To the graduate Council:

I am submitting herewith a thesis written by Nitinkumar R. Patel entitled "Sensorless Efficiency Optimized Induction Motor Drive with Neuro-Fuzzy Control". I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the Master of Science degree, with a major in Electrical Engineering.

B.K. Bose

Bimal K. Bose, Major Professor

We have read this thesis and
recommend its acceptance:

Marshall Pace

Belle R Upadhyaya

Accepted for the Council:

Lawrence Winkel

Associate Vice Chancellor and
Dean of The Graduate School

SENSORLESS EFFICIENCY OPTIMIZED INDUCTION MOTOR DRIVE WITH NEURO-FUZZY CONTROL

A Thesis

Presented for the

Master of Science

Degree

The University of Tennessee, Knoxville

Nitinkumar R. Patel

December 1996

DEDICATION

I dedicate this thesis to my parents and family, through whose nurturing spirit I have been endowed with the gifts that allow me to pursue the fine aspects of life.

ACKNOWLEDGMENTS

I express my sincere gratitude to the people through whose support and dedication, this project has been possible. My thanks first go to Dr. Rajashekara from Delphi Energy Systems for providing funding for this project. I am most grateful to my advisor, Dr. B. K. Bose, for his generosity of time, sharing of his intellectual expertise, and his wise guidance. Thanks are extended also to my committee members, Dr. Marshal O. Pace and Dr. B. R. Upadhyaya for their review and supervisory.

I recognize my colleagues from UT Power Electronics Laboratory for their thought provoking discussions and help in many aspects of my work, including: Marcelo G. Simoes, Sunil M. Chhaya, David R. Crecelius, M. Ebrahimi, Dr. Tae-Won Chun and Dhruvan Patel. Generous support was also received from Kermit O. Phipps and Doni Nastasi of the Power Electronics Application center.

Finally, my deepest appreciation goes to my wife, Manisha, for her continual encouragement and unending emotional support. My achievements while in Knoxville could not have been possible without her by my side. And, of course, to the little ones, Sanam and Saral, who brought the happiness that we have to feel in life!

ABSTRACT

Sensorless vector-controlled induction motor drives with neuro-fuzzy based intelligent control show tremendous future promise for high performance applications. This project describes a sensorless induction motor drive with stator flux oriented direct vector control particularly suited for electric vehicle type applications. The drive starts at zero speed and goes to field-weakening mode for all the four quadrants of operation. A start-up method has been developed that gracefully blends into vector control. Speed sensorless vector control at low speed has been possible by a novel method of flux vector estimation. The stator resistance variation effect on the vector control has been compensated by a new on-line fuzzy estimation technique. The efficiency of the drive at light load steady state condition has been enhanced by on-line neuro-fuzzy based search method. The total system has torque and flux control in the outer loops with synchronous current control in the inner loops.

All the proposed control and estimation strategies have been analyzed and then the complete drive system has been simulated by both SIMNON and MATLAB-SIMULINK languages to evaluate the drive performance. The control and estimation strategies have been implemented by state-of-the-art digital signal processors. A laboratory experimental system has been designed for a 5 hp drive and then tested extensively.

TABLE OF CONTENTS

1. INTRODUCTION 1

2. REVIEW OF PAST WORK 5

3. CONTROL SYSTEM DESCRIPTION 10

3.1 General Description 10

3.1.1 Vector Control 10

3.1.2 Flux Orientation 16

3.1.3 Hybrid Control Method 20

3.2 Cascaded Low-Pass Filter Integration 22

3.2.1 Description of Cascaded Low-Pass Filter 24

3.3 Feed Back Signal Estimation 34

3.4 Blaschke Equation Zero Speed Start-Up 38

3.4.1 System Operation 39

3.4.2 Start-Up Scheme 39

3.5 Quasi-Fuzzy Estimation of Stator Resistance of Induction Motor 44

3.5.1 Introduction 44

3.5.2 Description of the Estimator 46

3.6 Neuro-Fuzzy Based Light Load Efficiency Optimization 54

3.6.1 Introduction 54

3.6.2 Neuro-Fuzzy Efficiency Optimization of a Vector Control Drive 55

3.6.3 Neuro-Fuzzy Control 59

3.6.4 Transition to Optimum Transient Response Mode 65

3.7 Field-Weakening Mode 69

4. SIMULATION STUDY 74

5. SOFTWARE DESIGN 85
5.1 Software Description 85
5.2 Control Software in the DSP Board # 1 92
5.3 Control Software in the DSP Board # 2 98
5.4 Host Communication Program 103

6. SYSTEM DESIGN AND EXPERIMENTAL STUDY 105
6.1 Hardware Description 105
6.1.1 Hanning PWM 109
6.1.2 Inverter Interface 111
6.1.3 IGBT Inverter and Motor 114
6.1.4 Peripheral Circuits 117
6.2 Experimental Evaluation 126
6.3 Hardware Photos 153

7. CONCLUSION 159

REFERENCES 161

APPENDICES 166
A. Stator Flux Derivation 167
B. Blaschke Equations with Stator Flux Orientation 175
C. Simnon Programs Listing for the Drive System 178
D. Matlab-Simulink Simulation Block Diagrams for the Drive System 210
E. Simulation of Quasi-Fuzzy Stator Resistance Estimator Using C-Language 216
F. Software for DSP Boards and Host PC 229

VITA 289

LIST OF TABLES

3.1 Fuzzy rule base for estimation of steady state temperature rise.....52
3.2 Rule base for fuzzy efficiency controller.....62
4.1 Machine ratings and parameters (100 KW)..... 74
5.1 Interrupt names, section names and vector locations.....88
5.2 Memory mapped hardware addresses for DSP # 1..... 93
5.3 Memory mapped hardware addresses for DSP # 2..... 99
6.1 Induction machine parameters (5 hp)..... 117

LIST OF FIGURES

2.1 Hybrid vector control system diagram..... 7
2.2 IVC <=> DVC transition control algorithm..... 8
3.1 Vector diagram showing coupled I_a current..... 10
3.2 Induction motor and dc machine analogy in vector control..... 12
3.3 Increase of torque component..... 13
3.4 Increase of field component..... 14
3.5 Vector control implementation with machine model..... 15
3.6 D-Q equivalent circuits in synchronous reference frame..... 17
(a) q^e - axis circuit, (b) d^e - axis circuit
3.7 Stator flux decoupler control block diagram..... 19
3.8 Hybrid control system diagram..... 21
3.9 Single stage integrator input and output (with fixed time constant)..... 23
3.10 Hardware low pass filter phase shift and attenuation characteristics..... 27
3.11 Phase shift characteristics of cascaded low pass filters..... 28
3.12 Programmable time constant vs frequency plot..... 29
3.13 Programmable gain vs frequency plot..... 30
3.14 Cascaded low pass filter input and output at steady state..... 31
3.15 Cascaded low pass filter input and output at transient condition..... 32
3.16 Cascaded low pass filter implementation block diagram..... 33
3.17 Control block diagram of stator flux oriented vector control with cascaded low pass filter based flux estimation..... 35
3.18 Control block diagram of stator flux oriented vector control incorporating the Blaschke equation start-up scheme..... 40
3.19 Block diagram of Blaschke equation start-up scheme..... 42
3.20 Transition between Blaschke equation start-up and DVC control mode..... 43
3.21 Quasi-fuzzy estimation block diagram of stator resistance "R_s"..... 47

3.22 Measured stator temperature rise vs. stator current at different frequency (steady state)..... 50
3.23 Fuzzy estimation membership function..... 51
(a) stator current (1 pu = 12.7 A(rms)), (b) frequency ω_e (1 pu = 733 r/s),
(c) steady state temperature rise ΔT_{ss} (1 pu = 100 °c)
3.24 Experimental machine thermal time constant curve with frequency..... 53
3.25 Control system block diagram incorporating the neuro-fuzzy based efficiency optimization controller..... 56
3.26 Principle of efficiency optimization control with stator flux orientation..... 58
3.27 Details of the neuro-fuzzy controller..... 60
3.28 Membership functions for fuzzy efficiency control..... 61
(a) change in dc link power (P_d (pu)), (b) last change in stator flux ($L\Delta\psi^*_s$ (pu)),
(c) stator flux increment ($\Delta\psi_s$ (pu))
3.29 Topology of feed forward neural network with delayed feedback..... 66
3.30 Transition between efficiency optimization and transient response..... 68
3.31 Square-wave operation with variable battery voltage "V_d"..... 70
3.32 PWM/quasi-PWM operation by limiting phase voltage "V_s"..... 72
3.33 Field-weakening flux program block..... 73
4.1 Start-up performance of the system directly from DVC mode..... 76
4.2 Sustained operation with Blaschke mode..... 77
4.3 Start-up and transition performance between Blaschke and DVC modes..... 79
4.4 The drive system performance in the field-weakening region..... 80
4.5 Time domain optimum efficiency search curves by fuzzy efficiency controller at speed (N) = 1000 RPM and torque (T_e) = 4.5 Nm..... 81
4.6 Time domain optimum efficiency search curves by neuro-fuzzy efficiency controller at speed (N) = 1000 RPM and torque (T_e) = 4.5 Nm..... 82
4.7 Estimated stator temperature and stator resistance rise vs. stator current at different frequency (steady state) using quasi-fuzzy R_s estimator..... 84

5.1 Flow diagram for the multitasking real-time DSP code..... 90
5.2 Tasks coordination chart..... 91
5.3 Serial port communication between two DSPs..... 95
(a) transmit data through serial port, (b) receive data through serial port
5.4 Example of indexing for C program membership function evaluation..... 102
5.5 Host communication program screen interface..... 104
6.1 Hardware block diagram for induction motor drive system..... 106
6.2 Hanning PWM integrated circuit..... 108
6.3 DSPLINK interface..... 110
6.4 Hanning PWM interface with cable driver..... 112
6.5 Inverter protection circuit..... 113
6.6 Overcurrent signal processing circuit..... 115
6.7 IGBT inverter..... 116
6.8 Feedback interface circuit..... 120
6.9 DC link current sensor interface..... 121
6.10 DC link voltage sensing interface..... 121
6.11 Stator temperature interface circuit..... 123
6.12 A 300 v power supply circuit..... 125
6.13 Evaluation of Hanning PWM signals with low-pass filter at frequency of 50Hz..... 127
6.14 Evaluation of Hanning PWM signals with low-pass filter and stepped down frequency form 25 Hz to 12.5 Hz..... 128
6.15 Estimated $\sin\theta_e$ and θ_e signals in V/Hz control at 20 Hz..... 130
6.16 Phase currents i_a, i_b and i_c (2 A/div) in V/Hz control at 20 Hz..... 131
6.17 Unit vectors $\sin\theta_e$ and $\cos\theta_e$ in V/Hz at 20 Hz..... 132
6.18 Step response of ψ_s to 100% rated flux (0.2 webers/div) along with i_{ds} (4 A/div)..... 134
6.19 Input of cascaded low-pass filter $v_{qs}^{s'}$ (39 v/div) and q-axis stator flux $\psi_{qs}^{s'}$ (0.18 webers/div) after phase shift compensation in DVC..... 135

6.20 Input of cascaded low-pass filter $v_{ds}^{s'}$ (39 v/div) and q-axis stator flux $\psi_{ds}^{s'}$ (0.18 webers/div) after phase shift compensation in DVC 136
6.21 D-Q axis stationary reference frame fluxes ψ_{ds}^{s} and ψ_{qs}^{s} (0.18 webers/div) in DVC 137
6.22 Estimated $\sin\theta_e$ and θ_e signals in DVC 138
6.23 Unit vectors $\sin\theta_e$ and $\cos\theta_e$ in DVC at 23.4 Hz 140
6.24 Estimated i_{qs} (4.5 A/div) and i_{ds} (4 A/div) in DVC 141
6.25 Estimated T_e (4 Nm/div), ψ_s (0.18 webers/div) and i_{dq} (0.77 A/div) in DVC 142
6.26 Estimated angular frequency ω_e (60 r/s/div) in DVC 143
6.27 Phase currents i_a, i_b and i_c (4 A/div) in DVC 144
6.28 Step response of T_e^* from 6 Nm to 10 Nm (4 Nm/div) along with magnitude of stator flux "ψ_s" (0.18 webers/div) 145
6.29 Decoupled system behavior when torque is stepped-up from 1 Nm to 5 Nm T_e^* (1 Nm/div), ψ_s (0.18 webers/div) and i_{dq} (0.77 A/div) 146
6.30 Stator temperature estimator performance with dynamically varying stator current but at constant speed of 357 rpm 147
6.31 Stator resistance estimation by quasi-fuzzy estimator with dynamically varying stator current but at constant speed of 357 rpm 148
6.32 Stator temperature estimator performance with dynamically varying stator current but at constant speed of 725 rpm 149
6.33 Stator resistance estimation by quasi-fuzzy estimator with dynamically varying stator current but at constant speed of 725 rpm 150
6.34 Stator temperature estimator performance with dynamically varying stator current but at constant speed of 918 rpm 151
6.35 Stator resistance estimation by quasi-fuzzy estimator with dynamically varying stator current but at constant speed of 918 rpm 152
6.36 Overall view of the hardware set-up used for the drive system 153
6.37 Detail view of the IGBT inverter used in experiment 154
6.38 Detail view of the inverter protection circuit 155

6.39 Computer slots with the two DSP boards, two IO boards and the Hanning interface board..... 156
6.40 Detail view of stator temperature processing circuit..... 157
6.41 Dynamometer and induction machine..... 158

CHAPTER 1

INTRODUCTION

In recent years a considerable increase in the applications of inverter-fed induction motor drives has resulted in a great interest in devising suitable control systems. Fast transient response and high energy efficiency are extremely important for electric vehicle drive. It is also important not to use speed and flux sensors because sensors add cost, performance degradation and unreliability to the drive system. Electric vehicle requirement also includes accurate speed or torque control, an open loop control is unsatisfactory. Hence it is necessary to operate motor in closed loop mode.

In past years dc machine was famous for fast transient response, but in recent years induction motor with vector control is very popular to achieve same characteristics with more advantages over the dc machine such as higher efficiency and less maintenance. The induction machine can perform like dc machine by using vector control techniques. The flux producing component and torque producing component of the current can be decoupled and controlled independently by using vector control, thus achieving a dc machine like response. The vector control method was first developed in 1969 but was not practically used because it is computation intensive and require a fast digital signal processor (DSP) to make all the necessary calculations. With the rapid advancement of smaller and faster computers in the last decade, the vector control has become very popular among all other control algorithms to control the induction motor. The two main categories of vector control are the direct

vector control and indirect vector control methods. Between the two methods of vector control, i.e., the direct method and indirect method, the former does not require any motor speed information. In direct control method, the unit vectors for vector transformation and feedback flux are normally generated by processing the machine terminal voltages and currents. The stator flux oriented vector control (instead of conventional rotor flux oriented control) gives improved performance because it is dependent only on stator resistance parameter, so the voltage signal magnitude is somewhat higher at low frequency operation [2][3]. Only disadvantage of stator flux orientation is variation in stator resistance during the drive operation. Variation of stator resistance is compensated by using quasi-fuzzy controller. Stator temperature is being estimated by Fuzzy logic controller in order to estimate stator resistance. In a speed sensorless vector control, the machine can be started using Blaschke equations mode and then switched to direct vector control mode at very low stator frequency using the stator flux orientation. The integration problem at low speed has been solved by using cascaded low pass filters with programmable corner frequency [4]. This introduces magnitude error in the flux which was solved by frequency dependent attenuation.

As indicated before, the drive efficiency optimization is important for conservation of battery energy. In a standard vector-controlled drive, the flux is maintained at rated value in order to optimize the transient response. However it can be shown that at light load steady state condition, the core loss becomes excessive impairing the overall drive efficiency. The flux can be programmed at light load to improve the efficiency. If the flux is programmed on real time search basis for a certain load torque and speed so that the converter input power is minimum. In past Fuzzy logic based efficiency optimization control had been used which

is being replaced in this project by Neural network based efficiency optimization control. Torque control loop will help avoiding speed fluctuation while efficiency optimization controller is working because of decrease in input power. Field-weakening operation is being implemented by simple equation as a function of speed which start weakening the stator flux before the drive enter to the quasi-pwm mode.

The drive system was simulated extensively in PC-SIMNON language and MATLAB-SIMULINK toolbox using the lossy machine model for 100 KW induction machine. The simulation results confirmed that the control theory for sensorless efficiency optimized induction motor drive using neuro-fuzzy control was valid. In the simulation the drive was started with Blaschke start-up mode and then successfully transitioned to Direct vector Control mode near zero speed.

The drive sytem was implemented in hardware using a 100 KW IGBT inverter, a 5 hp Reliance induction machine (because 100 KW machine sent by Delphi Energy System was found defective), two 4-channel input/ouput data acquisition boards and two Texas Instruments TMS320C30 digital signal processors. Two of the three phase currents were sensed using NANA current sensors. The terminal voltages and currents were filtered, transformed into two phase quantities, and scaled by an analog interface circuit. The interface circuit signals were digitized for input into the DSP. A dedicated IC from the Hanning Corporation was used for vector rotation, two phase to three phase transformation, and sinusoidal pulse width modulation (SPWM) generation. The d-q axis quantities in the synchronous frame and the stator flux angle information were supplied to the Hanning PWM chip over the DSP data bus. The SPWM signals from the Hanning chip were then supplied

to the IGBT gate drivers through inverter protection circuit. The drive system control strategy was implemented in software using a modular structure. Most of the codes were written in "C" language except the interrupt service routine for reading the analog to digital converters and the Hanning chip interrupt. The software was structured with three tasks, a fast 128μsec. loop, a medium 1024μsec. loop and slower 1024msec. loop in each DSPs. The interrupt from the Hanning chip served as the software timer for the task loop timing.

Once the drive system control software was coded, the drive operation was evaluated. The drive was tested for different operating points in the torque-speed range. A 7.5 horsepower dynamometer was used to provide a load on the induction motor.

CHAPTER 2

REVIEW OF PAST WORK

This project is a continuation of the previous project, "Implementation of A Speed Sensor less Vector Controlled Induction Motor Drive With Zero Speed Start-Up", done by David Crecelius, which was sponsored by Delphi Energy Systems-G.M [1]. This chapter describes the previous work.

A new speed sensorless hybrid vector control method was illustrated in the past work which overcame the inherent problems of using only the direct vector control method. The machine started from zero speed by using indirect vector control and then transitioned to direct vector control with some develop torque near zero speed. A speed encoder was not used because at zero speed the electrical frequency of the stator is equal to the slip frequency "ω_{sl}" and indirect operation is valid. As soon as the rotor began to rotate, the drive was transitioned to the direct vector control mode . Then the machine could operate at any point in the torque-speed range except near zero speed. As the drive approached zero speed to stop, a small amount of torque current was allowed to flow, keeping the drive from having to exit direct vector control mode. In an electric vehicle, this torque would be similar to the torque produced by a car with an engine and automatic transmission while the car is in drive and no gas is being applied. To keep the car from rolling forward, a mechanical break must be applied.

Stator flux orientation was used for both indirect and direct vector control modes.

The main disadvantage of using stator flux orientation compared to rotor flux orientation is that the stator flux orientation produces a coupling effect between the d-axis and q-axis. Therefore, to overcome this effect a closed loop decoupling compensation block was added to the drive control. However, the stator flux orientation control has several advantages. First, the signals are less sensitive to machine parameter variations because the only parameter that effects the signal is the stator resistance. In the drive implementation, the stator temperature was being monitored by five thermistors mounted on the stator windings. Therefore, the stator resistance variation due to stator temperature change was being compensated. Secondly, the voltage magnitudes after the stator drop are larger near zero speed than the voltage signals after the leakage inductance drop, which are used in rotor flux orientation. When the machine speed is high, the counter EMF of the motor dominates and the stator drop becomes negligible and can be ignored.

Figure 2.1 shows the control diagram for the hybrid vector control system. A dc power feeds a PWM voltage-fed inverter that generates a variable voltage, variable frequency supply for the induction motor. A dynamic break is connected across the dc link to absorb the power during the regeneration mode. Closed loop control of the stator flux is implemented as well as synchronous current controllers for i_{qs} and i_{ds}. The terminal voltages and phase currents are used for computing and estimating the stator flux. Heavy filtering of the terminal voltages and phase currents is used to eliminate the harmonics. Therefore, a compensation algorithm is used to correct the analog filter phase shift. The angle θ_e and unit vectors $\cos\theta_e$ and $\sin\theta_e$ are computed from estimated flux signals. An inverse vector rotator transforms the currents i^s_{qs} and i^s_{ds} from the stationary reference frame to i^e_{qs} and i^e_{ds} in the

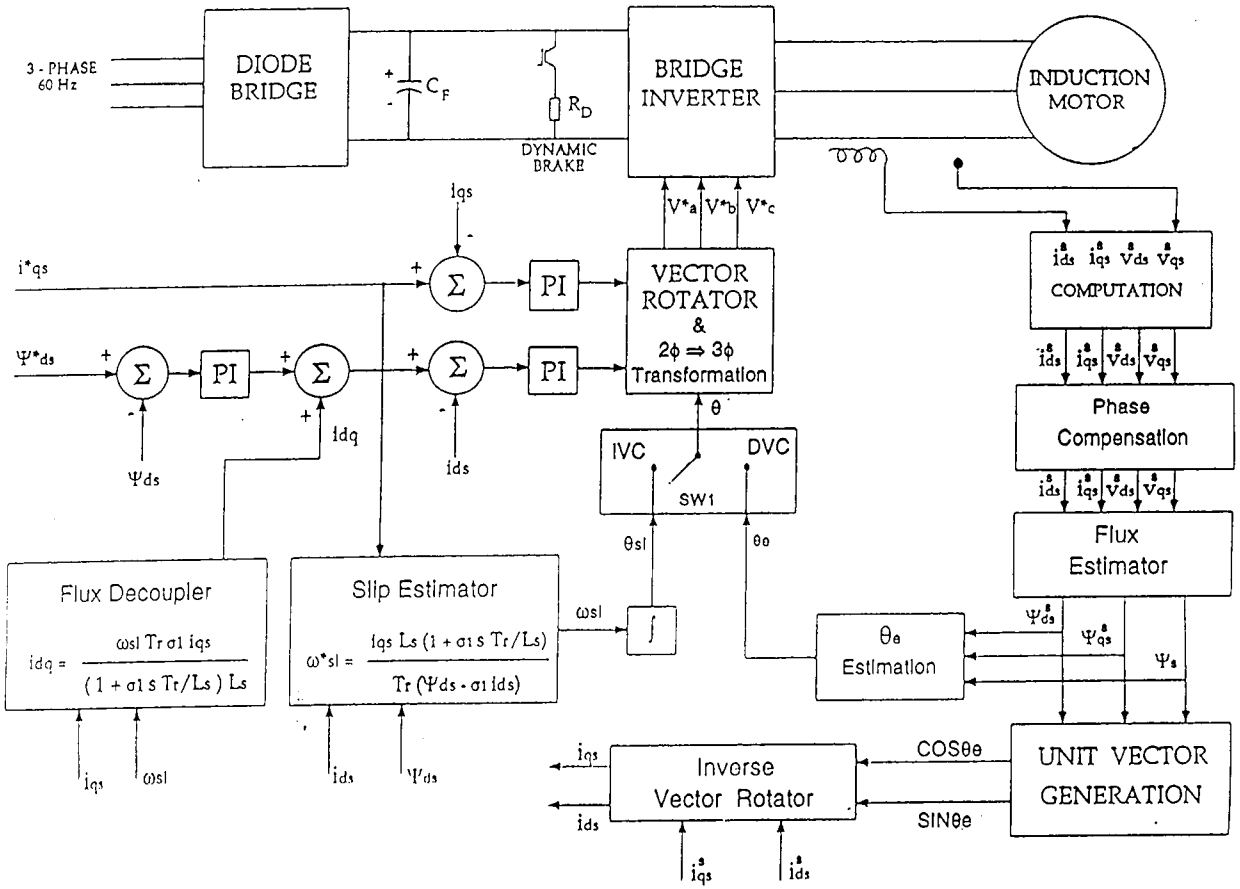


Figure 2.1 Hybrid vector control system diagram

synchronous reference frame, respectively. A feed forward stator flux decoupler is needed to completely decouple the d-q quantities when using stator flux orientation. The slip frequency ω_{sl} is estimated for use in the indirect vector control mode which is used during start-up of the drive. A transition block is shown which switches the drive from indirect vector control mode to direct vector control mode. The torque of the drive is controlled by the I^*_{qs} current and the stator direct axis flux "Ψ^*_s" is controlled by the feedback loop. All of the feedback estimations are computed from the terminal voltages and phase currents. Figure 2.2 shows the transition algorithm between indirect and direct vector control modes near zero speed.

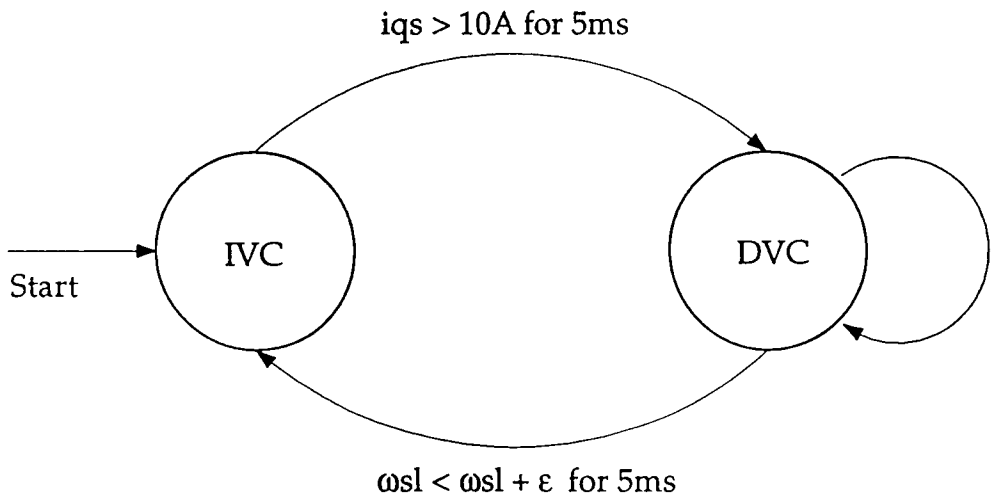


Figure 2.2 IVC <=> DVC transition control algorithm

The hybrid vector control drive was simulated by Marcelo G. Simoes using the SIMNON computer simulation software. The simulation results confirmed that the control theory for hybrid vector control method was valid. In the simulation, the drive was started with indirect vector control and then successfully transitioned to direct vector control mode near zero speed. The hybrid vector control drive was implemented in hardware using a five horsepower IGBT inverter, a five horsepower Reliance induction motor, and a Texas Instruments TMS320C30 digital signal processor.

CHAPTER 3

CONTROL SYSTEM DESCRIPTION

3.1 General Description

3.1.1 Vector Control

In the scalar control methods of VFI and CFI drives, the voltage or current and the frequency are the basic control variables of the induction motor. In VFI, for example, both the torque and air gap flux are functions of voltage and frequency. This coupling effect is responsible for sluggish response of the induction motor. Figure 3.1 shows a vector diagram of how a coupled system would have a component of the I_a vector (torque vector) in both Ψ_a and Ψ_f axes. The response time of the drive is lengthened because the torque sensitivity is diminished. This limitation can be overcome by applying vector or field-oriented control methods. Vector control methods have become very popular for high performance industrial drives.

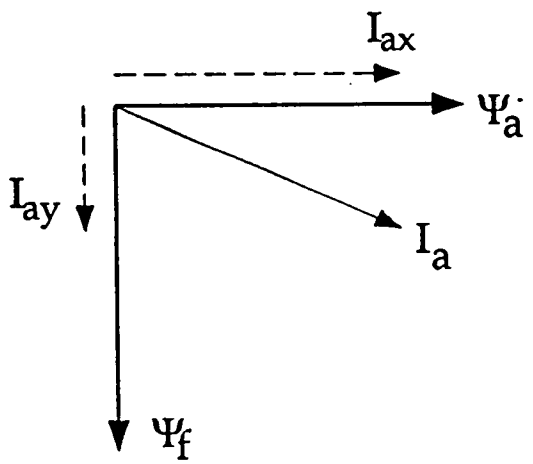


Figure 3.1 Vector diagram showing coupled I_a current.

The method of vector control allows the ac induction motor to behave as a dc machine having a separately excited armature and field coil. In a dc machine the torque is given by equations 3.1 and 3.2.

$$T_e = K_t \psi_g i_a \quad \text{..... (3.1)}$$

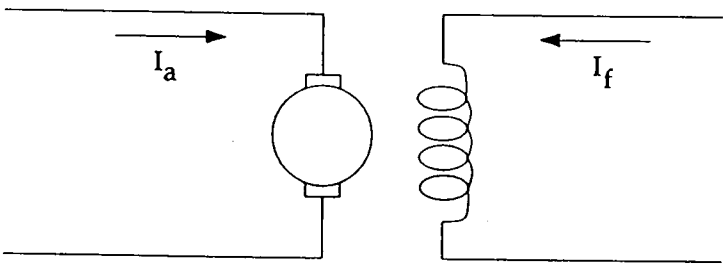
$$T_e = K'_t i_f i_a \quad \text{..... (3.2)}$$

Where, K_t is the torque constant, Ψ_g is the air gap flux, i_a is the armature or torque component of the current, and i_f is the field or flux component of the current. Therefore, the variables that control the dc machine are I_a (torque) and I_f (flux) component of current. These variables can be considered to be orthogonal or decoupled since changing one variable will not affect the other. The armature current and the flux current can be controlled independently. Usually for the fast transient response the flux is set constant to the rated value and then torque is varied as desired by changing I_a. Since the flux component current I and the torque component current I_a are decoupled from each other, the torque sensitivity will remain at the maximum value in both the steady state and transient modes [5].

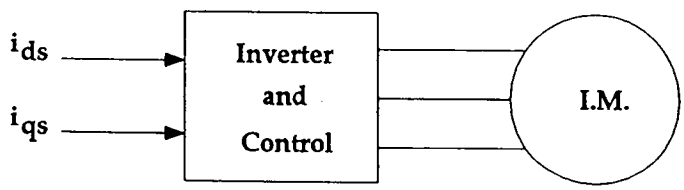
If the induction machine is considered in the synchronous reference frame then all of the sinusoidal signals are dc quantities. This means that the d-q axis quantities of the stator current are I^*_{ds} (flux component) and I^*_{qs} (torque component), respectively. Torque in the induction machine can be expressed as shown in equations 3.3 and 3.4. Figure 3.2 shows a comparison between the dc machine and the induction machine using vector control.

$$T_e = K_t|\psi_m|i_{qs} \quad \text{..... (3.3)}$$

$$T_e = K'_t i_{ds} i_{qs} \quad \text{..... (3.4)}$$



(a)



(b)

Figure 3.2 Induction motor and dc machine analogy in vector control

The current components i_{ds} and i_{qs} can be used as the control vectors for controlling the induction machine if they are considered in a synchronously rotating d^e-q^e reference frame. The phasor diagram in Figure 3.3 can aid in explaining how the vectors are related. The air gap voltage V_g is aligned with the q^e axis and the $_sI$ vector lags the $_gV$ vector by 90 º0 . Equations 3.5, 3.6, and 3.7 express the vector relationship of Figure 3.3.

$$I_{ds} = I_s \sin\theta \quad \text{..... (3.5)}$$

$$I_{qs} = I_s \cos\theta \quad \text{..... (3.6)}$$

$$I_s = \sqrt{i^2_{ds} + i^2_{qs}} \quad \text{..... (3.7)}$$

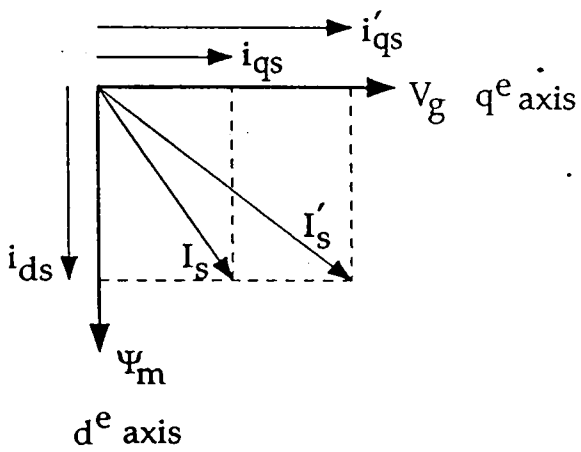


Figure 3.3 Increase of torque component

The variables i_{ds} and i_{qs} are orthogonal in nature which implies that they are decoupled and can be independently varied without affecting the other component. Figure 3.3 shows an increase in the i_{qs} vector and the corresponding change in the $\underline{I}_s$ vector [5]. Figure 3.4 shows the same vector diagram except that the i_{ds} vector has been increased and the corresponding change in the I_s vector can be seen [5]. For normal operation of the induction motor, the flux current i_{ds} is maintained constant at the rated value and the torque is varied by varying the i_{qs} component, as in a dc machine.

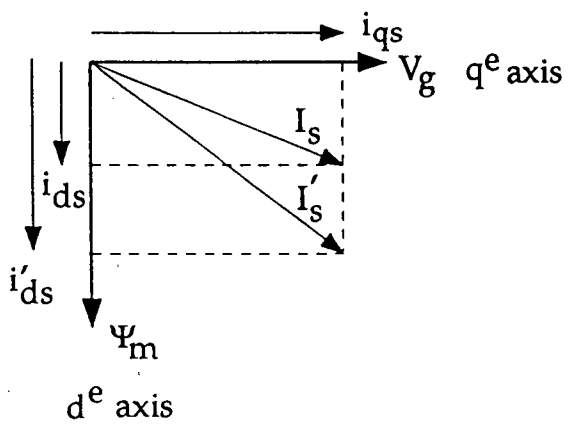


Figure 3.4 Increase of field component

When implementing vector control the i^s_{ds} and i^s_{qs} vectors must be transformed into the synchronous reference frame and then transformed back into the stationary reference frame, producing reference signals for the inverter. The transformation between reference frames is computed from unit vectors that are aligned with the flux. Figure 3.5 [5] shows a simple implementation of vector control using an induction machine model with the inverter omitted.

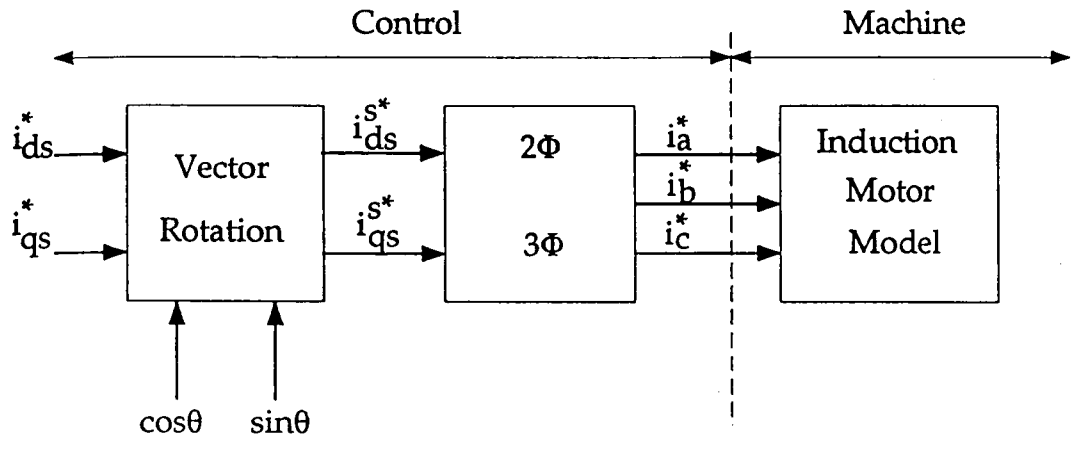


Figure 3.5 Vector control implementation with machine model

The commanded signals i_{ds} and i_{qs} are transformed into the stationary reference frame by the unit vectors $\cos\theta_e$ and $\sin\theta_e$. The resultant signals i^s_{ds} and i^s_{qs} are sinusoidal quantities that are then are then converted from two phases to three phases. The three signals i_a, i_b, and i_c are the reference signals given to the inverter to drive the motor. The unit vectors assure the alignment of i_{ds} with the Ψ_m vector and i_{qs} with the V_g vector. Four quadrant operation of the drive using vector control is easily implemented with the appropriate control. Therefore, the drive is suitable for high performance industrial applications where dc machines

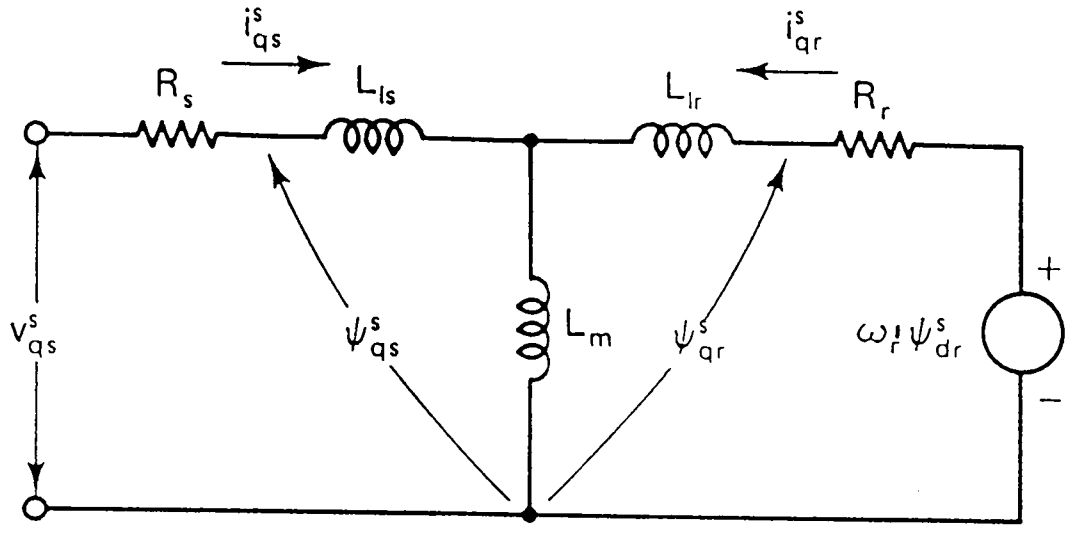
have been traditionally used.

There are essentially two general methods of vector control. One, called the direct method, was developed by F.Blaschke and the other, known as the indirect method, was developed by K.Hasse. The methods relate to how the unit vector signals are generated. In direct vector control method unit vectors are generated from air gap fluxes. The air gap fluxes can be estimated from the stator voltage and current signals. The direct method of vector control can be applied typically above 10% of the base speed because of the difficulty in accurate flux signal synthesis at low speed. In the indirect method the synthesis of the unit vectors is dependent on the speed information. The indirect vector control works better at low speed than the direct vector control method because unit vector generation is not dependent on the machine terminal conditions. This method is applicable from zero speed to full speed. However, a rotor position signal becomes mandatory in this method. And also the controller dependent on the machine parameters, and for ideal decoupling the controller parameters should match the machine parameters, which is very difficult to achieve.

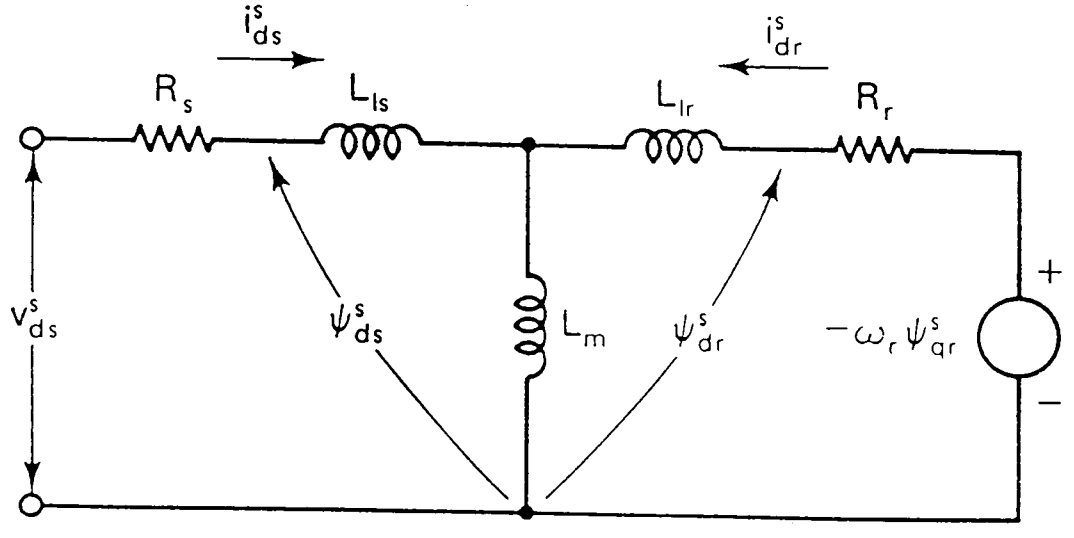
In the present work only direct vector control has been used and the start-up of the drive is performed by using current model of the stator flux oriented Blaschke equations.

3.1.2 Flux Orientation

The flux vector by which the unit vectors are aligned can have one of the three orientations. The possible flux orientations are stator flux, air gap flux, or rotor flux. Figure 3.6 shows the d-q equivalent circuits in the synchronously rotating reference frame [5]. The stator flux Ψ_s, air gap flux Ψ_m, and rotor flux Ψ_r are shown.



(a)



(b)

Figure 3.6 D-Q equivalent circuits in synchronous reference frame

(a) q^e - axis circuit; (b) d^e - axis circuit

The air gap flux Ψ_m is typically not used in practical applications due to the need of mounting air gap flux sensors on the stator and the air gap flux orientation neglects the rotor leakage inductance which can result in an undesirable stability problem [6]. To compensate for the rotor leakage inductance the rotor flux orientation can be used. The rotor flux can be calculated from the air gap fluxes or can be computed from the phase currents and rotor speed ω_r. Equations 3.8 and 3.9 are used to calculate the rotor flux signals from the air gap flux. Equations 3.10 and 3.11 are used to calculate the rotor flux signals from

$$\psi^s_{qr} = \frac{L_r}{L_m}\psi^s_{qm} - L_{lr} i^s_{qs} \quad \text{..... (3.8)}$$

$$\psi^s_{dr} = \frac{L_r}{L_m}\psi^s_{dm} - L_{lr} i^s_{ds} \quad \text{..... (3.9)}$$

$$\frac{d\psi^s_{qr}}{dt} = \frac{L_m}{T_r} i^s_{qs} + \omega_r \psi^s_{dr} - \frac{1}{T_r}\psi^s_{qr} \quad \text{..... (3.10)}$$

$$\frac{d\psi^s_{dr}}{dt} = \frac{L_m}{T_r} i^s_{ds} + \omega_r \psi^s_{qr} - \frac{1}{T_r}\psi^s_{dr} \quad \text{..... (3.11)}$$

the phase currents and the rotor speed ω_r. Rotor flux orientation is the most widely used form of flux orientation. The rotor flux orientation gives better performance because the rotor flux is actual flux which is producing the torque.

Stator flux orientation is typically not used very often. When using stator flux

orientation the d-q axes are not completely decoupled and a special decoupling block must be added to achieve true orthogonal control. Equations 3.12 and 3.13 show how the stator flux can be calculated from the machine terminal voltages and the phase currents. It can be seen by equations 3.14 and 3.15 that when using the stator flux, coupling exits between the d and q axes. If the torque component i_{qs} changes, the flux will also be changed. Therefore, a feed forward control

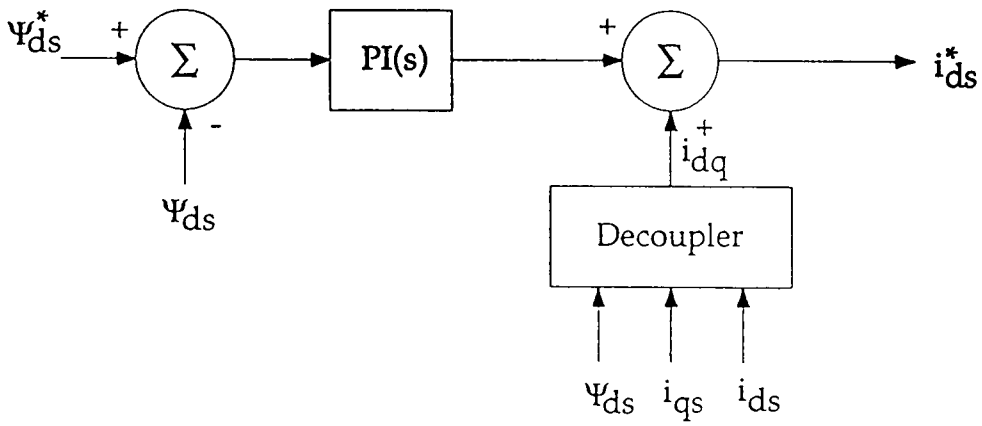


Figure 3.7 Stator flux decoupler control block diagram

method is used to completely decouple the i_{ds} and i_{qs} components. Figure 3.7 shows a block diagram of the feed forward stator flux decoupling method. It can be shown that equation 3.16 can be used to calculate the decoupling factor i_{dq}. The detailed derivations for the three flux orientations and equations 3.8 to 3.16 can be found in Appendix A.

3.1.3 Hybrid Control Method

The drive in direct vector control cannot be started from zero speed, as stated earlier, and has difficulty operating near zero speed. This limitation has been overcome by using Blaschke equation start-up in stator flux orientation. The hybrid control method utilizes the stator flux orientation for improved low speed operation. The drive is initially started using the Blaschke equation at zero speed. As the drive begins to rotate and before Blaschke mode becomes invalid, the drive is transitioned to the direct vector control mode. The drive then operates using the direct vector control method in the entire torque-speed region except zero speed. Figure 3.8 shows the control diagram for the hybrid control system. A dc power supply feeds a PWM voltage-fed inverter that generates a variable voltage, variable frequency supply for the induction motor. A dynamic break is connected across the dc link to absorb the power during the regeneration mode. Closed loop control of the developed torque, stator flux and synchronous currents i_{qs} and i_{ds} are implemented. The terminal voltages and phase currents are used to compute and estimate stator flux "Ψ_s". Due to heavy filtering of the terminal voltages and phase currents, to eliminate harmonics, a compensation algorithm is used to correct the analog filter phase shift. Cascaded low pass filters with programmable time constants are used to take care of the dc offset problem near zero speed, which gives improved performance of the direct vector control near zero speed [4]. The angle θ_e and unit vectors cos θ_e and sin θ_e are computed from the estimated d-q axes flux signals. An inverse vector rotator transforms the stationary reference frame currents i^s_{qs} and i^s_{ds} to synchronous reference frame i_{qs} and i_{ds}.

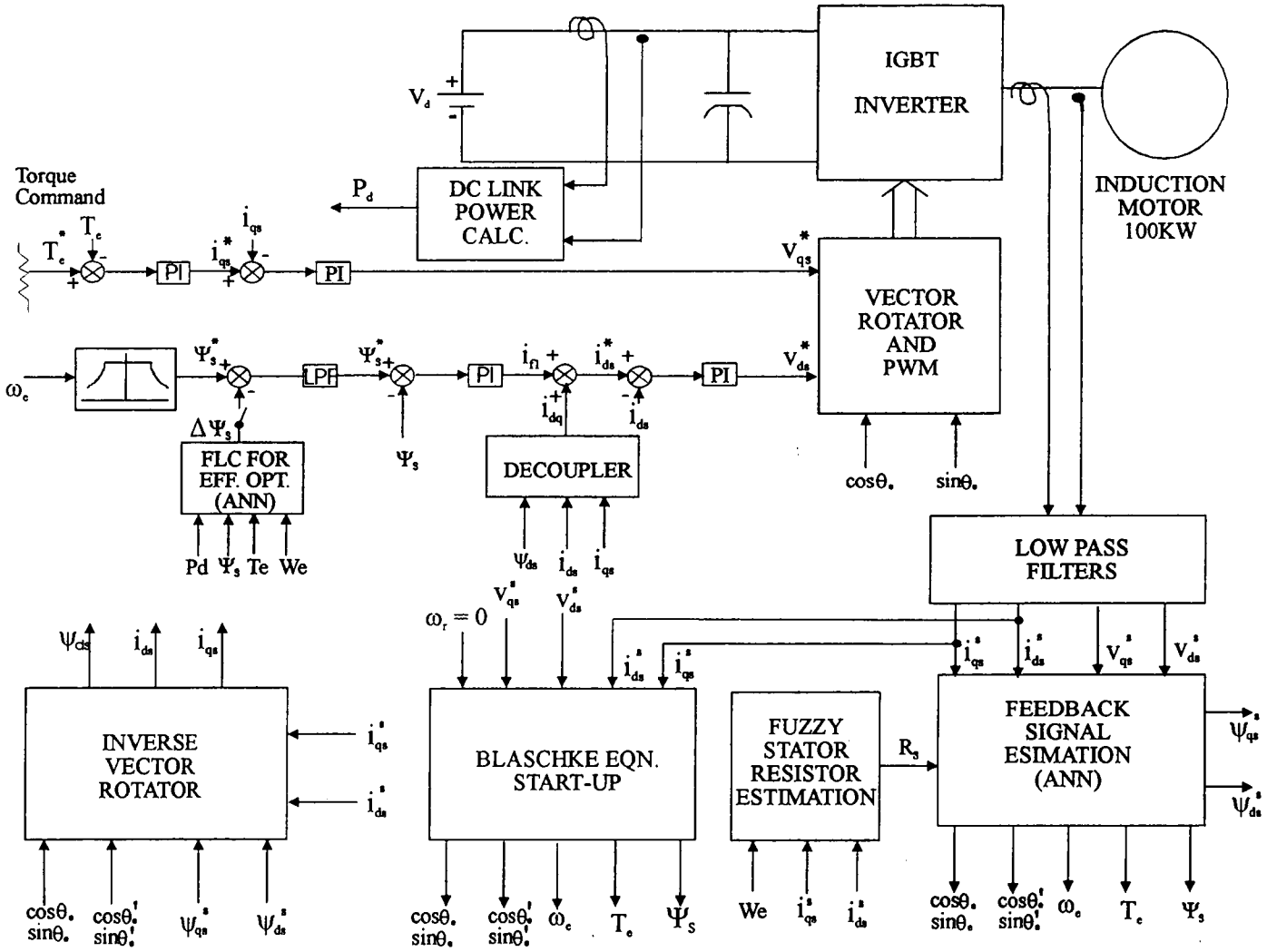


Figure 3.8 Hybrid control system diagram

A feed forward stator flux decoupler block is added to decouple the d-q quantities completely when using stator flux orientation. The Blaschke start-up block computes the d-q axes fluxes from the phase current information to start the system at zero speed. The transition block switches the drive from the Blaschke mode to direct vector control mode. The fuzzy stator resistance estimator block estimates the stator resistance from electric speed "ω_e" and stator current "I_s" signals. Artificial neural network-based efficiency optimization block optimizes the machine performance by decreasing overall losses inside the machine with the on line search of the optimum stator flux. A field weakening block is shown which weaken the stator flux after the base speed is reached. All of the feed back signals are estimated from the terminal voltages and phase currents.

3.2 Cascaded Low-Pass Filter Integration

Currently, speed and flux sensorless vector control of induction motor drive is receiving wide attention in the literature. Inspite of large effort, precision synthesis of speed and flux vector signals, particularly near zero speed operation, yet remains a challenge. The stator flux oriented vector control of induction motor drive [2] is recently gaining more popularity because the feedback signal estimation accuracy is dependent only to the stator resistance variation which can be compensated somewhat easily. Besides, the signal voltage behind the stator resistance drop that is to be integrated for stator flux estimation is somewhat larger than that for rotor flux estimation. This signal magnitude tends to be very low near zero speed operation. If an attempt is made to integrate this signal directly near zero speed, a dc offset will appear at the output as shown in Figure 3.9.

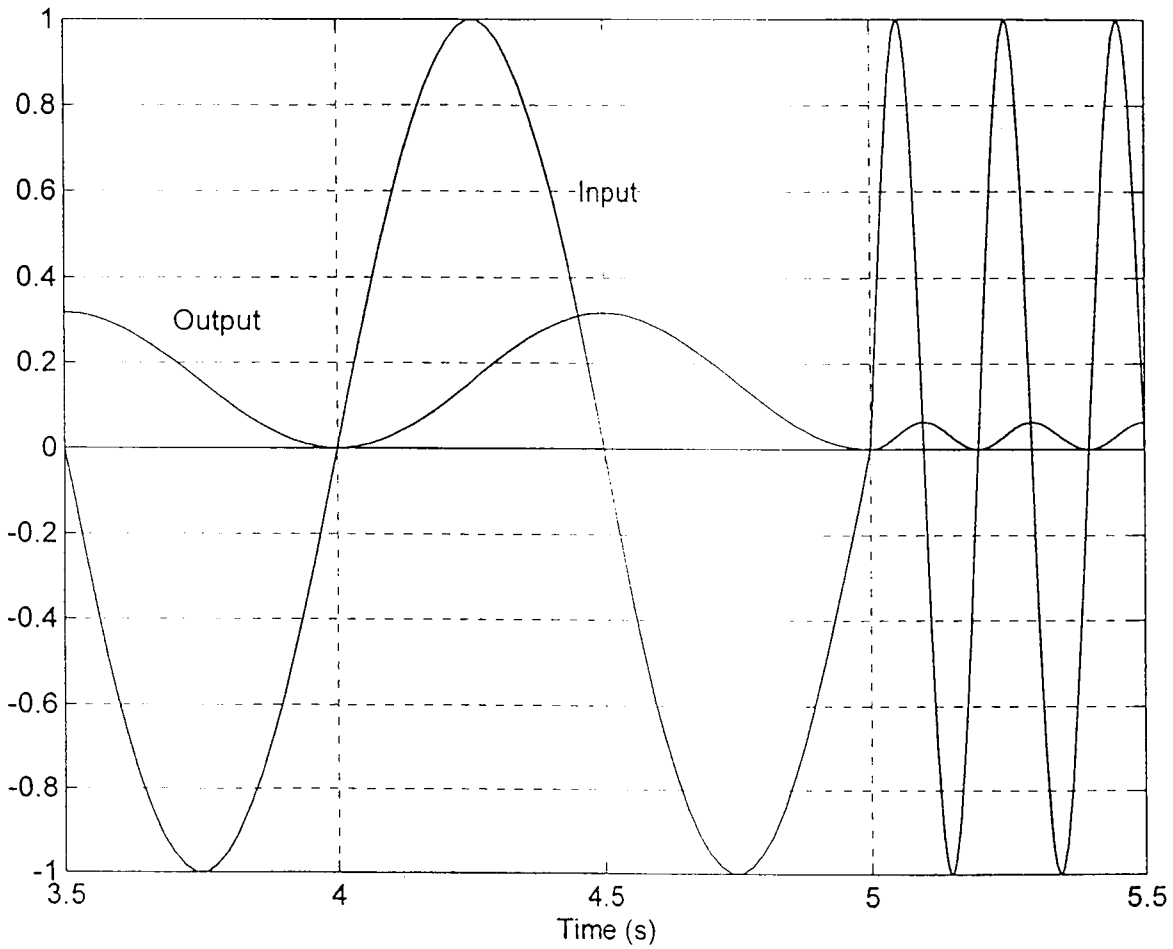


Figure 3.9 Single stage integrator input and output (with fixed time constant).

This problem can be solved by programmable multistage low-pass filter implemented by microcomputer/DSP based software. For a stator flux oriented vector-controlled EV type of drive where speed signal is not needed, this type of sensorless control becomes very attractive. In this chapter, the concept of programmable cascaded low-pass method of flux estimation will be expanded, systematically analyzed, designed and verified by simulation study for EV drive application.

3.2.1. Description of Cascaded Low-Pass Filter

Since the drive has to operate in wide frequency range (including zero speed condition), a single stage integrator has to be designed with very large time constant. This causes the problem of dc offset and its very slow decay as dictated by the time constant. If, on the other hand, a single stage integrator is resolved into a number of cascaded low-pass filters with short time constant, the problem of dc offset decay time can be sharply attenuated. Consider a low-pass filter with transfer characteristic as

$$Y/X = 1/1 + j\tau\omega_e \quad \text{.....} \quad (3.12)$$

where τ = time constant and ω_e = frequency. The phase lag and gain (or attenuation) of the filter at frequency ω_e can be given, respectively, as

$$\phi = \tan^{-1}(\tau\omega_e) \quad \text{.....} \quad (3.13)$$

$$K = (Y/X) = 1/\sqrt{1 + (\tau\omega_e)^2} \quad \text{.....} \quad (3.14)$$

If n number of filters are cascaded, the total phase shift angle and gain are given, respectively, as

$$\phi_T = \phi_1+\phi_2+.....+\phi_n = \tan^{-1}(\tau_1\omega_e)+\tan^{-1}(\tau_2\omega_e)+.....+\tan^{-1}(\tau_n\omega_e) \quad ..... \quad (3.15)$$

$$K_T = K_1K_2.....K_n = 1/\sqrt{[1+(\tau_1\omega_e)^2][1+(\tau_2\omega_e)^2].....[1+(\tau_n\omega_e)^2]} \quad ..... \quad (3.16)$$

If all the filter stages are identical, the corresponding expressions are

$$\phi_T = n\ \phi = n\ \tan^{-1}(\tau\omega_e) \quad ..... \quad (3.17)$$

$$K_T = nK = 1/\sqrt{[1+(\tau\omega_e)^2]^n} \quad ..... \quad (3.18)$$

If the cascaded filter performs integration of a sinusoidal voltage, then $\phi_T = 90^0$ and $G*K_T = 1/\omega_e$ where G = gain compensation needed for integration. Substituting these conditions in (3.17) and (3.18),

$$\tau = (1/\omega_e)\tan(90^0/n) \quad ..... \quad (3.19)$$

$$G = (1/\omega_e)\ \sqrt{[1+\tau\omega_e)^2]^n} \quad ..... \quad (3.20)$$

The above equations give the parameters τ and G as function of frequency. Large number of n is desirable but the software computation burden becomes heavy.

In present application hardware low pass filter is being used at the input of the programmable low pass filter to filter the noise from the input signals.

Figure 3.10 shows the phase shift and attenuation characteristics of the hardware low pass filter. Where, τ_1 is time constant of the hardware low pass filter and ϕ_h is the phase shift of the hardware low pass filter. In order to get the 90^0 phase shift and correct attenuation at the output of the cascaded low pass filter, phase shift ϕ_h and gain introduced by hardware low pass filter should also be considered. Figure 3.11 shows the phase shift characteristics of three cascaded low pass filters along with one hardware low pass filter. Where α is phase shift introduced by each low pass filter. Therefore, the correct gain and time constant expressions can be generalized as

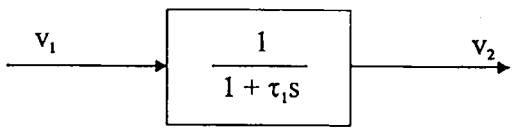
$$\tau = (1/\omega_e)\ \tan[(1/n)[\tan^{-1}(\tau_h\omega_e)+90^0]] = f(.)\ \omega_e \quad \ldots\ldots\ldots\ldots \quad (3.21)$$

$$G = (1/\omega_e)\sqrt{[1+(\tau\omega_e)^2]^n[1+(\tau_h\omega_e)^2]} = g(.)\omega_e \quad \ldots\ldots\ldots\ldots \quad (3.22)$$

where τ_h = hardware filter time constant (0.16 ms). Note that the hardware filters can be conservatively designed without fear of phase drift and attenuation of the signals because these are compensated in the software filters.

Figures 3.12 and 3.13 show the plots of τ and G as function of ω_e according to the above equations. Note that with programmable τ and G, the phase shift and gain are equal in all the filter stages and they remain invariant with frequency.

Figure 3.9 shows the performance of a single stage integrator with large and fixed time constant where the dc offset is evident. Figure 3.14 shows performance of the cascaded filter integrator at low frequency, and Figure 3.15 shows its transient performance. Note that



$$\text{Gain } (k_1) = \frac{1}{\sqrt{1+(W_e\tau_1)^2}} \quad \cdots\cdots\cdots \quad (1)$$

$$\text{Phase Shift } (\Phi_1) = \tan^{-1}(W_e\tau_1) \quad \cdots\cdots\cdots \quad (2)$$

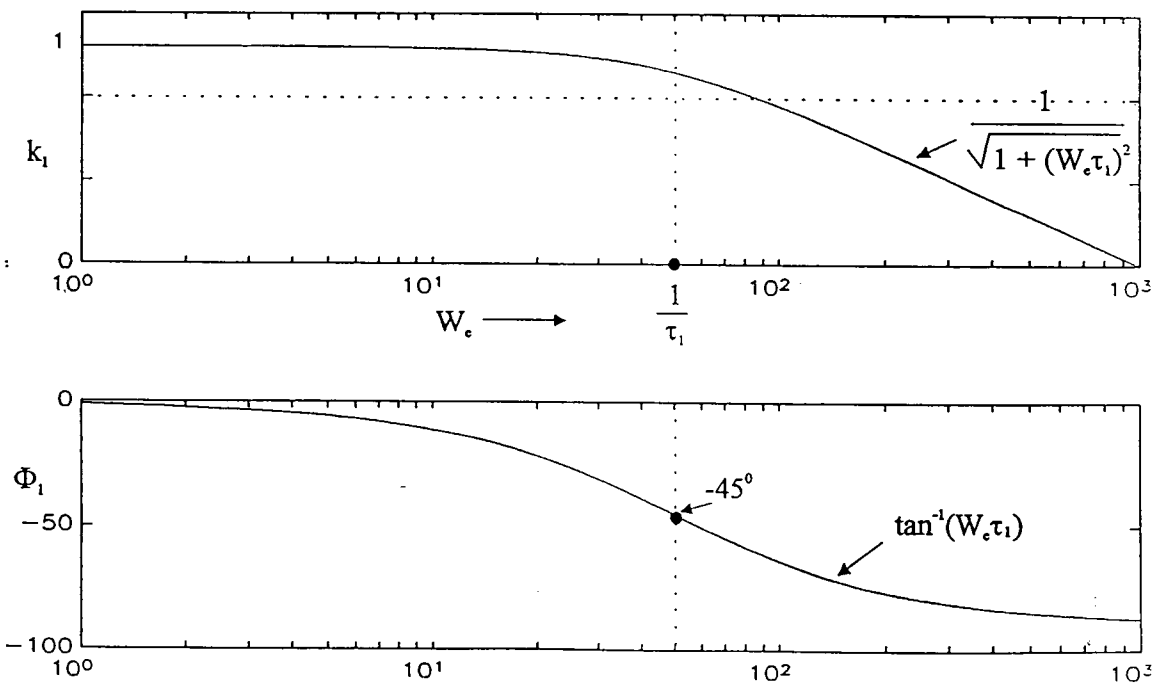


Figure 3.10 Hardware low pass filter phase shift and attenuation characteristics

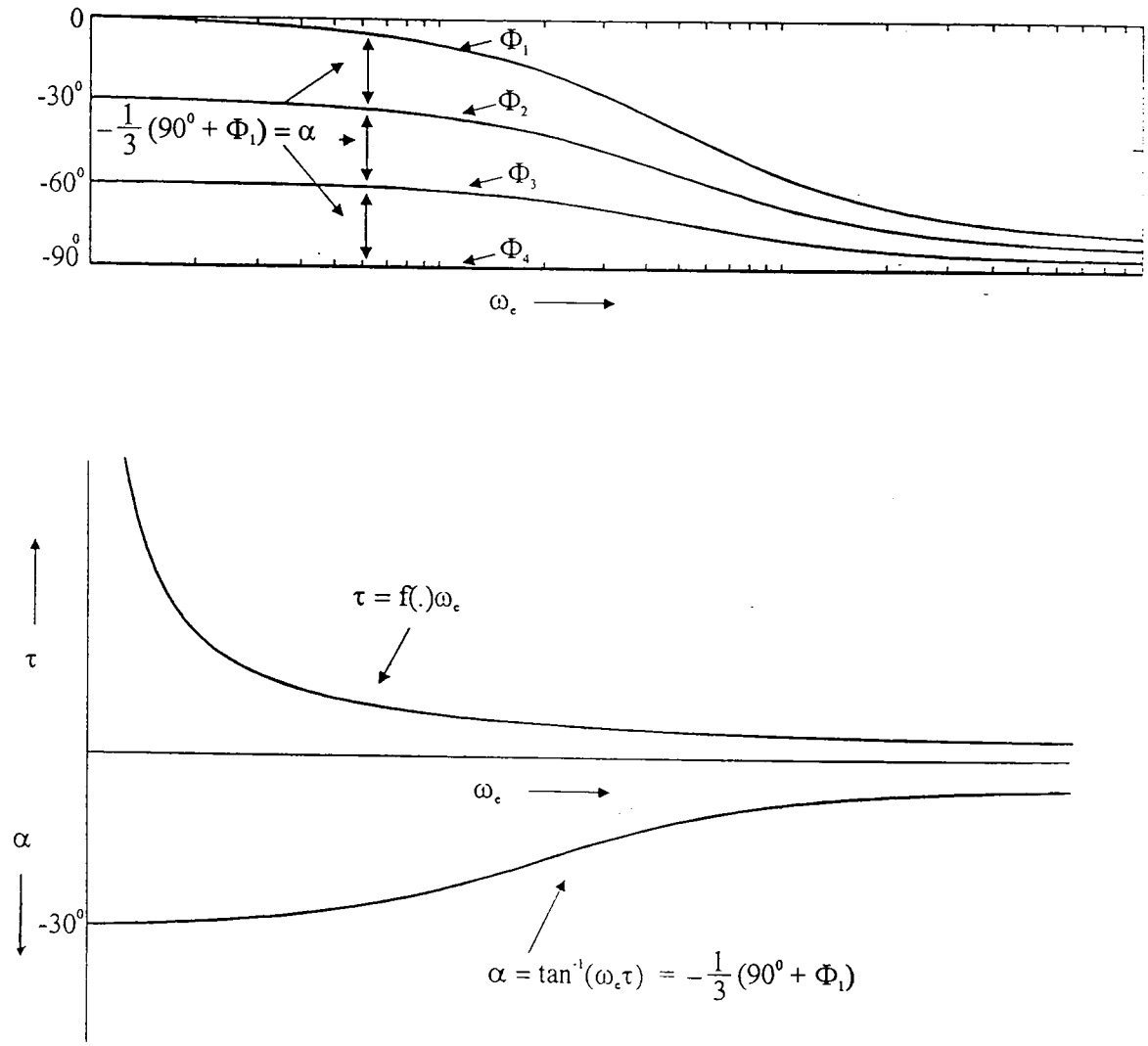


Figure 3.11 Phase shift characteristics of cascaded low pass filters

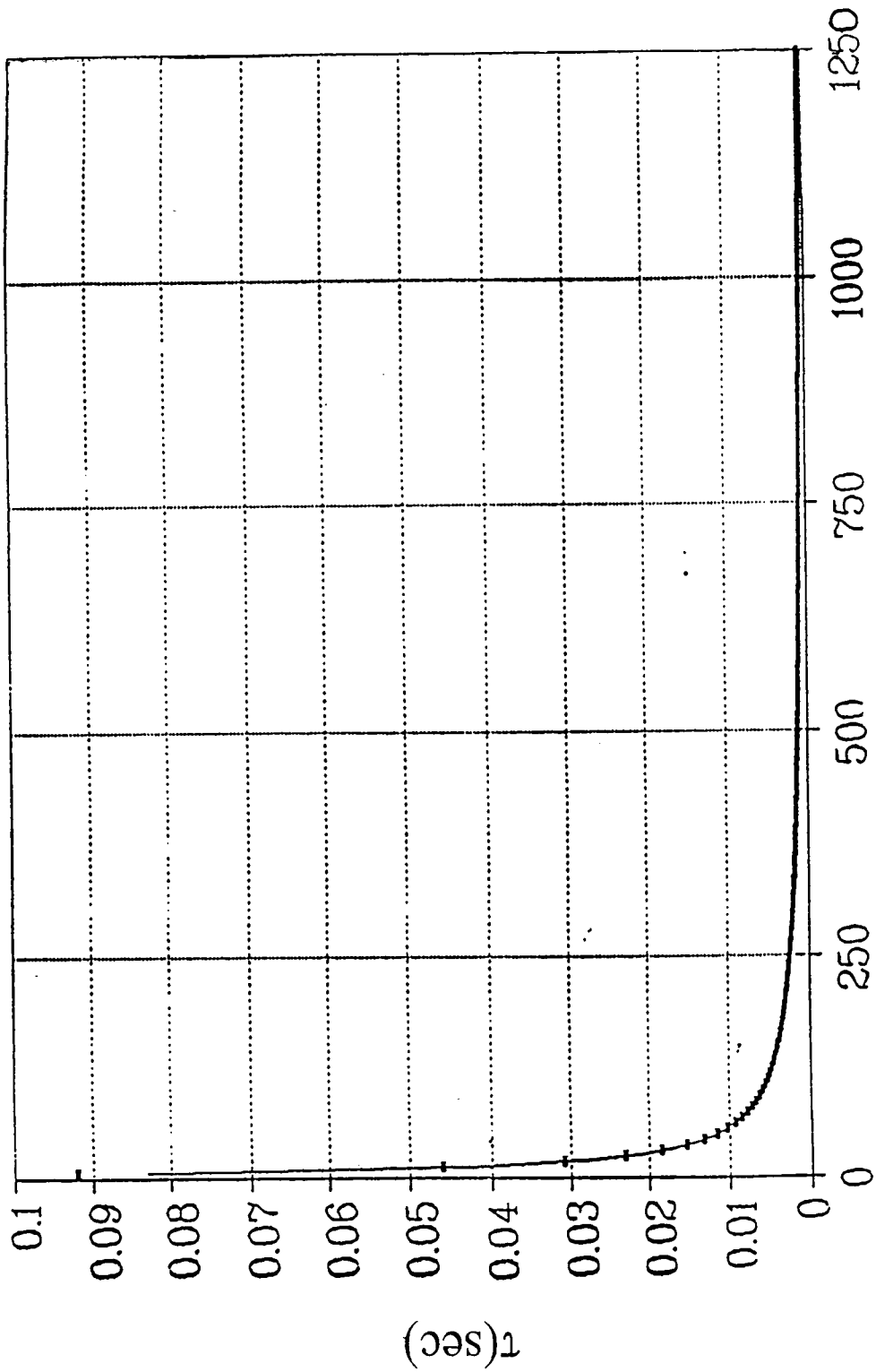


Figure 3.12 Programmable time constant vs frequency plot.

