Third Prize

FPGA-Based Smart Induction Motor Controller Design

Institution: Electrical Engineering Department, Yuan Ze University

Participants: Zhong Zhaoming, Lin Minghong, Chen Yilong

Instructor: Lin Zhimin

Design Introduction

From a control viewpoint, DC motors must be maintained frequently due to their brushes and rectifiers. Sometimes the motor is installed in a location that makes it difficult or impossible to maintain or repair the motor. Compared to the disadvantages of DC motors, AC motors have a variety of advantages. for example, AC motors feature small size, light weight, low rotary inertia, and low price. Generally speaking, induction motors are nonlinear and time-varying with a dynamic coupling system, so the controller design is complicated. When considering control problems, various control theories are often proposed, e.g., proportion-integral-derivative control, sliding mode control, adaptive control, etc. These methods aim to make the system's behavior comply with the design requirements for all system parameter variations and external interferences.

Most of these methods are based on knowledge of status equations for fully or partially controlled systems. However, in practice, the status equation is not easily obtained. Therefore, research for a smart control method with a self-learning capability for better control performance becomes an important subject. Our design uses a neural network (NN) for its amazing effect, which traditional controllers cannot achieve, when the system is involved in an uncertain, time-varying, or nonlinear status, etc.

The key for success using NN is the approximation characteristics. Currently, the methods commonly proposed are the back propagation algorithm, Lyapunov stability method, genetic algorithm (GA), etc. Although the back propagation algorithm is direct and straightforward to use, it is hard to ensure stability and robustness in a closed-loop system. A Lyapunov stability theory-based control structure can ensure system stability, but its computing process is complicated. GA can acquire the global optimization result, but its calculation is large and unsuitable for real-time control. Therefore, our team proposed using an adaptive, fuzzy neural network controller algorithm to control the induction motor.

The whole system can self-regulate parameters in real time based on the learning method deduced from Lyapunov stability theory and back propagation algorithm. The developed algorithm obtains the fastest parameter convergence rate, and is easy to perform and implement.

In actual use, digital controllers demonstrate higher stability, expectable output, and stronger anti-noise capability compared to analog controllers. In particular, the rapid growth of semiconductor technology in recent years makes single-component logic circuits the design trend. If an integrated circuit (IC) can be implemented based on digital circuit integration and control rules, the control system will certainly be less complicated and more reliable, providing smaller hardware, lower design cost, fast execution, and high flexibility. FPGAs are suitable for economic returns and research schedules. Based on these theories, we used Altera's Nios[®] II processor to implement the design control rules. When writing the program, we chose the Verilog HDL language for hardware and the Nios II processor for the control rule software. Meanwhile, we combined some hardware peripheral circuits to finish the design and construction of the entire experiment environment. Figure 1 describes the design of the proposed FPGA-based smart induction motor.





The most important and difficult part of this design is implementing the smart control algorithm because the proposed control rules involve many calculations and complicated operations (such as positive/negative numbers and floating-point arithmetic) and Verilog HDL uses binary concepts. Although the complementary code and fixed-point methods are available, users unfamiliar with Verilog HDL grammar can spend a lot of time writing code and it is difficult to perform program maintenance. Instead, using the Nios II embedded processor for the design we wrote our code with the familiar C language without considering positive/negative numbers and floating-point arithmetic, and wrote code directly using the decimal system. Compiling the control rules with the Nios II processor saved a lot of time in parameter adjustment because the Nios II processor is faster to compile than hardware. Additionally, the Nios II processor offers a floating-point custom instruction; adding this instruction greatly shortens the time required for the hardware to process floating-point operations, enhancing system efficiency.

