Advanced DSP Control of Induction Motors using Kalman Filter

by

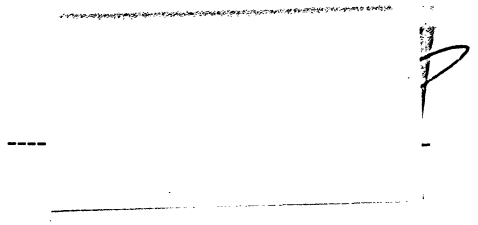
Shiping Zhu

A project
presented to Ryerson University
in partial fulfillment of the
requirement for the degree of
Master of Engineering
in the Program of
Electrical and Computer Engineering
Toronto, Ontario, Canada, 2003

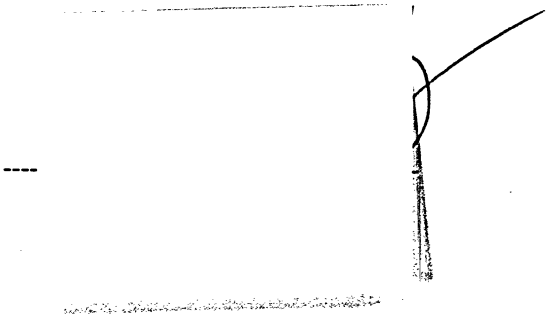
Author's Declaration

I hereby declare that I am the sole author of this research paper.

I authorize Ryerson University to lend this research paper to other institutions or individuals for the purpose of scholarly research.



I further authorize Ryerson University to reproduce this research paper by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.



Instructions on Borrowers

Ryerson University requires the signatures of all persons using or photocopying this research paper. Please sign below, and give address and date.

Advanced DSP Control of Induction Motors using Kalman Filter

Abstract

This research paper presents a novel method for the speed control of induction motors without using a speed sensor. The rotor speed can be accurately computed using an optimal control observer named the Kalman filter designed in this research paper. This replaces a speed sensor and eliminates the difficulty of the sensor installation in many applications. This research paper presents an advanced field oriented control of induction motors, based on a specific d-q coordinate model with the d-coordinate chosen to be in line with the rotor flux and the q-coordinate chosen to be 90° lagging. The position of the rotor flux can be accurately computed using the Kalman filter. This eliminates the position sensors required to monitor the flux. This research paper shows that as a result of this specific d-q transformation, the motor torque is proportional to the product of the rotor flux and the q-coordinate stator current. This significantly simplifies the induction motor control, such that the rotor flux is simply controlled by regulating the flux-related d-coordinate stator current and the motor torque is controlled just by regulating the q-coordinate stator current. This research paper presents a computational efficient recursive algorithm for Kalman filter which is specifically designed for the induction motor control. The Kalman filter provides the minimum variance state estimation and tolerates induction motor modeling and measurement errors. The Kalman filter can process all available measurements regardless of their precision (only two input current measurements required for the motor control), and provides a quick and optimal estimate of the variables of interest (the rotor speed and flux selected as outputs), as well as achieves a fast convergence. This research paper presents the digital signal processor (DSP) implementation of the field oriented control of induction motors using Kalman filter. The hardware requirements and all software modules are detailed. The experimental verification of the control method designed in this research paper is provided. Typical measurements are given to demonstrate the efficiency of the novel control presented in this research paper.

Key words: field oriented control, Kalman filter, DSP control, PID regulator

Acknowledgments

I would like to express my sincere gratitude and appreciation to my supervisor Professor Richard Cheung for his guidance in my study and project. I cordially thank Professor Bin Wu for his advice and help in my study period.

At the same time I would like to thank my classmates or friends in the laboratory of power electronics at Ryerson University for their help or assistance in my project.

Specially thanks to my wife and son for their full support.

Contents

1 Introduction 1
1.1 DSP Control of Induction Motors 2
1.2 Speed Sensorless Control of Induction Motors 3
1.3 Novel Control Method for Induction Motors 4
1.4 Outline of the Research Paper 6
2 Advanced Field Oriented Control of Induction Motors 8
2.1 3-phase to 2-phase Transformation 9
2.2 Space Vector PWM Control of Motor Drive Inverter 11
2.3 Field Oriented Control 15
2.4 Adaptive PID Regulator 17
3 Kalman Filter for Speed Sensorless Control of Induction Motors 20
3.1 Basics of Kalman Filter 20
3.2 Induction Motor Model for Implementation of Kalman Filter 23
4 Digital Signal Processor Implementation of Speed Sensorless Field Oriented Control 28
4.1 Overview of DSP FOC Operations 28
4.2 Hardware Implementation of Field Oriented Control 29
4.3 DSP Software Implementation of Field Oriented Control 31
4.4 Debugging of Software and Hardware 33
5 Experimental Verification of Field Oriented Motor Control using Kalman Filter 36
5.1 Internal States of Closed-Loop Control 36
5.2 Dynamic Response 44
5.3 Speed Control Accuracy 46
6 Conclusions 47
Bibliography 50
Appendix A Summary of Notations 52
Appendix B Induction Motor Parameters 53
Appendix C Characteristics of PID Control 54
Appendix D Assembly Code for TMS320LF2407 55

List of Figures

Figure 1: Methods of Sensorless Speed Control 4
Figure 2: The Control Structure of FOC with Kalman Filter 5
Figure 3: α-β Transformation 10
Figure 4: Park Transformation 11
Figure 5: Three-phase Voltage Source Inverter 12
Figure 6: Configuration of Space Vector PWM 13
Figure 7: Example of Space Vector PWM pattern 15
Figure 8: Field Oriented Control of Induction Motors 16
Figure 9: Relationship between Current, Voltage and Rotor Flux 16
Figure 10: Sensorless Controller for Induction Motors 17
Figure 11: The Adaptive PID Control 18
Figure 12: Principle of Kalman Filter 21
Figure 13: Extended Kalman Filter Algorithm 23
Figure 14. Experiment Setup for Sensorless Control System 29
Figure 15: Block Diagram of eZdsp™ LF2407 30
Figure 16: Software Flowchart 31
Figure 17: EKF Software Structure for an Induction Motor 32
Figure 18: Overview of Phase 2 Software Flow 34
Figure 19: Overview of Phase 2 Workspace 35
Figure 20: ia, ib Currents from ADC 37
Figure 21: Currents from Clarke Transformation 37
Figure 22: Currents Estimated by Kalman Filter 38
Figure 23: Flux Circle Estimated by Kalman filter 38
Figure 24: Currents from Park Transformation 39
Figure 25: Speed Control 39
Figure 26: iq Control 40
Figure 27: id Control 40
Figure 28: Voltage Outputs from id, iq Regulators 41
Figure 29: Ualfa, Ubeta from Inverse Park Transformation 41
Figure 30: Duty Ratios of PWM1, PWM3 and PWM5 in Carrier Period 42
Figure 31: PWM Outputs of IGBT Power Module 43
Figure 32: Line to Line Voltages of the Induction Motor 43
Figure 33: Speed Step Response 44
Figure 34: Speed Transient from 0 to 1000rpm 45
Figure 35: Speed Reversion 45
Figure 36: The Model of an Induction Motor 53

List of Tables

Table 1: Steady State Performance of Speed Sensorless Control 46
Table 2: Summary of Notations Used in this Research Paper 52

Chapter 1

Introduction

Induction motors are popular in the industry because they are reliable, rugged, low cost, and virtually maintenance free. Market analysis shows that most of industrial motor applications use induction motors. However, the use of induction motors may have several difficulties, which include control disadvantages due to complex motor model and nonlinear motor parameters caused by saturation and variations with temperature. In order to overcome these difficulties, many methods have been proposed for the control of induction motors [1-5]. The field-oriented control (FOC) is the most popular one. This control requires the instantaneous measurements of the motor speed, voltages and currents. The speed of the motor is usually measured using a speed/position sensor. The installation of the sensor not only increases the motor drive cost and the sensor maintenance cost, but also the installation could be very difficult or impractical for some applications such as the pumps used in oilrigs.

This research paper presents a novel method for controlling the speed of induction motors without the need of a speed or position sensor. The information of the motor speed is obtained by processing the stator voltages and currents which are simply measured at the motor terminals or at the supply terminals. Induction motor drives without speed or position sensors will increase the drives reliability and reduce their costs. This research paper has developed an efficient control for induction motors using a closed-loop optimal control observer named the Kalman filter to compute the motor speed and position, as well as using an adaptive feedback regulator to provide a reliable and accurate speed control.

The following lists the key work carried out in this research paper.

a. *Full investigation of Kalman filter for the control of induction motors.*
This research paper provides a full investigation of the practical application of Kalman filter for the control of induction motors. There has been very limited information about the use of Kalman filter for induction motor drives, possibly because of the difficulty of understanding the theory of the Kalman filter. This research paper research provides a detailed study of the control theory of the Kalman filter and its suitability for the control of induction motors. This research provides a concrete finding that Kalman filter is well suitable for induction motor control.

b. *Detailed formulations of the specific field oriented control with Kalman filter for real-time control of induction motors.*

This research paper develops all formulations required for the specific field oriented control of induction motors using Kalman filter. In particular, a recursive algorithm of the Kalman filter is given in the research paper. The algorithm and its corresponding formulations has provided an accurate computation of the rotor speed, the flux linkage, and the flux angle which are the essential components for the specific field oriented control of induction motors.

c. *DSP implementation of the specific field oriented induction motor control.*
This research paper develops the software required for the implementation of the specific field oriented motor control in DSP TMS320LF2407. The software is developed in modules which can be easily modified for different induction motor drive applications.

d. *Real-time verification of the specific field oriented control using Kalman field.*
A prototype of the induction motor drive is built to verify the specific field oriented control using Kalman filter. Typical measurements are given to demonstrate the efficiency of the novel control designed in this research paper.

This chapter provides an introduction of the control method developed in this research paper. The following outlines the sections in this chapter.

Section 1.1 discusses the digital signal processor (DSP) control of induction motors.
Section 1.2 assesses the research in speed sensorless control of induction motors.
Section 1.3 introduces the novel control method designed in this research paper. This control combines the field oriented control and the Kalman filter.
Section 1.4 shows the structure of this research paper.

1.1 DSP Control of Induction Motors

Induction motor controls were traditionally designed and implemented with analog components. There are several drawbacks with analog controllers including component aging and drifts. Analog controllers are usually hard-wired circuits that make modifications or upgrades fairly difficult. Also the implementation of advanced control algorithms using analog circuits requires an excessive number of components. On the other hand, digital control offers improvements over the analog control. The effect on the control due to analog component characteristic variations can be eliminated in the digital control. Upgrades in the digital control can fairly easily conducted in software, and the number of control circuit components required is significantly reduced since the digital controller can handle many control functions in a single chip.

Digital signal processor (DSP) can provide high speed and high accuracy control signals. Fixed-point DSPs are preferable in the cost-effective induction motor control, because

fixed-point DSPs cost much less than the floating-point ones and for most applications, the dynamic range of 16 bits is sufficient. If needed for certain small portions in the control computation, the dynamic range can be increased in the fixed-point processor by carrying out floating-point calculations.

The performance of an induction motor strongly depends on the characteristics of its control. DSPs can be used to enhance the real-time control of induction motors without the use of any electromechanical speed sensors. The DSP controller can reduce the number of control circuit components and can optimize the drive performance. The DSP for motor drive applications can perform the following.

1) Handle complex control schemes such as implementation of sensorless motor control algorithms to eliminate the need of a speed or position sensor.
2) Implement communication functions for fault and status information.
3) Re-program control schemes for different applications.
4) Generate high-resolution PWM signals for efficient control of the power electronic inverter and reduction of harmonics.
5) Provide a single chip drive control system and reduce the drive system cost.

1.2 Speed Sensorless Control of Induction Motors

The world market of induction motor drives is over ten thousand million US dollars annually with a rapid growth. Current researches on motor drives have focused on the elimination of the speed sensor at the motor shaft without deteriorating the dynamic performance of the drives [6, 7]. The speed estimation is of particular interest with the induction motor drives where the mechanical speed of the rotor is generally different from the speed of the revolving magnetic field. The advantages of speed-sensorless induction motor drives include reduction of hardware complexity and cost, increase of control-circuit noise immunity and drive reliability, and reduction of maintenance requirements. Operations in the hostile environments mostly require motor drives without speed/position sensors.

Figure 1 shows the block diagram of the speed-sensorless induction motor control. Basically there are two commonly-used control methods: the voltage-to-frequency (V/f) control and the field oriented control (FOC). Both methods for the speed sensorless control require a speed estimation algorithm. In the V/f control, the ratio of the stator voltage to stator frequency is kept constant using a feed forward control to maintain the magnetic flux in the motor at a desired level. This control is simple but it only can satisfy moderate motor dynamic requirements. On the other hand, high motor dynamic performance can be achieved using the FOC control which is also called the vector control. The stator currents are injected at a well-defined phase angle with respect to the spatial orientation of the rotating magnetic field, thus overcoming the complex dynamic properties of the induction motor. The spatial location of the magnetic field, that is the field angle, is difficult to measure.

There are several models and algorithms that can be used for the estimation of the field angle, for example the open-loop estimator such as model reference adaptive system (MRAS), or the close-loop observer such as the Kalman filter. The induction motor control using the field orientation usually refers to the rotor field. This research paper presents a novel rotor field oriented control.

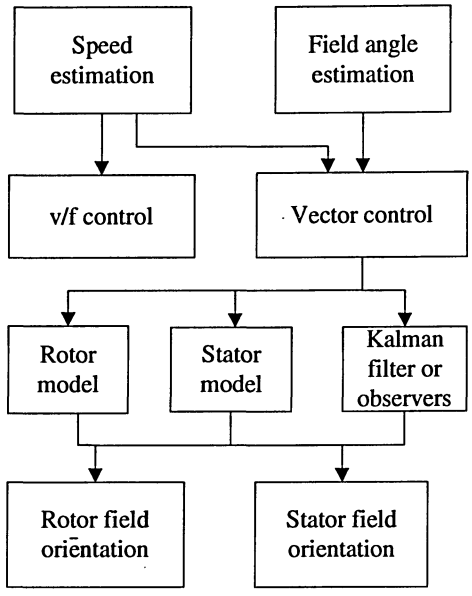


Figure 1: Methods of Sensorless Speed Control

1.3 Novel Control Method for Induction Motors

This research paper investigates the real-time field oriented control (FOC) of induction motors using a Kalman filter and an adaptive feedback controller. This FOC control has been successfully implemented in the Power Electronic Laboratory at Ryerson University. Figure 2 shows the basic control structure. The main contribution of this research paper consists of achieving accurate sensorless control using a Kalman fiter implemented in a DSP processor.

1.3.1 Field Oriented Control

An efficient vector control scheme for induction motors has been implemented in this research paper based on the following three major points:

1) Transform the motor's three-phase ever-changing rotor-position dependent model of control and computation disadvantages into a two-coordinate motor-speed model for efficient computation and control.
2) Formulate the current and voltage space vectors for high performance motor speed and torque control.

3) Generate the pulse-width-modulation (PWM) signals to accurately control the power electronic inverter that drives the induction motor.

Separately-excited dc motor drives are simple in control, because they independently control of the motor's flux and torque through two separate controls of the field current and the armature current. The vector control would like to make the induction motor drive similar to the dc motor drive with two separate controls of the motor flux and the torque. Furthermore, the vector control method can achieve an accurate steady state and transient control that leads to high dynamic performance of the induction motor in a fast response to the changes of its load.

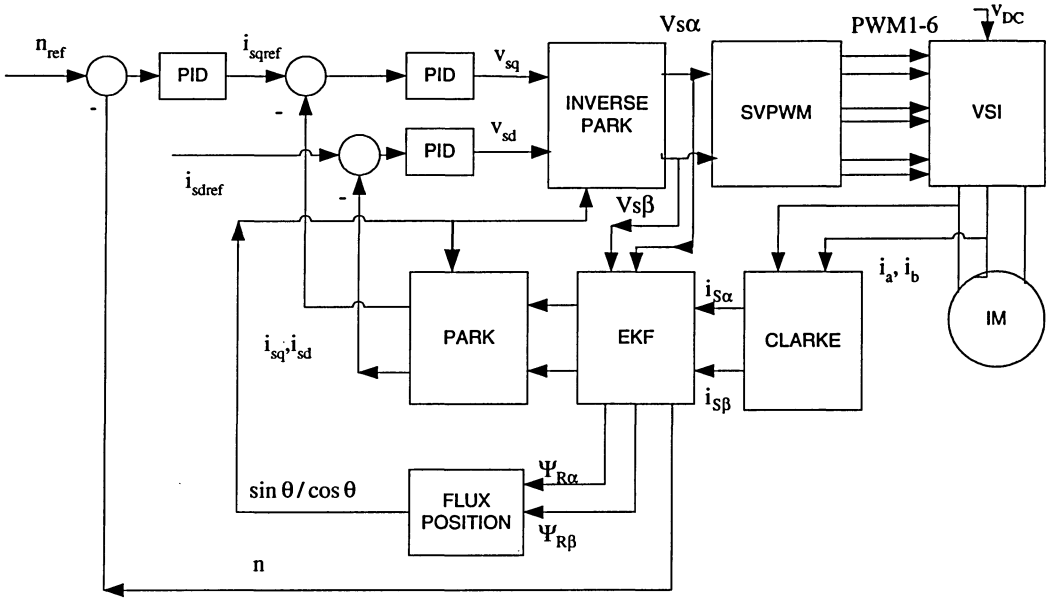


Figure 2: The Control Structure of FOC with Kalman Filter

1.3.2 Kalman Filter

The conventional induction motor drive uses electromechanical sensors to measure the position and speed of the rotor. However in many applications such as the pumps used in oil rigs working under the sea, it would not practical to use sensors for the speed or position measurement as it is either technically impossible or extremely expensive. Recently, there have been several proposals addressing this problem and showing that the motor speed can be calculated from the current and voltage inputs to the motor. For a better motor performance, this research paper has designed an observer named the Kalman filter for an efficient control of induction motors. This filter is a statistically optimal observer, which has a good dynamic performance, disturbance rejection, and capability of working in a standstill position.

In general, the implementation of a Kalman filter in the motor control is fairly involved,

because it requires an accurate model of the induction motor to be calculated in real time. The filter equation to be evaluated involves matrix computations. Digital signal processor (DSP) is especially suitable for this kind of numerical computations. An economical fixed-point DSP TMS320LF2407 from the Texas Instrument is selected for the implementation in this research paper. The design of the Kalman filter of induction motor control is detailed in Chapter3.

1.3.3 Adaptive PID Controller

An adaptive controller can accommodate unpredictable changes, whether these changes arise within the system or external to it, such as design and measurement errors or uncertainties. This research paper has designed an adaptive proportional-integral-derivative (PID) control to overcome the control uncertainties and to enhance the quality of the induction motor control. The design of adaptive PID controllers for the induction motor speed and torque control is detailed in Chapter 2.

1.3.4 Modular Software for Induction Motor Controls

The benefits of structured modulator software are well known especially for large control systems with many sub-blocks. In examination of various motor control systems, it becomes clear that a large degree of commonality exists between the control software modules. Therefore, if each module is implemented according to well-defined guidelines, then compatibility can be assured across all modules. Since the modularity approach allows efficient software re-uses, efforts have been put on expanding the module library base for greater functionality and features in this research paper. As a form of modularity strategy, the Kalman filter module designed in this research paper can be easily transferred to different induction motor control applications.

1.4 Outline of the Research paper

The outline of the following chapters is given below.

Chapter 2 presents an advanced field-oriented control for induction motors. This chapter examines the key components of the field oriented control including the 3-phase to 2-phase transformation, the space vector PWM technique, and the separate induction motor flux and torque control.

Chapter 3 develops the Kalman filter for induction motor control. This chapter details all the formulations required to accurately compute the flux angle for the speed sensorless control of an induction motor.

Chapter 4 discusses the hardware and software implementation of the novel control scheme designed in this research paper using the DSP TMS320LF2407. This chapter details all key hardware components / circuits and software modules.

Chapter 5 presents the experimental results to demonstrate the accuracy and efficiency of the FOC control designed in this research paper.

Chapter 6 provides a conclusion of this research paper research and recommendations for future work.

Appendix A summarizes the notations used in this research paper.
Appendix B provides the parameter values of the induction motor used in the experiment.
Appendix C shows the characteristics of the PID controllers.
Appendix D gives all software modules for speed sensorless control of the induction motor.

Chapter 2

Advanced Field Oriented Control of Induction Motors

This chapter presents an advanced field-oriented control (FOC) of induction motors, based on mathematical transformations of the standard three-phase motor model into a specific two-phase d-q coordinate model. The three-phase induction motor model is a complex matrix equation with sinusoidal functions of the rotor angle. This model has inherent control and computation disadvantages due to the ever-changing rotor angle. This can be greatly improved with a simple transformation of the three-phase model into a d-q model with the two coordinates rotate with respect to the rotor flux at the synchronous speed. This chapter shows the transformation which, as an extension of the well-known Park's Transformation, provides the induction motor control and computation efficiency.

The three-phase instantaneous stator currents of the induction motor may be modeled as a complex space vector which can be transformed into two currents: the stator d-coordinate current and the stator q-coordinate current, with the d-coordinate chosen to be in line with the rotor flux and the q-coordinate chosen to be 90° lagging. This chapter shows that as a result of the specific d-q transformation, the motor torque is proportional to the product of the rotor flux and the q-coordinate stator current. The rotor flux of the induction motor can be controlled through the regulation of the flux-related d-coordinate stator current. By maintaining the rotor flux to a desired value, there exists a linear relationship between the motor torque and the q-coordinate stator current. The motor torque can be controlled by regulating the q-coordinate stator current. Therefore, this transformation leads to a simple control structure similar to that of a separately-excited dc motor drive where the flux and the torque of the dc motor can be controlled separately. This transformation also provides significant computational advantage in the field-oriented control of induction motors.

The following outlines the sections in this chapter.

Section 2.1 presents the three-phase to two-phase transformation. This transformation is implemented in two steps. First, transform the three-phase system into the two orthogonal coordinate system. This transformation is known as α-β transformation or Clarke's transformation. Second, transform the α-β coordinates into the d-q coordinates, with the d-coordinate chosen to be in line with the rotor flux and the q-coordinate chosen to be 90° lagging. This transformation can be regarded as an extension of the Park's transformation.

Section 2.2 presents a space-vector pulse-width-modulation (PWM) technique for the control of a three-phase voltage-source inverter (VSI) that drives the induction

motor. The space-vector PWM technique is used to generate the desired instantaneous reference voltages for the VSI from the corresponding basic space vectors according to the switching states.

Section 2.3 presents the field oriented control which provides an efficient real-time control of the torque of the induction motor which in terms controls the motor mechanical speed.

Section 2.4 presents the design of PID regulator used to regulate the motor torque and flux to the desired values.

2.1 3-phase to 2-phase Transformation

The three-phase voltages, currents and fluxes of an induction motor can be modeled in terms of complex space vectors. The space vector for the stator currents of the motor is defined as follows.

Assuming that i_a, i_b, i_c are the three instantaneous currents in the stator, the complex stator current vector i_S is defined by:

$$\bar{i}_S = i_a + e^{j\frac{2}{3}\pi} i_b + e^{j\frac{4}{3}\pi} i_c \tag{1}$$

This current space vector can be transformed into two currents: the d-coordinate current and the q-coordinate current by the following two steps:

Step 1: Transform the three-phase system into two orthogonal coordinate system. This transformation is known as α-β transformation or Clarke's transformation.

Step 2: Transform the α-β coordinates into the d-q coordinates, with the d-coordinate chosen to be in line with the rotor flux and q-coordinates chosen to be 90° lagging. This transformation can be regarded as an extension of the Park's transformation.