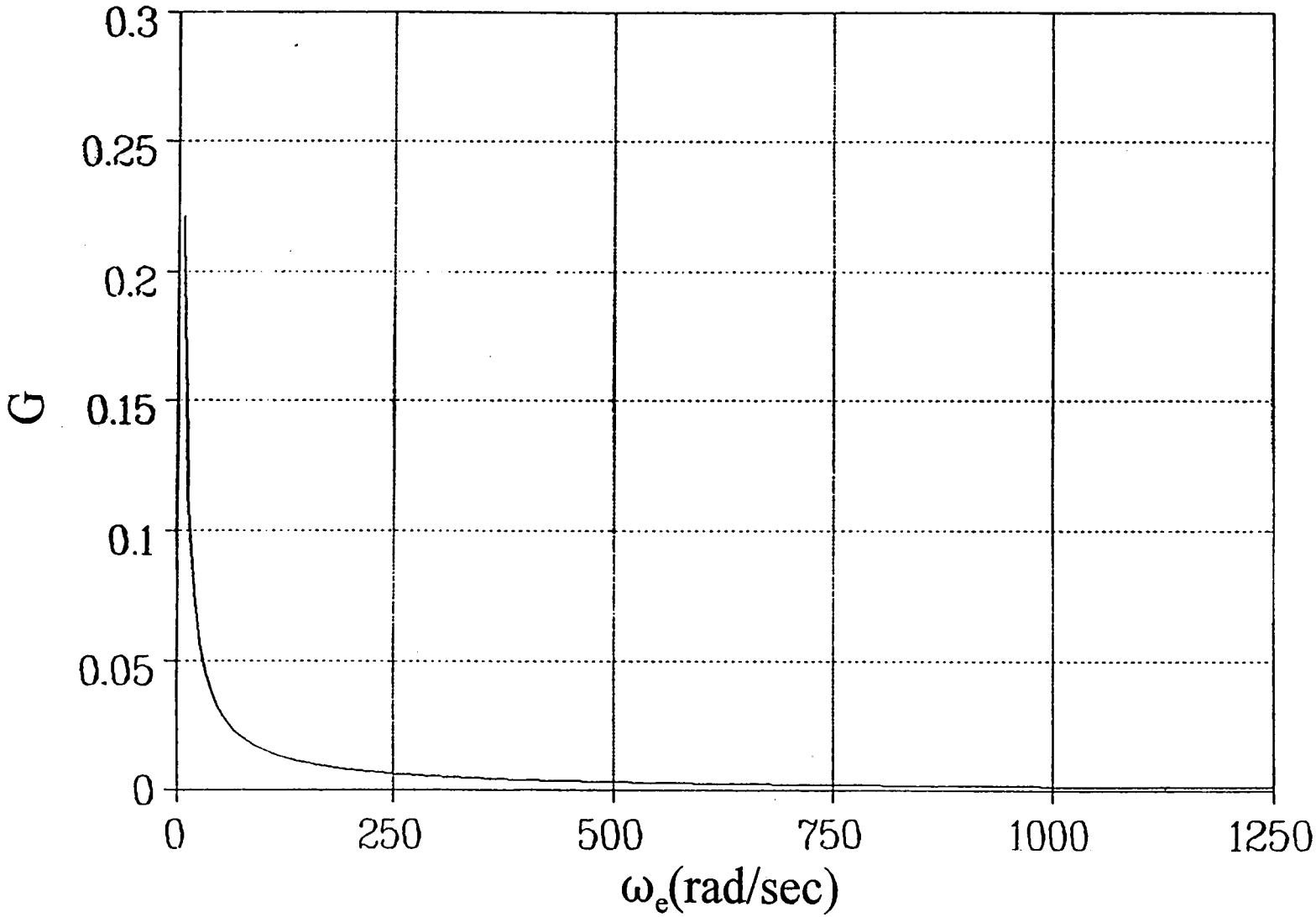


Figure 3.13 Programmable gain vs frequency plot.

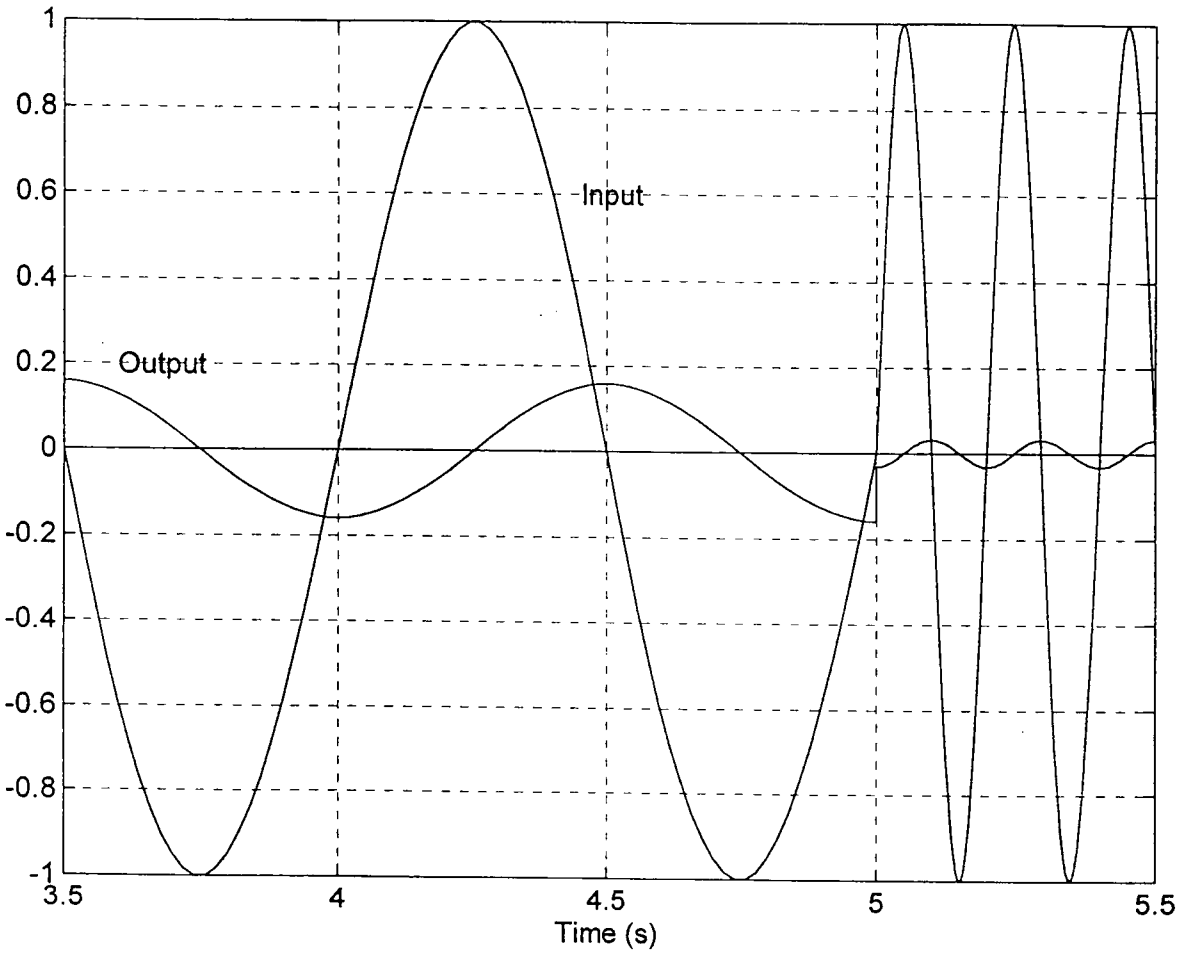


Figure 3.14 Cascaded low pass filter input and output at steady state.

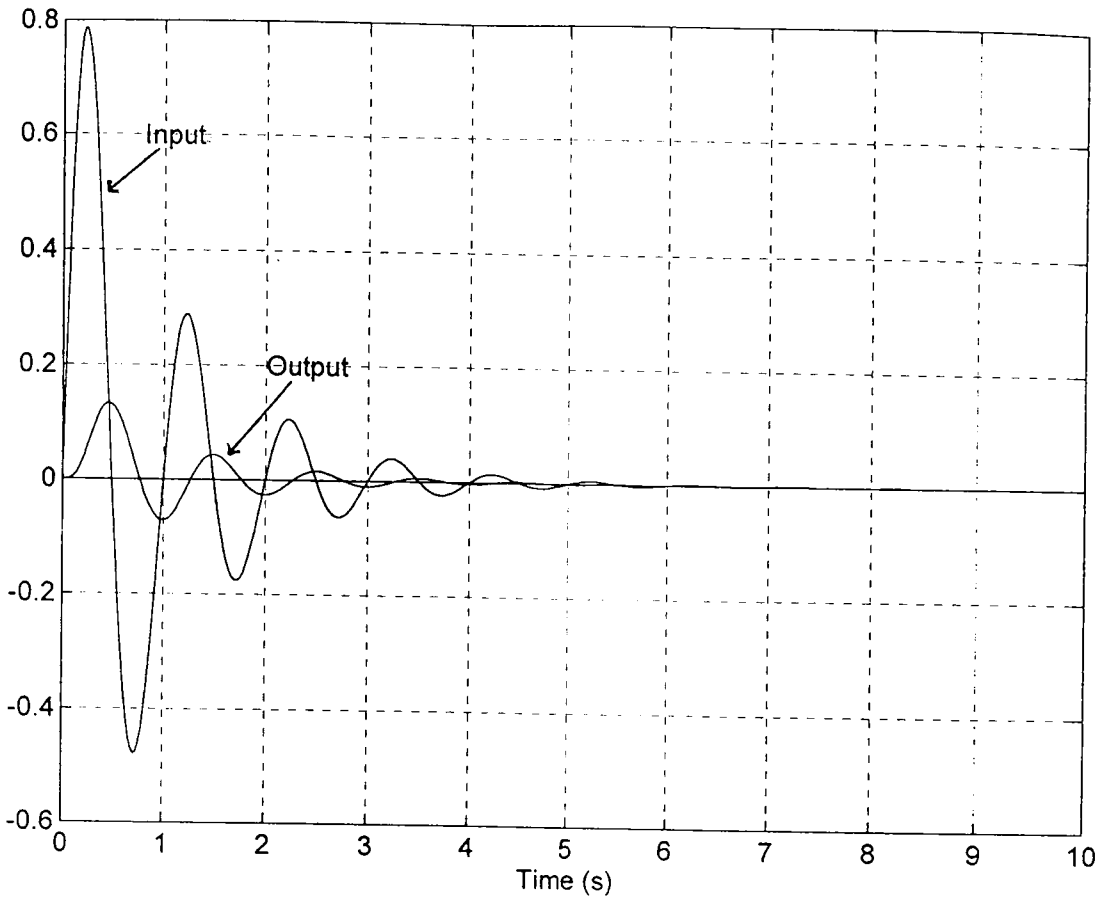


Figure 3.15 Cascaded low pass filter input and output at transient condition.

the filter performance depends on reasonably accurate computation of ω_e signal which tends to be difficult at very low frequency (fraction of a Hz.). Figure 3.16 shows the details of the cascaded low pass filter implementation. This block computes the fluxes ψ_{ds}^{s} and ψ_{qs}^{s} using programmable cascaded low pass filters. The fluxes ψ_{ds}^{s} and ψ_{qs}^{s} are further phase shifted by angle ϕ_h to generate $\psi_{ds}^{s'}$ and $\psi_{qs}^{s'}$. These auxiliary fluxes which are at phase quadrature with the respective currents i_{ds}^{s} and i_{qs}^{s} help to generate the unit vector signals for inverse vector rotation of currents.

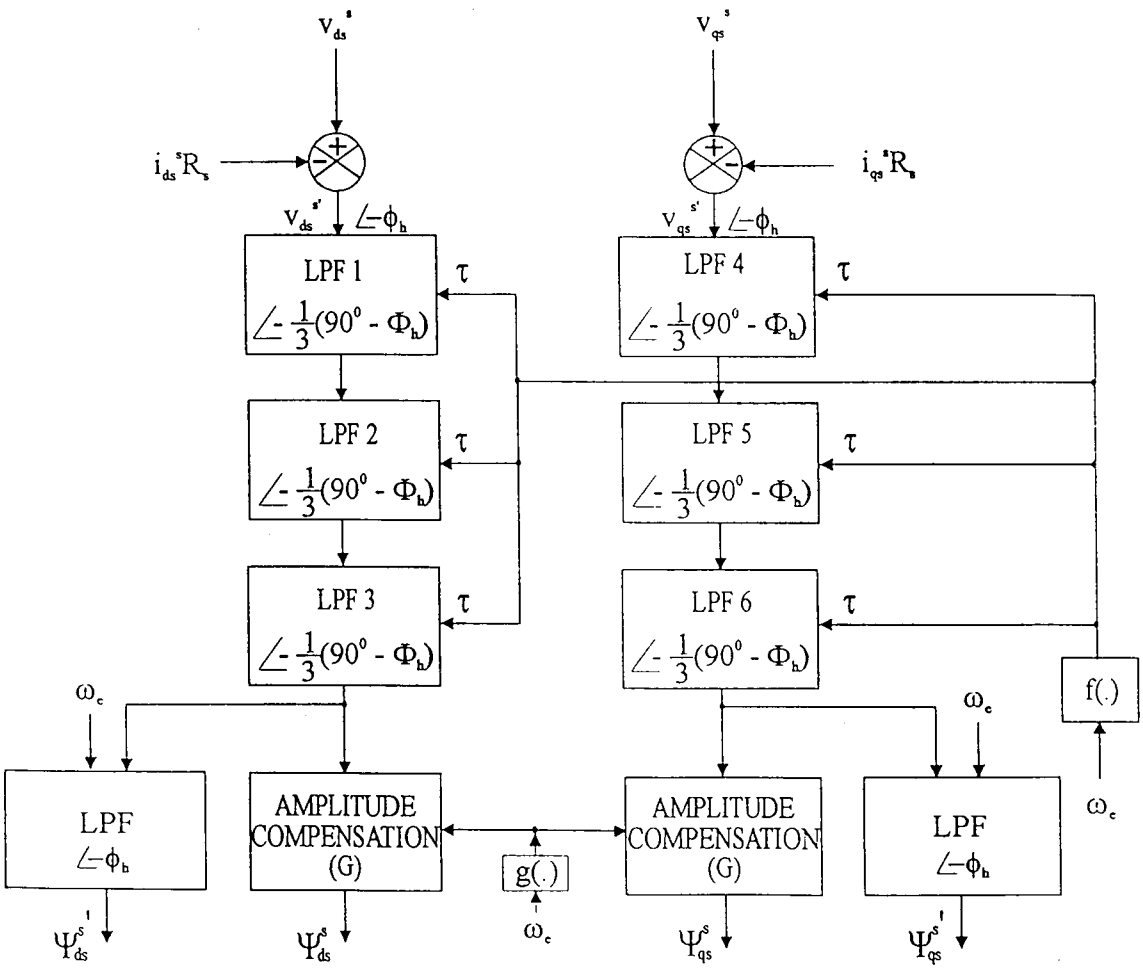


Figure 3.16 Cascaded low pass filter implementation block diagram.

3.3. Feed Back Signal Estimation

Figure 3.17 shows the control block diagram of a stator flux oriented EV induction motor drive that incorporates the cascaded low-pass filter flux estimation. The drive has torque and flux control in the outer loops. The flux loop is added with decoupling compensation current i_{dq} which is characteristic of stator flux oriented control [2]. The machine terminal voltages and currents are sensed and passed through hardware low-pass filters (not shown) before A/D conversion. The hardware filters also transform the input signals into d^s-q^s frame, as indicated. As described in previous section , q-axes flux Ψ^s_{qs}, and d-axes flux Ψ^s_{ds} computed by using cascaded low pass filter. The remaining computations for feed back signals are as follows:

$$\psi_s = \sqrt{(\psi^s_{ds})^2+(\psi^s_{qs})^2} \quad \text{.....} \quad (3.23)$$

$$\cos\theta_e = \psi^s_{ds}/\psi_s \quad \text{.....} \quad (3.24)$$

$$\sin\theta_e = \psi^s_{qs}/\psi_s \quad \text{.....} \quad (3.25)$$

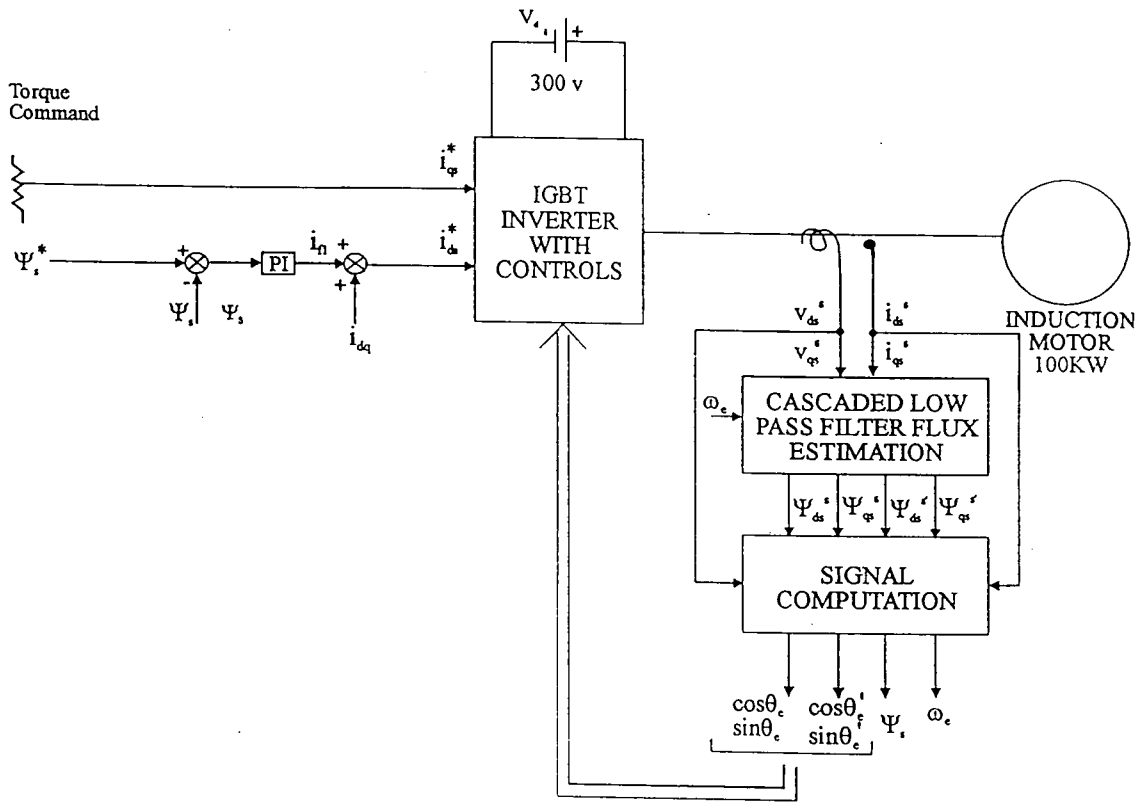


Figure 3.17 Control block diagram of stator flux oriented vector control with cascaded low pass filter based flux estimation

$$\theta_e = \sin^{-1}(\psi^s_{qs}/\psi_s) \quad \text{..... (3.26)}$$

$$\cos\theta'_e = \psi^{s'}_{ds}/\sqrt{(\psi^{s'}_{ds})^2 + (\psi^{s'}_{qs})^2} \quad \text{..... (3.27)}$$

$$\sin\theta'_e = \psi^{s'}_{qs}/\sqrt{(\psi^{s'}_{ds})^2 + (\psi^{s'}_{qs})^2} \quad \text{..... (3.28)}$$

$$\omega_e = [(v^s_{qs} - i^s_{qs}R_s)\psi^s_{ds} - (v^s_{ds} - i^s_{ds}R_s)\psi^s_{qs}]/\psi^2_s \quad \text{..... (3.29)}$$

Note that if the stator resistance variation is not adequately compensated, a small amount of error will be introduced in all the above estimated signals. The error in ω_e estimation will affect the accuracy of the cascaded filter.

$$i_{ds} = i^s_{qs}\cos\omega_e t - i^s_{ds}\sin\omega_e t \quad \text{..... (3.30)}$$

$$i_{qs} = i^{s}_{qs}\sin\omega_e t + i^{s}_{ds}\cos\omega_e t \quad \text{..... (3.31)}$$

$$\omega_{sl} = \frac{i_{qs}L_s(1+\sigma_1 s(T_r/L_s)}{T_r(\psi_s - \sigma_1 i_{ds})} \quad \text{..... (3.32)}$$

$$i_{dq} = \frac{\omega_{sl}T_r\sigma_1 i_{qs}}{(1+\sigma_1 s(T_r/L_s))L_s} \quad \text{..... (3.33)}$$

$$T_e = (3P)/4(\psi^s_{ds} i^s_{qs} - \psi^s_{qs} i^s_{ds}) \quad \text{..... (3.34)}$$

The angle of the stator flux θ_e is calculated in equation (3.26) by using an inverse sin look-up table. Inverse vector rotation transformation of the phase currents i^s_{ds} and i^s_{qs} are calculated by equation 3.30 and 3.31. The slip frequency calculation in equation 3.32 and stator flux decoupler equation 3.33 are derived using stator flux orientation and the details of the derivation can be viewed in Appendix A. For simplicity the derivative term of the equation 3.33 is not being implemented in the drive software. Feedback developed torque is being

calculated using equation 3.34.

3.4 Blaschke Equation Zero Speed Start-Up

Inspite of extensive effort, precision synthesis of speed and flux vector signals yet remains a challenge today particularly near zero speed operation. Among different types of vector control, the direct vector control with stator flux orientation [2] is very attractive because the feedback signal estimation accuracy depends only on the stator resistance variation which can be compensated somewhat easily. Besides, the signal voltage behind the stator resistance that is to be integrated for flux vector estimation is somewhat larger than that for rotor flux estimation which becomes particularly beneficial near zero speed. A new method of integration, called programmable cascaded low-pass filter as described in previous section, has been proposed in the literature [1][4] that works well with voltage model feedback signals from zero speed but with finite developed torque (i.e. slip frequency). However, the drive is not self-starting and if an attempt is made to start directly with this control, severe torque pulsation will occur (see Figure 4.1). A method of solving the problem is to start the drive with stator flux oriented indirect vector control that does not use speed sensor [1], and then transition the drive to direct vector control mode as speed begins to develop where it receives the flux vector signals estimated from the machine terminal voltages and currents This method is somewhat difficult to implement, and it introduces large torque jump during mode transitions.

This section describes zero speed start-up of the drive with stator flux oriented current model flux synthesis that does not use any speed signal. The current model (using the speed signal) flux vector synthesis was originally used by Blaschke for direct vector control servo

type drive (which can operate from zero speed). With speed sensorless Blaschke equation start-up, the drive always remains in direct vector control mode. It is somewhat easier to implement and the performance is improved compared with indirect vector control starting.

3.4.1 System Operation

Figure 3.18 shows the block diagram of the stator flux oriented direct vector control system that incorporates the Blaschke equation start-up scheme. The control uses programmable cascaded low-pass filter method of integration for feedback signal estimation, as discussed before. The drive has torque and flux control in the outer loops. The flux loop is added with decoupling compensation current i_{dq} which is characteristic of stator flux oriented control [2]. The machine terminal voltages and currents are sensed and passed through hardware low-pass filters (not shown) before A/D conversion. The hardware filters also transform the input signals into d^s - q^s frame, as indicated. The voltage signals, after subtraction of stator resistance drops, are integrated within the cascaded low pass filter estimation block. The signal computation block calculates the unit vectors, flux and frequency signals, as indicated. At start-up condition, it accepts the flux signals from the Blaschke equation start-up block (BE mode), and then transitions to the standard direct vector control mode (DVC) with a time delay td , as indicated.

3.4.2 Start-Up Scheme

As discussed before, the machine is started with the help of current model but no speed signal at zero speed and then transitioned to the standard direct vector control mode. The standard stationary frame current model equations (also known as Blaschke equations) can be given as [5]

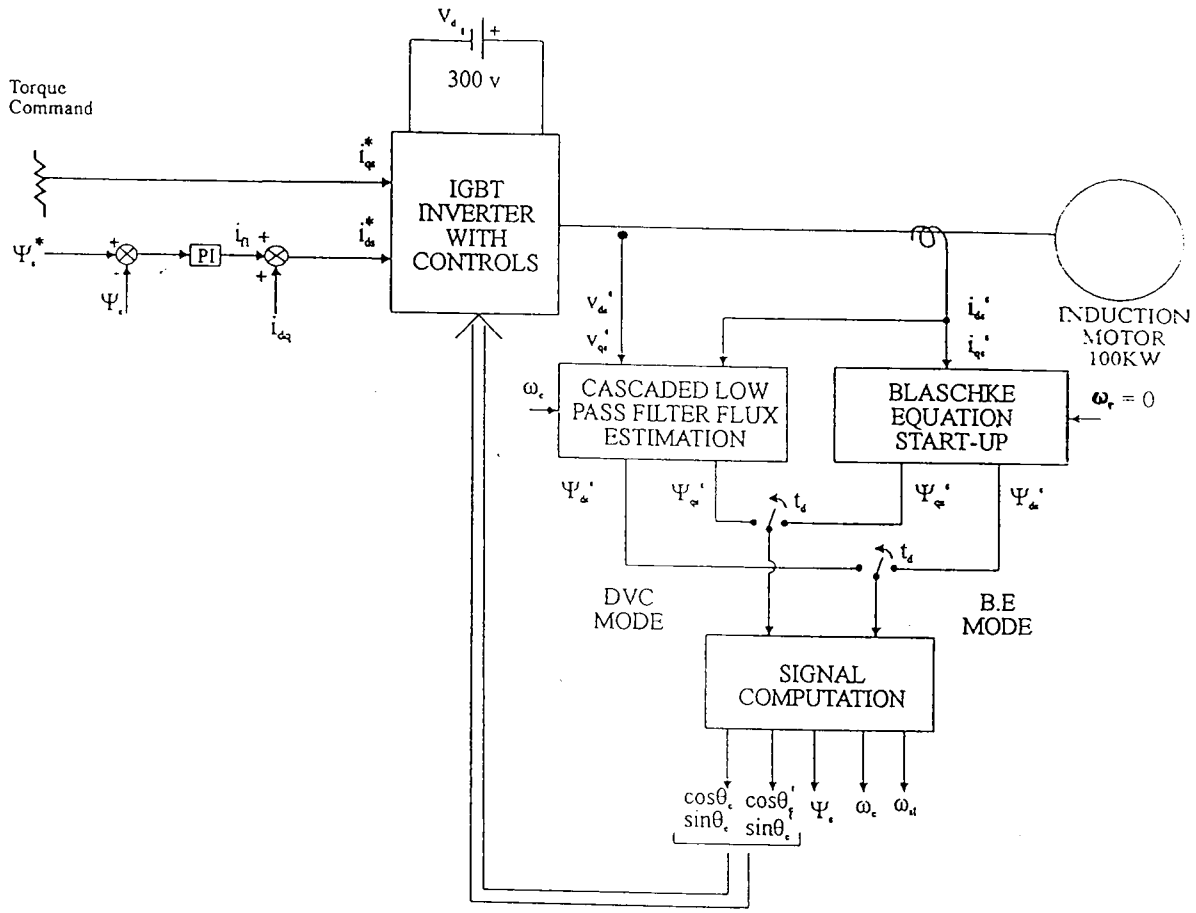


Figure 3.18 Control block diagram of stator flux oriented vector control incorporating the Blaschke equation start-up scheme

$$\frac{d\psi^s_{qr}}{dt} = \frac{L_m}{T_R} i^s_{qs} + \omega_r \psi^s_{dr} - \frac{1}{T_R} \psi^s_{qr} \quad \text{..... (3.35)}$$

$$\frac{d\psi^s_{dr}}{dt} = \frac{L_m}{T_R} i^s_{ds} - \omega_r \psi^s_{qr} - \frac{1}{T_R} \psi^s_{dr} \quad \text{..... (3.36)}$$

The equations accept stator currents (i^s_{qs} , i^s_{ds}) and speed (ω_r) at the input and estimate the rotor flux vector (ψ^s_{qr}, ψ^s_{dr}) at the output. These equations are to be transformed for stator flux vector estimation so that stator flux oriented vector control can be used at start-up.

$$\frac{d\psi^s_{qs}}{dt} = -\frac{1}{T_R} \psi^s_{qs} + A\frac{di^s_{qs}}{dt} + Bi^s_{qs} \quad \text{..... (3.37)}$$

$$\frac{d\psi^s_{ds}}{dt} = -\frac{1}{T_R} \psi^s_{ds} + A\frac{di^s_{ds}}{dt} + Bi^s_{ds} \quad \text{..... (3.38)}$$

where $A = L_{ls} + L_m L_r / L_r$ and $B = L_s R_r / L_r$. The equations (3.37) and (3.38) give current model or Blaschke equations for stator flux vector estimation. These equations will be used without the speed signal at zero speed to start the machine. The detail derivation of equation (3.37) and (3.38) is shown in Appendix B. A block diagram to solve stator fluxes Ψ^s_{qs}, and Ψ^s_{ds} is shown in Figure 3.19. Note that this block diagram is valid to start the machine with stator flux oriented vector control strictly at zero speed.

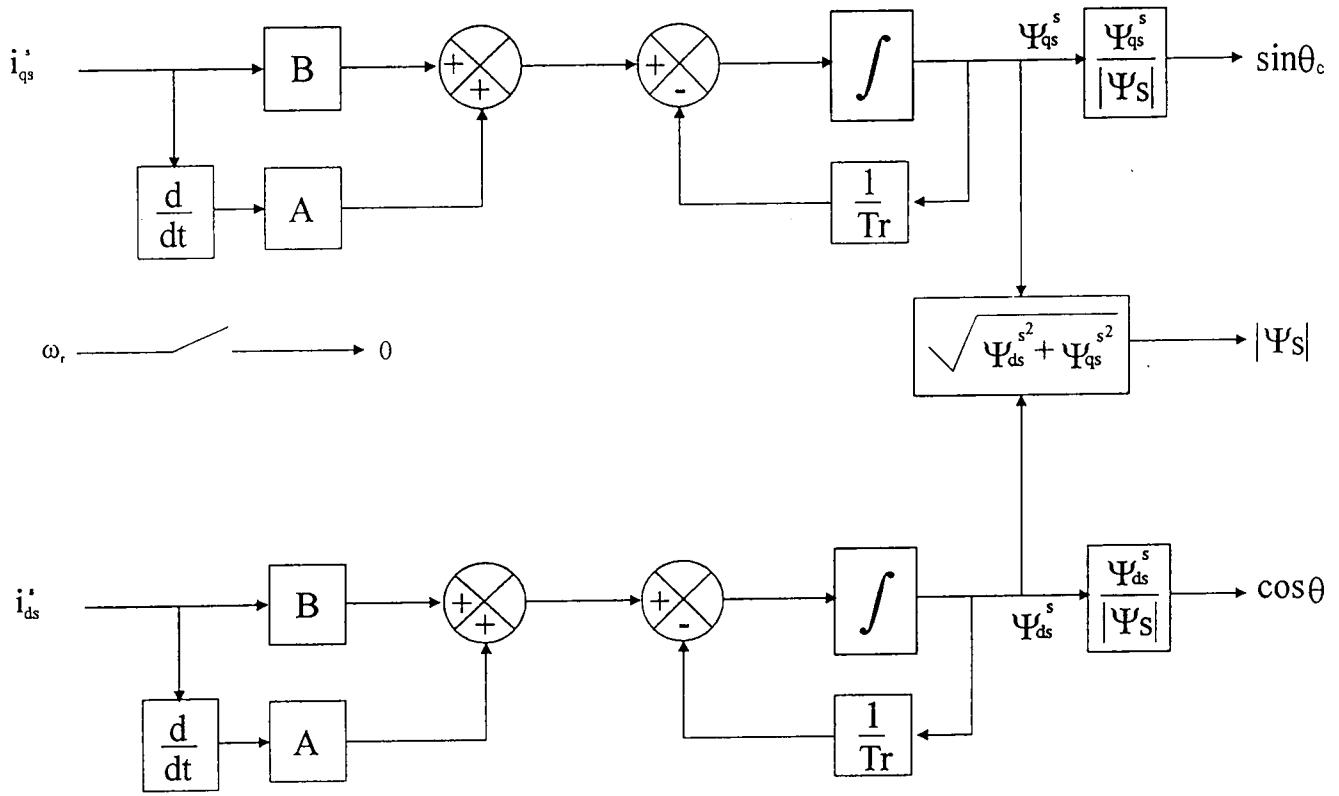


Figure 3.19 Block diagram of Blaschke equation start-up scheme

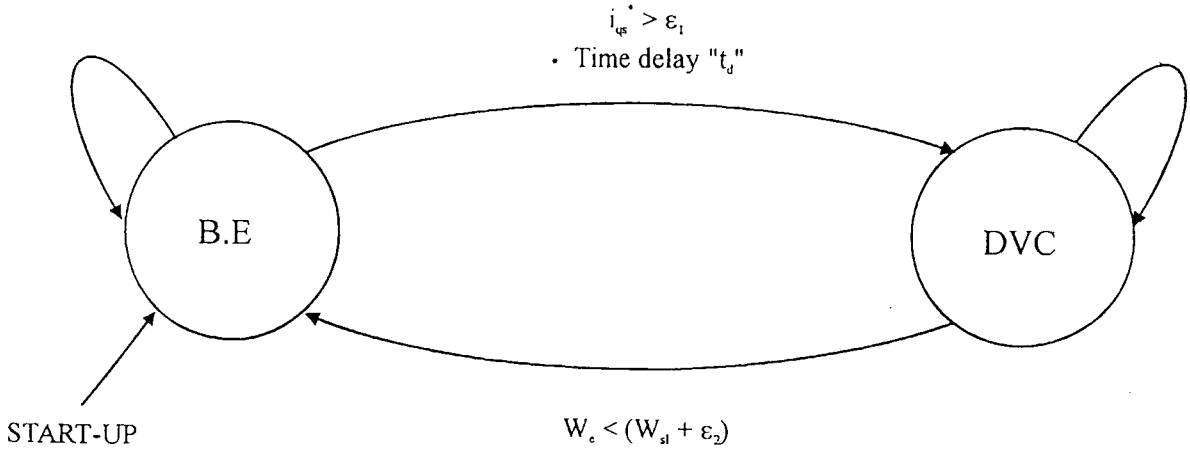


Figure 3.20 Transition between Blaschke equation start-up and DVC control mode

Figure 3.20 shows the transition between the start-up (B.E.) mode and the regular vector control (DVC) mode. As the command i_{qs}^* exceeds a small threshold ϵ_1, the drive transitions to DVC mode with a time delay t_d. This delay is programmed to be inversely proportional to the magnitude of i_{qs}^*. As the speed slows down to zero in regular DVC mode with finite developed torque (i.e. slip frequency), the drive transitions to the B. E. mode. Note that in B. E. mode, strongly parameter sensitive flux vector estimation will tend to give some mismatch with that of DVC mode and may cause some jerk at transition. This has to be fine tuned by on-line parameter correction.

3.5 Quasi-Fuzzy Estimation of Stator Resistance of Induction Motor

This section describes a quasi-fuzzy method of on-line stator resistance estimation of an induction motor where the resistance value is derived from stator winding temperature estimation as a function of stator current and frequency through a nonlinear dynamical thermal model of the machine. The estimator has been designed and iterated by simulation study, and then implemented by digital signal processor on a 5 hp stator flux oriented direct vector-controlled drive. The experimental performance of the estimator has been verified extensively both at static and dynamic conditions by a stator-mounted thermistor network based estimation, and gives excellent performance.

3.5.1 Introduction

Currently, sensorless vector control of induction motor drives is receiving wide attention in the literature. The flux and speed estimation of both the rotor and stator flux-oriented vector- controlled drives that are based on voltage model computation introduces error due to stator resistance variation. This error becomes serious at low speed (i.e. frequency) when the stator resistance drop tends to be comparable with the machine counter EMF. The same problem arises in stator flux estimation of direct self control (DSC) method of induction motor drive. Besides the error in magnitude, the phase angle error in the flux vector introduces coupling effect in direct vector control that affects response of the drive. Among all the vector control methods, the stator flux oriented direct vector control is recently gaining more popularity because the feedback signal estimation accuracy is dependent only on the stator resistance variation which can be compensated somewhat easily. Of course, the other advantage is that the signal voltage behind the stator resistance drop that is to be

integrated for stator flux estimation is somewhat larger than that for rotor flux estimation.

Neglecting the small amount of skin and stray loss effects, the stator winding resistance primarily varies with winding temperature which is given by the following simple equation:

$$R_s = R_{so} + \alpha R_{so} (T_s - 25\ ^\circ C) \quad \ldots\ldots\ldots\ldots \quad (3.39)$$

where R_s = resistance at T_s °C

R_{so} = name plate resistance (at 25 °C)

T_s = stator winding temperature (°C)

α = temperature coefficient of resistance of copper (3.928 X 10 -3 / °C)

If temperature sensing thermistors [1] [7] are inserted at several points in the stator winding, the average stator temperature can be monitored, and correspondingly, stator resistance can be estimated fairly accurately. However, introducing such temperature sensors in a "sensorless" drive will not be acceptable.

The question is: Is it possible to estimate the stator winding temperature with reasonable accuracy without introducing any additional sensors other than the minimum number of those required by the drive? Basically, the losses in the machine contribute to stator temperature rise, and these losses can be classified as stator copper loss, rotor copper loss, stator iron loss, rotor iron loss, and some amount of stray loss [8]. The heat generated by the losses flow through distributed parameter thermal equivalent circuit of the motor and cause temperature rise at different points. The dynamic thermal model of the machine is nonlinear, multi-dimensional, and is very complex. It is influenced by the cooling method used in the machine. The stator copper and iron losses dominantly contribute to stator winding temperature rise,

although the coupling effect of rotor losses will also contribute to this temperature rise. Fortunately, the iron loss is small in the rotor, and the rotor copper loss depends on the rotor current which is related to the stator current. In the past, attempt was made to estimate the stator resistance by simulation relating to stator current only (using fuzzy logic or neural network) [9][10] without considering stator iron loss and dynamic thermal model of the machine, but lack of experimental work never divulged the gross error of such estimation.

Stator resistance is derived from fuzzy logic based estimation of stator winding temperature which is defined as a function of stator current and frequency through an approximate dynamic thermal model of the machine. Both the fuzzy estimation algorithm and the dynamic thermal model were derived from the experimental data of a 5 hp thermistor mounted induction motor.

3.5.2. Description of the Estimator

Figure 3.21 shows the complete estimation block diagram of stator resistance which also includes a thermistor network (dotted figure) for calibration of stator temperature (T_s). The same thermistor network was also used to generate experimental data for formulation of the estimator which will be discussed later. In the present project, the estimated stator resistance provides compensation for correct estimation of stator flux in a stator flux oriented direct vector-controlled induction motor drive [4]. The fuzzy estimator, as shown, estimates the steady state stator temperature rise ($\Delta T_{ss} = T_{ss} - T_A$), where ΔT_{ss} = steady state temperature rise above ambient, T_{ss} = steady state stator temperature and T_A = ambient temperature. The ΔT_{ss} is defined as a fuzzy function of rms stator current (I_s) and frequency (ω_e) through the fuzzy estimator. Both the I_s and ω_e signals in a vector-controlled drive can

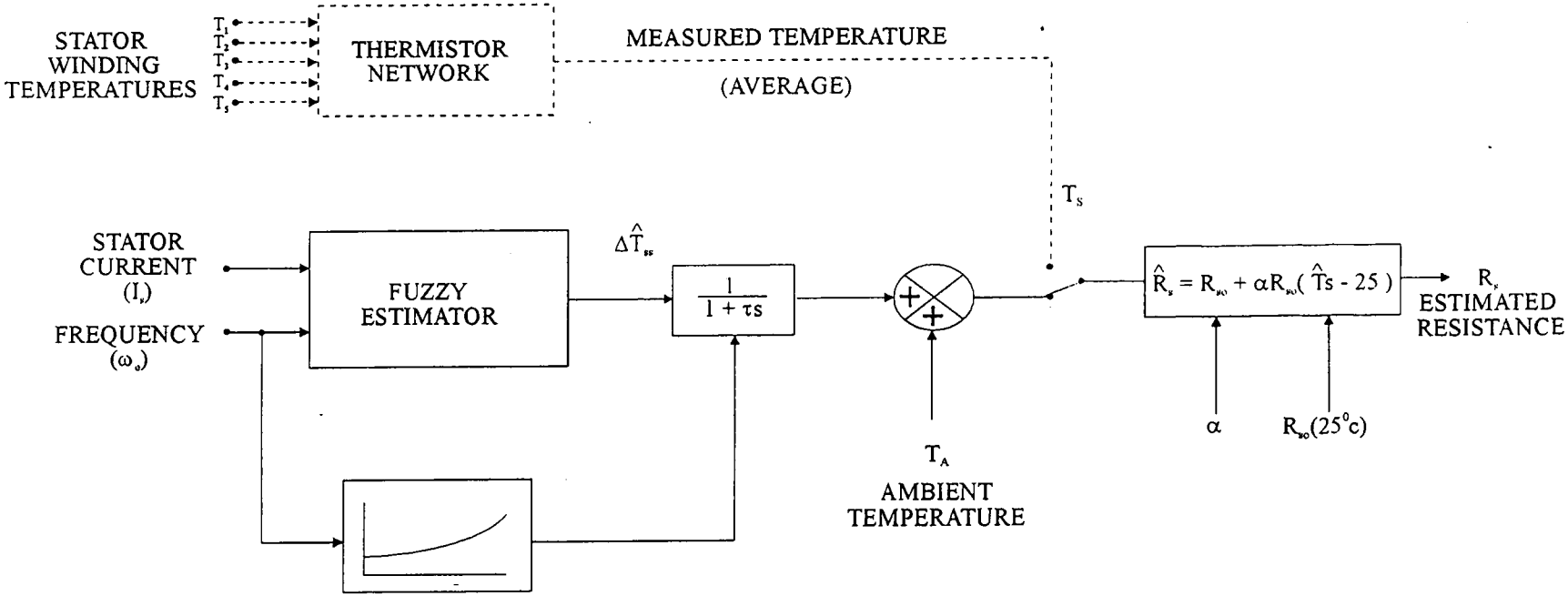


Figure 3.21 Quasi-fuzzy estimation block diagram of stator resistance "R_s"

be obtained from the following relations:

$$I_s = \frac{1}{\sqrt{2}}\sqrt{i_{ds}^{s^2} + i_{qs}^{s^2}} \quad \text{..... (3.40)}$$

$$\omega_e = \frac{[(\upsilon_{qs}^s - i_{qs}^s R_s)\psi_{ds}^s - (\upsilon_{ds}^s - i_{ds}^s R_s)\psi_{qs}^s]}{\psi_s^2} \quad \text{..... (3.41)}$$

where all the variables are defined by standard symbols. Since ω_e is close to ω or speed signal, it can also be obtained from the speed encoder, if used. The eq. (3.41) requires voltage signals which are also used for flux estimation in the drive. Note that the copper loss is approximately represented by the I_s signal whereas the core loss at the rated flux is represented by the ω_e signal. The fuzzy estimator essentially relates to stator current (i.e., copper loss) and frequency (i.e., iron loss) nonlinearly and computes the temperature rise through the thermal resistance that is embedded in the estimator. The advantage of fuzzy estimation is that it does not require any mathematical model which is extremely complex [11] [12]. In fact, the true power of fuzzy logic is exemplified in this application. In the present machine, cooling or heat transfer to the ambient occurs by two mechanisms: (1) natural convectional cooling which will contribute to dynamic thermal impedance, and (2) cooling by shaft-mounted fan which varies proportional to speed (i.e., frequency) that will contribute to nonlinear thermal resistance of the machine. The dynamic portion of the thermal model can be approximated by a first order low-pass filter [13] $1/(1 + \tau s)$, as indicated in the Figure, where τ is a nonlinear function of frequency. Once the dynamic stator temperature rise ΔT_{ss}

is estimated at the filter output, the stator resistance (R_s) computation becomes straightforward, as indicated in the Figure 3.21.

Figure 3.22 shows the experimentally determined stator temperature rise curves as function of stator current and frequency at steady state condition. The data for these curves were obtained from measured average temperature rise of a five-position thermistor network for different stator current and frequency conditions in the drive. Note that at higher frequency (i.e., speed), the iron loss increases tending to higher temperature rise, but the dominant cooling effect of the shaft-mounted fan essentially decreases the temperature. These curves are used to formulate the fuzzy membership functions shown in Figure 3.23 and the corresponding rule matrix in Table 3.1.

Basically, the fuzzy estimator interpolates the ΔT_{ss} information embedded in Figure 3.22. Note the asymmetric nature of membership functions that are contributed by asymmetric spacing of the curves in Figure 3.22. Once the fuzzy membership functions and the rule table, as discussed before, were derived, these were simulated in computer and iterated extensively until the calculated temperature rise data matched with the experimental curves given in Figure 3.22. Using "C" language, the quasi-fuzzy estimator was simulated for given fuzzy membership function and the rule table (Appendix E). MATLAB was used to draw the simulation study curves. Figure 3.24 gives the experimentally determined approximate thermal time constant curve with frequency where the improved cooling at higher speed increases the effective thermal constant simulation results are discussed in chapter 4 and experimental results are discussed in chapter 6.

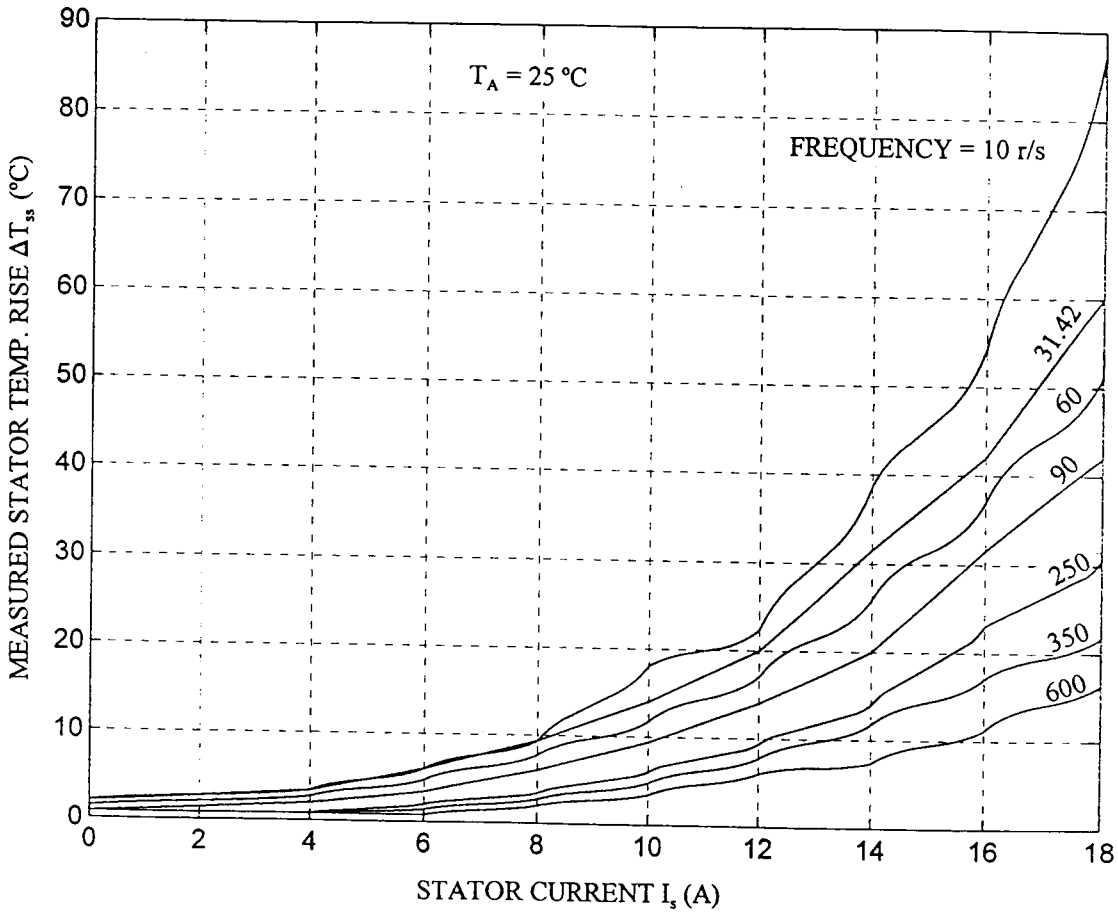


Figure 3.22 Measured stator temperature rise vs. Stator current at different frequency (steady state)

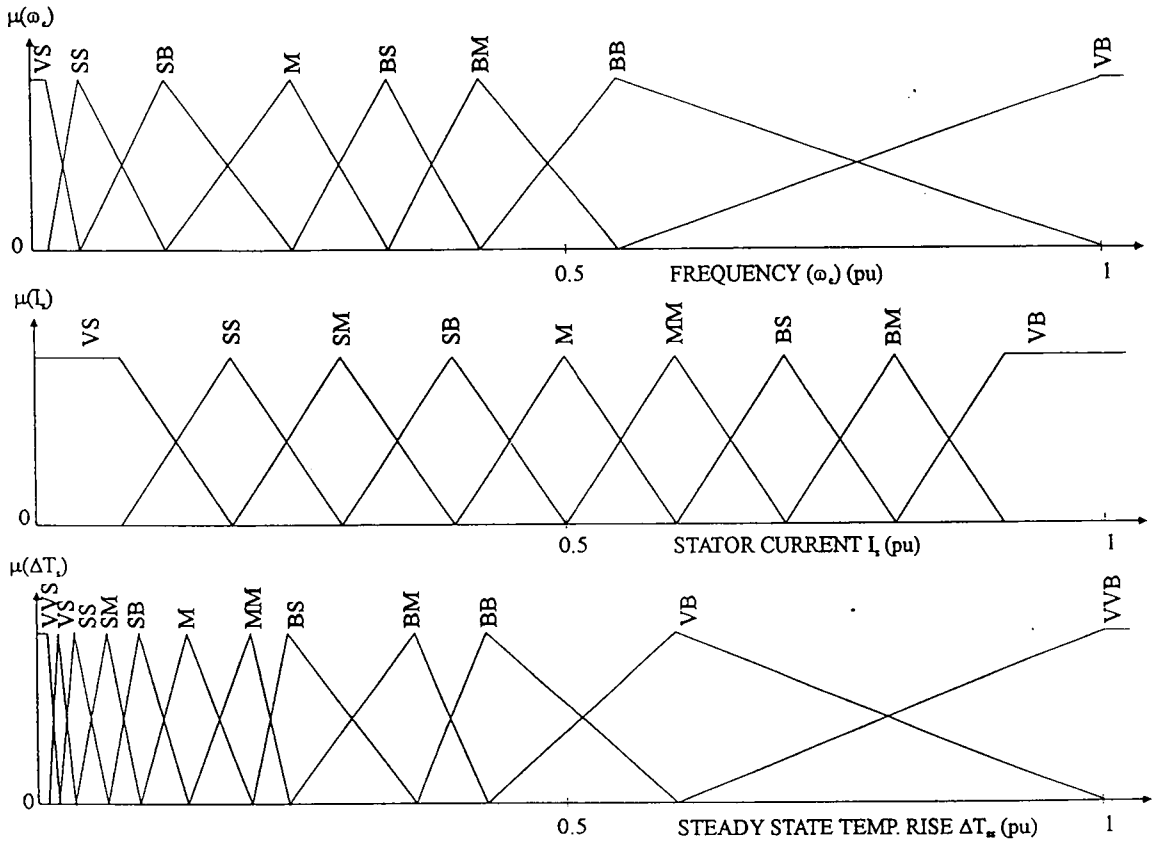


Figure 3.23 Fuzzy estimation membership function

(a) stator current (1 pu = 12.7 A(rms))

(b) frequency ω_e(1 pu = 733 r/s)

(c) steady state temperature rise ΔT_{ss} (1 pu = 100°c)

Table 3.1 Fuzzy rule base for estimation of steady state temperature rise

$\omega_{e\ (pu)}$ / $I_{s\ (pu)}$	VS	SS	SB	M	BS	BM	BB	VB
VS	VS	VS	VVS	VVS	VVS	VVS	VVS	VVS
SS	SS	SS	VS	VS	VVS	VVS	VVS	VVS
SM	SM	SM	SS	SS	VS	VS	VVS	VVS
SB	SB	SB	SM	SM	SS	SS	VS	VS
M	MM	M	SB	SB	SM	SM	SS	SS
MM	BS	MM	M	M	SB	SB	SM	SM
BS	BB	BM	MM	MM	M	M	SB	SM
BM	VB	BB	BM	BS	BS	MM	M	SB
VB	VVB	VB	BB	BM	BM	BS	MM	M

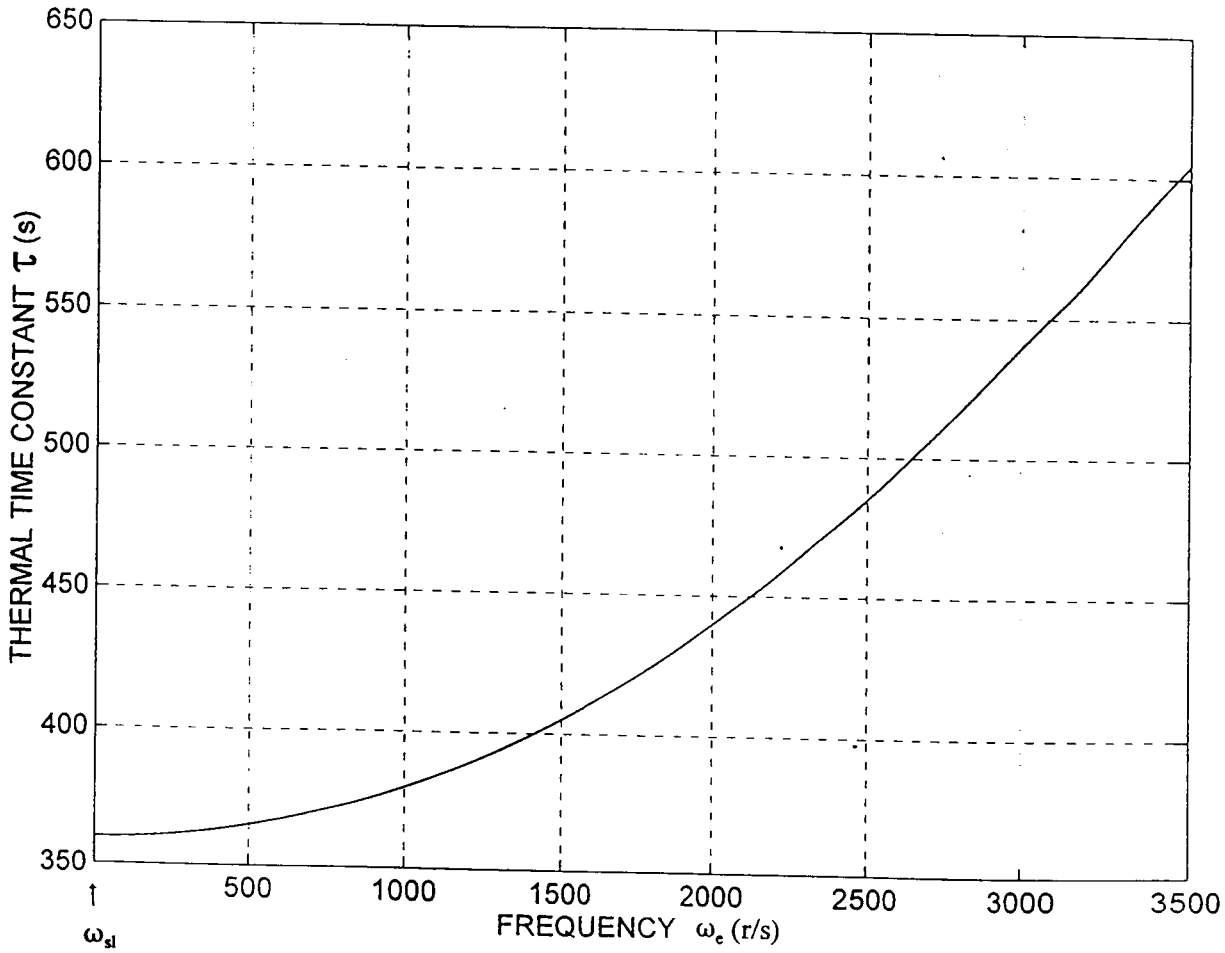


Figure 3.24 Experimental machine thermal time constant curve with frequency

3.6 Neuro-Fuzzy Based Light Load Efficiency Optimization

3.6.1 Introduction

Efficiency improvement in the variable frequency drives is getting a lot of attention in the recent years. Higher efficiency is important not only from the viewpoints of energy savings and its obvious financial payoff, but also from the broad prespective of environmental pollution control. In fact, as the use of variable speed drives continues to increase in areas traditionally dominated by constant speed drives, motivated by the quest for increased productivity, the financial and environmental payoffs reach new importance.

The efficiency of a drive system is a complex function of type of machine used, converter topology, type of power semiconductor switches and the selected PWM algorithm. In addition, the control system has profound effect on the drive efficiency. A drive system normally operating at rated flux gives the best transient response. However, at light loads, rated flux causes excessive core loss, compared to copper loss, thus impairing the efficiency of the drive [14]. Since the drive operate at light load most of the time, optimum efficiency can be obtained by flux control.

A number of methods for efficiency improvement through flux control has being proposed in the literature, since its principle was first introduced by Nola. They can be classified into three basic types:

1. The simple precomputed flux program as a function of torque, that is widely used for light load efficiency improvement. The scheme can be improved by generating the flux program at discrete speeds, to take the frequency dependency into consideration. This method, however, yields only a partial improvement in the system efficiency.

2. The real time computation of losses and corresponding selection of flux level that results in minimum losses is quite elegant. However, as the loss computation is based on a machine model, parameter variation caused by temperature and saturation effects tends to yield sub-optimal efficiency operation.

3. The online efficiency optimization control on the basis of search, where flux is reduced in steps until the measured input power settles down to the lowest value. The control does not require the knowledge of machine parameters, is completely insensitive to the parameter changes, and the algorithm is applicable universally to any arbitrary machine.

In this section, a neuro-fuzzy on-line efficiency optimization control is proposed for a direct vector controlled drive system. Fast convergence is achieved by using adaptive step size of the stator flux. The low frequency pulsating torque generated by the efficiency controller has been suppressed by closed loop torque control.

3.6.2 Neuro-Fuzzy Efficiency Optimization of a Vector Control Drive

The direct vector controlled induction motor drive incorporating the proposed efficiency optimization controller [15] is shown in Figure 3.25. As shown in the Figure 3.25, the power circuit of the EV drive consists of a 300v battery, IGBT bridge inverter and a 100 KW (peak power) induction motor. It is a sensorless stator flux oriented direct vector controlled system where developed torque is being controlled in the outer loop. The torque loop generates command for the i_{qs} (torque component of the current) loop. The stator flux Ψ_s^* is constructed by subtracting the neuro-fuzzy controller output from the rated flux Ψ_{sr}^*.

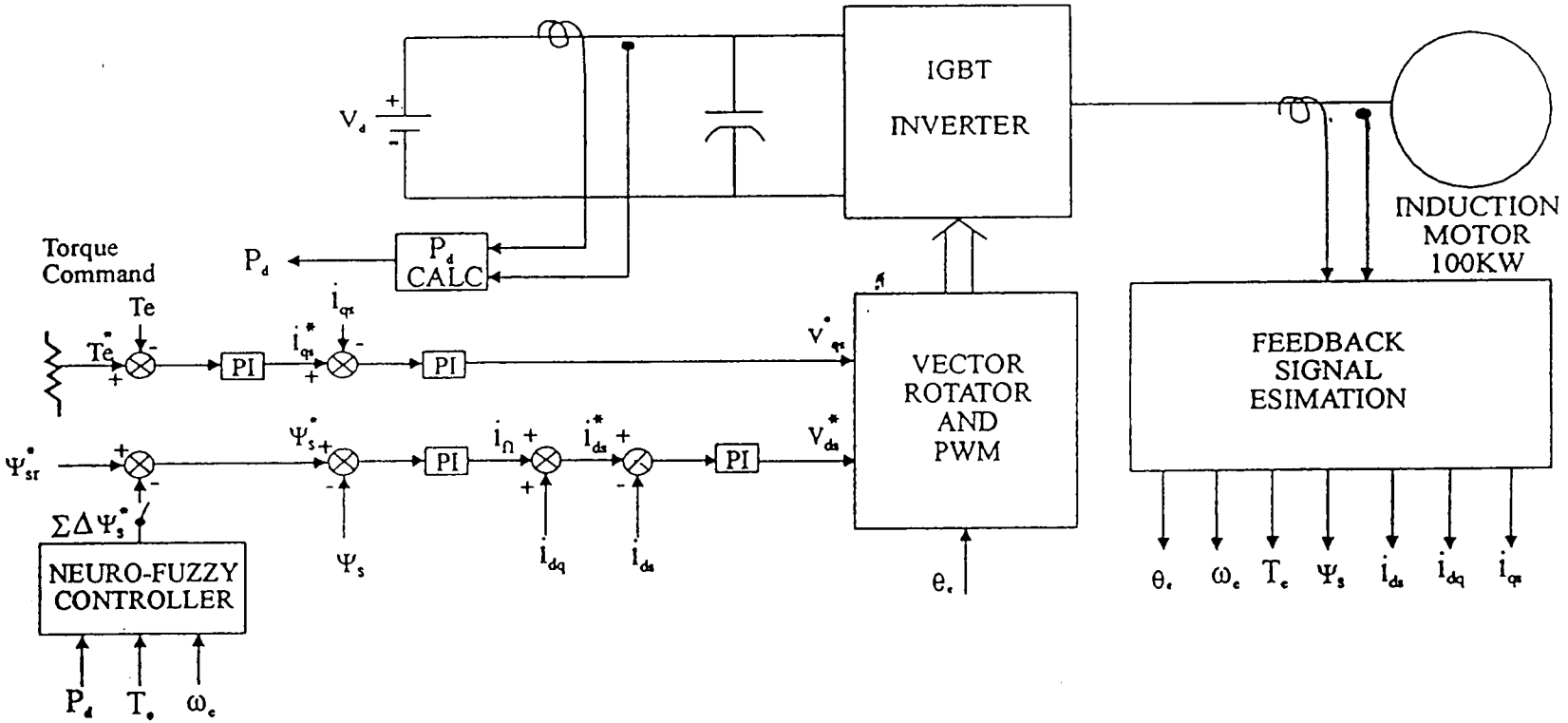


Figure 3.25 Control system block diagram incorporating the neuro-fuzzy based efficiency optimization controller.

The flux loop then generates the i_{ds} (flux component of the current) command after adding the compensation current i_{dq}, as shown. Both torque and flux loops then generates the voltage commands which are then rotated and PWM-controlled to drive the inverter.

The neuro-fuzzy controller becomes effective at steady state condition, i.e., when the machine speed is constant. Note that the dc link power P_d, instead of the input power, has been considered for the controller since both follow symmetrical profiles. This can be demonstrated as follows:

* Minimization of P_d translates into minimization of dc link current i_d, since the dc link voltage v_d is essentially constant,

* Rectifier losses are proportional to i_d,

* Minimum i_d yields minimum rectifier loss, and consequently, minimum input power.

The principle of the efficiency optimization control with stator flux programming at a steady state torque and speed condition is explained in Figure 3.26.

The decrease of the stator flux results in decrease of magnetizing current and ultimately increase of the torque current. As the flux is decreased, the iron loss decreases with the attenant increase of the copper loss. However, the total system loss (converter and machine) decreases, resulting in a decrease of dc link power. The search is continued until the system settles down at the minimum input power point A, as indicated. Any excursion beyond point A will force the controller to return to the minimum power point. This has been done using fuzzy logic before, in this project it is extended to a stator flux oriented direct vector controlled EV induction motor drive of 100 KW power. The fuzzy controller input-output transfer characteristics are then used to train a feed forward neural network with

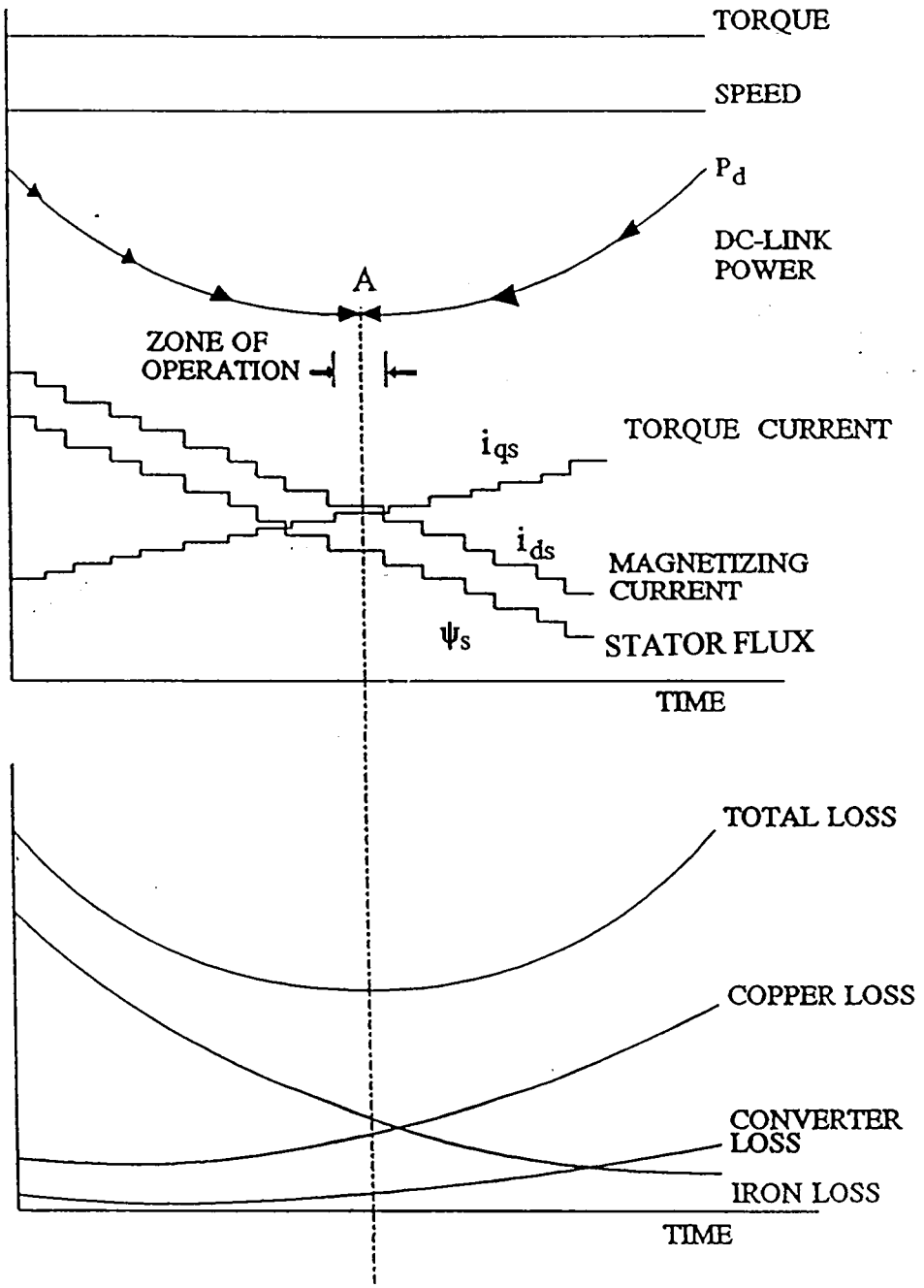


Figure 3.26 Principle of efficiency optimization control with stator flux orientation

delayed feedback which then replaces the fuzzy controller in the drive system. The basic implementation of fuzzy logic controller by a static feed forward neural network is described in [16] [17]. In such an implementation, all the advantages of the fuzzy control are retained. Besides, the neural network adds the advantage of fast computation either by a dedicated hardware chip or by DSP based software routine. In a complex control system, relief of such a heavy computational burden to the DSP appears be very attractive. Extensive simulation study indicates excellent performance of the neuro-fuzzy control.

3.6.3 Neuro-Fuzzy Control

Figure 3.27 gives the detailed functional diagram of the neuro-fuzzy controller indicated in Figure 3.25. Note that this control becomes effective only at steady state condition which can be detected by the torque loop error and the frequency signals (not shown). In the beginning, a complete fuzzy controller characterized by membership functions of ΔP_d (pu), $L\Delta\Psi_s$(pu) and $\Delta\psi_s^*$(pu) , and the corresponding rule table was developed as shown in Figure 3.28 and Table 3.2. Min-Max method was used to fuzzyfy the input signals and hieght method was used to defuzzyfy the output signal of the fuzzy controler. The operation principle of Figure 3.27 can be described as follows: At a certain steady state speed and load torque conditions, the input dc power (P_d) is sampled and compared with the previous value to determine the decrement ΔP_d. In addition, the last stator flux decrement ($L\Delta\ \Psi_s$(pu)) is reviewed. On these bases, the flux decrement step $\Delta\Psi^*_s$(pu) is generated from the fuzzy membership functions and rule table through fuzzy inference and defuzzification. The adjustable gains S_p and $S_{\Psi s}$, generated by the scale factor computation block, convert the input variable and control variable, respectively to and from per unit values, as indicated.