Function Description

Generally, the induction servomotor drive system can be simplified as

$$J\ddot{\theta} + B\dot{\theta} + T_i = T_e \tag{1}$$

Here, J is the rotary inertia, B is a damping coefficient, θ is the motor's rotation, and T_l is additional load interference. T_e is the electromagnetic torque and can be defined as below:

$$T_e = K_i \tilde{t}_{qs} \tag{2}$$

$$K_{t} = (3n_{p}/2)(L_{m}^{2}/L_{r})_{ds}^{*}$$
(3)

Where K_t is the torque constant, i_{qs}^* is the torque current command, i_{ds}^* is the outflow current command, n_p is the pole pair, L_m is the air gap magnetic flux, and L_r is the rotor inductance. In sum, the dynamic equation of induction motor can be rewritten as:-______

$$\ddot{\theta} = -\frac{B}{J}\dot{\theta} + \frac{K_i}{J}\dot{i}_{qs}^* - \frac{1}{J}T_i \stackrel{\wedge}{=} A_p\dot{\theta} + B_p u + D_pT_i$$
⁽⁴⁾

Here, $A_p = -B/J$, $B_p = K_t/J > 0$, $D_p = -1/J$, and $u(t) = i_{qs}^*(t)$ is the control command. The purpose of the entire induction motor control system is to design a control rule, allowing the motor angle to track the control command exactly. Herein, the tracking error is defined as:

$$e = \theta_c - \theta$$

First, provided that both system dynamic functions A_p and B_p of the induction motor and external interference T_l can be acquired, an ideal control rule can be obtained using the feedback control theory.

$$u^{*} = B_{p}^{-1} \left[-A_{p} \dot{\theta} - D_{p} T_{l} + \ddot{\theta}_{c} + k_{1} \dot{e} + k_{2} e \right]$$
(6)

Insert equation (6) of the ideal control rule into system dynamic equation (4) and obtain:

$$\ddot{e} + k_1 \dot{e} + k_2 e = 0 \tag{7}$$

If the appropriate selection of k_1 and k_2 allows equation (7) to be a Hurwitz multinomial (i.e., its roots are all on the left half plane), control objective $\lim_{t \to \infty} e = 0$ will be achieved. In actual use, however, the system dynamic function and external interference often cannot be acquired, i.e., ideal control rule (6) cannot come true.

To address the problem that the ideal controller cannot be realized owing to failed acquisition of the system dynamic function and external interference, we proposed an adaptive fuzzy neural network controller. Figure 2 shows the block diagram, including an NN controller and a compensation controller, and its arithmetic formula is:

$$u = u_{nn} + u_{cp}$$

(5)

The NN controller u_{nn} uses a fuzzy NN to learn ideal controller u^* and uses a compensation controller to overcome the learning error due to the neural controller. First, we define a sliding surface in packet identifier (PID) form as shown below:

$$s = \dot{e} + k_1 e + k_2 \int_0^t e(\tau) d\tau$$
(9)

Insert equation (8) into (4) and we get:

...

$$\theta = A_p \theta + B_p \left(u_{nn} + u_{cp} \right) + D_p T_l _$$
⁽¹⁰⁾

By subtracting equations (6) and (10), we use equation (9) and get the dynamic equation:

$$\dot{s} = B_{p} \left(u^{*} - u_{nn} - u_{cp} \right) \tag{11}$$

Figure 2 shows the adaptive fuzzy neural network control system block diagram.

Figure 2. Adaptive Fuzzy Neural Network Control System Block Diagram



According to the approximation theorem, we know that an optimal neural network controller is near to the ideal control (6), i.e.,

(12)

$$u^* = \mathbf{w}^{*T} \mathbf{\Theta} + \mathcal{E}$$

Where \mathbf{w}^* is the parameter vector of fuzzy rule of optimal value, $\boldsymbol{\Theta}$ is the activation parameter vector of the fuzzy rules, $\boldsymbol{\mathcal{E}}$ is the approximate error for network learning, and we assume $|\boldsymbol{\varepsilon}| \leq E$. In actual use, the optimal network parameter \mathbf{w}^* often cannot be acquired directly or it has multiple solutions, so the estimation machine estimates the entire optimal network parameter, i.e.,

$$u_{nn} = \hat{\mathbf{w}}^{T} \boldsymbol{\Theta} \tag{13}$$

Here, $\hat{\mathbf{w}}$ is estimation parameter vector of \mathbf{w}^* . Thus, the output learning error of network can be defined as:

$$\widetilde{\boldsymbol{u}} = \boldsymbol{u}^* - \boldsymbol{u}_{nn} = \widetilde{\mathbf{w}}^T \boldsymbol{\Theta} + \boldsymbol{\varepsilon}$$
(14)