2.1.1 α-β Transformation (Clarke's Transformation)

The current space vector can be transformed into two orthogonal axes named α and β. The α-coordinate is chosen in line with the phase "*a*", and the β-coordinate is 90° lagging. Then the α-β coordinate currents and the three phase instantaneous currents are related as follows:

$$\begin{aligned} i_{S\alpha} &= i_a \\ i_{S\beta} &= \frac{1}{\sqrt{3}} i_a + \frac{2}{\sqrt{3}} i_b \end{aligned} \tag{2}$$

where $i_a+i_b+i_c=0$ is assumed. To further illustrate the above relationship, assume the three phase currents are balanced as given below:

$$\begin{aligned} i_a &= I\sin(\omega t) \\ i_b &= I\sin(\omega t + 2\pi/3) \\ i_c &= I\sin(\omega t - 2\pi/3) \end{aligned} \tag{3}$$

Then the α-β coordinate current becomes:

$$\begin{aligned} i_{S\alpha} &= I\sin(\omega t) \\ i_{S\beta} &= I\sin(\omega t + \pi/2) \end{aligned} \tag{4}$$

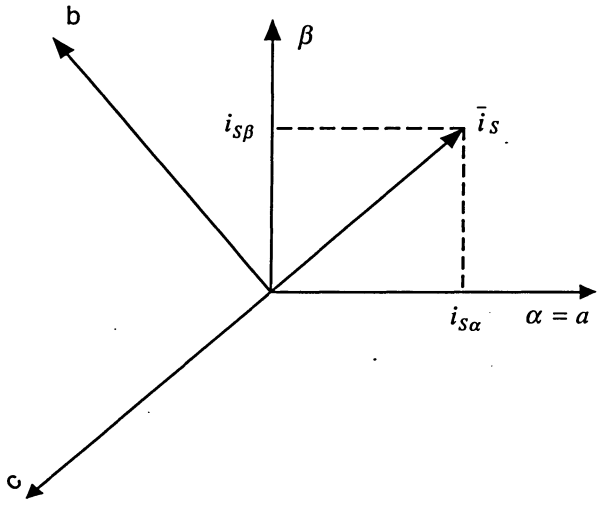


Figure 3: α-β Transformation

2.1.2 d-q Transformation (Park's Transformation)

The core transformation in the field oriented control (FOC) is the d-q transformation, which can be regarded as an extension of the Park's transformation. The α-β coordinates are transformed into the d-q coordinates, with the d-coordinate chosen to be in line with the rotor flux Ψ_R and the q-coordinate chosen to be 90° lagging. Figure 4 shows the relationship between the current vectors in the two reference frames.

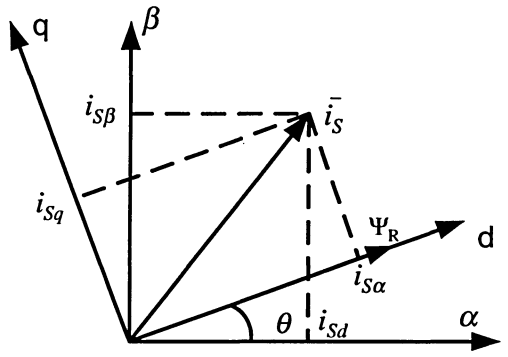


Figure 4: Park Transformation

In Figure 4, θ is the rotor flux angle. The d and q current components are determined by the following equations:

$$\begin{aligned} i_{Sd} &= i_{S\alpha}\cos\theta + i_{S\beta}\sin\theta \\ i_{Sq} &= -i_{S\alpha}\sin\theta + i_{S\beta}\cos\theta \end{aligned} \tag{5}$$

The d-q coordinate system has the following characteristics:

- It is a two-coordinate time-invariant system; both i_{sd} and i_{sq} are dc quantities.
- The current i_{sd} is a flux related component and the current i_{sq} is a torque related component.
- The torque of the motor can be easily controlled using these dc current data.

2.2 Space Vector PWM Control of Motor Drive Inverter

Space-vector pulse-width-modulation (PWM) technique has become a popular PWM technique for the control of three-phase voltage-source inverters (VSI) for applications such as induction motor drives. In comparison to the direct sinusoidal PWM technique, the space-vector PWM technique generates less harmonic distortion in the output voltages and currents and provides more efficient use of the dc supply voltage to the inverter [3].

Figure 5 shows a basic three-phase power inverter circuit, where *Va*, *Vb*, *Vc* are the voltages applied to the induction motor, and where *Vdc* is the inverter dc input voltage.

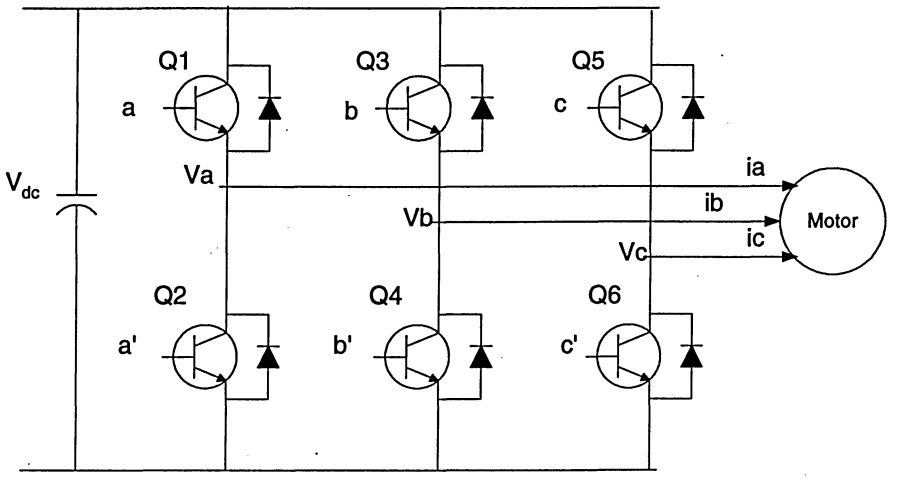


Figure 5: Three-phase Voltage Source Inverter

The six switches *Q1* to *Q6* are insulated-gate-bipolar-transistors (IGBT). The ON-OFF sequence of these switches follows the conditions below:

- Three of the switches must be always ON and three are always OFF.
- The upper and the lower switches of the same leg are driven with two complementary pulsed signals with dead band between the two signals to avoid short-circuit.

The induction motor is supplied with the required three phase voltages for the designed operating conditions using the PWM technique. In this research paper, the space-vector PWM method is used to generate the gating signals for the switches in the VSI inverter that drives the induction motor with high performance in terms of fast response to changes of loads and speed commands.

The relationship between the switching variable vector $[a\ b\ c]^T$ and the line-to-line output voltage vector $[V_{ab}\ V_{bc}\ V_{ca}]^T$ and the phase voltage vector $[V_a\ V_b\ V_c]^T$ is given by the following equations.

$$\begin{bmatrix} V_{ab} \\ V_{bc} \\ V_{ca} \end{bmatrix} = V_{dc} \begin{bmatrix} 1 & -1 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{6}$$

$$\begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} = \frac{1}{3} V_{dc} \begin{bmatrix} 2 & -1 & 1 \\ -1 & 2 & -1 \\ -1 & -1 & 2 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} \tag{7}$$

where $V_{ab}=V_a-V_b$, $V_{bc}=V_b-V_c$, $V_{ca}=V_c-V_a$. The stator α-β voltages corresponding to the three-phase voltages are:

$$V_{S\alpha} = V_a$$
$$V_{S\beta} = \frac{1}{\sqrt{3}} V_a + \frac{2}{\sqrt{3}} V_b \tag{8}$$

The above equation can also be expressed in matrix form by using the equation $V_a+V_b+V_c=0$.

$$\begin{bmatrix} V_{s\alpha} \\ V_{s\beta} \end{bmatrix} = \frac{2}{3} \begin{bmatrix} 1 & -\frac{1}{2} & -\frac{1}{2} \\ 0 & \frac{\sqrt{3}}{2} & -\frac{\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} V_a \\ V_b \\ V_c \end{bmatrix} \tag{9}$$

There are eight (2^3) possible combinations for the switch commands. These eight switch combinations determine the eight phase voltage vectors, of which the results are six non-zero vectors (V_1-V_6) and two zero vectors (V_0, V_7) as shown in Figure 6.

The objective of space vector PWM technique is to generate the desired instantaneous reference voltages from the corresponding basic space vectors based on the switching states. Figure 6 shows that the basic space vectors divide the plan into six sectors. Depending on the sector that the reference voltage is in, two adjacent basic vectors are chosen. The two vectors are time weighted in a sample period T (PWM period) to produce the desired output voltage.

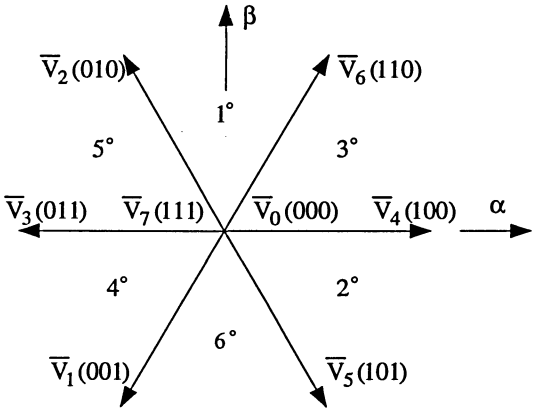


Figure 6: Configuration of Space Vector PWM

Assuming that the reference vector V_{out} is in the sector 3 as shown in Figure 7a, the application time of two adjacent vectors is given by:

$$T = T_4 + T_6 + T_0$$
$$\overline{V}_{out} = \frac{T_4}{T}\overline{V}_4 + \frac{T_6}{T}\overline{V}_6 \tag{10}$$

where T_4 and T_6 are the durations of the basic vectors V_4 and V_6 to be applied respectively. T_0 is the duration for the zero vectors (V_0 or V_7). Once the reference voltage V_{out} and the PWM period T are known, T_4, T_6 and T_0 can be determined according to the above equation [3].

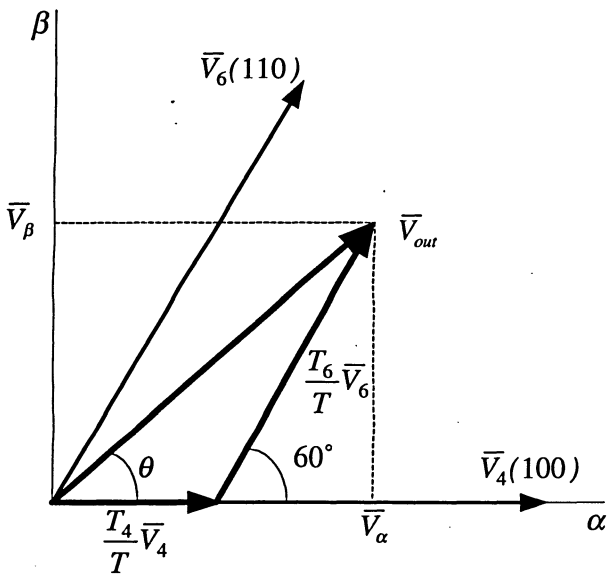
$$T_4 = \frac{T}{2V_{dc}}(3V_{S\alpha} - \sqrt{3}V_{S\beta}) \tag{11}$$

$$T_6 = \sqrt{3}\frac{T}{V_{dc}}V_{s\beta} \tag{12}$$

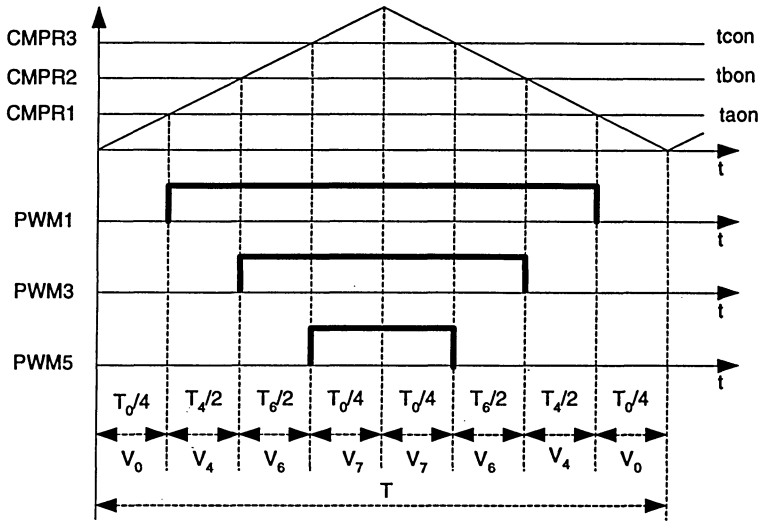
$$T_0 = T - (T_4 + T_6) \tag{13}$$

Where $V_{S\alpha}$, $V_{S\beta}$ are α-β components of V_{out}. The voltage V_{out} is an approximation of the desired output voltage based on the assumption that the change of output voltage is negligible within a PWM period T. Therefore it is critical that the PWM period is small with respect to the change of V_{out}. In practice, the approximation is very good because the calculation is performed in every PWM period (200μs).

Figure 7b shows the pattern of space vector PWM in Sector 3. The space vector PWM is implemented in the DSP TMS320LF2407 in Ryerson Power Electronic Laboratory, where t_{aon}, t_{bon} and t_{con} are the ON durations of switches *Q1*, *Q3* and *Q5* respectively. CMPR1, CMPR2 and CMPR3 are internal compare registers of DSP used to implement symmetrical PWM waveforms. Similarly, the patterns of the space vector PWM in other sectors can be formulated.



(a) Reference Voltage and Its Projections



(b) PWM Pattern in Sector 3

Figure 7: Example of Space Vector PWM Pattern

2.3 Field Oriented Control

The field oriented control (FOC) designed in this research paper provides an efficient real-time control of the torque of the induction motor, which in terms controls the motor mechanical speed. The FOC also regulates the phase currents to avoid any current surges during the motor transient operation. The following describes the FOC method.

2.3.1 The Basic FOC Scheme

Figure 8 shows the basic scheme for induction motors using the FOC method. In this control scheme, two motor input currents i_a, i_b are measured. These measurements are fed into the α-β transformation module. The outputs of this module are the two α-β currents: $i_{s\alpha}$ and $i_{s\beta}$. They are inputs to the d-q transformation module. This module computes the two currents i_{sd} and i_{sq} in the d-q rotating reference frame. Then i_{sd} and i_{sq} (they are virtually dc quantities) are compared to the flux reference i_{sdref} and the torque reference i_{sqref}. The torque reference is the output of the speed regulator in the speed feedback control module. The outputs of the two PID regulators are V_{sd} and V_{sq}. They become the inputs to the inverse d-q transformation module. The outputs of this module are $V_{s\alpha}$ and $V_{s\beta}$, which are the stator vector voltages in the α-β orthogonal reference frame. They are the inputs to the space vector PWM control module. The outputs of this module are the switching signals that operate the voltage source inverter (VSI) for driving the induction motor. Note that both d-q transformation and its inverse transformation require the value of the rotor flux

position for their computation. Therefore the rotor flux position needs to be determined prior to the transformation calculations.

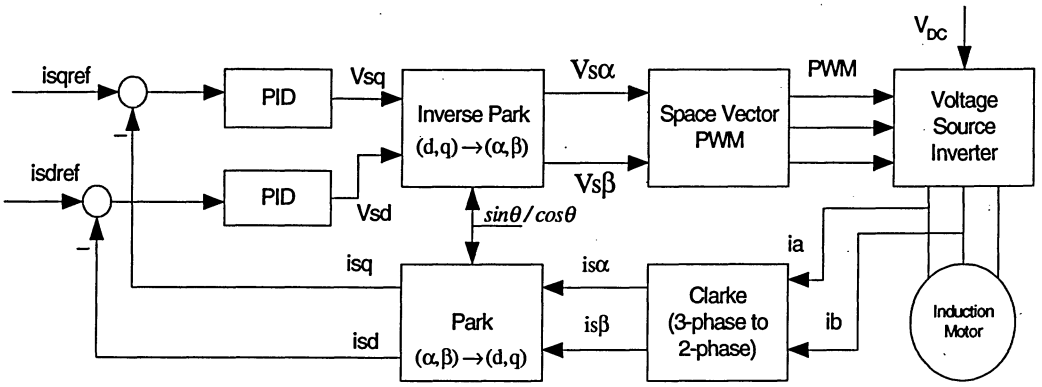


Figure 8: Field Oriented Control of Induction Motors

2.3.2 Rotor Flux Position

The information of the rotor flux position is crucial to the FOC control. Figure 9 shows the three reference frames, and the rotor flux position, the stator current space vector and the voltage space vector which rotate with respect to the d-q reference at the synchronous speed.

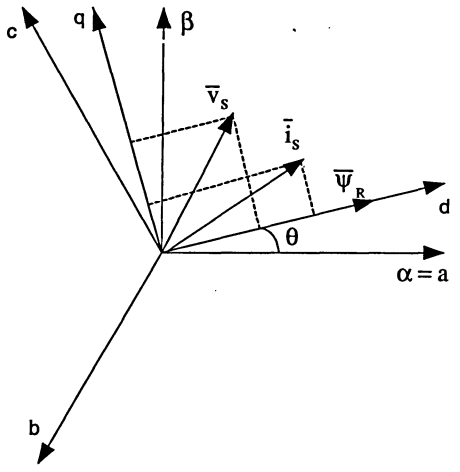


Figure 9: Relationship between Current, Voltage and Rotor Flux

In the induction motor, the speed of rotor is less than the speed of the rotor flux because of the slip due to friction and the mechanical load. For the FOC control, a specific method is needed to calculate θ. The basic method is the use of the current model that needs two equations of the motor model in d-q reference frame and a speed sensor [1]. Another method is to use a statistically optimal observer named the Kalman filter, to estimate the speed for achieving a sensorless control of induction motors. The details of this method is

presented in Chapter 3. Figure 10 shows the block diagram of the sensorless control. This research paper is focused on the development of this control method.

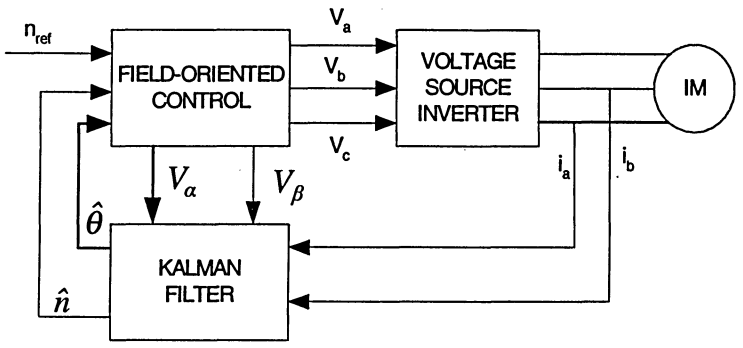


Figure 10: Sensorless Controller for Induction Motors

2.4 Adaptive PID Regulator

A motor drive based on the field-oriented control needs two inputs: the torque component reference i_{sqref} and the flux component reference i_{sdref}. The classical proportional and integral (PI) regulator is often used to regulate the motor torque and flux to the desired values. This regulator which is implemented in this research paper, is capable of reaching constant references by correctly setting both the P term (K_P) and the I term (K_I). The P term and I term respectively regulate the error sensibility and the steady state error. The regulation can be improved with the adaptive proportional-integral-derivative (PID) regulator [14].

To design a digital PID controller for the motor control, it may first consider the transfer function of an analog PID regulator:

$$D(s) = K_P + K_I \frac{1}{s} + K_D s \tag{14}$$

where K_P is the proportional gain, K_I is the integral gain, and K_D is the derivative gain. Similar to the *Laplace Transform* in continuous time domain, the integrator and differentiator can be represented by pulse transfer function in discrete domain.

$$\text{Integrator} = \frac{T(z+1)}{2(z-1)} \tag{15}$$

$$\text{Differentiator} = \frac{z-1}{Tz} \tag{16}$$

where T is the sampling period. Thus the transfer function of a digital non-adaptive PID controller is

$$D(z) = K_P + K_I \frac{T(z+1)}{2(z-1)} + K_D \frac{z-1}{Tz} = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}} \tag{17}$$

where

$$a_0 = K_P + \frac{K_I T}{2} + \frac{K_D}{T}$$

$$a_1 = -K_P + \frac{K_I T}{2} - \frac{2K_D}{T}$$

$$a_2 = \frac{K_D}{T} \tag{18}$$

Figure 11 shows the block diagram of an adaptive control system. In this figure, r is the input or set point, c is the output feedback, y is the output and D(z) is the adaptive PID controller. The adaptive control scheme consists of two parts. First, the regulator uses initial (or updated) PID parameters and feedback input samples to determine the regulation. Second, the regulator updates the PID parameters until the error signal e_2 is approaching zero.

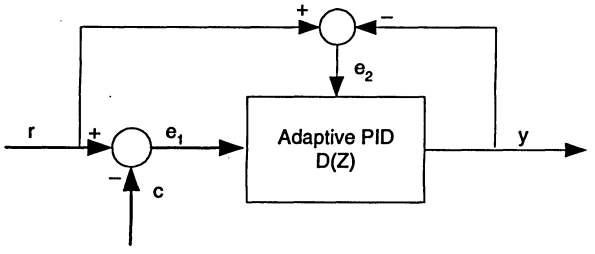


Figure 11: Adaptive PID Control

A quadratic objective function is used to minimize e_2 with respect to the regulator parameters.

$$\begin{aligned} f(a_0, a_1, a_2) &= \frac{1}{2}(e_2)^2 \\ &= \frac{1}{2}[r(z) - y(z)]^2 \\ &= \frac{1}{2}\left[r(z) - \left(\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}}\right) e_1\right]^2 \end{aligned} \tag{19}$$

The first order partial derivatives with respect to the regulator parameter a_0, a_1, a_2 are given below:

$$\frac{\partial f}{\partial a_0} = -r\left(\frac{1}{1 - z^{-1}}\right) e_1 + \left(\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1 - z^{-1}}\right)\left(\frac{1}{1 - z^{-1}}\right) e_1^{\,2}$$

$$\frac{\partial f}{\partial a_1} = -r\left(\frac{z^{-1}}{1-z^{-1}}\right)e_1 + \left(\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1-z^{-1}}\right)\left(\frac{z^{-1}}{1-z^{-1}}\right)e_1^{\,2}$$

$$\frac{\partial f}{\partial a_2} = -r\left(\frac{z^{-2}}{1-z^{-1}}\right)e_1 + \left(\frac{a_0 + a_1 z^{-1} + a_2 z^{-2}}{1-z^{-1}}\right)\left(\frac{z^{-2}}{1-z^{-1}}\right)e_1^{\,2} \qquad (20)$$

where a_0, a_1 and a_2 can be solved according to the *steepest decent method* of the gradient techniques, so that the following equations can be obtained.

$$a_n(k+1) = a_n(k) + \beta\frac{\partial f}{\partial a_n} \qquad n=0,1,2; \qquad k=0,1,2\cdots \qquad (21)$$

where β is the parameter along the search direction. When combining equations (20) and (21) and $r=e_2+y=e_2+D(z)e_1$, the following equations can be obtained.

$$\begin{aligned}
\frac{\partial f}{\partial a_0} &\approx -e_2\left(\frac{1}{1-z^{-1}}\right)e_1 \\
\frac{\partial f}{\partial a_1} &\approx -e_2\left(\frac{z^{-1}}{1-z^{-1}}\right)e_1 \\
\frac{\partial f}{\partial a_2} &\approx -e_2\left(\frac{z^{-2}}{1-z^{-1}}\right)e_1
\end{aligned} \qquad (22)$$

If ignoring the second order parts in $\partial f / \partial a_n$, the negative gradient is basically the product of error signal e_2 and error signal e_1 when using the first order approximation. Therefore, a modified gradient method is used as the search direction in updating the PID parameters. This also agrees with the general adaptive mechanism mentioned in [15]:

new parameter = old parameter +
(bounded step size) ×(function of input) × (function of error).