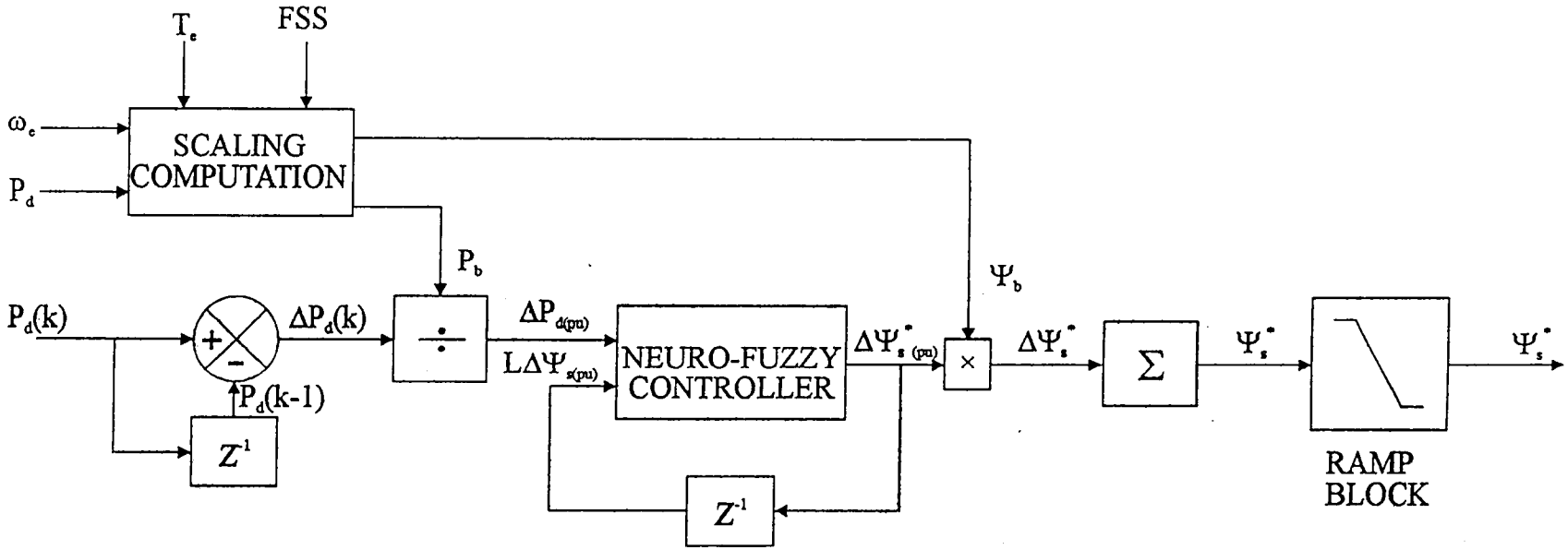


Figure 3.27 Details of the neuro-fuzzy controller

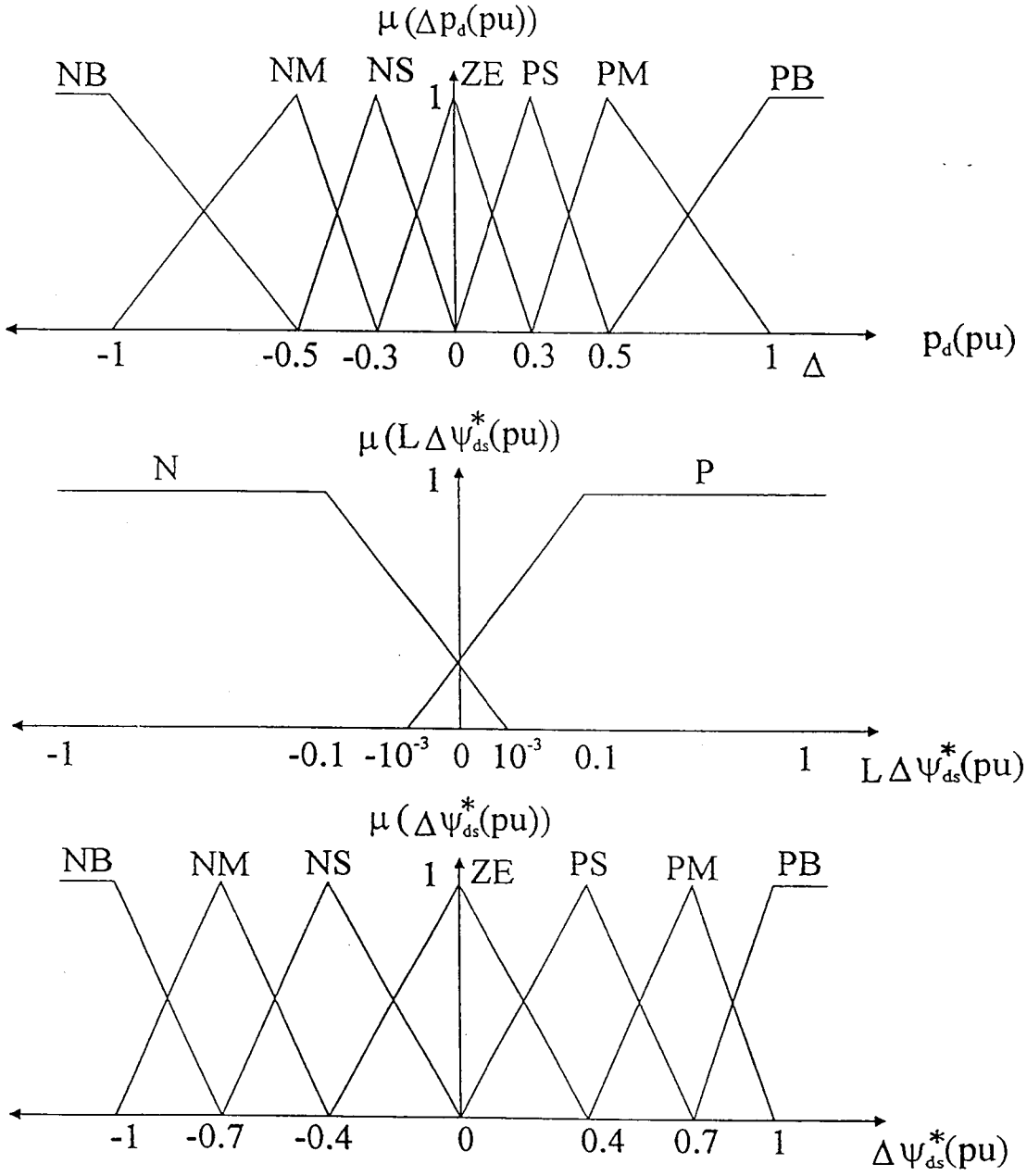


Figure 3.28 Membership functions for fuzzy efficiency control

(a) Change in dc link power (P_d(pu))

(b) Last change in stator flux ($L\Delta\psi^*_s$ (pu))

(c) stator flux increment ($\Delta\psi_s$ (pu))

Table 3.2 Rule base for fuzzy efficiency controller

$L\Delta i_{ds}(pu)$ / $\Delta P_d(pu)$	N	P
PB	PM	NM
PM	PS	NS
PS	PS	NS
ZE	ZE	ZE
NS	NS	PS
NM	NM	PM
NB	NB	PB

The scale factors are given by the equations:

$$S_p = A_1 * \frac{\omega_e}{\omega_e(rated)} + A_2 \quad \text{..... (3.42)}$$

$$S_{\psi s} = C_1 * \frac{\omega_e}{\omega_e(rated)} + C_2 * \frac{T_e}{T_e(rated)} + C_3 \quad \text{..... (3.43)}$$

where the symbols and constants are self-explanatory. The appropriate coefficients A1, A2, C1, C2 and C3 are derived from simulation studies as follows:

* Using simplified models, the approximated optimum stator flux Ψ_s is determined for selected operating points in the torque-speed plane;

* Least-square estimation is used to obtain a linear representation of stator flux as a function of machine torque T_e and speed ω_e;

* The output gain expression (3.43) is defined as one third of the distance between rated flux and approximated optimum flux;

* A new set of simulations is performed, for a load torque of 0.1 pu and different speeds. The stator flux is reduced by a step size $S_{\psi s}$ given by expression (3.43), and the corresponding reduction in input power Δp_d is obtained.

* The input power expression (3.42) is defined as the linear representation of Δp_d^{-1} as a function of speed, derived by least-square estimation.

A few word on the importance of the input and output gains are appropriate here. In the absence of the input and output gains, the efficiency optimization controller would react equally to a specific value of Δp_d, resulting from the past action $L\Delta \Psi_s$(pu), irrespective of

operating speed. Since the optimal efficiency point A. Figure 3.26 is speed dependant, the control action could easily be either too conservative, resulting in slow convergence, or excessive, yielding an overshoot in the search process, with possible adverse impact on the system stability. As both input and output gains are functions of the speed, this problem does not arise. Equation 3.43 also incorporates the "a priori" knowledge that the optimum value of Ψ_s is a function of torque as well as machine speed. In this way, for different speed and torque conditions, the same $\Delta\Psi_s$(pu) will result in different $\Delta\ \Psi_s$, ensuring a fast convergence. One advantage of per unit basis operation is that the same fuzzy controller can be applied to any arbitrary machine, by simply changing the coefficients of the input and output gains.

The $\Delta\psi_s$(pu) decrementation stops at the minimum P_d (or by the constraint of stator current limit) since any additional $\Delta\psi_s$(pu) in the same direction will reverse the polarity of ΔP_d(pu). In fact, the on-line search mechanism makes the operation oscillating about the minimum P_d point. The output flux decrements ($\Delta\Psi_s^*$) are summed and ramped before coupling to the system in Figure 3.25. The ramping of flux along with a high gain torque loop heavily attenuates the pulsating torque. If any transient condition of the drive is detected, the fuzzy control is abandoned and the rated flux is established to get the optimal transient response. Note that at the minimum P_d point with the stator current limit condition, the drive can not withstand any sudden load torque jump because of the sluggishness of the flux loop response. For the same reason, the increase of speed response is somewhat slowed down. However, these limitations do not affect EV type drives.

Once the fuzzy controller was developed, it was simulated with the complete drive system and iteratively tuned until the best performance was obtained. A dynamical feed forward

neural network was then trained to emulate the fuzzy controller. Figure 3.29 shows the feed forward neural network topology which uses one hidden layer, and the respective number of neurons at the input, hidden and output layer are two, ten and one. Basically, it represents an input-output nonlinear pattern matching network where the nonlinearity is introduced by hyperbolic-tan type transfer function (ϕ) at the hidden and output layer neurons. The training data were generated by simulating the fuzzy controller alone with the corresponding input-output signals. NeuralWare Professional II+ software based on backpropagation training algorithm was used to train the network. After extensive training, the fuzzy controller was replaced by the neural network controller in the drive system simulation and its performance was then evaluated. Simulation results are discussed in chapter 4.

Such a neuro-fuzzy control combines the advantages of fuzzy and neural controls. The control attains fast convergence with inherent adaptive step size signals of fuzzy control. The neural network implementation permits fast computation and can be implemented by a dedicated hardware chip or by DSP based software. Extensive simulation study verifies excellent performance of the controller.

3.6.4 Transition to Optimum Transient Response Mode

It may be noted from the preceding discussion that efficiency optimization control is only effective at steady state condition. A disadvantage of this control mode is that the transient response becomes sluggish, because the weaker flux reduces the maximum torque capability. For any change in load torque or speed command, fast transient response capability of the drive can be restored by establishing the rated flux. Figure 3.30 shows the criteria for transition between efficiency optimization (Mode 2) and transient response

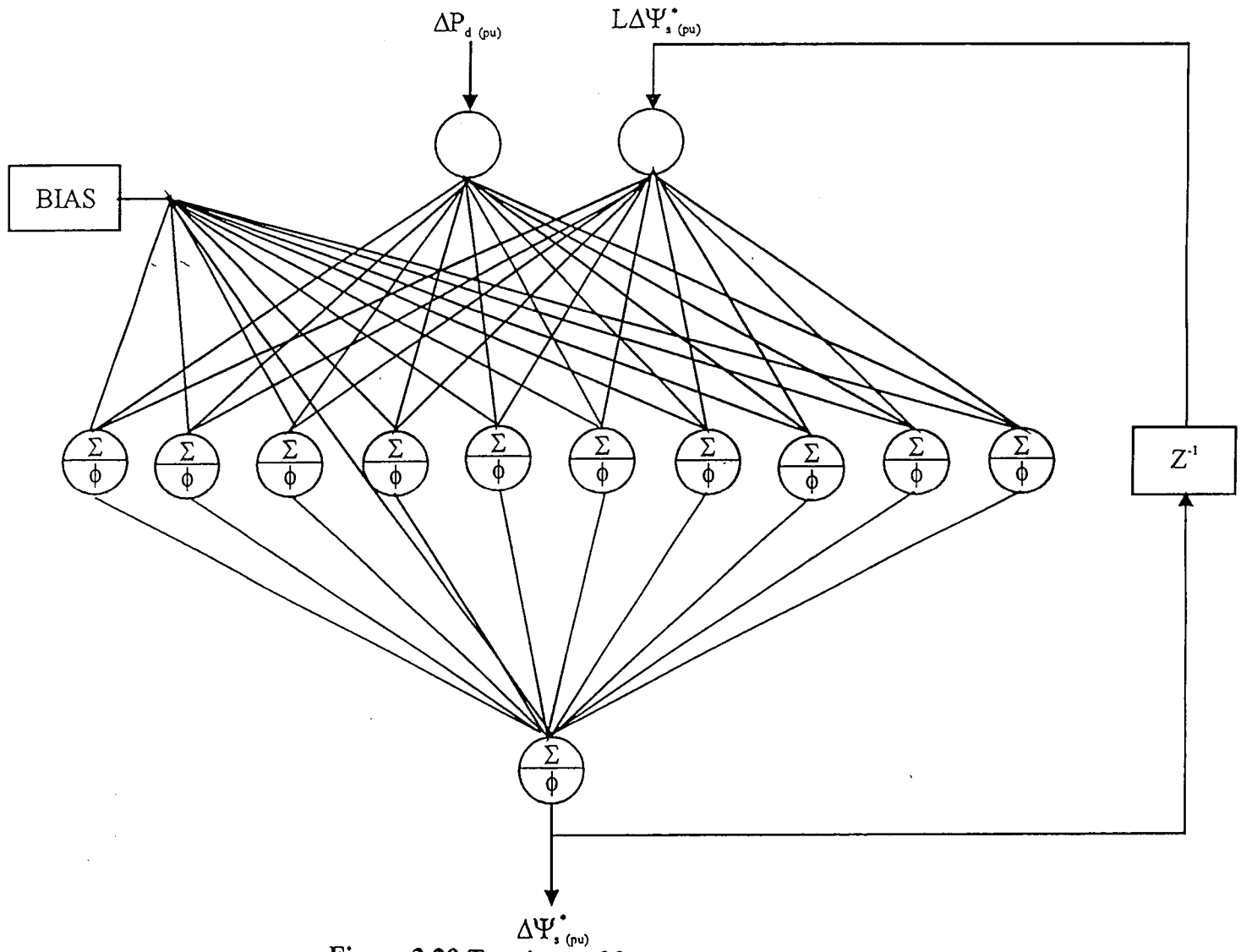


Figure 3.29 Topology of feed forward neural network with delayed feedback

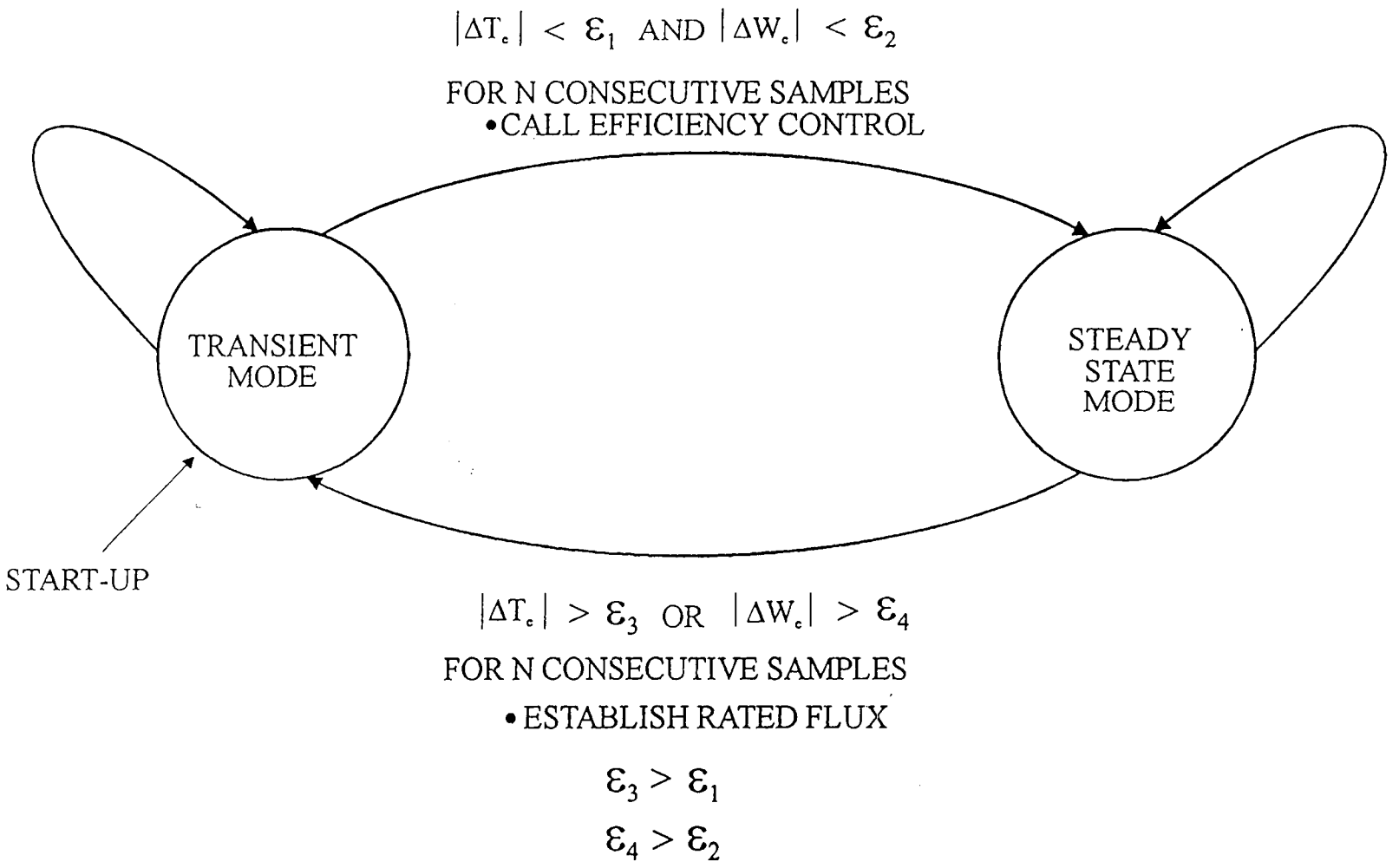


Figure 3.30 Transition between efficiency optimization and transient response

optimization (Mode 1). The system starts in Mode 1 and then switches to Mode 2 when the steady state criteria is met. In the event of a load distrubance or a change in set speed, the system switches back to Mode 1 by establishing the rated magnetizing current.

At steady state, current must be kept within the thermal limits of the machine and inverter, that is usually the rated machine current. During a transition from Mode 2 to Mode 1, this limit can be temporarily increased, (typically by 50% for 60 secs.) such that more torque is available to meet the acceleration/deceleration demand. This constraint is given by the relation

$$\sqrt{i_{ds}^{*2} + i_{qs}^{*2}} < I_{\text{lim}}$$

optimization modes

Where $I_{lim} = 1.5\ I_{sr}$, and I_{ar} is the rated stator current.

Notice that, as the stator flux does not change instantly, the maximum torque capability following a transition might not be enough to overcome the torque demand, even with the relaxed current limit. In such scenario, there is a possibility that the drive will stall, and preventive measures should be taken. One simple solution consists in imposing a minimum value for Ψ_s^*, during Mode 2, that would keep the maximum torque capability at a safe level, at the expenses of suboptimum efficiency operation.

3.7 Field-Weakening Mode

In the field-weakening region the machine EMF approaches the voltage limit of the inverter, leaving less margin for the current regulator [29]. Therefore, it is desirable to use a control scheme which yields the maximum torque/ampere ratio and constant EMF for various load torques. This section describes such a control scheme that weaken the stator flux, by sensing the angular frequency "ω_e", beyond the frequency from where quasi-PWM mode starts.

In EV drive battery voltage varies widely depending on state-of-charge and charging and discharging conditions. As dc link voltage "V_d" varies, the fundamental voltage "V_s" at square-wave mode will also vary proportionately. Figure 3.31 shows square-wave operation with variable V_d. For square-wave operation with variable V_d, the corresponding V_s will vary from V_{s1} to V_{s2}, as shown in Figure 3.31, and the base frequency "ω_b" will also vary from ω_{b1} to ω_{b2} respectively. Vector control will be assured in field weakening mode, if current control by PWM or quasi-PWM is maintained. This means that applied voltage in field-weakening mode should be below the square-wave voltage. So, it is required to establish rated stator flux "ψ_{sr}" in constant torque region and then weaken the stator flux "ψ_s" in field-weakening region so that square wave is never reached. This can be done as described in following derivation:

$$V_s = \frac{2}{\pi} V_d \quad \text{..... (3.44)}$$

V_s - A = Limit voltage, where A is constant. Note that pure PWM limit is 78.5% of square-wave voltage [5].

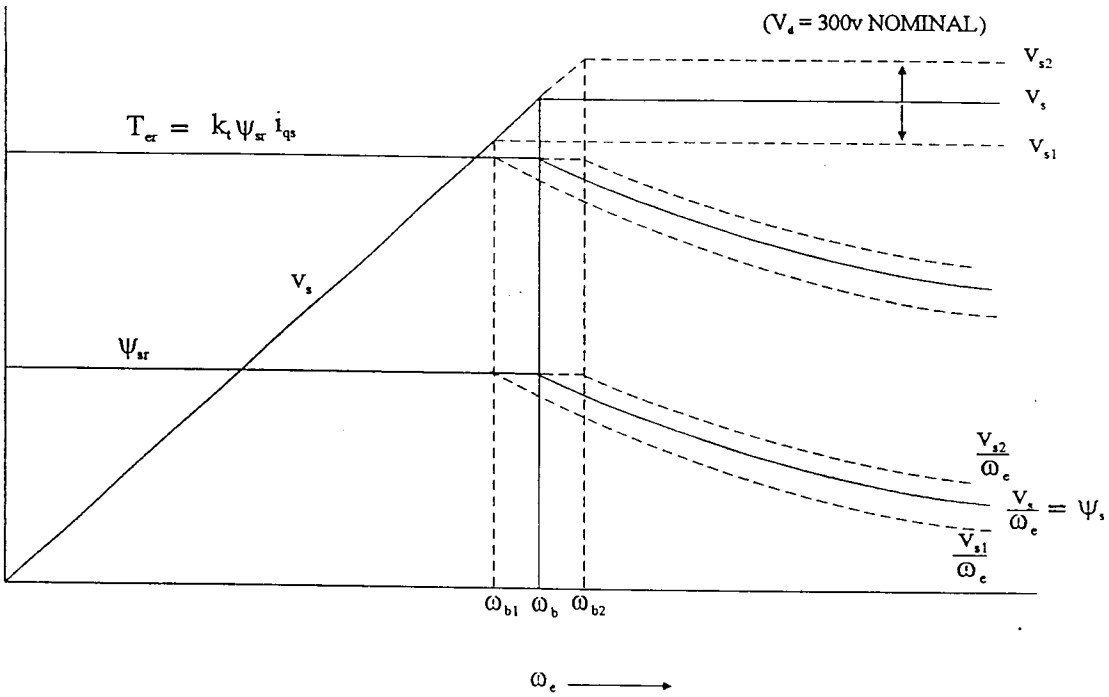


Figure 3.31 Square-wave operation with variable battery voltage "V_d"

$$\psi_{s(s.w.)} = \frac{V_s}{\omega_e} \quad \text{..... (3.45)}$$

$$\psi_{s(limit)} = \frac{V_s - A}{\omega_e} \quad \text{..... (3.46)}$$

$$\psi_{s(limit)} = \frac{\frac{2}{\pi} V_d - A}{\omega_e} \quad \text{..... (3.47)}$$

As shown in equation 3.47, stator flux can be calculated from the V_d and ω_e information in field-weakening region, where "A" can be fine tuned during experiment study. Figure 3.32 shows the PWM/quasi-PWM operation by limiting supply voltage to V_s - A. As shown in Figure 3.32 $\psi_{s(limit)} > \psi_{sr}$ if $\omega_e < \omega_{b1}$. This control scheme can be implemented as shown in Figure 3.33. Since ψ_s profile is symmetrical in two quadrants, it is not necessary to determine $+\omega_r$ and $-\omega_r$.

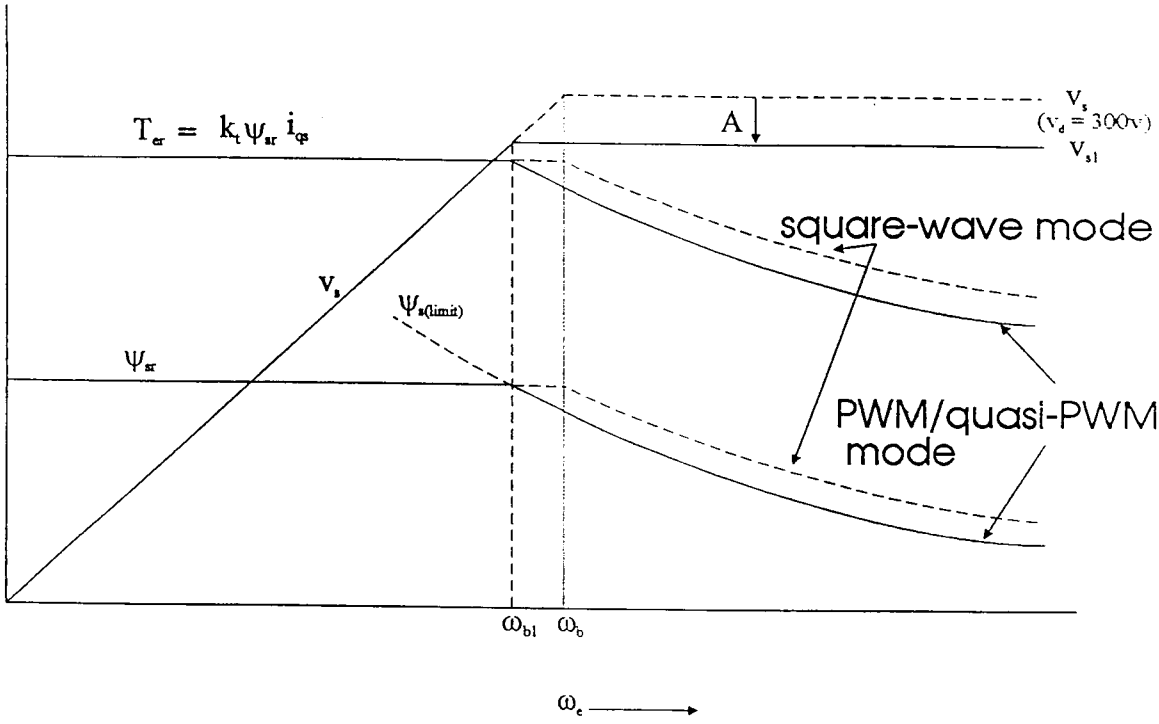


Figure 3.32 PWM/quasi/PWM operation by limiting phase voltage "V_s"

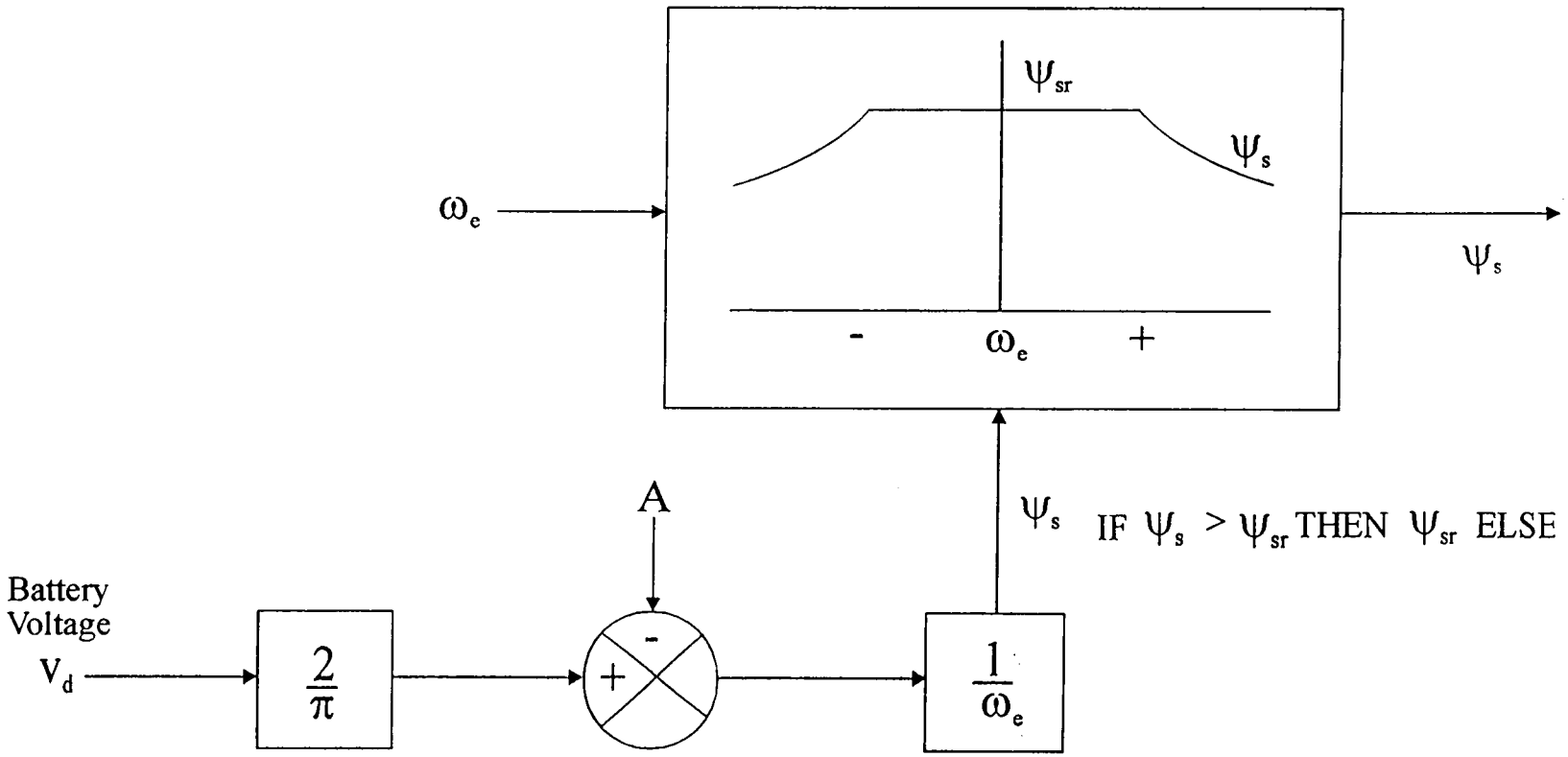


Figure 3.33 Field-weakening flux program block

CHAPTER 4

SIMULATION STUDY

The drive system with the proposed control strategy was simulated extensively prior to the laboratory experimentation. PC-SIMNON language [34] and MATLAB-SIMULINK [35] were used for simulation. The 100 KW induction motor was simulated with a dynamic d-q model using the nominal parameters as given in Table 4.1.

Table 4.1 Machine ratings and parameters.

Parameter Name	Value
R_s	0.01121 Ω
R_r	0.01243 Ω
L_{ls}	$43.8*10^{-6}$ H
L_{lr}	$43.8*10^{-6}$ H
L_m	0.00213 H
i_{ds} rated	66.6 A
i_{qs} rated	556 A
ψ_s rated	0.1419 Weber
ω_{sl} rated	4 Hz
I_s rated	396 A

Machine losses were introduced in the machine model simulation. The drive was operated in DVC mode with constant torque in the beginning, and then speed was gradually brought down to zero. A one stage digital low pass filter integrator with large time constant was found to generate excessive pulsating torque due to a dc offset problem. Thus, a multi stage programmable low pass filter was designed to overcome the dc offset problem near zero speed. The drive system could be started directly in DVC mode from zero speed, but there was excessive oscillatory torque as shown in Figure 4.1.

To over come this problem, Blaschke start-up mode was developed. Blaschke equations were originally derived in rotor flux orientation by F. Blaschke. Stator flux oriented Blaschke equations were derived to suit the current control strategy, which is described in Appendix B. The system cannot be operated by using only the Blaschke current model equations because when Blaschke equations for stator flux orientation were derived the speed was set equal to zero. So, before speed increase in the machine the control should be transferred to the DVC control mode. Otherwise, the stator flux estimation by Blashke mode will be wrong as the machine speed increases. Figure 4.2 shows the system performance if the control is not transferred to DVC mode at the correct instant. As shown in Figure 4.2, first stator flux was established to the rated value and then the torque command (T_e^*) of 100% rated torque was given. The Figure illustrates that since the control mode was not transferred from the Blaschke mode to the DVC mode at the correct instant, the stator flux estimated by Blaschke mode was falling down as speed built-up in the machine. Then, the stator flux was established by Blaschke mode, control was transferred to the DVC control, and after some time, the torque command was given. Transition from Blaschke to DVC and DVC to

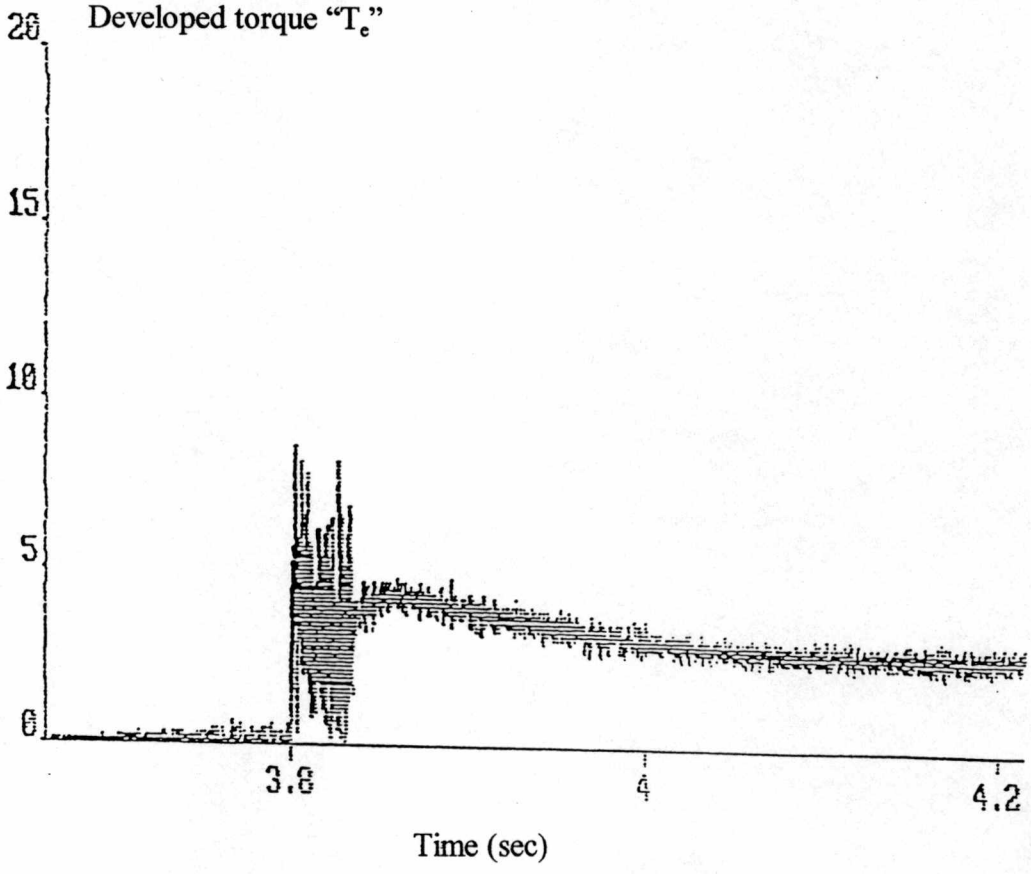


Figure 4.1 Start-up performance of the system directly from DVC mode

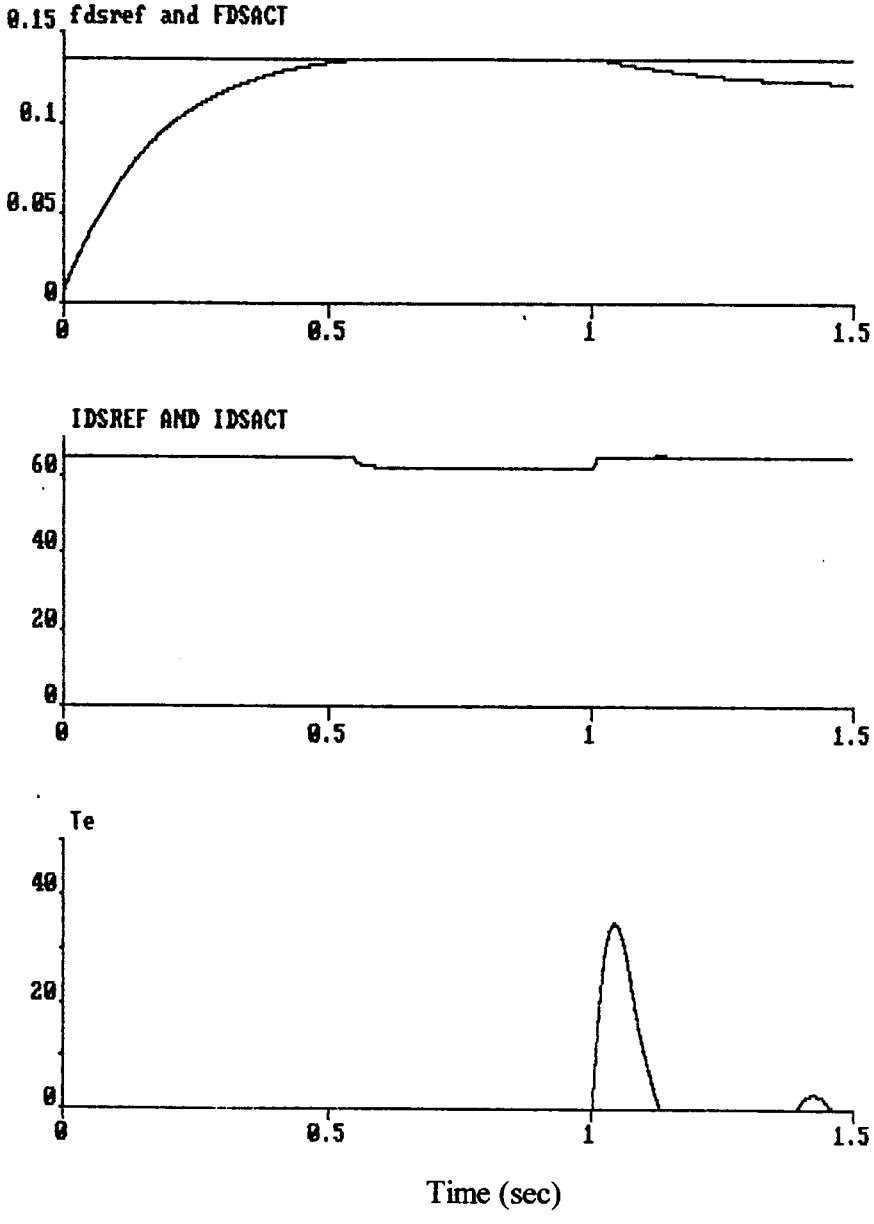


Figure 4.2 Sustained operation with Blaschke mode

Blaschke mode is shown in Figure 4.3. As shown in Figure 4.3, first the rated stator flux was established then reference torque command was given the step commands of 0 to 10 to 0 to 20 Nm to watch the system performance during transition from Blaschke to DVC and DVC to Blaschke control mode.

The system should also work in the field weakening region. So, above the base speed the stator flux was decreased as a function of frequency to maintain pure PWM operation. A detailed explanation of the field weakening region operation is given in Chapter.3. Figure 4.4 shows the system performance in the field weakening region.

The drive efficiency was optimized by using a neuro-fuzzy controller. The detail of this controller is described in Chapter 3. The machine losses were intentionally introduced into the machine model. Converter losses were not simulated. First the fuzzy controller was used to optimize the drive efficiency at different operating points. Figure 4.5 shows the fuzzy efficiency controller performance.

After fine tuning the fuzzy controller, the input-output data Table was generated to train the neural network controller. Afterwards, the neural network controller was tested at different operating points. Then neuro-fuzzy efficiency optimization was simulated using SIMNON, integrated with the complete system, and tested at different operating points. Figure 4.6 shows the performance of the neuro-fuzzy efficiency optimization controller at the operating condition of 1000 rpm and load torque of 4.5 Nm. As the steady state condition is detected by the torque loop error and frequency , the rated flux is decremented in steps. For constant speed and load torque (i.e., for constant output power), the input dc power decreases that indicates the improvement of efficiency. Note that as i_{ds} decreases with flux,

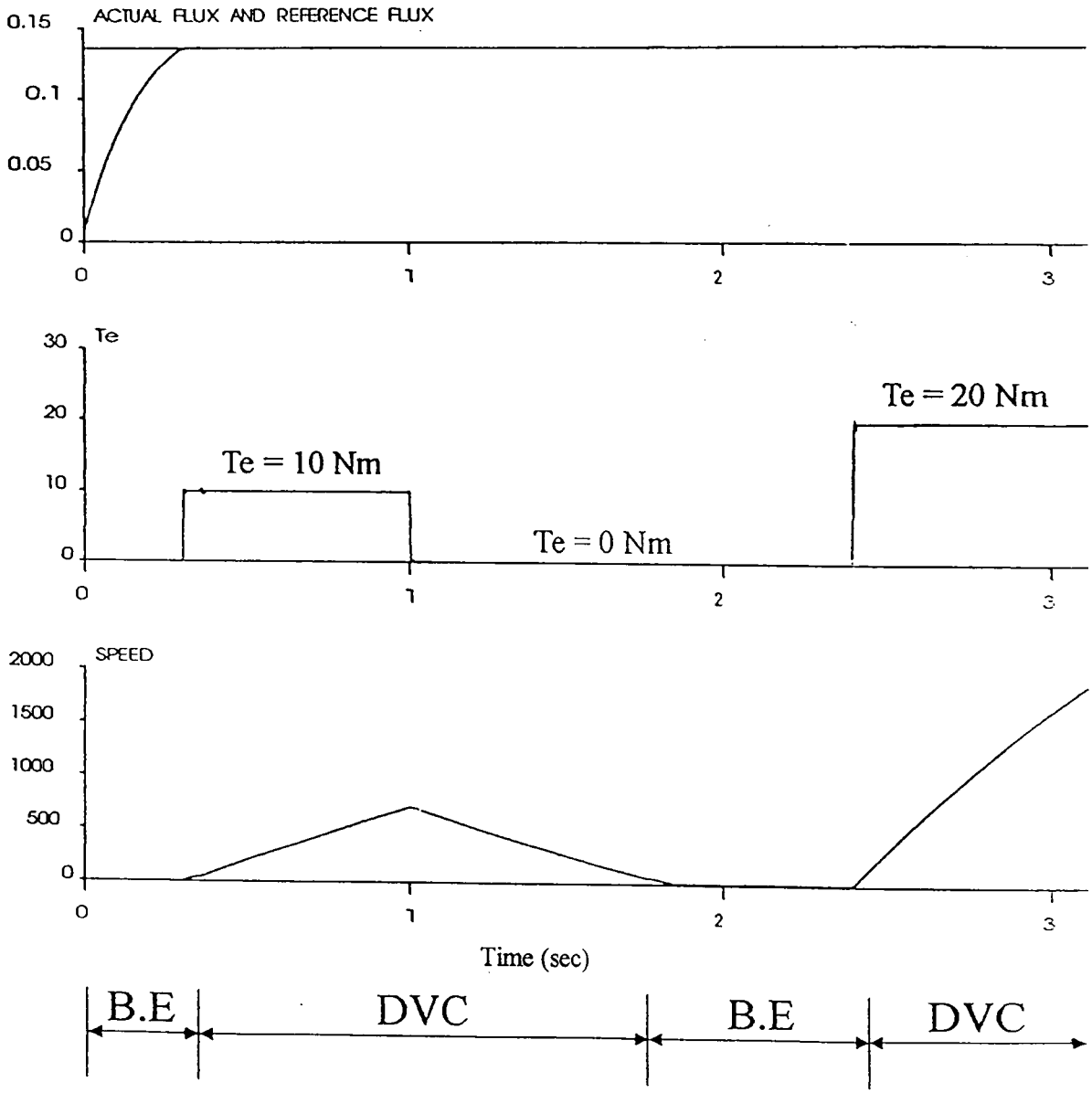


Figure 4.3 Start-up and transition performance between Blaschke and DVC modes

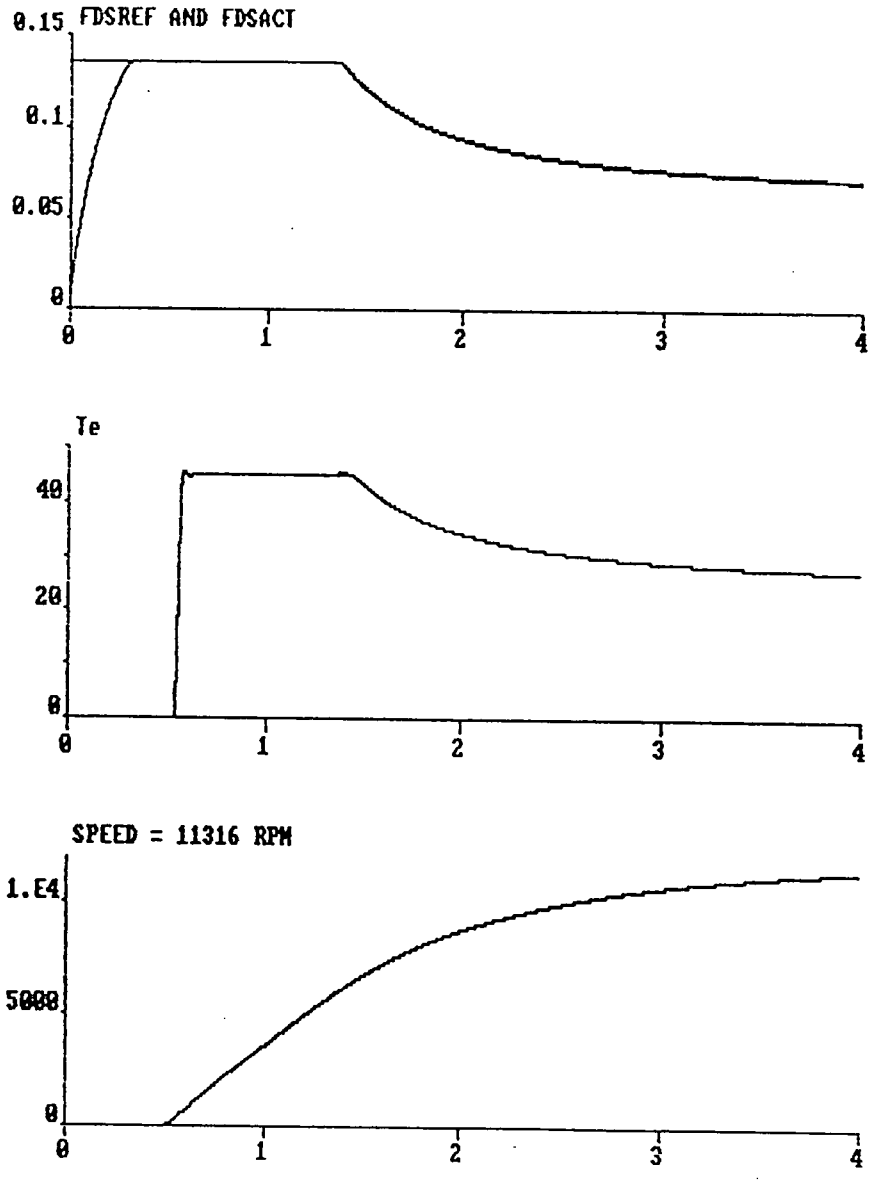


Figure 4.4 The drive system performance in the field-weakening region

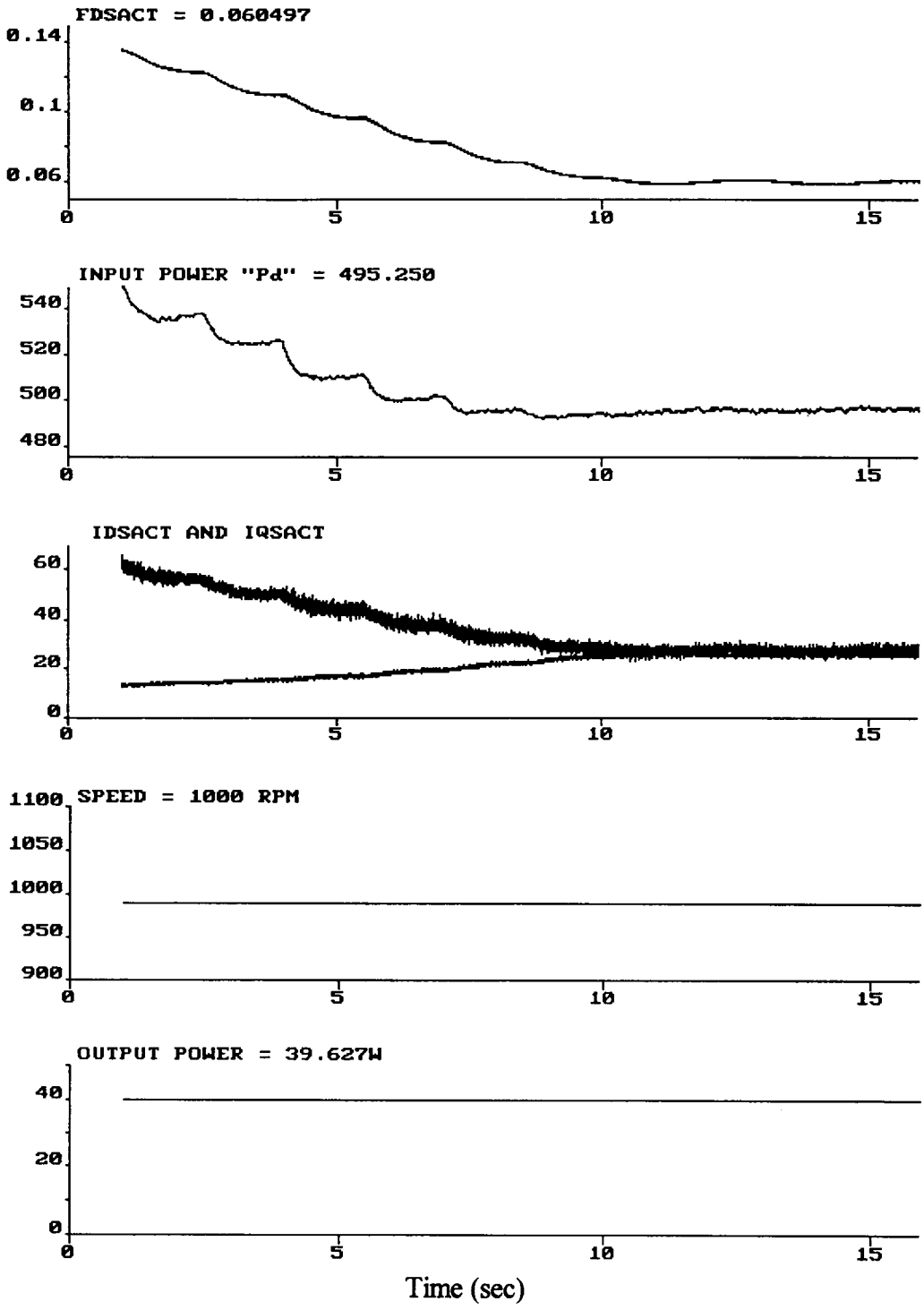


Figure 4.5 Time domain optimum efficiency search curves by fuzzy efficiency controller at speed (N) = 1000 RPM and torque (T_e) = 4.5 Nm

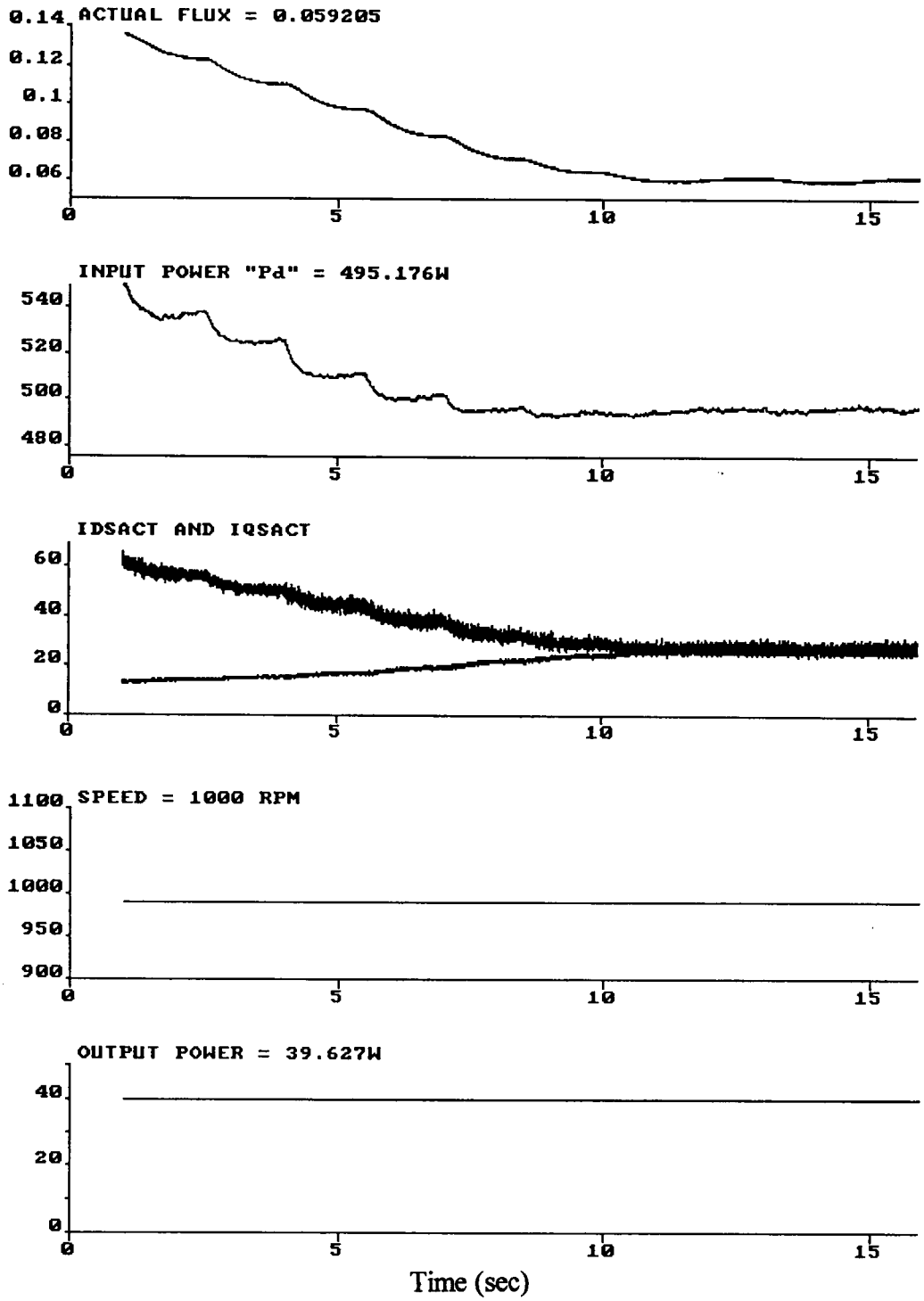


Figure 4.6 Time domain optimum efficiency search curves by neuro-fuzzy efficiency controller at speed (N) = 1000 RPM and torque (T_e) = 4.5 Nm

i_{qs} increases so that the developed torque always balances the load torque. The flux decrementation steps progressively decreases until optimum efficiency (i.e., minimum P_d) condition is attained. At steady state, the operation will oscillate about the optimum point. At every decrementation of flux, a pulsating torque is likely to develop which is not acceptable to EV drive. As mentioned before, the ramping of flux signal along with high gain torque loop practically eliminates any pulsating torque. Other operating points were found to give similar good performance.

In stator flux orientation, the correct stator resistance value at any operating point is very important, especially near zero speed where stator resistance drop becomes dominant [4]. To estimate the stator resistance precisely, a quasi-fuzzy stator resistance estimator was developed as described in Chapter 3. Thermal data for the 5 hp machine was obtained from the experiment study of the machine at different stator currents and different frequencies. Then on the basis of those data, a quasi-fuzzy R_s estimator was developed. Finally, C-language was used to simulate the estimator and test it separately without integrating with the drive system. Figure 4.7 shows the performance of the quasi-fuzzy Rs estimator at different speeds and different stator currents. As shown in Figure 4.7, it is very easy to evaluate that with the variations in stator current or the machine speed, the stator temperature varies and thus the stator resistance.

SIMNON simulation programs are given in Appendix C. MATLAB-SIMULINK block diagrams, for the complete drive and subsytems used into the drive system, are given in Appendix D. C-language simulation program for quasi-fuzzy stator resistance estimator is given in Appendix E.

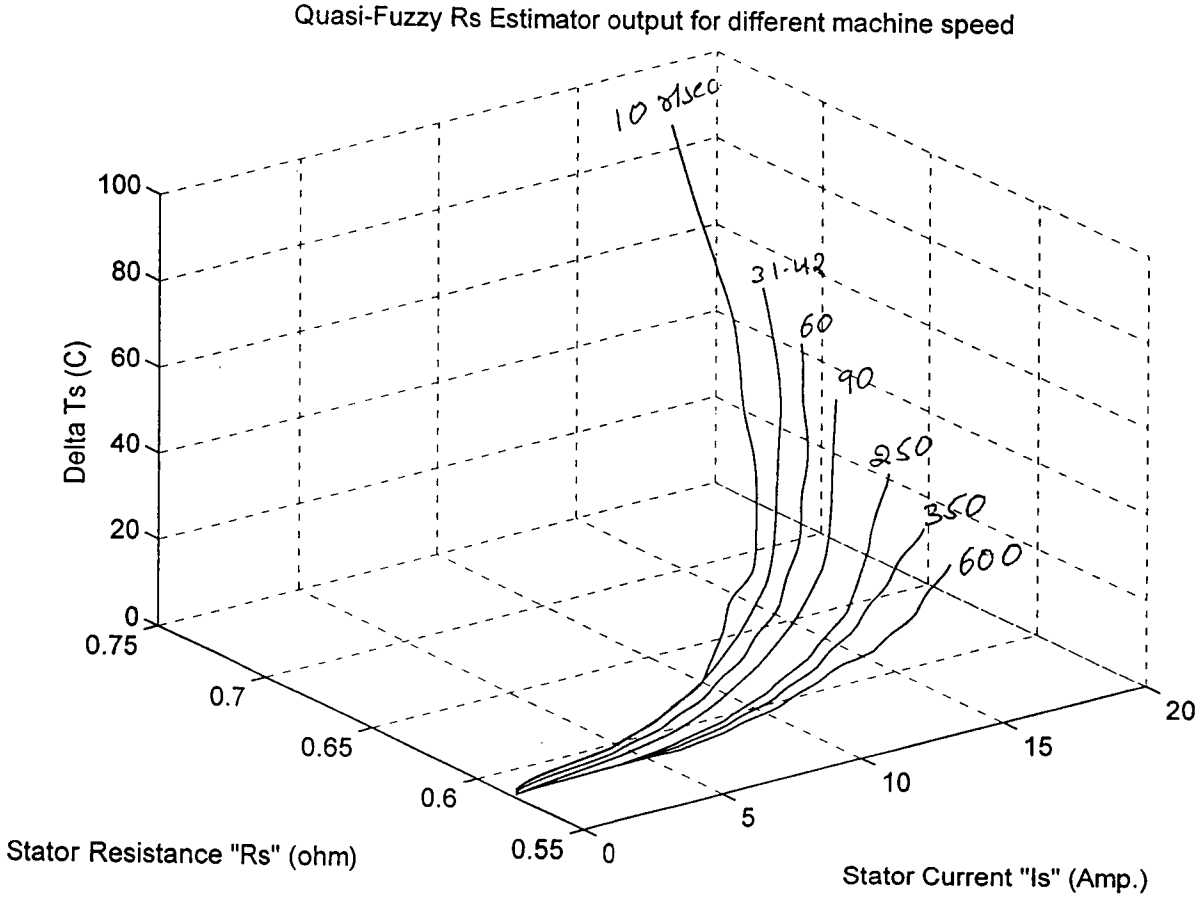


Figure 4.7 Estimated stator temperature and stator resistance rise vs. stator current at different frequency (steady state) using quasi-fuzzy R_s estimator

CHAPTER 5

SOFTWARE DESIGN

5.1 Software Description

The TMS320C30 has user-friendly tools for software development and debugging with a powerful optimizing C compiler and a Digital Signal Processing (DSP) operating system [36]. The C compiler for the C3x family has exceptional efficiency, which makes a C program as effective as a comparable assembly-language program. Such effectiveness is possible because Texas Instruments designed the TMS320C30 with extended memory space, a software stack and a large on-chip register file which results in an efficient compiler.

The assembler translates assembly-language source files into machine language object files. Source files can contain instructions, assembler directives, and macro directives. Assembler directives control various aspects of the assembly process such as the source-listing format, symbol definition, and method of placing the source code into sections. Macro directives permit a concise representation of groups of instructions that occur frequently.

The linker combines object files into one executable object module. As it creates the executable module, the linker performs relocation operations and resolves external references. The linker accepts relocatable COFF (Common Object File Format) object files, created by the assembler, as inputs. It can also accept archive library members and output modules created by a previous linker execution. Linker directives allow the user to combine object-file sections, bind sections or symbols to specific addresses or within specific portions of the DSP

memory, and define or redefine global symbols. An associated archiver can create macro or object-file libraries.

The C-source debugger is a very important tool for debugging programs. Its interface consists of a screen broken into windows that display the internal registers, the reverse-assembled program, memory, breakpoints and more [36]. There is a window where the C code is listed and can be stepped together with the assembly code.

There are two DSP boards, named DSP board #1 and DSP board #2, plugged into a host computer. The DSP board #1 runs the program "bdl.out" which corresponds to the C source program "bdl.c," the DSP board #2 runs the program "bd2.out" which corresponds to the source code "bd2.c," and the PC runs the DOS-executable program "run.exe," which corresponds to the C code "run.c."

The resident program in the PC is compiled with Microsoft Quick-C. By including the header TMS30.H in the source file all the library functions for using the DSP board are declared with function prototypes. Such library routines are used to download TMS320C30 object code to one or more boards, to start execution of that code (or stop if already running) and to pass data back and forth between the program running on each of the TMS320C30 boards and on the PC.

The function SelectBoard () is used to select a board for the first time. Then all functions used after a call to this function apply to the current board. To switch between boards after each one has been initialized by SelectBoard () the function WarmSelect() is used instead. Both functions need a hexadecimal I/O port base address; e.g. to select DSP board #1 the function should call SelectBoard(0x290), whereas to select DSP board #2 the function

should call SelectBoard(0x390).

The best way to move data between the PC and DSP is to use the dual-access memory on the DSP board. The board is normally supplied with 64K of memory in this dual-access area, addressed as locations 0x030000 through 0x3FFFF. Although the PC can read and write from and to other memory areas on the board, it incurs more overhead because the DSP chip must be "held" during PC accesses. The variables or arrays that are being passed must be defined as global variables. This will ensure that they are placed in the dual-access memory, in accordance with the ".bss" memory section defined in the mapfile "LSICMAP.CMD." The linker automatically reserves the first sixteen words in the dual-access memory to hold variables pointed from 0x030000 through 0x3000F. Those locations can hold pointers to variables or arrays of pointers, therefore, they are used to communicate between the DSP boards and the PC.

Before describing the overall control software, it is important to understand how the interrupts are handled in the C programs that run in the DSP boards. The system comes pre-configured with "sections" for setting up interrupt vectors by assembler directives that are automatically passed to the C30 assembler after the C Compiler runs. The interrupt names, section names, and vector locations are given in the Table 5.1

In order to set up an interrupt vector (a pointer to the interrupt service routine placed in an absolute location in low memory), the following statements must be used at the beginning of the interrupt routine:

```
asm(" .sect\".int02\"");
asm(" .word_c_int02");
asm(" .text");
```

Table 5.1 Interrupt names, section names and vector locations.

Interrupt Names	"Section" Name	Absolute location
RESET	.int00	0x00
INT0	.int01	0x01
INT1	.int02	0x02
INT2	.int03	0x03
INT3	.int04	0x04
XINT0	.int05	0x05
RINT0	.int06	0x06
XINT1	.int07	0x07
RINT1	.int08	0x08
TINT0	.int09	0x09
TINT1	.int10	0x0a
DINT	.int11	0x0b

The first statement tells the assembler to create a program section named ".int02." The assembler output is passed to the linker, which places this section in absolute memory location 02. It is essential to direct the linker to use the map file LSICMAP.CMD for this process. The second statement tells the assembler to form the address of the interrupt service routine function, which gets placed into an absolute memory location. The underscore must be used because the C compiler puts an underscore in front of all function and variable names before passing them to the assembler. The third statement tells the assembler to go back to

assembling things into the "text" (code) section, instead of the absolute-located sections that were created for the interrupt vectors.

In order to initialize and enable the interrupts, a statement similar to the following in-line assembly language statements must be used. The following are statements for INT1:

```
asm(" OR   2h,   IE");
asm(" OR   2000h, ST");
```

The first statement sets one of the bits in the C30's Interrupt Enable (IE) register by OR'ing a value of 02h with the current contents of the register. The second statement sets the Global Interrupt Enable bit (bit #13) in the C30's Status Register. This bit must be set or the C30 will not respond to any interrupts, even if they are enabled in the IE register.

The control software that runs in each of the DSP boards is a multitasking, real-time program. Therefore, it is necessary to trigger the interrupts by software in such a way that the lower level interrupts are able to be interrupted by higher level ones with automatic context save/restore of variables. The following statement is used to trigger the interrupt #2 (INT2) by software, as if one hardware interrupt has occurred:

```
asm(" OR   2h,    IF");
```

The most critical time assignment task is the operation of writing to the Hanning chip the 16-bit words for the reference frame voltages (v_{ds}^*, v_{qs}^*) and the angle reference (θ_e^*). For this reason, the control software timing is tied to the interrupt generated by the Hanning chip. The programs "bd1.c" and "bd2.c" have a basic structure as indicated Figure 5.1, where the background loop for host communication is interrupted by four services: ADC interrupt service routine (INT0), and the corresponding decreasing priorities interruptions for task #1

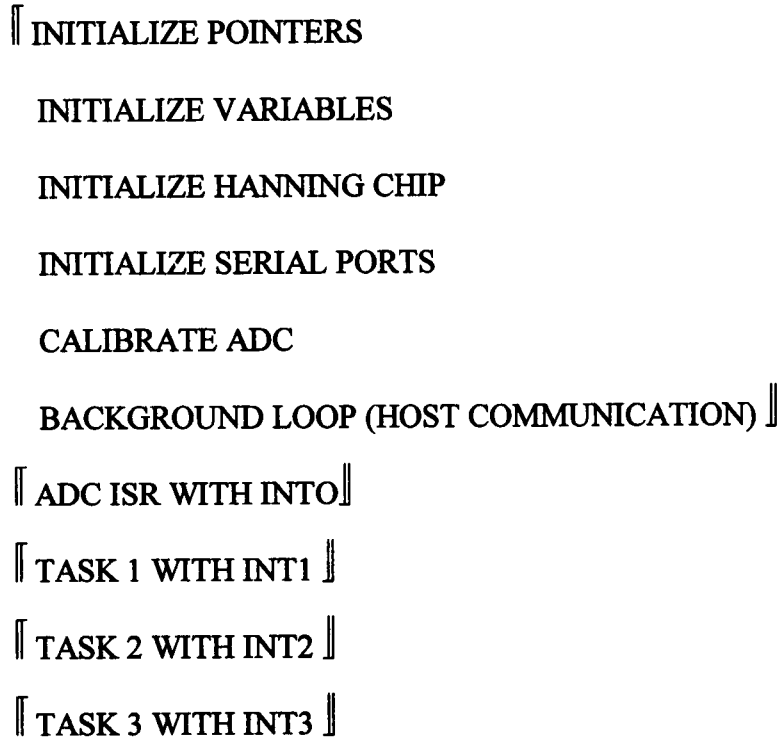


Figure 5.1 Flow diagram for the multitasking real-time DSP code

(INT1), task #2 (INT2) and task #3 (INT3).

Figure 5.2 shows the tasks coordination chart. The four-channel ADC conversion is triggered at the end of task 1 and although the ADC is INT0-based, the complete conversion of the four channels takes 16 μs, which is within the 128 μs timing for INT1. The context save/restore is automatically handled by the DSP, due to the built in priorities for the interrupts INT0, INT1, INT2, and INT3. The interrupts INT2 and INT3 are triggered within the INT1 by a loop counter, so as to generate 1.028 msec and 1.024 sec interrupts as explained later.

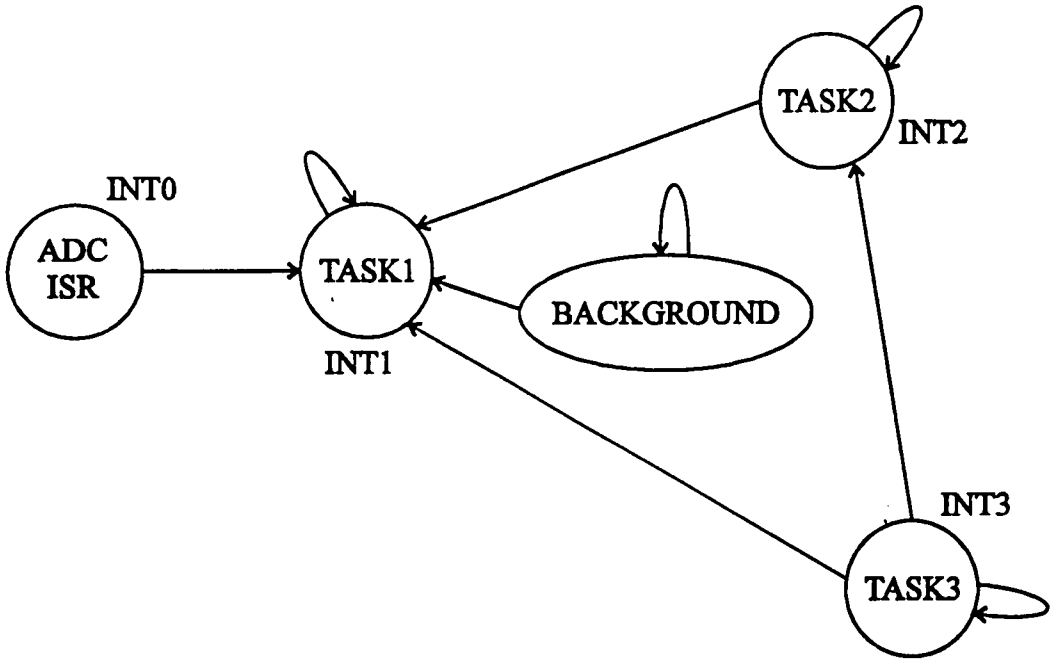


Figure 5.2 Tasks coordination chart

The interrupt service routine that converts the four channels from the ADC board is triggered at the end of task 1. The total conversion time is 16μs. Therefore, the conversions are over before the next sampling time (128 μs) for task 1. Since the software code for conversion is common for both DSP boards, the explanation of operation follows.

The interrupt service routine for the ADC conversion is embraced by c-int01(). The IO board is triggered to start the conversion of the first channel at the end of task 1 by writing to the corresponding C pointer (identified by *timer). Only one analog to digital converter is used to convert all four channels, therefore, each channel is converted one at a time. At the end of the conversion, an interrupt (INT0) is generated after each channel is finished being converted, producing four interrupts after a single ADC software trigger. The variable "adc_value" reads the converted channel in the C pointer named *adc_result.

Because the ADC has a resolution of 12 bits, the variable adc_value is shifted 20 positions to the right, due to the fact the all DSPLINK communications are made on the upper 16 bit word of the TMS320C30. The variable isr0 is initially at value 1, indicating that the first channel needs to be converted. After each conversion, isr0 is updated for the next channel and the pointer *control triggers the IO board again. At the forth channel, isr0 is set to the value 1 and the IO board conversions are held until the next cycle

5.2 Control Software in the DSP Board #1

Interaction with Hanning chip, IO board, serial port, and host communication are memory mapped. Therefore, the corresponding C pointers are initialized in accordance to the Table 5.2

The hardware interrupt INT1 indicates that the Hanning chip is ready to receive the words for v_{ds}^* represented by the variable UA, v_{qs}^* by UB, and θ_e^* by PHI1. The interrupt is initiated at the end of a Hanning processing cycle with the positive edge of the Hanning INT signal which is counted four times by the two flip-flops connected to the Hanning INT. The software for the interrupt is coded inside the function c_int02(). Initially, the global interrupts are disabled and the control word is written means of write address (WAD = 0) into status register (A1=1) to write UA, UB and PHI1. The UA, UB and PHI1 words are separated by assembly "NOP" commands in order to achieve the required processing timing in the Hanning chip. The voltage words UA and UB are normalized in 2's complement format from 0x8000 to 0x7FFFF as the synchronous current controller outputs v_{ds}^* and v_{qs}^* vary from -1 to +1. The angle word PHI1 is normalized from 0x0000 to 0x60000 as the phase angle varies from 0 to 2π.

Table 5.2 Memory mapped hardware addresses for DSP # 1

Memory pointers	Hex address
*pwm_data	0x0800000
*pwm_status	0x0800001
*encoder_read	0x0800002
*control	0x0800008
*timer	0x0800009
*adc_result	0x080000a
*dac0_output	0x080000b
*serial_global()	0x0808040
*xmit_control()	0x0808042
*rece_control()	0x0808043
*ser_tim_control()	0x0808044
*ser_tim_count()	0x0808045
*ser_tim_period()	0x0808046
*data_xmit()	0x0808048
*data_received()	0x080804C
*serial_global 1	0x0808050
*xmit_control 1	0x0808052
*rece_control	0x0808053
*sec_tim_control 1	0x0808054
*ser_tim_control	0x0808055
*ser_tim_period 1	0x0808056
*data_xmit 1	0x0808058
*data_received 1	0x080805C
*PC Proceedflag	0x0030000

Memory pointers	Hex address
*outputvec	0x0030001
*LOG3	0x0030003
*DAC0_reg	0x0030008
*DAC1_reg	0x0030009
*SWITCH	0x003000B

After the operation of writing to the Hanning chip, the codes for the serial communication are implemented. There are six signals to be sent to Board 2 and three signals to be received from Board 2. The flow charts below show how this communication is accomplished. Figure 5.3 (a) and 5.3 (b) show the sequences to transmit and to receive the data from one DSP to the other DSP, respectively.