Where $\tilde{\mathbf{w}} = \mathbf{w}^* - \hat{\mathbf{w}}$. Insert equation (14) into (11) and simplify (11) as:

$$\dot{s} = B_{p} \left(\widetilde{\mathbf{w}}^{T} \mathbf{\Theta} + \varepsilon - u_{cp} \right)$$
(15)

To learn the needed controller parameter online and ensure the stability of the entire closed-loop system, this design deduces the required learning rules based on the Lyapunov stability theorem. Herein, Lyapunov functions are selected as shown below:

$$V = \frac{1}{2}s^2 + B_p \left(\frac{\tilde{\mathbf{w}}^T \tilde{\mathbf{w}}}{2\eta_1} + \frac{\tilde{E}^2}{2\eta_2}\right)$$
(16)

Where η_1 and η_2 are learning speeds and $\tilde{E} = E - \hat{E}$. Adjust equation (16) for time differential and insert equation (15) into it to obtain:

$$\dot{V} = s\dot{s} + B_{p}\left(\frac{\tilde{\mathbf{w}}^{T}\dot{\tilde{\mathbf{w}}}}{\eta_{1}} + \frac{\tilde{E}\dot{\tilde{E}}}{\eta_{2}}\right)$$

$$= s[B_{p}\left(\tilde{\mathbf{w}}^{T}\Theta + \varepsilon - u_{cp}\right)] + B_{p}\left(\frac{\tilde{\mathbf{w}}^{T}\dot{\tilde{\mathbf{w}}}}{\eta_{1}} + \frac{\tilde{E}\dot{\tilde{E}}}{\eta_{2}}\right)$$

$$= B_{p}\tilde{\mathbf{w}}^{T}(s\Theta + \frac{\dot{\tilde{\mathbf{w}}}}{\eta_{1}}) + sB_{p}(\varepsilon - u_{cp}) + B_{p}\left(\frac{\tilde{E}\dot{\tilde{E}}}{\eta_{2}}\right) - \frac{1}{2}$$
(17)

The learning rule is chosen as shown below:

$$\dot{\hat{\mathbf{w}}} = -\dot{\widetilde{\mathbf{w}}} = \eta_1 s \boldsymbol{\Theta}$$
(18)

The compensation controller is chosen as shown below:

$$u_{cp} = \hat{E} \operatorname{sgn}(s) \tag{19}$$

And

$$\dot{\hat{E}} = -\dot{\tilde{E}} = \eta_2 |s| \tag{20}$$

(17) can be simplified as:

$$\dot{V} = \varepsilon \, sB_{\rho} - E |s|B_{\rho} \le -(E - |\varepsilon|) |s|B_{\rho} \le 0$$
(21)

Thus, the designed control system can ensure the system stability based on the Lyapunov stability theorem. To increase the network learning performance, our research introduces a steepest descent algorithm to adjust more parameters. First, an energy function is defined as:

$$E = \frac{1}{2}e^2 \tag{22}$$

Based on the steepest descent algorithm, the adjustment of fuzzy rule is:

$$\Delta w_{k} = -\eta_{w} \frac{\partial E}{\partial w_{k}} = \left[-\eta_{w} \frac{\partial E}{\partial y_{o}^{4}} \frac{\partial y_{o}^{4}}{\partial n e t_{o}^{4}} \right] \left[\frac{\partial n e t_{o}^{4}}{\partial w_{k}^{4}} \right] = -\eta_{w} \delta_{o}^{4} x_{k}$$
(23)

Here $\delta_o^4 = \frac{\partial E}{\partial net_o^4}$. Comparing the coefficient of equation (18) with that of (23) and obtain:

$$\delta_o^4 = s \tag{24}$$

This is a Jacobian item of the entire system and the membership function parameter can be adjusted as:

$$\Delta m_{ij}^{2} = -\eta_{m} \frac{\partial E}{\partial m_{ij}^{2}} = \left[-\eta_{m} \frac{\partial E}{\partial y_{o}^{4}} \frac{\partial y_{o}^{4}}{\partial net_{o}^{4}} \frac{\partial net_{o}^{4}}{\partial y_{k}^{3}} \frac{\partial y_{k}^{3}}{\partial net_{k}^{3}} \frac{\partial net_{k}^{3}}{\partial y_{j}^{2}} \frac{\partial y_{j}^{2}}{\partial net_{j}^{2}} \frac{\partial net_{j}^{2}}{\partial m_{ij}^{2}} \right]$$