More specifically equation (21) can be written into:

$$a_n(k+1) = a_n + \beta e_2(k)e_1(k-n) \qquad n=0,1,2;\ k=0,1,2\cdots \qquad (23)$$

Chapter 3

Kalman Filter

for Speed Senforless Control of Induction Motors

This chapter presents the design of a control observer named the Kalman filter for efficient induction motor control which does not require any speed / position sensors. The Kalman filter is an optimal recursive algorithm, which provides the minimum variance state estimation for a time-varying linear system. It can tolerate system modeling and measurement errors, which are considered as noise processes in the state estimation. The Kalman filter processes all available measurements regardless of their precision, and provides a quick and optimal estimate of the variables of interest, as well as achieves a fast convergence. Its extension to applications for non-linear systems is called the Extended Kalman Filter (EKF). This chapter describes the Kalman filter and its application to induction motor drives.

Section 3.1 presents the basics for Kalman filter. First, the basic formulation of the Kalman filter control for the linear systems is provided. Second, the Kalman filter is extended for the nonlinear systems, in particular for the induction motor control systems. Third, an efficient recursive algorithm for the DSP implementation of the Kalman filter is provided.

Section 3.2 presents the specific formulation of induction motors for the implementation of Kalman filter in the speed sensorless field-oriented control. First, the d-q model of the induction motor is re-formulated such that the rotor flux linkage and speed can be computed efficiently. Second, the per unit formulation is provided for the implementation in the fixed-point DSPs.

3.1 Basics of Kalman Filter

Assume a linear system with the following equations.

$$\text{State Equation:} \quad \dot{x} = Ax + Bu + w \tag{24}$$

$$\text{Output Equation:} \quad y = Cx + v \tag{25}$$

where x is the state variable vector, y is the output vector, u is the input vector, and A, B, C are the system matrices. w and v are the system and the measurement noises respectively.

Assuming that w and v are stationary, white, uncorrelated and *Gauss* noises, their

expectations are zero, their covariance matrices are Q and R respectively.

$$\begin{aligned} \text{cov}(w) &= E\{ww^T\} = Q \\ \text{cov}(v) &= E\{vv^T\} = R \end{aligned} \tag{26}$$

where $E\{.\}$ denotes the expected value, that is:

$$E(w) = \int_{-\infty}^{\infty} wf(w)dw \tag{27}$$

where $f(w)$ is the probability of w.

Briefly speaking, Kalman filter has the same structure as an observer shown in Figure 12 . The system equation of the Kalman filter is as follows.

$$\dot{\hat{x}} = A\hat{x} + Bu + K(y - C\hat{x}) = (A - KC)\hat{x} + Bu + Ky \tag{28}$$

where $\hat{x}$ is the estimate of state x. Kalman gain matrix K is a solution based on the well-known *Riccati equation*, which is developed to minimize the following quadratic objective function [6].

Minimize: $$J = \sum_{i=1}^{n} E\{e_i^2\}, \quad e = x - \hat{x} \tag{29}$$

Kalman gain: $$K = PC^T R^{-1} \tag{30}$$

Riccati equation: $$AP + PA^T - PC^T R^{-1} CP + Q = 0 \tag{31}$$

The error covariance matrix P is the solution of *Riccati equation*, which is used to determine the feedback matrix K of the Kalman filter. The matrix K, which is called Kalman gain, determines how the estimate of the state vector $\hat{x}$ is modified after the output of Kalman filter is compared with the real output of the system.

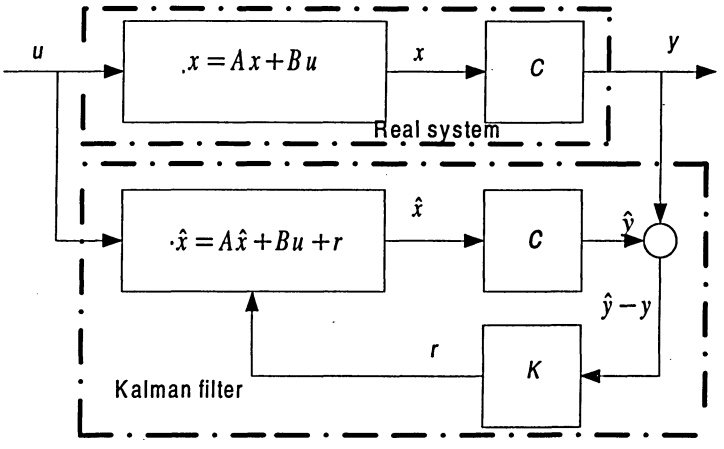


Figure 12: Principle of Kalman Filter

The extended Kalman filter (EKF) is developed to deal with the non-linearity of the induction motor. Assume g and h are the functions of the state equation and the output equation for a nonlinear system.

$$\begin{aligned} \dot{x} &= g(x,u,t) \\ y &= h(x) \end{aligned} \tag{32}$$

The discrete time model of the nonlinear system can be represented in the following equations.

$$\begin{aligned} x(k+1) &= x(k)+Tg(x(k),u(k),k) = \Phi(x(k),u(k),k) \\ y(k) &= h(x(k)) \end{aligned} \tag{33}$$

Where T is the sample period. The linearizing model of equation (33) is as follows.

$$x(k+1) = A(k)x(k) + B(k)U(k) + w(k) \tag{34}$$

$$y(k) = C(k)x(k) + v(k) \tag{35}$$

where $\quad A(k) = \left.\frac{\partial \Phi}{\partial x}\right|_{x=x_k}$, $\quad B(k) = \left.\frac{\partial \Phi}{\partial u}\right|_{u=u_k}$, $\quad C(k) = \left.\frac{\partial h}{\partial x}\right|_{x=x_k}$

The $v(k)$ and $w(k)$ are considered as noises, with their covariance matrices R and Q. If the input U_k is perfectly known, then the prediction of state x_k is as follows:

$$\hat{x}(k+1) = A(k)\hat{x}(k) + B(k)U(k) \tag{36}$$

The state prediction error is found by subtracting equations (36) from equation (34) as given below.

$$\hat{x}(k+1) - x(k+1) = A(k)[\hat{x}(k) - x(k)] - w(k) \tag{37}$$

For simplicity, equation (37) can be written as:

$$\begin{aligned} \hat{x}_{k+1} - x_{k+1} &= A_k(\hat{x}_k - x_k) - w_k \\ e_{k+1} &= A_k e_k - w_k \\ e_k &= \hat{x}_k - x_k \end{aligned} \tag{38}$$

Hence, the covariance matrix P_k of state prediction error e_k is:

$$P_{k+1} = A_k P_k A_k^{\,T} + Q \tag{39}$$

The objective function for estimating x_{k+1} is set as follows.

$$\hat{x}_{k+1} = \arg\min_x [(\hat{x}_k - x)^T P_k^{-1}(\hat{x}_k - x) + (y_k - C_k x)^T R^{-1}(y_k - C_k x)] \tag{40}$$

The minimum is found by putting the partial derivative with respect to x equal to zero. This yield:

$$\hat{x}_{k+1} = \hat{x}_k + K_k(y_k - C_k\hat{x}_k) \tag{41}$$

$$P_{k+1} = P_k - K_k C_k P_k \tag{42}$$

$$K_k = P_k C_k^T (R + C_k P_k C_k^T)^{-1} \tag{43}$$

The extended Kalman filter (EKF) can be implemented in a DSP using a recursive algorithm given below. The implementation requires the following five steps:

1) State vector prediction at instant *k+1*.

$$x_{k+1/k} = \Phi(x_{k/k}, u_k, k) \tag{44}$$

2) Prediction error covariance matrix computation.

$$P_{k+1/k} = A_k P_{k/k} A_k^T + Q \tag{45}$$

3) Kalman gain computation at instant *k+1*.

$$K_{k+1} = P_{k+1/k} C_k^T \left(C_k P_{k+1/k} C_k^T + R\right)^{-1} \tag{46}$$

4) State vector estimation at instant *k+1*.

$$x_{k+1/k+1} = x_{k+1/k} + K_{k+1}(y_{k+1} - h(x_{k+1/k})) \tag{47}$$

5) Filter error covariance matrix computation at instant *k+1*.

$$P_{k+1/k+1} = P_{k+1/k} - K_{k+1} C_k P_{k+1/k} \tag{48}$$

Equation (47) is crucial to the EKF algorithm, because it provides the state estimation. Figure 13 shows the EKF algorithm used in induction motor control.

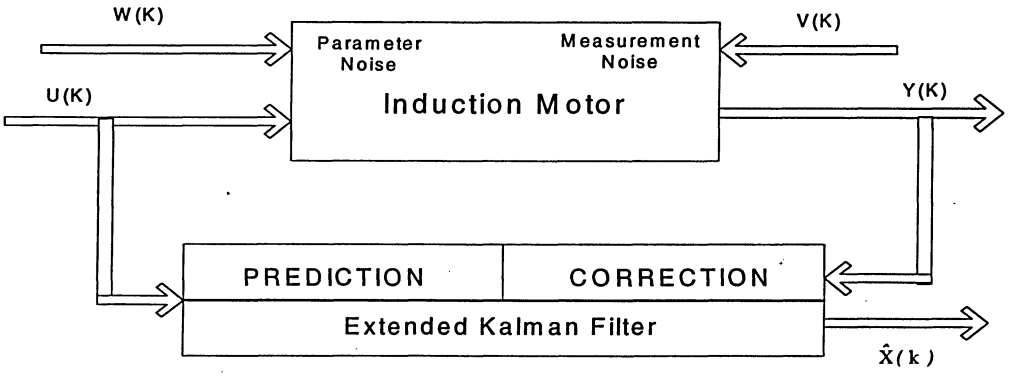


Figure 13: Extended Kalman Filter Algorithm

3.2 Induction Motor Model for Implementation of Kalman Filter

As shown in previous section, an appropriate model of induction motor is required to

implement the Kalman filter [16]. The following presents the formulation of induction motors for the implementation of the Kalman filter.

The stator and rotor flux in the stator reference frame are defined as

$$\begin{aligned}\Psi_{S\alpha} &= L_S i_{S\alpha} + L_H i_{R\alpha} \\ \Psi_{S\beta} &= L_S i_{S\beta} + L_H i_{R\beta} \\ \Psi_{R\alpha} &= L_R i_{R\alpha} + L_H i_{S\alpha} \\ \Psi_{R\beta} &= L_R i_{R\beta} + L_H i_{S\beta}\end{aligned} \tag{49}$$

The α-β stator voltages in the stator reference frame are:

$$\begin{aligned}V_{S\alpha} &= R_S i_{S\alpha} + \frac{d}{dt}\Psi_{S\alpha} \\ V_{S\beta} &= R_S i_{S\beta} + \frac{d}{dt}\Psi_{S\beta}\end{aligned} \tag{50}$$

The α-β rotor voltages in the stator reference frame are:

$$\begin{aligned}V_{R\alpha} &= R_R i_{R\alpha} + \omega\Psi_{R\beta} + \frac{d}{dt}\Psi_{R\alpha} \\ V_{R\beta} &= R_R i_{R\phi} - \omega\Psi_{R\alpha} + \frac{d}{dt}\Psi_{R\beta}\end{aligned} \tag{51}$$

For squirrel-cage induction motor, the rotor voltages are zero that is $V_{R\alpha}=0$, $V_{R\beta}=0$. Since the digital computation period for the speed control loop is very short as compared to the induction motor speed, the speed can be considered constant to have no change during one sampling period, that is $d\omega/dt=0$. If the required rotor flux $\Psi_{R\alpha}$, $\Psi_{R\beta}$, stator current $i_{s\alpha}$, $i_{s\beta}$ and speed ω are selected as state variables, the state equation and output equation of the induction motor can be written as:

1) State equation is

$$\frac{d}{dt}\begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix} = \begin{bmatrix} -\frac{K_R}{K_L} & 0 & \frac{L_H R_R}{L_R^2 K_L} & \frac{L_H \omega}{L_R K_L} & 0 \\ 0 & -\frac{K_R}{K_L} & -\frac{L_H \omega}{L_R K_L} & \frac{L_H R_R}{L_R^2 K_L} & 0 \\ \frac{L_H}{T_R} & 0 & -\frac{1}{T_R} & -\omega & 0 \\ 0 & \frac{L_H}{T_R} & \omega & -\frac{1}{T_R} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix} + \frac{1}{K_L}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} u_{S\alpha} \\ u_{S\beta} \end{bmatrix} \tag{52}$$

2) Output equation is

$$\begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{S\alpha} \\ i_{S\beta} \\ \Psi_{R\alpha} \\ \Psi_{R\beta} \\ \omega \end{bmatrix} \tag{53}$$

where $K_R=R_S+(L_H/L_R)^2R_R$, $K_L=\sigma L_S$. The *per unit model* is useful for the digital implementation as it converts various quantity ranges into one single range. This allows to choose one accurate numerical representation for the complete system. With the implementation in the fixed-point DSP TMS320LF2407, the *Q15* format is used in this research paper for its greatest precision. In per unit model, all quantities are scaled with respect to their base values.

$$\begin{aligned} i_{S\alpha} &= i^n_{S\alpha} I_b \quad \Psi_{R\alpha} = \Psi^n_{R\alpha} \Psi_b \quad u_{S\alpha} = u^n_{S\alpha} U_b \\ i_{S\beta} &= i^n_{S\beta} I_b \quad \Psi_{R\beta} = \Psi^n_{R\beta} \Psi_b \quad u_{S\beta} = u^n_{S\beta} U_b \end{aligned}, \quad \omega = \omega^n \omega_b \tag{54}$$

$$\begin{aligned} V_b &= I_b Z_b \\ \Psi_b &= V_b / \omega_b \end{aligned} \tag{55}$$

where I_b, V_b are the base values of the phase current and voltage. ω_b is the nominal speed, Ψ_b is the base flux, Z_b is the base impedance. $i^n_{s\alpha}$, $i^n_{s\beta}$, $\Psi^n_{R\alpha}$, $\Psi^n_{R\beta}$ $u^n_{s\alpha}$, $u^n_{s\beta}$, ω^n are normalized currents, fluxes, voltages and speed respectively.

By using normalized values of state variables, equations (52) and (53) can be written as the *Per Unit Model of Induction Motors* below.

1) State equation is

$$\frac{d}{dt}\begin{bmatrix} i^n_{S\alpha} \\ i^n_{S\beta} \\ \Psi^n_{R\alpha} \\ \Psi^n_{R\beta} \\ \omega^n \end{bmatrix} = \begin{bmatrix} -\frac{K_R}{K_L} & 0 & \frac{L_H R_R}{L_R^2 K_L}\frac{Z_b}{\omega_b} & \frac{L_H Z_b \omega^n}{L_R K_L} & 0 \\ 0 & -\frac{K_R}{K_L} & -\frac{L_H Z_b \omega^n}{L_R K_L} & \frac{L_H R_R}{L_R^2 K_L}\frac{Z_b}{\omega_b} & 0 \\ \frac{L_H}{T_R}\frac{\omega_b}{Z_b} & 0 & -\frac{1}{T_R} & -\omega_b \omega^n & 0 \\ 0 & \frac{L_H}{T_R}\frac{\omega_b}{Z_b} & \omega_b \omega^n & -\frac{1}{T_R} & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i^n_{S\alpha} \\ i^n_{S\beta} \\ \Psi^n_{R\alpha} \\ \Psi^n_{R\beta} \\ \omega^n \end{bmatrix} + \frac{Z_b}{K_L} \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} u^n_{S\alpha} \\ u^n_{S\beta} \end{bmatrix} \tag{56}$$

2) Output equation is

$$\begin{bmatrix} i_{S\alpha}^n \\ i_{S\beta}^n \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} i_{S\alpha}^n \\ i_{S\beta}^n \\ \Psi_{R\alpha}^n \\ \Psi_{R\beta}^n \\ \omega^n \end{bmatrix} \tag{57}$$

Using the linearizing form of the induction motor model, the matrices and vectors that are needed for the EKF recursive algorithm [equations (44)-(48)] can be calculated.

$$\Phi = \begin{bmatrix} (1-T\frac{K_R}{K_L})i_{S\alpha}^n + T\frac{L_H R_R}{L_R^2 K_L}\frac{Z_b}{\omega_b}\Psi_{R\alpha}^n + T\frac{L_H Z_b}{L_R K_L}\omega^n \Psi_{R\beta}^n + T\frac{Z_b}{K_L}u_{S\alpha}^n \\ (1-T\frac{K_R}{K_L})i_{S\beta}^n - T\frac{L_H Z_b}{L_R K_L}\omega^n \Psi_{R\alpha}^n + T\frac{L_H R_R}{L_R^2 K_L}\frac{Z_b}{\omega_b}\Psi_{R\beta}^n + T\frac{Z_b}{K_L}u_{S\beta}^n \\ T\frac{L_H}{T_R}\frac{\omega_b}{Z_b}i_{S\alpha}^n + (1-\frac{T}{T_R})\Psi_{R\alpha}^n - T\omega_b \omega^n \Psi_{R\beta}^n \\ T\frac{L_H}{T_R}\frac{\omega_b}{Z_b}i_{S\beta}^n + T\omega_b \omega^n \Psi_{R\alpha}^n + (1-\frac{T}{T_R})\Psi_{R\beta}^n \\ \omega^n \end{bmatrix} \tag{58}$$

$$h = Cx = \begin{bmatrix} i_{S\alpha}^n \\ i_{S\beta}^n \end{bmatrix} \tag{59}$$

$$A = \frac{\partial \Phi}{\partial x} = \begin{bmatrix} 1-T\frac{K_R}{K_L} & 0 & T\frac{L_H R_R}{L_R^2 K_L}\frac{Z_b}{\omega_b} & T\frac{L_H Z_b}{L_R K_L}\omega^n & T\frac{L_H Z_b}{L_R K_L}\Psi_{R\beta}^n \\ 0 & 1-T\frac{K_R}{K_L} & -T\frac{L_H Z_b}{L_R K_L}\omega^n & T\frac{L_H R_R}{L_R^2 K_L}\frac{Z_b}{\omega_b} & -T\frac{L_H Z_b}{L_R K_L}\Psi_{R\alpha}^n \\ T\frac{L_H}{T_R}\frac{\omega_b}{Z_b} & 0 & 1-\frac{T}{T_R} & -T\omega_b \omega^n & -T\omega_b \Psi_{R\beta}^n \\ 0 & T\frac{L_H}{T_R}\frac{\omega_b}{Z_b} & T\omega_b \omega^n & 1-\frac{T}{T_R} & T\omega_b \Psi_{R\alpha}^n \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{60}$$

$$C = \frac{\partial h}{\partial x} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{61}$$

First, the matrices and vectors for the induction motor model are calculated. Second, the recursive algorithm for the extended Kalman filter is executed. Third, the rotor flux and the rotor speed are computed.

The recursive Kalman filter algorithm can accurately determine the speed of the induction motor and the rotor flux position or angle. This allows an accurate control of the motor speed without the need of a speed or position sensor. However, the implementation of the Kalman filter is a fairly involved process. The model of the induction motor must be calculated in real time, which involves many matrix multiplications. The DSP with high computation capability is most suitable for this type of control process.

Appendix B lists all the internal parameters of the induction motor used in this research paper, which are needed for the calculations of equations (44) to (48).

Chapter 4

Digital Signal Processor Implementation of Speed Sensorless Field Oriented Control

This chapter presents the digital signal processor (DSP) implementation of the field oriented control (FOC) of induction motors without any speed or position sensors. The hardware and the software implementation of the speed sensorless FOC system is detailed in this chapter. The following outlines the sections in this chapter.

Section 4.1 presents an overview of the DSP FOC operation. This section describes the key operations which include determination of the rotor speed and flux angle using Kalman filter, d-q transformation with respect to the rotor flux, speed and torque regulation, and generation of PWM control signals.

Section 4.2 presents the hardware implementation of the FOC system for induction motors. This section describes the power electronic inverter circuit, the DSP control circuit, and their interface used for the implementation.

Section 4.3 presents the software implementation of the FOC operation. This section describes the software modules used to implement the sensorless speed control of induction motors.

Section 4.4 presents the debugging of hardware and software. This section describes the six phases used in debugging the FOC software and hardware.

4.1 Overview of DSP FOC Operations

The fixed-point DSP TMS320LF2407 is the core of the control system designed in this research paper. The following provides an overview of the DSP FOC operations.

1) The DSP operates its analog-to-digital converter (ADC) to collect the instantaneous induction motor input currents measured by a current transducer at the motor terminal.

2) The DSP executes the recursive algorithm of the extended Kalman filter, using the measured motor currents, to compute the α and β currents, the rotor speed and flux angle.

3) The DSP carries out a specific d-q transformation with the d-coordinate chosen to be in line with the rotor flux, using the α-β currents and the flux angle, to compute the d-coordinate and q-coordinate currents.

4) The DSP executes three feedback regulators for the motor speed, the rotor torque, and the rotor flux, to determine the d-coordinate and the q-coordinate stator reference voltages.

5) The DSP carries out an inverse d-q transformation, using the d-q reference voltages, to compute the stator α and β reference voltages.

6) The DSP executes the space vector pulse-width-modulation (PWM) module, using the α-β reference voltages, to compute the PWM control signals.

7) The DSP finally outputs the PWM control signals to the gating circuits of the power electronic inverter that drives the induction motor.

4.2 Hardware Implementation of Field Oriented Control

Figure 14 shows the block diagram of the hardware required to implement a sensorless speed control system for an induction motor. The DSP control system designed in this research paper consists of the following major hardware components:

1) DSP controller: Texas Instruments, eZdspTMS320LF2407 development kit.
2) Current sensors: Two LEM, 10A current transducers.
3) Power inverter: Semikron-SkiiP2, IGBT module.
4) Induction motor: Teco-Westinghouse, 3-Phase, 60Hz induction motor.

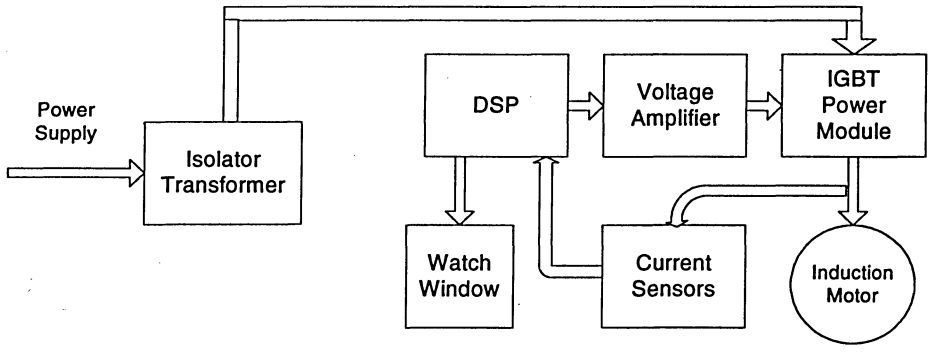


Figure 14. Experiment Setup for Sensorless Control System

The power hardware used to implement and test the control scheme developed in this research paper is based on the six-power IGBT module (SkiiP2 from *SEMIKRON*). This module is controlled by a DSP (TMS320LF2407 from Texas Instruments) via a voltage-level shifter (MC14504). This shifter amplifies the PWM output voltages from 3.3V to 15V for gating the IGBT switches. The current sensing is performed via two current-to-voltage transducers (*LEM* type) supplied with ±12V. Their maximum input current is ±10A, which is converted into a 3.3V output voltage to match the requirement of inputs to DSP. The induction motor is a 3-Phase, 60Hz motor (EPACT from *TECO-WESTINGHOUSE)*, of which the rated power is 5 horsepower.

DSP is the key control element in this research paper design of the FOC of induction motors. The settings of the DSP TMS320LF2407 are given as follows.

- Development/Emulation: Code Composer 4.1 supports real-time debugging.
- CPU Clock: 30MHz.
- PWM frequency: 5 kHz.
- PWM mode: Symmetrical with dead band 1.8μs.
- Interrupts: 2 (T1 Underflow - System time base).
- System time base/PWM period: 200μS.
- Peripheral Usage: Timer 1, PWM1-6, ADC (2 channels).
- Current loop sampling frequency: 5 kHz.

Two ADC input pins (Pin 4, 6 in port P1) are used to sense the two stator currents i_a, i_b. PWM 1-PWM6 output pins (Pin 9-14 in port P2) are used to drive the IGBT power inverter.