The codes for serial port communications are as follows:

```
/** Test if serial port # 0 is ready to transmit seven signals to the other DSP ****/
 cal_status = (*serial_global0 & 0x02) && (*serial_global0 & 0x00800000);

if( cal_status != 0 );
{
if(i<5)
{
*data_xmit0 = trans[i];
I++;
}
else
I=0;
}

/** Test if serial port # 0 is ready to receive three signals from the other DSP ****/

  cal1_status = (*serial_global0 & 0x01) && (*serial_global0 & 0x02000000);
if( cal1_status != 0 );
```

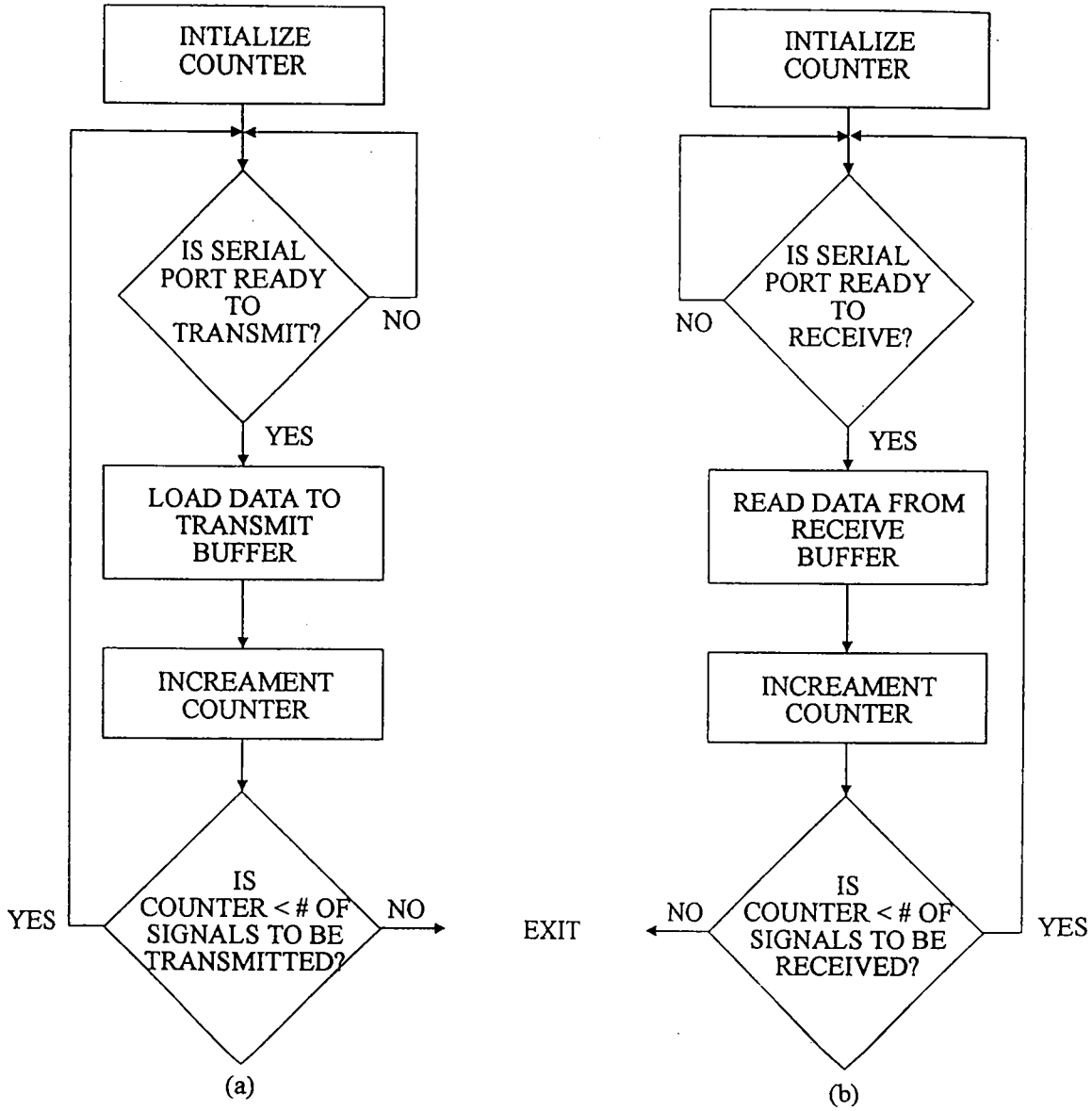


Figure 5.3 Serial port communication between two DSPs

(a) Transmit data through serial port.

(b) Receive data through serial port.

```
{
if(j<3)
{
     rec[j] = *data_received0;
j++;
}
else
j=0;
}
```

The serial port # 0 sends v_{qs}^{s}, v_{ds}^{s}, i_{qs}^{s}, i_{ds}^{s}, ψ_{qs}^{s} and ψ_{ds}^{s} to the DSP2 and receives v_{qs}^{*},v_{ds}^{*} and ω_e from the DSP2.

Programmable time constant and amplitude compensation gain of the cascaded low-pass filter are estimated using ω_e information. Then, q and d-axes stator fluxes are being calculated. Using the q and d-axes information, magnitude stator flux and $\sin\theta_e$ are being calculated. θ_e signal, to feed to the Hanning chip, is calculated using arc sine expression. Then v_{qs}^{*},v_{ds}^{*} and θ_e are scaled for 16 bit words and then left shifted 16 bits because the DSPLINK interface uses only the upper 2 bytes in the TMS320C30 data bus. The last assignment for this interrupt service routine is to trigger the IO board for conversion and to trigger the task2 and task3 at every 1.024 msec and 1.024 sec respectively. There are two loop counters lpc2 and lpc3 which counts 8 times and 10000 times respectively to trigger the interrupts by OR'ing the interrupt flag (IF). For example;

```
if (lpc2 == 8)
    {
     lpc2  = 1;
     asm(" OR 0004h, IF");     /* Trigger interrupt # 2 */
    }
  else {++lpc2;}
```

```
if (lpc3 == 10000)
    {
     lpc3  = 1;
     asm(" OR 0008h, IF");      /* Trigger interrupt # 3 */
    }
  else {++lpc3;}
```

At the end of the task 1 the global CPU interrupt is enabled and the IO board is triggered to receive the four channels one at a time by the following code

```
  asm("  OR  2000h, ST");  /* Enable global CPU interrupts    */

 *timer = adc_trigger;     /* Triggering all the S-H to receive the   */
                           /* four-channels one at the time           */
```

The interrupt INT2 corresponds to the control software that must be executed each 1.024 msec and is triggered by software within task 1 with the loop counter lpc2. This interrupt is identified by the software codes embraced by c_int03. Software codes to display selected signals on the oscilloscope through the dual port memory is written in this interrupt routine. The last piece of software executed by the task 2 is the signal log service. The reason for such software is for documentation and debugging purposes. It is possible to write an ASCII file in the PC with five columns. The first column is the time interval for the corresponding logging. The following four columns are four signals which can be chosen at the time of compilation. Here it is demonstrated for the case of magnitude flux estimated by a cascaded low pass filter (ys_lp), magnitude flux estimated by Blaschke equation (ys_bl), Hanning chip input (vqs) and estimated ω_e (we).

```
Comm2 = outputvec; /* Put starting address of array in absolute    */
                   /* location so the PC can find the array        */
if(logindex == 39)    /* The logging will happen at every 79.872ms */
 {
```

```
        logindex = 0;
        if(write_to_log == 0)
         {
                write_to_log = 1;
                *PCproceedflag = 1;
                if(x<k)
                 {
                  outputvec[x++] =ys_lp;
                  outputvec[x++] =ys_bl;
                  outputvec[x++] =vqs;
                  outputvec[x]  =we;
                  x=0;

                 }
         }
        else
         {
                write_to_log = 0;
                *PCproceedflag = 0;
         }
}
```

5.3 Control Software in the DSP Board #2

The DSP board # 2 has a similar software structure as the DSP board # 1 except writing to the Hanning chip. The DSP board # 2 also has an IO board for acquisition of analog signals, it communicates with the DSP board # 1 through the serial port, and it also interacts with the PC through a host communication dual-port memory. Table 5.3 shows the initialization of the required C pointers for hardware interface.

Table 5.3 Memory mapped hardware addresses for DSP #2

Memory pointers	Hex address
*pwm_data	0x0800000
*pwm_status	0x0800001
*encoder_read	0x0800002
*control	0x0800008
*timer	0x0800009
*adc_result	0x080000a
*dac0_output	0x080000b
*serial_global()	0x0808040
*xmit_control()	0x0808042
*rece_control()	0x0808043
*ser_tim_control()	0x0808044
*ser_tim_count()	0x0808045
*ser_tim_period()	0x0808046
*data_xmit()	0x0808048
*data_received()	0x080804C
*serial_global 1	0x0808050
*xmit_control 1	0x0808052
*rece_control	0x0808053
*sec_tim_control 1	0x0808054
*ser_tim_control	0x0808055
*ser_tim_period 1	0x0808056
*data_xmit 1	0x0808058
*data_received 1	0x080805C
*PC_proceedflag	0x0030000

Memory pointers	Hex address
*FLUXREF	0x0030001
*outputvec	0x0030002
*TE_REF	0x0030003
*KP_T	0x0030004
*KI	0x0030005
*KI_T	0x0030006
*FLUX_MODE	0x0030007
*DAC0_reg	0x0030008
*DAC1_reg	0x0030009
*KP_F	0x003000A
*TE_MODE	0x003000B
*IQS	0x003000C
*IDS	0x003000D
*PIZERO	0x003000E
*KI_F	0x003000F

Task1 is being performed within INT1 service subroutine which is triggered by a hardware trigger from the Hanning control board. Task 1 performs following operations:

1. Serial port communication.
2. Scale received and sent signals.
3. Calculate $\sin\theta_e$, $\cos\theta_e$, $\sin\theta_e$' and $\cos\theta_e$'.
4. Calculate inverse vector rotated signals such as: i_{qs}^e, i_{ds}^e and ψ_{ds}.
5. Calculate decoupling current i_{dq}.

6. Calculate machine torque T_e.

7. Calculate slip frequency ω_{sl}.

8. Calculate angular frequency ω_e.

9. PI controllers for i_{qs} loop, i_{ds} loop, torque loop and flux loop.

10. Trigger INT2 , INT3 and 4-channel board at the end of the task 1 using same method as described in DSP board #1.

Task2 is being performed within the INT2 service subroutine which is triggered by using a software trigger within INT1 service subroutine. As described for DSP board # 1, Task2 in the DSP board # 2 also performs the operation such as: displaying selected signals on the oscilloscope through dual port memory , storing selected variables in ASCII form inside PC memory as a file and displaying selected variables on the screen for debugging purpose.

The next assignment for the Task2 is to perform Quasi-Fuzzy stator resistance estimation using the stator current and angular frequency information. The fuzzification, rule base evaluation and defuzzification are performed as already discussed. Initially, the control software identifies where the inputs stator current I_s and angular frequency ω_e belong. As an example, the indexing for the universe of discourse for I_s is arranged in Figure 5.4. The index "I" indicates one interval where two straight line equations defined the degree of membership. Each fuzzy set is indexed by two values of the variable I, e.g. if the I_s in pu is "VS" then I = 1 or I = 2, such fuzzification is indicated in the following software segment:

```
if (IS < 0.076) { I = 1;
} else if (IS < 0.114) { I = 2;
}
```

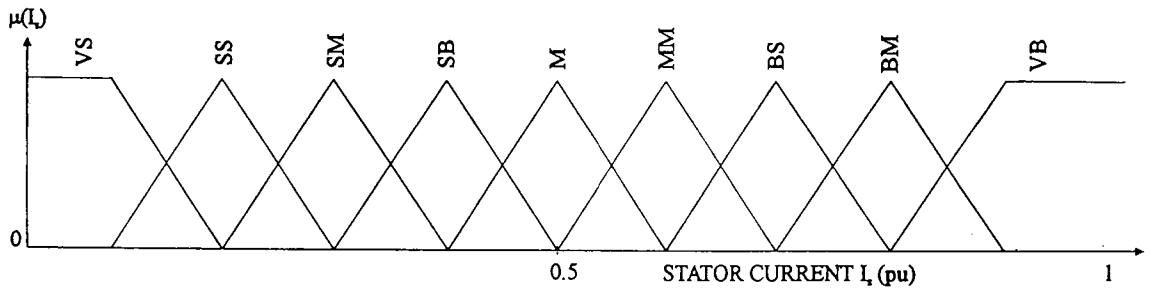


Figure 5.4 Example of indexing for C program membership function evaluation

The indexing of the intervals helps to evaluate membership functions MIS1 and MIS2 and corresponding rules R11 and R12. Using the same method the indexing pointer for the angular frequency is found and from that information the corresponding membership functions MW1 and MW2 and rules R21 and R22 can be evaluated. So at any instant four rules can be fired. The rule base evaluation can be made as follows:

/*"******** Perform the rule base evaluation*/

```
MRA = MIN(MW2, MIS2);
MRB = MIN(MW1, MIS2);
MRC = MIN(MW2, MIS1);
MRD = MIN(MW1, MIS1);
```

After finding four rules and four appropriate membership functions, the height method of the

defuzzification is used to defuzzify the output as follows:

```
SUM = MRA + MRB + MRC + MRD;
DT = (R11*MRA+R12*MRB+R21*MRC+R22*MRD)/SUM;
DTS = DT*100;
```

Where DTS is the change in the stator temperature. This output is then fed to the low pass filter with programmable time constant to achieve dynamic transient response of increase or decrease of the stator temperature.

The interrupt INT3 corresponds to the control software that must be executed each 1.024 sec and is triggered by software in the task1 with the loop counter lpc3. This interrupt is identified by the software code embraced by c_int04(). The neuro-fuzzy efficiency optimization codes are implemented inside this interrupt routine.

5.4. Host Communication Program

The host communication program downloads the two DSP programs in the corresponding DSP boards and wait for the user command. There is a screen with several options which can be accessed by typing the corresponding numbers, as indicated in Figure 5.5. Any time that "0" is entered the menu screen is printed on the screen. To switch between DSP boards the selection "4" should be typed. To halt the DSP program "1" should type. To reset the DSP "2" should be typed and to quit the host program " 3" should be typed and so on.

The control software codes for the DSP board # 1, the DSP board # 2 and the Host communication are given in Appendix F.

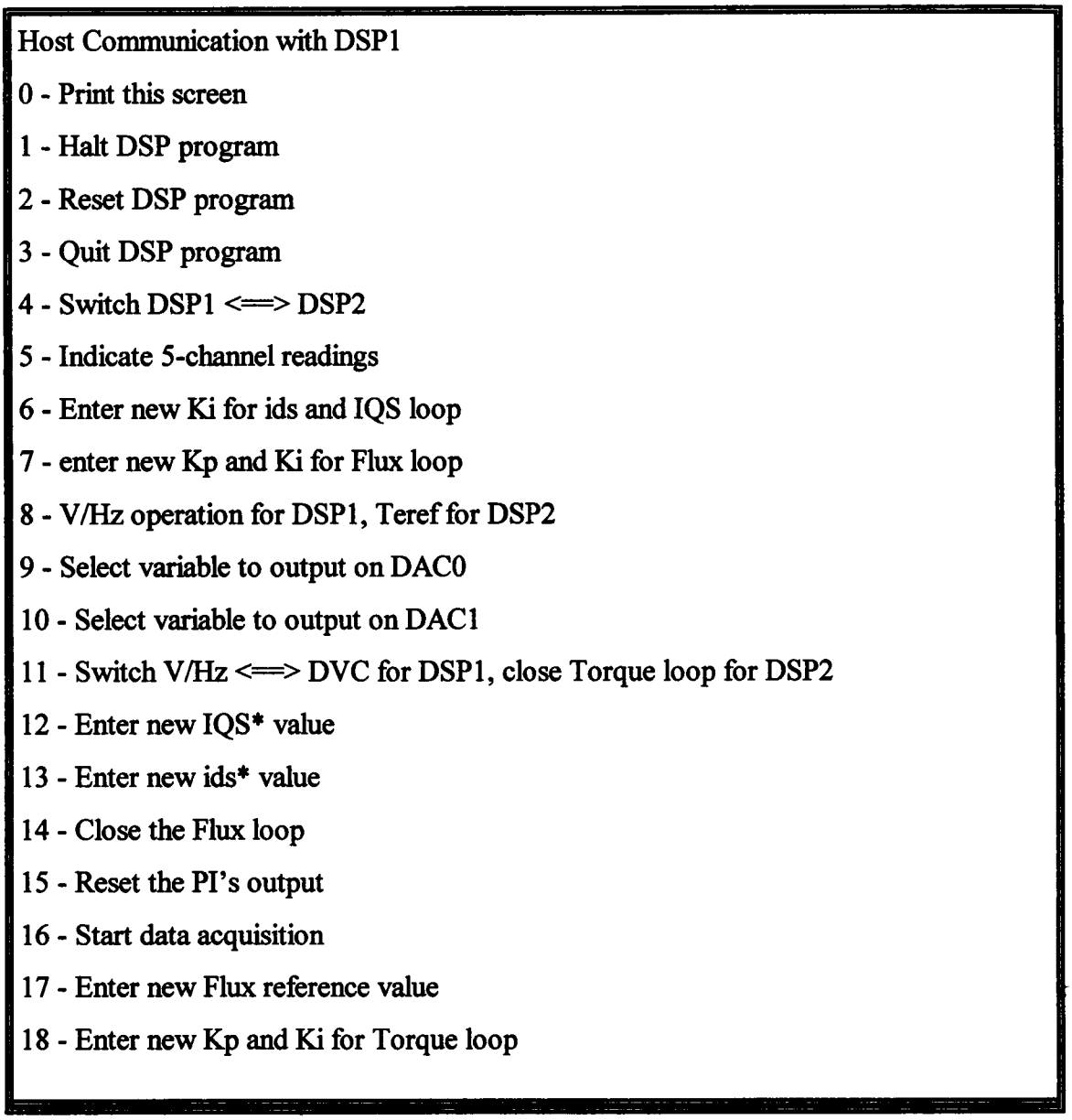


Figure 5.5 Host Communication program screen interface.

CHAPTER 6

SYSTEM DESIGN AND EXPERIMENTAL STUDY

6.1 Hardware Description

The sensorless efficiency optimized induction motor drive with neuro-fuzzy control was implemented in an experimental setup to validate the control algorithms and the overall design of the hardware and software. Figure 6.1 shows the hardware block diagram for the experimental setup. There are two Texas Instruments TMS320C30 digital signal processor boards and two analog I/O boards [38], manufactured by Spectrum, constituting a multi-processor, multi-tasking DSP system. The remaining circuitry was completely designed, built and debugged specifically for this project and will be described later.

Special purpose microprocessors, such as the TMS320 family of processors, have been used in digital signal processing since the early 1980s. The Intel 2920 appeared first, followed by NEC's uPD7720. In 1982, Texas Instruments Inc., introduced the first-generation TMS32010 digital signal processor, followed by the second-generation TMS32020 in 1985 and the faster C-MOS version TMS320C25 in 1986. The third-generation TMS320C30 is the one used in this project, and is a true 32-bit processor containing both integer and floating-point arithmetic units, 2048 x 32 bit words of on-chip RAM, 4096 x 32 bit words of on-chip ROM, a control unit and parallel/serial interfaces. This microprocessor operates from a 33.3 MHZ clock, achieving a performance of 16.7 million

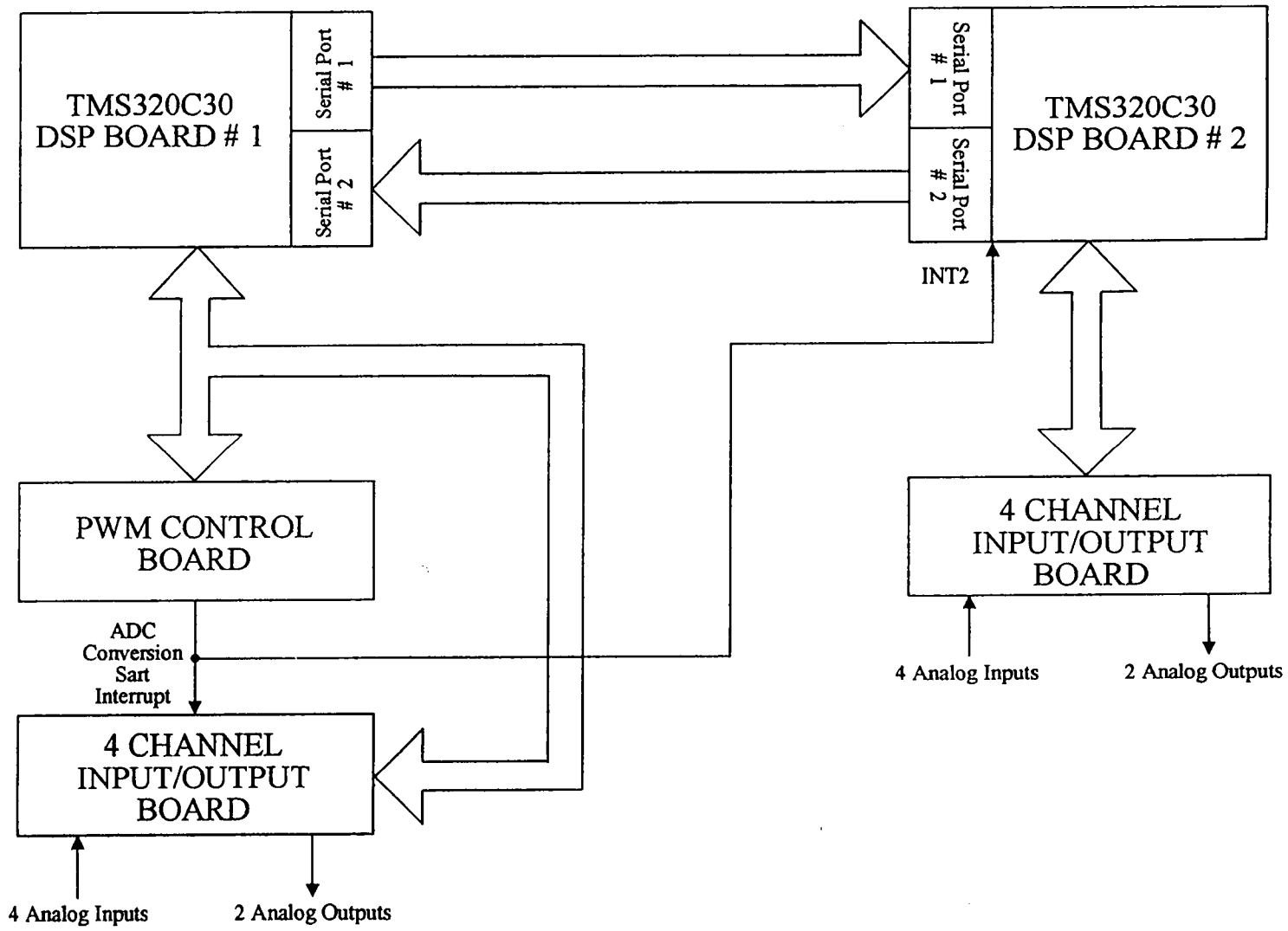


Figure 6.1 Hardware block diagram for induction motor drive system

instructions per second with features such as a 32-bit by 32-bit floating point multiplication in one instruction cycle and special addressing modes for circular buffering and bit reversal.

The board manufactured by Spectrum contains the TMS320C30 chip along with special features for ease of software and hardware development. The board is suitable for PC-AT compatibles as it uses the 16-bit ISA-bus. There is a dual port interface which allows the PC to access the memory in the DSP board by using the Hold/Hold Acknowledge feature of the TMS320C30. This feature allows the host program in the personal computer to communicate and change parameters on-the-fly in the software which controls the DSP board. The board occupies a single 16-bit slot within a PC-AT expansion bus and the base address is specified by links (LK2). The Processor Board #1 was configured for the address 0x290 and the Processor Board #2 for the address 0x390.

The TMS320C30 has two programmable serial ports which permit 8, 16, 24 or 32 bit transfers for connecting external CODEC interface or any other synchronous serial circuitry at a rate of 6 Mbits/sec. These ports have been buffered in the board and are accessible from the rear end plate of the PC. They were used for transferring signals between Processor Board #1 and Processor Board #2.

A parallel expansion system is provided as a memory-mapped peripheral area. It has a 16-bit width, and follows the proprietary DSPLINK bus [37]. Transfers over this expansion use 2 wait-states to achieve a 180 nsec transfer cycle which is suitable for ribbon-cable connection to peripheral boards. The DSPLINK in each Processor Board is connected to both the 4-channel I/O board and to the Hanning PWM integrated circuit. There is a prototype interface module which consists of bidirectional buffers and address decoders to

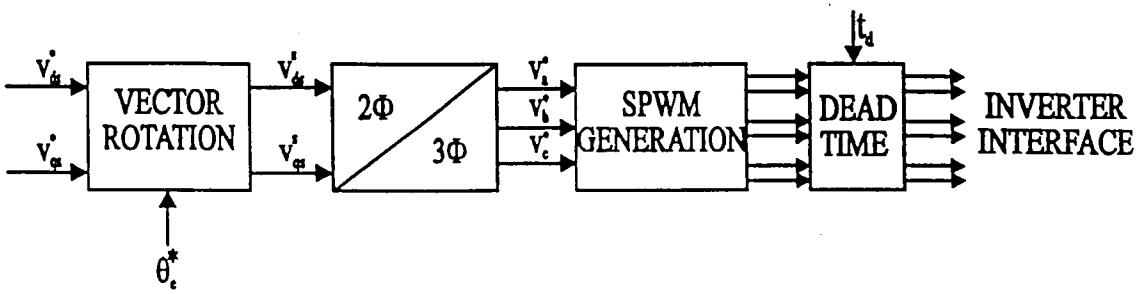


Figure 6.2 Hanning PWM integrated circuit

help interface module the DSPLINK bus to the Hanning PWM. The Hanning PWM chip generates the PWM pulse pattern signals [39] for the IGBT's gate drives. The inputs, as indicated in Figure 6.2, are the d-q synchronous rotating reference frame voltages (v_{ds}^*, v_{qs}^*) and the correspondent angle (θ_e^*). The Hanning PWM integrated circuit has the following features:

* 3-phase pulse width modulator (PWM) for 3-phase motors.
* Pulse pattern generation for a 3-phase supply at the required voltage, frequency and phase angle.
* Switching frequencies up to 20 kHz and resolution of the switching signals up to 50 ns.
* Pre-setable dead-band, turn-on and turn-off times.

* Provisions for turn-on, turn-off and dead-band compensation.

* Input voltage conversions from cartesian to polar form

* Optional 8/16 bit wide bus interface compatible with a range of 8 bit single-chip microprocessors and signals processors.

The pulse patterns output from the Hanning PWM integrated circuit are connected to the inverter interface circuit which consists of a low impedance RS-422 cable driver, a buffer that disables the gate pulses in case of fault, a latch-up to hold the signal fault and optocouplers to isolate the gate pulses from the IGBT's gate drives. The power IGBT inverter (100 KW) has built-in gate drivers, overcurrent, overvoltage and temperature protection. The inverter was built and tested at the Delphi Energy Systems Laboratory. Each subsystem is discussed in detail in the following sections.

6.1.1 Hanning PWM

The Hanning integrated circuit is mounted in a PC prototype vector board with wire-wrapped connections. This board is inserted in the PC ISA slot. The interface with the C30 DSP board is achieved with DSPLINK prototype board which is also mounted in the PC vector board. The circuit of the DSPLINK interface is given in Figure 6.3. All control signals and data signals which make up the DSPLINK interface with the TMS320C30 are physically carried on a 50-way conductor ribbon cable. The header connectors have two rows of 25 pins, with 2.54 mm spacing between all pins. The fully buffered data, port access strobes and control lines are brought out as wirewrap pins, which are inserted through holes in the prototyping card. There are read (Rx) and write (Wx) signals for the circuits that can be conveniently addressed by hardware links in the address decoder IC4.

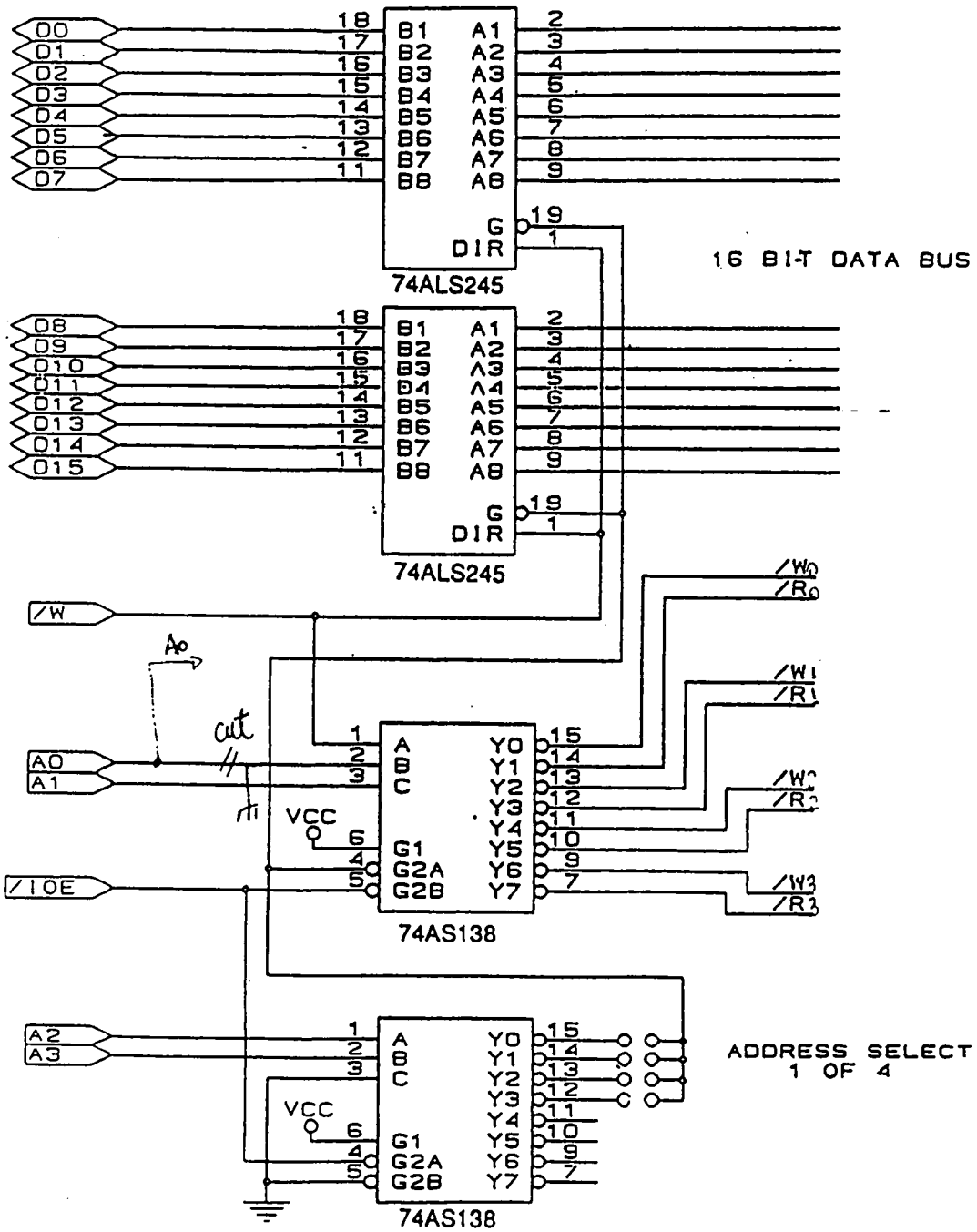


Figure 6.3 DSPLINK Interface

The Hanning PWM interface is shown in Figure 6.3. It has a 16 MHZ clock. The16-bit data, address, and control signals are connected to the DSPLINK interface. The interruption that drives the control software in the DSP is generated by the Hanning chip each 32 μs. This pulse is counted four times by the two flip-flops 74HC74 in order to generate the fastest interruption service routine of Task 1 (INT1). The monostable 74HC221 shapes the pulse into a 140 ns low level pulse to drive the interruption INT1 in the TMS320C30. A low level of the 140 ns pulse is required to not trigger multiple interruptions in all software required for the fast software tasks. The switching gate pulse outputs for the six devices (O1, U1, O2, U2, O3, and U3) are connected to the cable drivers DS8922A. This configuration was used because the interface from the Hanning board to the interface board in the inverter is physically distant. A direct connection from the Hanning output pins to the inverter would by very sensitive to noise due the high impedance in those lines. The cable driver DS8922A is an RS482 compatible driver, able to transmit the pulses from Hanning to the inverter interface with a low impedance and differential connection, where the logic level "1" is the flow of the current in the cable in one direction, whereas the logic level "0" is the flow of the current in the cable in the opposite direction.

6.1.2 Inverter Interface

The inverter interface, as shown in Figure 6.5, receives the gate drive signals through the cable driver DS8922A and sends them into the 74HC365 buffer which can be disabled in case of fault, such as over current or manually detected fault in the system. The monostable integrated circuit, 74HC221, is used for the power inverter which requires a longer pulse for disabling the gate drives. There are two control signals which disable the gate drive signals:

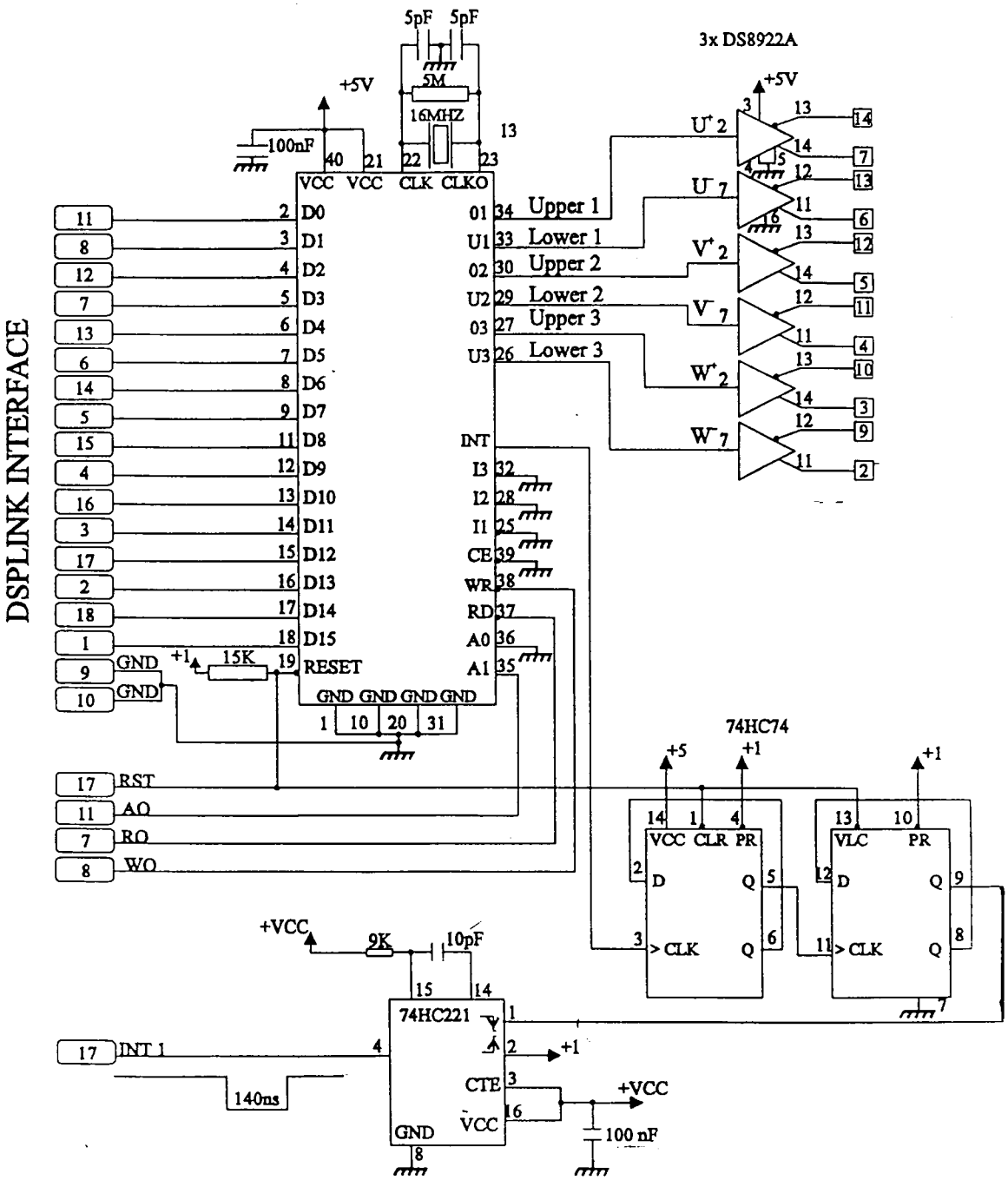


Figure 6.4 Hanning PWM interface with cable driver

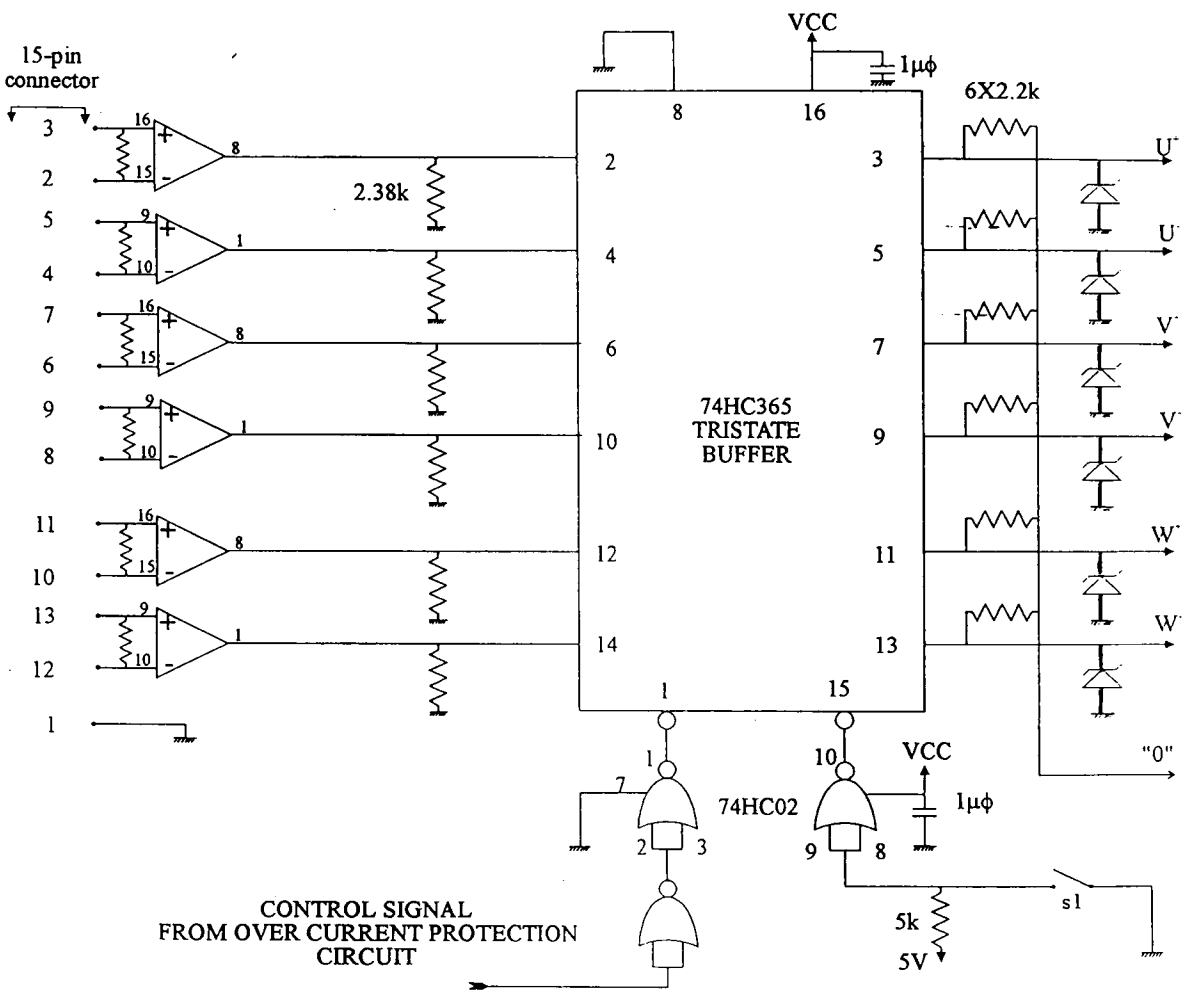


Figure 6.5 Inverter protection circuit

(1) manual turn off signal controlled by using an ON-OFF switch S1 and (2) an overcurrent detected by the dc link current signal processing circuit as shown in Figure 6.6. In the overcurrent signal processing circuit, a voltage comparator, LM 319, is used to compare the input voltage send by the current sensor installed on the dc link bus with the threshold value which is adjustable. The output of this circuit is then supplied to the FLIP-FLOP 7474. The control signal can be reset manually once the fault has been detected. This can be performed by pushing the toggle switch S2 as shown in Figure 6.6. When overcurrent is found in the dc link, the LED, shown in Figure 6.6, will turn ON and then it will turn OFF when the toggle switch is pushed manually. The output of the overcurrent signal processing circuit is then supplied to the tristate buffer through the NOR gates.

6.1.3 IGBT Inverter and Motor

The 100 KW IGBT inverter was built and tested at the Delphi Energy Systems power electronics laboratory. It has built in protection circuitry. The six incoming PWM signals are connected to the external connector on the inverter as shown in Figure 6.7. Figure 6.7 shows the top view of the inverter and the connections for the six PWM signals. A 300 v dc power supply was used to supply dc voltage to the inverter.

The induction motor used in the implementation was a Reliance 5 horsepower high efficiency standard industrial induction motor. For accurate estimation in the control, the machine parameters were measured by performing a no load test. Five thermistors were installed on the stator of the motor for measuring the stator temperature. The output of the five thermistors was averaged by a thermistor interface circuit described in section 6.4.4. Table 6.1 displays the measured motor quantities and the rated machine operating parameters.

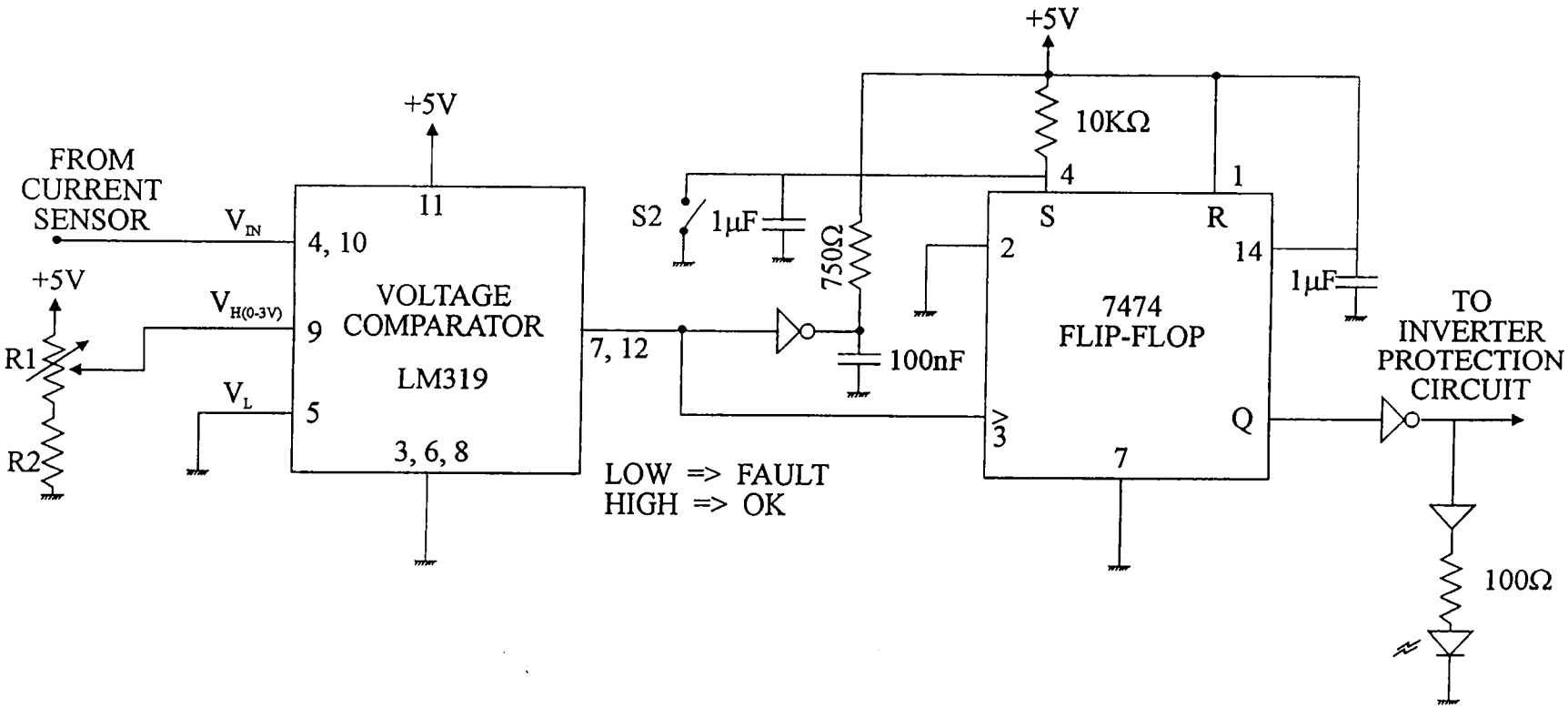


Figure 6.6 Overcurrent signal processing circuit.

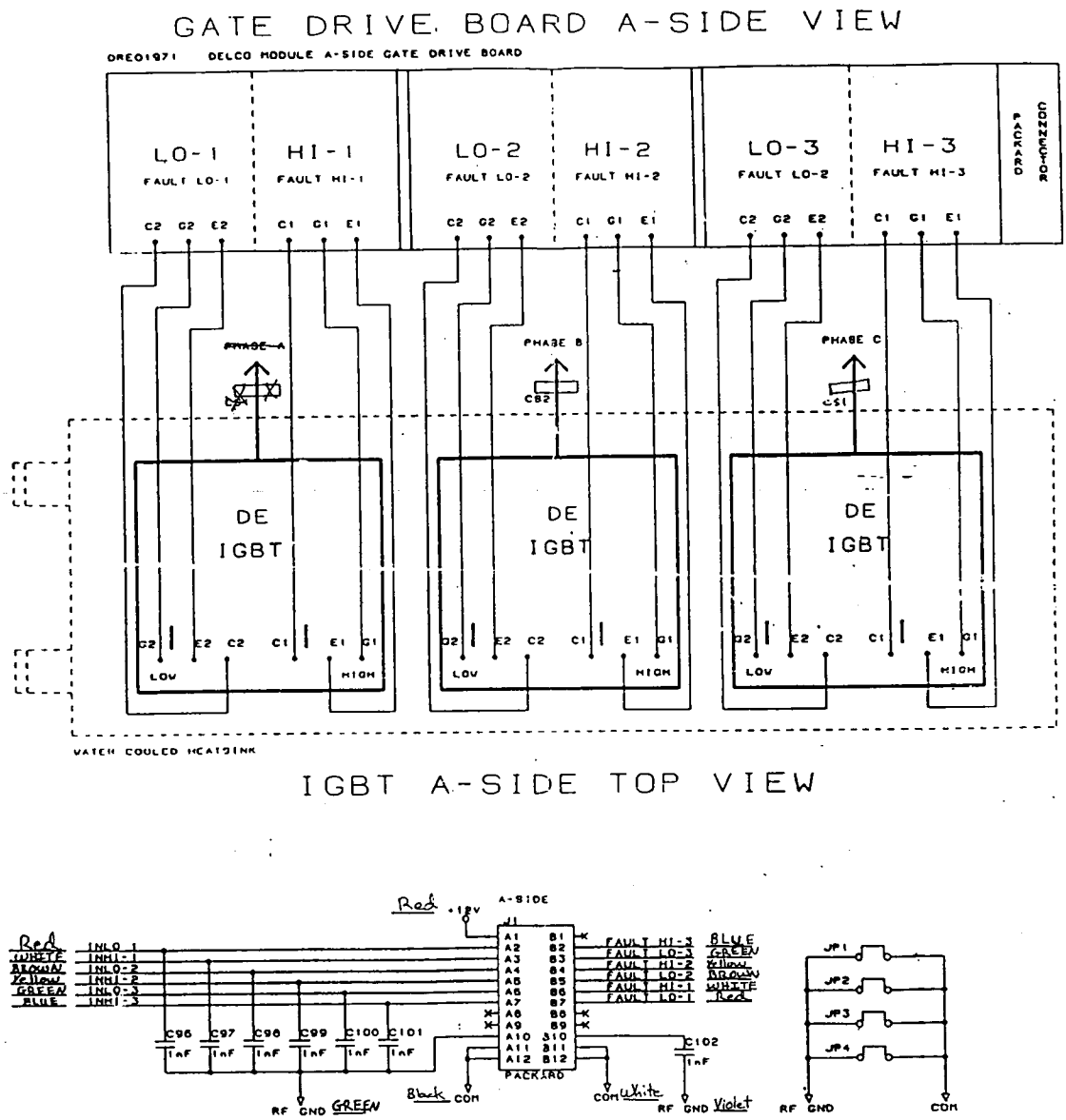


Figure 6.7 IGBT Inverter

Table 6.1 Induction machine parameters.

Parameter Name	Value
R_s	0.5814Ω
R_r	0.4265Ω
L_{ls}	3.479 mH
L_{lr}	4.15 mH
L_m	78.25 mH
i_{ds} rated	10 A
i_{qs} rated	15 A
ψ_s rated	0.45 webers
ω_{sl} rated	2.62 Hz
I_s rated	18 A

6.1.4 Peripheral Circuits

Several interface circuits are needed in the drive system. First, a feedback interface circuit transforms the terminal voltages and phase currents from three phases to two phases, filters out harmonics, and scales the signals for the analog to digital converters. Second, the dc link current sensing circuit senses the dc link current through the current sensor installed on the dc link bus, filters out the harmonics, and scales the signal for the analog to digital converters. Third, the dc link voltage signal processing circuit senses the dc link voltage from the dc link bus, filters out the harmonics, and scales the signal for the analog to digital converters. Fourth, a temperature interface circuit reads the thermistor values, averages the five temperatures, and scales the averaged signal for the analog to digital converter. Lastly,

a power supply circuit is used to supply the 300 v supply to the inverter

The feedback interface circuit first isolates and attenuates the line to line terminal voltages by using isolation amplifiers AD202JN from Analog Devices. The line voltages are then converted to phase voltages in the stationary D-Q reference frame by using equations 6.1 and 6.2. To eliminate the rich harmonics in the phase voltages, a two stage

$$V_{qs}^{s} = \frac{1}{3}(2V_{ac} + V_{cb}) \quad \text{..... (6.1)}$$

$$V_{ds}^{s} = \frac{1}{\sqrt{3}}(V_{cb}) \quad \text{..... (6.2)}$$

filtering scheme is used. The first filter is a passive RC network with cutoff frequency of 2 KHZ while the second is an active filter with cutoff frequency of 1 KHZ. The phase voltages are scaled to +/- 2.5 volts for input into the 12-bit analog to digital converters.

The phase currents were sensed by NANA current sensors on the phases I_a and I_b. By using equations 6.3 and 6.4 the phase currents are transformed into stationary D-Q

$$I_{qs}^{s} = I_{a} \quad \text{..... (6.3)}$$

$$I_{ds}^{s} = -(\frac{2}{\sqrt{3}} i_b + \frac{1}{\sqrt{3}} i_a) \quad \text{..... (6.4)}$$

reference frame quantities. The phase currents are filtered at the same cut off frequency of 1 KHZ because the resultant phase shift produced by the filters needs to be the same for both for the voltages and the currents. The current signals do not require the heavy filtering but if the phase shifts in voltages and currents are not the same an error will be introduced when the flux is estimated. The phase currents are scaled to +/- 2.5 volts for input to the 12-bit analog to digital converters.

A voltage clamp circuit is connected to each output signal of the feedback interface circuit because the analog to digital converter can be damaged if a voltage larger than +/- 2.5 volt is applied. An offset potentiometer is provided for each output signal to zero out any dc component in the signals. Figure 6.8 shows the schematic diagram for the feedback interface circuit.

The dc link current and voltage sensing circuits are shown in Figures 6.9 and 6.10. Adequate dc link current I_d and voltage V_d sensing is crucial for dc link power P computation, and consequently, for the correct operation of the efficiency controller. DC link current sensing is performed by a high precision NANA current sensor (HEC-04A), as indicated in Figure 6.9. The sensor is placed between rectifier output and the dc capacitor bank. This location ensures that the PWM current harmonics generated by the inverter operation are filtered out by the dc link capacitor. To ensure a better utilization of A/D dynamic range, a low-pass filter with feedback gain is used. The reduction in dc link

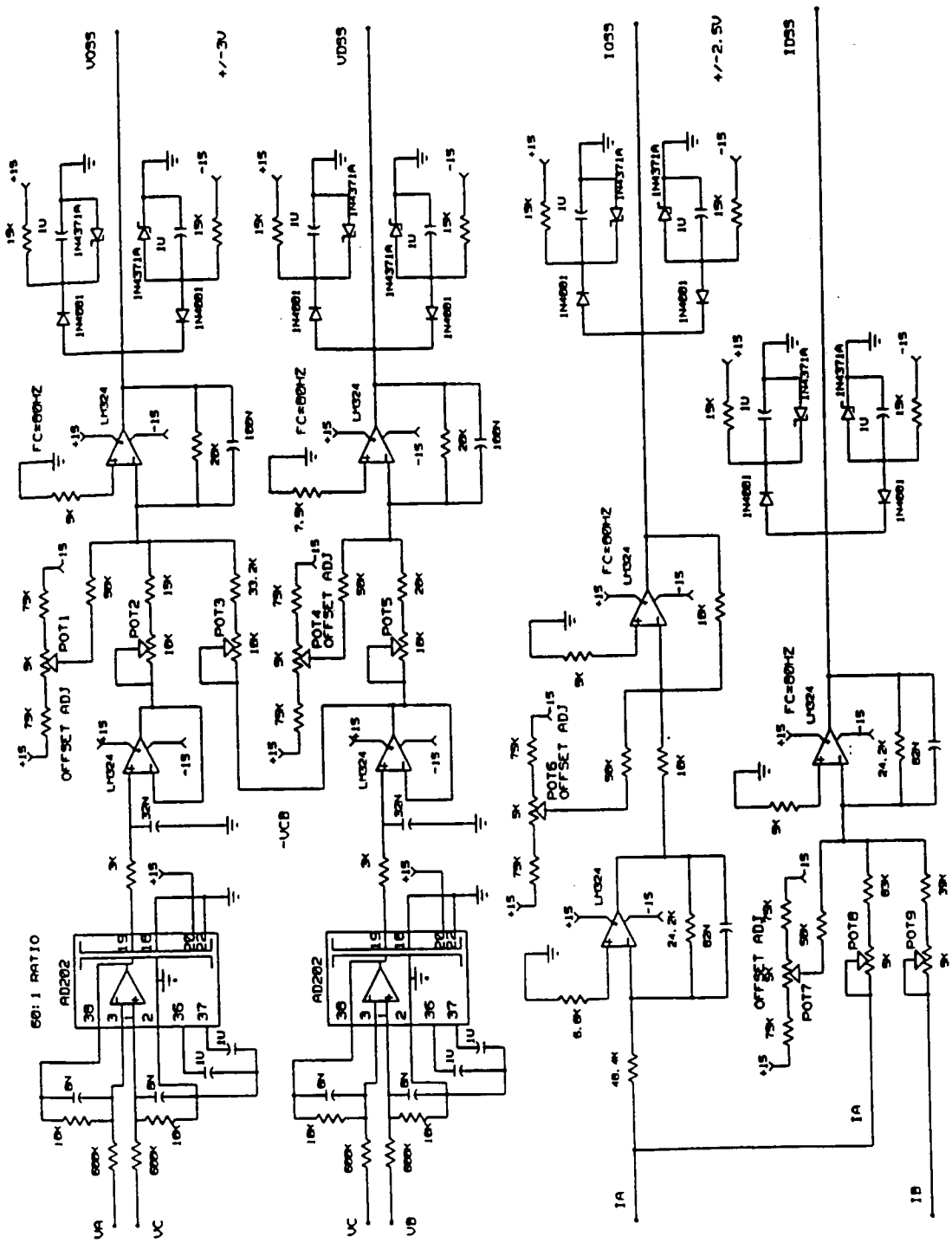


Figure 6.8 Feedback interface circuit

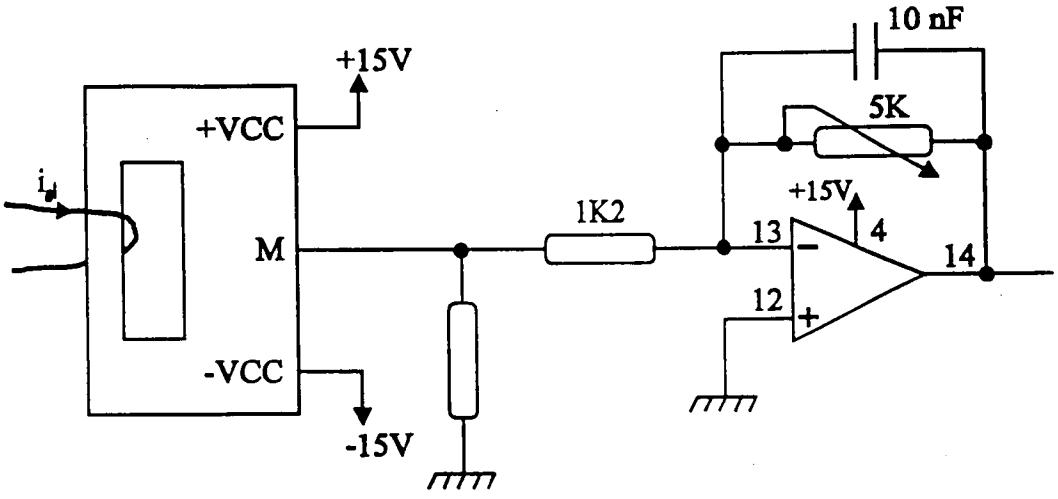


Figure 6.9 DC link current sensor interface.

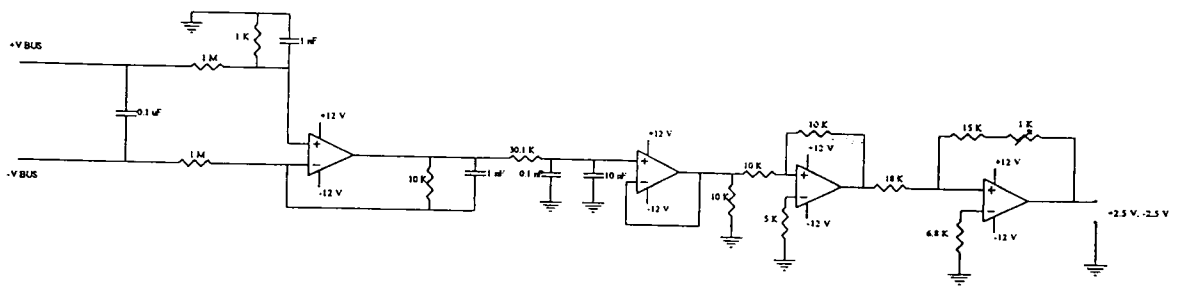


Figure 6.10 DC link voltage sensing interface

current bandwidth has no detrimental impact on P_d computation since only its average value is relevant.

The DC link voltage sensing circuit is built using a high slew rate, wide bandwidth, dual supply operational amplifier MC34074. The input of the voltage sensing circuit is connected across the power resistor connected in the dc link bus. The output signal is then scaled between +/-2.5 volts for analog to digital converter.

The last interface circuit is used to read, average, and scale the stator temperature signal sent to the DSP. Five thermistors are used to sense the temperature of the stator. As the stator temperature increases the thermistor resistance increases causing the temperature interface circuit output voltage to decrease. The five output voltages are averaged and scaled by an operational amplifier. Figure 6.11 shows the schematic circuit diagram of the temperature interface circuit. From the diagram in Figure 6.11, the voltage V_a can be expressed by equation 6.5 and the thermistor resistance value R_{ta} can be determined by equation 6.6. Substituting equation 6.6 into equation 6.5 yields the circuit output voltage v_a as a function of stator temperature. The stator temperature can be expressed by equation 6.7. This stator temperature information was used to design and fine tune the quasi-fuzzy R_s estimator. Once the measured and the estimated stator temperature were matching at different operating points, the stator resistance was derived from the equation 6.8, where α is the temperature coefficient of copper and R_{so} is the nominal stator resistance at 25°C.

$$V_a = 5\frac{R_{ta}}{10} + R_{ta} \quad \text{..... (6.5)}$$

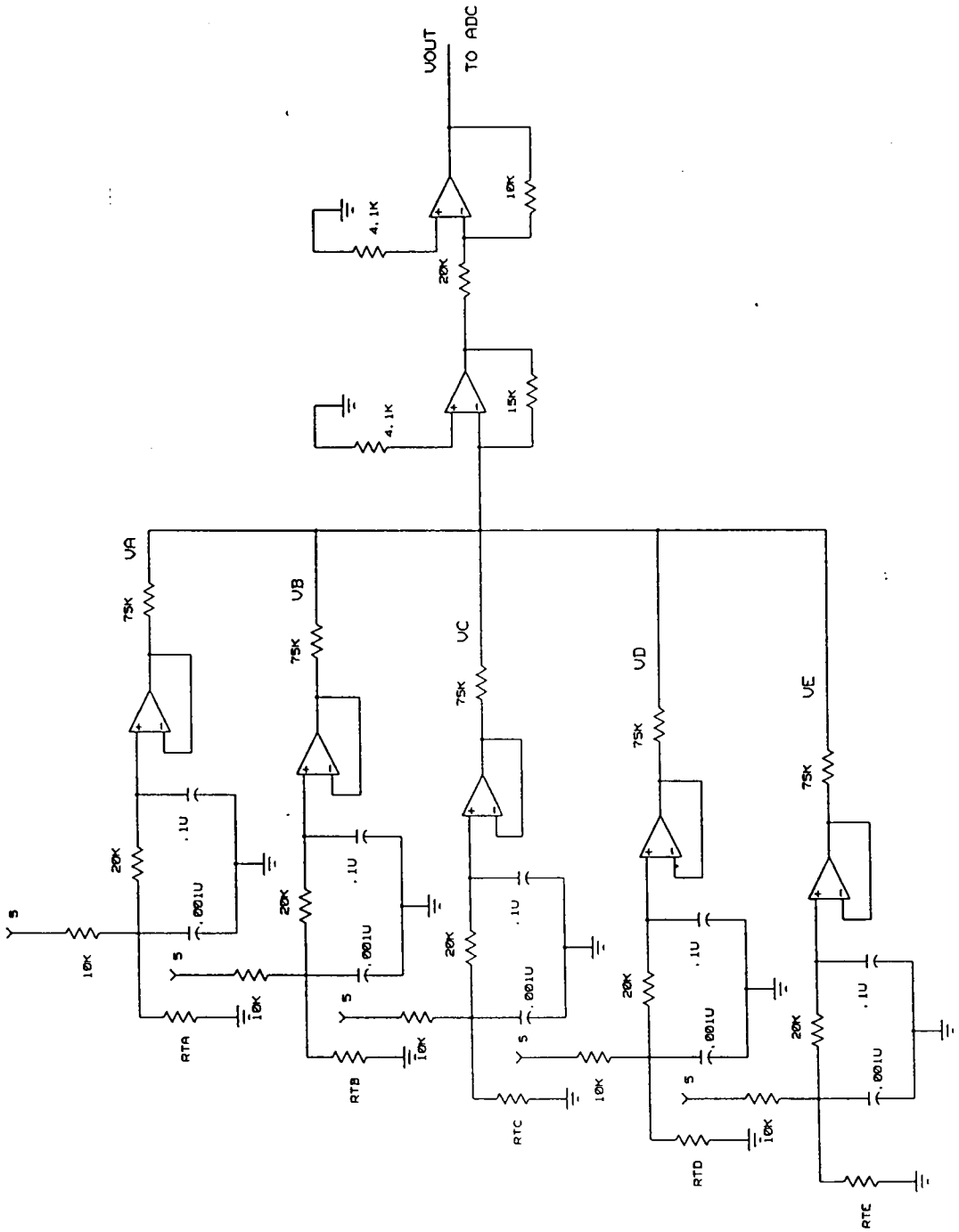


Figure 6.11 Stator temperature interface circuit

$$R_{ta} = 10^5 \exp[3960(\frac{1}{273.15 + T_s} - 3.354x10^{-3})] \quad \text{..... (6.6)}$$

$$T_s = \frac{1}{3.354e-3 + \frac{1}{3960}\ln(\frac{0.1 * V_a}{5 - va})} - 273.15 \quad \text{..... (6.7)}$$

$$R_s = R_{so} + \alpha R_{so}[T_s - 25^0 c] \quad \text{..... (6.8)}$$

The 300 v power supply circuit schematic is shown in Figure 6.12. A three phase bridge diode rectifier (98F6716) is used to design the power supply circuit. As shown in the Figure, a variac is used at the input of the power supply circuit to control the input voltage of the power supply circuit. A ON-OFF switch (S1) is connected in parallel to the 20 kΩ power resistor to by pass the resistor once the capacitor is charged. Initially, this circuit was used to supply a 300 v dc voltage to the IGBT inverter then it was replaced by a regulated 300 v power supply for the inverter protection purpose.

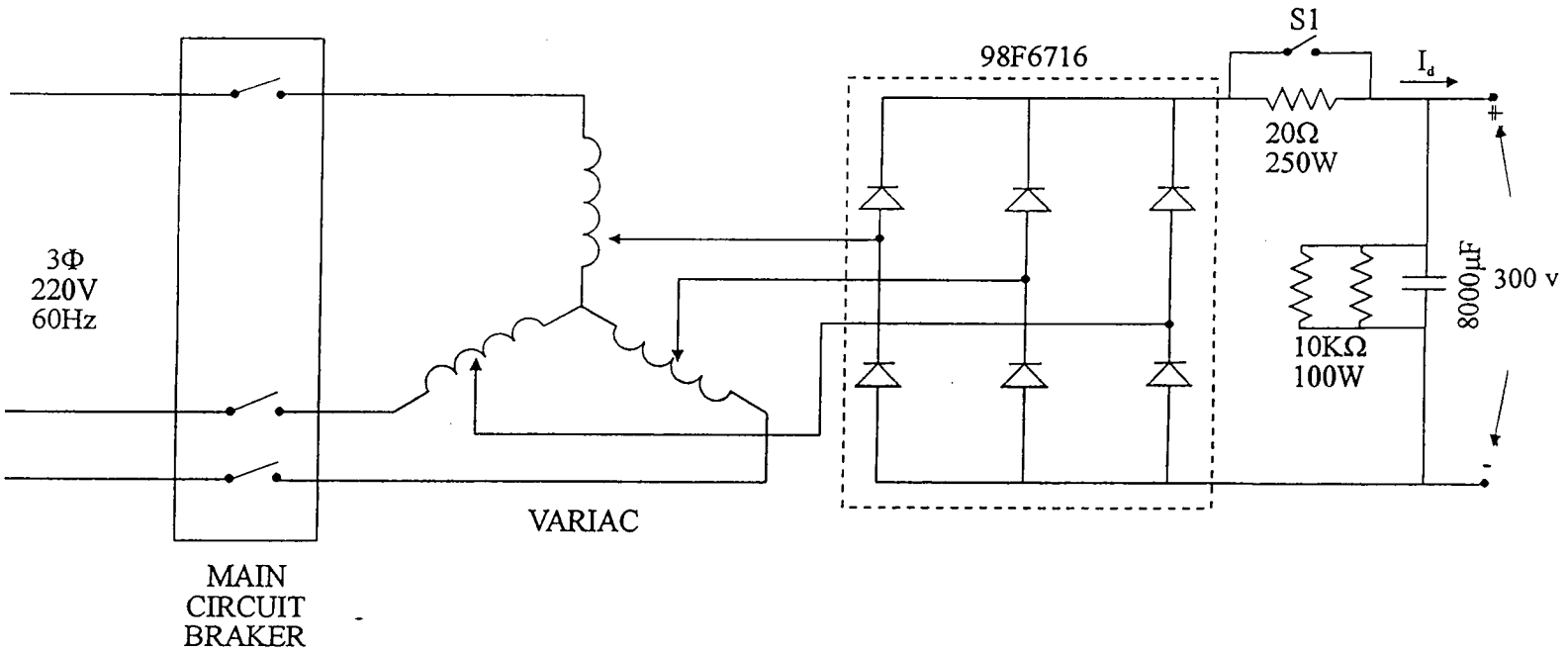


Figure 6.12 A 300 v power supply circuit

6.2 Experimental Evaluation

After careful software and hardware systems integration and debug, the system was tested with a Reliance 5 hp standard industrial induction motor, rated at 1750 rpm, 230 V, whose parameters are given in Table 6.1. Depending on the particular equipment, the results were acquired in several ways. Some results were taken via a GPIB oscilloscope. The internal software variables were sent to the DAC for visualization in the oscilloscope and photographs were taken of the scope screen. Transient responses were captured in the oscilloscope and then sent to the printer connected to the oscilloscope. In order to prepare the Figures for the steady state conditions, the required variables were observed and noted in a laboratory notebook.