$$\Delta \sigma_{ij}^{2} = -\eta_{\sigma} \frac{\partial E}{\partial \sigma_{ij}^{2}} = \left[-\eta_{\sigma} \frac{\partial E}{\partial y_{o}^{4}} \frac{\partial y_{o}^{4}}{\partial net_{o}^{4}} \frac{\partial net_{o}^{4}}{\partial y_{k}^{3}} \frac{\partial y_{k}^{3}}{\partial net_{k}^{3}} \frac{\partial net_{k}^{3}}{\partial y_{j}^{2}} \frac{\partial y_{j}^{2}}{\partial net_{j}^{2}} \frac{\partial net_{j}^{2}}{\partial \sigma_{ij}^{2}} \right]$$

$$(25)$$

Figure 3 shows a block diagram of the FPGA-based induction motor smart control system. The hardware circuit includes a frequency divider (Divider), induction motor angle counting module (Theta-Acc), and two rows of digital-to-analog converter (DAC) modules (DAC_1, DAC_2). The software is the Nios II embedded processor (Nios II CPU). The following sections provide a detailed description of each module.

Figure 3. Hardware of FPGA Induction Motor Control System Block Diagram



Divider

Because the FPGA's input frequency is 50 MHz, it is divided into the required frequency. First, we designed two frequencies, clk and clk1. One frequency controls the DAC chip select (CS) signal, which updates the DAC data signal (LDAC) and controls the system to input/output data once for every 1

millisecond (ms). The other frequency controls the DAC data selection signal (A0, A1). Control using clk1 is slower than clk, ensuring that the data selection signal is not replaced until the output data arrives at the input latch and avoiding incorrect data output.

Theta_Acc

This module increases the motor angle (en) calculated by the optical encoder's 12-bit count circuit to 15 bits using an accumulator. The optical encoder's count circuit output is only 12 bits (0 to 4,095), so a 15-bit register (*Theta*) accumulates the optical encoder's output to make the motor's rotation angle larger than 4,095 for forward or backward rotation.

We designed a judgment condition to determine whether the current angle and the subsequent angle, which are 12 bits beyond the optical encoder (4,095) and smaller than 0, are forward or reverse rotation to calculate the correct rotation angle. A prerequisite is that the current and subsequent rotation angle shall not be larger than 2,047°. Then, the design accumulates the result with an accumulator and outputs it.

Function Generator Module

This module stores the sinusoidal function value in memory. It reads the memory content value using a look-up table to generate the sinusoidal function (Theta_ref) of the motor tracking command. The content value of the sinusoidal function memory has output-enabled amplitude of 2 V and a frequency of $1/2\pi$. It increases as the sinusoidal function of $5/3\pi$ after 5.5 seconds.

Nios II CPU

The Nios II CPU writes the induction motor's proposed smart control rule and uses interrupts to control its calculation cycle as 1 ms. When an interrupt is generated during program execution, the design first inputs the motor angle and tracking command output with the induction motor's angle counting module hardware circuit and the function generator module into the Nios II CPU. Then, it calculates the motor's control effort using a CPU control rule. Because the calculated motor control effort ranges from -5 to +5 V and the induction motor's control voltage ranges from 0 to 10 V—in which 5 V means stop, >5 V means forward rotation, and <5 V means reverse rotation—5 V must be added in the displacement method to make the control voltage range 0 to 10 V. Finally, this control effort is output to the next module.

Digital-to-Analog Control Module (DAC_1, DAC_2)

This first group of digital-to-analog (D/A) modules outputs tracking commands and the motor angle, and the output voltage ranges from -5 to +5 V. Because the DAC IC can output two groups of signals (12 bits) but can only receive 8 bits of data, the tracking command and motor angle are output four times. First, the system separates the 12 -bit tracking commands and motor angle for output into 8 low-bit data and 4 high-bit data, and outputs the data to the DAC input latch with data selection signal (A1, A0), respectively. It uses the update signal (LDAC) to transfer the input latch data to the DAC latch to output the updated data. The output order is: output 8 low-bit, tracking command data and its 4 high-bit data.