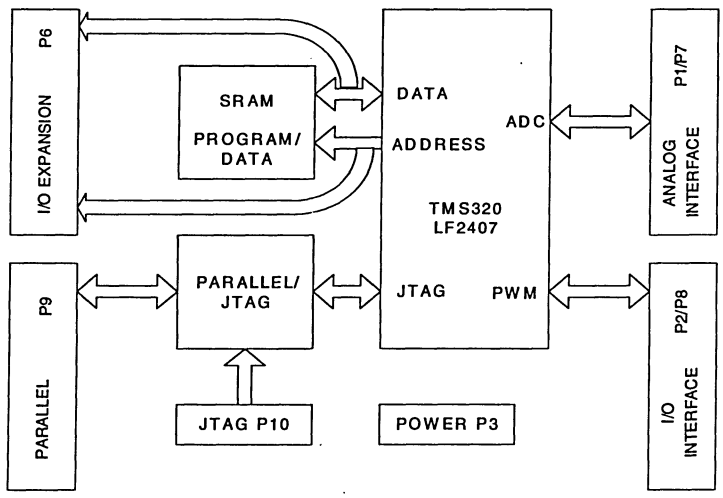


Figure 15: Block Diagram of eZdsp^{TM} LF2407

The eZdsp LF2407 platform is used to develop and run software for the DSP TMS320LF2407. With 64K words of onboard program/data RAM, this platform allows a full speed verification of software code. To simplify code development and shorten

debugging time, a C2000 Tools Code Composer driver is provided, which could run the software in real time. Figure 15 shows the block diagram of this platform.

4.3 DSP Software Implementation of Field Oriented Control

DSP control software developed in this research paper is based on two modules: the initialization and the run module. The initialization module is performed only once at the beginning of the software execution. The run module is based on a user interface loop interrupted by the PWM underflow. When this interrupt flag is set, the corresponding *Interrupt Service Routine* (ISR) is acknowledged and served. The complete extended Kalman filter (EKF) and the FOC algorithms are computed within a PWM interrupt at the same period as the chopping frequency 5 KHz. The overview of the software is given in Figure 16.

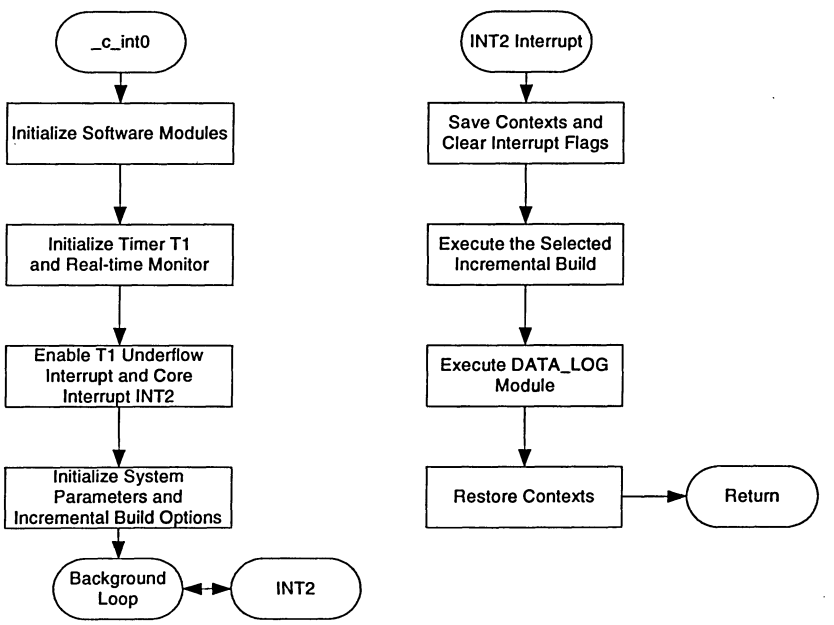


Figure 16: Software Flowchart

The DSP Controller Full Compare Units are designed to generate symmetrical complementary PWM signals to the power inverter at a frequency of 5KHz. TIMER1 is used as the time base with a sampling period T= 200 μs.

The benefits of structured modulator software are well known. This is especially true for large complex systems, such as the FOC motor control, with many sub-blocks. It reduces the developing time, and could be reused in the future project. Therefore, a typical

incremental build process is used in this research paper.

```
*****************************************************************
Phase commissioning options
*****************************************************************
phase_commissioning1   .set    1    ; hardware checkup
phase_commissioning2   .set    0    ; open loop start up
phase_commissioning3   .set    0    ; current measurement chain checking
phase_commissioning4   .set    0    ; current regulation implementation
phase_commissioning5   .set    0    ; Kalman filter estimation
phase_commissioning6   .set    0    ; closed current and speed loop
```

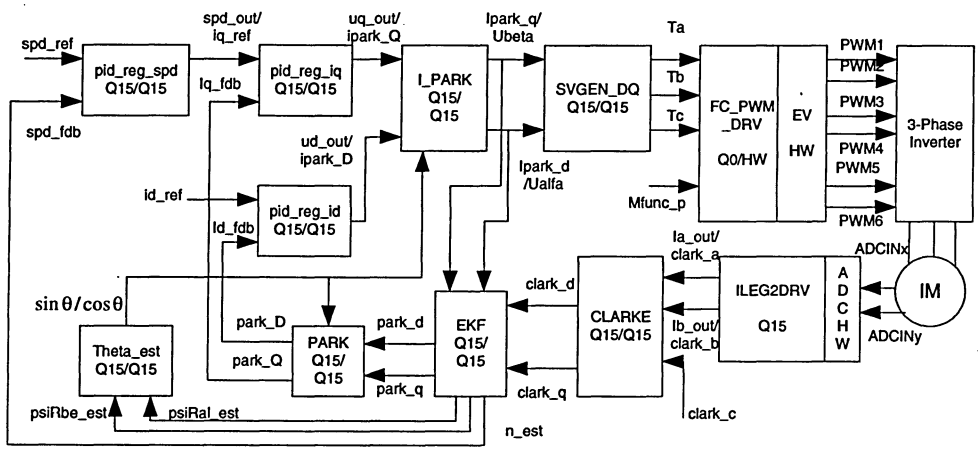


Figure 17: EKF Software Structure for an Induction Motor

The Figure 17 shows the block diagram of the software developed in this research paper. The system implemented here uses the following assembly software modules, which meet the requirements of reusability, compatibility, predictability and expandability. Appendix D gives all of the software modules used in this research paper.

CLARKE: Clarke transformation module converts the three-phase stator currents into two-phase quadrature currents.

EKF: Extended Kalman filter computes the rotor flux and speed.

DATA_LOG: Store the real-time values of two selected variables for graphical display.

I_PARK: Inverse Park transformation converts rotating reference frame into stationary frame.

ILEG2DRV-ADC: Convert two current inputs i_a, i_b into digital representations.

PARK: Park transformation converts two-phase α-β stationary frame into d-q rotating reference frame.

PID_REG_ID/IQ: Control module for regulatinge currents i_d, i_q.

PID_REG_SPD: Control module for speed control loop.

FC_PWM_DRV: Generate PWM outputs.

THETA_EST: Calculate the sine value and cosine value of rotor flux angle ($sin\ \theta / cos\ \theta$) for Park transformation and inverse Park transformation.

RAMP_GEN: Generate 60 Hz saw-tooth waveform to emulate the rotor flux angle in debugging.

SINTB360: Look-up table for sine value from 0° to 360°.

SQRT_TAB: Square-root table from 0 to 1.22.

SVGEN_DQ: Use space vector PWM technique to calculate duty ratios for PWM1-PWM6.

SYS_INIT: Initialize the software.

4.3 Debugging of Software and Hardware

After the hardware work and software programming have been completed, the software needs to be carefully debugged. The Incremental Build Process plays a key role in debugging, which decomposes the whole control task into several small tasks. There are totally six phases in debugging of the control system:

Phase 1: Confirm operation of hardware connection and the code framework.
Phase 2: Check the forward control path and PWM driver.
Phase 3: Use the 60 Hz open-loop start to check the current sensing and feedback path.
Phase 4: Check the correct operation of the current regulators.
Phase 5: Examine the correct operation of extended Kalman filter estimation and insert it in the control system.
Phase 6: Implement close loop control of current and speed.

For simplicity, Phase 2 is explained as an example. The other phase debugging procedure can be found in the software of main program foc2.asm (Appendix D).

a) Goal of the Phase 2

The objective of this phase is to check the PWM driver and PWM outputs with a simple RC filter. Following this verification, the induction motor will be connected to the power inverter board and started to run smoothly

b) Building blocks interconnection

Figure 18 shows all software modules to be used in Phase 2. The output of the inverse Park module "I_PARK" is used as the stator voltage reference. The amplitude and phase of this voltage vector are translated into commutation durations by the SVGEN_DQ module, which uses the space vector PWM technique. The FC_PWM_DRV module uses the timing information to generate the PWM outputs to control inverter for driving the induction motor. The DATA_LOG module provides a dual memory buffer to display two graphical waveforms in real time, which could be used to show any interested variable such as *Ta*, *Tb*, *Tc* or PWM outputs. The RAMP_GEN module is used to generate a 60-Hz saw-tooth waveform to emulate the rotor flux angle in debugging.

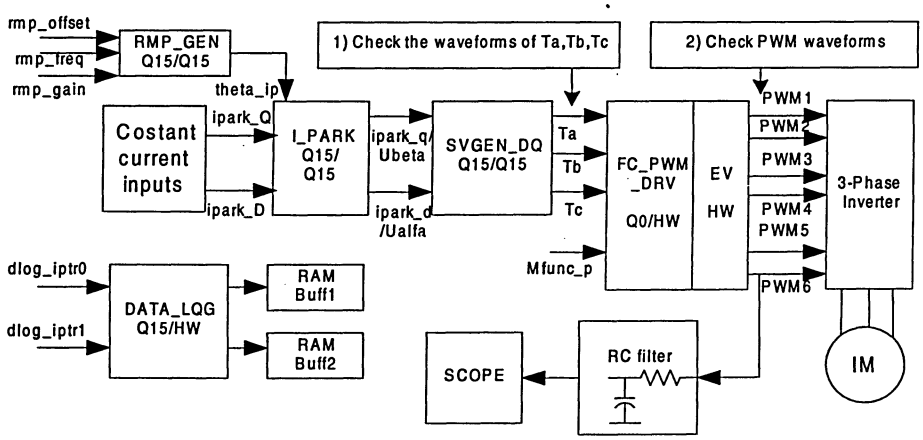


Figure 18: Overview of Phase 2 Software Flow

c) Hardware and software debugging in Phase 2

1. Select the build option *phase_commissioning2 .set 1* (the others stay to zero) in the main program foc2.asm, then build the project and load the program *foc2.out* and the corresponding workspace (see Figure 19).
2. Connect a RC filter to one PWM output.
3. Verify that the filtered waveform.
4. The values of resistor and capacitor in the RC low-pass filter can be changed to have different cutoff frequencies, which must be well below the PWM carrier frequency.
5. Connect the PWM outputs to the power inverter.
6. Increase the input voltage gradually until the motor starts running.
7. Change the value of *rmp_freq* (one of inputs in RAMP_GEN module) in a real-time watch window to vary the motor speed.

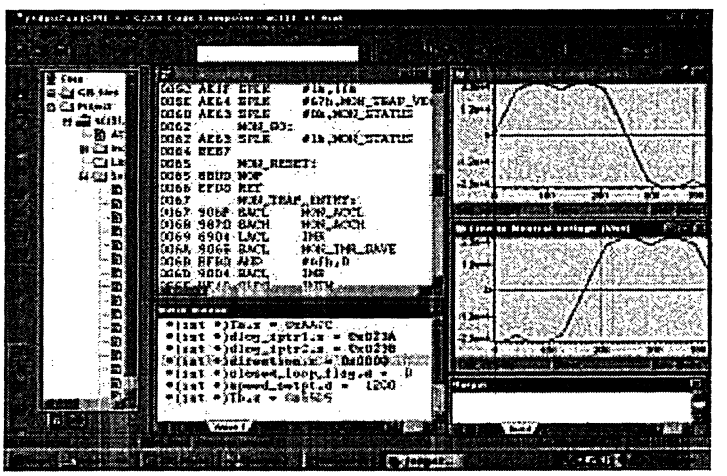


Figure 19: Overview of Phase 2 Workspace

It should be pointed out that the real time watch window is very helpful in the debugging period, since it can be used to watch each internal variable defined in the software and can be updated continuously.

Chapter 5

Experimental Verification of Field Oriented Motor Control using Kalman Filter

This chapter presents the experimental verification of the field oriented control of induction motor using Kalman filter. Typical measurements are given to demonstrate the efficiency of the novel control designed in this research paper.

The following outlines the sections in this chapter.

Section 5.1 presents all the waveforms of the internal states in the close-loop speed control to demonstrate the efficiency of the control process designed in this research paper.

Section 5.2 demonstrates the dynamic response of the speed sensorless control scheme, and results of the speed-step response are given.

Section 5.3 shows the quality of the speed sensorless control, and results of the relative speed errors are given.

5.1 Internal States of Closed-Loop Control

This section presents all the waveforms of the internal states in the close-loop speed control to demonstrate the efficiency of the control process designed in this research paper. The control scheme was shown in Figure 17 (Chapter 4). In the following demonstration, the reference speed *spd_ref* was set to 1758 rpm

1) Phase currents (i_a, i_b) are measured by current transducers, then the outputs of the transducers are converted into digital signals (*Ia_out*, *Ib_out*) by two analog-to-digital (ADC) channels in the DSP analog interface. Figure 20 shows the phase currents.

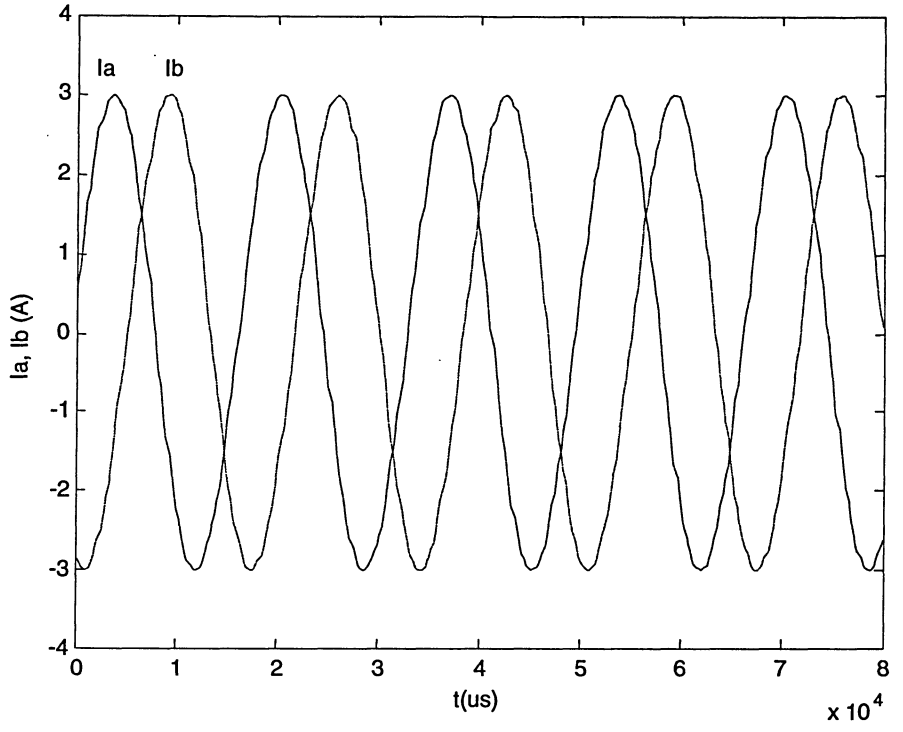


Figure 20: i_a, i_b Currents from ADC

2) The digital signals of the two phase currents are transformed into α-β currents (*clark_d*, *clark_q*) using Clarke transformation. Figure 21 shows the α-β currents.

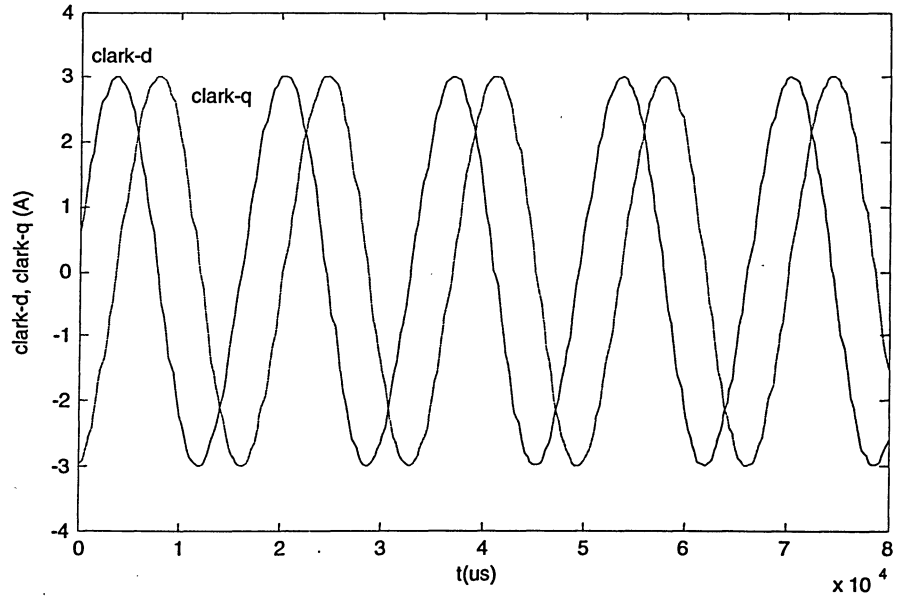


Figure 21: Currents from Clarke Transformation

3) Kalman filter module has four inputs (two currents *clark_d*, *clark_q* and two voltages *Ubeta,Ualfa*) and five estimate outputs (two currents *iSal_est*, *iSbe_est*, two rotor fluxes *psiRal_est*, *psiRbe_est* and one speed estimation *n_est*). Figure 22 and Figure 23 show the estimated outputs from the Kalman filter module.

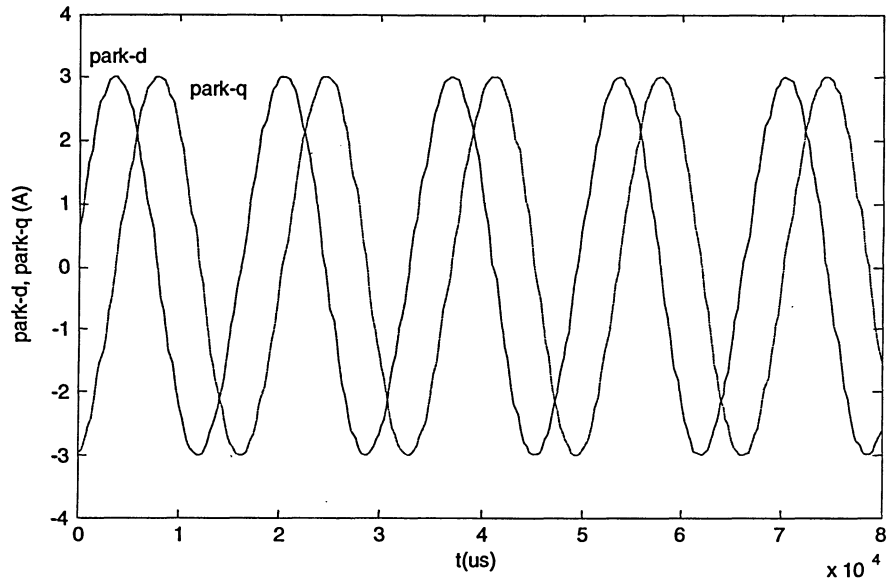


Figure 22: Current Estimated by Kalman Filter

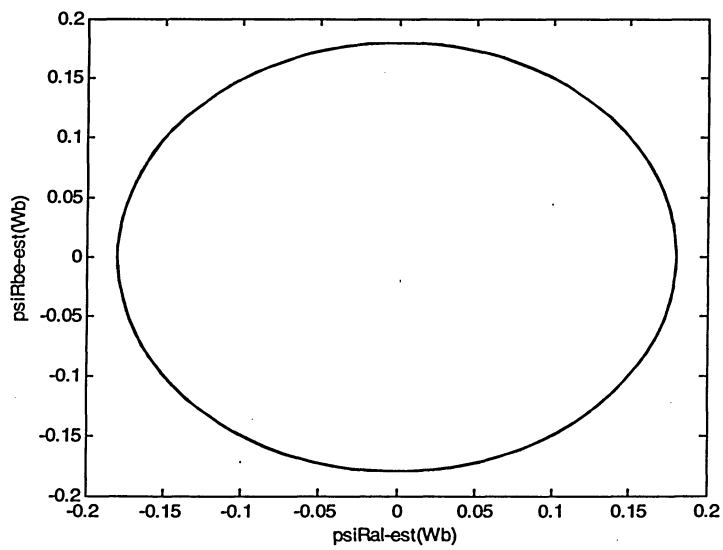


Figure 23: Flux Circle Estimated by Kalman Filter

4) α-β currents (*park_d*, *park_q*) computed by Kalman filter module are transformed into two quadrature currents (*park_D*, *park_Q*) with reference to the rotor flux using extended Park transformation. Figure 24 shows the quadrature currents.

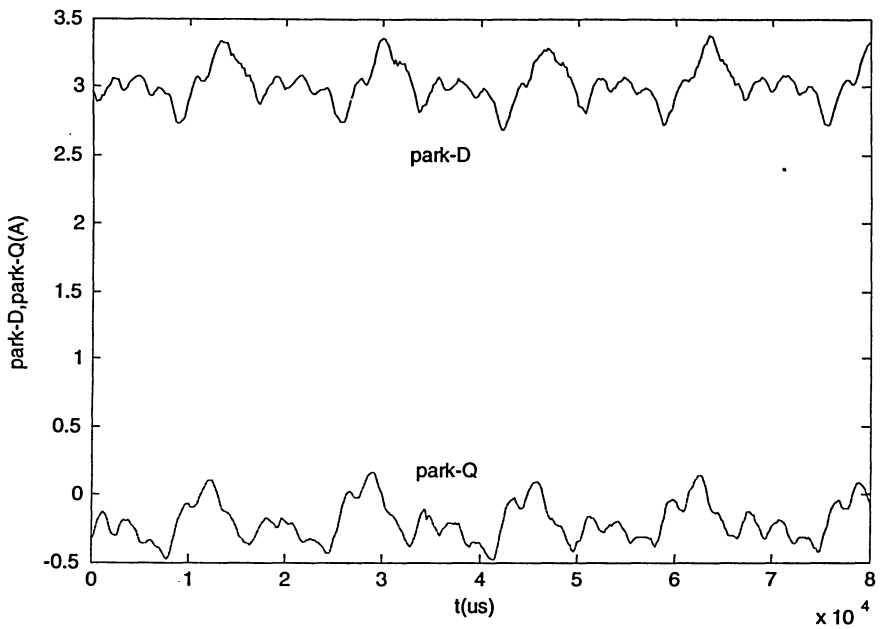


Figure 24: Currents from Park Transformation

5) Speed regulator has two inputs. One is reference speed (*spd_ref*), and the other is speed feedback (*spd_fdb*) estimated by Kalman filter. The output of the regulator is used as the reference i_q current (*iq_ref*). Figure 25 shows the speed control.

