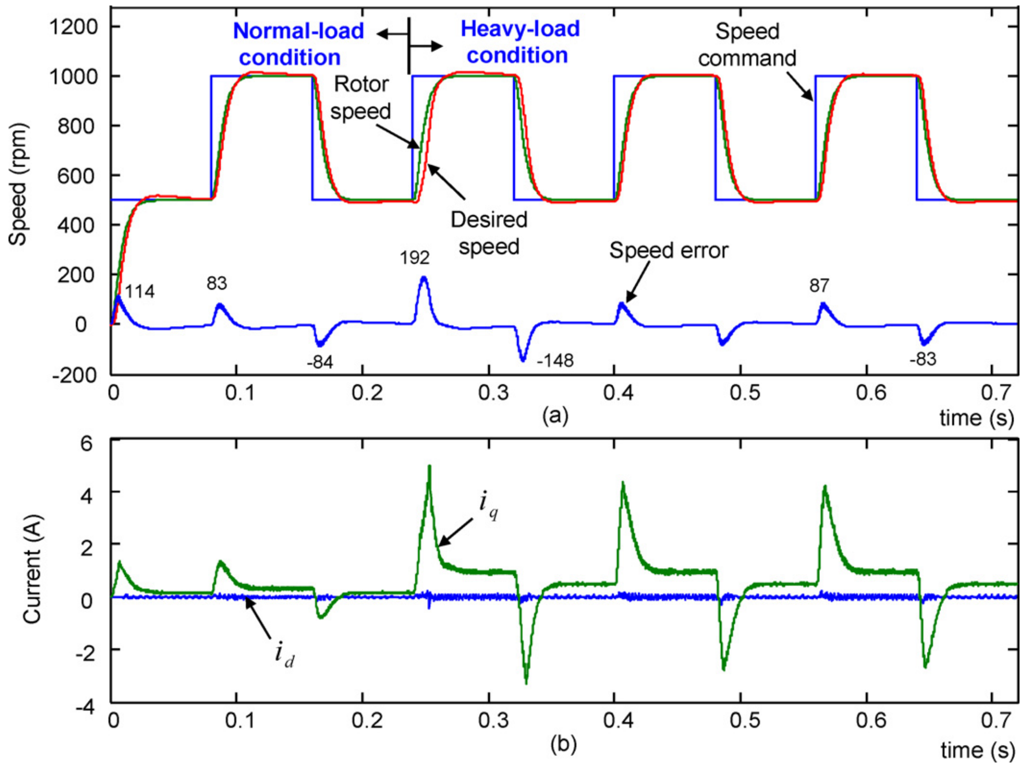


Fig. 17. Simulation results when NFC is used and PMSM is operated varying from normal-load condition to heavy-load condition.

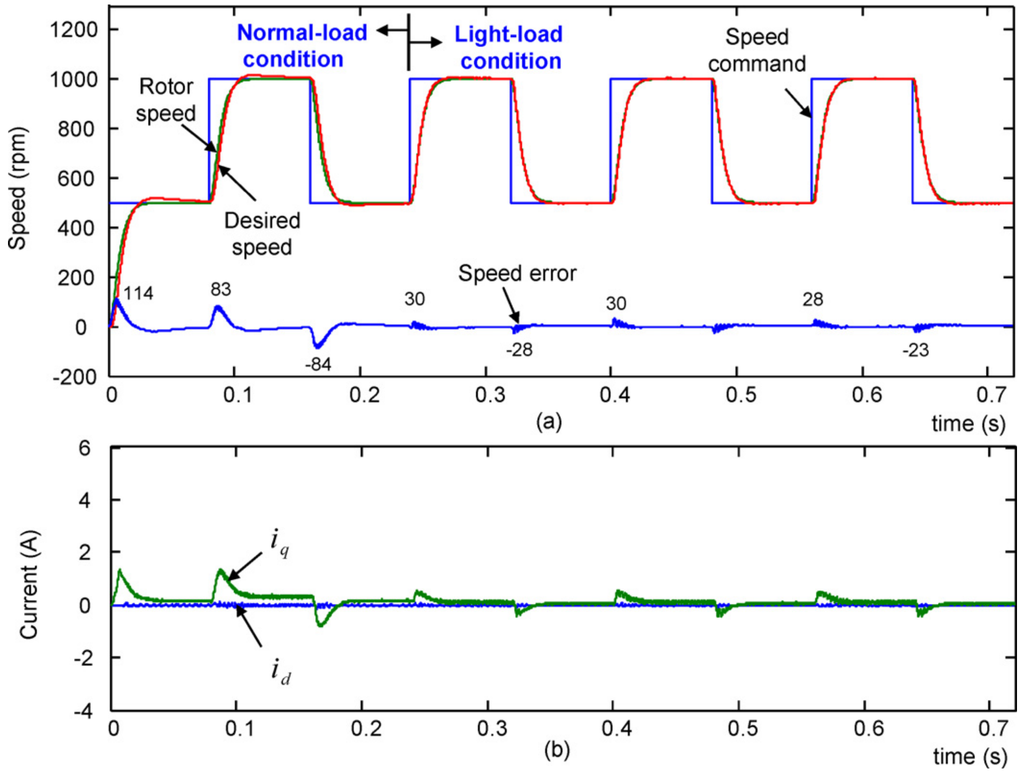


Fig. 18. Simulation results when NFC is used and PMSM is operated varying from normal-load condition to light-load condition.

extern load is changed from normal-load to heavy-load and from normal-load to light-load condition, are considered and evaluated, and the simulated results are shown in Figs. 17 and 18. In the former case, the control current i_q in transition conditions of Fig. 17(b) is apparently increased to speed up the motor running, but in the latter case, the control current i_q in transition conditions of 18(b) is decreased to slow down the motor running. However, it shows that due to the tuning of control current i_q by NFC, the output of rotor speed in Figs. 17(a) and 18(a) can track the desired speed well. Therefore, the simulation results in Figs. 12–18 demonstrate that the proposed NFC-based speed controller for PMSM drive is effective and robust.