The second group of D/A modules outputs the motor control. Because the DAC IC can output two groups of signals, the two groups of 12-bit motor control effort are also separated into 8 low-bit data and 4 high-bit data and output four times. The motor control input voltage of one group ranges from 0 to 10 V for controlling induction motor; the other group ranges from -5 to +5 V for connecting the oscillometer for observation.

Performance Parameters

Because we used an FPGA in this design, both the angle sampling cycle and induction motor control frequency can reach 1 kHz. Compared to a computer or single chip, which was used in the past, the FPGA remarkably boosts the control performance. Particularly, using the Nios II embedded processor to calculate the proposed smart control rule not only simplifies writing the program with Verilog HDL

but greatly shortens the development time. Additionally, because we can add peripheral devices quickly and easily, and many intellectual property (IP) cores are available, we can easily adjust the entire control system according to our needs and amend the control rule operation parameters and algorithm rules quickly.

Design Architecture

The design's peripheral circuit, as shown in Figure 4, contains an optical encoder count circuit and two groups of D/A signal circuits with adjustable output voltage. The entire induction motor positioning control experiment environment is shown in Figure 5 and the Nios II system circuit designed with the Quartus II software is shown in Figure 6. Figure 7 shows the software flow chart of the C language program flow written using the Nios II Integrated Development Environment (IDE).

Figure 4. Peripheral Hardware Printed Circuit Board



Figure 5. FPGA-Based Induction Motor Smart Control System Experiment Environment





Figure 6. FPGA-Based Induction Motor Smart Control System Circuit Design





Design Methodology

We used the methodology described in the following steps:

- 1. Design the peripheral hardware circuit of the induction motor system, including one group of the optical encoder count circuit and 2 D/A circuit groups. The optical encoder count circuit receives the rotation angle of the optical scale/encoder climate count induction motor, and the MC14584B IC postpones the optical encoder's phase A and phase B, causing fourfold resolution. The SN74159 IC uses the fourfold frequency signal to determine whether the induction motor rotates forward or backward and calculates its rotation angle with three 4-bit SN74193 ICs, obtaining the motor's actual angle. For the D/A design, we use the AD7237 IC, which has two channels capable of outputting 0 to 10 V and -5 to 5 V voltage signals, respectively. The 74245 IC prevents the current from back-flowing to the FPGA-based development board.
- 2. Use the Quartus II software to write Verilog HDL code (motor angle count module, D/A control module) corresponding to the optical encoder count circuit and D/A circuit. Perform simulation and actual hardware testing using the Quartus II software and peripheral hardware circuit, and use an oscillogram and embedded logic analyzer to verify the functionality.
- 3. Write a divider and control command function generator module. Use the divider to manage the system hardware control cycle as 1 ms. Pinpoint the command function value as 3 bits after the decimal point and stored it in the sinusoidal function memory.

- 4. Create a Nios II embedded processor for the induction motor control system using SOPC Builder. The system contain a 32-bit Nios II CPU, floating-point custom instruction, Nios II flash storage and SDRAM, Avalon[®] tri-state bridge, system ID peripheral, JTAG UART, timer for the Nios II processor, timer for interrupts, PLL providing the CPU and SRDAM clock, and an Avalon PIO for the motor angle, tracking command, and control effort. Combine the system with the hardware program to complete the hardware structure of the induction motor control system.
- 5. Use the Nios IDE to write a software program containing the peripheral device, smart control rule of the induction motor, control program, and main program. Design a 1-kHz interrupt program that executes the induction motor's smart control rule once every 1 ms to calculate the induction motor's output control effort.
- 6. Integrate a peripheral hardware circuit, Verilog HDL hardware program, Nios II software program, and induction motor to complete the FPGA-based induction motor smart control system.
- 7. After the induction motor is booted, its actual angle is first obtained by the optical encoder's count circuit and is increased by 15 bits through the FPGA hardware's motor angle count module. It is then sent to the Nios II CPU with motor angle tracking command generated by the control command function generator module. After the program is interrupted, the count circuit calculates the motor control effort using the induction motor's smart control rule and conveys it to the D/A module. It outputs to the external D/A chip to control the induction motor after it is converted into analog voltage.
- 8. Use a digital oscillometer to observe the control result and verify the performance of the entire induction motor smart control system. Figure 8 shows the experiment result, including the tracking command and motor position and induction motor control effort. The result shows that the method proposed provides a good response after it learns the control parameters, and it is not bad when the control command changes. Therefore, we conclude that our proposed method can effectively control the induction motor's rotation angle.