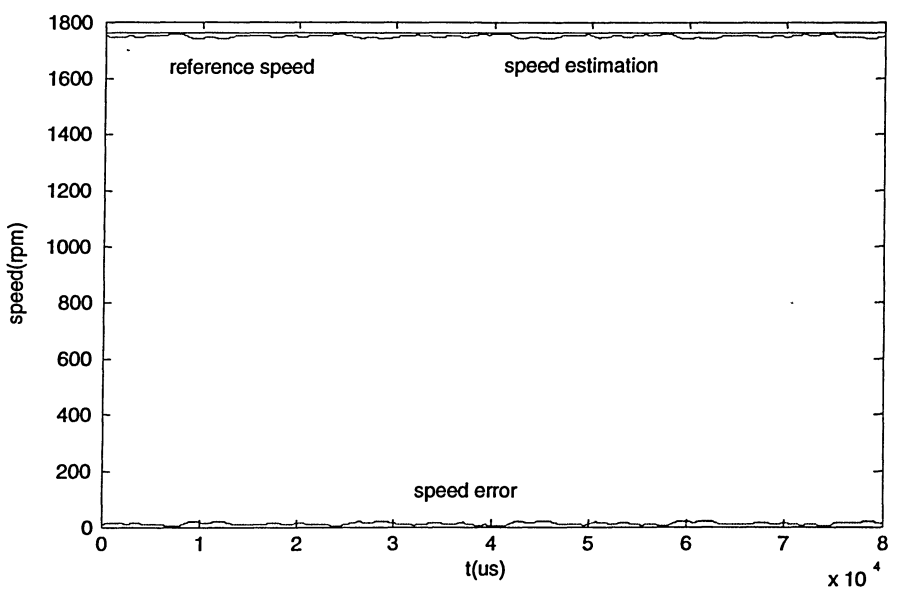


Figure 25: Speed Control

6) There are two regulators for i_d and i_q. The i_d regulator has two inputs: one is reference i_d (*id_ref*), which is dependent on the flux requirement; the other is feedback i_d (*id_fdb*), which is obtained from the Park transformation. Similarly, the i_q regulator has two inputs: one is reference i_q (*iq_ref*), which is dependent on the speed requirement; the other is feedback i_q (*iq_fdb*), which is obtained from the Park transformation. Figure 26 shows the torque control and Figure 27 shows the flux control. Figure 28 shows the outputs from the two regulators.

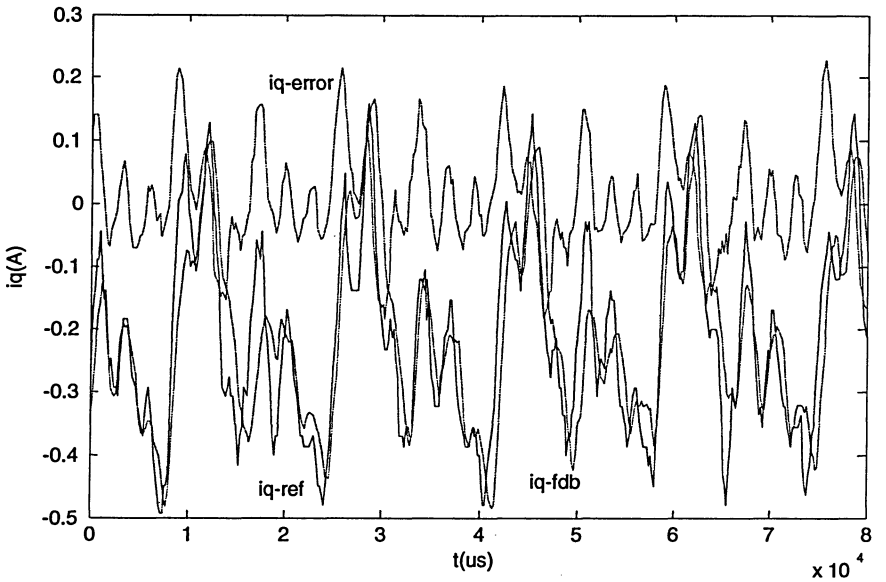


Figure 26: i_q (torque component) Control

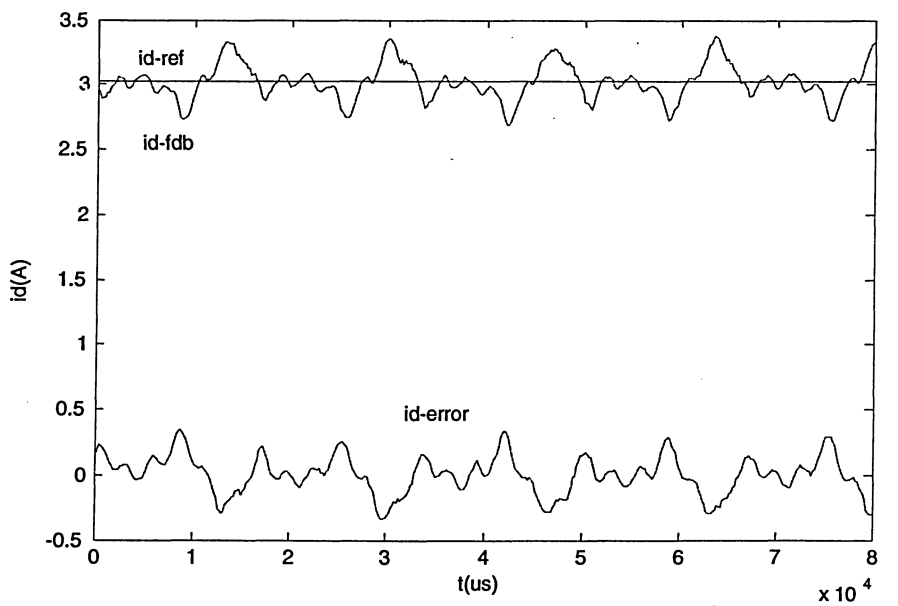


Figure 27: i_d (flux component) Control

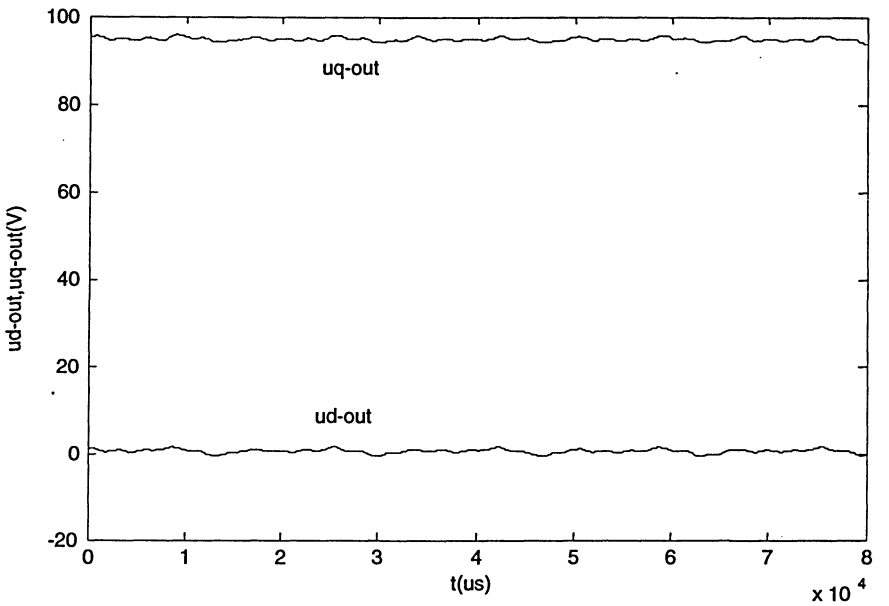


Figure 28: Voltage Outputs from i_d, i_q Regulators

7) Inverse Park transformation is used to convert the outputs of the two current regulators (*ud_out*, *uq_out*) into α-β voltages (*Ualfa*, *Ubeta*). Figure 29 shows the α-β voltages.

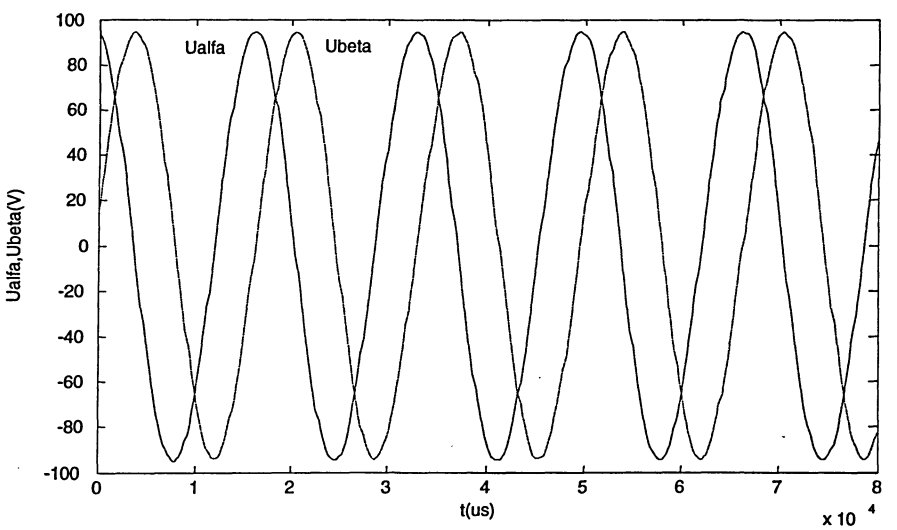


Figure 29: Ualfa, Ubeta from Inverse Park Transformation

8) Flux angle estimation (*theta_est*) is used to calculate the sine and cosine values of flux angle, which are crucial to Park and inverse-Park transformations, based on the estimated flux inputs from Kalman filter.

9) Given the required α-β voltages (*Ualfa*, *Ubeta*), space vector PWM module is used to generate corresponding ON durations: *Ta*, *Tb* and *Tc* in a sample period *T*. Figure 30

shows the durations.

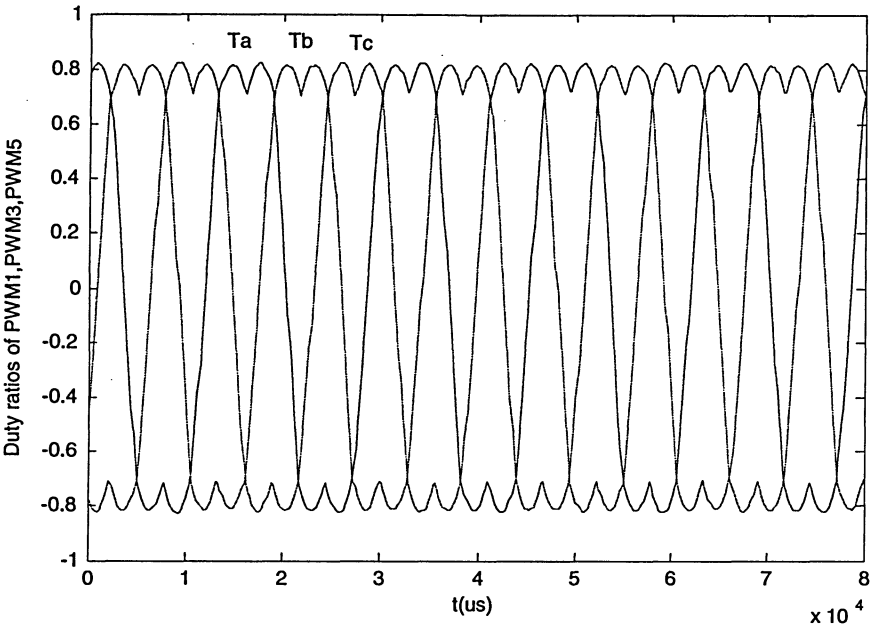


Figure 30: Duty Ratios of PWM1, PWM3 and PWM5 in Carrier Period

10) Three full compare units in the DSP are used to produce the PWM1-PWM6 based on the *Ta*, *Tb*, *Tc* and sample period *T*.

11) The PWMs from the DSP are fed to the voltage source inverter (VSI). The outputs of the VSI are three-phase voltages used to control the induction motor. Figure 31 shows the PWM outputs from the IGBT module and Figure 32 shows the line-to-line voltages fed to the induction motor.

From above figures, the following results can be obtained.

1) i_a leads i_b 120 degrees.
2) After Clark transformation, *clark_d* leads *clark_q* by 90 degrees.
3) Currents (*park_d*, *park_q*), estimated by Kalman filter, are almost the same as the *clark_d* and *clark_q*.
4) After Park transformation, i_d, i_q currents (*Park_D*, *Park_Q*) are close to dc currents.
5) The performances of the speed control, i_d regulation and i_q regulation are very good.
6) *Ualfa* leads *Ubeta* by 90 degrees.
7) Flux is constant in the control process.
8) *Ta* leads *Tb* 120 degrees, *Tb* leads *Tc* 120 degrees.
9) Phase voltages are 120 degrees shift from each other.

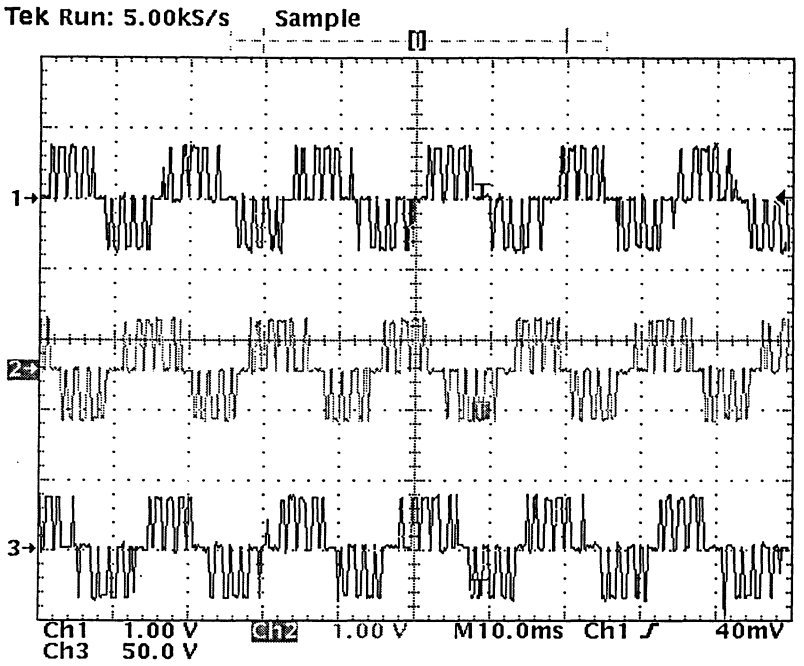


Figure 31: PWM Outputs of IGBT Power Module

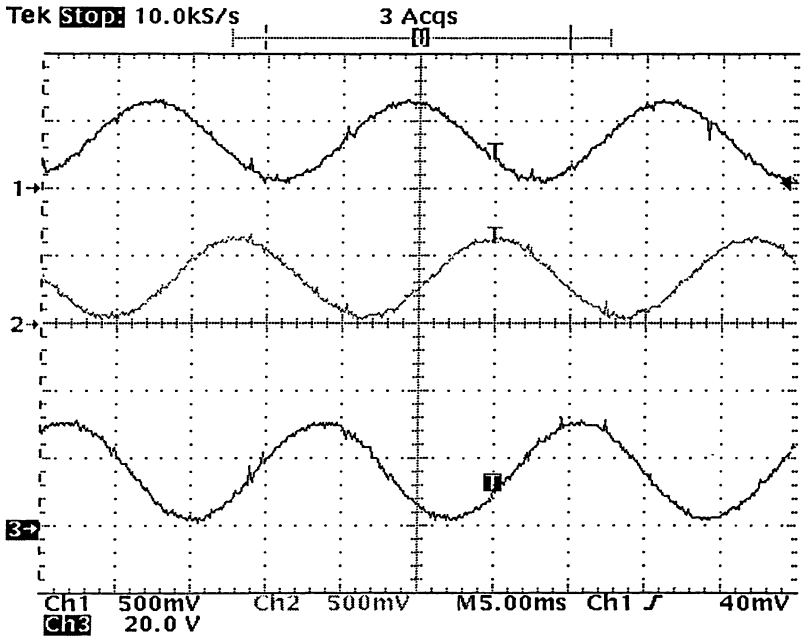
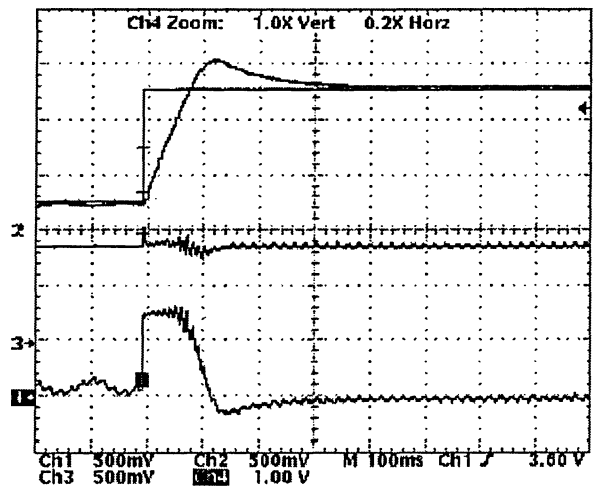


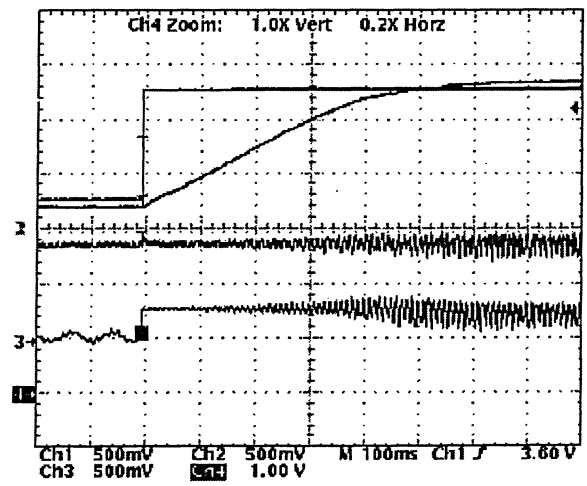
Figure 32: Line to Line Voltages of the Induction Motor

5.2 Dynamic Response

This section demonstrates the dynamic response of the speed sensorless control scheme. Figure 33 shows speed-step response from 500rpm to 1500rpm. The scope Channel #1 is the speed reference, Channel #2 is the measured speed, Channel #3 is the flux component i_d, and Channel #4 is the torque component i_q.



(a) without load



(b) with load

Figure 33: Speed Step Response (from 500rpm to 1500rpm)

The Figure 33a shows the speed control without load. Figure 33b shows the speed control with load (another EPACT motor is used as a load). The response time varies between

500ms and 800ms. Figure 34 shows the stator currents starting from 0 to 1000rpm.

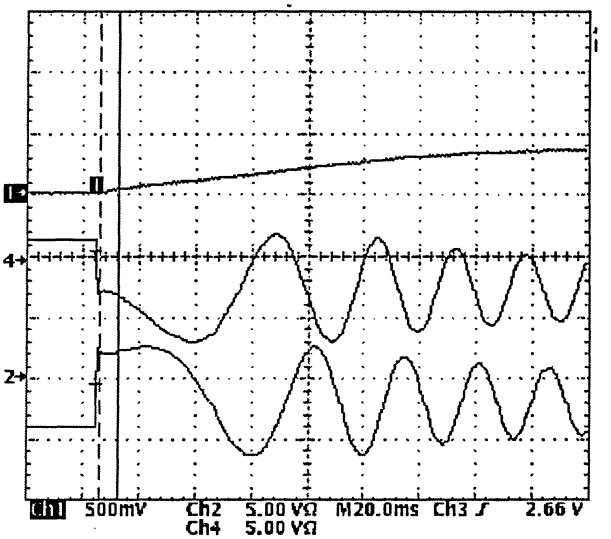


Figure 34: Speed Transient from 0 to 1000rpm

Ch1: n: mechanical speed; Ch2: Ib stator phase current; Ch4: Ia stator phase current

The following scope picture shows a speed reversion test from -1000rpm to 1000rpm with load.

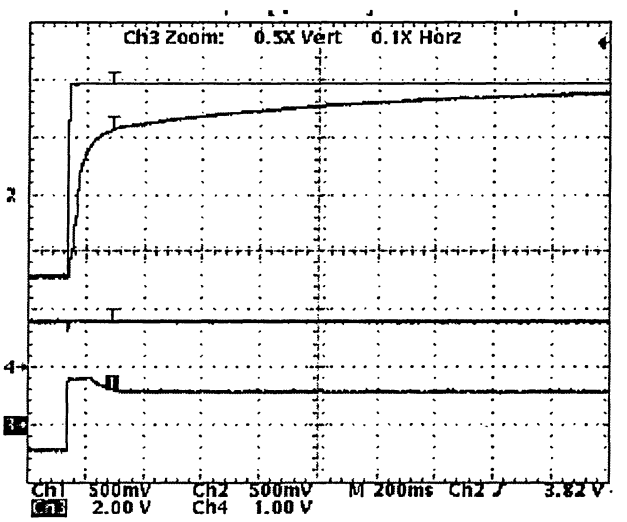


Figure 35: Speed Reversion (from -1000rpm to 1000rpm)

5.3 Speed Control Accuracy

For testing the quality of the speed sensorless control performance, the relative error in speed is defined as follows:

$$e(\%) = \frac{n - n_{ref}}{n_{ref}} 100\% \tag{35}$$

Table 1 shows the relative speed error at different speeds. The measurements run from 500 rpm up to 1800 rpm (the nominal speed of the induction motor used in the test).

TABLE 1

Steady State Performance of Speed Sensorless Control

Reference speed (rpm)	Measured speed (rpm)	Estimated speed (rpm)	Speed control error (100%)
549.3	550.5	550.8	0.22%
824.0	818.9	819.6	-0.62%
1098.6	1095.0	1096.5	-0.33%
1373.3	1376.0	1377.6	0.20%
1648.0	1650.0	1651.5	0.12%
1702.9	1702.0	1702.5	-0.05%
1757.9	1758.0	1759.5	0.005%
1785.3.	1789.0	1790.7	0.21%

Chapter 6

Conclusions

This research paper has presented a novel method for the speed control of induction motors without using a speed sensor. The rotor speed is accurately computed using the Kalman filter software module designed in this research paper. The computation, which has been implemented in the DSP TMS320LF2407, simply uses the two current measurements at the motor terminal. This replaces a speed sensor and eliminates the difficulty of the sensor installation in many applications.

This research paper has presented an advanced field oriented control of induction motors based on a specific d-q coordinate model, with the d-coordinate chosen to be in line with the rotor flux and the q-coordinate chosen to be 90° lagging. The position of the rotor flux is accurately computed using the Kalman filter module. This eliminates the position sensors required to monitor the flux. This research paper has shown that as a result of the specific d-q transformation, the motor torque is proportional to the product of the rotor flux and the q-coordinate stator current. This significantly simplifies the induction motor control, such that the rotor flux is controlled simply by regulating the flux-related d-coordinate stator current and the motor torque is controlled just by regulating the q-coordinate stator current.

This research paper has presented the Kalman filter which is an optimal control observer and its recursive algorithm and formulations which are specifically designed for the induction motor control. The Kalman filter provides the minimum variance state estimation and tolerates system modeling and measurement errors, which are considered as noise processes in the state estimation. The filter processes all available measurements regardless of their precision, and provides a quick and optimal estimate of the variables of interest, as well as achieves a fast convergence.

This research paper has presented the implementation of the field oriented control of induction motors using Kalman filter. The hardware requirements and all software modules are detailed. The experimental verification of the control method designed in this research paper has been provided. Typical measurements are given to demonstrate the efficiency of the novel control designed in this research paper.

Key Contributions of this Research Paper

The following lists the key contributions in this research paper.

1. Full investigation of Kalman filter for the control of induction motors.

This research paper has provided a full investigation of the practical application of Kalman filter for the control of induction motors. There has been very limited information about the use of Kalman filter for induction motor drives, possibly because of the difficulty of understanding the theory of the Kalman filter. This research paper research has carried out a detailed study of the control theory of the Kalman filter and its suitability for the control of induction motors. This research has provided a concrete finding that Kalman filter is well suitable for induction motor control.

2. Detailed formulations of the specific field oriented control with Kalman filter for real-time control of induction motors.

   This research paper has developed all formulations required for the specific field oriented control of induction motors using Kalman filter. In particular, a recursive algorithm of the Kalman filter is given in the research paper. The algorithm and its corresponding formulations has provided an accurate computation of the rotor speed, the flux linkage, and the flux angle which are the essential components for the specific field oriented control of induction motors.