The drive system was tested and systematically integrated step-by-step. In the beginning, the DSP Processors were tested and the analog IO boards were checked, along with the current and voltage interface circuits. The Hanning interface board was tested and the main interruption from Hanning chip was connected to each of the DSP boards. The Hanning PWM IC signals were evaluated by sending the PWM signals through a digital circuit that simulated the operation of a three -phase inverter. With a low-pass filter connected to the output of the digital circuit, the fundamental waveform was observed. Figure 6.13 shows three outputs of the Hanning chip at a constant frequency $f_e = 50$ Hz. Also shown is the third harmonic that the Hanning chip intentionally adds to the fundamental. The third harmonic helps to improve the PWM efficiency and does not circulate in the machine phase windings because the neutral is isolated. Figure 6.14 shows the one output of the Hanning chip supplied to the digital circuit with the RC load at the output, with the frequency

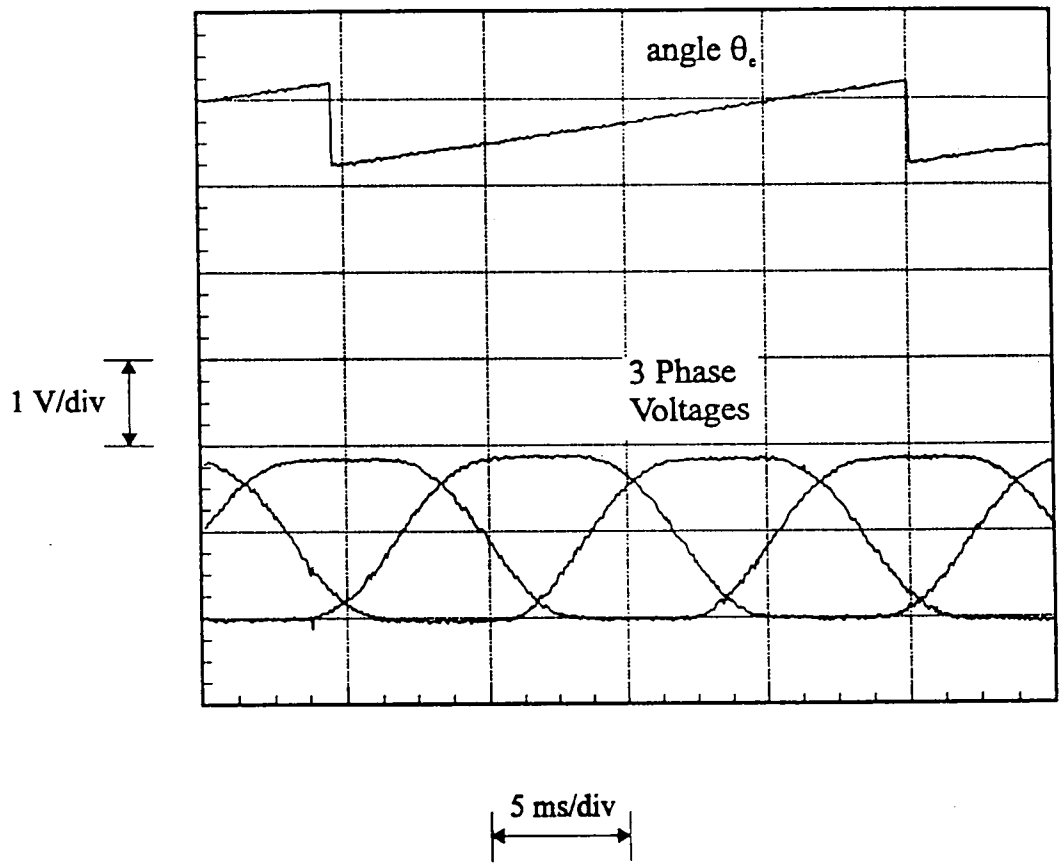


Figure 6.13 Evaluation of Hanning PWM signals with low-pass filter at frequency of 50 Hz

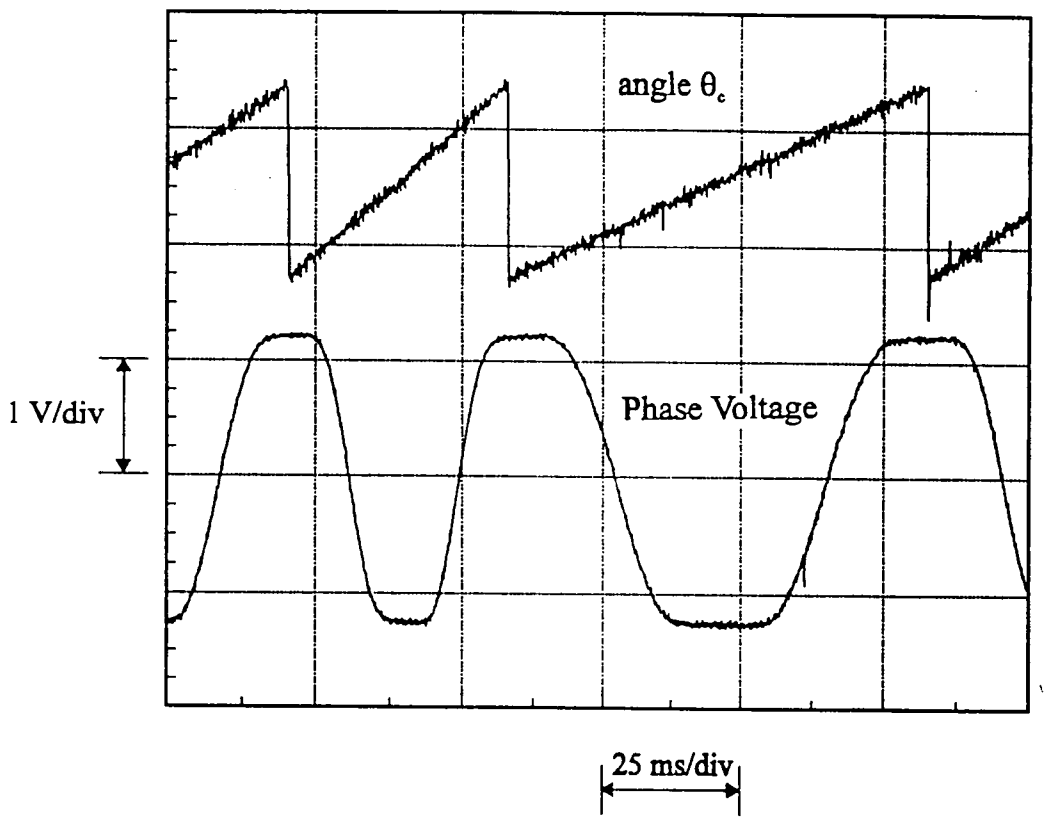


Figure 6.14 Evaluation of Hanning PWM signals with low-pass filter and stepped down frequency form 25 Hz to 12.5 Hz

being stepped down from 25 Hz to 12.5 Hz. Such tests were convenient not only to test the Hanning chip, but also to become familiar with the interruption service routine software and to implement a host communication program that could control the operation of the Hanning PWM. After checking the operation of the PWM Hanning circuit, the cable driver interface and inverter protection circuits were tested. Then, the Hanning chip output was connected to the IGBT inverter through the protection circuit.

The drive was initially started in volts/hertz open loop control mode to evaluate and debug the feedback signals used in the drive control system. The drive was tested over the complete operating range starting from 0 rpm to 3500 rpm and all of the feed back signals were checked and debugged in this range. The drive was run at 20 Hz frequency and data were collected. Figure 6.15 shows the sine wave and the reference angle θ_e for vector rotation. Figure 6.16 shows the phase currents i_a, i_b and i_c. Figure 6.17 shows the estimated unit vectors sin θ_e and cos θ_e which are 90^0 phase shifted from each other. Since the drive can not be started in the direct vector control mode, the motor was started using the V/Hz control and then the drive was transitioned to the direct vector control mode and then the synchronous current control loops, flux loop and torque loop were closed sequentially.

The drive was started in V/Hz mode with inertial load and the machine speed was increased to 600 rpm.The dynamometer, which was set in speed control mode, was preset to 600 rpm. Then, the reference value for the PI controllers, used in the synchronous current control loop, was set. Also, the PI controllers proportional and integral gains were set to minimum values. PI controllers gains were set to the minimum value initially because if the gains are too high the system will go to instability when it is closed. Then, the synchronous

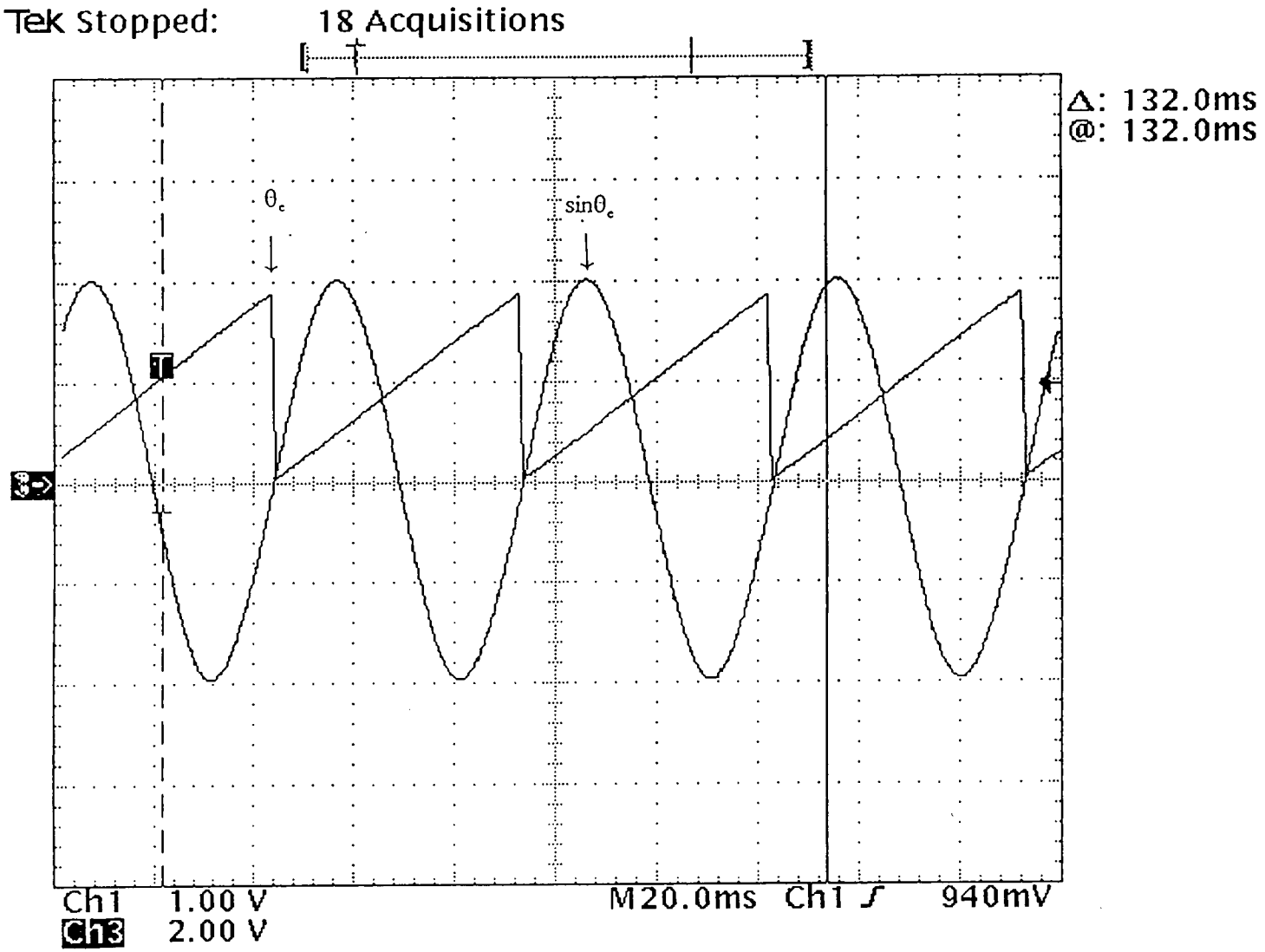


Figure 6.15 Estimated $\sin\theta_e$ and θ_e signals in V/Hz control at 20 Hz.

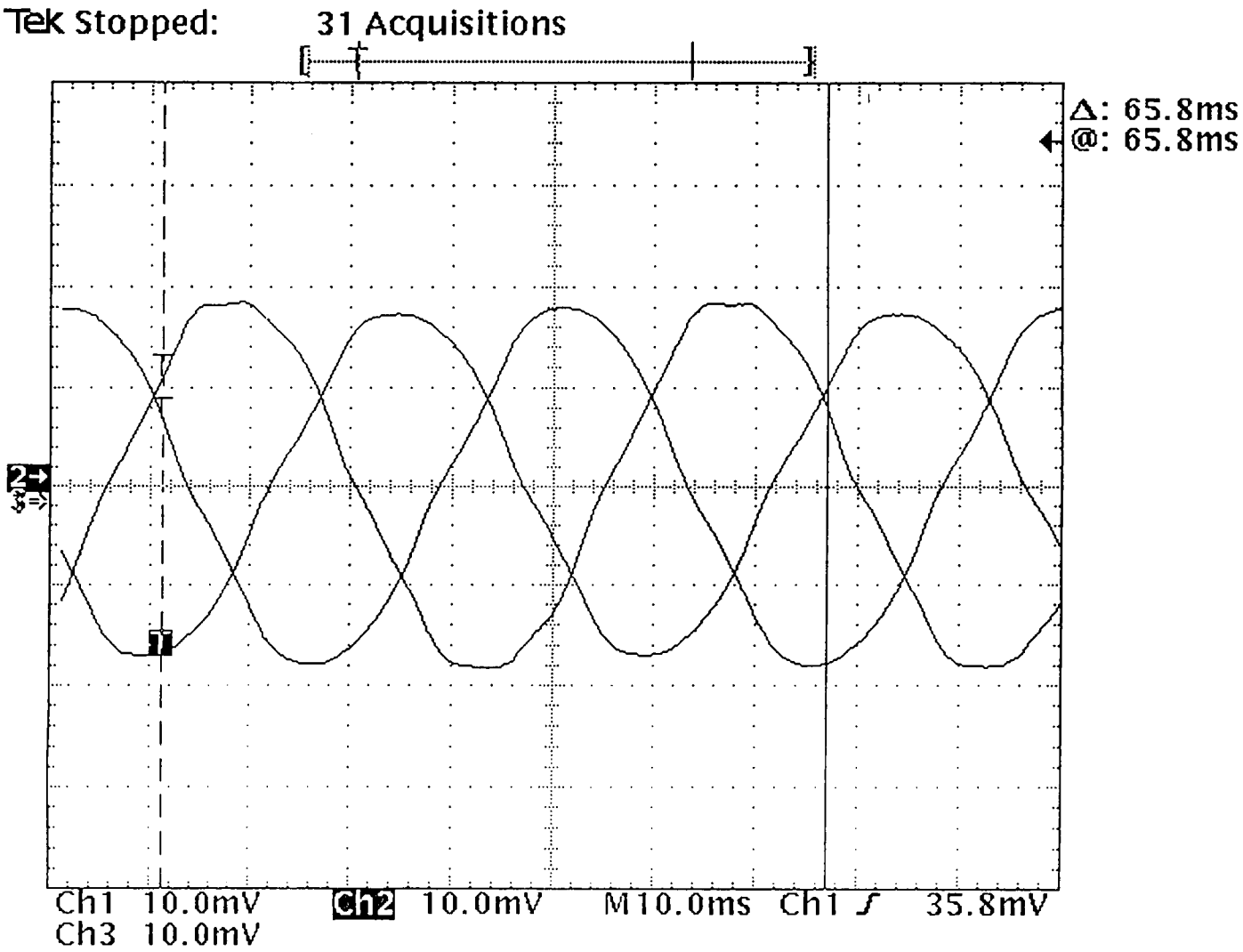


Figure 6.16 Phase currents i_a, i_b and i_c (2 A/div) in V/Hz control at 20 Hz

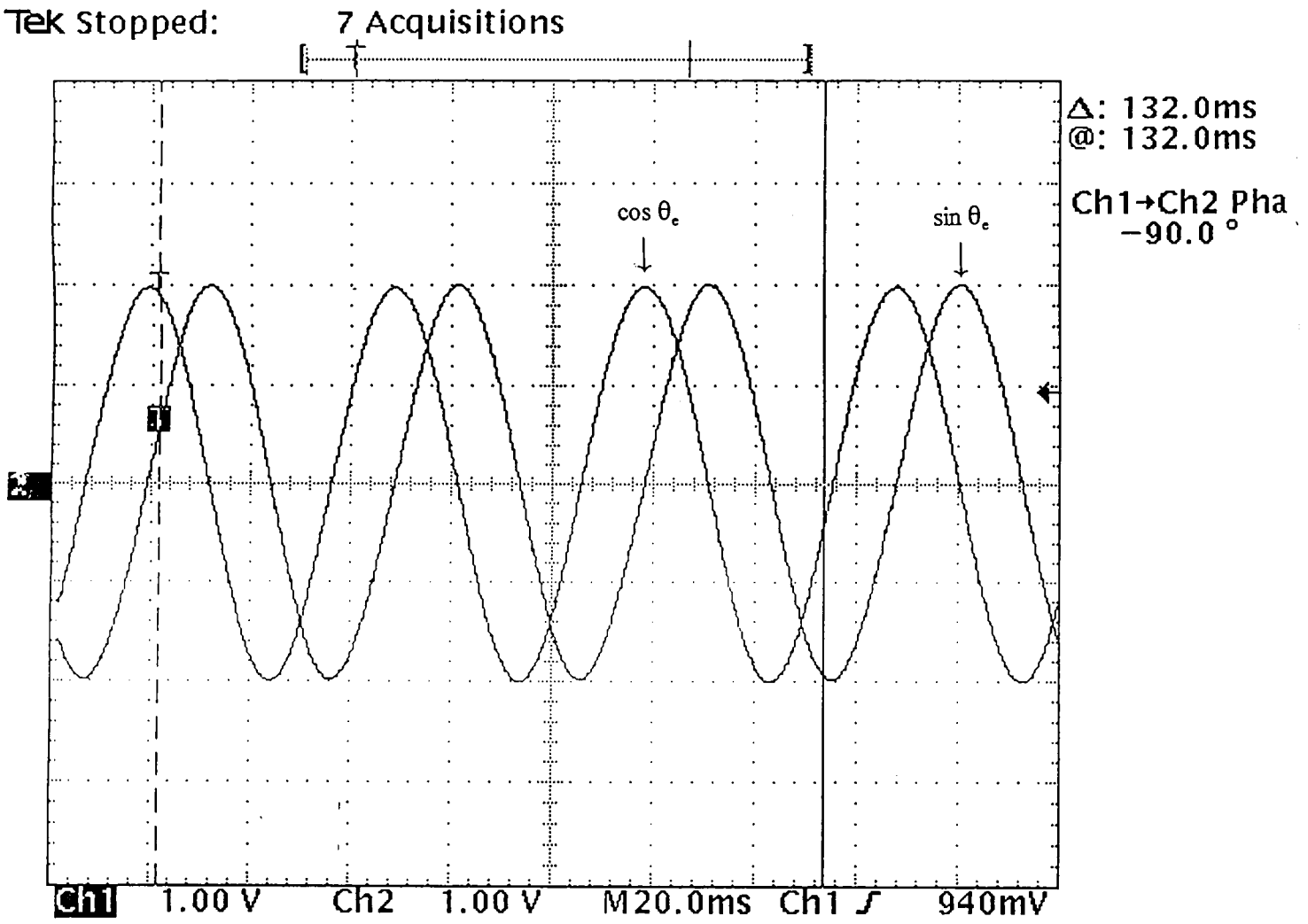


Figure 6.17 Unit vectors $\sin\theta_e$ and $\cos\theta_e$ in V/Hz at 20 Hz

current loops were closed while the proportional and integral gains of the PI controllers were fine tuned. Once the gains were fine tuned, the stator flux, ψ_s, loop was ready to be closed. The reference value of the stator flux (ψ_s^*) was set and the PI controller's proportional and integral gains were set to minimum values. Finally, the system was closed and the gains were fine tuned until the transient response of the flux loop was acceptable. Figure 6.18 shows the transient response of the flux (ψ_s) control loop along with the estimated magnetizing current i_{ds}. At that point, the torque (T_e) control loop was ready to be closed. The reference torque (T_e^*) value and PI controller's gains were set. Then, the torque control loop was closed while the PI controller's gains were fine tuned until the transient response of the torque was acceptable. The system operation was checked at different speed conditions especially near zero speed because DVC operation near zero speed is very important. The operation was found to be excellent over the entire operating range. The drive was run in DVC control mode with the torque and the flux loop closed by applying a reference torque (T_e^*) of 10 Nm (50% of the rated torque) and a reference stator flux (ψ_s^*) of 0.45 webers (rated flux). The important steady state results were stored for this operating point and are shown on pages 134 to 144. Figure 6.19 shows the input, q-axes voltage ($v_{qs}^{s'}$), to the cascaded low-pass filter and the q-axes flux ($\psi_{qs}^{s'}$) after phase shift compensation. Figure 6.20 shows the input, d-axes voltage ($v_{ds}^{s'}$), to the cascaded low-pass filter and the d-axes flux ($\psi_{ds}^{s'}$) after phase shift compensation. As described in Figure 3.16 in section 3.2.1, Figure 6.19 and 6.20 show the correct operation of the cascaded low-pass filter by displaying the 90^0 phase shift between the two waveforms in each Figure. Figure 6.21 shows the d-axes and q-axes stationary reference frame stator fluxes ψ_{ds}^{s} and ψ_{qs}^{s}, respectively. Figure 6.22 shows the estimated

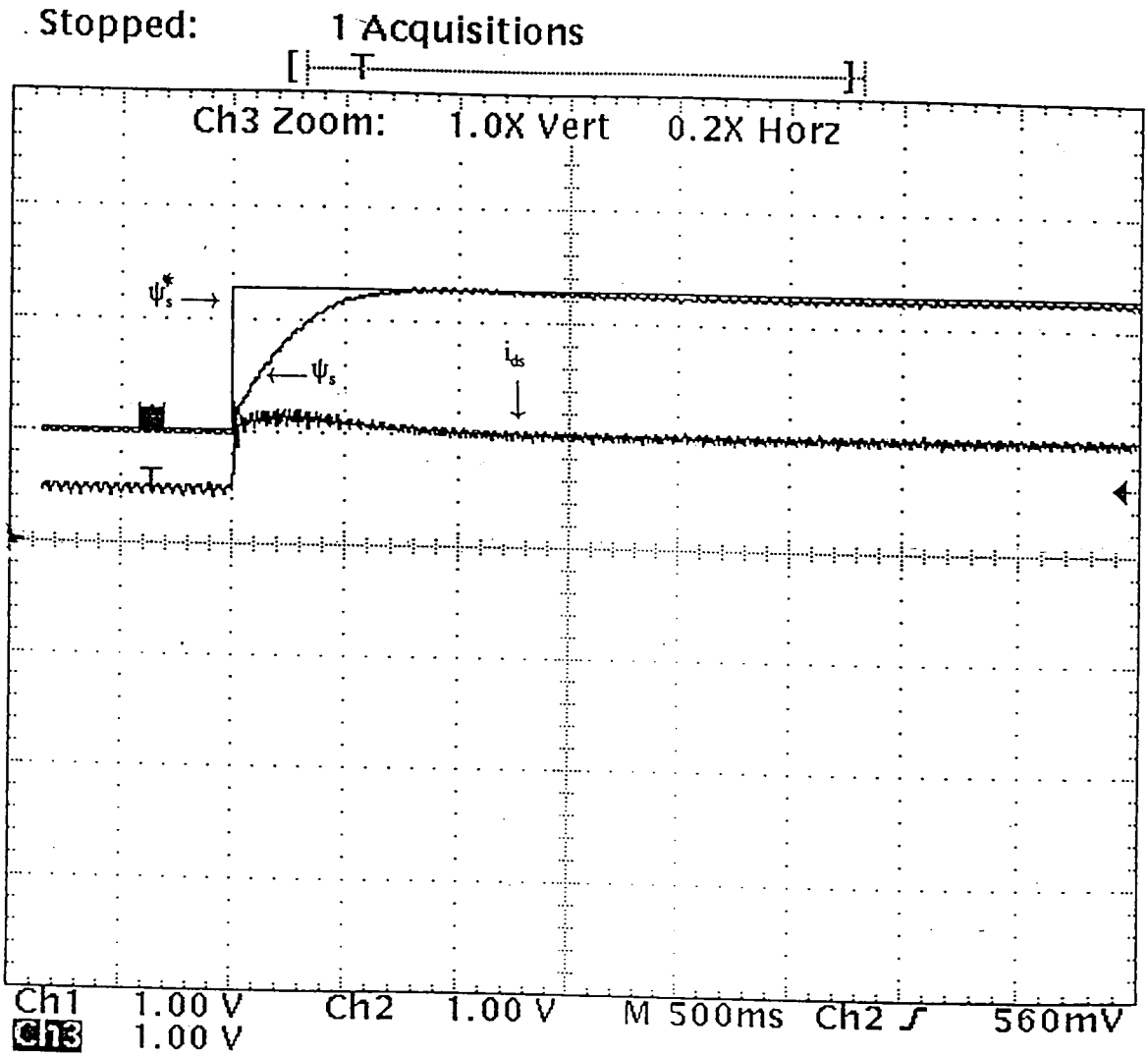


Figure 6.18 Step response of ψ_s to 100% rated flux (0.2 webers/div)

along with i_{ds} (4 A/div)

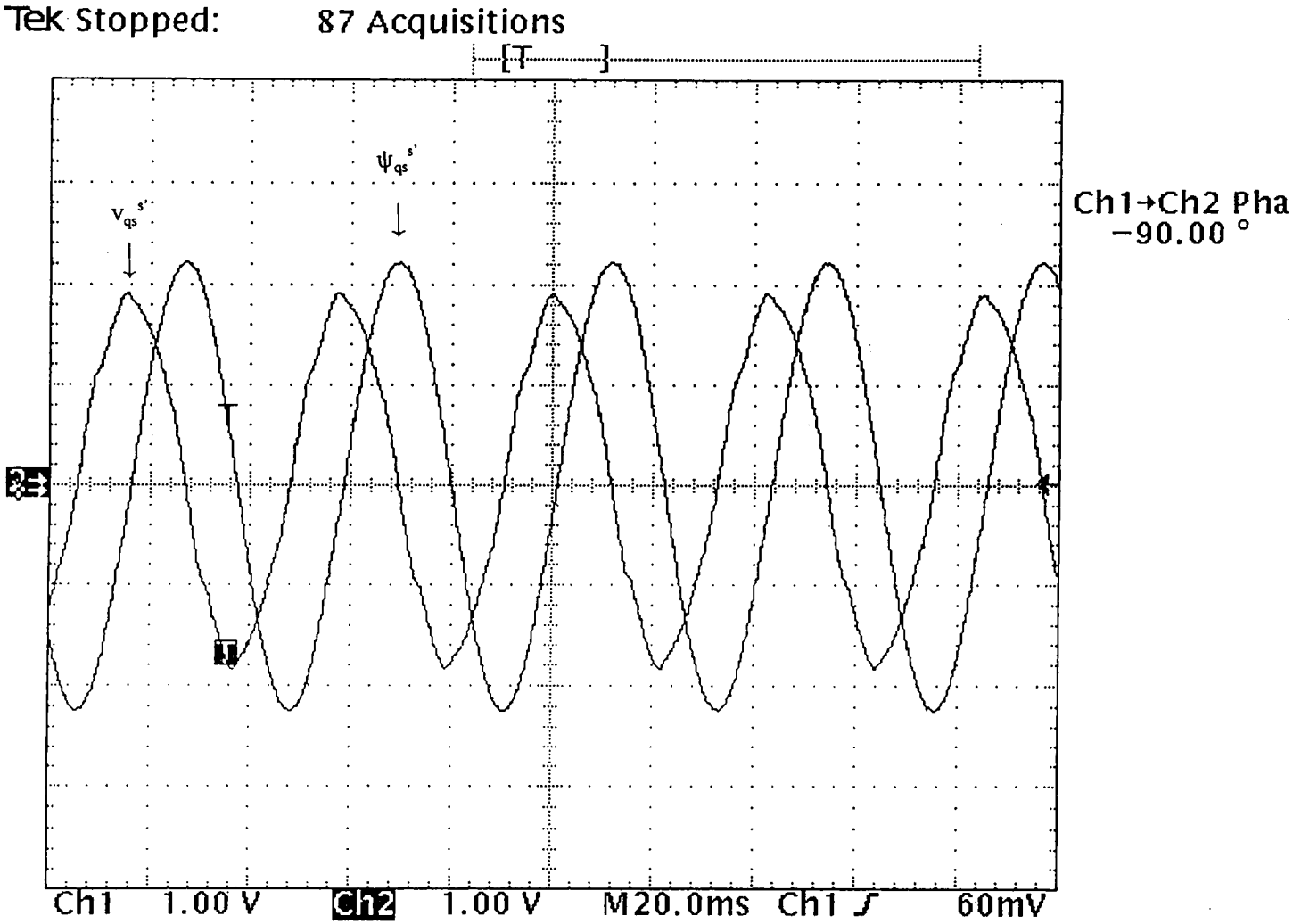


Figure 6.19 Input of cascaded low-pass filter $v_{qs}^{s'}$ (39 v/div) and q-axis stator flux $\psi_{qs}^{s'}$ (0.18 webers/div) after phase shift compensation in DVC

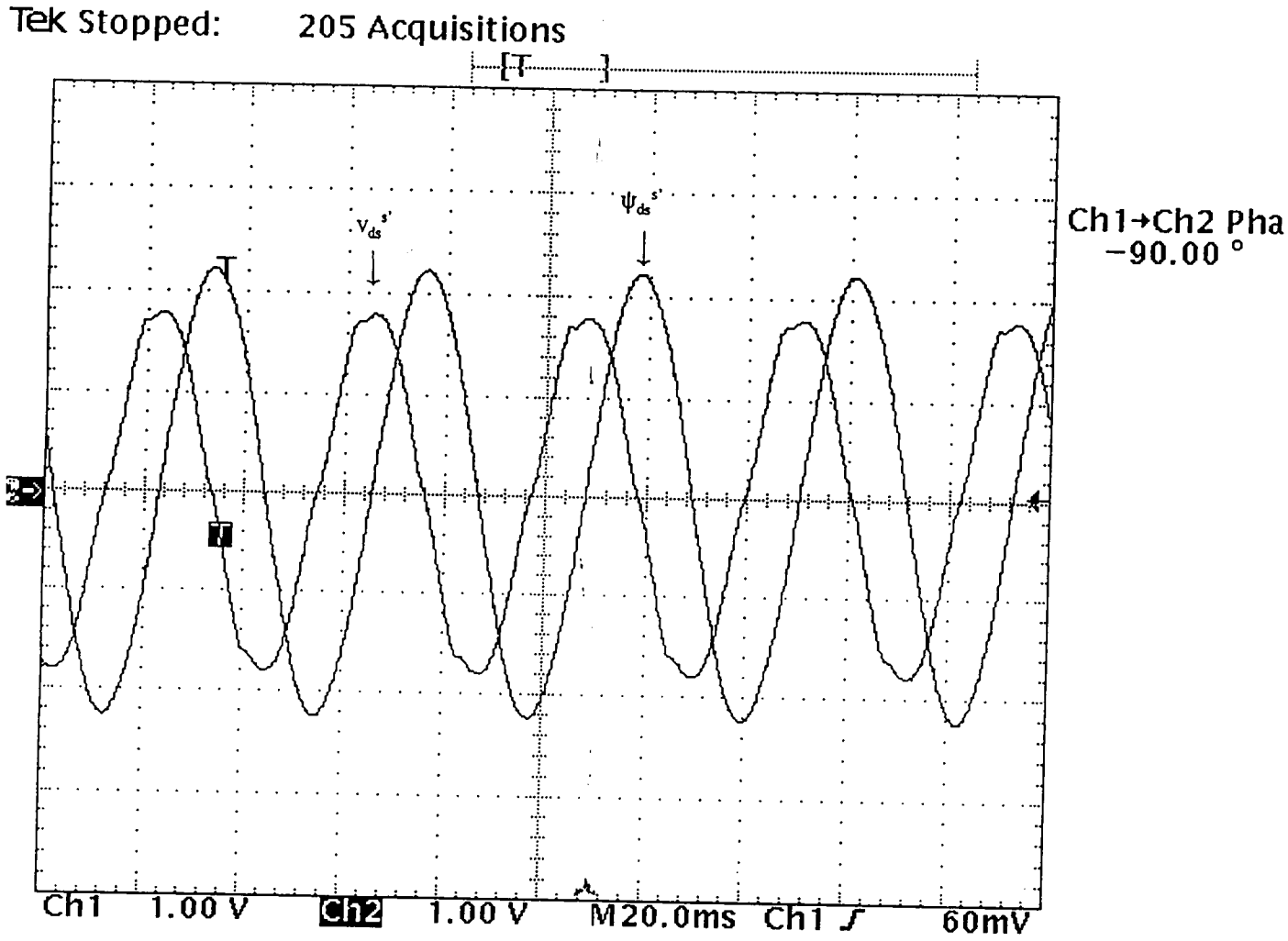


Figure 6.20 Input of cascaded low-pass filter $v_{ds}^{s'}$ (39 v/div) and q-axis stator flux $\psi_{ds}^{s'}$ (0.18 webers/div) after phase shift compensation in DVC

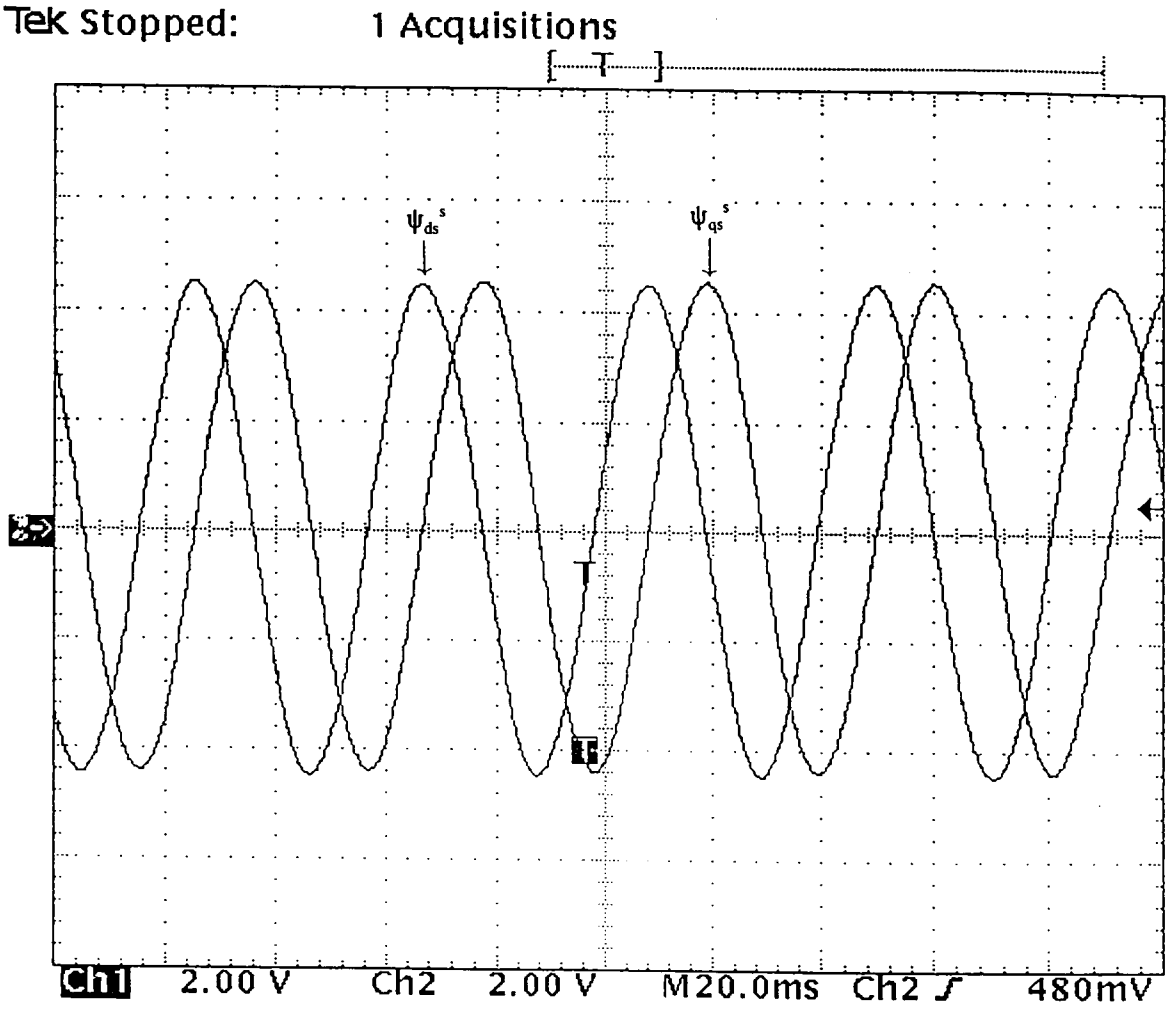


Figure 6.21 D-Q axes stationary reference frame fluxes ψ_{ds}^{s} and ψ_{qs}^{s} (0.18 webers/div) in DVC

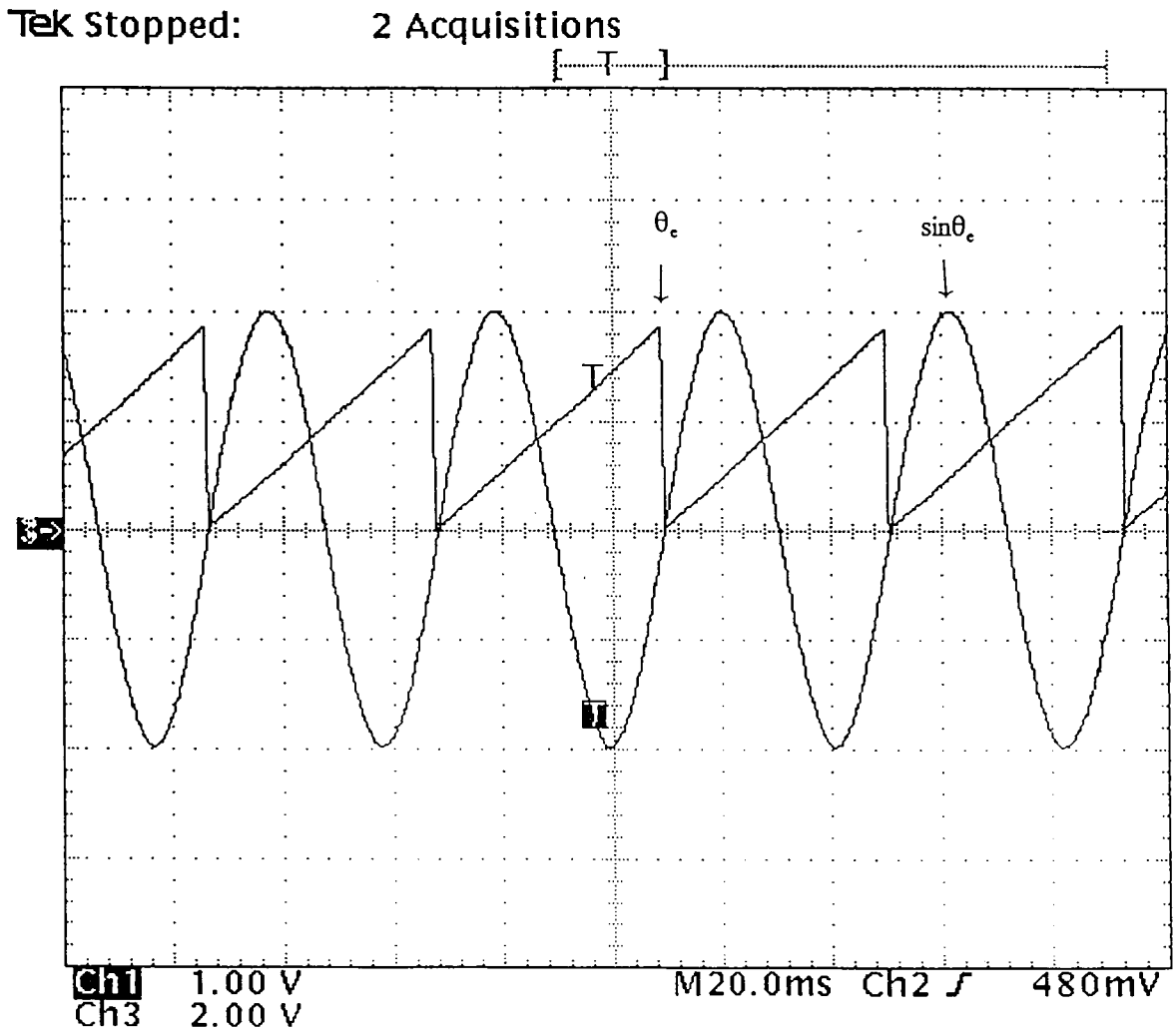


Figure 6.22 Estimated $\sin\theta_e$ and θ_e in DVC

$\sin\theta_e$ and θ_e. Figure 6.23 shows the unit vectors $\cos\theta_e$ and $\sin\theta_e$ generation in DVC control mode. Figure 6.24 shows estimated i_{qs} and i_{ds}. Figure 6.25 shows estimated torque (T_e), magnitude of stator flux (ψ_s) and decoupling current (i_{dq}). Figure 6.26 shows estimated angular frequency (ω_e). Figure 6.27 shows machine phase currents i_a, i_b and i_c.

The transient response of the torque along with the estimated magnitude of stator flux are shown in Figure 6.28. As shown in the Figure, the magnitude of stator flux was unchanged while the step command was given to the torque control loop from 6 Nm to 10 Nm that proves the system was completely decoupled. While the torque increases or decreases, the decoupling current i_{dq} is injected proportionally to the d-axes control loop to decouple the system. This is illustrated in Figure 6.29 wherein the reference torque was stepped-up from 1 Nm to 5 Nm, the magnitude of stator flux remains unchanged and the decoupling current increases.

The stator temperature and the corresponding stator resistance were estimated and fine tuned extensively under steady and dynamic conditions. The quasi-fuzzy stator resistance estimation accuracy was found to be very good. Figure 6.30 shows the temperature estimation accuracy at different currents and a constant speed of 357 rpm and Figure 6.31 shows the corresponding resistance estimation accuracy. Figures 6.32, 6.33, 6.34 and 6.35 show the results of quasi-fuzzy stator resistance estimation at speeds of 725 rpm and 918 rpm, respectively.

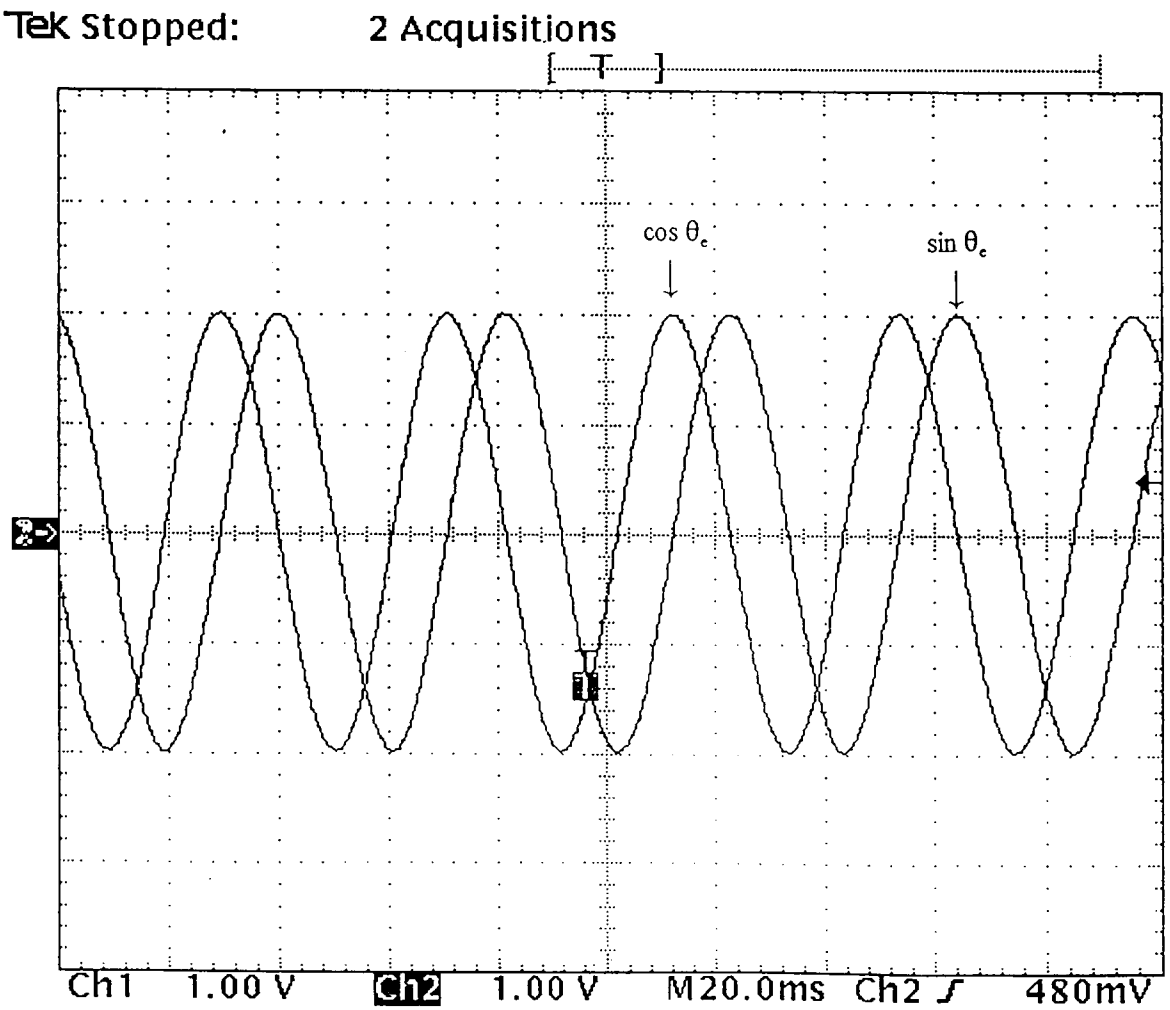


Figure 6.23 Unit vectors $\sin\theta_e$ and $\cos\theta_e$ in DVC at 23.4 Hz

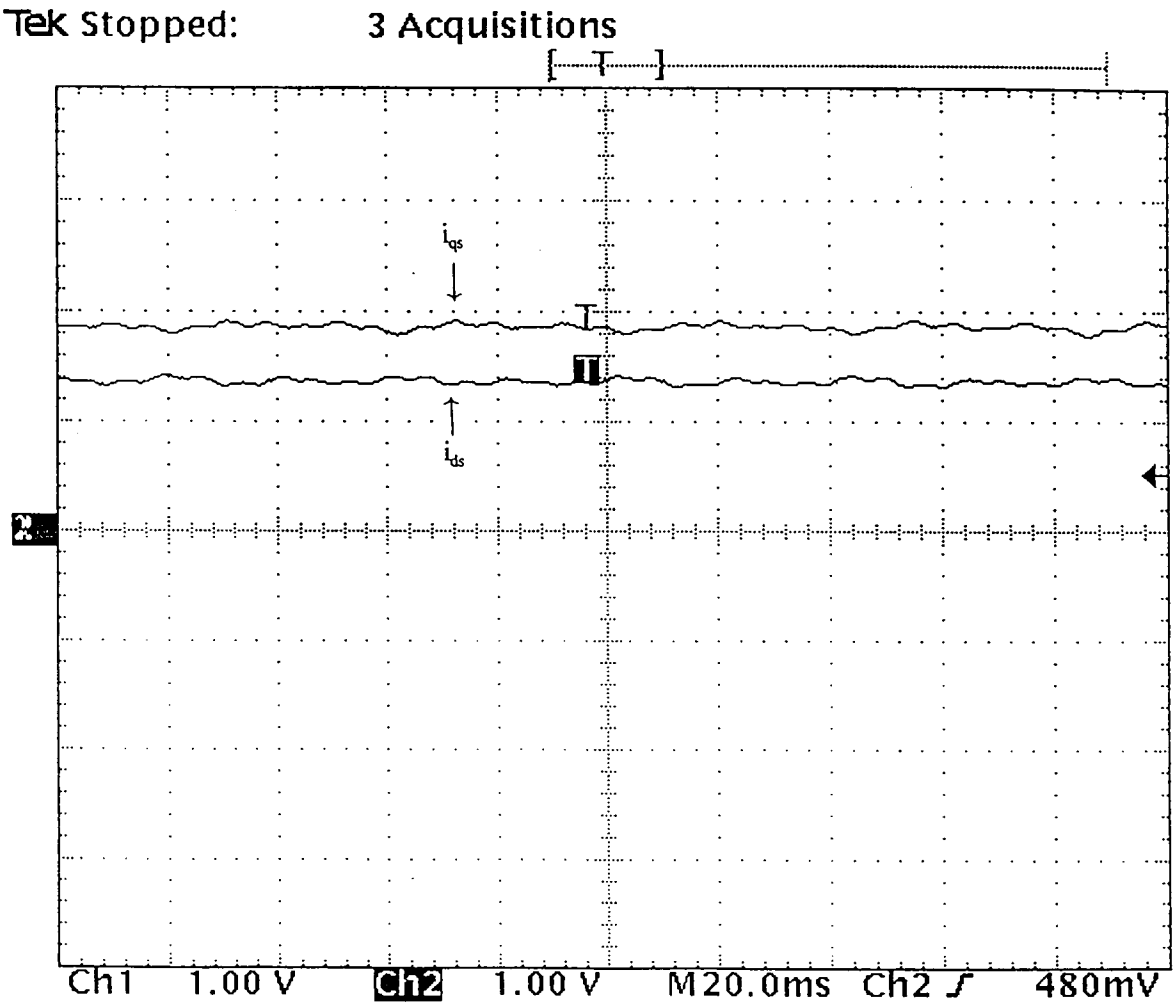


Figure 6.24 Estimated i_{qs} (4.5 A/div) and i_{ds} (4 A/div) in DVC

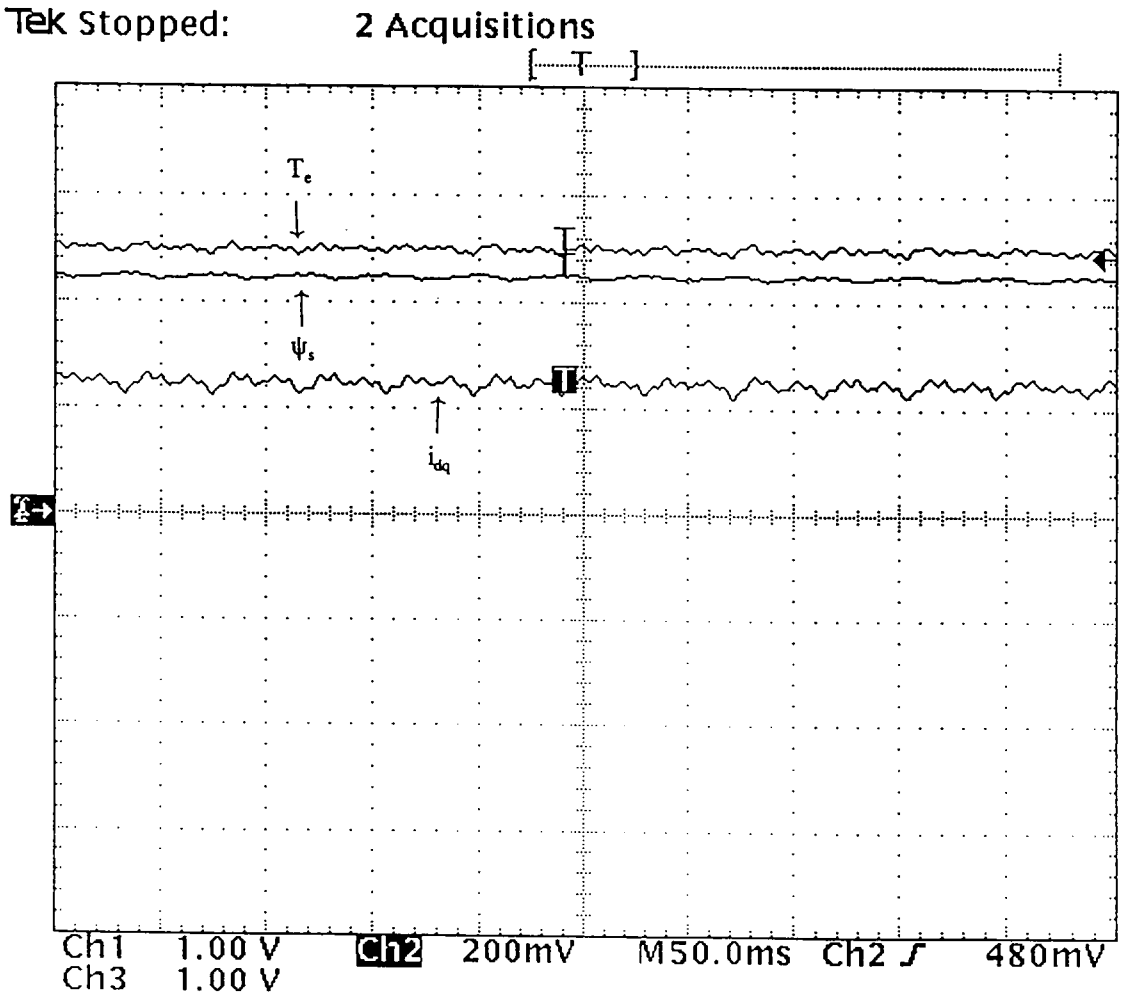


Figure 6.25 Estimated T_e (4 Nm/div), ψ_s (0.18 webers/div) and i_{dq} (0.77 A/div) in DVC

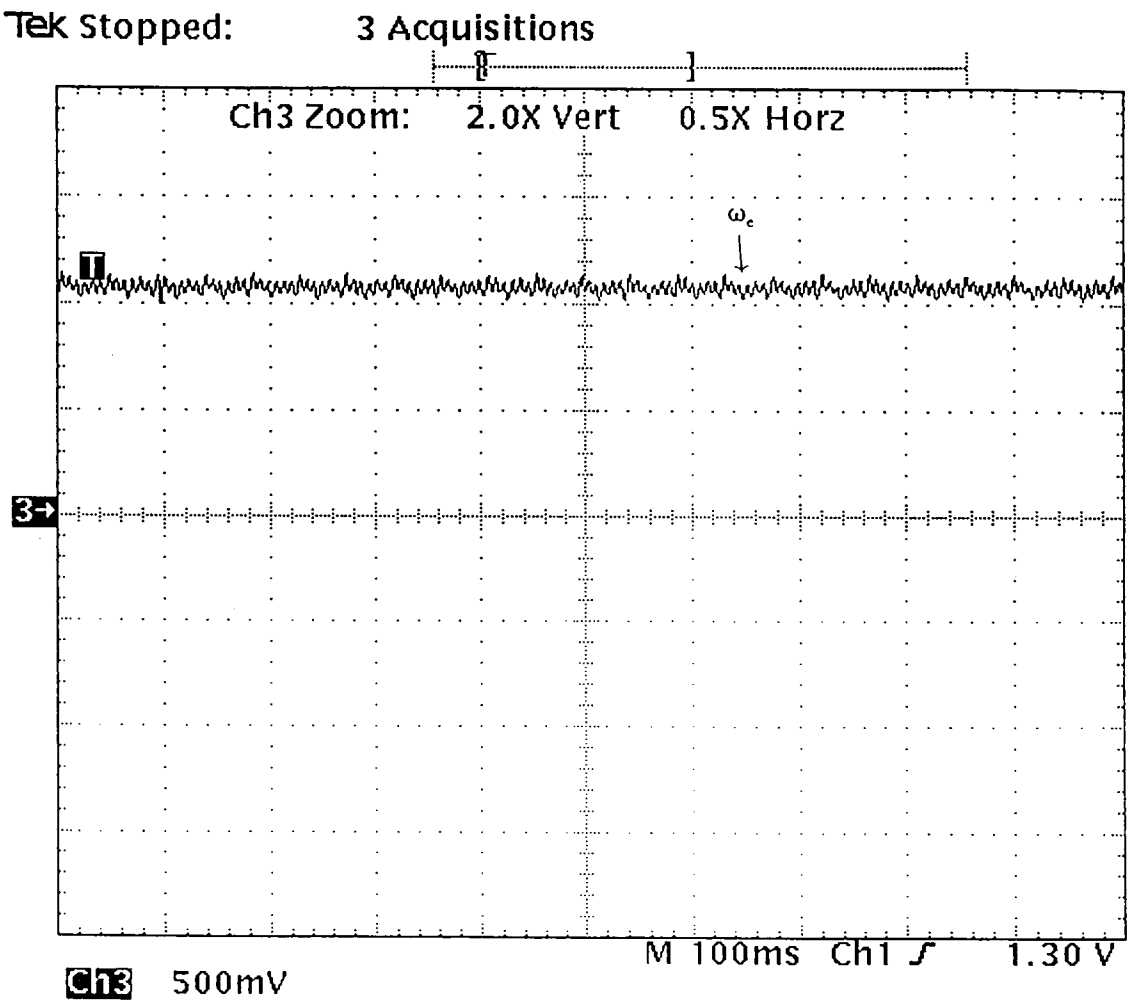


Figure 6.26 Estimated angular frequency ω_e (60 r/s/div) in DVC

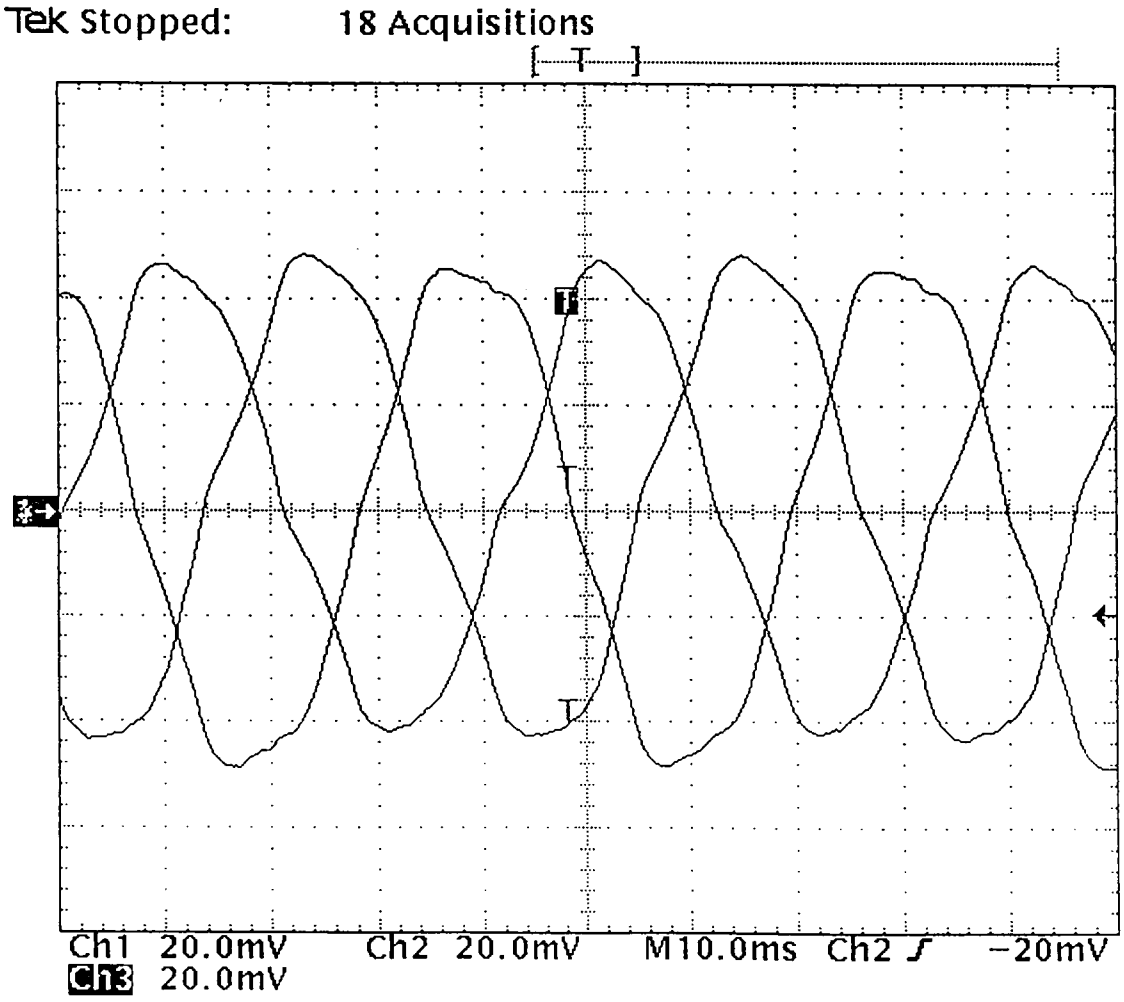


Figure 6.27 Phase currents i_a, i_b and i_c (4 A/div) in DVC

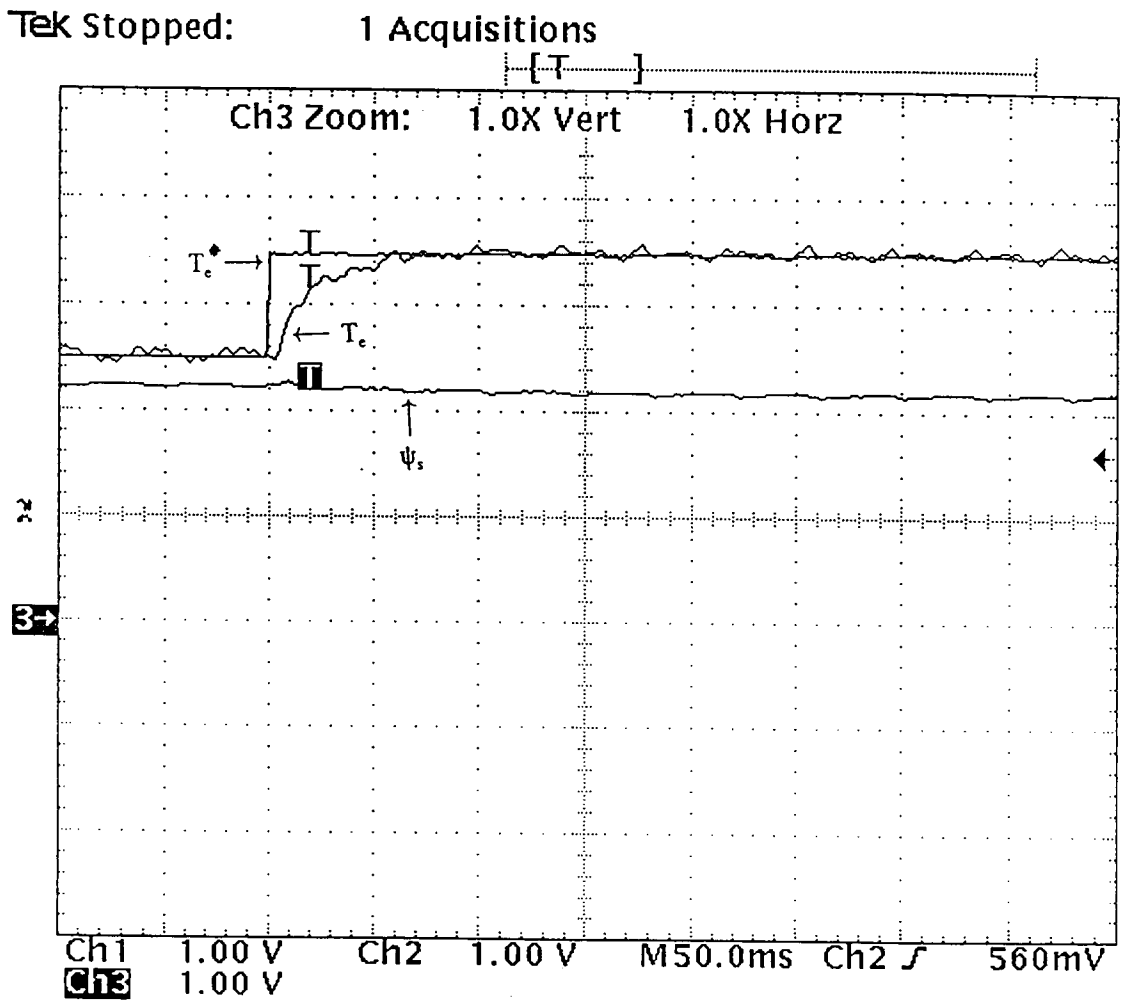


Figure 6.28 Step response of T_e^* from 6 Nm to 10 Nm (4 Nm/div) along with magnitude of stator flux "ψ_s" (0.18 webers/div)

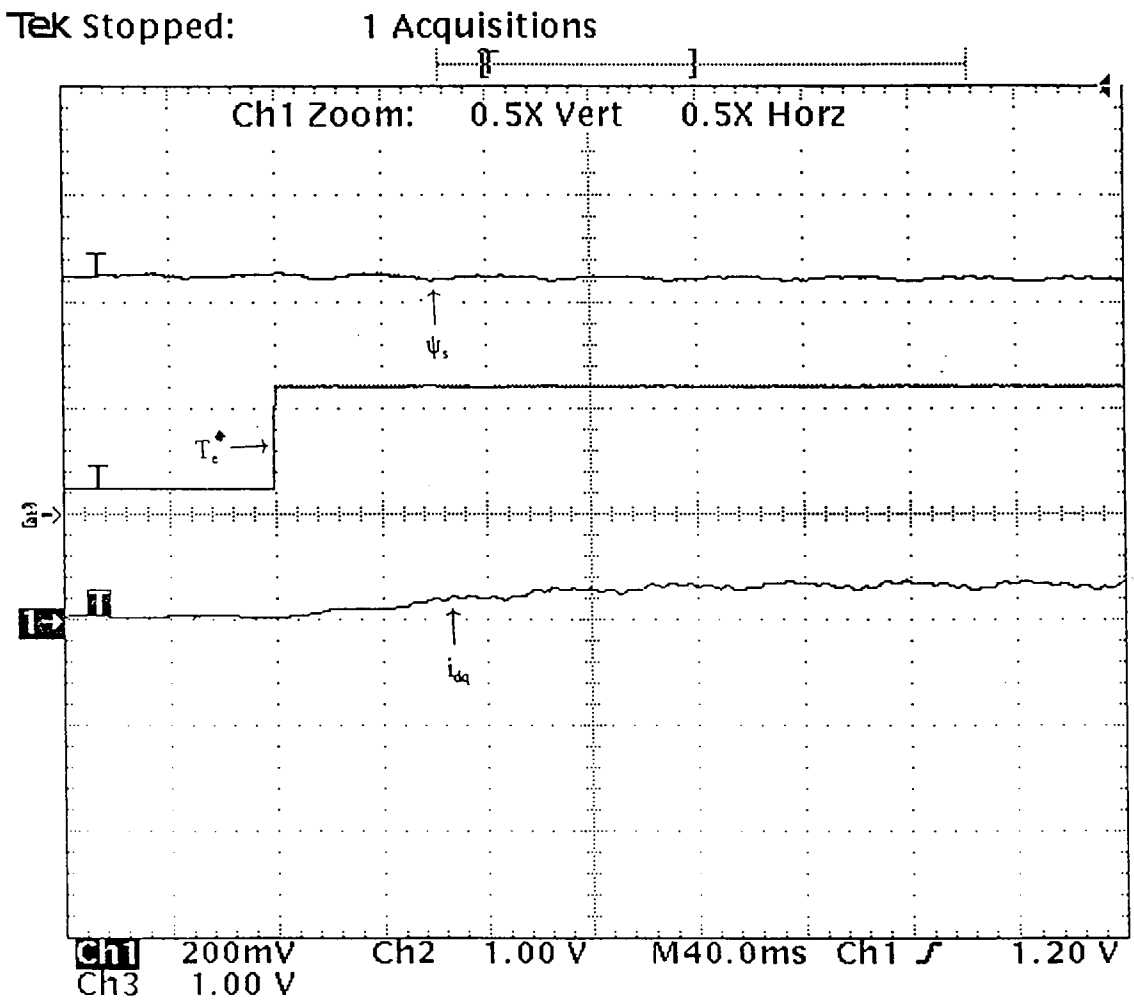


Figure 6.29 Decoupled system behavior when torque is stepped-up from 1 Nm to 5 Nm

T_e^*(1 Nm/div), ψ_s (0.18 webers/div) and i_{dq} (0.77 A/div)

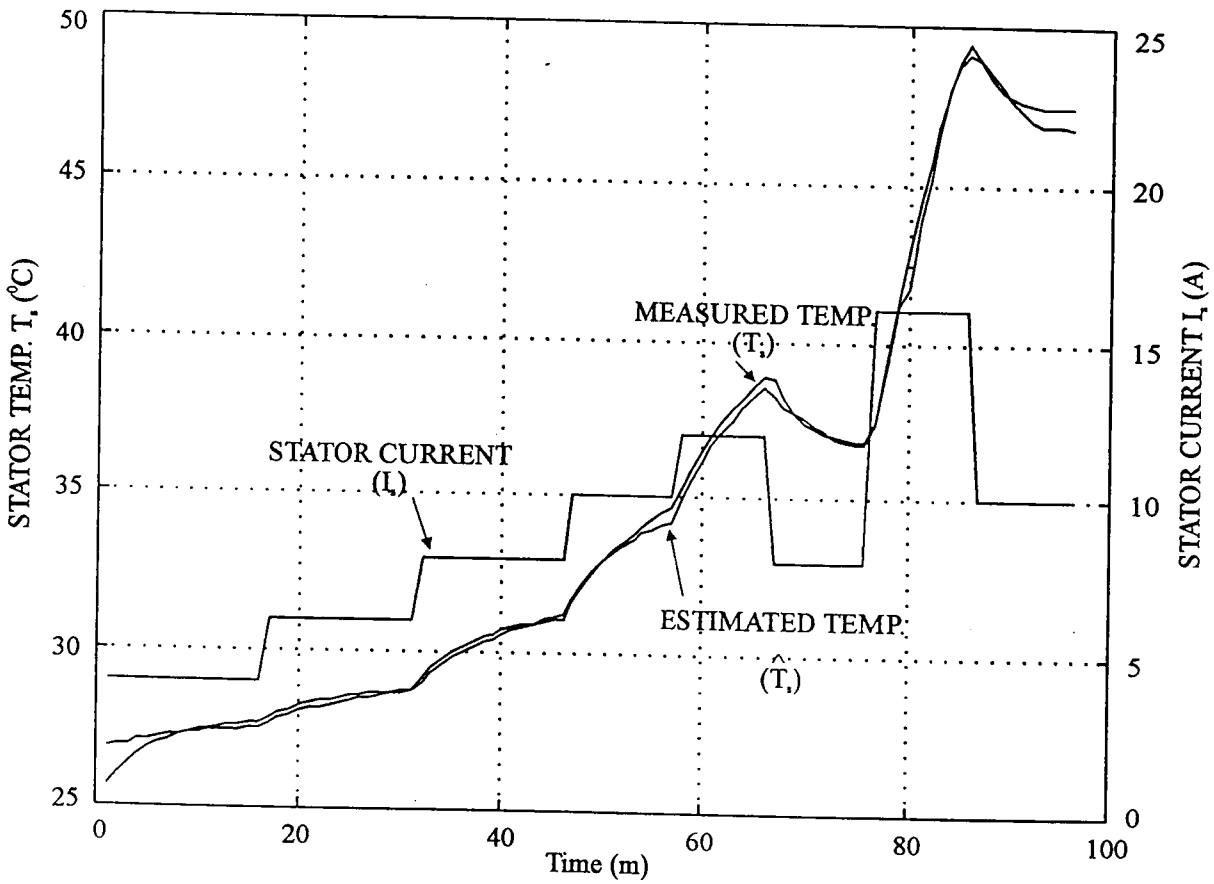


Figure 6.30 Stator temperature estimator performance with dynamically varying stator current but at constant speed of 357 rpm

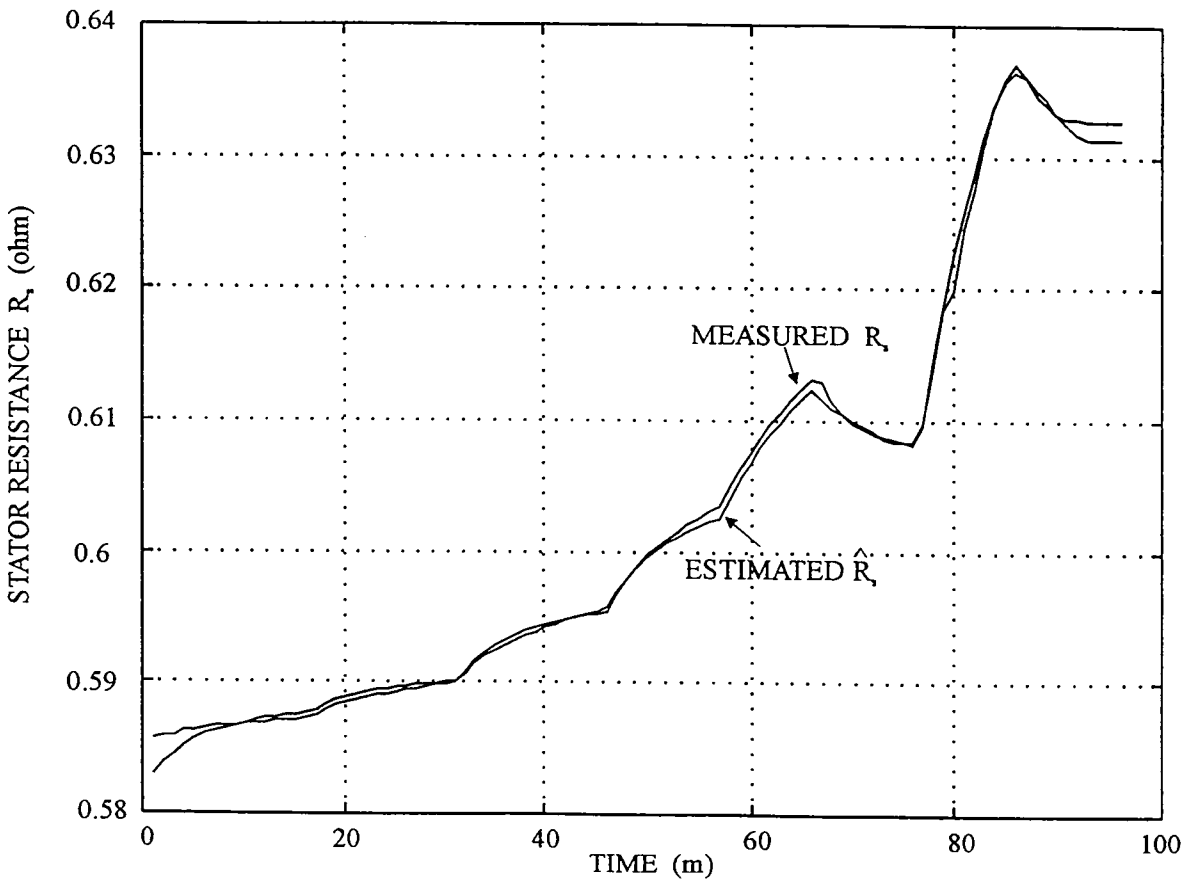


Figure 6.31 Stator resistance estimation by quasi-fuzzy estimator with dynamically varying stator current but at constant speed of 357 rpm.