Figure 8. FPGA-Based Induction Motor Smart Control System Experiment Results



9. Subject to the contest, we have no way to demonstrate the result of the FPGA-based induction motor smart control system, so we display the hardware performance by controlling a brushless DC motor when exhibiting the project. In this case, it further reveals and proves the Nios II functionality and convenience as well as the adaptability of the smart control system. When the control objects are different, we do not need to change the hardware design or the controller parameters because of the on-board artificial intelligence.

Design Features

Integrated with an artificial intelligence smart control technology, we developed this design and applied it to various examples. Applying induction motor positioning control demonstrates the superiority of the design. This design has the following features:

- **Fast learning capability and robust control rules**—Because induction motor control is complicated and motor parameters easily vary with operation, traditional methods cannot provide effective controls. Therefore, we proposed an adaptive fuzzy neutral network controller. The entire control system includes an NN controller and a compensation controller, of which the former uses a fuzzy neural network to learn and is near to an ideal controller and the latter ensures the stability of the entire system. The controller can self-regulate its parameters in real time according to the learning method deduced by the Lyapunov stability theory, so that stability of the entire closed-loop circuit system ensures convergence. An induction motor control system integrated with the these two controllers achieves robustness and accurate positioning control of the induction motor control system. The adaptive fuzzy neural network controller has learning ability and adjusts its internal parameter value promptly according to external interference, achieving good control performance. Therefore, even for control systems requiring high reliability and real-time reaction, the designer does not need to worry about poor control caused by parameter changes or equipment failure due to long-term use.
- Improved performance with FPGA—In recent years, fueled by improving IC process technology, FPGA technology has become mature and is often applied to various hardware-enabled algorithms. Moreover, more and more systems on chip (SOC) have been used, so the proposed adaptive fuzzy neural network controller is implemented by an FPGA in this design. Compared with computers that were used in the past, FPGAs reduce the size of the control system, improve flexibility, lower cost, and boost the system's calculation and execution speed. In particular, by virtue of the FPGA's reprogrammability, designers can keep changing and planning devices to cater to users' needs.
- *Coded in Verilog HDL*—In the aspect of software writing, the hardware peripheral circuit is designed in Verilog HDL. We used the Nios II processor for the control rules, which simplifies the program and accelerates design and parameter adjustment.
- Control system meets industry demands—Using an adaptive fuzzy neural network controller in an FPGA, this design offer low cost, high performance, and high reliability for the induction motor control system. A slightly modified program can be used to control various motors. Featuring small size, high flexibility, low cost, fast processing speed, short production period, and modularized design of the hardware architecture, the system does not require much effort when applied to different control systems. Therefore, the design can be used in various highly efficient controls, including on-line and real-time learning systems (e.g., a machine arm in industrial and medical fields), household robot control systems, car back-up systems, automatic driving systems, and digital wheel chair systems. We believe that the design will contribute to many areas, including industrial and medical fields.

Conclusion

Before the contest we had a basic knowledge of Verilog HDL programming and the Quartus II software, but we were not very familiar with them. We did not know anything about the Nios II processor. During the contest we improved our hardware design ability and found that the powerful Nios II processor can not only accelerate hardware design, but also simplify and facilitate the design methodology. Thus, we could use many tools—such as an embedded processor, floating-point custom instruction, and memory—to plan the hardware quickly and easily. From the results, we were impressed by the FPGA performance. Moreover, the user-friendly Nios IDE allowed us to write, compile, and execute our programs easily. This project was not only helpful for our future research, but also strengthened our competitiveness for development in this field.