3. DSP implementation of the specific field oriented induction motor control.

   This research paper has developed the software required for the implementation of the specific field oriented motor control in DSP TMS320LF2407. The software is developed in modules which can be easily modified for different induction motor drive applications.

4. Real-time verification of the specific field oriented control using Kalman field.

   This research paper has built a prototype of the induction motor drive to verify the specific field oriented control using Kalman filter. Typical measurements are given to demonstrate the efficiency of the novel control designed in this research paper.

Future Work in Field Oriented Control using Kalman Filter

This research paper has provided a concrete proof that Kalman filter can be used to accurately compute the motor speed and flux which are crucial for the specific field oriented control of induction motors. The efficiency of the novel control designed in this research paper has been demonstrated. Key future researches in the area of the induction motor control are proposed in the following.

1. Tune the Q and R matrices in the Kalman filter to get better performance.

Q and R are the covariance matrices of the state and measurement noises in the Kalman filter formulation. To date, there has been no exceptional good method to determine the matrices of Q and R.

2. Find lower order of the state equation for the Kalman filter computation.

   If a lower order state equation, instead of the 5-order equation presented in this research paper, can be found and can represent accurately the rotor speed and flux, the computation can be significantly reduced. The reduction of computation time is very important for the real-time control of induction motors.

3. Compare the motor drive with or without a speed sensor.

   Usually the sensorless control has good performance in the normal motor speed. However, the performance may deteriorate in very low motor speed. In fact, low speed sensorless control for induction motors is a hot topic in recent research on sensorless control.

Bibliography

[1] R. Di Gabriele, F. Parasiliti, M. Tursini, "Digital Field Oriented Control for induction motors: implementation and experimental results", Universities Power Engineering Conference (UPEC'97)

[2] Texas Instruments, "Implementation of a Speed Field Orientated Control of Three phase AC Induction Motor using TMS320F240", Literature number: BPRA076, January 1998

[3] Texas Instruments, "Field Orientated Control of Three phase AC-motors", Literature number: BPRA073, December 1997

[4] R. Di Gabriele, F. Parasiliti, M. Tursini, "Digital Field Oriented Control for induction motors: implementation and experimental results", Universities Power Engineering Conference (UPEC'97)

[5] L. Loron, G. H. L. Peterson, "Application of an Extended Kalman Filter to the estimation of the parameters of a solid rotor asynchronous drive with two degrees of freedom", EPE 1991, Firenze

[6] Texas Instruments, "Sensorless Control with Kalman Filter on TMS320 Fixed-Point DSP", Literature number: BPRA057, July 1997

[7] A. Germano, F. Parasiliti, M.Tursini, "Sensorless Speed Control of a PM Synchronous Motor drive by Kalman Filter", International Conference on Electrical Machine (ICEM '94)

[8] C. Manes, F. Parasiliti, M.Tursini, "A Comparative Study of Rotor Flux Estimation in Induction Motors with a Nonlinear Observer and the Extended Kalman Filter", IEEE International Conference on Power Electronics, Control and Instrumentation (IECON '94)

[9] Werner Leonard, "Control of Electrical Drives", 2nd Completely Revised and Enlarged Edition, Springer, ISBN 3-540-59380-2

[10] D. W. Novotny and T. A. Lipo, "Vector Control and Dynamics of AC Drives", Oxford Science Publications, ISBN 0-19-856439-2

[11] K. Astrom, B. Wittenmark, *Adaptive Control*, Addison Wesley, 1995.

[12] C. Phillips, H. Nagle, *Digital Control Systems Analysis and Design*, Prentice Hall, 1995.

[13] Y. Dote, *Servo Motor and Motion Control Using Digital Signal Processors*, Prentice Hall, 1990.

[14] J. Tang, R. Chassaing, *"PID Controller Using the TMS320C31 DSK for Real-Time DC Motor Control,"* Proc. of the 1999 Texas Instruments DSPS Fest, Houston, Texas, August, 1999.

[15] B. Widrow, S. Stearns, *Adaptive Signal Processing*, Prentice Hall, 1985

[16] R. Krishnan, *Electrical Motor Drives: Modeling, Analysis, and Control*, Prentice Hall, 2001.

Appendix A

Summary of Notations

Meaning	Notation
Electrical and Mechanical Torque	T_e, T_L
Flux	Ψ
Flux Angle	θ
Flux Speed	ω_{mR}
Inertia	J
Leakage Factor	$\sigma = 1 - \frac{L_H^2}{L_R L_S}$
Magnetic Pole Number	p
Magnetizing Current	i_{mR}
Phase Currents	i_a, i_b, i_c
Rotor, Stator and Main Inductances	L_R, L_S, L_H
Rotor and Stator Resistances	R_R, R_S
Rotor and Stator Time constants	T_R, T_S
Rotor Angle	ε
Rotor Currents Scalar Components	$i_{R\alpha}, i_{R\beta}$
Rotor Speed	ω
Stator and Rotor Currents	i_R, i_S
Stator Currents in Field Coordinates	i_{Sd}, i_{Sq}
Stator Currents Scalar Components	$i_{S\alpha}, i_{S\beta}$
Voltage	u

Table 2: Summary of Notations Used in this Research Paper

Appendix B

Induction Motor Parameters

The induction motor used in this research paper is a single cage, three-phase, 60Hz, Y-connected induction motor. The rated values and the parameters of this motor are as follows:

Rated power	P=5HP
Rated voltage	V=133V rms (phase)
Rated current	I=12A rms
Rated speed	1800rpm
Pole pairs	2
Slip	0.0278
Stator resistance (R_S)	0.375Ω
Rotor resistance (R_R)	0.405Ω
Magnetizing inductance (L_H)	77mH
Stator leakage inductance ($L_{\sigma s}$)	2.63mH
Stator inductance ($L_S=L_{\sigma S}+L_H$)	79.63mH
Rotor leakage inductance ($L_{\sigma R}$)	2.63mH
Rotor inductance ($L_R=L_{\sigma R}+L_H$)	79.63mH
Leakage coefficient ($\sigma=1-L_H/L_RL_S$)	0.065
Rotor inertia	$19.36*10^{-3}kgm^2$

These values are used in the electromagnetic equation of the induction motor as well as the mechanical equation and the control algorithm.

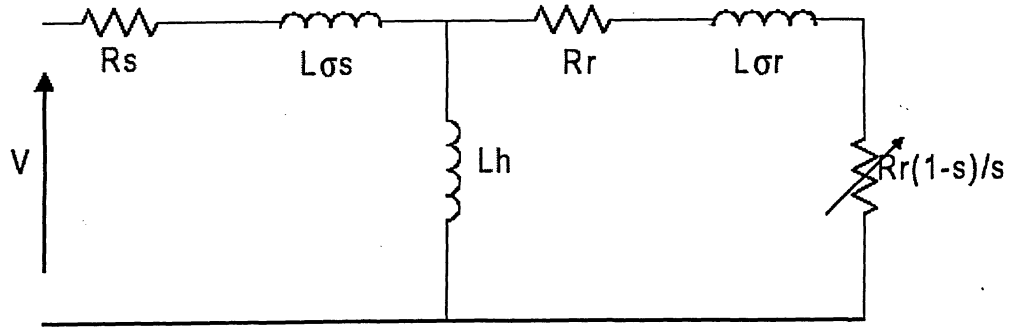


Figure 36: The Model of an Induction Motor

Appendix C

Characteristics of PID Control

The proportional controller (Kp) will have the effect on reducing the rise time a step response and will reduce, but not eliminate, the steady-state error. An integral control (Ki) will have the effect of eliminating the steady-state error, but it may make the transient response worse. A derivative control (Kd) will have the effect of increasing the stability of the system, reducing the overshoot, and improving the transient response. Effects of each of controllers Kp, Kd, and Ki on a closed-loop system are summarized in the table shown below.

CL RESPONSE	RISE TIME	OVERSHOOT	SETTLING TIME	S-S ERROR
Kp	Decrease	Increase	Small Change	Decrease
Ki	Decrease	Increase	Increase	Eliminate
Kd	Small Change	Decrease	Decrease	Small Change

Note that the correlations shown in the table may not be exactly accurate, because Kp, Ki, and Kd are dependent of each other. In fact, changing one of these variables can change the effect of the other two. For this reason, the table should only be used as a reference for determining the values of Ki, Kp and Kd.