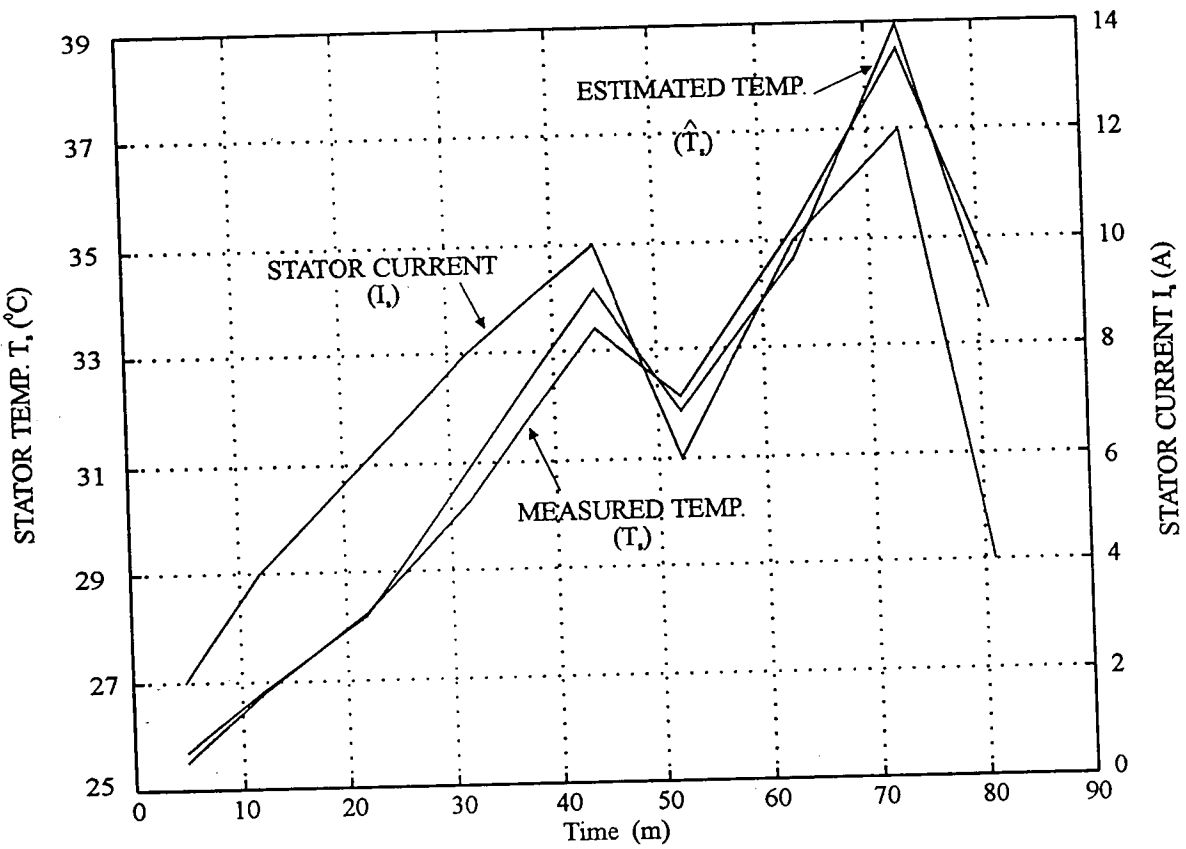


Figure 6.32 Stator temperature estimator performance with dynamically varying stator current but at constant speed of 725 rpm

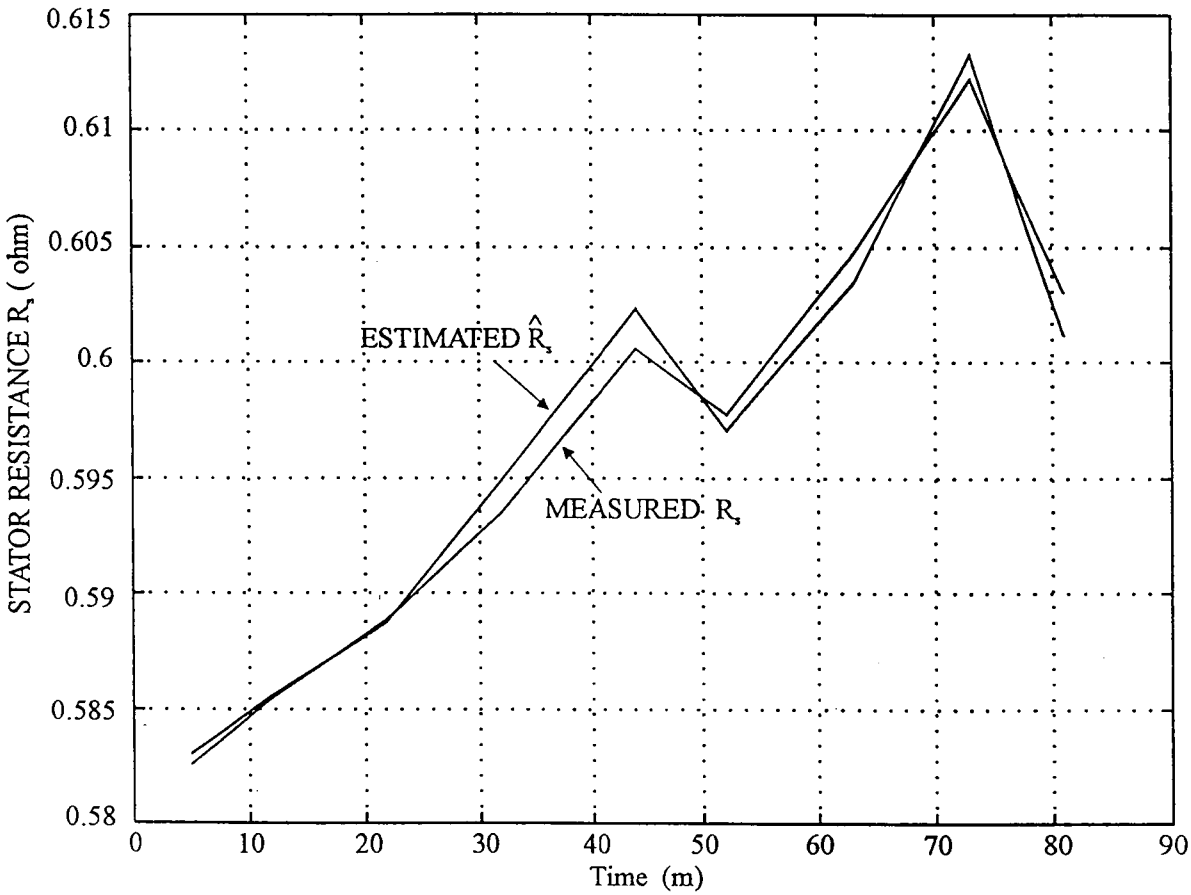


Figure 6.33 Stator resistance estimation by quasi-fuzzy estimator with dynamically varying stator current but at constant speed of 725 rpm.

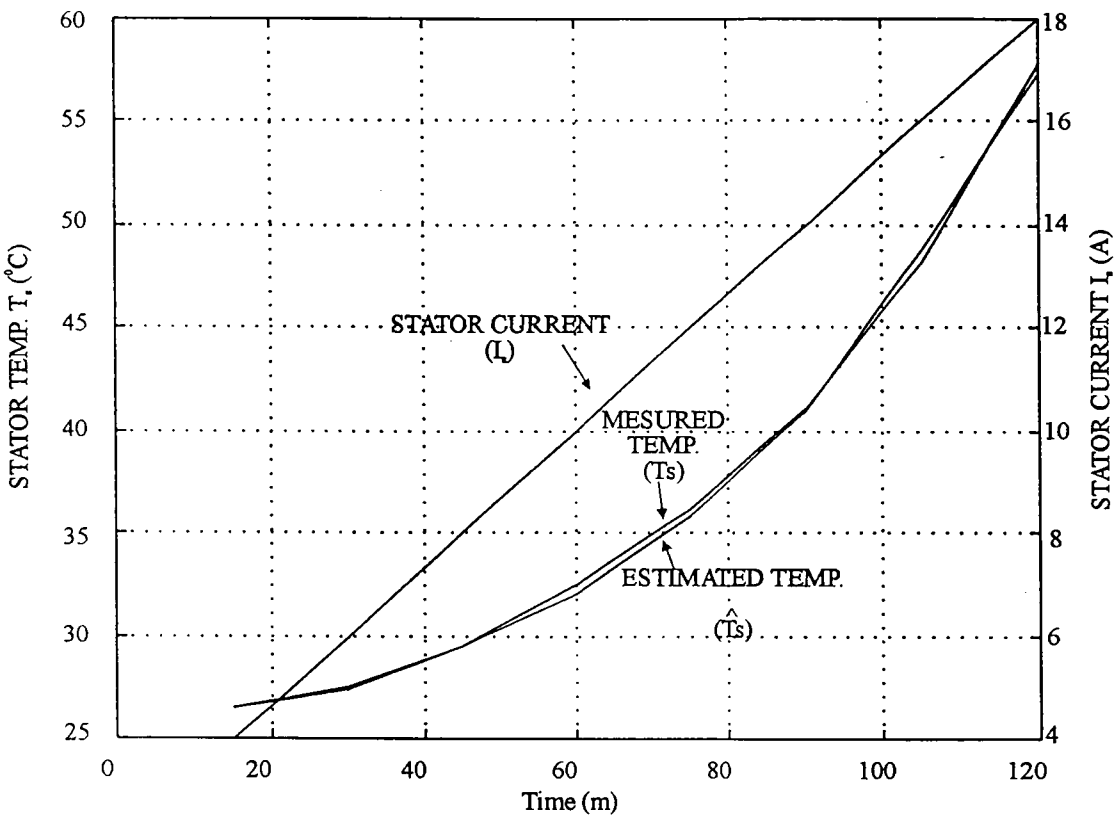


Figure 6.34 Stator temperature estimator performance with dynamically varying stator current but at constant speed of 918 rpm

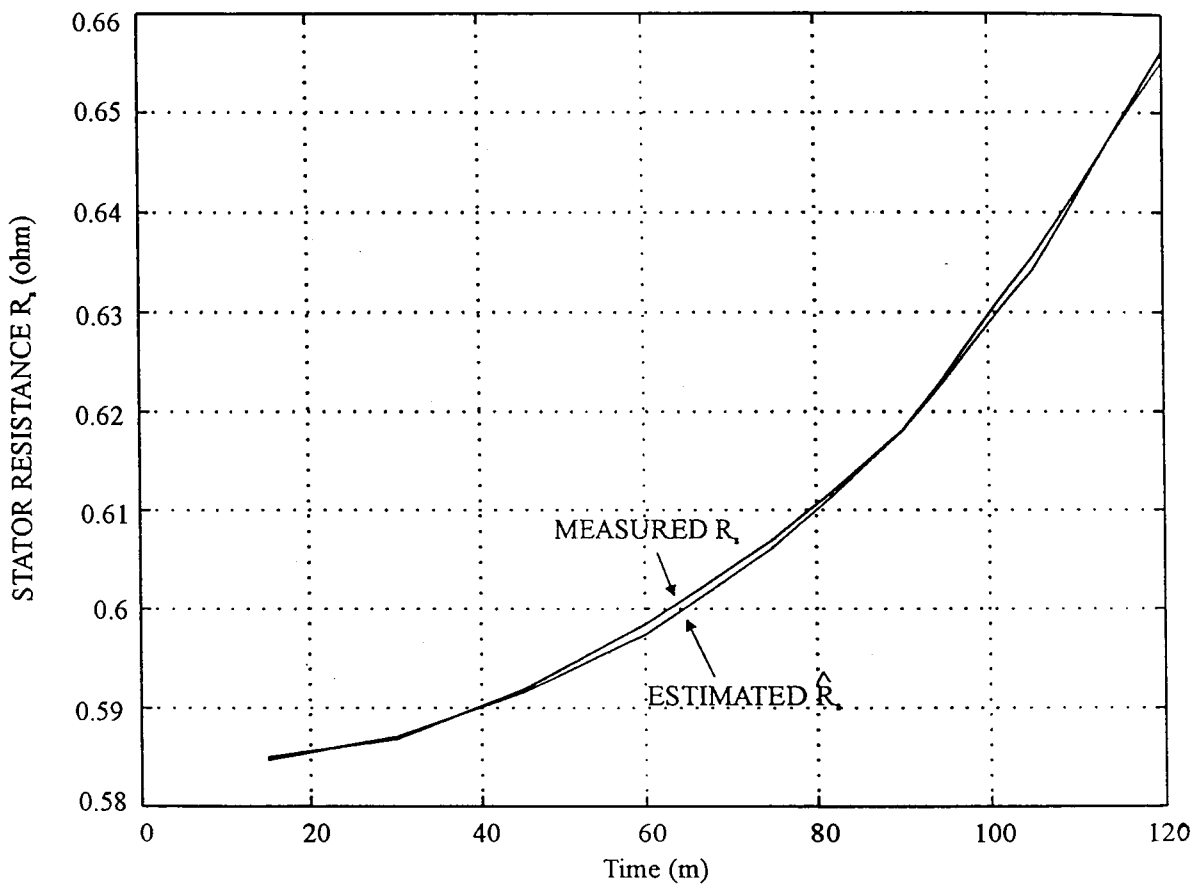


Figure 6.35 Stator resistance estimation by quasi-fuzzy estimator with dynamically varying stator current but at constant speed of 918 rpm.

6.3 Hardware Photos

This section shows views of the experimental hardware used in the validation of sensorless induction motor drive system. Figure 6.36 shows the detail hardware set-up used for the drive system. Figure 6.37 shows the detail view of 100 KW IGBT inverter. Figure 6.38 shows the view of the inverter protection circuit. Figure 6.39 shows the computer slots with two DSP boards, two IO boards and the Hanning interface board. Figure 6.40 shows the detail view of the stator temperature processing circuit. Figure 6.41 shows the connection between dynamometer and induction machine.

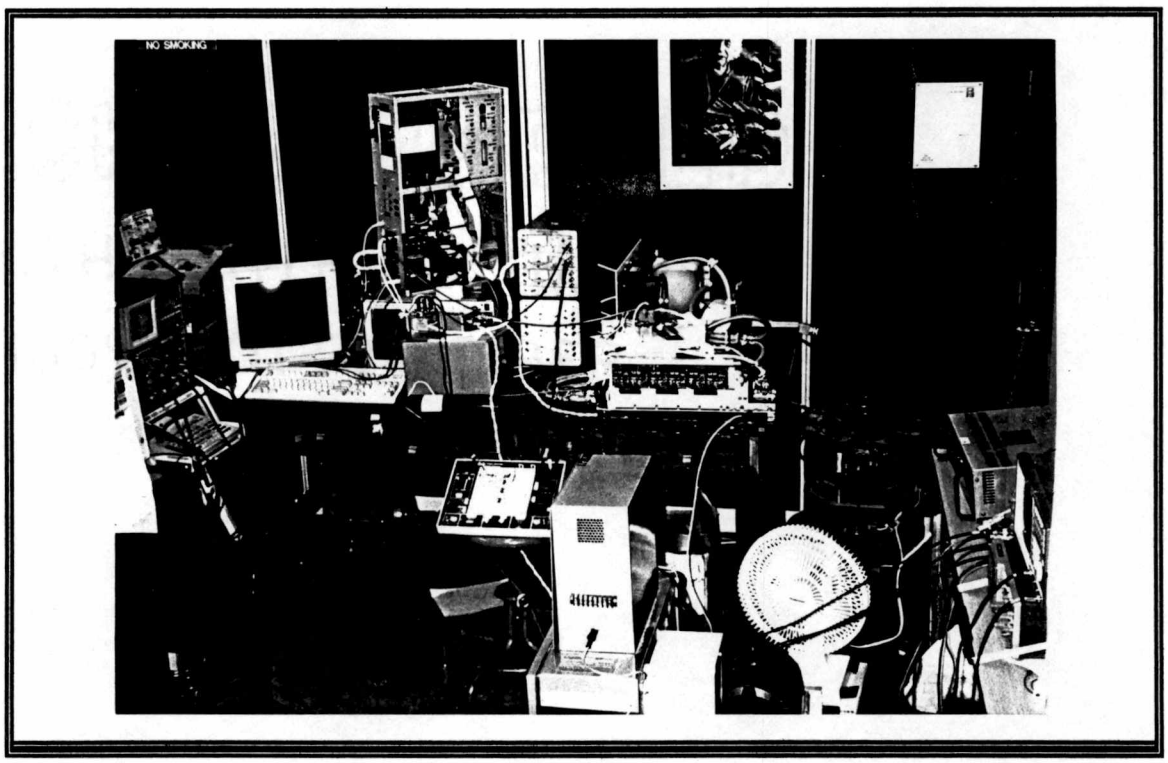


Figure 6.36 Overall view of the hardware set-up used for the drive system

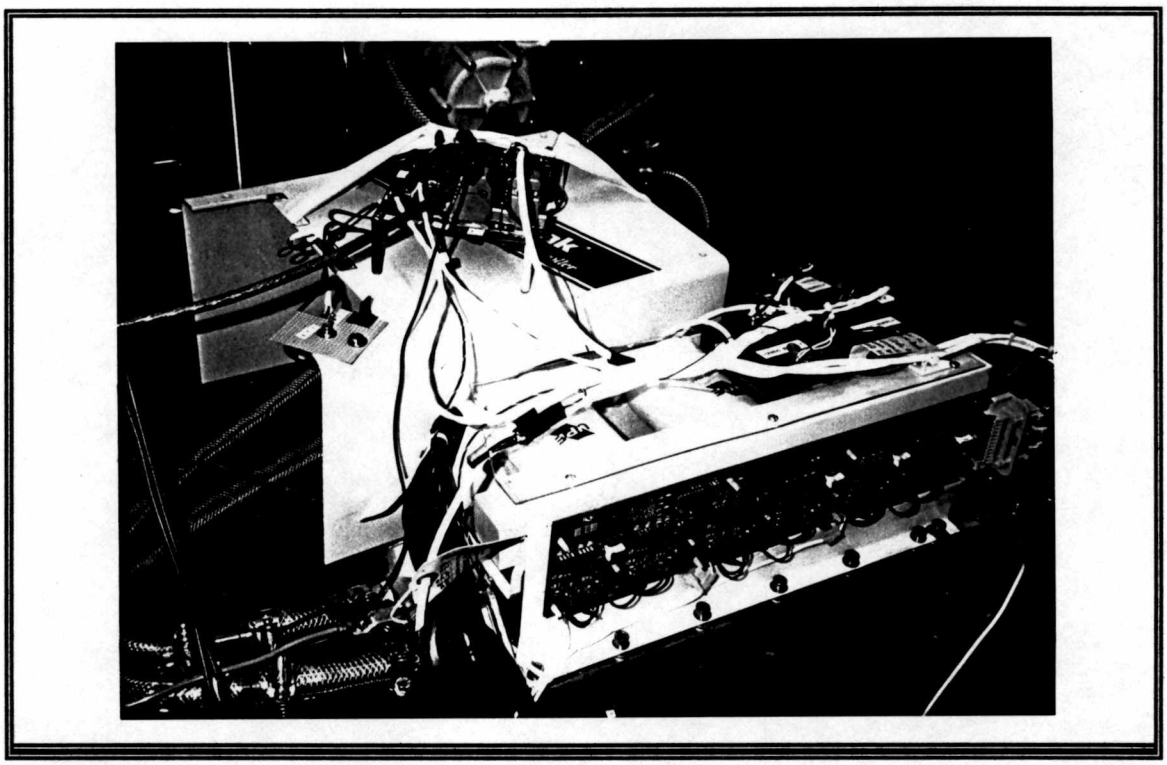


Figure 6.37 Detail view of the IGBT inverter used in experiment

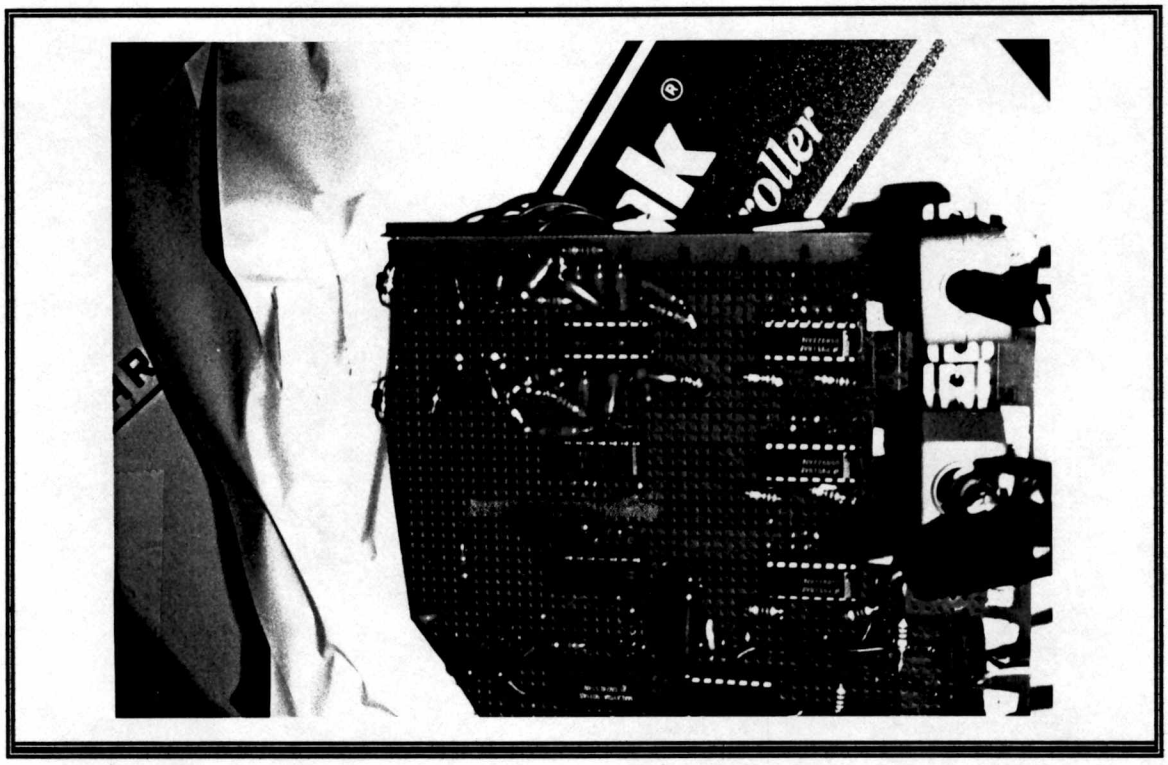


Figure 6.38 Detail view of the inverter protection circuit

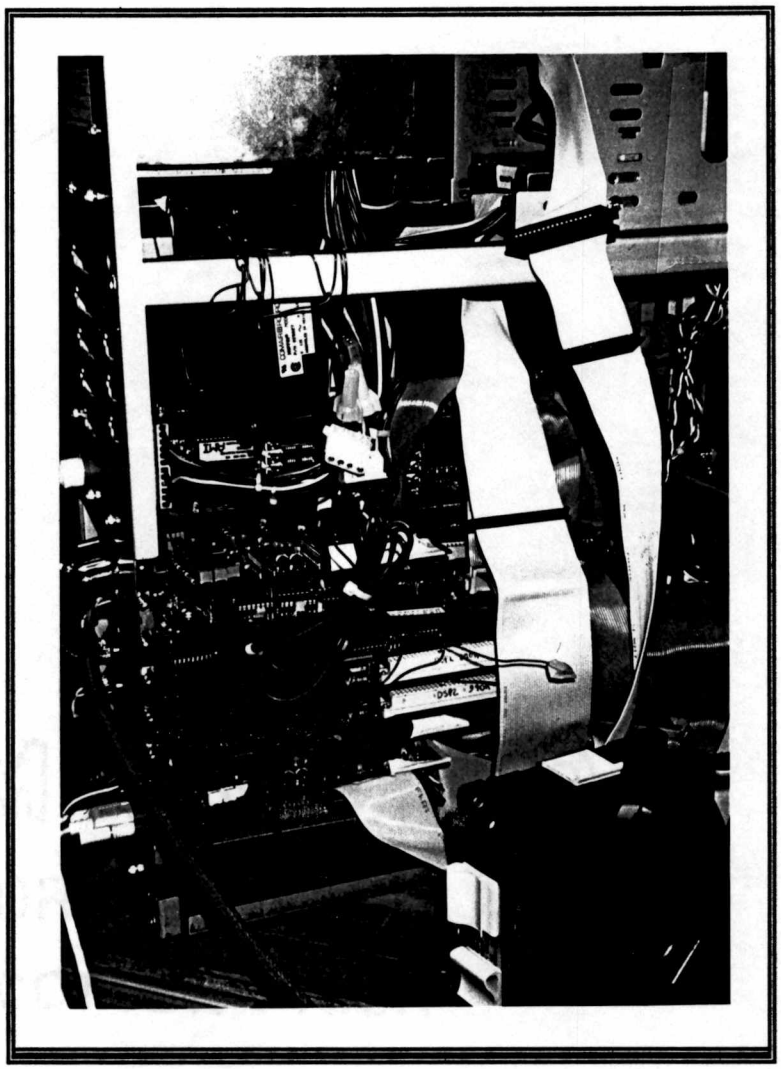


Figure 6.39 Computer slots with the two DSP boards, two IO boards and the Hanning interface board

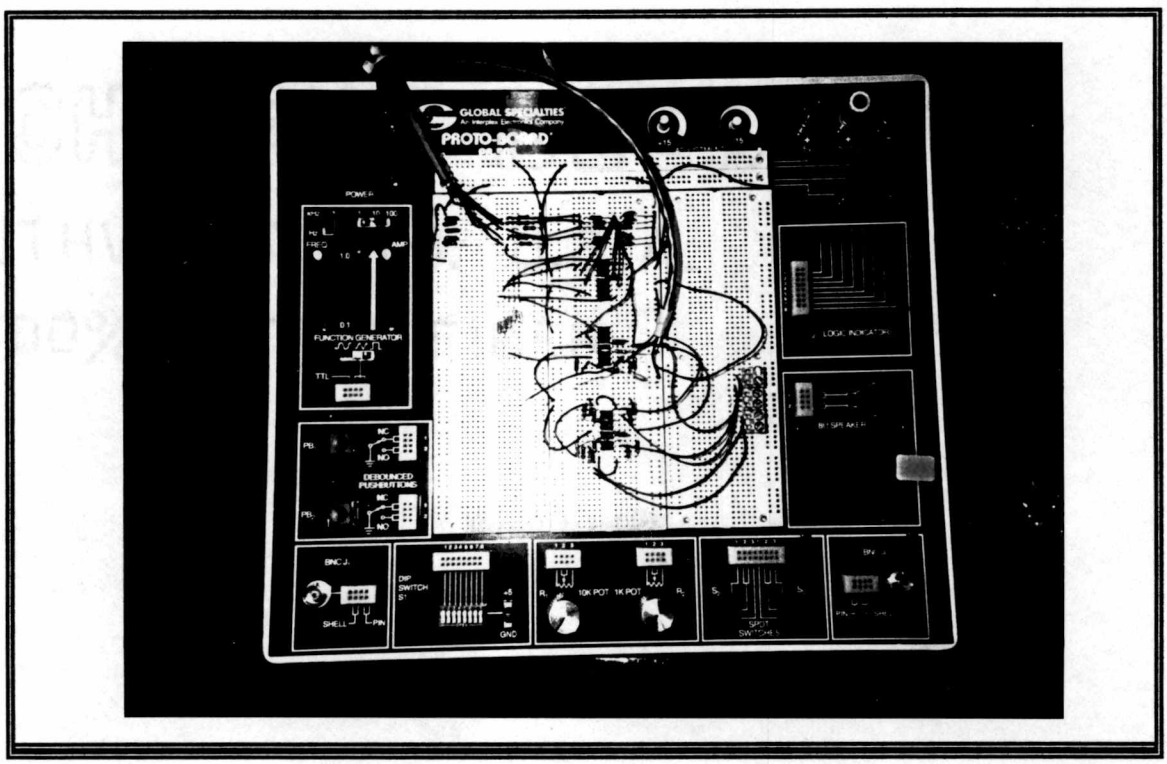


Figure 6.40 Detail view of stator temperature processing circuit

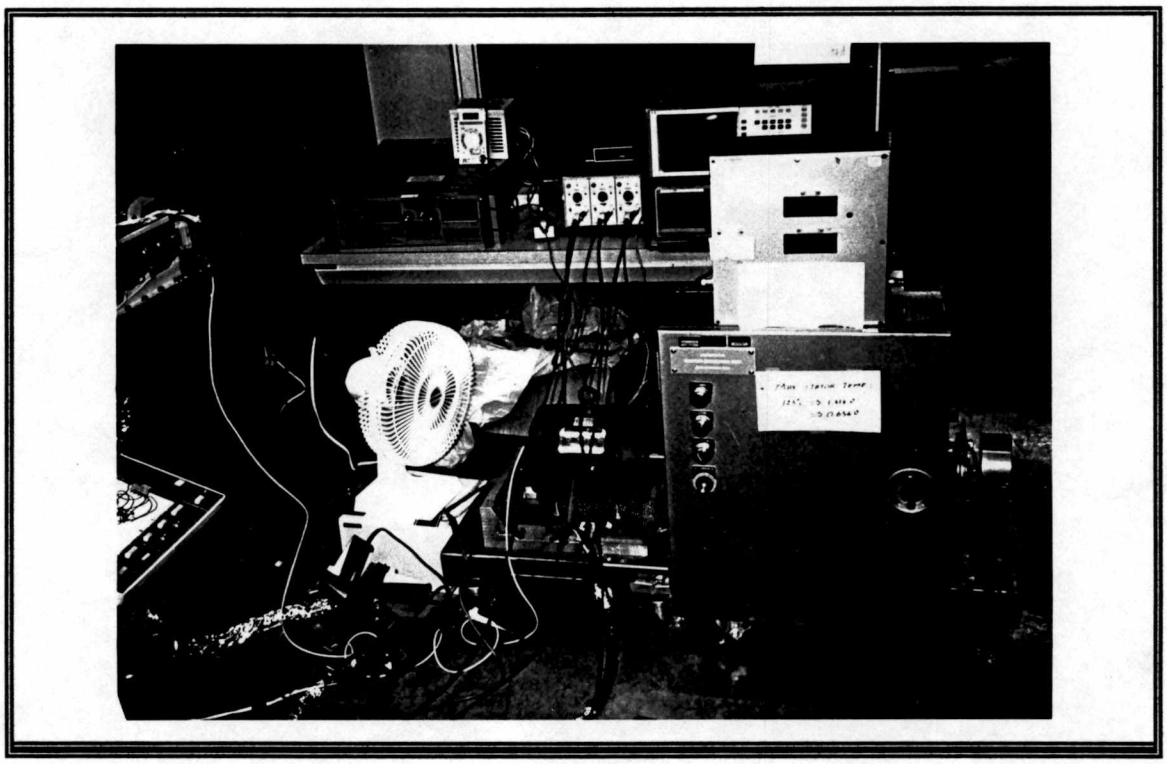


Figure 6.41 Dynamometer and induction machine

CHAPTER 7

CONCLUSION

A sensorless stator flux-oriented direct vector-controlled induction motor drive has been investigated in this thesis. The efficiency and general performance of this drive have been optimized using fuzzy logic and neural network techniques. More specifically, the work involved in the project can be summarized as follows:

(1) A novel zero speed start-up method of the vector-controlled drive has been developed using current model (Blaschke equation) of the machine that is only valid at zero speed. The drive starts at zero speed with finite torque, and then transitions to direct vector control mode where the machine picks up the speed. As the drive slows down to zero speed, transitioning occurs back to Blaschke mode.

(2) A software programmable cascaded low-pass filter has been designed that performs integration of machine terminal voltages to synthesize the flux vector for direct vector control of the drive. The integrator performs very well from very low machine frequency to the field-weakening region particularly avoiding dc offset problem near zero speed of the machine.

(3) An on-line method of stator resistance estimation of the machine has been developed using fuzzy logic. The resistance correction has been used for the flux vector synthesis which is particularly important near zero speed to avoid the coupling effect.

(4) An on-line method of efficiency optimization of the drive at light-load condition

has been designed by using fuzzy logic technique, and then translated into feedforward neural network which simplifies the real time computation considerably.

The work involved in the project consists of analysis, extensive digital simulation and experimental implementation on a complete laboratory 5 hp drive. Most of the multi-tasking software has been implemented by C language for a dual TI-TMS320C30 type DSP boards. The experimental performance (yet in progress) of the drive appears to be excellent.

The project was sponsored by Delphi Engine & Energy Management Systems - General Motors Corporation. They are likely to implement this development on a future electric vehicle drive system.

REFERENCES

REFERENCES

[1] B. K. Bose, M. G. Simoes, D. R. Crecelius, K. Rajashekara and R. Martin, "Speed sensorless hybrid vector controlled induction motor drive", Conf. Rec. IEEE-IAS Annual Meeting, 1995, pp. 137-143.

[2] X. Xu, R. W. De Donker and D. W. Novotny, "A stator flux oriented induction motor drive", Conf. Rec. IEEE-PESC, 1988, pp. 870-876.

[3] X. Xu and D. W. Novotny, "Implementation of direct stator flux orientation control on a versatile DSP based system", IEEE Transactions on Industrial Applications, Vol. 27, No. 4, July/August 1991, pp. 694-700.

[4] B. K. Bose and N. R. Patel, "A programmable cascaded low-pass filter synthesis for a stator flux oriented vector controlled induction motor drive", to be published in IEEE-IES Trans., vol. 43, no. 6, December 1996.

[5] B. K. Bose, Power Electronics and AC Drives, Prentice Hall, 1986.

[6] F. Blaschke, "The principle of field orientation as applied to the new TRANSVECTOR closed loop control system for rotating field machines", Siemens Review, Vol. 34, pp. 217-220, May 1972.

[7] Delphi Energy and Engine Management Systems note on thermistor network in 5 hp machine, April, 1992. Electronics, vol. 9, pp. 397-404, July 1994.

[8] G. C. D. Sousa and B. K. Bose, "Loss modeling of converter induction machine system for variable speed drive", Conf. Rec. Of IEEE-IECON, pp. 114-120, 1992.

[9] S. A. Mir, D. S. Zinger and M. E. Elbuluk, "Fuzzy controller for inverter fed induction machines", Conf. Rec. IEEE-IAS Annual Meeting, pp. 464-471, 1992.

[10] L. A. Cabrera and M. E. Elbuluk, "Tuning the stator resistance of induction motors using artificial neural network", Conf. Rec. of IEEE-PESC, pp. 421-427, 1995.

[11] L. A. Zadeh, "Fuzzy sets," Inform. Contr., vol. 8, pp. 338-3353, 1965.

[12] B. K. Bose (Ed.), Power Electronics and Variable Frequency Drives, IEEE Press, 1997.

[13] B. K. Bose, "A high performance drive control system of an interior permanent magnet synchronous machine", IEEE Trans. Ind. Appl., vol. 24, pp. 987-997, Nov.-

Dec. 1988.

[14] G. C. D. Sousa, B. K. Bose, and J. G. Cleland, "Fuzzy logic based on-line efficiency optimization control of an indirect vector controlled induction motor drive," IEEE/IECON'93 Conf. Rec., pp. 116801174, 1993.

[15] B. K. Bose, N. R. Patel and K. Rajashekara, "A neuro-fuzzy based on-line efficiency optimization control of a stator flux oriented direct vector controlled induction motor drive", IEEE-IES Trans., February 1997. (accepted)

[16] G. S. Buja and F. Todesco, "Neural network implementation of a fuzzy logic controller", IEEE Trans. Ind. Electron., vol. 41, no. 6, pp. 663-665, December 1994.

[17] M. M. Gupta, and G. K. Knopf, "Fuzzy neural network approach to control systems," Proc. of First Int. Symp. on Uncertainty Modeling and Analysis," Maryland, Dec. 3-5, 1990, pp. 483-488.

[18] D. S. Kirschen, D. W. Novotny and T. A. Lipo, " On-line efficiency optimization of a variable induction motor drive", in Conf. Rec. IEEE-IAS Ann. Meet., 1984, pp. 488-492.

[19] S. Horikawa, T. Furuhashi, S. Okuma, and Y. Uchikawa, "Composition methods of fuzzy neural networks," IEEE/IECON Conf. Rec., pp. 1253-1258, 1990.

[20] M. G. Simoes, "Using fuzzy neural networks (FNN) in modeling fuzzy estimators on power electronic waveforms," Internal Report of The University of Tennessee, December 1992.

[21] A. M. Trzynadlowski and S. Legowski, "Application of neural networks to the optimal control of three-phase voltage controlled inverters," IEEE Trans. On Power.

[22] B. K. Bose, "Expert system, fuzzy logic and neural network applications in power electronics and motion control," Proceedings of the IEEE, Special Issue on Power Electronics and Motion Control, pp. 1303-1323, August 1994.

[23] K. S. Narendra and K. Parthasarathy, "Identification and control of dynamical systems using neural networks," IEEE Trans. Neural Networks, vol. 1, pp. 4-27, March 1990.

[24] M. H. Kim, M. Godoy Simoes and B. K. Bose, "Neural network based estimation of power electronic waves," Accepted for publication in IEEE Transactions on Power Electronics.

[25] F. Harashima, Y. Demizu, S. Kondo, and H. Hashimoto, "Application of neural networks to power converter control," IEEE/IAS Annu. Meeting Conf. Rec., pp. 1086-1091, 1989.

[26] M. R. Buhl and R. D. Lorenz, "Design and implementation of neural networks for digital current regulation of inverter drives," IEEE/IAS Annu. Meeting Conf. Rec., pp. 415-421, October 1991.

[27] J. Theocharis and V. Petridis, "Neural network observer for induction motor control," IEEE Control Systems Magazine, pp. 26-37, April 1994.

[28] A. K. P. Toh, E. P. Nowicki, and F. Ashrafzadeh, "A Flux estimator for field oriented control of an induction motor using an artificial neural network", Conf. Rec. IEEE Ind. Appl. Soc. Annu. Meeting, pp. 585-592, October 1994.

[29] X. Xu, R. W. De Donker and D. W. Novotny, "Stator flux orientation control of inction machine in the field weakening region", IEEE-IAS Ann. Meet. Conf. Rec., pp.437-443, 1988.

[30] M. Godoy Simoes, B. K. Bose, and Ron Spiegel, "Fuzzy logic based intelligent control of a variable speed cage machine wind generation system," IEEE Power Electronics Specialists Conference, pp. 389-395, 1995.

[31] G. C. D. Sousa and B. K. Bose, "A fuzzy set theory based control of a phase controlled converter dc machine drive," IEEE-IAS Annu. Meeting Conf. Rec., Oct. 1991, pp. 854-861.

[32] M. G. Simoes and B. K. Bose, "Neural network based estimation of feedback signals for a vector controlled induction motor drive," IEEE/IAS Annu. Meeting Conf. Rec., pp. 471-489, 1994.

[33] M. G. Simoes and B. K. Bose, "Application of fuzzy logic in the estimation of power electronic waveforms," IEEE/IAS Annu. Meeting Conf. Rec., pp. 853-861, 1993.

[34] "Simnon user's guide," SSPA Systems, Goteborg, Sweden, 1992.

[35] MATLAB User's Guide, Mathworks, 1992.

[36] Texas Instruments, TMS320C30 User's Guide, Houston, TX., 1992.

[37] DSPLINK Specifications, Spectrum Systems, 1994.

[38] 4-channel AD/DA Board User's Guide, Spectrum System, 1994.

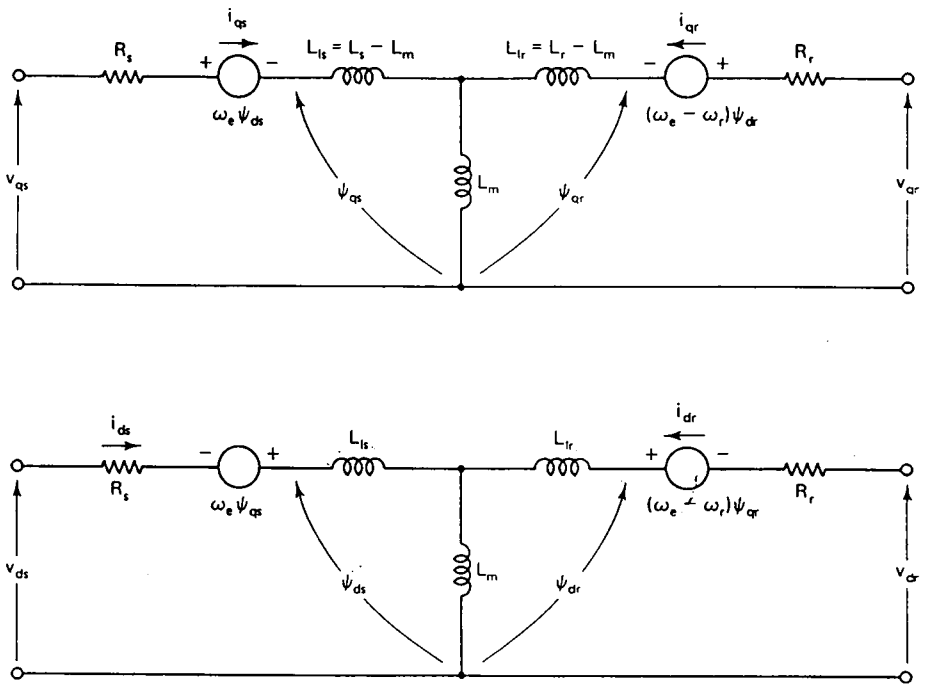
[39] Hanning Elecktro-Werke, Pulse Width Modulator PBM 1/87 Data Sheets, Oerlinghausen, Germany, 1993.

[40] Spectrum Signal Processing, Product Specifications Catalog, Blaine, WA., March 1992.

APPENDICES

APPENDIX A

Stator Flux Derivation



d^e - q^e equivalent circuit

We will derive stator flux from rotor flux equations:
from eq. ckt write loop equation for rotor side:

$$\frac{d\psi_{qr}}{dt} + R_r i_{qr} + (\omega_e - \omega_r)\psi_{dr} = 0 \text{.....(1)}$$

$$\frac{d\psi_{dr}}{dt} + R_r i_{dr} - (\omega_e - \omega_r)\psi_{qr} = 0 \text{.....(2)}$$

Also from eqn. ckt we know:

$$\psi_{qr} = L_m(i_{qs} + i_{qr}) + L_{er} i_{qe} = L_m i_{qs} + L_m i_{qr} + L_{er} i_{qr}$$
$$= i_{qr}(L_m + L_{er}) + L_m i_{qs}$$

$$\psi_{dr} = L_m(i_{ds} + i_{dr}) + L_{er}i_{dr} = L_m i_{ds} + L_m i_{dr} + L_{er}i_{dr}$$
$$= i_{dr}(L_m + L_{er}) + L_m i_{ds}$$

Let $L_r = L_{er} + L_m$, simplifying above:

$$\psi_{qr} = L_r i_{qr} + L_m i_{qs} \text{.....(3)}$$

$$\psi_{dr} = L_r i_{dr} + L_m i_{ds} \text{.....(4)}$$

Solve eqn. (3) and (4) for currents i_{qr} and i_{dr}

$$i_{qr} = \frac{\psi_{qr}}{L_r} - \frac{L_m i_{qs}}{L_r} \text{.....(5)}$$

$$i_{dr} = \frac{\psi_{dr}}{L_r} - \frac{L_m i_{ds}}{L_r} \text{.....(6)}$$

In order to eliminate the rotor currents from eqn's (1) & (2), substitute eqn's (5) & (6) into (1) & (2) for i_{qr} and i_{dr}:

$$\frac{d\psi_{qr}}{dt} + R_r[\frac{\psi_{qr}}{L_r} - \frac{L_m i_{qs}}{L_r}] + (\omega_e - \omega_r)\psi_{dr} = 0$$

$$\frac{d\psi_{qr}}{dt} + \frac{R_r}{L_r}\psi_{qr} - \frac{R_r}{L_r}L_m i_{qs} + \omega_{se}\psi_{dr} = 0$$

similarly for the d-axis eqn.:

$$\frac{d\psi_{dr}}{dt} + \frac{R_r}{L_r}\psi_{dr} - \frac{R_r}{L_r}L_m i_{ds} - \omega_{se}\psi_{qr} = 0$$

multiply above eqns by $L_r / R_r = T_r$ and simplify:

$$(1 + sT_r)\psi_{qr} - L_m i_{qs} + \omega_{se} T_r \psi_{dr} = 0 \text{.....(7)}$$

$$(1 + sT_r)\psi_{dr} - L_m i_{ds} + \omega_{se} T_r \psi_{qr} = 0 \text{.....(8)}$$

Now write the stator flux linkage equations from the eqn. ckt:
Let: $L_s = L_{es} + L_m$

$$\Psi_{qs} = L_m(i_{qs} + i_{qr}) + L_{es} i_{qs}$$
$$\Psi_{qs} = L_s i_{qs} + L_m i_{qr} \text{.....(9)}$$

$$\Psi_{ds} = L_m(i_{ds} + i_{dr}) + L_{es} i_{ds}$$
$$\Psi_{ds} = L_s i_{ds} + L_m i_{dr} \text{.....(10)}$$

solve for rotor currents i_{qr} & i_{dr} in eqn. (9) and (10)

$$i_{qr} = \frac{\psi_{qs}}{i_m} - \frac{L_s i_{qs}}{L_m} \text{.....(11)}$$

$$i_{dr} = \frac{\psi_{ds}}{i_m} - \frac{L_s i_{ds}}{L_m} \text{.....(12)}$$

substitute eqn (11) and (12) into eqn. (3) and (4) eliminating the rotor currents:

$$\psi_{qr} = L_r[\frac{\psi_{qs}}{L_m} - \frac{L_s}{L_m} i_{qs}] + L_m i_{qs}$$

$$\psi_{dr} = L_r[\frac{\psi_{ds}}{L_m} - \frac{L_s}{L_m} i_{ds}] + L_m i_{ds}$$

This is the relationship between rotor and sludor fluxes simplify the above eqn's to get:

$$\psi_{qr} = \frac{L_r}{L_m}\psi_{qs} - \frac{L_r L_s}{L_m} i_{qs} + L_m i_{qs}$$

$$\psi_{qr} = \frac{L_r}{L_m}\psi_{qs} + i_{qs}[L_m - \frac{L_r L_s}{L_m}] \quad \text{.....(13)}$$

Similarly,

$$\psi_{dr} = \frac{L_r}{L_m}\psi_{ds} + i_{ds}[L_m - \frac{L_r L_s}{L_m}] \quad \text{.....(14)}$$

$$(1 + sT_r)[\frac{L_r}{L_m}\psi_{qs} + i_{qs}[L_m - \frac{L_r L_s}{L_m}]] - L_m i_{qs} + \omega_{se} T_r[\frac{L_r}{L_m}\psi_{ds} + i_{ds}[L_m - \frac{L_r L_s}{L_m}]] = 0$$

substitute eqn. (13) and (14) into eqn's (7) and (8) to eliminate the rotor flux ψ_{dr} and ψ_{qr}:

$$\frac{L_r}{L_m}\psi_{qs} + sT_r\frac{L_r}{L_m}\psi_{qs} + i_{qs}\sigma + sT_r\sigma i_{qs} - L_m i_{qs} + \omega_{se} T_r\frac{L_r}{L_m}\psi_{ds} + \omega_{se} T_r\sigma i_{ds} = 0$$

$$\psi_{qs}(\frac{L_r}{L_m} + sT_r\frac{L_r}{L_m}) = i_{qs}(L_m - \sigma - sT_r\sigma) - \omega_{se} T_r(\frac{L_r}{L_m}\psi_{ds} + i_{ds}\sigma)$$

Mulitply both sides by $\frac{L_m}{L_r}$:

$$\psi_{qs}(1 + sT_r) = i_{qs}[\frac{L_m^2}{L_r} + L_s - \frac{L_m^2}{L_r} + [L_s - \frac{L_m^2}{L_r}]sT_r] - \omega_{se} T_r[\psi_{ds} - [L_s - \frac{L_m^2}{L_r}]i_{ds}]$$

$$\psi_{qs}(1 + sT_r) = i_{qs}[L_s + (L_s - \frac{L_m^2}{L_r})sT_r] - \omega_{se}T_r[\psi_{ds} - (L_s - \frac{L_m^2}{L_r})i_{ds}]$$

$$\psi_{qs}(1 + sT_r) = i_{qs}L_s[1 + (L_s - \frac{L_m^2}{L_r})sT_r] - \omega_{se}T_r[\psi_{ds} - (L_s - \frac{L_m^2}{L_r})i_{ds}] \quad \text{.....(15)}$$

Similarly for the d axis equation:

$$(1 + sT_r)[\frac{L_r}{L_m}\psi_{ds} + i_{ds}\sigma] - L_m i_{ds} - \omega_{se}T_r[\frac{L_r}{L_m}\psi_{qs} + i_{qs}\sigma] = 0$$

$$\frac{L_r}{L_m}\psi_{ds} + sT_r\frac{L_r}{L_m}\psi_{ds} + i_{ds}\sigma + sT_r\sigma i_{ds} - L_m i_{ds} - \omega_{se}T_r\frac{L_r}{L_m}\psi_{qs} - \omega_{se}T_r i_{qs}\sigma = 0$$

$$\psi_{ds}(\frac{L_r}{L_m} + sT_r\frac{L_r}{L_m}) = i_{ds}(L_m - \sigma - sT_r\sigma) + \omega_{se}T_r(\frac{L_r}{L_m}\psi_{qs} + \sigma i_{qs})$$

multiply by L_m/L_r on both sides

$$\psi_{ds}(1 + sT_r) = i_{ds}L_s(1 + (L_s - \frac{L_m^2}{L_r})sT_r) + \omega_{se}T_r(\psi_{qs} - (L_s - \frac{L_m^2}{L_r})i_{qs}) \quad \text{.....(16)}$$

For vector control: $\psi_{qs} = 0$, $\psi_{ds} = \psi_s$

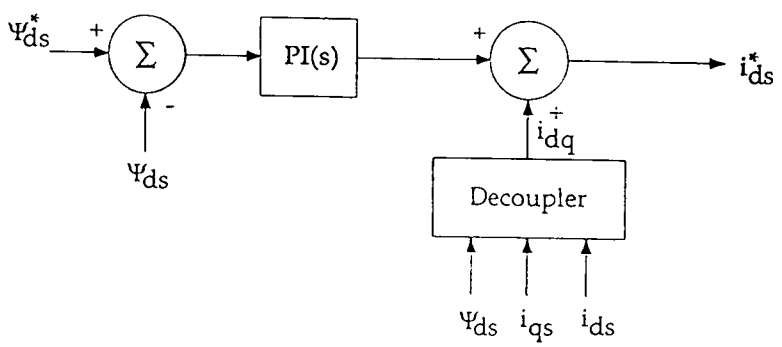
Now (15) and (16) simplify to:

$$i_{qs}L_s(1 + (L_s - \frac{L_m^2}{L_r})sT_r) = \omega_{se}T_r(\psi_{ds} - (L_s - \frac{L_r^2}{L_r})i_{ds}).....(17)$$

$$\psi_{ds}(1 + sT_r) = i_{ds}L_s(1 = (L_s - \frac{L_m^2}{L_r})sT_r) - \omega_{se}T_r i_{qs}(L_s - \frac{L_m^2}{L_r}).....(18)$$

Equations (17) and (18) are seen to be coupled. If the torque component i_{ds} is changed the flux will else be changed. Therefore, a feed forward control method is used to decouple the i_{ds} & i_{qs}.

Block diagram of Decoupler:



$$i_{ds} = PI(s)(\psi_{ds}^* - \psi_{ds}) + i_{dq}.....(19)$$

Substituting eqn. (19) into eqn (18) for i_{ds} will give,

$$\psi_{ds}(1 + sT_r) = (1 + \sigma_1 sT_r)L_s[PI(s)(\psi_{ds}^* - \psi_{ds}) + i_{dq}] - \omega_{se}T_r i_{qs}\sigma_1$$

where, $\sigma_1 = L_s - L_m^2/L_r$

simplfy the above eqn.,

$$\psi_{ds}(1 + sT_r) = (1 + \sigma_1 sT_r)L_s \mathrm{PI}(s)(\psi^*_{ds} - \psi_{ds}) + (1 + \sigma_1 sT_r)L_s i_{dq} - \omega_{se} T_r i_{qs} \sigma_1 \text{.....(20}$$

To decouple ψ_{ds} from i_{qs} in eqn (20) we must have,

$$(1 + \sigma_1 sT_r)L_s i_{dq} - \omega_{se} T_r i_{qs} \sigma_1 = 0 \text{.....(21)}$$

If these terms equal to zero then eqn (20) becomes,

$$\psi_{ds}(1 + sT_r) = (1 + \sigma_1 sT_r)L_s \mathrm{PI}(s)(\psi^*_{ds} - \psi_{ds})$$

and this equation represents a decupled system.
Therefore, from eqn. (21) solve for i_{dq},

$$i_{dq} = \frac{\omega_{sl} T_r i_{qs} \sigma_1}{(1 + \sigma_1 sT_r)L_s} \text{.....(22)}$$

where, $\sigma_1 = L_s - L_m^2/L_r$

ω_{sl} is determined from eqn. (17)

$$\omega_{sl} = \frac{i_{qs} L_s (1 + \sigma_1 sT_r)}{T_r(\psi_{ds} - \sigma_1 i_{ds})} \text{.....(23)}$$

The decoupled torque is,

$$T_e = \frac{3}{4} p(i_{qs}\psi_{ds} - i_{ds}\psi_{qs}) = \frac{3}{4} p i_{qs}\psi_{ds}$$

So,

$$i_{qs} = \frac{4T^*_e}{3p\psi^*_{ds}}$$

substitute eqn. (23) into (22) for ω_{sl} from which we can derive the decoupling current eqn.,

$$i_{dq} = \frac{i_{qs}^2 \sigma_1}{(\psi_{ds} - \sigma_1 i_{ds})} \quad \text{.....(24)}$$

Eqn . (24) would be used in the decoupler block of the main block diagram.

APPENDIX B

Blaschke Equations with Stator Flux Orientation

The Blaschke equations for rotor flux orientation are as follows:

$$\frac{d\psi_{qr}^{s}}{dt} = \frac{L_m}{T_R} i_{qs}^{s} + \omega_r \psi_{dr}^{s} - \frac{1}{T_R} \psi_{qr}^{s} \quad \text{..... (1)}$$

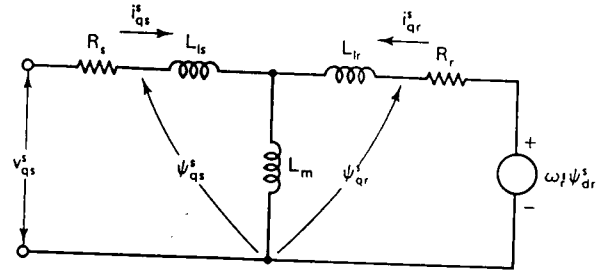
$$\frac{d\psi_{dr}^{s}}{dt} = \frac{L_m}{T_R} i_{ds}^{s} - \omega_r \psi_{qr}^{s} - \frac{1}{T_R} \psi_{dr}^{s} \quad \text{..... (2)}$$

The blaschke current model equations are valid at $\omega_r = 0$ for direct vector control method. So the unit vectors generated by current model equation will also be valid at zero speed. At $\omega_r = 0$, equations (1) and (2) will be expressed as follows:

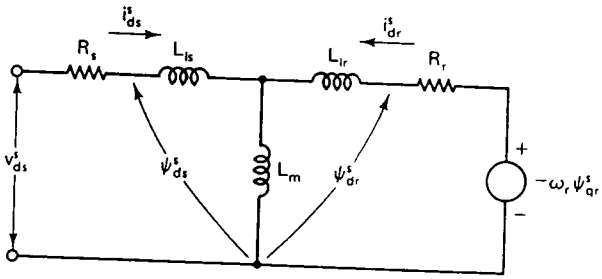
$$\frac{d\psi_{qr}^{s}}{dt} = \frac{L_m}{T_R} i_{qs}^{s} - \frac{1}{T_R} \psi_{qr}^{s} \quad \text{..... (3)}$$

$$\frac{d\psi_{dr}^{s}}{dt} = \frac{L_m}{T_R} i_{ds}^{s} - \frac{1}{T_R} \psi_{dr}^{s} \quad \text{..... (4)}$$

The following is the machine equivalent circuit,



(a)



(b)

From above circuit,

$$\psi^s_{qs} = L_{ls} i^s_{qs} + L_m i^s_{qm} \quad \text{.....(5)}$$

where,

$$i^s_{qm} = i^s_{qs}((sL_{lr} + R_r)/(sL_r + R_r)) \quad \text{.....} \quad (6)$$

Therefore,

$$\psi^s_{qs} = i^s_{qs}(L_{ls} + L_m(sL_{lr} + R_r)/(sL_r + R_r)) \quad \ldots\ldots\ldots\ldots \quad (7)$$

rearrange the equation (7), we get,

$$\psi^s_{qs} = i^s_{qs}(sL_rL_{ls} + L_{ls}R_r + sL_mL_{lr} + L_mR_r)/(sL_r + R_r)) \quad \ldots\ldots\ldots\ldots \quad (8)$$

rearrange the equation (8), we get,

$$(sL_r + R_r)\psi^s_{qs} = si^s_{qs}(L_rL_{ls} + L_mL_{lr}) + i^s_{qs}L_sR_r \quad \ldots\ldots\ldots\ldots(9)$$

solve the equation (9) for $d\Psi^s_{qs} / dt$ or $s\Psi^s_{qs}$,

$$d\psi^s_{qs}/dt = -(1/T_r)\psi^s_{qs} + A(di^s_{qs}/dt) + Bi^s_{qs} \quad \ldots\ldots\ldots\ldots \quad (10)$$

Similarly for d-axes flux can be evaluate from the following equation,

$$d\psi^s_{ds}/dt = -(1/T_r)\psi^s_{ds} + A(di^s_{ds}/dt) + Bi^s_{ds} \quad \ldots\ldots\ldots\ldots \quad (11)$$

where, $T_r = L_r / R_r$

$A = (L_r L_{ls} + L_m L_{lr}) / L_r$

$B = L_s R_r / L_r$

Equations (10) and (11) are current model equations of stator flux oriented BlaschKe Model. These equations are valid only at zero speed.