5. Conclusions

This study has presented a FPGA-based NFC controller for PMSM drives and successfully demonstrated its performance through co-simulation by using Simulink and ModelSim. In control algorithm, to cope with the system uncertainty, a NFC is proposed and a RBF NN is used to identify the plant dynamic and provided more accuracy plant information for parameters tuning of FC. In realization, a sequential execution using FSM is applied to model the computing process of NFC for reducing the FPGA resource usage. Under the proposed design method, the execution time and FPGA resource usage for computing a NFC spend only 7.36 μs and 13,806 LEs, respectively. It not only does not loss any control performance for the overall system, but also can greatly save the FPGA resource usage. At last, some simulation results demonstrate that in step response, the speed of PMSM can fast track the prescribed dynamic response accurately after the proposed controller has been conducted. However, after confirming the effective of VHDL code of NFC-based speed controller in co-simulation by using Simulink and ModelSim, the VHDL code except A/D and QEP interface circuit, can be directly used in the experimental FPGA-based PMSM drive system for further verifying its function in the future work.

Acknowledgment

This work was supported by National Science Council of the R.O.C. under grant no. NSC 100-2221-E-218-001.

References

[1] B.K. Bose, Expert system, fuzzy logic, and neural network applications in power electronics and motion control, Proceedings of the IEEE 82 (8) (1994) 1303–1323.
[2] S.T. Brassai, L. Bako, G. Pana, S. Dan, Neural control based on RBF network implemented on FPGA, in: Proceedings of the 11th International Conference on Opimization of Electrical and Electronic Equipment, 2008, pp. 41–46.
[3] M.F. Castoldi, M.L. Aguiar, Simulation of DTC strategy in VHDL code for induction motor control, in: Proceedings of the IEEE International Symposium on Industrial Electronics (ISIE), 2006, pp. 2248–2253.
[4] M.F. Castoldi, G.R.C. Dias, M.L. Aguiar, V.O. Roda, Chopper-controlled PMDC motor drive using VHDL code, in: Proceedings of the 5th Southern Conference on Programmable Logic, 2009, pp. 209–212.
[5] J.W. Jung, Y.S. Choi, V.Q. Leu, H.H. Choi, Fuzzy PI-type current controllers for permanent magnet synchronous motors, IET Electric Power Applications 5 (1) (2011) 143–152.
[6] J.S. Kim, S. Jung, Implementation of the RBF neural chip with the on-line learning back-propagation algorithm, in: Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN 2008), 2008, pp. 337–383.
[7] Y.S. Kung, M.H. Tsai, FPGA-based speed control IC for PMSM drive with adaptive fuzzy control, IEEE Transactions on Power Electronics 22 (6) (2007) 2476–2486.
[8] Y.S. Kung, N. Vu Quynh, C.C. Huang, L.C. Huang, Simulink/ModelSim co-simulation of sensorless PMSM speed controller, in: Proceedings of the 2011 IEEE Symposium on Industrial Electronics and Applications (ISIEA 2011), 2011, pp. 24–29.
[9] J. Lázaro, A. Astarloa, J. Arias, U. Bidarte, A. Zuloaga, Simulink/Modelsim simulable VHDL PID core for industrial SoPC multiaxis controllers, in: Proceedings of the IEEE Industrial Electronics 32nd Annual Conference (IECON), 2006, pp. 3007–3011.
[10] Y. Li, J. Huo, X. Li, J. Wen, Y. Wang, B. Shan, An open-loop Sin microstepping driver based on FPGA and the co-simulation of Modelsim and Simulink, in: Proceedings of the International Conference on Computer, Mechatronics, Control and Electronic Engineering (CMCE), 2010, pp. 223–227.
[11] L. Lin, X. Peng, A PID neural network control for permanent magnet synchronous motor servo system, in: Proceedings of the 5th International Conference on Computer Science and Education, 2010, pp. 1174–1178.
[12] Mathworks, Matlab/Simulink Users Guide: Application Program Interface Guide, 2004.
[13] Modeltech, ModelSim Reference Manual, 2004.
[14] E. Monmasson, L. Idkhajine, M.W. Naouar, FPGA-based controllers, IEEE Industrial Electronics Magazine 5 (1) (2011) 14–26.

[15] E. Monmasson, L. Idkhajine, M.N. Cirstea, I. Bahri, A. Tisan, M.W. Naouar, FPGAs in industrial control applications, IEEE Transactions on Industrial Informatics 7 (2) (2011) 224–243.
[16] S. Sanchez-Solano, A.J. Cabrera, I. Baturone, F.J. Moreno-Velo, M. Brox, FPGA Implementation of embedded fuzzy controllers for robotic applications, IEEE Transactions on Industrial Electronics 5 (4) (2007) 1937–1945.
[17] M.G. Zhang, X.G. Wang, M.Q. Liu, Adaptive PID control based on RBF neural network identification, in: Proceedings of the 17th IEEE International Conference on Tools with Artificial Intelligence, 2005, pp. 1854–1857.
[18] Z. Zhou, T. Li, T. Takahahi, E. Ho, FPGA realization of a high-performance servo controller for PMSM, in: Proceedings of the 9th IEEE Application Power Electronics Conference and Exposition, vol. 3, 2004, pp. 1604–1609.