Appendix D

Assembly Code for TMS320LF2407

1) Main program- foc2.asm

```
;=================================================================
; System Name:   FOC
;
; File Name:    foc2.asm
;
; Description:  3-phase Field Oriented control with Kalman filter
;
; Originator:   Shiping Zhu - Ryerson University
;
; Target dependency:      LF2407
; To Select the target device see x24x_app.h file.
;
;=================================================================
; Date:
;-----------------------------------------------------------------------------
; 3-9-2003
;=================================================================
;*****************************************************************
;                          SYSTEM OPTIONS
;*****************************************************************
real_time               .set    1       ; 1 for real time mode, otherwise set 0

* Phase commissioning options

phase_commissioning1    .set    0       ; Ramp and sine_wave signal generation
phase_commissioning2    .set    1       ; open_loop start up
phase_commissioning3    .set    0       ; current measurement chain checking
phase_commissioning4    .set    0       ; current regulation implementation
phase_commissioning5    .set    0       ; Kalman filter estimation
phase_commissioning6    .set    0       ; final regulation:closed current and speed loop
                                        ; in FOC control

PWM_PERIOD      .set  200                       ; PWM period in uS (5KHz)
T1PER_          .set  PWM_PERIOD*15             ;*1000nS/(2*33nS)
;*****************************************************************

        .if (phase_commissioning3)
current_observation     .set    1               ; check of Ia_out/Ib_out, clark_d/clark_q

        .endif
* Internal phase commissioning options
        .if (phase_commissioning4)
constant_current        .set    1               ; Theta/Theta_ip switch (0 or rmp_out).
park_sign               .set    0               ; check of park_D/park_Q sign
        .endif

        .if (phase_commissioning5)
theta_est_mod           .set    1               ; 1--rmp_out / 0-- theta_est
        .endif

;-----------------------------------------------------------------------------
; External references
;-----------------------------------------------------------------------------
```

```
        .include   "x24x_app.h"

        .global MON_RT_CNFG

        .ref  SYS_INIT

        .ref  RAMP_GEN, RAMP_GEN_INIT              ;function call
        .ref  rmp_gain, rmp_offset, rmp_freq    ;Inputs
.ref    step_angle_max                          ;Input
        .ref  rmp_out                               ;Outputs

        .ref  I_PARK, I_PARK_INIT                   ;function call
        .ref  ipark_D, ipark_Q, theta_ip        ;Inputs
        .ref  isin_theta,icos_theta             ;Inputs
        .ref  ipark_d, ipark_q                  ;Outputs

        .ref  DATA_LOG, DATA_LOG_INIT                       ;function call
        .ref  dlog_iptr1, dlog_iptr2                ;Inputs
        .ref  trig_value                            ;Inputs

        .ref  SVGEN_DQ,SVGEN_DQ_INIT                ;function call
        .ref  Ualfa,Ubeta                           ;Inputs
        .ref  Ta,Tb,Tc                          ;Outputs
        .ref     FC_PWM_DRV,FC_PWM_DRV_INIT                 ;function calls
        .ref     Mfunc_c1,Mfunc_c2,Mfunc_c3,Mfunc_p  ;Inputs
        .ref     n_period                           ;Input

        .ref  ILEG2DRV, ILEG2DRV_INIT                       ;function call
        .ref  Ia_gain,Ib_gain,Ia_offset,Ib_offset       ;Inputs
        .ref  Ia_out, Ib_out                            ;Outputs

.ref    CLARKE, CLARKE_INIT                     ;function call
        .ref  clark_a, clark_b                  ;Inputs
        .ref  clark_d, clark_q                  ;Outputs

        .ref  PARK, PARK_INIT                       ;function call
        .ref  park_d, park_q, theta_p           ;Inputs
        .ref  psin_theta,pcos_theta             ;Inputs
        .ref  park_D, park_Q                        ;Outputs
.ref      D_sign, Q_sign

        .ref  pid_reg_id,pid_reg_id_init        ; function call
        .ref  id_fdb,id_ref,Kp_d,Ki_d,Kc_d; Inputs
        .ref  ud_int                                ; Input
        .ref  ud_out                                ; Outputs

        .ref  pid_reg_iq,pid_reg_iq_init    ; function call
.ref    iq_fdb,iq_ref,Kp_q,Ki_q,Kc_q; Inputs
        .ref  uq_int                    ; Input
        .ref  uq_out                    ; Outputs

.ref    EKF,EKF_INIT                    ; function call
        .ref  _u_al,_u_be,                  ; Inputs
        .ref  _i_be,_i_al                       ; Inputs
        .ref  iSal_est,iSbe_est,psiRal_est,psiRbe_est,n_est ; Outputs

.ref    theta_est,theta_est_init    ; function call
        .ref  psiRal,psiRbe                 ; Inputs
        .ref  sinTeta_est,cosTeta_est       ; Outputs

.ref    pid_reg_spd,pid_reg_spd_init ; function call
        .ref  spd_fdb,spd_ref           ; Inputs
        .ref  spd_out                   ; Outputs
```

```
;-----------------------------------------------------------------------
; Variable Declarations
;-----------------------------------------------------------------------
        .def  GPR0              ;General purpose registers.
        .bss  GPR0,1        ;General purpose register
        .bss    my_iq_ref,1
        .bss    my_id_ref,1
        .bss    speed_reference,1

;=======================================================================
; V E C T O R     T A B L E     ( including RT monitor traps )
;=======================================================================
            .include "c200mnrt.i"     ; Include conditional assembly options.

        .sect "vectors"
        .def  _c_int0
*       .def  _c_int4         ; int4-EV group C dispatcher/service

RESET       B     _c_int0             ; 00
INT1        B     PHANTOM             ; 02
INT2        B     T1_PERIOD_ISR        ; 04
INT3        B     PHANTOM             ; 06
INT4        B     PHANTOM             ; 08
INT5        B     PHANTOM             ; 0A
INT6        B     PHANTOM             ; 0C

        .include   "rtvecs.h"

; Note : The above include line must be AFTER the user configurable
;        vectors. Do not change the place where this line is included.

;=======================================================================
; M A I N    C O D E   - starts here
;=======================================================================
        .text
_c_int0:
        CALL    SYS_INIT
        CALL    FC_PWM_DRV_INIT ; Here is defined the Timer
                                ;frequency (5KHz), it is thus mandatory
                                ;to include this initialisation since the beginning
        CALL    DATA_LOG_INIT
    .if (phase_commissioning1)
        CALL    RAMP_GEN_INIT
        CALL    I_PARK_INIT

; DATA_LOG display
        LDP #dlog_iptr1;
        SPLK #ipark_d, dlog_iptr1;
*       SPLK #ipark_q, dlog_iptr2
        SPLK #rmp_out, dlog_iptr2
    .endif

    .if (phase_commissioning2)
        CALL    RAMP_GEN_INIT
        CALL    I_PARK_INIT
        CALL    SVGEN_DQ_INIT

; DATA_LOG display

        LDP #dlog_iptr1;
        SPLK #ipark_d, dlog_iptr1;
*       SPLK #ipark_q, dlog_iptr2
```

```
*           SPLK #rmp_out, dlog_iptr2
            SPLK #Ta, dlog_iptr2
      .endif

      .if (phase_commissioning3)
            CALL     RAMP_GEN_INIT
            CALL     I_PARK_INIT
            CALL    SVGEN_DQ_INIT
            CALL     ILEG2DRV_INIT
            CALL     CLARKE_INIT
            CALL     PARK_INIT

; DATA_LOG display

            LDP #dlog_iptr1;
            SPLK #Ia_out, dlog_iptr1;
*           SPLK #Ib_out, dlog_iptr2
*           SPLK #clark_d, dlog_iptr1
            SPLK #clark_q, dlog_iptr2
*           SPLK #park_D, dlog_iptr1
*            SPLK #park_Q, dlog_iptr2

      .endif

      .if (phase_commissioning4)
            CALL     RAMP_GEN_INIT
            CALL     I_PARK_INIT
            CALL    SVGEN_DQ_INIT
            CALL     ILEG2DRV_INIT
            CALL     CLARKE_INIT
            CALL     PARK_INIT
*           CALL     QEP_THETA_DRV_INIT
*           CALL     SPEED_FRQ_INIT
            CALL     pid_reg_id_init
            CALL     pid_reg_iq_init

; DATA_LOG display

            LDP #dlog_iptr1; this is inserted by Shiping
            SPLK #rmp_out, dlog_iptr1;
            splk #Ia_out, dlog_iptr2
      .endif

      .if (phase_commissioning5)
            CALL     RAMP_GEN_INIT
            CALL     I_PARK_INIT
            CALL    SVGEN_DQ_INIT
            CALL     ILEG2DRV_INIT
            CALL     CLARKE_INIT
            CALL     PARK_INIT
            CALL     EKF_INIT
            CALL     pid_reg_id_init
            CALL     pid_reg_iq_init
            CALL     theta_est_init

; DATA_LOG display
      .endif

      .if (phase_commissioning6)
            CALL     I_PARK_INIT
            CALL    SVGEN_DQ_INIT
            CALL     ILEG2DRV_INIT
            CALL     CLARKE_INIT
```

```
            CALL    PARK_INIT
            CALL    EKF_INIT
            CALL    pid_reg_id_init
            CALL    pid_reg_iq_init
            CALL    theta_est_init
            CALL    pid_reg_spd_init

; DATA_LOG display
        .endif

;---Real Time option----------
      .if (real_time)
            CALL    MON_RT_CNFG         ;For Real-Time
      .endif
;-----------------------------------
;       Variables initialization
;-----------------------------------
          LDP   #n_period
          SPLK  #T1PER_,n_period; initialize the PWM period to 5kHz

          LDP   #rmp_freq
          SPLK  #64AAh,rmp_freq; 60Hz frequency for RAMPGEN(5kHz)/=25770d

          LDP   #my_iq_ref
          SPLK  #0000h,my_iq_ref
          LDP   #my_id_ref
          SPLK  #0000h,my_id_ref
          LDP   #speed_reference
          SPLK  #0000h,speed_reference

       .if (phase_commissioning6)
          POINT_B0
          SPLK  #0500h,my_id_ref
       .endif

;-----------------------------------------------------
; System Interrupt Init.
;-----------------------------------------------------
      ;Event Manager
            POINT_EV
            SPLK    #0000001000000000b,IMRA ;Enable T1 Underflow Int (i.e. Period)
            SPLK    #0000000000000000b,IMRC ;
            SPLK    #0FFFFh,IFRA ; Clear all Group A interrupt flags
            SPLK    #0FFFFh,IFRB ; Clear all Group B interrupt flags
            SPLK    #0FFFFh,IFRC ; Clear all Group C interrupt flags

      ;C2xx Core
            POINT_PG0

;---Real Time option ------------------------------------------------
      .if (real_time)
            SPLK    #0000000001000010b,IMR    ;En Int lvl 3,7 (T2 ISR)
                     ;5432109876543210
      .endif


      .if (real_time != 1)
*           SPLK    #0000000000001010b,IMR    ;En Int lvl 4 (CAP3/QEP ISR)
                     ;||||!!!!||||!!!!
                ;5432109876543210
            SPLK    #0000000000000010b,IMR    ;Disable Int lvl 4 (CAP3/QEP ISR)
      .endif
```

```
            SPLK    #0FFFFh, IFR        ;Clear any pending Ints
            EINT                        ;Enable global Ints
            POINT_B0
;---------------------------------------------------------------------

;=====================================================
MAIN:       ;Main system background loop
;=====================================================

M_1   NOP
      NOP
      NOP
      CLRC XF
      B  MAIN
;=====================================================



;=============================================================
; Routine Name: T1_PERIOD_ISR                    Routine Type: ISR
;
; Description:
;
;
; Originator: Shiping Zhu - Ryerson University
;
;; Last Update: 20-Apr 03
;=============================================================
==
T1_PERIOD_ISR:
;Context save regs
            MAR     *,AR1           ;AR1 is stack pointer
            MAR     *+              ;skip one position
            SST #1, *+          ;save ST1
            SST    #0, *+           ;save ST0
            SACH    *+              ;save acc high
            SACL    *           ;save acc low

;NOTE: should use "read-modify-write" to clear Int flags & not SPLK!
            POINT_EV
            SPLK    #0FFFFh,IFRA        ; Clear all Group A interrupt flags (T1 ISR)
            SETC    XF
            SETC    SXM                 ; set sign extension mode
            CLRC    OVM                 ; clear overflow mode

;=====================================================
;Start main section of ISR
;=====================================================
      .if (phase_commissioning1)

; Ramp_Generation module
            CALL    RAMP_GEN
; Inverse Park Module
            LDP     #theta_ip
            BLDD    #rmp_out,theta_ip
           CALL     I_PARK
; PWM driver
           CALL     FC_PWM_DRV; called just to set the PWM period to 5kHz

      .endif

      .if (phase_commissioning2)
```

```
; Ramp generation module
        CALL    RAMP_GEN
; Inverse-Park module
        LDP     #theta_ip
        BLDD    #rmp_out,theta_ip
        CALL     I_PARK
; Space-Vector DQ module
        LDP     #Ualfa
        BLDD    #ipark_d,Ualfa
        BLDD    #ipark_q,Ubeta
        CALL    SVGEN_DQ
; PWM driver
        LDP     #Mfunc_c1
        BLDD    #Ta,Mfunc_c1
        BLDD    #Tb,Mfunc_c2
        BLDD    #Tc,Mfunc_c3
        CALL     FC_PWM_DRV

    .endif

    .if (phase_commissioning3)

; Current leg measurement, Ileg2drv module
        CALL       ILEG2DRV
; Clarke module
        LDP       #clark_a
        BLDD       #Ia_out,clark_a
        BLDD       #Ib_out,clark_b
        CALL       CLARKE

; PARK module
        LDP       #park_d
        BLDD       #clark_d,park_d
        BLDD       #clark_q,park_q
        BLDD       #rmp_out,theta_p
        CALL       PARK
; Ramp generation module
        CALL    RAMP_GEN
; Inverse-Park module
        LDP     #theta_ip
        BLDD    #rmp_out,theta_ip
        CALL     I_PARK
; Space-Vector DQ module
        LDP     #Ualfa
        BLDD    #ipark_d,Ualfa
        BLDD    #ipark_q,Ubeta
        CALL    SVGEN_DQ
; PWM driver
        LDP     #Mfunc_c1
        BLDD    #Ta,Mfunc_c1
        BLDD    #Tb,Mfunc_c2
        BLDD    #Tc,Mfunc_c3
        CALL     FC_PWM_DRV

    .endif

    .if (phase_commissioning4)

; Current leg measurement, Ileg2drv module
        CALL       ILEG2DRV
; Clarke module
        LDP       #clark_a
        BLDD       #Ia_out,clark_a
```

```
            BLDD    #Ib_out,clark_b
            CALL    CLARKE
; PARK module
            LDP     #park_d
            BLDD    #clark_d,park_d
            BLDD    #clark_q,park_q
            .if constant_current        ; here we apply 0 or rmp_out
            SPLK    #0000h,theta_p;
            .else
            BLDD    #rmp_out,theta_p
            .endif
            CALL    PARK
; D-axis current regulator
            LDP     #id_ref
            BLDD    #my_id_ref,id_ref   ;
            BLDD    #park_D,id_fdb
            CALL    pid_reg_id

; Q-axis current regulator
            LDP     #iq_ref
            .if constant_current        ; here we apply 0 or my_iq_ref (user)
            SPLK    #0000h,iq_ref ; check the internal phase_commissioning option
            .else
            BLDD    #my_iq_ref,iq_ref
            .endif
            BLDD    #park_Q,iq_fdb
            CALL    pid_reg_iq
; Ramp generation module
            CALL    RAMP_GEN
; Inverse-Park module
            LDP     #ipark_D
            BLDD    #ud_out,ipark_D
            BLDD    #uq_out,ipark_Q     ; here we apply 0 or rmp_out
            .if     constant_current    ; check the internal phase_commissioning option
            SPLK    #0000h,theta_ip
            .else
            BLDD    #rmp_out,theta_ip
            .endif
            CALL    I_PARK
; Space-Vector DQ module
            LDP     #Ualfa
            BLDD    #ipark_d,Ualfa
            BLDD    #ipark_q,Ubeta
            CALL    SVGEN_DQ
; PWM driver
            LDP     #Mfunc_c1
            BLDD    #Ta,Mfunc_c1
            BLDD    #Tb,Mfunc_c2
            BLDD    #Tc,Mfunc_c3
            CALL    FC_PWM_DRV

        .endif
*****************************************

        .if (phase_commissioning5)

; Current leg measurement, Ileg2drv module
            CALL    ILEG2DRV
; Clarke module
            LDP     #clark_a
            BLDD    #Ia_out,clark_a
            BLDD    #Ib_out,clark_b
            CALL    CLARKE
```

```
; EKF module
        LDP     #_u_al
        BLDD    #ipark_d,_u_al
        BLDD    #ipark_q,_u_be
        BLDD    #clark_d,_i_al
        BLDD    #clark_q,_i_be
        CALL    EKF

; PARK module
        LDP     #park_d
        BLDD    #iSal_est,park_d
        BLDD    #iSbe_est,park_q
        .if theta_est_mod               ; here we connect rmp_out or theta_est_mod
        BLDD    #rmp_out,theta_p        ; check the internal phase_commissioning option
        .else
        BLDD    #sinTeta_est,psin_theta
        BLDD    #cosTeta_est,pcos_theta
        .endif
        CALL    PARK

; theta_est
        LDP     #psiRal
        BLDD    #psiRal_est,psiRal
        BLDD    #psiRbe_est,psiRbe
        CALL    theta_est

; D-axis current regulator
        LDP     #id_ref
        BLDD    #my_id_ref,id_ref   ; 1EB8h gives a current of 2.5A peak
        BLDD    #park_D,id_fdb
        CALL    pid_reg_id
; Q-axis current regulator
        LDP     #iq_ref
        BLDD    #my_iq_ref,iq_ref
        BLDD    #park_Q,iq_fdb
        CALL    pid_reg_iq
; Ramp generation module
        CALL    RAMP_GEN
; Inverse-Park module
        LDP     #ipark_D
        BLDD    #ud_out,ipark_D
        BLDD    #uq_out,ipark_Q
        .if theta_est_mod               ; here we connect rmp_out or theta_est_mod
        BLDD    #rmp_out,theta_ip       ; check the internal phase_commissioning option
        .else
        BLDD    #sinTeta_est,isin_theta
        BLDD    #cosTeta_est,icos_theta
        .endif

        CALL    I_PARK
; Space-Vector DQ module
        LDP     #Ualfa
        BLDD    #ipark_d,Ualfa
        BLDD    #ipark_q,Ubeta
        CALL    SVGEN_DQ
; PWM driver
        LDP     #Mfunc_c1
        BLDD    #Ta,Mfunc_c1
        BLDD    #Tb,Mfunc_c2
        BLDD    #Tc,Mfunc_c3
        CALL    FC_PWM_DRV

    .endif
```

```
****************************************

      .if (phase_commissioning6)

; Current leg measurement, Ileg2drv module
          CALL       ILEG2DRV
; Clarke module
          LDP        #clark_a
          BLDD       #Ia_out,clark_a
          BLDD       #Ib_out,clark_b
          CALL       CLARKE
; SPEED REGULATION
          LDP       #spd_ref
          bldd      #speed_reference,spd_ref
          bldd      #n_est,spd_fdb
          CALL      pid_reg_spd
; EKF module
          LDP        #_u_al
          BLDD      #ipark_d,_u_al
          BLDD      #ipark_q,_u_be
          BLDD       #clark_d,_i_al
          BLDD       #clark_q,_i_be
          CALL       EKF
; PARK module
          LDP        #park_d
          BLDD       #iSal_est,park_d
          BLDD       #iSbe_est,park_q
          BLDD       #sinTeta_est,psin_theta
          BLDD       #cosTeta_est,pcos_theta
          CALL       PARK
; theta_est
          LDP       #psiRal
          BLDD      #psiRal_est,psiRal
          BLDD       #psiRbe_est,psiRbe
          CALL      theta_est

; D-axis current regulator
          LDP        #id_ref
          ;SPLK       #1EB8h,id_ref   ; 1EB8h gives a current of 2.5A peak
          BLDD       #my_id_ref,id_ref
          BLDD       #park_D,id_fdb
          CALL       pid_reg_id
; Q-axis current regulator
          LDP        #iq_ref
          BLDD       #spd_out,iq_ref
          BLDD       #park_Q,iq_fdb
          CALL       pid_reg_iq
; Inverse-Park module
          LDP       #ipark_D
          BLDD       #ud_out,ipark_D
          BLDD       #uq_out,ipark_Q
          BLDD       #sinTeta_est,isin_theta
          BLDD       #cosTeta_est,icos_theta
          CALL       I_PARK
; Space-Vector DQ module
          LDP       #Ualfa
          BLDD      #ipark_d,Ualfa
          BLDD      #ipark_q,Ubeta
          CALL      SVGEN_DQ
; PWM driver
          LDP       #Mfunc_c1
          BLDD      #Ta,Mfunc_c1
          BLDD      #Tb,Mfunc_c2
```

```
        BLDD    #Tc,Mfunc_c3
        CALL     FC_PWM_DRV

    .endif
;Allow variable viewing via DAC o/ps
        CALL    DATA_LOG
*        CALL      DAC_VIEW_DRV

;=================================================================
;End main section of ISR
;=================================================================
;Context restore regs
END_ISR:
        POINT_PG0

        MAR     *, AR1          ;make stack pointer active
        LACL    *-          ;Restore Acc low
        ADDH    *-          ;Restore Acc high
        LST #0, *-          ;load ST0
        LST     #1, *-          ;load ST1
        EINT
        RET


*****************************************************


;=================================================================
; I S R   -   PHANTOM
;
; Description:  Dummy ISR, used to trap spurious interrupts.
;
; Modifies:
;
; Last Update:  11-03-03
;=================================================================
PHANTOM    B    PHANTOM
```

```
;*********************************************************
; File name:     iv2drv.asm
;
; Module Name:      IV2DRV
;
; Initialization Routine: ILEG2DRV_INIT
;
; Description:  Configures and starts ADC for converting two analog
;               inputs with programmable gains and offsets.
;
;                          |~~~~~~~~~~~~~~~~~~~|
; I_ch_sel          o------->|                   |
; Ia_gain           o------->|Q13   ILEG2DRV Q15|----->o Ia_out
; Ib_gain           o------->|Q13             Q15|----->o Ib_out
; Ia_offset         o------->|Q15                |
; Ib_offset         o------->|Q15                |
;                            |___________________|
;
; Target:           x2407
;
;=================================================================================
; History:
;---------------------------------------------------------------------------------
; 3-11-2003
;=================================================================================
; Notes on Configuration
;-----------------------------------------------------------
; 1. Ix_gain has range of -3.999999 --> +3.99999 (i.e. Q13)
; 2. Ix_offset has range -0.999999 --> +0.99999 (i.e. Q15)
; 3. Ix_out has range -0.999999 --> +0.99999 (i.e. Q15)
;       with:
;       1.0 x (VrefHi - VrefLo) = +0.999999 (7FFFh)
;       0.5 x (VrefHi - VrefLo) =   0          (0000/FFFFh)
;       0.0 x (VrefHi - VrefLo) = -0.999999 (8000h)
;
; I_ch_sel HEX values vs Channels selected:
; ____________________________________________
;   | x    |     x     | Ib select | Ia select |
; --------------------------------------------
;
;
; For x2407
;       Ia select: 0,1,2...F   -->   Ch0,1,2,...15
;       Ib select: 0,1,2...F   -->   Ch0,1,2,...15

;-----------------------------------------------------------
; Reference/Prototypes
;-----------------------------------------------------------
;           .ref  ILEG2DRV, ILEG2DRV_INIT                       ;function call
;           .ref  Ia_gain,Ib_gain,Ia_offset,Ib_offset  ;Inputs
;           .ref  Ia_out, Ib_out                                ;Outputs

;-----------------------------------------------------------
; Global Definitions
;-----------------------------------------------------------
            .def  ILEG2DRV, ILEG2DRV_INIT                       ;function call
            .def  Ia_gain,Ib_gain,Ia_offset,Ib_offset  ;Inputs
            .def  Ia_out, Ib_out                                ;Outputs

*****************************************************
* Define Peripherals
*****************************************************
            .include   "x24x_app.h"
```

```
*****************************************************
* Variables
*****************************************************
I_ch_sel  .usect "ileg2drv",1
Ia_gain         .usect "ileg2drv",1
Ib_gain         .usect "ileg2drv",1
Ia_offset .usect "ileg2drv",1
Ib_offset .usect "ileg2drv",1
Ia_out          .usect "ileg2drv",1
Ib_out          .usect "ileg2drv",1
I_temp          .usect "ileg2drv",1

*****************************************************
* Configuration Parameters
*****************************************************
;       .if x2407       ; '2407 EVM
I_ch_sel_ .set  0032h           ; Select Ch3(Phase B),2(Phase A)
;       .endif

;       .if x2407
ACQ_PS_     .set  0001b         ; Acquisition clk p/s=1/2*(conv p/s) -> Acuisition win=4*clk
CON_PS_     .set  0         ; Conversion clk p/s=1/1
CAL_        .set  0         ; Calibration register = 0
;       .endif

Ia_offset_.set  32      ; Offset introduced by XOR
Ib_offset_.set  32      ;
Ia_gain_  .set  1fffh   ; gain=1.0 (Q13)
Ib_gain_  .set  1fffh   ; gain=1.0 (Q13)

ILEG2DRV_INIT

        LDP #I_ch_sel           ;
        SPLK    #I_ch_sel_,I_ch_sel ; Set channel select
        SPLK    #Ia_gain_,Ia_gain       ; Set gains
        SPLK    #Ib_gain_,Ib_gain       ;
        SPLK    #Ia_offset_,Ia_offset; Set offsets
        SPLK    #Ib_offset_,Ib_offset   ;

        ldp  #GPTCON>>7     ; Set T1UF as ADC trigger
        splk #0000h, GPTCON
;       lacc GPTCON
;       and  #0FE7Fh ;1111,1110,0111,1111
;       or   #01b<<7 ;0000 0000 1000 0000
;       sacl GPTCON

;       .if x2407
        LDP #CALIBRATION>>7     ; Configure CALIBRATION
        SPLK    #CAL_,CALIBRATION  ; Set Calibration register
        SPLK    #4000h,ADCL_CNTL1  ; Reset entire Module
        SPLK    #0010000101010000b,ADCL_CNTL1 ;Acq = 4 x Clks,Cascaded mode
                                                ; continuous run
;       SPLK    #(03000h+ACQ_PS_<<8+CON_PS_<<7),ADCL_CNTL1    ;           Config
ADCL_CNTL1
                ;0011000100000000
;       SPLK    #0302h,ADCL_CNTL2 ; Allow EVA to start conversion
                ;0000001100000010
        SPLK    #1,MAXCONV          ; 2 conversions
        bldd #I_ch_sel,CHSELSEQ1  ; Configure channel select
;       .endif
        SPLK #0010000000000000b,ADCL_CNTL2 ; Start the conversions
        ret
```

```
ILEG2DRV:

*       .if x2407
        ldp  #ADCL_CNTL2>>7     ; Check SEQ_BSY bit
;Wait
;       bit  ADCL_CNTL2,BIT12   ;
;       bcnd Wait,TC            ;

        LACC    ADC_RESULT0         ; Read 1st converted value
        XOR     #8000h              ; Convert to Q15
        ldp  #Ia_out
        SACL    Ia_out
        LT   Ia_gain            ; Ia_gain in Q13
        MPY     Ia_out              ; Q13 x Q15 = Q28
        PAC
        ADD     Ia_offset,13    ; add offset in Q28
*       neg                     ; positive => going into motor
        SACH    Ia_out,3            ; Convert final result to Q15

        LDP #ADC_RESULT1>>7     ; Read 2nd converted value
        LACC    ADC_RESULT1
        XOR     #8000h              ; Convert to Q15
        ldp  #Ib_out
        SACL    Ib_out
        LT   Ib_gain            ; Ib_gain in Q13
        MPY     Ib_out              ; Q13 x Q15 = Q28
        PAC
        ADD     Ib_offset,13    ; add offset in Q28
*       neg                     ; positive => going into motor
        SACH    Ib_out,3            ; Convert final result to Q15

;       bldd I_ch_sel,#CHSELSEQ1    ; Reconfigure channel select
*       .endif

        RET
```

```
;=================================================================
; File Name:    park.asm
;
; Module Name:      PARK
;
;
; Description:  This transformation converts vectors in balanced 2-phase
;           orthogonal stationary system into orthogonal rotating
;           reference frame.
;
;               id = ialfa * cos_teta + ibeta * sin_teta
;               iq = -ialfa *sin_teta + ibeta * cos_teta
;
;                       |~~~~~~~~~~~~~~~~~~|
;     park_d        o------>|                  |----->o   park_D
;     park_q        o------>|       PARK       |
;     theta_p       o------>|                  |----->o   park_Q
;or psin_theta               |__________________|
;     pcos_theta
;
;     Note:     0 < theta_p < 7FFFh     (i.e. equivalent to     0 < theta_p < 360 deg   )
```

```
;
;
;
; Target dependency:C2xx core only
;
;
;===========================================================================
; History:
;---------------------------------------------------------------------------
; 3-11-2003
;===========================================================================
;(To use this Module, copy this section to main system file)
;           .ref  PARK, PARK_INIT                    ;function call
;           .ref  park_d, park_q, theta_p          ;Inputs
;          .ref     psin_theta,pcos_theta           ;Inputs
;           .ref  park_D, park_Q                     ;Outputs
;===========================================================================
;Module definitions for external reference.
           .def  PARK, PARK_INIT                    ;function call
           .def  park_d, park_q, theta_p          ;Inputs
          .def     psin_theta,pcos_theta           ;Inputs
           .def  park_D, park_Q                     ;Outputs
          .def     D_sign, Q_sign                  ;D,Q sign
;===========================================================================
;Options
;--------
High_precision.set  0     ;Set to 1 for High prec / Set to 0 for low prec
theta_est_mod      .set     1     ;1--rmp_out / 0-- theta_est
negtive_sign       .set     0
;===========================================================================

           .ref  SINTAB_360
*           .ref     theta_est_mod    ;1--rmp_out / 0-- theta_est


park_d          .usect "park",1
park_q          .usect "park",1
theta_p         .usect "park",1
park_D          .usect "park",1
park_Q          .usect "park",1
t_ptr       .usect "park",1
ip_val          .usect "park",1
cos_theta .usect "park",1
sin_theta  .usect "park",1
nxt_entry .usect "park",1
delta_angle     .usect "park",1
GPR0_park     .usect "park",1
psin_theta     .usect "park",1
pcos_theta     .usect "park",1
D_sign          .usect "park",1
Q_sign          .usect "park",1