APPENDIX C

Simnon Programs Listing for the Drive System

```
"----------------------------------------------------------------------------------------
"                    MACRO FOR VECTOR CONTROL SYSTEM
"----------------------------------------------------------------------------------------

MACRO EV
switch log on
switch trace on
SYST  PI  STASYN  SYNSTA  T2F3F  T3F2F  SPWM  IM &
      FILTER  LPF  DECOUP  SWITCH  ECALC  BTY &
      TORQUE  BLASTA  CURDER  ANNOPT  RAMP  TRNCTRL  CSTART

store i11 i12 i13 i14 i15 i16 i17 i18 -add
store o11 o12 o13 o14 o15 o16 o17 o18 o19 -add
store magfs[lpf] magfs[flux] SPD[IM]-add
store ud[pi] uq[pi] iqsref[pi] idsref[pi] VAN[SPWM]-add
store if1[pi] fdsref[pi] e3[pi] sine[blasta] cosine[blasta]-add
store IA[IM] TEREF[PI] -add

algor dopri45r
error 1E-6

split 3 1
area 1 1
axes h 0 9 v 0 0.2
text 'FDSACT'
area 2 1
axes h 0 9 v 0 14000
TEXT 'SPEED'
area 3 1
axes h 0 9 v 0 150
TEXT 'Te'

plot  fdsact[PI]:1 1  SPD[IM]:2 1  testi[torque]:3 1
"get ini1
simu 0 1  1E-6/n1 20E-6
"par Tecom:146
"simu 1 3 1e-6/n2 50E-6
"par tecom:5
```

```
"SIMU 3 5 -cont/n3  50e-6
"par tecom:10
"SIMU 5 7 -CONT/n4  50e-6
"SIMU 7 9 -cont/n5  20e-6
END



"-----------------------------------------------------------------------------------------------
"     PI CONTROLLERS FOR Iqs, Ids, FLUX AND TORQUE CONTROL LOOPS
"-----------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM PI
INPUT Idsact Iqsact  Fdsact Idecoup Testi  Fr weact  vqss vdss
OUTPUT Ud Idsdec Uq Iqsref Idsref iq1
STATE x1  x2 x3 x4 i j y
DER  Dx1 Dx2 Dx3  Dx4 Di Dj Dy
TIME t

"----------COMPUTE ERRORS AND CONTROL SIGNALS
IDSDEC = IF1 + Idecoup
E1 = Idsref - Idsact
E2 = Iqsref - Iqsact
E3 = Fdsref - Fdsact
E4 = Teref - Testi
Dy = (1/Tc1)*terefl - (1/Tc1)*y
teref = y
Terefl = tecom "IF t<0.3 THEN Tecom ELSE TecomA
Tecom:0
TECOMA = IF t<0.7 THEN TECOMB ELSE TECOM1
TECOMB = 365*(t - 0.3)
Tecom1 =IF t<1.5 THEN Tecom2 ELSE Tecom3
Tecom2 =-245*t+ 317.5
Tecom3 =IF t<2 THEN Tecom4 ELSE Tecom5
Tecom4 = 100*(t-2)
Tecom5 =100 "30 "IF t>2.5 THEN Tecom6 ELSE Tecom7
"Tecom6=10*(t - 1.5) + 5
"tecom7:10

"---------- COMPUTE REFERENCE FLUX IN CONSTANT TORQUE REGION
"----------AND FIELD WEAKEING REGION.

Fdsref = IF abs(fdsfw) > fr THEN Fr ELSE Fdsfw
```

```
fdsfw = (0.63662*vi - A) /(weact + 0.00000001)
vi : 312
a:35.32 "19.303 "28.35   "CORRESPONDES TO 6000 RPM [ CRITICAL
                         "SPEEDD:6558 RPM]

Dx1 = Kid*E1
Dx2 = Kiq*E2
Dx3 = Kif*E3
Dx4 = Kit*E4

vd = Kpd*E1+X1
vq = Kpq*E2+X2
Ifl= Kpf*E3+X3
iq = Kpt*E4+X4
iqaux = If iq < -iqlim THEN -iqlim ELSE IF iq > iqlim THEN iqlim ELSE iq

Ud=IF vd<-vdlim THEN -vdlim ELSE IF vd>vdlim THEN vdlim ELSE vd
Uq=IF vq<-vqlim THEN -vqlim ELSE IF vq>vqlim THEN vqlim ELSE vq
IDSREF=IF IDSDEC<-idlim THEN -idlim ELSE IF IDSDEC>idlim THEN idlim ELSE
IDSDEC

Di = (1/Tc)*iqaux - (1/Tc)*i
iqsref = i
Dj =(1/Tc)*iqsact - (1/Tc)*j
iq1 = j

"----------REGULATOR GAIN

Kpd:0.003
Kid:0.75
Kpq:0.003
Kiq:0.75
Kpf:1000 "7000 "14000
Kif:15000 "6500 "4400
Kpt :150 "50 "150
Kit :50 "30"50

"----------LIMITER VALUES

vdlim:20 "0.75
vqlim:20 "1.2
idlim:323 "100 "75
iqlim = 400 " sqrt(560*560 - idsref*idsref) "556 "112
```

```
"---------- PARAMETERS

Tc:10e-3 "10E-3
Tc1:50e-3
"----------INITIAL VALUES

END


"-------------------------------------------------------------------------------------------------
"              STATIONARY TO SYNCHRONOUS TRANSFORMATION
"-------------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM STASYN
INPUT Idss Iqss costhe sinthe Fqss Fdss ncosthe nsinthe
OUTPUT Ids Iqs Fds Fqs

TIME t

Iqs = Iqss*ncosthe-Idss*nsinthe
o16 = iqs
Ids = Iqss*nsinthe+Idss*ncosthe
o17 = ids
Fds = Fqss*sinthe+Fdss*costhe
Fqs = Fqss*costhe-Fdss*sinthe
END


"-------------------------------------------------------------------------------------------------
"              SYNCHRONOUS TO STATIONARY TRANSFORMATION
"-------------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM SYNSTA
INPUT  Ud Uq costhe sinthe
OUTPUT Vqss Vdss
TIME t

"costhe = cos(theta)
"sinthe = sin(theta)

"----------Synchronous to Stationary D-Q Ref. Frame
```

```
Vqss = Uq*costhe + Ud*sinthe
Vdss = Ud*costhe - Uq*sinthe

END

"----------------------------------------------------------------------------------------------
"                                  2-ϕ TO 3-ϕ TRANSFORMATION
"----------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM T2F3F
INPUT Vqss Vdss
OUTPUT Vac Vbc Vcc
TIME t

"----------Stationary to ABC Ref. Frame

Vac=Vqss
Vbc=-(Vdss*sqrt(3)+Vqss)*0.5
Vcc=-(Vac+Vbc)

END

"----------------------------------------------------------------------------------------------
"                                  3-ϕ TO 2-ϕ TRANSFORMATION
"----------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM T3F2F
INPUT  Iaf Ibf Icf Vaf Vbf Vcf
OUTPUT Idss Iqss Vqss Vdss

sqr3 = sqrt(3)

Iqss = Iaf
Idss = (Icf-Ibf)/sqr3

Vqss = Vaf
Vdss = (Vcf-Vbf)/sqr3

i15 = iqss
```

```
i16 = idss

END


"----------------------------------------------------------------------------------------------
"                              SIMPLIFIED SPWM MODEL
"----------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM SPWM

"----------SINUSOIDAL PWM INVERTER

INPUT  Varef Vbref Vcref Ia Ib Ic "Vd
OUTPUT  Van Vbn Vcn Ii
TIME t
pi = 3.141592654

"---------- TRIANGLE GENERATOR

Ts = 1/fs                "Period of carrier wave
tn = MOD(t,Ts)*fs
trg = IF tn<0.25 THEN 4*tn ELSE IF tn<0.75 THEN -4*tn+2 ELSE 4*tn-4
y = amp*trg

"----------PWM GENERATOR

Vam = Varef                          "Modulating voltages
Vbm = Vbref
Vcm = Vcref

Vao = IF Vam>y THEN 1 ELSE 0     "state switches.
Vbo = IF Vbm>y THEN 1 ELSE 0     " = 1 , upper switch is on.
Vco = IF Vcm>y THEN 1 ELSE 0     " = 0 , lower switch is on.

Vdb3 = Vd/3

Van = (2*vao-vbo-vco)*Vdb3       "Output phase voltages
Vbn = (2*vbo-vao-vco)*Vdb3
```

```
Vcn = (2*vco-vao-vbo)*Vdb3

Vab = Vao-Vbo                    "Output line voltages
Vbc = Vbo-Vco
Vca = Vco-Vao

Ii = Vao*Ia + Vbo*Ib + Vco*Ic    "inverter input current.

"----------PARAMETERS
amp:1 "1.0
Vd:312
fs:15600

END



" ------------------------------------------------------------------------------------------
"   INDUCTION MOTOR D-Q MODEL WITH CORE LOSSES
" DESCRIPTION:
"       THIS ROUTINE REPRESENTS AN INDUCTION MACHINE
"       WITH THE FUNDAMENTAL CORE LOSSES INCORPORATED INTO
"       A STATIONARY FRAME D-Q MODEL.
"       NOTES:      1) USING SYNCHONOUS REFERENCE FRAME
"                   2) CURRENT-BASED MODEL
"                   3) USING A SECUNDARY MAGNETIZING INDUCTANCE.
"---------------------------------------------- ----------------------------------------------

CONTINUOUS SYSTEM IM
INPUT  va vb vc sinthe costhe we
OUTPUT Ia Ib Ic Rs Ls Tr pole vqss vdss sigma Te Lls Llr Lm Rr pin Po
STATE iqss  idss  idms  iqms  iqrms  idrms  w
DER  diqss didss didms diqms diqrms didrms dw
TIME t

"---------- DEFINE SOME USEFULL CONSTANTS

pi = atan(1)*4
p2 = pi*2
pb2s = p2*p2
pt2b3 = p2/3
sq2 = sqrt(2)
sqr3 = sqrt(3)
```

```
"WE:0
spd = wr*60/p2                "speed in rpm
wr=w*2/pole                   "wr is in mech. rad./s
wsl = we - w                  "Slip frequency

"---------- EQUATIONS FOR FLUXES AND ROTOR CURRENT.

fdlss = Lls*idss              "Stator leakage fluxes.
fqlss = Lls*iqss
fdms = Lm*idms                "Air-gap fluxes.
fqms = Lm*iqms
fdrs = fdms + Llr*idrs        "Rotor fluxes.
fqrs = fqms + Llr*iqrs

iqrs = iqrms + iqms - iqss    "Rotor currents.
idrs = idrms + idms - idss

"---------- COMPUTE AUXILIARY EXPRESSIONS

Leq3 = 1/Lm + 1/Llr
Leq2 = 1/Llm + Leq3
Leq1 = Lls*(1/Lls + Leq2)

F1qs = -vqss + Rs*iqss
F2qs = -(F1qs*Leq2 + Rm*iqrms/Llm - (-w*fdrs+Rr*iqrs)/Llr)/Leq1

F1ds = -vdss + Rs*idss
F2ds = -(F1ds*Leq2 + Rm*idrms/Llm + (-w*fqrs-Rr*idrs)/Llr)/Leq1

"---------- STATE EQUATIONS

diqss = F2qs
diqms = -(F1qs + Lls*F2qs)/Lm
diqrms = -(F1qs + Lls*F2qs + Rm*iqrms)/Llm
didss = F2ds
didms = -(F1ds + Lls*F2ds)/Lm
didrms = -(F1ds + Lls*F2ds + Rm*idrms)/Llm

dw = (te-tl-b*abs(wr)-kfw*wr*wr)*pole/(j*2)  "w is in elec. rd/s
"dw = (te -tl)/j*pole/2

"---------- VOLTAGE TRANSFORMATIONS
```

```
vqss = va                          "abc to d-q stationary
vdss = (vc-vb)/sqr3
"Vqs = vqss*costhe-vdss*sinthe     "d-q stationary to synchronous
"Vds = vqss*sinthe+vdss*costhe

"---------- COMPUTATION OF TORQUE

te = -(fdrs*iqrs-fqrs*idrs)*0.75*pole
tl = c2*wr*wr  "+ c1               "EV load, except inertia

"---------- CURRENT TRANSFORMATIONS

"iqss = costhe*iqs+sinthe*ids      "synchonous to stationary
"idss = -sinthe*iqs+costhe*ids
ia = iqss                          "d-q stat. to abc
ib = -(0.5*iqss)-(0.5*sqr3*idss)
ic = -(0.5*iqss)+(0.5*sqr3*idss)

"---------- DEFINE PARAMETERS FOR THE MACHINE MODEL

f= we/p2                                "supply frequency
s = wsl/(we+0.001)                      "Per-unit slip.
RmA =  (pb2s*f)/((1+abs(s))*Kh+(1+s*s)*Ke*f)
Rm = max(RmA,5)                    "Impose a min. value to Rm

"-------COMPUTATION OF MACHINE LOSSES, INPUT AND OUTPUT POWERS

im=sqrt(iqms*iqms+idms*idms)/sq2        "magnetizing current
irm=sqrt(iqrms*iqrms+idrms*idrms)/sq2   "current in rm
is=sqrt(iqss*iqss+idss*idss)/sq2        "stator current
ir=sqrt(iqrs*iqrs+idrs*idrs)/sq2        "rotor current

ps=3*is*is*rs                           "stator copper loss
pr=3*ir*ir*rr                           "rotor copper loss
pc=3*irm*irm*rm                         "core losses
pl=ps+pr+pc                             "total losses
Po=tl*wr                                "output power
pin=va*ia+vb*ib+vc*ic
N = (Po*100) / (pin+0.01)               "Efficiency of the machine

"---------- PARAMETERS FOR THE INDUCTION MOTOR
"---------- 100  hp MACHINE PARAMETERS FROM DELPHI ENERGY SYSTEMS
```

```
Rs=11.21E-03  "0.0085                    "Stator resistance.
Rr=12.43e-03  "0.00316                   "Rotor resistance.

ke:1.394  "0.1394                        "Eddy current coefficient.
kh:60  "30.668                           "Hysteresis coefficient.

"Lm = if Im<Imo then Lmo else Lmo - m*(Im-Imo)
Lm=2.13E-03 "2.645e-03                   "Unsatureted value.
Imo:43.77 "20.83                         "Linear region breakpoint
m:0.02116e-03                            "Saturation coefficient

Llm:8.5732E-05  "1.065e-04               "Secundary magnetizing inductance.
Lls=43.8E-06   "0.044e-03                "Stator leakage inductance.
Llr=43.8E-06  "0.037E-03                 "Rotor leakage inductance.
j :0.5 "10e20                            "freez spd "0.005:accerlate  "0.05:start
b:0.00502                                "Estimated viscous damping coefficient.
pole=4
Ls = Lls+Lm
Tr = Lr/Rr
sigma = (1-(Lm*Lm)/(Ls*Lr))
Lr = Llr+Lm

"---------- LOAD CONSTANTS

c1 = if wr<1 then 0 else 3.615           "2.5% constant load.
c2: 8e-6 "9.245e-5 "0.000001 "35.51e-06  "7.5% at speed = 0.8 pu .
kfw:2.346e-07                            "Friction and windage loss coefficient.

"---------- INITIAL VALUES
"ids:0    "63.92                         "Rated flux condition.
"idm:0     "63.92

END



"-------------------------------------------------------------------------------------------------
"          FILTERING MACHINE PHASE CURRENTS AND  VOLTAGES
"-------------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM filter
STATE y1  y2  y3  y4  y5  y6
```

```
DER  Dy1 Dy2 Dy3 Dy4 Dy5 Dy6
INPUT Ia Ib Ic Va Vb Vc                    "Phase Currents and Voltages
OUTPUT Iahe Ibhe Iche Vahe Vbhe Vche
TIME t
pi=3.14159265359

Dy1 = (1/RCcur)*Ia - (1/RCcur)*y1
Iahe = y1

Dy2 = (1/RCcur)*Ib - (1/RCcur)*y2
Ibhe = y2

Dy3 = (1/RCcur)*Ic - (1/RCcur)*y3
Iche = y3

Dy4 = (1/RCvol)*Va - (1/RCvol)*y4
Vahe = y4

Dy5 = (1/RCvol)*Vb - (1/RCvol)*y5
Vbhe = y5

Dy6 = (1/RCvol)*Vc - (1/RCvol)*y6
Vche = y6

"---------- PARAMETERS FOR FILTERING

fc1cur:1000
fc1vol:1000
RCcur=1/(2*pi*fc1cur)
RCvol=1/(2*pi*fc1vol)

"---------- INITIAL CONDITIONS

END



"-------------------------------------------------------------------------------------------------
"                              CASCADED LOW-PASS FILTER
"-------------------------------------------------------------------------------------------------

continuous system LPF
INPUT Iqss Idss iqsref Vqss Vdss Rs wbl fqbl fdbl
```

```
OUTPUT costhe sinthe MagFs Fqss Fdss wen ncosthe nsinthe
OUTPUT fdss1 fqss1

STATE y2  y3 y4  y7 y6 y8 f5 f6  w
DER  Dy2 Dy3 Dy4  Dy7 Dy6 Dy8 Df5 Df6 Dw
TIME t

xq1 = Vqss - Iqss*Rs
xd1 = Vdss - Idss*Rs
i17 = xq1
i18 = xd1
xq = g*xq1
xd = g*xd1
wen = ABS(W2)

"---------- CALCULATING PRGRAMMABLE TIME CONSTANTS AND GAINS
"---------- AS A  FUNCTION OF ωe

"---------- Tc AND G FOR 1000HZ PROGAMMABLE LOW PASS FILTER

Tcb = -1.2832565E-6 + 4.0434856E-11*wen + cx
cx = 0.57735377/(wen+eps) - 4.1367812E-5/((wen)^2+eps)
gb = -2.3749952E-6 + 1.9270465E-8*wen + 1.5395992/(wen+eps)

eps=IF abs(wen) < 1E-5 THEN 1 ELSE 0.000001 "0.00000000001

Tc = Tcb
g = gb

"---------- CALCULATE Q-AXIS STATOR FLUX

Dy2 = (1/Tc)*xq - (1/Tc)*y2
v2 = y2
Dy3 = (1/Tc)*v2 - (1/Tc)*y3
v3 =  y3
Dy4 = (1/Tc)*v3 - (1/Tc)*y4
v4 = y4

Fqss = v4
Fqlp = v4

"---------- CALCULATE D-AXIS STATOR FUX
```

```
Dy6 = (1/Tc)*xd - (1/Tc)*y6
v6 = y6
Dy7 = (1/Tc)*v6 - (1/Tc)*y7
v7 = y7
Dy8 = (1/Tc)*v7 - (1/Tc)*y8
v8 = y8

Fdlp = v8
Fdss = FDLP

"---------- CALCULATE MAGNITUDE OF STATOR FLUX

MagFs = sqrt(Fdss*Fdss + Fqss*Fqss) + 0.000000000000001

"---------- CALCULATE UNIT VECTORS

mcos = Fdss/MagFs
msin = Fqss/MagFs
costhe = IF abs(iqsref) <0.00001 THEN co ELSE mcos
sinthe = IF abs(iqsref) <0.00001 THEN s ELSE msin

"---------- ESTIMATE ANGULAR FREQUENCY

WAUX = ((Vqss - Rs*Iqss)*Fdss - (Vdss - Rs*Idss)*Fqss)/norma
norma =((Fdss*Fdss + Fqss*Fqss) + 0.000000000000001)
w1 =abs(WAUX)

Dw = (1/Tc2)*w1 - (1/Tc2)*w          "Pass "we" signal through LPF to brake the
                                     "algibric loop
w2 = w

"---------- ADD PHASE LAG OF PHI(1) TO GENERATE UNIT VECTORS FOR
"---------- INVERS VECTOR ROTATION

Df5 = (1/Tc3)*Fqss - (1/Tc3)*f5
Fqslp = f5
Fqss1 = FQSLP

Df6 = (1/Tc3)*Fdss - (1/Tc3)*f6
Fdslp = f6
Fdss1 = FDSLP

MagFs1 = sqrt(Fdss1*Fdss1 + Fqss1*Fqss1) + 0.000000000000001
```

```
scos = Fdss1/MagFs1
ssin = Fqss1/MagFs1
ncosthe = IF abs(iqsref) <0.00001 THEN co ELSE scos
nsinthe = IF abs(iqsref) <0.00001 THEN s ELSE ssin

co : 1
s : 0

"---------- PARAMETERS FOR LPF

Tc1 = IF  abs(wen) < 10 THEN 1.591549431E-5 ELSE Tcb
Tc2 : 5E-4 "10
Tc3 : 1.591549431E-4
fc:0.000001 "0.0005 "0.01
RC =1/(2*pi*fc)
Kcorre=RC
pi:3.141593

End



"----------------------------------------------------------------------------------------------------
"                    DECOUPLING CURRENT COMPUTATION
"----------------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM DECOUP
INPUT Fds sigma ids iqs Ls Tr
OUTPUT Idecoup
TIME t

Idecoup = Ls*sigma*iqs*iqs/(Fds - sigma*Ls*ids+0.00001)
o18 = idecoup

wsl = iqs*Ls/(Tr*(Fds-sigma*ids+0.000001))
o19 = wsl


END



"----------------------------------------------------------------------------------------------------
```

```
"                    TRANSITION BETWEEN B.E. AND DVC MODE
"-------------------------------------------------------------------------------------------------

DISCRETE SYSTEM SWITCH
INPUT cobla sinbla ncobla nsinbla fqbl fdbl wbl magfbl iq iqsref
INPUT colp sinlp ncolp nsinlp fqlp fdlp wlp magflp
INPUT fqbl1 fdbl1 fqss1 fdss1
OUTPUT costhe sinthe ncosthe nsinthe MagFs fqss fdss we
OUTPUT nfdss nfqss

STATE CR
NEW  NCR
TIME t
TSAMP ts
ts = t + h
h =20E-6
NCR = IF t < 1 THEN 0 ELSE CR+1

costhe = If NCR < 2500 AND ABS(IQ) < 24.288 THEN cobla ELSE colp
o13 = costhe

sinthe = IF NCR < 2500  AND ABS(IQ) < 24.288 THEN sinbla ELSE sinlp
o14 = sinthe

ncosthe = If NCR < 2500 AND ABS(IQ) < 24.288 THEN ncobla ELSE ncolp

nsinthe = IF NCR < 2500 AND ABS(IQ) < 24.288 THEN nsinbla ELSE nsinlp

MagFs = If NCR < 2500 AND ABS(IQ) < 24.288 THEN magfbl ELSE maglp
o15 = magfs

fqss = IF NCR < 2500 AND ABS(IQ) < 24.288 THEN fqbl ELSE fqlp
i11 = fqss

fdss = IF NCR < 2500 AND ABS(IQ) < 24.288 THEN fdbl ELSE fdlp
i12 = fdss

nfqss = IF NCR < 2500 AND ABS(IQ) < 24.288 THEN fqbl1 ELSE fqss1
i13 = nfqss

nfdss = IF NCR < 2500 AND ABS(IQ) < 24.288 THEN fdbl1 ELSE fdss1
i14 = nfdss
```

```
we = If NCR < 2500 AND ABS(IQ) < 24.288 THEN wbl ELSE wlp
o11 = we

A:5

END


"---------------------------------- ----------------------------------------------------------
" COMPUTATION OF AVERAGE BATTERY POWER  AND SYSTEM EFFICIENCY
" DESCRIPTION:
"       THE INSTANTANEOUS BATTERY POWER IS PASSED THROUGH A
"       SECOND ORDER LOW-PASS FILTER, THAT IS A CONVENIENT WAY TO
"       OBTAIN THE CORRECT SYSTEM TREND UNDER EFFICIENCY "
OPTIMIZATION.
"       THE FILTERED ("AVERAGED") VALUE IS THEN USED IN THE "
EFFICIENCY OPTIMIZATION ROUTINE (ANNOPT).
"---------------------------------- ----------------------------------------------------------

CONTINUOUS SYSTEM ECALC
INPUT  Pd  Po                   "Pd= Battery instantaneous power.
OUTPUT Pi                       "Po= Machine output power.
STATE  y  "X1  X2               "Pi= Average battery power.
DER  Dy  "DX1 DX2
TIME T

"-------COMPUTE AVERAGE (FILTERED) VALUE OF POWER
"DX1 = X2                       "Second order filter state equations.
"DX2 = A21*X1 + A22*X2 + Pd
"Pi = -A21*X1                   "Pi= Average battery power.
Dy = (1/Tc)*pd - (1/Tc)*y
pi = y


Nsys = 100*Po/(Pi+0.001)        "Overall system efficiency.

"------------ PARAMETERS
Tal2:10E-03                     "Filter time constant.
A21 = -1/(Tal2*Tal2)            "Filter coefficients.
A22 = -2/Tal2

Tc:0.04  "10E-03
```

```
"X1:0  "0.64                    "Initial values of state variables.
"X2:0   "-0.24

END


"------------------------------------------ ----------------------------------------
"                BATTERY MODEL WITH DC LINK CAPACITOR
"-------------------------------------------------------------------------------------

CONTINUOUS SYSTEM BTY
INPUT Ii                        "Ii=Inverter input current.
OUTPUT Vd Pd                    "Vd= DC link capacitor voltage (=Vc).
STATE Vc  Ib                    "Ib=  battery current.
DER  DVc Dib                    "Pd= battery instantaneous power.
TIME t

"---------- STATE EQUATIONS.

DVc = (Ib-Ii)/Cd                "Capacitor voltage
DIb = (Vb-Vc-Rb*Ib)/Ld          "Current through stray inductance Ld.
Vd = Vc                         "  (Also battery current)

"---------- INSTANTANEOUS POWER

Pd = Vd*Ib                      "Instantaneous battery power.

"---------- INITIAL VALUES

Vc:312                          "Assume Vc(0) = Vb.
"Ib:0.15
"---------- PARAMETERS.

Vb:312                          "Battery average voltage.
Cd:0.0053                       "DC link capacitor.
Ld:10e-06                       "DC link stray inductance.
Rb:0.156                        "Battery internal resistance (including
                                " interconections).
END
```

```
"------------------------------------------------------------------------------------------
"                                TORQUE COMPUTATION
"------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM TORQUE
INPUT MagFs iqsact fdss fqss idss iqss iqs ids fqs fds
OUTPUT Testi
TIME t

testi1 = k*(fdss*iqss -fqss*idss)
testi = IF t<1 THEN 0 ELSE testi1
"testi = k*(fds*iqs-fqs*ids)
o12=testi
k:3

END



"------------------------------------------------------------------------------------------
"                          BLASCHKE MODE COMPUTATIONS
"------------------------------------------------------------------------------------------

continuous system BLASTA
INPUT iqss idss vqss vdss iqsref iqsder idsder Lls Llr Lm Rr Rs Tr
OUTPUT cosine sine ncosine nsine MagFs wbl Fqss Fdss fqbl1 fdbl1
STATE x y z1 z2
DER Dx Dy Dz1 Dz2

Dx = -(1/Tr)*x + A*iqsder + B*iqss
Dy = -(1/Tr)*y + A*idsder + B*idss

MagFs = sqrt(x*x + y*y) + 0.0000000000001

"mcos = y/MagFs
"msin = x/MagFs

the = arcsin(x/magfs)
cosine = cos(the)
sine = sin(the)

"cosine = IF abs(iqsref) < 0.0000001 THEN co ELSE mcos
"sine = IF abs(iqsref) < 0.0000001 THEN s ELSE msin
```

```
Fqss = x
Fdss = y

wbl = ((vqss-Rs*iqss)*y - (vdss-Rs*idss)*x)/(norma)
norma = (x*x+y*y) + 0.000000000001

Dz1 = (1/Tc)*x - (1/Tc)*z1
x1=z1
Dz2 = (1/Tc)*y - (1/Tc)*z2
y1 = z2

MagFs1 = sqrt(x1*x1 + y1*y1) + 0.0000000000001
Fqbl1 = x1
Fdbl1 = y1

"nsin = x1/MagFs1
"ncos = y1/MagFs1
nthe = arcsin(x1/magfs1)
ncosine = cos(nthe)
nsine = sin(nthe)
"ncosine = IF abs(iqsref) < 0.0000001 THEN co ELSE ncos
"nsine = IF abs(iqsref) < 0.0000001 THEN s ELSE nsin

A = Lls + (Lm*Llr)/Lr
B = (Ls*Rr)/Lr
Ls = Lls + Lm
Lr = Llr + Lm
co:1
s:0

Tc = 1.59154943E-4

END




"----------------------------------------------------------------------------------------------------
"DISCRETE MODEL TO COMPUTE DERIVATIVE OF CURRENT COMPONENTS
"----------------------------------------------------------------------------------------------------
```

```
Discrete system CURDER
INPUT iqss idss
OUTPUT iqsder idsder
STATE x y
NEW nx ny
TIME t
TSAMP ts
ts = t + h

nx = iqss
ny = idss
iqsder = (iqss - x)/h
idsder = (idss - y)/h
h = 100E-6 "50E-6

END



"----------------------------------------------------------------------------------------------------
"              NEURO-FUZZY EFFICIENCY OPTIMIZATION ROUTINE
" DESCRIPTION
"       THIS ROUTINE DERIVES THE STEP CHANGE FOR THE
"       STATOR FLUX COMMAND (DELTA Fds* or DFds) BASED ON
"       MEASURED CHANGES IN INPUT (BATTERY) POWER (DPi), AND THE
"       PAST CONTROL ACTION (LAST DELTA Fds*), TRHOUGH ANN.
"----------------------------------------------------------------------------------------------------

DISCRETE SYSTEM ANNOPT
INPUT  We Pi TLest "FSS                    "FSS = steady state flag (1-->ss, 0-->tr )
OUTPUT fl
STATE  Po Dfds fs CR
NEW   NPo NDfds Nfs NCR
TIME T
TSAMP TS

TS=T+DT
FSS:1
NCR = IF FSS<1 THEN 0 ELSE  CR+1       "Counter of eff. opt. steps.

"------------ DEFINE BASE VALUE FOR DFds (GFD)
GfD1 = C1*We/Werated + C2*TLest/Trated + C3                "Preliminary value.
GFDA = GFD1*GFDAUX
```

```
GFDAUX:1.5  "0.5
GfD =  IF ABS(GfDA) < TOL4 THEN SIGN(GfDA)*TOL4 ELSE GfDC     "MIN
GFDGfDC = IF ABS(GfDA) > TOL5 THEN SIGN(GfDA)*TOL5 ELSE GfDA
"MAX GFD

"---------- COMPUTE POWER GAIN (1/Pbase) AND SCALED DELTA Pi (DPi)

Gpb1 = -34E-03*We/Werated + 42.55E-03          "Preliminary value
GPb = MIN( GPbmax, MAX(GPBmin,GPbA))          "Final (bounded) value.
GPbA = GPAUX*GPb1
GPAUX :2
NPo  = Pi                                     "Po=Pi(k-1), dc link power.
DPi = (NPo - Po) * Gpb                        "Delta Pi.

I1 = DPI * 0.8
I2 = LDFDS * 0.8

"---------- EVALUATE CFd USING ANN

X1 = I1*W42 + I2*W43 + W41
X2 = I1*W52 + I2*W53 + W51
X3 = I1*W62 + I2*W63 + W61
X4 = I1*W72 + I2*W73 + W71
X5 = I1*W82 + I2*W83 + W81
X6 = I1*W92 + I2*W93 + W91
X7 = I1*W102 + I2*W103 + W101
X8 = I1*W112 + I2*W113 + W111
X9 = I1*W122 + I2*W123 + W121
X10 = I1*W132 + I2*W133 + W131

"---------- COMPUTE OUTPUT OF HIDDEN LAYER

S1 = TANH(X1)
S2 = TANH(X2)
S3 = TANH(X3)
S4 = TANH(X4)
S5 = TANH(X5)
S6 = TANH(X6)
S7 = TANH(X7)
S8 = TANH(X8)
S9 = TANH(X9)
S10 = TANH(X10)
```

```
"---------- COMPUTE OUTPUT LAYER SIGNAL

O1 = S1*W144 + S2*W145 + S3*W146 + S4*W147 + S5*W148 + S6*W149 + O2
O2 = S7*W1410 + S8*W1411 + S9*W1412 + S10*W1413 + W141

O3 = TANH(O1)

FV = O3/0.8                                              "Preliminary DFds.

Cfd = IF NCR<1 THEN 0 ELSE IF NCR<2 THEN -1 ELSE FV      "At first iteration,
                                                         "assume DFds=NB.
"---------- CALCULATION OF NEW Dfds
NDfds = Cfd                                              "Per-unit Dfds.
LDFDS = SIGN(DFDS)*1
DAUX = NDfDS*GfD
DfDEC = IF ABS(DAUX)<TOL2 THEN SIGN(DAUX)*TOL2*FSS ELSE DAUX
                "Impose a min. value to Dids, to keep the search active
"faux = IF abs(FSS) < 1 THEN fr ELSE fs
Nfs = dfdec + fs  "faux
f1 = Nfs

"---------- PARAMETERS

DT:1.5                                   "Sampling interval of the discrete systems

C1:0.0063333 "0.011109                   "Coefficients for gid comutation
C2:-0.0180948 "-0.03174
C3:0.0090474 "0.01587

GPbmax:0.1   "33.3E-03                   "Max and min values for power gain
GPbmin:8.33E-03

Trated:148.5  "144.686                   "Rated motor torque
werated:1466                             "Rated wr in elec. Rad./s
TOL2: 3.00E-03 "5.29E-04                 "Min. Change for Dfds
TOL4: 2.645E-03                          "Min. Value for Gfd
TOL5: 0.1 "0.01 "7.935E-03               "Max. Value for Gfd

"----------WEIGHTS FOR ANN CONTROLLER

W41: 0.1181
W42: -0.3870
W43: 0.0253
```

W51: 0.1020
W52: -0.6457
W53: 0.0268

W61: 0.1842
W62: -1.7023
W63: -0.0938

W71: 0.1837
W72: -0.3883
W73: 0.0224

W81: 0.5796
W82: 0.9686
W83: 1.4316

W91: 0.1244
W92: -0.3939
W93: -0.0456

W101: -0.5359
W102: -0.6796
W103: -0.8491

W111: 0.1054
W112: -0.2612
W113: 0.0105

W121: 0.0519
W122: 0.0756
W123: 0.1036

W131: 1.7063
W132: -1.8764
W133: 2.6004

W141: -0.1055
W144: 0.0904
W145: 0.1028
W146: 0.4626
W147: 0.0485
W148: 0.6649
W149: 0.1059

```
W1410: -0.3133
W1411: 0.0859
W1412: -0.0124
W1413: -0.9438

END


"-----------------------------------------------------------------------------------------------
"              RAMP THE OUTPUT OF THE NEURO-FUZZY CONTROLLER
"-----------------------------------------------------------------------------------------------

CONTINUOUS SYSTEM RAMP
INPUT fl
OUTPUT fs
TIME t
STATE R s
DER DR Ds


DR = (1/Tc)*fl - (1/Tc)*R
"fs = R
Ds = (1/Tc)*R - (1/Tc)*s
fs = s

Tc = 0.3

"---------- INITIAL CONDITION
"R : 0.095

END


"-----------------------------------------------------------------------------------------------
"                              TRANSITION CONTROL ROUTINE
" Notes: 1)The steady state flag (FSS) indicates the current state:
"          FSS = 1: Steady state mode.
"          FSS = 0: Transient mode.
"         2)The transitions are dependent on speed variation and changes in
"        command iqs*.
"-----------------------------------------------------------------------------------------------
```

```
DISCRETE SYSTEM TRNCTRL

INPUT iqec p testi
OUTPUT FSSA
STATE FSS  Iqs  pin  DIqs te                "Define DIqs as state var. to break alg. loop.
NEW  NFSS NIqs Npin NDIqs Nte

TIME T
Tsamp Ts
Ts = T + AT

"---------- DEFINES CHANGES IN Wr AND iqs*

Niqs = iqec
NDIqs = ABS(NIqs - IQs)                    "Delta iqs*
Npin = p
Dpin = ABS(Npin - pin)                     "Delta pin

Nte = testi
Dte = ABS(Nte - te)

"---------- DEFINES FLAG FOR STEADY STATE CONDITION

"M1 = DELAY(DIqs,TW)
"M2 = DELAY(M1,TW)
"M3 = DELAY(DPIN,TW)
"M4 = DELAY(M3,TW)

"SUM1 = ABS(DIQS) + ABS(M1) + ABS(M2)
"SUM2 = ABS(DPIN) + ABS(M3) + ABS(M4)

"AVIQ = SUM1/3
"AVPI = SUM2/3

NFSS = IF iqec<0 THEN 0 ELSE IF FSS THEN TOEX ELSE TOST
TOST = IF DIQS < E1 AND DPIN < E2 THEN 1 ELSE 0  "Test to swithc to ss mode.
TOEX = IF DIQS > E3 OR  DPIN > E4 THEN 0 ELSE 1   "Test to exit ss mode.
FSSA = FSS

"NCR = IF FSS < 1 THEN 0 ELSE CR+1

"---------- PARAMETERS
E1:4   "0.27                  "Delta iqs to enter SS mode.(0.1% of iqs rated = 0.27)
```

```
E2:3  "20  "5          "Delta Wr to enter SS mode (0.2% of pin rated = 1.38)
E3:4.5  "0.32          "Delta iqs to exit SS mode (0.12% of iqs rated = 0.32)
E4:4  "25  "7          "Delta Wr to exit SS mode (0.24% of Wr rated = 1.65)
AT:10E-03              "Sampling time.

TW:10E-03

"---------- INITIALS VALUES
"iqs:-1
"pin:-2

END



"------------------------------------------------------------------------------------------
"                                  CONNECTING SYSTEM
"------------------------------------------------------------------------------------------

CONNECTING SYSTEM CSTART
TIME t

Varef[SPWM] = Vac[T2F3F]
Vbref[SPWM] = Vbc[T2F3F]
Vcref[SPWM] = Vcc[T2F3F]
Ia[SPWM] = Ia[IM]
Ib[SPWM] = Ib[IM]
Ic[SPWM] = Ic[IM]
Vd[SPWM] = Vd[BTY]

Vqss[T2F3F] = Vqss[SYNSTA]
Vdss[T2F3F] = Vdss[SYNSTA]

Va[IM]=VAN[SPWM]
Vb[IM]=VBN[SPWM]
Vc[IM]=VCN[SPWM]
costhe[IM] = costhe[SWITCH]
sinthe[IM] = sinthe[SWITCH]
we[IM] = we[SWITCH]

Va[FILTER]=VAN[SPWM]
Vb[FILTER]=VBN[SPWM]
Vc[FILTER]=VCN[SPWM]
```

```
Ia[FILTER]=Ia[IM]
Ib[FILTER]=Ib[IM]
Ic[FILTER]=Ic[IM]

Iaf[T3F2F] = Iahe[FILTER]
Ibf[T3F2F] = Ibhe[FILTER]
Icf[T3F2F] = Iche[FILTER]
Vaf[T3F2F] = Vahe[FILTER]
Vbf[T3F2F] = Vbhe[FILTER]
Vcf[T3F2F] = Vche[FILTER]

Vdss[LPF] = Vdss[T3F2F]
Vqss[LPF] = Vqss[T3F2F]
Idss[LPF] = Idss[T3F2F]
Iqss[LPF] = Iqss[T3F2F]
Rs[LPF] = Rs[IM]
iqsref[LPF] = iqsref[PI]
wbl[LPF] = wbl[BLASTA]
fqbl[LPF] = fqss[BLASTA]
fdbl[LPF] = fdss[BLASTA]

Uq[SYNSTA] = Uq[PI]
Ud[SYNSTA] = Ud[PI]
costhe[SYNSTA] = costhe[SWITCH]
sinthe[SYNSTA] = sinthe[SWITCH]

Idss[STASYN] = Idss[T3F2F]
Iqss[STASYN] = Iqss[T3F2F]
Fqss[STASYN] = Fqss[SWITCH]
Fdss[STASYN] = Fdss[SWITCH]

costhe[STASYN] = costhe[SWITCH]
sinthe[STASYN] = sinthe[SWITCH]
ncosthe[STASYN] = ncosthe[SWITCH]
nsinthe[STASYN] = nsinthe[SWITCH]

Idecoup[PI] =Idecoup[DECOUP]
Fr[PI] = 0.1419 "fs[RAMP]  "0.1355
Fdsact[PI] = MagFs[SWITCH]
Idsact[PI] = Ids[StaSyn]
Iqsact[PI] = Iqs[StaSyn]
Testi[PI] = Testi[TORQUE]
vqss[PI] = vqss[T3F2F]
```

vdss[PI] = vdss[T3F2F]
weact[PI] = we[SWITCH]

Fds[DECOUP] = Fds[StaSyn]
sigma[DECOUP] = sigma[IM]
Ids[DECOUP] = Ids[StaSyn]
Iqs[DECOUP] = Iqs[StaSyn]
Ls[DECOUP] = Ls[IM]
Tr[DECOUP] = Tr[IM]

iqss[BLASTA] = iqss[T3F2F]
idss[BLASTA] = idss[T3F2F]
iqsder[BLASTA] = iqsder[CURDER]
idsder[BLASTA] = idsder[CURDER]
Lls[BLASTA] = Lls[IM]
Llr[BLASTA] = Llr[IM]
Lm[BLASTA] = Lm[IM]
Rr[BLASTA] = Rr[IM]
Rs[BLASTA] = Rs[IM]
Tr[BLASTA] = Tr[IM]
vqss[BLASTA] = vqss[T3F2F]
vdss[BLASTA] = vdss[T3F2F]
Iqsref[BLASTA] = Iqsref[PI]

iqss[CURDER] = iqss[T3F2F]
idss[CURDER] = idss[T3F2F]

colp[SWITCH] = costhe[LPF]
sinlp[SWITCH] = sinthe[LPF]
cobla[SWITCH] = cosine[BLASTA]
sinbla[SWITCH] = sine[BLASTA]
Magfbl[SWITCH] = MagFs[BLASTA]
wbl[SWITCH] = wbl[BLASTA]
wlp[SWITCH] = wen[LPF]
magflp[SWITCH] = MagFs[lpf]
ncolp[SWITCH] = ncosthe[LPF]
nsinlp[SWITCH] = nsinthe[LPF]
ncobla[SWITCH] = ncosine[BLASTA]
nsinbla[SWITCH] = nsine[BLASTA]
Fqbl[SWITCH] = Fqss[BLASTA]
Fdbl[SWITCH] = Fdss[BLASTA]
Fqlp[SWITCH] = Fqss[LPF]

```
Fdlp[SWITCH] = Fdss[LPF]
Fqbl1[SWITCH] = Fqbl1[BLASTA]
Fdbl1[SWITCH] = Fdbl1[BLASTA]
Fqss1[SWITCH] = Fqss1[LPF]
Fdss1[SWITCH] = Fdss1[LPF]
iq[SWITCH] = iq1[PI]
iqsref[switch] = iqsref[pi]

we[annopt] = we[SWITCH]
pi[annopt] = pi[ECALC]
Tlest[annopt] = Testi[TORQUE]
"FSS[annopt] = FSSA[trnctrl]

fl[RAMP] = fl[annopt]

testi[trnctrl] = testi[torque]
iqec[trnctrl] = iqsref[PI]
p[trnctrl] = pi[ECALC]

MagFs[TORQUE] = MagFs[SWITCH]
iqsact[TORQUE] = iqsact[PI]
fdss[TORQUE] = fdss[SWITCH]
fqss[TORQUE] = fqss[SWITCH]
iqss[TORQUE] = iqss[T3F2F]
idss[TORQUE] = idss[t3f2f]
fqs[torque] = fqs[stasyn]
fds[torque] = fds[stasyn]
iqs[torque] = iqs[stasyn]
ids[torque] = ids[stasyn]

Ii[BTY] = Ii[SPWM]

Pd[ECALC] = Pd[BTY]
Po[ECALC] = Po[IM]

END
```

```
"----------------------------------------------------------------------------------------------------
"                              INITIAL CONDITIONS FILE FORMAT
"----------------------------------------------------------------------------------------------------
```

"SPWM T=1 TE =0
[PI]
x1: 5.69026E-3
x2: -3.36771E-4
x3: 65.2969

x4: 2.11142E-7
i: 6.33427E-5
j: 5.79423E-2

y: 0.

[STASYN]
[SYNSTA]
[T2F3F]
[T3F2F]
[SPWM]
[IM]
iqss: -33.1861
idss: 57.2846
idms: 56.5574

iqms: -32.6609
iqrms: 3.2463E-2
idrms: -5.60714E-2
w: 5.96353E-3

[FLUX]
y1: -7.08387E-2
y2: 0.122951
X1: 5.4189E-4

z1: -7.08318E-2
z2: 0.122938

[filter]
y1: -33.2765
y2: -33.1027
y3: 66.3791

y4: -0.339849
y5: -0.339853
y6: 0.679701

[LPF]
y2: -3.24147E-2
y3: -5.68958E-2
y4: -4.98005E-2
y7: 9.86026E-2
y6: 5.68269E-2
y8: 8.62273E-2

f5: -4.98002E-2
f6: 8.62267E-2
w: 8.49479E-2

[DECOUP]
[SWITCH]
CR: 0.

[ECALC]
y: 73.7142

[BTY]
Vc: 311.963
Ib: 0.242975

[TORQUE]
[BLASTA]
x: -7.10012E-2
y: 0.122966
z1: -7.09977E-2
z2: 0.12296

[CURDER]
x: -33.183
y: 57.2706

[ANNOPT]
Po: 0.
Dfds: 0.
fs: 0.

CR: 0.

[RAMP]
R: -1.30652E-2

s: -1.14142E-2
[TRNCTRL]
FSS: 1.
Iqs: 0.
pin: 73.9018

DIqs: 0.
te: 0.

[CSTART]

APPENDIX D

Matlab-Simulink Simulation Block Diagrams for the Drive System

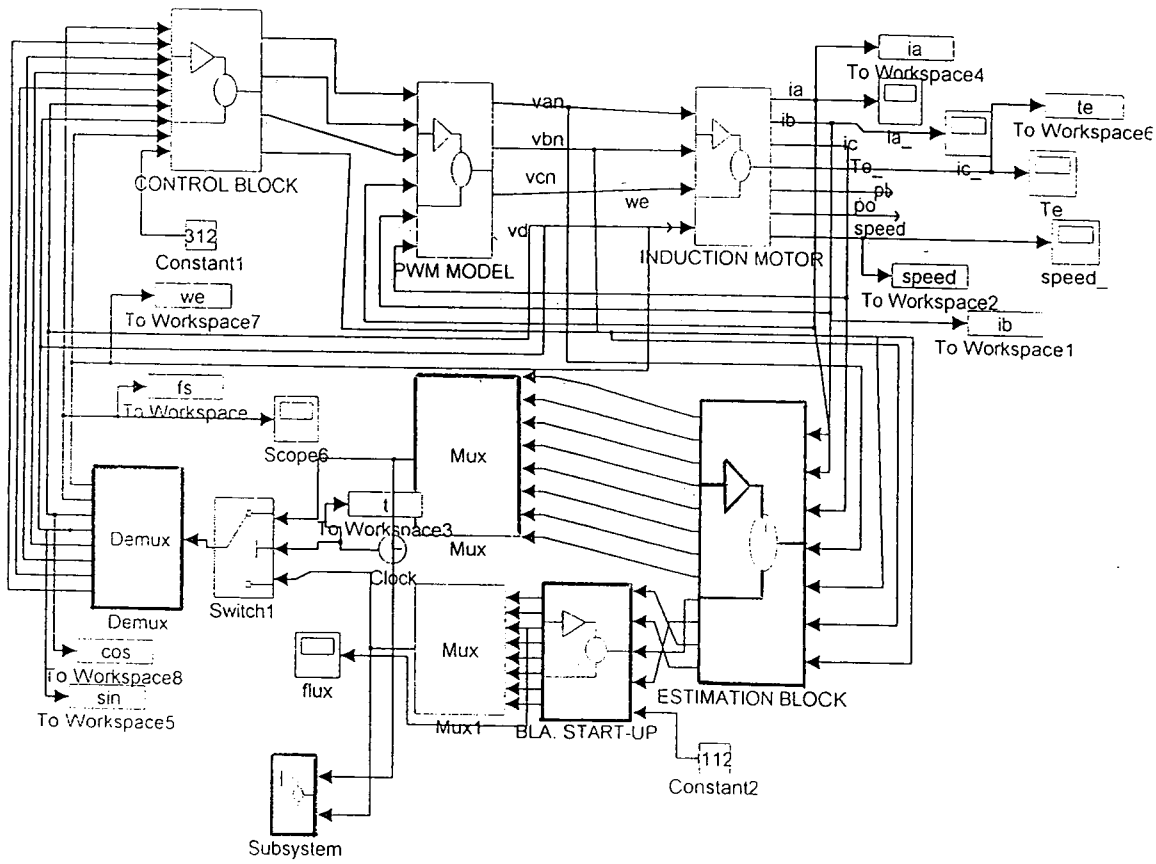


Figure 1 Complete system block diagram

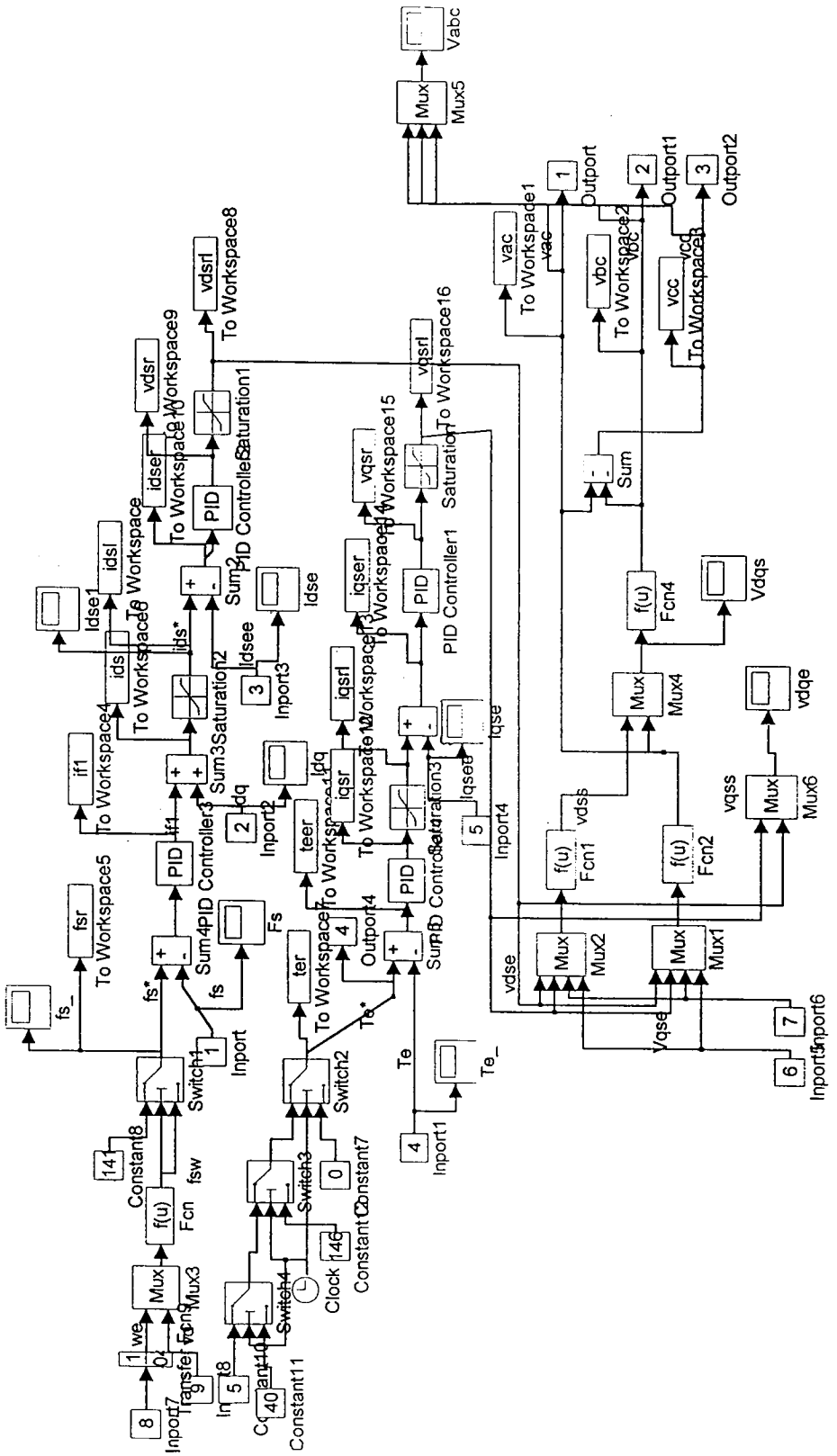


Figure 2 PI control block diagram

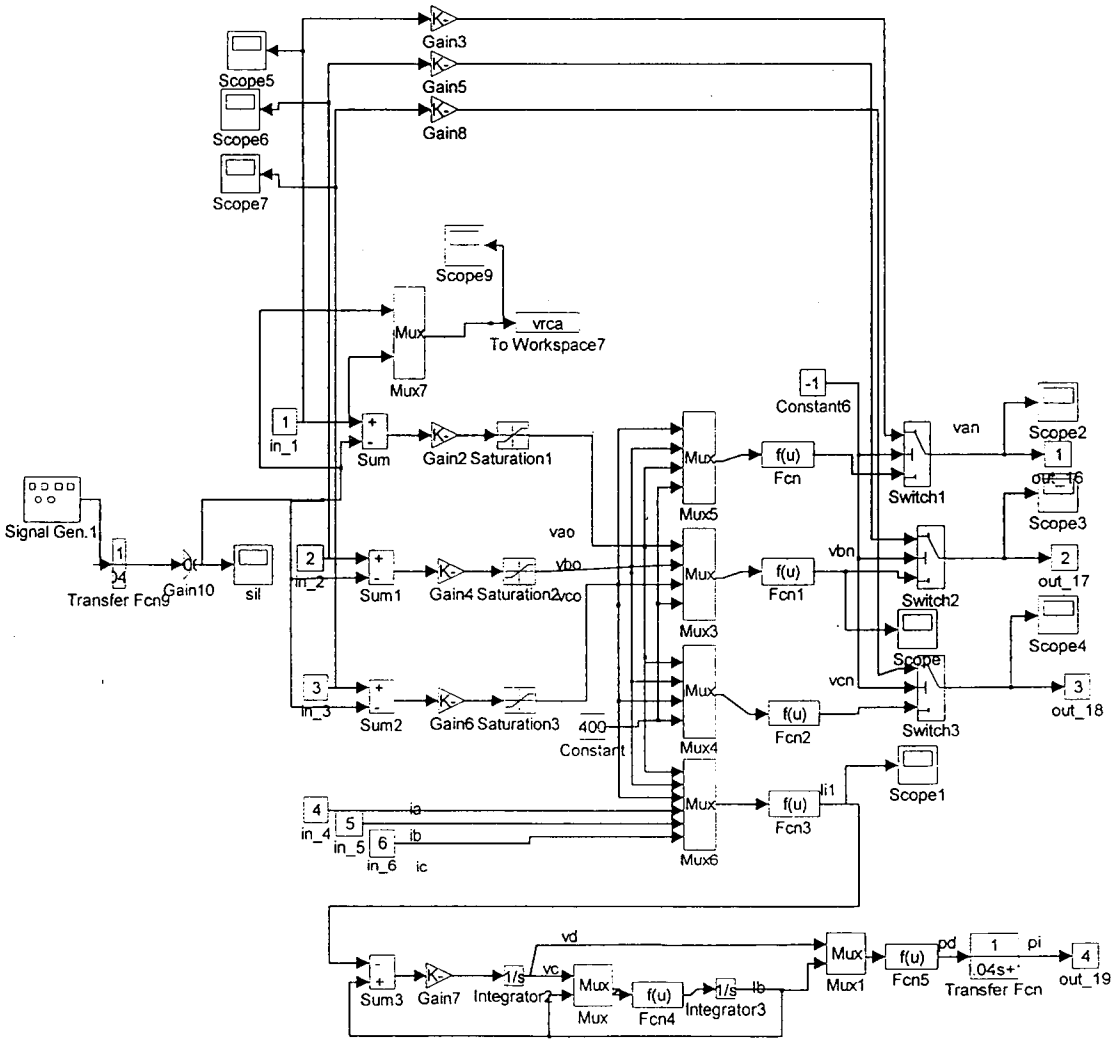


Figure 3 PWM control block diagram

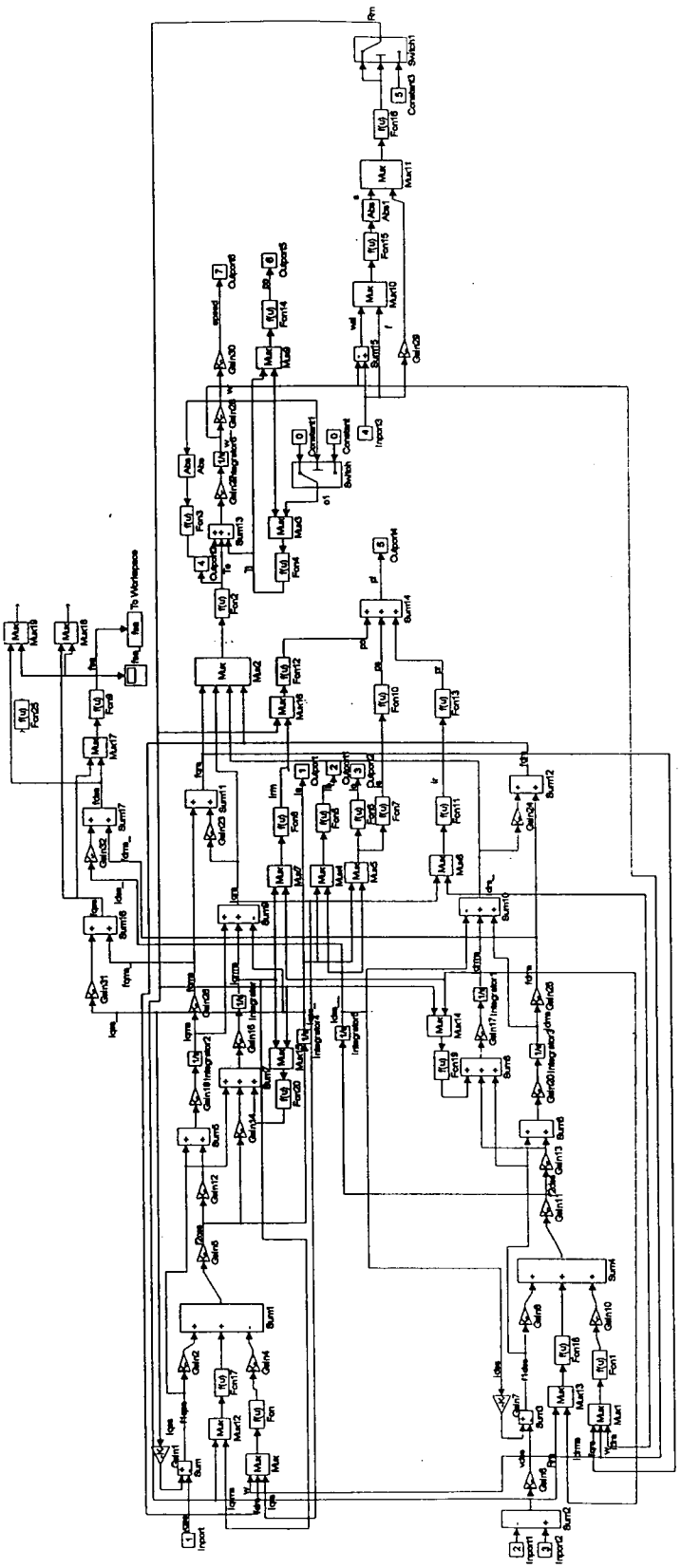


Figure 4 Lossy induction machine block diagram

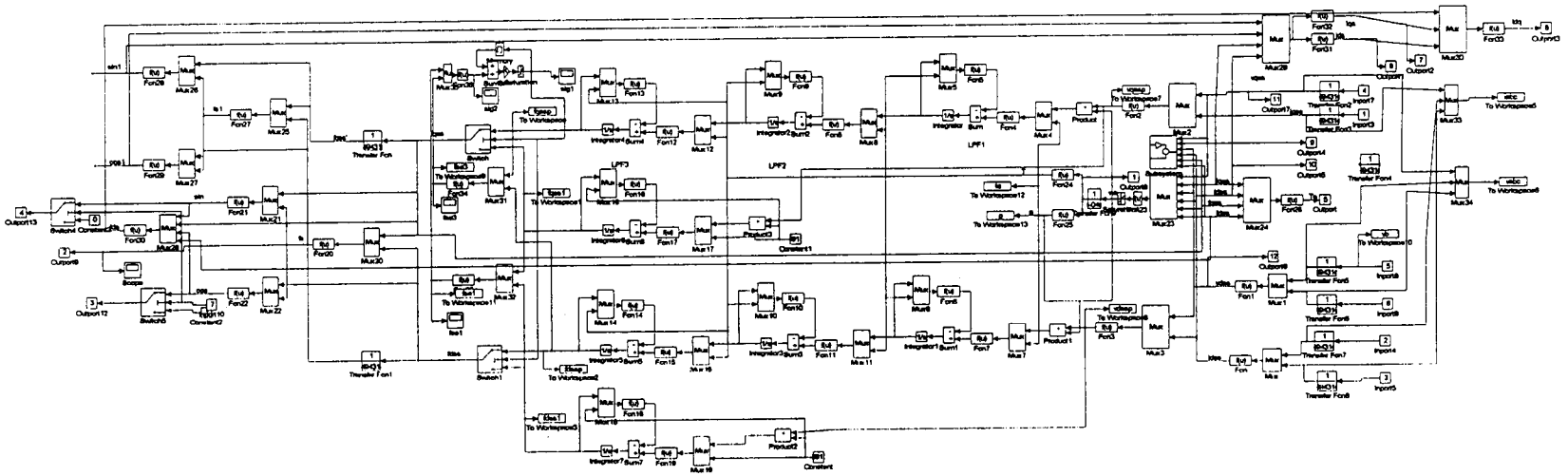


Figure 5 Feedback signal estimation block diagram in DVC mode

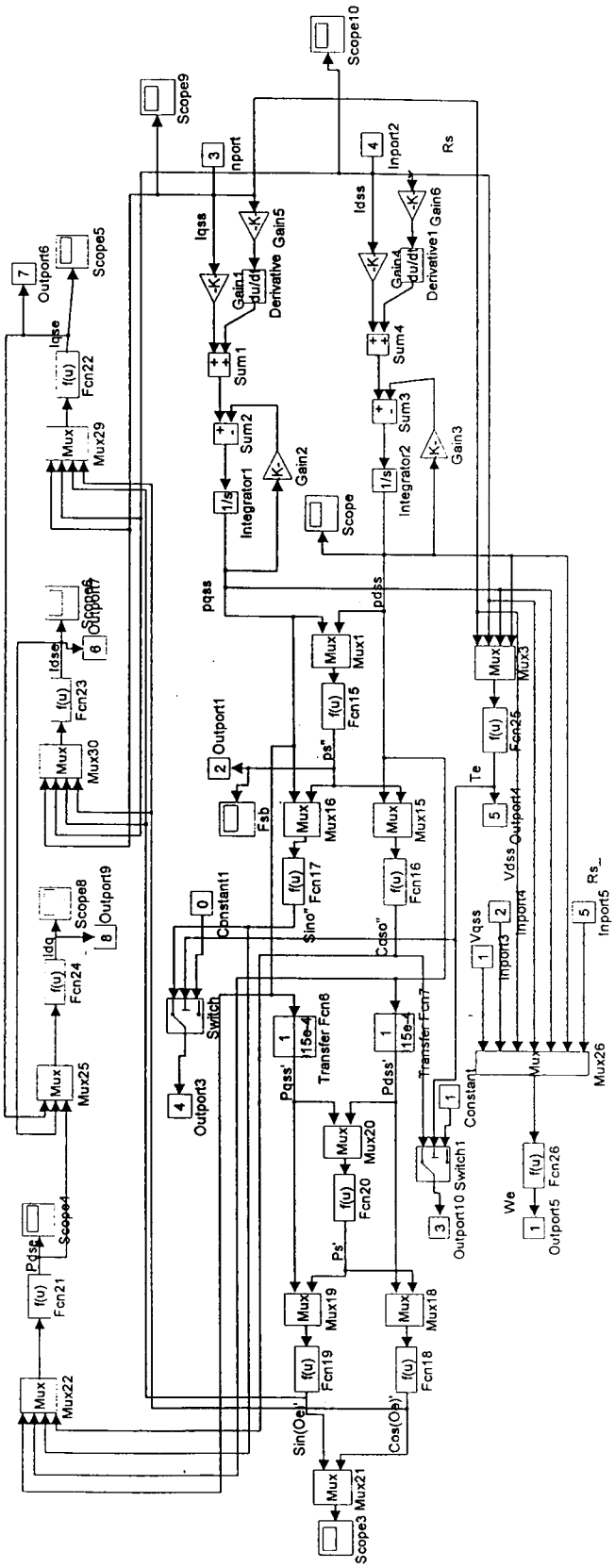


Figure 6 Feedback signal estimation block diagram in Blaschke mode

APPENDIX E

Simulation of Quasi-Fuzzy Stator Resistance Estimator using C-Language

```
/*****************************************************************/
/*************  Fuzzy Rs estimator for 5 Hp Machine  *************/
/*** Description;                                                */
/***   This program estimate the stator resistance using quasi-fuzzy  */
/***   estimator.                                                */
/***   INPUT: We(rad/sec) and Is(Amp)                            /
/***   OUTPUT: Delta Ts                                          */
/***   FUZZY SUBSETS:                                            */
/***         We: # 8,  Is: #  9,  Delta Ts: # 12.                */
/***   FUZZY RULES:   72                                         */

/***   Author: Nitin Patel                    */
/*****************************************************************/

#include<stdio.h>
#include<stdlib.h>
#include<math.h>
#include<string.h>
#include <graphics.h>
#include <conio.h>
#include "gtool.c"


#define MIN(a,b) (((a) < (b)) ? (a) : (b))
#define MAX(a,b) (((a) > (b)) ? (a) : (b))
#define Tmax 50
#define Tmin 0.0
#define Td   0.001024

float  wcom,iscom;
float WE,IS;
float W1,W2,W3,W4,W5,W6,W7,W8;
float IS1,IS2,IS3,IS4,IS5,IS6,IS7,IS8,IS9;
float J1,I1,L1,N1;
float MW1,MW2,MIS1,MIS2;
float MRA,MRB,MRC,MRD;
float R11,R12,R21,R22;
float C1,C2,C3,C4,C5,C6,C7,C8,C9;
```

```
float D1,D2,D3,D4,D5,D6,D7,D8,D9;
float SUM,DT,TS1;
float TSD,TSD_OLD;
float TSD1;
float t,RS;
float   is_cont = 3.6;

void graph(float t, float iscom, float RS);

void main()
 {
int     mode, key;
TSD_OLD = 0;
/*      iscom = 100;*/
// fp1 = fopen("input","w");
// fp2 = fopen("optput","w");
  intgraphic();
TSD1 = 0;
TERROR = 0;
CR = 0;
CR1 = 0;

for(t = 1; t < Tmax ; t += Td)
{
if(t < 15)
iscom = 4;
else if(t < 30)
iscom = 6;
else if(t < 45)
iscom = 8;
else if(t < 57)
iscom = 10;
else
iscom = 12;
/*else if(t < 75)
iscom = 6;
else if(t < 90)
iscom = 10;
else
iscom = 4;*/
/*iscom = 3.6*t;*/
wcom = 74.77;
WE = wcom/733;
```

```
IS = iscom/20;

/*"***** Compute preliminary values of degree of membership function
"     of we and  Is  */
        W1 = MIN(1, WE*23.33);
        W2 = (WE*12.5-0.53575);
        W3 = (WE*8.0619-0.9905);
        W4 = (WE*11.94-2.948);
        W5 = (WE*11.71-3.871);
        W6 = (WE*7.88-3.278);
        W7 = MIN(1,(WE*2.188-1.188));


        IS1 = MIN(1,IS*5);
        IS2 = (IS*10-2);
        IS3 = (IS*10-3);
        IS4 = (IS*10-4);
        IS5 = (IS*10-5);
        IS6 = (IS*10-6);
        IS7 = (IS*10-7);
        IS8 =MIN(1,(IS*10-8));

/*"******** Get interval indices J and I, for WE and IS respectively*/

        if (WE < 0.04286) {
                J1 = 1;
        } else if (WE < 0.12286) {
                J1 = 2;
        } else if (WE < 0.2469) {
                J1 = 3;
        } else if (WE < 0.3306) {
                J1 = 4;
        } else if (WE < 0.416) {
                J1 = 5;
        } else if (WE < 0.5429) {
                J1 = 6;
        } else if (WE < 0.6712) {
                J1 = 7;
        } else {
                J1 = 8;
        }

        if (IS < 0.2) {
```

```
        I1 = 1;
} else if (IS < 0.3) {
        I1 = 2;
} else if (IS < 0.4) {
        I1 = 3;
} else if (IS < 0.5) {
        I1 = 4;
} else if (IS < 0.6) {
        I1 = 5;
} else if (IS < 0.7) {
        I1 = 6;
} else if (IS < 0.8) {
        I1 = 7;
} else if (IS < 0.9) {
        I1 = 8;
} else {
        I1 = 9;
}/* else {
        I1 = 10;
} */

/*"******** Cmpute degree of membership function for relevant fuzzy sets*/

if (J1 < 2) {
        MW1 = W1;
} else if (J1 < 3) {
        MW1 = W2;
} else if (J1 < 4) {
        MW1 = W3;
} else if (J1 < 5) {
        MW1 = W4;
} else if (J1 < 6) {
        MW1 = W5;
} else if (J1 < 7) {
        MW1 = W6;
} else {
        MW1 = W7;
}

MW2 = 1 - MW1;

if (I1 < 2) {
        MIS1 = IS1;
```

```
        } else if (I1 < 3) {
                MIS1 = IS2;
        } else if (I1 < 4) {
                MIS1 = IS3;
        } else if (I1 < 5) {
                MIS1 = IS4;
        } else if (I1 < 6) {
                MIS1 = IS5;
        } else if (I1 < 7) {
                MIS1 = IS6;
        } else if (I1 < 8) {
                MIS1 = IS7;
        } else  {
                MIS1 = IS8;
        } /*else {
                MIS1 = IS9;
        }  */

        MIS2 = 1-MIS1;

/*"********* Perform the rule base evaluation*/

        MRA = MIN(MW2,MIS2);
        MRB = MIN(MW1,MIS2);
        MRC = MIN(MW2,MIS1);
        MRD = MIN(MW1,MIS1);

/*"********* Retrive the Delta Ts from the rule tabel for each fired rule
"First Rule:J1=1,I1=1*/



        if (I1 < 2) {
                C1 = 0.0201;
                C2 = 0.0201;
                C3 = 0.0079;
                C4 = 0.0079;
                C5 = 0.0079;
                C6 = 0.0079;
                C7 = 0.0079;
                C8 = 0.0079;
        } else if (I1 < 3) {
                C1 = 0.0342;
```

```
            C2 = 0.0342;
            C3 = 0.0201;
            C4 = 0.0201;
            C5 = 0.0079;
            C6 = 0.0079;
            C7 = 0.0079;
            C8 = 0.0079;
        } else if (I1 < 4) {
            C1 = 0.0601;
            C2 = 0.0601;
            C3 = 0.0342;
            C4 = 0.0342;
            C5 = 0.0201;
            C6 = 0.0201;
            C7 = 0.0079;
            C8 = 0.0079;
        } else if (I1 < 5) {
            C1 = 0.0937;
            C2 = 0.0937;
            C3 = 0.0601;
            C4 = 0.0601;
            C5 = 0.0342;
            C6 = 0.0342;
            C7 = 0.0201;
            C8 = 0.0201;
        } else if (I1 < 6) {
            C1 = 0.1998;
            C2 = 0.1397;
            C3 = 0.0937;
            C4 = 0.0937;
            C5 = 0.0601;
            C6 = 0.0601;
            C7 = 0.0342;
            C8 = 0.0342;
        } else if (I1 < 7) {
            C1 = 0.2346;
            C2 = 0.1998;
            C3 = 0.1397;
            C4 = 0.1397;
            C5 = 0.0937;
            C6 = 0.0937;
            C7 = 0.0601;
            C8 = 0.0601;
```

```
} else if (I1 < 8) {
        C1 = 0.42;
        C2 = 0.35;
        C3 = 0.1998;
        C4 = 0.1998;
        C5 = 0.1397;
        C6 = 0.1397;
        C7 = 0.0937;
        C8 = 0.0601;
} else {
        C1 = 0.6;
        C2 = 0.42;
        C3 = 0.35;
        C4 = 0.2346;
        C5 = 0.2346;
        C6 = 0.1998;
        C7 = 0.1397;
        C8 = 0.0937;
}

if (J1 < 2) {
        R11 = C1;
} else if (J1 < 3) {
        R11 = C2;
} else if (J1 < 4) {
        R11 = C3;
} else if (J1 < 5) {
        R11 = C4;
} else if (J1 < 6) {
        R11 = C5;
} else if (J1 < 7) {
        R11 = C6;
} else {
        R11 = C7;
}

/*"******** Second Rule: J1=2,I1=1*/

L1 = J1+1;

if (L1 < 3) {
        R12 = C2;
} else if (L1 < 4) {
```

```
        R12 = C3;
} else if (L1 < 5) {
        R12 = C4;
} else if (L1 < 6) {
        R12 = C5;
} else if (L1 < 7) {
        R12 = C6;
} else if(L1 < 8) {
        R12 = C7;
} else  {
        R12 = C8;
}


/*"******** Third Rule:J1=1,I1=2 */

        N1 = I1+1;

        if (N1 < 3) {
                D1 = 0.0342;
                D2 = 0.0342;
                D3 = 0.0201;
                D4 = 0.0201;
                D5 = 0.0079;
                D6 = 0.0079;
                D7 = 0.0079;
                D8 = 0.0079;
        } else if (N1 < 4) {
                D1 = 0.0601;
                D2 = 0.0601;
                D3 = 0.0342;
                D4 = 0.0342;
                D5 = 0.0201;
                D6 = 0.0201;
                D7 = 0.0079;
                D8 = 0.0079;
        } else if (N1 < 5) {
                D1 = 0.0937;
                D2 = 0.0937;
                D3 = 0.0601;
                D4 = 0.0601;
                D5 = 0.0342;
```

```
        D6 = 0.0342;
        D7 = 0.0201;
        D8 = 0.0201;
} else if (N1 < 6) {
        D1 = 0.1998;
        D2 = 0.1397;
        D3 = 0.0937;
        D4 = 0.0937;
        D5 = 0.0601;
        D6 = 0.0601;
        D7 = 0.0342;
        D8 = 0.0342;
} else if (N1 < 7) {
        D1 = 0.2346;
        D2 = 0.1998;
        D3 = 0.1397;
        D4 = 0.1397;
        D5 = 0.0937;
        D6 = 0.0937;
        D7 = 0.0601;
        D8 = 0.0601;
} else if (N1 < 8) {
        D1 = 0.42;
        D2 = 0.35;
        D3 = 0.1998;
        D4 = 0.1998;
        D5 = 0.1397;
        D6 = 0.1397;
        D7 = 0.0937;
        D8 = 0.0601;
} else if(N1 < 9) {
        D1 = 0.6;
        D2 = 0.42;
        D3 = 0.35;
        D4 = 0.2346;
        D5 = 0.2346;
        D6 = 0.1998;
        D7 = 0.1397;
        D8 = 0.0937;
} else  {
        D1 = 1;
        D2 = 0.6;
        D3 = 0.42;
```

```
            D4 = 0.35;
            D5 = 0.35;
            D6 = 0.2346;
            D7 = 0.1998;
            D8 = 0.1397;
        }

        if (J1 < 2) {
            R21 = D1;
        } else if (J1 < 3) {
            R21 = D2;
        } else if (J1 < 4) {
            R21 = D3;
        } else if (J1 < 5) {
            R21 = D4;
        } else if (J1 < 6) {
            R21 = D5;
        } else if (J1 < 7) {
            R21 = D6;
        } else {
            R21 = D7;
        }


/*"********* Fourth Rule:J1=2,I1=2 */

        if (L1 < 3) {
            R22 = D2;
        } else if (L1 < 4) {
            R22 = D3;
        } else if (L1 < 5) {
            R22 = D4;
        } else if (L1 < 6) {
            R22 = D5;
        } else if (L1 < 7) {
            R22 = D6;
        } else if (L1 < 8) {
            R22 = D7;
        } else  {
            R22 = D8;
        }
```

```
/*"********** Defuzzyfication using height method */

        SUM = MRA + MRB + MRC + MRD;
        DT = (R11*MRA+R12*MRB+R21*MRC+R22*MRD)/SUM;
        TS1 = DT*100;    /* " OUTPUT OF FUZZY TS ESTIMATOR*/

/*"TS2 = IF TS1<TSMIN THEN TSMIN ELSE MIN(TS1,TSMAX) */

/*
NTSD is the new value
TSD is the old value
*/

        TC  = 180;

        TC1 = 0.001024 + TC;
        TC2 = 1/TC1;
        TC3 = 0.001024*TC2;
        TC4 = 1-TC3;

        TSD = 0.004149377*TS1 + 0.995850622*TSD_OLD;
        RS = 0.5814 + 3.928E-03*0.5814*(TSD_OLD +25-25);



/*   printf("%f        %f     %f     %.5f \n",wcom,iscom,RS,TSD);*/
   fprintf(fp1,"%f     %f     %f     %f\n",iscom,TSD_OLD,RS,t);
/*   fprintf(fp2,"%f   %f\n",t,RS);*/
        TSD_OLD =TSD;
}


graph( t, iscom, TS1);

                if (kbhit() != 0) key = getch();
                   else   key = 1 ;
                switch ( key ) {
                   case 'p':
                   case 'P': getch();
                             break;
                   case 'q':
```

```
                case 'Q': closegraph();
                          exit( 1);
                default:
                          break;
              }


        getch();
}
// fclose(fp1);
// fclose(fp2);

/*************************************************/
/*      GRAPHICS  DISPLAY                        */
/*************************************************/
int    ng = 0;
double  xi = 110.0, xe = 960.0, xs; /*  1000 * 600 Imaginary Screen */
double  yi = 30.0, ye = 540, ys;
double  ass;
double  Id, b1, b2;
double  h1 = 0.0, h2 = 0.0;
int  i11=0, i1=0, i21=0, i2=0, i31=0, i3=0, i41=0, i4=0;
char buf[100];

void  graph(float t, float iscom, float TS1)
{

   int  i, z=0;
   int  d1=0, d2=0;
   double  Sb1 = 5.0;

     if (ng == 0) {
          cleardevice();
          xs = xe - xi;
          ys = ye - yi;
          ng = 1;

          changetextstyle( 0, 0, 1);

          drawborder( xi, yi, xe, ye, 15);

           ass = xs / (Tmax-Tmin);
           Id  = xs / 10.0;
```

```
        i11 = b1 = yi + ys * 0.95;

         h1  = xi;
         gprintfxy( xi - 20, ye+10, "%-4.2f", Tmin * is_cont);

          for (i = 1;i < 10; i++) {
              z = i * Id + xi;
              d1 = (b1-4);
              gline( z, d1, z, d1+8, 12);
gprintfxy( z-20, ye+10, "%-4.2f", (Tmax-Tmin)*is_cont*0.1*(double)(i)+Tmin);
          }
         gprintfxy( xe - 20, ye+10, "%-4.2f", Tmax * is_cont);
         gprintfxy( xi+xs/2.0-30, ye+25, " Is ");

         gline( xi, b1, xi+xs, b1, 7);

         changetextstyle( 3, 0, 1);
         setcolor(12);
         gprintfxy( 10, b1 - 40,"DTs");
        }

//       if( !((int)(t / Td) % (int)(Tmax/Td / 10.))){
           changetextstyle( 0, 0, 1);
           setcolor( 0);
           gprintfxy(160, 0,"%s", buf);
           setcolor(15);
           sprintf( buf,"iscom= %-8.4g TS1 = %-8.5g", iscom, TS1);
           gprintfxy( 160, 0,"%s", buf);
           changetextstyle( 3, 0, 1);
//        }
           h2 = (int)((t-Tmin) * ass) + xi;
           i1 = (int)(b1 - TS1 * Sb1);
           gline( h1, i11, h2, i1, 12);

          h1 = h2;
          i11 = i1;
  return;

}

 /* END OF GRAPHICS */
}
```

APPENDIX F

Software for DSP Boards and Host PC

```
/*****************************************************************/
/****************     MAIN PROGRAM FOR DSP1   ****************/

/*      Description:This program performs following tasks:                */   /*
        1. Read 4-channel ADC board and scale all signals                 */
/*      2. Generate Theta for V/Hz operation                              */
/*      3. Write to Hanning PWM control board                             */
/*      4. Calculate q and d-axes fluxes                                  */
/*      5. Calculate sin(theta) and theta for vector rotation             */
/*      6. Trigger interrupt #3 and 4 and ADC start of conversion         */
/*      7. Serial communication with DSP2                                 */
/*      8. Display selected signals on oscilloscope                       */
/*      9. Write selected variables on screen                             */
/*      10. Acquire and store selected signals into a file                */

/*      Author: Nitin Patel                                               */
/*****************************************************************/


asm("  .length 58 ");
asm("  .width  120 ");

#include <c:\c30tools\stdlib.h>
#include <c:\c30tools\math.h>
#include <c:\c30tools\errno.h>

#define calmask         0x00200000      /* ADC calibration mask */
#define eoc_mask        0x00800000      /* ADC eoc bit mask (bit 7 status reg) */
#define zero            0x00000000      /* zero variable to toggle calibration */
#define adc_trigger     0xffff0000      /* ADC trigger */
#define tog             0x00200000      /* toggle the calibration bit high */
#define write_mask      0x00010000
#define init_stat_reg   0x00800000
#define enable_pwm      0x00830000
#define vds_wad         0x00830000

#define vqs_wad             0x01830000
```

```
#define angle_wad           0x02830000
#define frequency_wad       0x03830000
#define serial_is_ready     0x2
#define new_data_is_in      0x1
#define reset_serial_port   0x02BC0044
#define k                   6
#define Rs                  0.5814          /** 0.01121  **/
#define pi                  3.14159265359
#define twopi               6.28318530718
#define PRIMWD              0x00000800  /* bus control registers */
#define EXPWD               0x00000000
#define PRIMCTL             ((unsigned int *) 0x808064)
#define EXPCTL              ((unsigned int *) 0x808060)


float  outputvec[k];
long   *Pcproceedflag;              /* PC communication flags.                */
extern long   Comm0, Comm1;         /* Use these absolute-location            */
                                    /* communication parameters for flags.    */
extern float  *Comm2;
int logindex,write_to_log,x;

long UA, UB, PHI1;                  /* Integer variables after shifting to MS word*/
long *pwm_data, *pwm_status;        /* PWM register pointers                  */
long wr_flag;                       /* PWM write clear flag                   */
long pwm_initial[13];
long    *control;                   /* ADC control register pointer           */
long    *status;                    /* ADC status register pointer            */
long    *timer;                     /* ADC timer register pointer             */
long    *adc_result;                /* ADC port register pointer              */
long    *dac0_output;               /* dac0 port register pointer             */
long    *dac1_output;               /* dac1 port register pointer             */
long    adc_value;                  /* 4 channel adc result after             */
                                    /* masking the extra sign bits            */
int     isr0, write_to_log, logindex;
long    adc_value;                  /* 4 ch. adc result after masking         */
                                    /* the extra sign bits                    */
long   iaint;                       /* chan0 pointer non scaled value         */
long    ibint;                      /* chan1 pointer non scaled value         */
long    dclinkvoltint;              /* chan2 pointer non scaled value         */
long   dclinkcurint;                /* chan3 pointer non scaled value         */
int lpc2,lpc3;
long cal_status,cal1_status;
```

```
long *serial_global0, *xmit_control0;
long *rece_contro0, *ser_tim_control0, *ser_tim_count0, *ser_tim_period0;
long *data_xmit0, *data_received0, DATA_RECEIVED;
long *serial_global1, *xmit_control1;
long *rece_contro1, *ser_tim_control1, *ser_tim_count1, *ser_tim_period1;
long *data_xmit1, *data_received1;
float ia, ib, ic;
long indx;
long out0_tmp, out1_tmp;
float *LOG3;
float theta_new,theta_old,delta_theta;
long theta_count;
long vqss,vdss,iqss,idss;
float vqss_f,vdss_f,vqss_a,vdss_a;
float iqss_f,idss_f;
float vq,vd;
float we;
float we_est;
float we_aux,we_p;
float g,Tc;
float cx,cx1,cx2;
float Tc_a,Tc_b,A,B;
float yqss1,yqsp1;
float yqss2,yqsp2;
float yqss3,yqsp3;
float ydss1,ydsp1;
float ydss2,ydsp2;
float ydss3,ydsp3;
float ys_lp;
float iqss_new,iqss_old;
float iqss_der;
float delta_yqss,yqss_bl;
float yqss_bl_old;
float delta_iqss;
float ys_mag;
float Te;
long dac0,dac1;
int dac0_var,dac1_var;
long *DAC0_reg,*DAC1_reg;
float *look_up_sine, *look_up_cosine, *init_sine_add;
float  *init_cosine_add;
float *last_look_up_sine, *last_look_up_cosine;
long  last_look_up_count, look_up_count;
```

```
long step_table;
float sinthe, costhe;
long ind;
long dac0_aux,dac1_aux;
long dac0_aux1,dac1_aux1;
float inv_ys_lp;
float sin_e,cos_e;
float iqs_e,ids_e;
float yqss_n,ydss_n;
float yqss_np,ydss_np;
float ys_n,inv_ys_n;
float yds_e;
long angle;
float theta_e;
long theta_int;
float zqs,zqsp;
long theta_int,angle_out;
long i,j;
long trans[10];
long rec[10];
long isr1;
long yqss_int,ydss_int,ys_lp_int;
long *CHANGE;
long change_we;
long ser_comm;
float yds_e;
float idq_aux,idq_aux_inv,idq;
float inv_ys_mag;
long yqss_int,ydss_int;
long theta_last;
float theta_out;
long theta_tmp;
float we_v;
float ia_tmp;
float we_vhz;
float we_est;
float yqss_f,ydss_f;
float vs_aux,vs_mag;
float uq,ud;
long mode;
long *SWITCH;
long *TR;
long *RE;
```

```
long init,init1,cntr;
float we_u;
int adj;
float IS;

void main() /** Main function starts **/
{

/******* Declaring the external variables  ********/
*PRIMCTL = PRIMWD;
*EXPCTL = EXPWD;

logindex = 0;
x = 0;
PCproceedflag = &Comm0;
write_to_log = 0;
outputvec[0] = 0;
outputvec[1] = 0;
outputvec[2] = 0;
outputvec[3] = 0;

theta_old = 0;
theta_new = 0;
theta_count = 0;
logindex = 0;
x = 0;

g = 0;
Tc = 0;
yqss1 = 0;
yqsp1 = 0;
yqss2 = 0;
yqsp2 = 0;
yqss3 = 0;
yqsp3 = 0;
ydss1 = 0;
ydsp1 = 0;
ydss2 = 0;
ydsp2 = 0;
ydss3 = 0;
ydsp3 = 0;

yqss_n = 0;
```

```
yqss_np = 0;
ydss_n = 0;
ydss_np = 0;
iqss_old = 0;
delta_yqss = 0;
yqss_bl_old = 0;

we_aux = 0;
we_p = 0;
dac0_var = 0;
dac1_var = 0;
last_look_up_count = 0;
look_up_count = 0;
angle = 0;
i = 0;
j = 0;
zqsp = 0;
trans[0] = 0;
trans[1] = 0;
trans[2] = 0;
trans[3] = 0;
trans[4] = 0;
trans[5] = 0;
trans[6] = 0;
rec[0] = 0;
rec[1] = 0;
rec[2] = 0;
rec[3] = 0;
rec[4] = 0;
rec[5] = 0;
rec[6] = 0;
theta_last  = 0;
mode = 0;
init = 1;
init1 = 1;
cntr = 0;

   /******* Define register addresses  *******/
pwm_data      = (long *)  0x00800000;  /* PWM data register pointer              */
pwm_status    = (long *)  0x00800001;  /* PWM status register pointer            */
control       = (long *)  0x00800008;  /* Control register in 4-channel I/O board   */
status        = (long *)  0x00800008;  /* Status register pointer in 4-channel I/O board */
timer         = (long *)  0x00800009;  /* Timer register pointer in 4-channel I/O board  */
```

```
adc_result    = (long *)  0x0080000a;  /* ADC  result register in 4-channel I/O board   */
dac0_output   = (long *)  0x0080000a;  /* DAC output register in 4-channel I/O board*/
dac1_output   = (long *)  0x0080000b;  /* DAC output register in 4-channel I/O board*/
serial_global0  = (long *)  0x808040;
xmit_control0   = (long *)  0x808042;
rece_contro0    = (long *)  0x808043;
ser_tim_control0 = (long *)  0x808044;
ser_tim_count0        = (long *)  0x808045;
ser_tim_period0       = (long *)  0x808046;
data_xmit0            = (long *)  0x808048;
data_received0        = (long *)  0x80804C;
serial_global1        = (long *)  0x808050;
xmit_control1         = (long *)  0x808052;
rece_contro1          = (long *)  0x808053;
ser_tim_control1      = (long *)  0x808054;
ser_tim_count1        = (long *)  0x808055;
ser_tim_period1       = (long *)  0x808056;
data_xmit1            = (long *)  0x808058;
data_received1        = (long *)  0x80805C;

LOG3                  = (float *) 0x00030003;
DAC0_reg              = (long *) 0x00030008;
DAC1_reg              = (long *) 0x00030009;
SWITCH                = (long *) 0x0003000B;
/******* Program the serial port #0 *******/
        *serial_global0  = reset_serial_port;
        *serial_global0  = 0x0EBC0044;
        *xmit_control0   = 0x0111;
        *rece_contro0    = 0x0111;
        *ser_tim_control0 = 0x01CF;
        *ser_tim_count0  = 0x0;
        *ser_tim_period0  = 0x01;
   /*************************************************/
    /*** The following loop waits for the second DSP board to be ***/
   /*** ready, allowing the dual-communication with serial port ***/
   indx = 1000000;
    while(indx>0)
    {
     indx = indx - 1;
    }
    indx = 0;
/******* Program the serial port #1 *******/
 *serial_global1   = reset_serial_port;
```

```
   *serial_global1  = 0x0EBC0044;
   *xmit_control1   = 0x0111;
   *rece_control    = 0x0111;
   *ser_tim_control1 = 0x01CF;
   *ser_tim_count1  = 0x0;
   *ser_tim_period1 = 0x01;
   /*****************************************************/
   /*** The following loop waits for the second DSP board to be ***/
   /*** ready, allowing the dual-communication with serial port ***/
   indx = 1000000;
   while(indx>0)
   {
  indx = indx - 1;
   }
  indx = 0;


   /******** Initial values for some variables ********/
*DAC0_reg = 0;
*DAC1_reg = 0;
*LOG3 = 0;
ser_comm = 0;
UA = 0;
UB = 0;
ia = 0;
ib = 0;
ic = 0;
iaint  = 0;
ibint  = 0;
vqss = 0;
vdss  = 0;
iqss = 0;
idss = 0;
lpc2 = 1;
lpc3 = 1;
isr0=1;
isr1 = 1;
PHI1 = 0x00000000;
change_we = 0;
*CHANGE = 0;
*SWITCH = 0;
*TR = 1;
*RE = 1;
```

```
/******** End of variables initialization  ********/

/******** Initial Hanning chip values          ********/
pwm_initial[0] = 0x00000000; /* UA   - vds value   */
pwm_initial[1] = 0x00000000; /* UB   - vqs value   */
pwm_initial[2] = 0x00000000; /* PHI1 - theta value */
pwm_initial[3] = 0x00000000; /* DPHI1    */
pwm_initial[4] = 0x00000000; /* PHI0     */
pwm_initial[5] = 0x00000000; /* DPHI0    */
pwm_initial[6] = 0x00000000; /* PHIADD   */
pwm_initial[7] = 0x00000000; /* not used */
pwm_initial[8] = 0x00180000; /* 0x00180000 - TAUS - min off time  - 1.5us */
pwm_initial[9] = 0x006F0000; /* 0x007f0000 - TTOT - dead time     - 4.0us */
pwm_initial[10] = 0x00180000; /* 0x00180000 - TMIN - min on time   - 3.0us */
pwm_initial[11] = 0x00000000; /* VORTL - fsw = 15.6KHz */
pwm_initial[12] = 0x00BE0000; /* TSTART value */
/******** End of initial Hanning chip values   ********/

/****** Initialize Hanning chip status in 16-bit mode  ******/
*pwm_status = init_stat_reg;
*pwm_status = init_stat_reg;
/****** End of 16-bit mode initialization           ******/

/****** Load the initial array for Hanning chip ******/
for (indx=0;indx<=12;indx+=1)
  {
   do
     {
      wr_flag = write_mask & *pwm_status;
     }
     while (wr_flag==write_mask);
   *pwm_data=pwm_initial[indx];
     asm(" NOP ");
  }
indx = 0;
/****** End of initial array loading          ******/


/******* Finish initializations  **********/

/************************************************/
/***** Starts doing the I/O board calibration  ****/
/************************************************/
```

```
 *control = zero;      /* toggle calibration bit on 4-channel A/D */
 *control = tog;
 *control = zero;

 do             /* loop until calibration is complete */
 { cal_status = calmask & *control;
 }
 while ( cal_status == 0 );
 /********** Finish the calibration ***************/



  /****** Enable interruptions and start PWM calculations          ******/
  asm(" OR 0Fh, IE");      /* Enable INT1, INT3 and INT4                */
  asm(" OR 2000h, ST");    /* Enable global CPU interrupts              */
  *control = 0x00800000;   /* Setup I/O board for interruptions         */
  *pwm_status = enable_pwm; /* Enable PWM calculations on Hanning chip */

/*************************************************************************/
while(1)                /* Loop here forever.          */
{
;
} /* End of while(1).          */


} /**** end of main function ****/

/***** This routine receives the four converted channels from *****/
/***** AD board one at the time. Such routine can interrupt   *****/
/***** Task2 and Task3, but it will be finished before Task1  *****/
/***** starts again (4 conversions will take appr. 20 us)     *****/

asm("  .sect \".int01\"");
asm("  .word _c_int01 ");
asm("  .text    ");

c_int01()
{
  adc_value = *adc_result >> 20; /* Shift for DSPLINK bus */

  if (isr0==1)
    {   vqss = adc_value;
        vqss_f = 9.1842e-2*vqss;
```

```
        trans[1] = vqss<<20;
        *control=0x00810000;
    }

  if (isr0==2)
    {   vdss = adc_value;
        vdss_f = 9.1842e-2*vdss;
        trans[2] = vdss<<20;
        *control=0x00820000;
    }

  if (isr0==3)
    {   iqss = adc_value;
        trans[3] =iqss<<20;
        iqss_f = 0.0250761*iqss;
        *control=0x00C30000;
    }

  if (isr0==4)
    {   idss = adc_value;
        trans[4] = idss<<20;
        idss_f = 0.0250761*idss;
        *control=0x00800000;
        isr0 = 0;
    }
  ++isr0;

} /* End of interruption service routine #0


/*** This routine performes the TASK1 each 128 us (hardware trigger) and triggers ***/
/*** the correspondent TASK2  in the correct timing.  ***/
/*** At the end of TASK1 the ADC's conversion are triggered.   ***/
asm("   .sect \".int02\"");
asm("   .word _c_int02 ");
asm("   .text     ");
c_int02()
{
    asm("         PUSH IE ");
    asm("       LDI 0, IE ");
    *pwm_status = vds_wad;
    *pwm_data = UA;
    asm("         AND   0DFFFh, ST");
```

```
    asm(" POP IE ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    /***************************************/

    /**** Write vqs  ****/
    *pwm_data = UB;
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    /***************************************/


    /**** Write theta  ****/
    *pwm_data = PHI1;
    asm(" NOP ");
    asm(" NOP ");
    asm(" NOP ");
    /*****************************************************/
    /************** End of Hanning chip writing *********/
    /*****************************************************/


ia_tmp = *LOG3;
ia = ia_tmp*188;
we=2.5*ia;      /* 2.5*ia; */

delta_theta = we*128e-6;
theta_new = delta_theta + theta_old;
if(theta_new >= 6.283)
```

```
theta_new = 0;
theta_old = theta_new;
theta_count = theta_new*3911.39;


out0_tmp = ia_tmp*32767;

mode = *SWITCH;
if(mode == 0)
{
UA = out0_tmp << 16;
UB = 0;
PHI1 = theta_count << 16;
we_v = we;
}
else
{
UA = rec[2];
UB = rec[3];
PHI1 = theta_tmp << 16;
we_v = we_est;
}

trans[0] = theta_count << 16;


/** Check and transmit six signals to the other dsp    ****/
cal_status = (*serial_global0 & 0x02) && (*serial_global0 & 0x00800000);

if( cal_status != 0 );
{
        if(i<7)
        {
                *data_xmit0 = trans[i];
                i++;
        }
else
        i=0;
}
/** Check and receive four signals from the other dsp          ****/
 cal1_status = (*serial_global1 & 0x01) && (*serial_global1 & 0x02000000);
if( cal1_status != 0 );
{
```

```
        if(j<4)
        {
        rec[j] = *data_received1;
                j++;
        }
else
        j=0;
}

we_u = 2.1057e-2*(rec[1]>>16);

if(we_u <= 1)
        we_est = 1;
else
        we_est = we_u;



/*change_we = *CHANGE;*/


cx1 = 1/we_v;
cx2 = cx1*cx1;
Tc = -1.2832565E-6 + 4.0434856E-11*we_v + cx;
cx = 0.57735377*cx1 - 4.1367812E-5*cx2;
g = -2.3749952E-6 + 1.9270465E-8*we_v + 1.5395992*cx1;

Tc_a = Tc + 128e-6;
Tc_b = 1/Tc_a;
A = 128e-6*Tc_b;                    /**** Programmable TC ****/
B = 1-A;
vqss_a = vqss_f - iqss_f*Rs;
vq = vqss_a*g;

/**** q-axes three cascaded programmable lpfs ****/
yqss1 = A*vq + B*yqsp1;
yqsp1 = yqss1;

yqss2 = A*yqss1 + B*yqsp2;
yqsp2 = yqss2;

yqss3 = A*yqss2 + B*yqsp3;
yqsp3 = yqss3;
```

```
/*yqss_f = yqss3*g;*/
yqss_int = 4094*yqss3;

trans[5] = yqss_int<<20;

vdss_a = vdss_f - idss_f*Rs;
vd = vdss_a*g;

/**** d-axes three cascaded programmable lpfs ****/

ydss1 = A*vd + B*ydsp1;
ydsp1 = ydss1;

ydss2 = A*ydss1 + B*ydsp2;
ydsp2 = ydss2;

ydss3 = A*ydss2 + B*ydsp3;
ydsp3 = ydss3;
/*ydss_f = ydss3*g;*/
ydss_int  = 4094*ydss3;
trans[6] = ydss_int<<20;


ys_mag = yqss3*yqss3+ydss3*ydss3;
ys_lp = sqrt(ys_mag);  /**** compute flux magnitude ****/
if(ys_lp != 0)
{
inv_ys_lp = 1/ys_lp;
}
else
{
inv_ys_lp = 1;
}

sin_e = yqss3*inv_ys_lp;

zqs = asin(sin_e);


if((yqss3>=0) && (ydss3>=0))
theta_out = zqs;
if(ydss3<0)
theta_out = pi-zqs;
```

```
if((yqss3<0) && (ydss3>=0))
theta_out = twopi+zqs;

theta_tmp = theta_out*3911.4;

  /************************************************/
  /*** Triggering the correspondent tasks      ***/
  /************************************************/
 if (lpc2 == 8)
     {
      lpc2  = 1;
      asm(" OR 0004h, IF");     /* Trigger interruption # 3 */
     }
   else {++lpc2;}


 if (lpc3 == 10000)
     {
      lpc3  = 1;
      asm(" OR 0008h, IF");     /* Trigger interruption # 4 */
     }
   else {++lpc3;}

  /************************************************/
  /*** End of Tasks Triggering                ***/
  /************************************************/

  asm("  OR  2000h, ST");  /* Enable global CPU interrupts   */
  /************************************************************/
  /***** Trigger ADC for starting conversion          *****/
  /************************************************************/

  *timer = adc_trigger;    /* Triggering all the S-H to receive the   */
                  /* four-channels one at the time          */
  }  /**** End of Interrupt Service Routine #3 ****/




asm("   .sect \".int03\"");
asm("   .word _c_int03 ");
asm("   .text     ");
c_int03()
```

```
{
ud = 9.7656e-4*(rec[2] >> 16);
uq = 9.7656e-4*(rec[3] >> 16);

/**dac1_output = theta_count << 16;*/
dac0_var = *DAC0_reg;

if(dac0_var == 0)
{ dac0 = rec[0]>>20;
}
else if(dac0_var == 1)
{ dac0= we_est;
}
else if(dac0_var == 2)
{ dac0 = vdss;
}
else if(dac0_var == 3)
{ dac0 =iqss;
}
else if(dac0_var == 4)
{ dac0 = idss;
}

else if(dac0_var == 5)
{ dac0 = ys_lp*2047;
}
else if(dac0_var == 6)
{ dac0 = theta_tmp>>4;
}
 *dac0_output = dac0<<20;

/*dac1_var = *DAC1_reg;*/

if(dac1_var == 0)
{ dac1 = 4094*yqss3;
}
else if(dac1_var == 1)
{ dac1= 4094*ydss3;  /*we_est;*/
}
else if(dac1_var == 2)
{ dac1 = vqss;
}
else if(dac1_var == 3)
```

```
{ dac1 =vdss;
}
else if(dac1_var == 4)
{ dac1 = iqss;
}

else if(dac1_var == 5)
{ dac1 = idss;
}

 *dac1_output = dac1<<20;



Comm2 = outputvec; /* Put starting address of array in absolute    */
                   /* location so the PC can find the array        */
if(logindex == 39)    /* The logging will happen at every 79.872ms */
 {
        logindex = 0;
        if(write_to_log == 0)
         {
                write_to_log = 1;
                *PCproceedflag = 1;
                if(x<k)
                 {
                  outputvec[x++] =ia;
                  outputvec[x++] =we_est;
                  outputvec[x++] =ys_lp;
                  outputvec[x++]  =we;
                  outputvec[x++] =uq;
                  outputvec[x++]  =ud;

                  x=0;

                 }
         }
        else
         {
                write_to_log = 0;
                *PCproceedflag = 0;
         }
 }
```

```
else
 {
        ++logindex;
 }
/************** End of Log signals service ****************************/

}


asm("  .sect \".int04\"");
asm("  .word _c_int04 ");
asm("  .text    ");
c_int04()
{
}
```

```
/*****************************************************************/
/****************     MAIN PROGRAM FOR DSP2  ***************/

/*      Description: This program performs following tasks:              */
/*      1. Read 4-channel ADC board and scale all signals                */
/*      2. Estimate all required feed back signals such as:              */
/*      q and d-axes fluxes,magnitude flux,we,Te,iqs,ids,idq,wsl etc.    */
/*      3. PI controllers for iqs,ids,Te and Flux loops                  */
/*      4. Trigger interrupt #4 and ADC start of conversion              */
/*      5. Serial communication with DSP1                                */
/*      6. Display selected signals on oscilloscope                      */
/*      7. Write selected variables on screen                            */
/*      8. Acquire and store selected signals into a file                */
/*      9. Fuzzy Rs estimation                                           */
/*      Author: Nitin Patel                                              */
/*****************************************************************/

asm("  .length 58 ");
asm("  .width  120 ");

#include <c:\c30tools\stdlib.h>
#include <c:\c30tools\math.h>
```

```
#define calmask         0x00200000   /* ADC calibration mask */
#define eoc_mask        0x00800000   /* ADC eoc bit mask (bit 7 status reg) */
#define zero            0x00000000   /* zero variable to toggle calibration */
#define adc_trigger     0xffff0000   /* ADC trigger */
#define tog             0x00200000   /* toggle the calibration bit high */
#define write_mask      0x00010000
#define serial_is_ready 0x2
#define new_data_is_in  0x1
#define reset_serial_port 0x02BC0044
#define k       6
#define Rs      0.5814
#define MAXVAL 2047
#define MINVAL 0
#define pi          3.14159265359
#define twopi       6.28318530718
#define look_up_table_size 768       /* Corresponds to 0.46 deg. resolution */
#define PRIMWD              0x00000800  /* bus control registers
*/
#define EXPWD               0x00000000
#define PRIMCTL             ((unsigned int *) 0x808064)
#define EXPCTL              ((unsigned int *) 0x808060)
#define vds_lim             28.8
#define vqs_lim             28.8
#define zeroband            0.4
#define         kpq         1
#define         kpd         1
float  outputvec[k];
long  *PCproceedflag;        /* PC communication flags.   */
extern long   Comm0, Comm1;        /* Use these absolute-location                */
                                   /* communication parameters for flags.  */
extern float  *Comm2;
int logindex,write_to_log,x;

long    *control;               /* ADC control register pointer */
long    *status;                /* ADC status register pointer */
long    *timer;                         /* ADC timer register pointer */
long    *adc_result;            /* ADC port register pointer */
long    *dac0_output;                   /* dac0 port register pointer */
long    *dac1_output;                   /* dac1 port register pointer */
long    adc_value;              /* 4 channel adc result after masking the extra sign bits*/
int     isr0, write_to_log, logindex;
long    adc_value;              /* 4 ch. adc result after masking the extra sign bits*/
long   iaint;           /* chan0 pointer non scaled value */
```

```
long    ibint;          /* chan1 pointer non scaled value */
long    dclinkvoltint;          /* chan2 pointer non scaled value */
long   dclinkcurint;            /* chan3 pointer non scaled value */
int lpc2,lpc3;
long cal_status,cal1_status;
long *serial_global0, *xmit_control0;
long *rece_contro0, *ser_tim_control0, *ser_tim_count0, *ser_tim_period0;
long *data_xmit0, *data_received0, DATA_RECEIVED;
long *serial_global1, *xmit_control1;
long *rece_contro1, *ser_tim_control1, *ser_tim_count1, *ser_tim_period1;
long *data_xmit1, *data_received1;
long indx;
float *LOG3;
float *scale1,*scale2;
float *scale3,*scale4;

float vqss_f,vdss_f;
float iqss_f,idss_f;
float yqss_f,ydss_f;
long vqss_int,vdss_int;
long iqss_int,idss_int;
long yqss_int,ydss_int;

float ys_lp,ys_mag;
float inv_ys_lp;
float sin_e,cos_e;
float Te;
long dac0,dac1;
int dac0_var,dac1_var;
long *DAC0_reg,*DAC1_reg;
float sinthe, costhe;
long ind;
long dac0_aux,dac1_aux;
long dac0_aux1,dac1_aux1;
float iqs_e,ids_e;
float yqss_n,ydss_n;
float yqss_np,ydss_np;
float ys_n,inv_ys_n;
float yds_e;
long angle;
float theta_e;
long theta_int;
float zqs,zqsp;
```

```
long theta_int,angle_out;
long i,j;
long rec[10];
long trans[10];
long isr1;
float ia,ib;
long sc1,sc2,sc3,sc4;
float inv_ys_mag;
float we_aux,we_est,we_p,Te;
float idq,idq_aux,idq_aux_inv;
float iqs,ids,yq,yd;
float error_iqs,error_ids;
float last_yq,last_yd;
float iqsref,idsref,ids_add;
float vqs,vds;
float yt,yf;
float error_ys;
float ysref;
float iqs_aux,ids_aux;
long we_int;
float wsl_aux,wsl;
long vqs_int,vds_int;
float *IQS;
float *IDS;
float *KP_Q;
float *KP_D;
float *KI;
float kiq,kid;
float vds_l,vqs_l;
long *RE;
long init,init1,cntr;
float last_error_iqs,last_error_ids;
float last_vqs,last_vds;
long *PIZERO;
float iqs_old,iqs_aux;
float ids_old,ids_aux;
float ids_pi,last_ids_pi;
float error_ys,last_error_ys;
float ysref,kpf,kif;
float *FLUXREF;
float *KP_F;
float *KI_F;
float *FLUX_MODE;
```

```
float flux_close;
long teclose;
long *TE_MODE;
float kpt,kit,teref;
float *KP_T,*KI_T;
float *TE_REF;
float error_te,teref,iqs_pi;
float last_error_te,last_iqs_pi;
float Te_aux,Te_old;
float yqs_e;
float wcom,iscom;
float WE,IS;
float W1,W2,W3,W4,W5,W6,W7,W8;
float IS1,IS2,IS3,IS4,IS5,IS6,IS7,IS8,IS9;
float J1,I1,L1,N1;
float MW1,MW2,MIS1,MIS2;
float MRA,MRB,MRC,MRD;
float R11,R12,R21,R22;
float C1,C2,C3,C4,C5,C6,C7,C8,C9;
float D1,D2,D3,D4,D5,D6,D7,D8,D9;
float SUM,DT,TS1;
float TSD,TSD_OLD,TSD_OLD1,RS;
long rs_v_int;
float rs_v,rs_cal;
float I2,I3,I4;
float TC,TC1,TC2;
float I_AUX,I_OLD;
float TERROR;
float A1,B1;
float CR,CR1;
float TERROR_AUX;
float TC_AUX;

void main() /** Main function starts **/
{

/******* Declaring the external variables  ********/

*PRIMCTL = PRIMWD;
*EXPCTL = EXPWD;

logindex = 0;
x = 0;
```

```
PCproceedflag = &Comm0;
write_to_log = 0;
outputvec[0] = 0;
outputvec[1] = 0;
outputvec[2] = 0;
outputvec[3] = 0;

yqss_n = 0;
yqss_np = 0;
ydss_n = 0;
ydss_np = 0;

dac0_var = 0;
dac1_var = 0;
angle = 0;
i = 0;
j=0;
zqsp = 0;

rec[0] = 0;
rec[1] = 0;
rec[2] = 0;
rec[3] = 0;
rec[4] = 0;
rec[5] = 0;
rec[6] = 0;

trans[0] = 0;
trans[1] = 0;
trans[2] = 0;
trans[3] = 0;
trans[4] = 0;
trans[5] = 0;
trans[6] = 0;
yqss_f= 0;
ydss_f= 0;
vqss_int= 0;
vdss_int = 0;
iqss_int = 0;
idss_int = 0;
yqss_int = 0;
ydss_int = 0;
theta_int =0;
```

```
we_p =0;

error_iqs = 0;
error_ids = 0;
vqs = 0;
vds = 0;
yq = 0;
yd = 0;
last_yq = 0;
last_yd = 0;
error_ys = 0;
ids_aux = 0;
yt = 0;
yf = 0;
last_yt = 0;
last_yf = 0;
init = 1;
init1 = 1;
cntr = 0;
last_error_iqs = 0;
last_error_ids = 0;
last_vqs = 0;
last_vds = 0;
last_error_ys = 0;
last_ids = 0;
last_error_te = 0;
last_iqs_pi = 0;
Te_old = 0;
rs_v_int =0;
TSD_OLD = 0;
I1 = 0;
I2 = 0;
I3 = 0;
I_AUX = 0;
I_OLD = 0;

CR = 0;
CR1 = 0;

   /******* Define register addresses  *******/
control     = (long *)  0x00800008;  /* Control register in 4-channel I/O board      */
status      = (long *)  0x00800008;  /* Status register pointer in 4-channel I/O board */
timer       = (long *)  0x00800009;  /* Timer register pointer in 4-channel I/O board  */
```

```
adc_result    = (long *) 0x0080000a;  /* ADC result register in 4-channel I/O board    */
dac0_output   = (long *) 0x0080000a;  /* DAC output register in 4-channel I/O board*/
dac1_output   = (long *) 0x0080000b;  /* DAC output register in 4-channel I/O board*/
serial_global0  = (long *) 0x808040;
xmit_control0   = (long *) 0x808042;
rece_contro0    = (long *) 0x808043;
ser_tim_control0 = (long *) 0x808044;
ser_tim_count0  = (long *) 0x808045;
ser_tim_period0 = (long *) 0x808046;
data_xmit0      = (long *) 0x808048;
data_received0  = (long *) 0x80804C;
serial_global1  = (long *) 0x808050;
xmit_control1   = (long *) 0x808052;
rece_contro1    = (long *) 0x808053;
ser_tim_control1 = (long *) 0x808054;
ser_tim_count1  = (long *) 0x808055;
ser_tim_period1 = (long *) 0x808056;
data_xmit1      = (long *) 0x808058;
data_received1  = (long *) 0x80805C;
        FLUXREF             = (float *) 0x00030001;
        TE_REF              = (float *) 0x00030003;
        KP_T                = (float *)0x00030004;
        KI                  = (float *) 0x00030005;
        KI_T                = (float *) 0x00030006;
        FLUX_MODE           = (float *) 0x00030007;
        DAC0_reg            = (long *) 0x00030008;
        DAC1_reg             = (long *) 0x00030009;
        KP_F                = (float *) 0x0003000A;
        TE_MODE             = (long *) 0x0003000B;
        IQS                 = (float *) 0x0003000C;
        IDS                  = (float *) 0x0003000D;
        PIZERO              = (long *) 0x0003000E;
        KI_F                = (float *) 0x0003000F;


/******* Program the serial port #0 *******/
        *serial_global0  = reset_serial_port;
        *serial_global0  = 0x0EBC0044;
        *xmit_control0   = 0x0111;
        *rece_contro0    = 0x0111;
        *ser_tim_control0 = 0x01CF;
        *ser_tim_count0  = 0x0;
```

```
    *ser_tim_period0 = 0x01;
  /*******************************************************/
  /*** The following loop waits for the second DSP board to be ***/
  /*** ready, allowing the dual-communication with serial port ***/
  indx = 1000000;
  while(indx>0)
  {
      indx = indx - 1;
  }
  indx = 0;

  /******* Program the serial port #1 *******/
*serial_global1  = reset_serial_port;
  *serial_global1  = 0x0EBC0044;
  *xmit_control1   = 0x0111;
  *rece_control    = 0x0111;
  *ser_tim_control1 = 0x01CF;
  *ser_tim_count1  = 0x0;
  *ser_tim_period1 = 0x01;
  /*******************************************************/
  /*** The following loop waits for the second DSP board to be ***/
  /*** ready, allowing the dual-communication with serial port ***/
  indx = 1000000;
  while(indx>0)
  {
  indx = indx - 1;
  }
  indx = 0;

  /******** Initial values for some variables ********/
*DAC0_reg = 0;
*DAC1_reg = 0;
*LOG3 = 0;
*KP_Q = 0;
*KP_D = 0;
*KI = 0;
iaint = 0;
ibint  = 0;
lpc2 = 1;
lpc3 = 1;
isr0=1;
isr1 = 1;
dclinkvoltint = 0;
```

```
dclinkcurint = 0;
wsl = 0;
*IQS = 0;
*IDS = 0;
iqsref = 0;
idsref = 0;
*RE = 1;
*PIZERO = 0;
*KI_F = 0;
*KP_F = 0;
*YS = 0;
*FLUX_1 = 0;
    /******** End of variables initialization  ********/


  /*****************************************************/
  /***** Starts doing the I/O board calibration  ****/
  /*****************************************************/
  *control = zero;      /* toggle calibration bit on 4-channel A/D */
  *control = tog;
  *control = zero;

  do              /* loop until calibration is complete */
  { cal_status = calmask & *control;
  }
  while ( cal_status == 0 );
  /********** Finish the calibration ***************/



   /****** Enable interruptions and start PWM calculations       ******/
  asm(" OR 0Fh, IE");      /* Enable INT1, INT3 and INT4       */
  asm(" OR 2000h, ST");    /* Enable global CPU interrupts       */
  *control = 0x00830000;   /* Setup I/O board for interruptions     */

/*************************************************************************/
while(1)                  /* Loop here forever.          */
{
;
}  /* End of while(1).          */


} /**** end of main function ****/
```

```
/***** This routine receives the four converted channels from *****/
/***** AD board one at the time. Such routine can interrupt  *****/
/***** Task2 and Task3, but it will be finished before Task1  *****/
/***** starts again (4 conversions will take appr. 20 us)    *****/

asm("  .sect \".int01\"");
asm("  .word _c_int01 ");
asm("  .text    ");

c_int01()
{
  adc_value = *adc_result >> 20; /* Shift for DSPLINK bus */

  if (isr0==1)
    {   iaint = adc_value;
/*      trans[0] = iaint<<20;*/
        *control=0x00C30000;
        isr0 = 0;
    }
  ++isr0;

} /* End of interruption service routine #0


/*** This routine performes the TASK1 each 128 us(hardware trigger  and triggers ***/
/*** the correspondent TASK2 TASK3 in the correct timing.  ***/
/*** At the end of TASK1 the ADC's conversion are triggered.  ***/
asm("  .sect \".int02\"");
asm("  .word _c_int02 ");
asm("  .text    ");
c_int02()
{
/**dac1_output = MAXVAL<<20;*/

/* Check and receive six signals from the other dsp  **/

cal_status = (*serial_global0 & 0x01) && (*serial_global0 & 0x02000000);
if( cal_status != 0 );
{
        if(i<7)
        {
        rec[i] = *data_received0;
                i++;
```

```
        }
        else
        i=0;
}

/** Check and transmit four signals to the other dsp ****/

call_status = (*serial_global1 & 0x02) && (*serial_global1 & 0x00800000);

if( call_status != 0 );
{
        if(j<4)
        {
                *data_xmit1 = trans[j];
                j++;
        }
        else
        j=0;
}

vqss_int = rec[1]>>20;
vdss_int = rec[2]>>20;
iqss_int = rec[3]>>20;
idss_int = rec[4]>>20;
yqss_int = rec[5]>>20;
ydss_int = rec[6]>>20;

trans[0] = rec[0];

vqss_f = 0.091842*vqss_int;
vdss_f = 0.091842*vdss_int;
iqss_f = 0.0250761*iqss_int;
idss_f = 0.0250761*idss_int;
yqss_f = 2.4426e-4*yqss_int;
ydss_f = 2.4426e-4*ydss_int;

ys_mag = yqss_f*yqss_f+ydss_f*ydss_f;

ys_lp = sqrt(ys_mag);  /**** compute flux magnitude ****/

if(ys_lp != 0)
{
        inv_ys_lp = 1/ys_lp;
```

```
}
else
{
        inv_ys_lp = 1;
}

sin_e = yqss_f*inv_ys_lp;
cos_e = ydss_f*inv_ys_lp;
inv_ys_mag = inv_ys_lp*inv_ys_lp;

we_aux = (vqss_f*ydss_f-vdss_f*yqss_f)*inv_ys_mag;
we_est = 0.203822*we_aux + 0.7961783*we_p;  /*Tc :0.5msec */
we_p = we_est;
we_int = we_est*47.49;
trans[1] = we_int << 16;
/**** Te calculation ****/

Te = 3*(ydss_f*iqss_f-yqss_f*idss_f);

yqss_n = 0.439096*yqss_f + 0.560904*yqss_np;
yqss_np = yqss_n;
ydss_n = 0.439096*ydss_f + 0.560904*ydss_np;
ydss_np = ydss_n;

ys_n = sqrt(yqss_n*yqss_n + ydss_n*ydss_n);
if(ys_n != 0)
{
        inv_ys_n = 1/ys_n;
}
else
{
         inv_ys_n = 1;
}

sinthe = yqss_n*inv_ys_n;
costhe = ydss_n*inv_ys_n;


iqs_e = iqss_f*costhe - idss_f*sinthe;
ids_e = iqss_f*sinthe + idss_f*costhe;

yds_e = yqss_f*sin_e+ydss_f*cos_e;
```

```
idq_aux = yds_e - 7.41999e-3*ids_e;
if(idq_aux != 0)
{
        idq_aux_inv = 1/idq_aux;
}
else
{
        idq_aux_inv = 1;
}

idq= 7.41999e-3*iqs_e*iqs_e*idq_aux_inv; /** 8.671747e-5 **/

if(idq > 5)
{       idq = 5;}
if(idq < -5)
{       idq = -5; }



wsl_aux = ydss_f*iqss_f-yqss_f*idss_f;
wsl = 0.39552*wsl_aux*inv_ys_mag;

teref = *TE_REF;
teclose = *TE_MODE;

kpt = *KP_T;
kit = *KI_T;

error_te = teref-Te;

iqs_pi = last_iqs_pi + (kpt + 0.000128*kit)*error_te-kpt*last_error_te;

last_error_te = error_te;
last_iqs_pi = iqs_pi;

if(teclose == 0)
{
iqsref =*IQS;
}
else
{
iqsref = iqs_pi;
}
```

```
ysref = *YS;
kpf = *KP_F;
kif = *KI_F;
error_ys = ysref- ys_lp;
ids_aux = last_ids + (kpf + 0.000128*kif)*error_ys-kpf*last_error_ys;

last_error_ys = error_ys;
last_ids = ids_aux;



check = *FLUX_1;
if(check == 0)
{
        idsref = *IDS;
}
else if(check == 1)
{
        idsref = ids_aux;
}

ids_add = idsref + idq;

if(ids_add > 18)
{       ids_add = 18; }
if(ids_add < -18)
{       ids_add = -18; }


kiq = *KI;
kid = *KI;

iqsref=*IQS;

error_iqs = iqsref-iqs_e;
error_ids = ids_add-ids_e;


vqs = last_vqs + (kpq + 0.000128*kiq)*error_iqs-kpq*last_error_iqs;
vds = last_vds + (kpd + 0.000128*kid)*error_ids-kpd*last_error_ids;

last_error_iqs = error_iqs;
last_error_ids = error_ids;
```

```
if(vds <= -vds_lim)
        { vds_l = -vds_lim; }
else if(vds < -zeroband)
        { vds_l = vds; }
else if(vds <= zeroband)
        { vds_l = 0; }
else if(vds < vds_lim)
        { vds_l = vds; }
else    { vds_l = vds_lim; }

if(vqs <= -vqs_lim)
        { vqs_l = -vqs_lim; }
else if(vqs < -zeroband)
        { vqs_l = vqs; }
else if(vqs <= zeroband)
        { vqs_l = 0; }
else if(vqs < vqs_lim)
        { vqs_l = vqs; }
else    { vqs_l = vqs_lim; }

if(*PIZERO == 0)
{
        last_vqs = vqs_l;
        last_vds = vds_l;
}
else
{
        last_vqs = 0;
        last_vds = 0;
        *PIZERO = 0;
}


vds_int = vds_l*1024;
trans[2] =vds_int<<16;
vqs_int  = vqs_l*1024;
trans[3] = vqs_int<<16;
WE = we_est/733;     /* Max. Speed:3500 rpm */
IS = iscom/20;

   /***********************************************/
   /*** Triggering the correspondent tasks     ***/
```

```
  /*************************************************/
  if (lpc2 == 8)
    {
     lpc2  = 1;
     asm(" OR 0004h, IF");     /* Trigger interruption # 3 */
    }
  else {++lpc2;}

   if (lpc3 == 10000)
    {
     lpc3  = 1;
     asm(" OR 0008h, IF");     /* Trigger interruption # 4 */
    }
  else {++lpc3;}


  /*************************************************/
  /*** End of Tasks Triggering              ***/
  /*************************************************/

  asm("  OR  2000h, ST");  /* Enable global CPU interrupts   */
  /*************************************************************/
  /***** Trigger ADC for starting conversion          *****/
  /*************************************************************/

  *timer = adc_trigger;    /* Triggering all the S-H to receive the  */
                  /* four-channels one at the time          */
  }  /**** End of Interrupt Service Routine #3 ****/


asm("   .sect \".int03\"");
asm("   .word _c_int03 ");
asm("   .text     ");
c_int03()
{


/*"***** Compute preliminary values of degree of membership function
"     of we and  Is  */


        W1 = MIN(1, WE*23.33);
        W2 = (WE*12.5-0.53575);
```

```
W3 = (WE*8.0619-0.9905);
W4 = (WE*11.94-2.948);
W5 = (WE*11.71-3.871);
W6 = (WE*7.88-3.278);
W7 = MIN(1,(WE*2.188-1.188));


IS1 = MIN(1,IS*5);
IS2 = (IS*10-2);
IS3 = (IS*10-3);
IS4 = (IS*10-4);
IS5 = (IS*10-5);
IS6 = (IS*10-6);
IS7 = (IS*10-7);
IS8 =MIN(1,(IS*10-8));

/*"******** Get interval indices J and I, for WE and IS respectively*/

if (WE < 0.04286) {
        J1 = 1;
} else if (WE < 0.12286) {
        J1 = 2;
} else if (WE < 0.2469) {
        J1 = 3;
} else if (WE < 0.3306) {
        J1 = 4;
} else if (WE < 0.416) {
        J1 = 5;
} else if (WE < 0.5429) {
        J1 = 6;
} else if (WE < 0.6712) {
        J1 = 7;
} else {
        J1 = 8;
}

if (IS < 0.2) {
        I1 = 1;
} else if (IS < 0.3) {
        I1 = 2;
} else if (IS < 0.4) {
        I1 = 3;
} else if (IS < 0.5) {
```

```
        I1 = 4;
} else if (IS < 0.6) {
        I1 = 5;
} else if (IS < 0.7) {
        I1 = 6;
} else if (IS < 0.8) {
        I1 = 7;
} else if (IS < 0.9) {
        I1 = 8;
} else {
        I1 = 9;
}

/*"******** Cmpute degree of membership function for relevant fuzzy sets*/

if (J1 < 2) {
        MW1 = W1;
} else if (J1 < 3) {
        MW1 = W2;
} else if (J1 < 4) {
        MW1 = W3;
} else if (J1 < 5) {
        MW1 = W4;
} else if (J1 < 6) {
        MW1 = W5;
} else if (J1 < 7) {
        MW1 = W6;
} else {
        MW1 = W7;
}

MW2 = 1 - MW1;

if (I1 < 2) {
        MIS1 = IS1;
} else if (I1 < 3) {
        MIS1 = IS2;
} else if (I1 < 4) {
        MIS1 = IS3;
} else if (I1 < 5) {
        MIS1 = IS4;
} else if (I1 < 6) {
        MIS1 = IS5;
```

```
        } else if (I1 < 7) {
                MIS1 = IS6;
        } else if (I1 < 8) {
                MIS1 = IS7;
        } else  {
                MIS1 = IS8;
        }

        MIS2 = 1-MIS1;

/*"******** Perform the rule base evaluation*/

        MRA = MIN(MW2,MIS2);
        MRB = MIN(MW1,MIS2);
        MRC = MIN(MW2,MIS1);
        MRD = MIN(MW1,MIS1);

/*"******** Retrive the Delta Ts from the rule tabel for each fired rule
"First Rule:J1=1,I1=1*/



        if (I1 < 2) {
                C1 = 0.0201;
                C2 = 0.0201;
                C3 = 0.001;
                C4 = 0.001;
                C5 = 0.001;
                C6 = 0.001;
                C7 = 0.001;
                C8 = 0.001;
        } else if (I1 < 3) {
                C1 = 0.0342;
                C2 = 0.0342;
                C3 = 0.0201;
                C4 = 0.0201;
                C5 = 0.001;
                C6 = 0.001;
                C7 = 0.001;
                C8 = 0.001;
        } else if (I1 < 4) {
                C1 = 0.0601;
                C2 = 0.0601;
```

```
        C3 = 0.0342;
        C4 = 0.0342;
        C5 = 0.0201;
        C6 = 0.0201;
        C7 = 0.001;
        C8 = 0.001;
} else if (I1 < 5) {
        C1 = 0.0937;
        C2 = 0.0937;
        C3 = 0.0601;
        C4 = 0.0601;
        C5 = 0.0342;
        C6 = 0.0342;
        C7 = 0.0201;
        C8 = 0.0201;
} else if (I1 < 6) {
        C1 = 0.1998;
        C2 = 0.1397;
        C3 = 0.0937;
        C4 = 0.0937;
        C5 = 0.0601;
        C6 = 0.0601;
        C7 = 0.0342;
        C8 = 0.0342;
} else if (I1 < 7) {
        C1 = 0.2346;
        C2 = 0.1998;
        C3 = 0.1397;
        C4 = 0.1397;
        C5 = 0.0937;
        C6 = 0.0937;
        C7 = 0.0601;
        C8 = 0.0601;
} else if (I1 < 8) {
        C1 = 0.42;
        C2 = 0.35;
        C3 = 0.1998;
        C4 = 0.1998;
        C5 = 0.1397;
        C6 = 0.1397;
        C7 = 0.0937;
        C8 = 0.0601;
} else {
```

```
        C1 = 0.6;
        C2 = 0.42;
        C3 = 0.35;
        C4 = 0.2346;
        C5 = 0.2346;
        C6 = 0.1998;
        C7 = 0.1397;
        C8 = 0.0937;
}
if (J1 < 2) {
        R11 = C1;
} else if (J1 < 3) {
        R11 = C2;
} else if (J1 < 4) {
        R11 = C3;
} else if (J1 < 5) {
        R11 = C4;
} else if (J1 < 6) {
        R11 = C5;
} else if (J1 < 7) {
        R11 = C6;
} else {
        R11 = C7;
}

/*"******* Second Rule: J1=2,I1=1*/

L1 = J1+1;

if (L1 < 3) {
        R12 = C2;
} else if (L1 < 4) {
        R12 = C3;
} else if (L1 < 5) {
        R12 = C4;
} else if (L1 < 6) {
        R12 = C5;
} else if (L1 < 7) {
        R12 = C6;
} else if(L1 < 8) {
        R12 = C7;
} else  {
        R12 = C8;
```

```
        }


/*"******** Third Rule:J1=1,I1=2 */
        N1 = I1+1;


        if (N1 < 3) {
                D1 = 0.0342;
                D2 = 0.0342;
                D3 = 0.0201;
                D4 = 0.0201;
                D5 = 0.001;
                D6 = 0.001;
                D7 = 0.001;
                D8 = 0.001;
        } else if (N1 < 4) {
                D1 = 0.0601;
                D2 = 0.0601;
                D3 = 0.0342;
                D4 = 0.0342;
                D5 = 0.0201;
                D6 = 0.0201;
                D7 = 0.001;
                D8 = 0.001;
        } else if (N1 < 5) {
                D1 = 0.0937;
                D2 = 0.0937;
                D3 = 0.0601;
                D4 = 0.0601;
                D5 = 0.0342;
                D6 = 0.0342;
                D7 = 0.0201;
                D8 = 0.0201;
        } else if (N1 < 6) {
                D1 = 0.1998;
                D2 = 0.1397;
                D3 = 0.0937;
                D4 = 0.0937;
                D5 = 0.0601;
                D6 = 0.0601;
```

```
        D7 = 0.0342;
        D8 = 0.0342;
} else if (N1 < 7) {
        D1 = 0.2346;
        D2 = 0.1998;
        D3 = 0.1397;
        D4 = 0.1397;
        D5 = 0.0937;
        D6 = 0.0937;
        D7 = 0.0601;
        D8 = 0.0601;
} else if (N1 < 8) {
        D1 = 0.42;
        D2 = 0.35;
        D3 = 0.1998;
        D4 = 0.1998;
        D5 = 0.1397;
        D6 = 0.1397;
        D7 = 0.0937;
        D8 = 0.0601;
} else if(N1 < 9) {
        D1 = 0.6;
        D2 = 0.42;
        D3 = 0.35;
        D4 = 0.2346;
        D5 = 0.2346;
        D6 = 0.1998;
        D7 = 0.1397;
        D8 = 0.0937;
} else  {
        D1 = 1;
        D2 = 0.6;
        D3 = 0.42;
        D4 = 0.35;
        D5 = 0.35;
        D6 = 0.2346;
        D7 = 0.1998;
        D8 = 0.1397;
}

if (J1 < 2) {
        R21 = D1;
} else if (J1 < 3) {
```

```
            R21 = D2;
        } else if (J1 < 4) {
            R21 = D3;
        } else if (J1 < 5) {
            R21 = D4;
        } else if (J1 < 6) {
            R21 = D5;
        } else if (J1 < 7) {
            R21 = D6;
        } else {
            R21 = D7;
        }


/*"********** Fourth Rule:J1=2,I1=2 */

        if (L1 < 3) {
            R22 = D2;
        } else if (L1 < 4) {
            R22 = D3;
        } else if (L1 < 5) {
            R22 = D4;
        } else if (L1 < 6) {
            R22 = D5;
        } else if (L1 < 7) {
            R22 = D6;
        } else if (L1 < 8) {
            R22 = D7;
        } else  {
            R22 = D8;
        }


/*"********** Defuzzyfication using height method */

        SUM = MRA + MRB + MRC + MRD;
        DT = (R11*MRA+R12*MRB+R21*MRC+R22*MRD)/SUM;
        TS1 = DT*100;    /* " OUTPUT OF FUZZY TS ESTIMATOR*/

/*"TS2 = IF TS1<TSMIN THEN TSMIN ELSE MIN(TS1,TSMAX) */

        TSD = A1*TS1 + B1*TSD_OLD;
```

```
TERROR = TSD - TSD_OLD;
RS = 0.5814 + 0.0022837*TSD;
TSD_OLD =TSD;

TERROR_AUX = 0;
TC_AUX = 350 + 0.0034*we;

if(*FLUX_1 == 1)
{

if(TERROR > TERROR_AUX)
{ CR = 1;
  CR1 = 0; }
if(TERROR < -TERROR_AUX)
{ CR1 = 1;
  CR = 0; }

if(CR == 1)
{ TC = TC_AUX; }
if (CR1 == 1)
{ TC = 6*TC_AUX; }
}
else
TC = TC_AUX;

TC1 = TC + 0.001024;
TC2 = 1/TC1;
A1 =0.001024*TC2;
B1 = 1-A1;


dac0_var = *DAC0_reg;

if(dac0_var == 0)
{ dac0 = rec[0]>>20; }
else if(dac0_var == 1)
{ dac0 = rec[1]>>20; }
else if(dac0_var == 2)
{ dac0 = rec[2]>>20; }
else if(dac0_var == 3)
{ dac0 = rec[3]>>20; }
else if(dac0_var == 4)
{ dac0 = rec[4]>>20; }
```

```
else if(dac0_var == 5)
{ dac0 = rec[5]>>20; }
else if(dac0_var == 6)
{ dac0_aux = sin_e*4094;
dac0 = dac0_aux>>4;
}
else if(dac0_var == 7)
{ dac0_aux = cos_e*4094;
dac0 = dac0_aux>>4;
}
else if(dac0_var == 8)
{ dac0 = we_est; }
else if(dac0_var == 9)
{ dac0 = Te*100; }
else if(dac0_var == 10)
{ dac0 = iqs_e*100; }
else if(dac0_var == 11)
{ dac0 = ids_e*100; }
else if(dac0_var == 12)
{ dac0 = yds_e*2047; }
else if(dac0_var == 13)
{ dac0 = idq*100; }
else if(dac0_var == 14)
{ dac0 = wsl*100; }
else if(dac0_var == 15)
{ dac0 = vqs_int>>4; }
else if(dac0_var == 16)
{ dac0 = vds_int>>4; }
else if(dac0_var == 17)
{ dac0_aux = 4094*sinthe;
dac0 = dac0_aux>>4; }

else if(dac0_var == 18)
{ dac0_aux = 4094*costhe;
dac0 = dac0_aux >> 4;}




*dac0_output = dac0<<20;

/*dac1_var = *DAC1_reg;*/
```

```
if(dac1_var == 0)
{ dac1 = rec[0]>>20; }
else if(dac1_var == 1)
{ dac1 = rec[1]>>20; }
else if(dac1_var == 2)
{ dac1 = rec[2]>>20; }
else if(dac1_var == 3)
{ dac1 = rec[3]>>20; }
else if(dac1_var == 4)
{ dac1 = rec[4]>>20; }
else if(dac1_var == 5)
{ dac1 = rec[5]>>20; }
else if(dac1_var == 6)
{ dac1 = rec[6]>>20; }
else if(dac1_var == 7)
{ dac1_aux = sin_e*4094;
dac1 = dac1_aux>>4;
}
else if(dac1_var == 8)
{ dac1_aux = cos_e*4094;
dac1 = dac1_aux>>4;
}

else if(dac1_var == 9)
{ dac1 = iqsref*50; }
else if(dac1_var == 10)
{ dac1 = idsref*50; }
else if(dac1_var == 11)
{ dac1 = idq*50;   }
else if(dac1_var == 12)
{ dac1_aux = 4094*sinthe;
dac1 = dac1_aux>>4;}

else if(dac1_var == 13)
{ dac1_aux = 4094*costhe;
dac1 = dac1_aux>>4;}
*dac1_output = dac1<<20;

Comm2 = outputvec; /* Put starting address of array in absolute    */
                   /* location so the PC can find the array        */
if(logindex == 39)    /* The logging will happen at every 79.872ms */
 {
        logindex = 0;
```

```
        if(write_to_log == 0)
         {
                write_to_log = 1;
                *PCproceedflag = 1;
                if(x<k)
                 {
                  outputvec[x++] =ys_lp;
                  outputvec[x++] =ids_e;
                  outputvec[x++] =iqs_e;
                  outputvec[x++]  =check;
                  outputvec[x++] =idsref;
                  outputvec[x++]  =ids_aux;

                  x=0;

                 }
         }
        else
         {
                write_to_log = 0;
                *PCproceedflag = 0;
         }
}
else
 {
        ++logindex;
 }
/************** End of Log signals service ******************************/
} /**** End of Interrupt Routine #3 ****/

asm("  .sect \".int04\"");
asm("  .word _c_int04 ");
asm("  .text    ");
c_int04()
{
/***  Neuro-Fuzzy efficiency optimization codes      ***/

}

/*****************************************************************/
/************************ HOST PROGRAM ***********************/
/**** Description:                                              */
/****   This program runs in the host PC. Initially, it downloads */
```

```
/****  the executable file into the DSP. Afterwards, it receives          */
/****  commands from the keyboard, enabling change of parameters,         */
/****  change of waveforms that are being outputted into DAC's and        */
/****  triggering the acquisition of some signals into the PC             */

/**** Author: Nitin Patel                                                 */
/*************************************************************************/


#include <tms30.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
#include <dos.h>

#define board1                  0x290
#define board2                  0x390
#define VIDEO                   0x10
#define SCROLLUP                6
#define ROWS                    25
#define COLS                    80
#define NORMAL                  7

#define  COMM0                  0x30000   /* Start of .bss memory area.      */
#define  PCPROCEEDFLAG          COMM0 + 0  /* Absolute memory locations  */
#define  FLUXREF                COMM0 + 1
#define  OUTPUTVEC              COMM0 + 2
#define  vqs                    COMM0 + 3
#define  KPD                    COMM0 + 4
#define  KI                     COMM0 + 5
#define  KPQ                    COMM0 + 6
#define  FLUX_MODE              COMM0 + 7
#define  DAC0_reg               COMM0 + 8
#define  DAC1_reg               COMM0 + 9
#define  KP_F                   0x3000A
#define  SWITCH                 0x3000B
#define  IQS                    0x3000C
#define  IDS                    0x3000D
#define  N                      6       /* # of signals to acquire        */
#define  PIZERO                 0X3000E
#define KI_F                    0X3000F

void Clearscr()
```

```
{
 union REGS reg;
 reg.h.ah = SCROLLUP;
 reg.h.al = 0;
 reg.h.ch = 0;
 reg.h.cl = 0;
 reg.h.dh = ROWS - 1;
 reg.h.dl = COLS - 1;
 reg.h.bh = NORMAL;
 int86(VIDEO, &reg, &reg);
}



void main()
{
        void Clearscr();
        int    disp,dsp1,dsp2;                  /* Flags for switching DSP boards    */
        int    loadstat;                         /* code loading status flag          */
        int    reg1,i,k;                         /* register used in switch statement  */
        int    loop1,loop3;                      /* while loop index flag             */
         int flag,last_flag;
         float taq,value;
         int write_to_log,last_write_to_log;
         float vh_com;
         int maxacq,index,half_index;
          unsigned long  outputvecloc;
        static float   outputvec[N],log0[800],log1[800];
        static float   log2[800],log3[800];
        static float val,value1;
        float  kp_q,kp_d;
        float ki_qd;
        float kpf,kif;
        float flux_close,ys_ref;

        float      volt[71] =      {0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1,1.1,1.2,
                                    1.3,1.4,1.5,1.6,1.7,1.8,1.9,2,2.1,2.2,2.3,2.4,
                                    2.5,2.6,2.7,2.8,2.9,3,3.1,3.2,3.3,3.4,3.5,3.6,3.7,
                                    3.8,3.9,4,4.1,4.2,4.3,4.4,4.5,4.6,4.7,4.8,4.9,5,
                                    5.1,5.2,5.3,5.4,5.5,5.6,5.7,5.8,5.9,6,6.1,6.2,6.3,
                                    6.4,6.5,6.6,6.7,6.8,6.9,7};

        int        vhz,vhz1;
```

```
    char chvhz;
    float kv,kfe;
    int DAC0_var;
    int DAC1_var;
    int in1,in2,in3;
    int change_we;
    int mode;
    float iqs_ref,ids_ref;
    float iqs_old,ids_old;
    long adj,adj1;
    float teref;
    FILE *fp,*fopen();

    taq = 5;
    write_to_log = 0;
    last_write_to_log = 0;
    index = 0;
    half_index = 0;
    loop1 =1;
    k =0;
    vhz1 = 0;
    vhz = 0;
    kv = 0.1;
    kfe = 1.2;
    change_we = 0;
    mode = 0;
    iqs_old = 0;
    ids_old = 0;
    teref = 0;

/***** Initialize boards *****/
SelectBoard(board1);                /* Select DSP1   */
loadstat = coffLoad("bd1.out");     /* Download code */

if (loadstat != 0)     /* Check if code loaded properly */
{
 printf("\n\nError During Program Load in DSP1!!!!\n");
 printf("coffLoad() returned %x\n\n", loadstat);
 exit (0);
}
Reset();
```

```
  SelectBoard(board2);                /* Select DSP1   */
  loadstat = coffLoad("bd2.out");     /* Download code */

  if (loadstat != 0)     /* Check if code loaded properly */
  {
   printf("\n\nError During Program Load in DSP1!!!!\n");
   printf("coffLoad() returned %x\n\n", loadstat);
   exit (0);
  }
  Reset();

dsp1 = 1;
dsp2 = 0;
WarmSelect(board1);


PutFloat(vqs,DUAL,0);
PutFloat(KPD,DUAL,0);
PutFloat(KI,DUAL,0);
PutInt(DAC0_reg,DUAL,0);
PutInt(DAC1_reg,DUAL,0);
PutInt(SWITCH,DUAL,0);
PutFloat(IQS,DUAL,0);
PutFloat(IDS,DUAL,0);
PutFloat(KPQ,DUAL,0);
PutInt(PIZERO,DUAL,0);
PutFloat(FLUX_MODE,DUAL,0);
PutFloat(FLUXREF,DUAL,0.45);
PutFloat(KP_F,DUAL,0);
PutFloat(KI_F,DUAL,0);

   printf("\nC30 Program is running!\n");

   /***** Print menu selections *****/
   printf("\n\n Host Communication with DSP1");
   printf("\n\n0 - Print this screen\n");
   printf("1 - Halt DSP Program\n");
   printf("2 - Reset DSP Program\n");
  printf("3 - Quit\n");
  printf("4 - Switch DSP1 <==> DSP2\n");
  printf("5 - Indicate 2-channel readings \n");
  printf("6 - Enter new Ki for ids and iqs loop\n");
  printf("7 - Enter new kp and Ki for Flux loop \n");
```

```
  printf("8 - V/Hz operation for dsp1, Teref for dsp2\n");
  printf("9 - Select variable to output on DAC0\n");
  printf("10 - Select variable to output on DAC1\n");
  printf("11 - SWITCH V/HZ <==> DVC for dsp1, close torque loop for dsp2\n");
  printf("12 - Enter new iqs* value\n");
  printf("13 - Enter new ids* value\n");
  printf("14 - Close the Flux loop\n");
  printf("15 - Reset the PI's output\n");
  printf("16 - Start data acquisition\n");
  printf("17 - Enter new flux reference value\n");
  printf("18 - Enter new Kp and Ki for Torque loop \n");

 while (loop1 > 0)                    /* Main loop of program - keep looping until quit is
selected */
      {
       if (dsp1 > 0)
           {
           printf("\n\nEnter Selection for DSP1: ");
           }
         if (dsp2 > 0)
           {
           printf("\n\nEnter Selection for DSP2: ");
           }
         scanf("%d", &reg1);              /* Read selection entered by user */
         switch (reg1)
            {  case 0:
       if (dsp1 > 0)
              {
                  printf("\n\n Host Communication with DSP1");
              }
                  if (dsp2 > 0)
                        {
                        printf("\n\n Host Communication with DSP2");
                        }
                             printf("\n\n0 - Print this screen\n");

                             printf("1 - Halt DSP Program\n");
                             printf("2 - Reset DSP Program\n");
                             printf("3 - Quit\n");
                             printf("4 - Switch DSP1 <==> DSP2\n");
                             printf("5 - Indicate 6-channel readings \n");
                             printf("6 - Enter new Ki for ids and iqs loop\n");
                             printf("7 - Enter new Kp and Ki for Flux loop \n");
```

```
                printf("8 - V/Hz operation for dsp1, Teref for dsp2\n");
                printf("9 - Select variable to output on DAC0\n");
                printf("10 - Select variable to output on DAC1\n");
                printf("11 - SWITCH V/HZ <==> DVC for dsp1, close
                torque loop for dsp2\n");
                printf("12 - Enter new iqs* value\n");
                printf("13 - Enter new ids* value\n");
                printf("14 - Close the Flux loop\n");
                printf("15 - Reset the PI's output\n");
                printf("16 - Start data acquisition\n");
                printf("17 - Enter new flux reference value\n");
                printf("18 - Enter new Kp and Ki for Torque loop \n");
break;


case 1: AssertReset();
        Reset();
break;

case 2: Reset();
break;


case 3: WarmSelect(board1);
        AssertReset();
        WarmSelect(board2);
        AssertReset();
        loop1 = -1;
 break;
 case 4: if (dsp1 > 0)
          {
            dsp1 = 0;
            dsp2 = 1;
            WarmSelect(board2);
            Clearscr();
            printf("\nNow the selected board is DSP2");
            printf("\nType '0' for menu");
           }
          else
           {
            dsp2 = 0;
            dsp1 = 1;
            WarmSelect(board1);
```

```
            Clearscr();
            printf("\nNow the selected board is DSP1");
            printf("\nType '0' for menu");
           }
 break;

 case 5:        outputvecloc = Get32Bit(OUTPUTVEC,DUAL);
                RdBlkFlt(outputvecloc,DUAL,N,outputvec);
                /* Print values to PC screen:          */
        if (dsp1 > 0)
       {
            printf("\n\n Displaying variables from DSP1\n");
       }
          if (dsp2 > 0)
            {
            printf("\n\n Displaying variables from DSP2\n");
       }
        for (i=0; i<N; i++)
        {
         printf(" %8.4f\n", outputvec[i] );
        }
break;

case 6:
        printf("\nEnter new Ki for q and d-axes = ");
        scanf("%f", &ki_qd);
        PutFloat(KI, DUAL, ki_qd);
        break;

 case 7:
        printf("\nEnter new Kp for flux loop = ");
        scanf("%f", &kpf);
        PutFloat(KP_F, DUAL, kpf);
        printf("\nEnter new Ki for flux loop = ");
        scanf("%f", &kif);
        PutFloat(KI_F, DUAL, kif);

break;

case 8:
        if(dsp1>0)
        {
        vhz = vhz1;
```

```
        loop3 = 1;
        if(vhz == 0)
        {
         PutFloat(vqs, DUAL, kv*volt[vhz]);
        }
        printf("\n Type `u' to increase and `d' to decrease frequency");
        printf("\n OR type `q' to quit the v/hz operation\n");

        while(loop3 > 0)
        {
        chvhz = getche();
        switch(chvhz)
        {
                case 'u': if(vhz == 69)
                {
        printf("\n This is already the maximum frequency\n");
                break;
                }
        PutFloat(vqs, DUAL, kv*volt[vhz]);
        ++vhz;
        printf("\n Succesfully stepped up the frequency - type u/d or q\n");
        break;
        case 'd': if(vhz == 0)
                {
        printf("\n This is already the lowest frequency\n");
                break;
                }
        PutFloat(vqs, DUAL, kv*volt[vhz]);
        --vhz;
        printf("\n Succesfully stepped down the frequency - type u/d or
        q\n");
break;
case 'q': loop3 = -1;
        vhz1 = vhz;
        break;
        default: printf("Not a valid choice\n");
        break;

        }
        }
        }
if (dsp2 > 0)
        {
```

```
        printf("\nEnter new Reference value for torque = ");
        scanf("%f", &teref);
        PutFloat(vqs, DUAL, teref);
        }

break;



case 9:
        printf("Enter selection to display corresponding
        signal on DAC0 = \n");
        scanf("%d",&in1);
        PutInt(DAC0_reg,DUAL,in1);

break;

case 10:
        printf("Enter selection to display corresponding
        signal on DAC1 = \n");
        scanf("%d",&in2);
        PutInt(DAC1_reg,DUAL,in2);
break;

case 11:
        if(dsp1 > 0)
                {
                        printf("Enter 0-for V/Hz, 1-for DVC:");
                        scanf("%d",&mode);
                        PutInt(SWITCH,DUAL,mode);
                }
        if(dsp2 > 0)
                {
                        printf("Enter 0-for open, 1-for close torque loop:");
                        scanf("%d",&mode);
                        PutInt(SWITCH,DUAL,mode);
                }
break;

case 12:

                 printf("\n Enter new value:iqs* = ");
                scanf("%f",&iqs_ref);
```

```
                PutFloat(IQS,DUAL,iqs_ref);
                taq = 15;
                maxacq = taq/0.079872;  /* Write_to_log occurs each
                                           79.872ms */

            /**** Receive the signals to be logged  *****/
            index = 0;
            do
             {
                        write_to_log =
Get32Bit(PCPROCEEDFLAG,DUAL);
                 if  (write_to_log == 0x1L)
                {
                 if (last_write_to_log == 0)
                  {
                        outputvecloc = Get32Bit(OUTPUTVEC,DUAL);
                        RdBlkFlt(outputvecloc,DUAL,N,outputvec);
                        log0[index] = outputvec[0];
                        log1[index] = outputvec[1];
                        log2[index] = outputvec[2];
                        log3[index] = outputvec[3];
                         index++;
                   }
                }
          last_write_to_log = write_to_log;
             }
            while (index<=maxacq);
            index = 0;

             if (dsp1 > 0)
          {
        /**** Write the logged signals into a file *****/
              if ((fp = fopen("log1", "a")) == NULL)
              printf("Write to disk in the log1 file has failed\n");
              }

             if (dsp2 > 0)
          {
        /**** Write the logged signals into a file *****/
              if ((fp = fopen("log2", "a")) == NULL)
              printf("Write to disk in the log2 file has failed\n");
              }
```

```
                    for (index=0; index <= maxacq; index++)
                {
                        fprintf(fp, "%8.4f  %8.4f  %8.4f  %8.4f
%8.4f\n",index*0.079872,log0[index],log1[index],log2[index],log3[index]);
                    }
                    fclose(fp);
                    printf("\nThe acquisition service is finished");

            break;

            case 13:

                printf("\n Enter new value:ids* = ");
                scanf("%f",&ids_ref);
                PutFloat(IDS,DUAL,ids_ref);
            break;
            case 14:
                if(dsp2 > 0)
                    {
                        printf("Enter 1 to close flux loop: ");
                        scanf("%f",&flux_close);
                        PutFloat(FLUX_MODE,DUAL,flux_close);
                    }
            break;
            case 15: if (dsp2 > 0)
                    {
                        PutInt(PIZERO,DUAL,1);
                    }
            break;

            case 16:    if (dsp1 > 0)
                        {
                            printf("\n\n Acquisition with DSP1\n");
                        }
                       if (dsp2 > 0)
                        {
                            printf("\n\n Acquisition with  DSP2\n");
                        }

                    printf("\ntaq = %.3f, Enter acquisition time (max 60s) =  ", taq);
                    scanf("%f", &value);
                      taq = value;
                      maxacq = taq/0.079872;  /* Write_to_log occurs each 79.872ms
```

```
*/

    /**** Receive the signals to be logged  *****/
    index = 0;
    do
     {
  write_to_log = Get32Bit(PCPROCEEDFLAG,DUAL);
          if  (write_to_log == 0x1L)
          {
          if (last_write_to_log == 0)
           {
                outputvecloc = Get32Bit(OUTPUTVEC,DUAL);
                RdBlkFlt(outputvecloc,DUAL,N,outputvec);


          log0[index] = outputvec[0];
          log1[index] = outputvec[1];
          log2[index] = outputvec[2];
          log3[index] = outputvec[3];

             index++;
             }
          }
  last_write_to_log = write_to_log;
       }
     while (index<=maxacq);
     index = 0;

      if (dsp1 > 0)
  {
/**** Write the logged signals into a file *****/
        if ((fp = fopen("log1", "a")) == NULL)
        printf("Write to disk in the log1 file has failed\n");
        }

      if (dsp2 > 0)
  {
/**** Write the logged signals into a file *****/
        if ((fp = fopen("log2", "a")) == NULL)
        printf("Write to disk in the log2 file has failed\n");
        }
```

```
                    for (index=0; index <= maxacq; index++)
                {
                        fprintf(fp, "%8.4f  %8.4f  %8.4f  %8.4f
%8.4f\n",index*0.079872,log0[index],log1[index],log2[index],log3[index]);
                    }
                    fclose(fp);
                    printf("\nThe acquisition service is finished");
            break;

        case 17:
                    printf("\n Enter new value:Fs* = ");
                    scanf("%f",&ys_ref);
                    PutFloat(FLUXREF,DUAL,ys_ref);
        break;

        case 18:    printf("\nEnter new Kp for  Torque loop = ");
                    scanf("%f", &kp_d);
                    PutFloat(KPD, DUAL, kp_d);
                    printf("\nEnter new Ki for Torque loop = ");
                    scanf("%f", &kp_q);
                    PutFloat(KPQ, DUAL, kp_q);

        break;
                    default: printf("\nInvalid Selection - Try Again\n");
                     break;
            } /* End switch */
        } /* End while loop */
} /*  End of main()  */
```

VITA

Nitinkumar R Patel was born in Gujarat, India, on July 24, 1968. He received his Bachelor of Science degree in Instrumentation and Controls from University of Poona, India in December 1991. Upon graduation he joined Unix classes with Apple Computers Inc, India. In January 1993, he accepted a position for an assistant engineer at P. C. Tech., a computer company in Nashville, TN. In January 1994 he was accepted toThe University of Tennessee, Knoxville, as a graduate student in the Department of Electrical Engineeering. Since then he is actively engaged in the research of the application of fuzzy logic and neural networks to power electronics, drives and machine controls. During his study at The University of Tennessee, two of three papers were accepted for publication in IEEE Industrial Electronics Transactions and the third one is under review.

Upon graduation he would like to continue his research and development on artificial intelligence in power electronics and drives.