;===========================================================================
PARK_INIT:
;===========================================================================
           RET


;===========================================================================
PARK:
;===========================================================================
*      ldp #theta_est_mod
   .if   theta_est_mod
```

```
;Calculate Cos(theta_p)
;--- High_precision option -------
      .if (High_precision)      ;see park for its definition
      ;Higher precision using look-up + interpolation method

            ldp  #theta_p
            LACC    theta_p
            ADD     #8192           ;add 90 deg, i.e. COS(A)=SIN(A+90)
            AND     #07FFFh  ;Force positive wrap-around
            SACL    GPR0_park           ;here 90 deg = 7FFFh/4=8192d
            LACC    GPR0_park,9

            SACH    t_ptr       ;Table pointer
            SFR             ;Convert Interpolation value(ip_val) to Q15
            AND     #07FFFh  ;Force ip_val to a positive number
            SACL    ip_val
            LACC    #SINTAB_360
            ADD     t_ptr
            TBLR    cos_theta ;cos_theta = Cos(theta) in Q15
            ADD     #1h         ;Inc Table pointer
            TBLR    nxt_entry ;Get next entry i.e. (Entry + 1)
            LACC    nxt_entry
            SUB cos_theta ;Find Delta of 2 points
            SACL    delta_angle
            LT   delta_angle
            MPY     ip_val          ;ip_val = interpolation value
            PAC
            SACH    ip_val,1
            LACC    ip_val
            ADD     cos_theta
            SACL    cos_theta ;cos_theta = Final interpolated value
      .endif
;-----------------------------------

;--- Normal precision option -------
      .if (High_precision != 1)
      ;Normal precision with simple 256 word look-up

            ldp  #theta_p
            LACC    theta_p
            ADD     #8192           ;add 90 deg, i.e. COS(A)=SIN(A+90)
            AND     #07FFFh  ;Force positive wrap-around
            SACL    GPR0_park           ;here 90 deg = 7FFFh/4
            LACC    GPR0_park,9
            SACH    t_ptr
            LACC    #SINTAB_360
            ADD     t_ptr
            TBLR    cos_theta ;cos_theta = Cos(theta_p) in Q15
      .endif
;-----------------------------------

;Calculate Sin(theta_p)
;--- High_precision option -------
      .if (High_precision)
      ;Higher precision using look-up + interpolation method

            LACC    theta_p,9
            SACH    t_ptr       ;Table pointer
            SFR             ;Convert Interpolation value(ip_val) to Q15
            AND     #07FFFh  ;Force ip_val to a positive number
            SACL    ip_val
            LACC    #SINTAB_360
            ADD     t_ptr
```

```
            TBLR    sin_theta  ;sin_theta = Sin(theta) in Q15
            ADD     #1h        ;Inc Table pointer
            TBLR    nxt_entry ;Get next entry i.e. (Entry + 1)
            LACC    nxt_entry
            SUB sin_theta  ;Find Delta of 2 points
            SACL    delta_angle
            LT  delta_angle
            MPY     ip_val          ;ip_val = interpolation value
            PAC
            SACH    ip_val,1
            LACC    ip_val
            ADD     sin_theta
            SACL    sin_theta  ;sin_theta = Final interpolated value
        .endif
;-----------------------------------

;--- Normal precision option -------
        .if (High_precision != 1)
        ;Lower precision simple 256 word look-up
            LACC    theta_p,9
            SACH    t_ptr
            LACC    #SINTAB_360
            ADD     t_ptr
            TBLR    sin_theta  ;sin_theta = Sin(theta_p) in Q15
        .endif
;-----------------------------------
    .else
        ldp     #theta_p
        splk    #psin_theta, sin_theta
        splk    #pcos_theta, cos_theta
    .endif
;Calculate the Park transform
            SPM 1           ; SPM set for Q15 multiplication
            ZAC             ; Reset accumulator
            LT  park_q      ; TREG = ibeta
            MPY     sin_theta       ; PREG = ibeta * sin_teta
            LTA park_d      ; ACC = ibeta * sin_teta and TREG = ialfa
            MPY     cos_theta       ; PREG = ialfa * cos_teta
            MPYA    sin_theta       ; ACC = ibeta*sin_teta + ialfa*cos_teta and PREG=ialfa*sin_teta
            SACH    park_D          ; id = ialfa * cos_teta + ibeta * sin_teta

            LACC    #0              ; Clear ACC
            LT  park_q      ; TREG = ibeta
            MPYS    cos_theta       ; ACC = -ialfa*sin_teta and PREG = ibeta*cos_teta
            APAC                    ; ACC = -ialfa *sin_teta + ibeta * cos_teta
            SACH    park_Q      ; iq = -ialfa *sin_teta + ibeta * cos_teta

            SPM0            ; SPM reset
            RET
```

```
;=====================================================================
; File Name:    clarke.asm
;
; Module Name:      CLARKE
;
; Initialization Routine: CLARKE_INIT
;
;
; Description:  Converts balanced three phase quantities into balanced
;           two phase quadrature quantities.
;                       (a,b,c) -> (d,q) Transformation
;                       clark_d = clark_a
;                       clark_q = (2 * clark_b + clark_a) / sqrt(3)
;
;                           |~~~~~~~~~~~~~~~~~~|
;     clark_a       o------>|                  |----->o   clark_d
;     clark_b       o------>|      CLARKE      |
;     clark_c*      o------>|                  |----->o   clark_q
;                           |__________________|
;
;     *Note: Not needed (for completeness only)
;
;
; Target dependency:C2xx core only
;
;
;=====================================================================
; History:
;---------------------------------------------------------------------
; 3-11-2003
;=====================================================================
;(To use this Module, copy this section to main system file)
;           .ref  CLARKE, CLARKE_INIT                   ;function call
;           .ref  clark_a, clark_b                  ;Inputs
;           .ref  clark_d, clark_q                  ;Outputs
;=====================================================================
;Module definitions for external reference.
           .def  CLARKE, CLARKE_INIT                   ;function call
           .def  clark_a, clark_b                  ;Inputs
           .def  clark_d, clark_q                  ;Outputs
;=====================================================================
           .include "x24x_app.h"

; Variables
clark_a         .usect "clarke",1
clark_b         .usect "clarke",1
clark_d         .usect "clarke",1
clark_q         .usect "clarke",1
sqrt3inv   .usect "clarke",1                    ; 1/sqrt(3)
clk_temp  .usect "clarke",1

;=========================
CLARKE_INIT:
;=========================
; sqrt3inv = (1/sqrt(3))=0.577350269
           ldp        #sqrt3inv
           SPLK       #018830,sqrt3inv     ; 1/sqrt(3) (Q15)
           RET

;=========================
CLARKE:
;=========================
           SPM 1                ; Set SPM for Q15 math
```

```
        SETC    SXM                     ; Sign extension mode on

;clark_d = clark_a
        ldp  #clark_a
        LACC    clark_a                 ; ACC = clark_a
        SACL    clark_d             ; clark_d = clark_a

;clark_q = (2 * clark_b + clark_a) / sqrt(3)
        SFR                             ; ACC = clark_a/2
        ADD     clark_b             ; ACC = clark_a/2 + clark_b
        SACL    clk_temp                ; clk_temp = clark_a/2 + clark_b
        LT   clk_temp           ; TREG = clark_a/2 + clark_b
        MPY     sqrt3inv        ; PREG = (clark_a/2 + clark_b)*(1/sqrt(3))
        PAC                             ; ACC =   (clark_a/2 + clark_b)*(1/sqrt(3))
        SFL                             ; ACC =   (clark_a + clark_b*2)*(1/sqrt(3))
        SACH    clark_q         ; clark_q = (clark_a + clark_b*2)*(1/sqrt(3))
        SPM0                            ; SPM reset
        RET



;=============================================================================
; Filename:     I_PARK.asm
; Routine Name:     I_PARK
; Routine Type:     SR
; Description:
;
; id = ialfa * cos_teta - ibeta * sin_teta
; iq = ialfa *sin_teta + ibeta * cos_teta
;
;                       |~~~~~~~~~~~~~~~~|
; ipark_D       o---->|                |----->o   ipark_d
; ipark_Q       o---->|     I_PARK     |
; theta_ip      o---->|                |----->o   ipark_q
;or isin_theta          |________________|
;    icos_theta
;
; Note: 0<theta_ip<7FFFh    (i.e. equivalent to 0<theta_ip<360 deg   )
;
; # Cycles (including call/ret):
;
; Target dependency:C2xx core only
;
; Last Update: 3-11-2003
;=============================================================================
;(To use this Module, copy this section to main system file)
;           .ref  I_PARK,I_PARK_INIT            ; function call
;           .ref  ipark_D,ipark_Q,theta_ip      ; Inputs
;           .ref  isin_theta,icos_theta             ;Inputs
;           .ref  ipark_d,ipark_q                   ; Outputs
;=============================================================================
;Module definitions for external reference.
           .def  I_PARK,I_PARK_INIT             ; function call
           .def  ipark_D,ipark_Q,theta_ip ; Inputs
          .def      isin_theta,icos_theta       ; inputs
           .def  ipark_d,ipark_q                   ; Outputs

;=============================================================================
;Options
;--------
High_precision.set  0     ; Set to 1 for High prec / Set to 0 for low prec
theta_est_mod     .set      1     ;1--rmp_out / 0-- theta_est
;=============================================================================
```

```
            .ref  SINTAB_360   ; Sine table
*           .ref      theta_est_mod


ipark_d    .usect "I_park",1
ipark_q    .usect "I_park",1
theta_ip   .usect "I_park",1
ipark_D    .usect "I_park",1
ipark_Q    .usect "I_park",1
isin_theta .usect "I_park",1
icos_theta .usect "I_park",1

t_ptr      .usect "I_park",1
ip_val     .usect "I_park",1
cos_theta  .usect "I_park",1
sin_theta  .usect "I_park",1
nxt_entry  .usect "I_park",1
delta_angle     .usect "I_park",1
GPR0_ipark     .usect "I_park",1
;=========================
I_PARK_INIT:
;=========================
           ldp  #ipark_D
           SPLK     #3FFFh, ipark_D
           SPLK     #3FFFh, ipark_Q
           SPLK      #0000h, theta_ip; this is added by Shiping
           RET

;=========================
I_PARK:
;=========================
*      ldp #theta_est_mod
      .if   theta_est_mod
      ;Calculate Cos(theta_p)
;--- High_precision option -------
      .if (High_precision)
      ;Higher precision using look-up + interpolation method
           ldp  #theta_ip
           LACC     theta_ip
           ADD      #8192           ;add 90 deg, i.e. COS(A)=SIN(A+90)
           AND      #07FFFh  ;Force positive wrap-around
           SACL     GPR0_ipark     ;here 90 deg = 7FFFh/4=8192d
           LACC     GPR0_ipark,9
           SACH     t_ptr       ;Table pointer
           SFR              ;Convert Interpolation value(ip_val) to Q15
           AND      #07FFFh  ;Force ip_val to a positive number
           SACL     ip_val
           LACC     #SINTAB_360
           ADD      t_ptr
           TBLR     cos_theta  ;cos_theta = Cos(theta) in Q15
           ADD      #1h         ;Inc Table pointer
           TBLR     nxt_entry ;Get next entry i.e. (Entry + 1)
           LACC     nxt_entry
           SUB cos_theta ;Find Delta of 2 points
           SACL     delta_angle
           LT   delta_angle
           MPY      ip_val      ;ip_val = interpolation value
           PAC
           SACH     ip_val,1
           LACC     ip_val
           ADD      cos_theta
           SACL     cos_theta  ;cos_theta = Final interpolated value
     .endif
```

```
;-----------------------------------

;--- Normal precision option -------
      .if (High_precision != 1)
      ;Normal precision with simple 256 word look-up
            ldp   #theta_ip
            LACC    theta_ip
            ADD     #8192           ;add 90 deg, i.e. COS(A)=SIN(A+90)
            AND     #07FFFh  ;Force positive wrap-around
            SACL    GPR0_ipark     ;here 90 deg = 7FFFh/4
            LACC    GPR0_ipark,9
            SACH    t_ptr
            LACC    #SINTAB_360
            ADD     t_ptr
            TBLR    cos_theta  ;cos_theta = Cos(theta_p) in Q15
      .endif
;-----------------------------------

;Calculate Sin(theta_p)
;--- High_precision option -------
      .if (High_precision)
      ;Higher precision using look-up + interpolation method
            LACC    theta_ip,9
            SACH    t_ptr       ;Table pointer
            SFR             ;Convert Interpolation value(ip_val) to Q15
            AND     #07FFFh  ;Force ip_val to a positive number
            SACL    ip_val
            LACC    #SINTAB_360
            ADD     t_ptr
            TBLR    sin_theta  ;sin_theta = Sin(theta) in Q15
            ADD     #1h         ;Inc Table pointer
            TBLR    nxt_entry ;Get next entry i.e. (Entry + 1)
            LACC    nxt_entry
            SUB sin_theta  ;Find Delta of 2 points
            SACL    delta_angle
            LT   delta_angle
            MPY     ip_val      ;ip_val = interpolation value
            PAC
            SACH    ip_val,1
            LACC    ip_val
            ADD     sin_theta
            SACL    sin_theta  ;sin_theta = Final interpolated value
      .endif
;-----------------------------------

;--- Normal precision option -------
      .if (High_precision != 1)
      ;Lower precision simple 256 word look-up
            LACC    theta_ip,9
            SACH    t_ptr
            LACC    #SINTAB_360
            ADD     t_ptr
            TBLR    sin_theta  ;sin_theta = Sin(theta_p) in Q15
      .endif
;-----------------------------------
      .else
        ldp     #theta_ip
        splk    #isin_theta, sin_theta
        splk    #icos_theta, cos_theta
      .endif

;Calculate the Inverse Park transform
```

```
        SETC    SXM             ; Sign extension mode
        SPM1            ; SPM set for Q15 multiplication

;park_q = ipark_Q * cos_theta + ipark_D * sin_theta

        LACC    #0              ; Clear ACC
        LT  ipark_D         ; TREG = Udref
        MPY     sin_theta       ; PREG = Udref * sin_theta
        LTA ipark_Q         ; ACC = Udref*sin_theta and TREG=Uqref
        MPY     cos_theta       ; PREG = Uqref * cos_teta
        MPYA    sin_theta       ;   ACC   =   Uqref*cos_theta   +   Udref*sin_theta   and
TREG=Uqref*sin_theta
        SACH    ipark_q         ; Ubeta = Uqref*cos_theta + Udref*sin_theta

;park_d = ipark_D * cos_theta - ipark_Q * sin_theta
        LACC    #0              ; Clear ACC
        LT  ipark_D         ; TREG = Udref
        MPYS    cos_theta       ; ACC = -Uqref*sin_theta and PREG = Udref*cos_theta
        APAC                    ; ACC = -Uqref*sin_theta + Udref*cos_theta
        SACH    ipark_d         ; Ualfa = -Uqref*sin_theta + Udref*cos_theta

        SPM0            ; SPM reset
        RET
```

```
;=====================================================================
; File Name:    EKF.asm
;
; Module Name:      Kalman filter
;
; Initialization Routine: EKF_INIT
;
;
; Description:  Kalman filter for field oriented control
;                   of an AC induction machine
;
;               |~~~~~~~~~~~~~~~~~~|
; _u_al           ->|                  |->iSal_est
; _u_be           ->|  KALMAN FILTER |->iSbe_est
; _i_al           ->|                  |->psiRal_est
; _i_be           ->|                     |->psiRbe_est
;                   |_____________________|->n_est
;
;
; Target:       x2407
;
;=====================================================================
; History:
```

```
;-------------------------------------------------------------------------------
; 3-17-2003
;-------------------------------------------------------------
; Reference/Prototype
;-------------------------------------------------------------
;           .ref  EKF,EKF_INIT                    ; function call
;           .ref  _u_al,_u_be,                    ; Inputs
;           .ref  _i_be,_i_al                     ; Inputs
;           .ref  iSal_est,iSbe_est,psiRal_est,psiRbe_est,n_est ; Outputs
*             .include "matcal.inc"
             .copy   "matcal.inc"
*            .mlib   "mac.lib"          ;EKF macros file
*            .copy   "matGC24.inc"      ;EKF macros file
*             .copy   "maddG.asm"
*             .copy   "madiG.asm"
*             .copy   "mmsclG.asm"
*             .copy   "mmtfraG.asm"
*             .copy   "msubG.asm"
*             .copy   "detG.asm"
*             .copy   "mmfraG.asm"
*             .copy   "mdsclG.asm"
*             .copy   "madjG.asm"
;-------------------------------------------------------------
; Global Definitions
;-------------------------------------------------------------
     .def  EKF,EKF_INIT                       ; function call
           .def  _u_al,_u_be,                 ; Inputs
           .def  _i_be,_i_al                  ; Inputs
           .def  iSal_est,iSbe_est,psiRal_est,psiRbe_est,n_est ; Outputs

;-------------------------------------------------------------
; Variable Definitions
;-------------------------------------------------------------
;public variables
;_u_al          .usect "ekf",1
;_u_be          .usect "ekf",1                                              i
;_i_be          .usect "ekf",1
;_i_al          .usect "ekf",1
iSal_est      .usect "ekf",1      ;alfa-axis estimated current, x(1)
iSbe_est      .usect "ekf",1      ;beta-axis estimated current, x(2)
psiRal_est    .usect "ekf",1      ;alfa-axis estimated rotor flux, x(3)
psiRbe_est    .usect "ekf",1      ;beta-axis estimated rotor flux, x(4)
n_est         .usect "ekf",1      ;estimated electrical speed, x(5)

;private variables

*psiR_est       .usect "ekf" ,1     ;rotor flux amplitude
*psiRal_est1    .usect "ekf",1      ;1.15 format (alfa-axis estimated rotor flux), x(3)
*psiRbe_est1    .usect "ekf",1      ;1.15 format (beta-axis estimated rotor flux), x(4)
*psiR_est1      .usect "ekf",1      ;1.15 format rotor flux amplitude
*run             .usect "ekf",1     ;initialization flag
*est             .usect "ekf",1     ;estimation flag:0h=initialization, 1000h=sensorless


;-------------------------------------------------------------
; Default parameters
;-------------------------------------------------------------
************************************************************
* Variables and constants initializations
************************************************************

                .data
```

```
*** data page EKF algorithm***

kal_fil                         ;EKF algorithm data page pointer

u                               ;Input vector
_u_al   .word   0                   ;ua
_u_be   .word   0                   ;ub

y                               ;Output vector
_i_al   .word   0                   ;ia
_i_be   .word   0                   ;ib

diff_f  .word   29591, 0, 747, 0, 0         ;LAFERT_TI, with T=300us
        .word   0, 29591, 0, 747, 0
        .word   696, 0, 32442, 0, 0
        .word   0, 696, 0, 32442, 0
        .word   0, 0, 0, 0, 32767

_x      .word   0, 0, 0, 0, 0       ;state vector x(k,k)

Cx_1                            ;Cx_1=[x(1) x(2)]'
_x_1    .word   0, 0, 0, 0, 0       ;Predicted state vector x(k+1,k)

C4      .word   14164               ;[1,4] & [1,5] coeff of diff_f
C9      .word   -6177                   ;[3,4] & [3,5] coeff of diff_f
C5      .word   15400               ;wbase*T/xsig coeff
C8      .word   6177                ;[4,3] & [4,5] coeff of diff_f
C10     .word   -14164                  ;[2,3] & [2,5] coeff of diff_f

C       .word   32767, 0, 0, 0, 0   ;C matrix
        .word   0, 32767, 0, 0, 0


P       .word   320, 0, 0, 0, 0             ;Prediction err cov mat P(k+1,k)
        .word   0, 320, 0, 0, 0
        .word   0, 0, 320, 0, 0
        .word   0, 0, 0, 320, 0
        .word   0, 0, 0, 0, 320

P_1     .word   0, 0, 0, 0, 0       ;Filter err cov mat P(k+1,k+1)
        .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0
        .word   0, 0, 0, 0, 0

Q       .word   1000, 1000, 300, 300, 500 ;state noise cov matrix

R       .word   20000, 20000            ;Measure Noise Cov Matrix

RSLT    .space 25*16                ;Result mat, not bigger than 5*5

_K      .word   0, 0
        .word   0, 0
        .word   0, 0
        .word   0, 0
        .word   0, 0

Pole_pairs_number   .set    2

;-------------------------------------------------------------
; Initialization
;-------------------------------------------------------------
```

```
        .text

EKF_INIT

        ldp     #n_est
        splk #0000h,iSal_est
        splk #0000h,iSbe_est
        splk    #0000h,n_est
        splk    #0000h,psiRal_est
        splk    #0000h,psiRbe_est
        ret


EKF
*************************************************************
*** EXTENDED KALMAN FILTER                                        ***
*** estimated variables (iSal_est,iSbe_est)                 ***
***                     (psiRal_est,psiRbe_est)             ***
***                     (n_est)                                 ***
*************************************************************

;diff_f matrix update
;diff_f=
;[1-T/Ts2,0,(xH*T)/(xsig*xR*TR),((xH*wBase*T)/(xsig*xR))*x(5),((xH*wBase*T)/(xsig*xR))*x(4);
;0,1-T/Ts2,-((xH*wBase*T)/(xsig*xR))*x(5),(xH*T)/(xsig*xR*TR),-((xH*wBase*T)/(xsig*xR))*x(3);
;               (xH*T)/TR, 0, 1-T/TR,-wBase*T*x(5),-wBase*T*x(4) ;
;               0, (xH*T)/TR, wBase*T*x(5),1-T/TR, wBase*T*x(3);
;               0, 0, 0, 0, 1];
    ldp     #kal_fil
    lar     AR2,#_x+4;load in AR4 the address of _x+4=x(5)
    lar     AR3,#diff_f+3   ;load in AR3 the address of A(k)[1,4]
    lar     AR6,#C4   ;load in AR6 the address of C4
    mar      *,AR2          ;ARP=2
    lt      *-,AR6          ;Treg=x(5), AR2->x(4), ARP=6
    mpy      *+,AR3           ;Preg=C4*x(5), AR6->C9, ARP=3
    pac          ;Acc=Preg
    sach    *,1             ;A(k)[1,4]=C4*x(5) in Q15 format
    neg          ;Acc= -Preg
    adrk    4           ;add 4 to the address loaded in AR3 <-> AR3->A(k)[2,3]
    sach    *,1,AR6         ; A(k)[2,3]=-C4*x(5) in Q15 format, ARP=6 (N.B. AR6->C9)
    mpy      *,AR3            ;Preg=C9*x(5)=-C8*x(5), ARP=3
    pac          ;Acc=Preg
    adrk    6           ;add 6 to the address loaded in AR3 <-> AR3->A(k)[3,4]
    sach    *,1             ;A(k)[3,4]=C9*x(5)=-C8*x(5) in Q15 format
    neg          ;Acc=-Preg
    adrk    4           ;add 4 to the address loaded in AR3 <-> AR3->A(k)[4,3]
    sach    *,1,AR2         ;A(k)[4,3]=C8*x(5) in Q15 format, ARP=2
    lt      *-,AR6          ;Treg=x(4), AR2->x(3), ARP=6
    mpy      *-,AR3           ;Preg=C9*x(4)=-C8*x(4), AR6->C4, ARP=3
    pac          ;Acc=Preg
    sbrk    3           ;sub 3 to the address loaded in AR3 <-> AR3->A(k)[3,5]
    sach    *,1,AR6         ;A(k)[3,5]=-C8*x(4) in Q15 format, ARP=6
    mpy      *,AR3            ;Preg=x(4)*C4, ARP=3
    pac          ;Acc=Preg
    sbrk    10          ;sub 10 to the address loaded in AR3 <-> AR3->A(k)[1,5]
    sach    *,1,AR2         ;A(k)[1,5]=C4*x(4) in Q15 format, ARP=2
    lt      *               ;Treg=x(3)
    lar     AR6,#C10  ;Load in AR6 the address of C10
    mar      *,AR6          ;ARP=6
    mpy      *-,AR3           ;Preg=x(3)*C10=-C4*x(3), AR6->C8, ARP=3
    pac          ;Acc=Preg
    adrk    5           ;add 5 to the address loaded in AR3 <-> AR3->A(k)[2,5]
    sach    *,1,AR6         ;A(k)[2,5]=C10*x(3)=-C4*x(3) in Q15 format, ARP=6
```

```
      mpy     *,AR3           ;Preg=x(3)*C8, ARP=3
      pac           ;Acc=Preg
      adrk    10          ;add 10 to the address loaded in AR3 <-> AR3->A(k)[4,5]
      sach    *,1               ; A(k)[4,5]=C8*x(3) in Q15 format.

;diff_f matrix is now updated

;x_1= state vector prediction = x(k/k-1)
;
;x_1=[diff_f(1,1)*x(1)+diff_f(1,3)*x(3)+diff_f(1,4)*x(4);
;       diff_f(2,2)*x(2)+diff_f(2,3)*x(3)+diff_f(2,4)*x(4);
;       diff_f(3,1)*x(1)+diff_f(3,3)*x(3)+diff_f(3,4)*x(4);
;       diff_f(4,2)*x(2)+diff_f(4,3)*x(3)+diff_f(4,4)*x(4);
;       diff_f(5,5)*x(5)]...
;    +[((wBase*T)/xsig)*u(1);((wBase*T)/xsig)*u(2); 0; 0; 0];

      spm     1               ;set default left shift on Preg for Q15 calculations
      setc    ovm
      lar     AR4,#_x_1       ;load in AR4 the address of x_1 (=x(k/k-1))
      lar     AR2,#_x                 ;load in AR2 the address of x (=x(k-1/k-1))
      lar     AR3,#diff_f             ;load in AR3 the address of diff_f
      lar     AR6,#C5         ;load in AR6 the address of C5
      mar     *,AR2                 ;ARP=2
      lt      *+              ;Treg=x(1), AR2->x(2)
      mar     *+,AR3                ;AR2->x(3), ARP=3
      mpy     *+              ;Preg=x(1)*A(k)[1,1], AR3-> A(k)[1,2]
      mar     *+,AR2                ;AR3-> A(k)[1,3], ARP=2
      ltp     *+,AR3                ;Acc=Preg, Treg=x(3), AR2->x(4), ARP=3
      mpy     *+,AR2                ;Preg=x(3)*A(k)[1,3], AR3-> A(k)[1,4], ARP=2
      lta     *+,AR3                ;Acc=Acc+Preg, Treg=x(4), AR2->x(5), ARP=3
      mpy     *+,AR2                ;Preg=x(4)* A(k)[1,4], AR3-> A(k)[1,5], ARP=2
      lar     AR2,#u                  ;load in AR2 the address of u(1)
      lta     *,AR6           ;Acc=Acc+Preg, Treg=u(1), ARP=6
      mpy     *,AR4                     ;Preg=C5*u(1), ARP=4
      apac              ;Acc=Acc+Preg
      sach    *+,AR2                  ;store in x_1(1), AR4->x_1(2), ARP=2

      lar     AR2,#_x+1       ;load in AR2 the address of x(2)
      lar     AR3,#diff_f+6         ;load in AR3 the address of A(k)[2,2]
      lt      *+,AR3                ;Treg=x(2), AR2->x(3), ARP=3
      mpy     *+,AR2                ;Preg=x(2)*A(k)[2,2], AR3->A(k)[2,3], ARP=2
      ltp     *+,AR3                ;Acc=Preg, Treg=x(3), AR2->x(4), ARP=3
      mpy     *+,AR2                ;Preg=x(3)*A(k)[2,3], AR3->A(k)[2,4], ARP=2
      lta     *+,AR3                ;Acc=Acc+Preg, Treg=x(4), AR2->x(5), ARP=3
      mpy     *+,AR2                ;Preg=x(4)*A(k)[2,4], AR3->A(k)[2,5], ARP=2
      lar     AR2,#u+1        ;load in AR2 the address of u(2)
      lta     *,AR6           ;Acc=Acc+Preg, Treg=u(2), ARP=6
      mpy     *,AR4                     ;Preg=C5*u(2), ARP=4
      apac              ;Acc=Acc+Preg
      sach    *+,AR2                  ;store in x_1(2), AR4->x_1(3), ARP=2

      lar     AR2,#_x         ;load in AR2 the address of x(1)
      lar     AR3,#diff_f+10        ;load in AR3 the address of A(k)[3,1]
      mar     *,AR2                 ;ARP=2
      lt      *+              ;Treg=x(1), AR2->x(2), ARP=3
      mar     *+,AR3                ;AR2->x(3), ARP=3
      mpy     *+              ;Preg=x(1)*A(k)[3,1], AR3-> A(k)[3,2]
      mar     *+,AR2                ;AR3->A(k)[3,3], ARP=2
      ltp     *+,AR3                ;Acc=Preg, Treg=x(3), AR2->x(4), ARP=3
      mpy     *+,AR2                ;Preg=x(3)*A(k)[3,3], AR3->A(k)[3,4], ARP=2
      lta     *+,AR3                ;Acc=Acc+Preg, Treg=x(4), AR2->x(5), ARP=3
      mpy     *+,AR4                ;Preg=x(4)*A(k)[3,4], AR3->A(k)[3,5], ARP=4
      apac              ;Acc=Acc+Preg
```

```
        sach    *+,AR2                  ;store in x_1(3), AR4->x_1(4), ARP=2

        lar     AR2,#_x+1       ;load in AR2 the address of x(2)
        lar     AR3,#diff_f+16          ;load in AR3 the address of A(k)[4,2]
        lt      *+,AR3                  ;Treg=x(2), AR2->x(3), ARP=3
        mpy      *+,AR2                 ;Preg=x(2)*A(k)[4,2], AR3-> A(k)[4,3], ARP=2
        ltp     *+,AR3                  ;Acc=Preg, Treg=x(3), AR2->x(4), ARP=3
        mpy      *+,AR2                 ;Preg=x(3)*A(k)[4,3], AR3->A(k)[4,4], ARP=2
        lta     *+,AR3                  ;Acc=Acc+Preg, Treg=x(4), AR2->x(5), ARP=3
        mpy      *+,AR4                 ;Preg=x(4)*A(k)[4,4], AR3->A(k)[4,5], ARP=4
        apac                    ;Acc=Acc+Preg
        sach    *+,AR2                  ;store in x_1(4), AR4->x_1(5), ARP=2

        lar     AR2,#_x+4               ;load in AR2 the address of x(5)
        lar     AR3,#diff_f+24          ;load in AR3 the address of A(k)[5,5]
        lt      *,AR3           ;Treg=x(5), ARP=3
        mpy      *,AR4                  ;Preg=x(5)*A(k)[5,5]
        pac                     ;Acc=Preg
        sach    *,AR2                   ;store in A(k)[5,5]
        spm      0
        clrc    ovm


;P_1= prediction error covariance matrix = P(k/k-1)
;P   = filter error covariance matrix = P(k-1/k-1)
;P_1=diff_f*P*diff_f' +Q
      mmtfraG P, diff_f, RSLT, 5,5,5 ;RSLT=P*diff_f'
      mmfraG   diff_f, RSLT, P_1, 5,5,5  ;P_1=diff_f*P*diff_f'
      madiG    P_1,Q,5                ;P_1= diff_f*P*diff_f'+Q

;Kalman gain matrix calculation
;K= P_1 * C' * inv(C*P_1*C' +R);
      mmtfraG P_1, C, RSLT, 5,5,2       ;RSLT=P_1*C'
      mmfraG   C, RSLT, _K, 2,5,2               ;_K is a temp matrix, _K=C*P_1*C'
      madiG    _K, R, 2               ;_K=_K+R
      madjG    _K, P                          ;P is a temp matrix, P=adj(_K)
      mmfraG   RSLT, P, _K, 5, 2, 2     ;_K=RSLT*P=P_1*C'*adj(C*P_1*C'+R)
      detG     P, RSLT                        ;RSLT=det(P)
      mdsclG   _K, RSLT, _K, 5, 2, RSLT+1  ;_K=1/det(P)* P_1*C'*adj(C*P_1*C'+R)

;state vector estimation
;x_1= state vector prediction = x(k/k-1)
;x   = state vector estimation
;x = x_1 + K*(y-C*x_1)
      msubG     y, Cx_1, P, 2,1         ;P is a temp matrix, P= y-C*x_1
      mmfraG    _K, P, RSLT, 5,2,1      ;RSLT=_K*P=_K*(y-C*x_1)
      maddG     _x_1, RSLT, _x, 5,1             ;_x= x_1 + K*(y-C*x_1)

;Filter error covariance matrix
;P=P_1-K*C*P_1;
      mmfraG     _K, C, RSLT, 5, 2, 5         ;RSLT=_K*C
      mmfraG    RSLT, P_1, P , 5, 5, 5   ;P=_K*C*P_1
      msubG     P_1,P,P,5,5              ;P=P_1-P

      ldp       #kal_fil
      lacc      _x                      ;notice that _x (estimated state) is in 1.15 format
      ldp       #iSal_est
*     sfr
*     sfr
      sacl      iSal_est,1              ;iSalfa estimated*2

      ldp       #kal_fil
      lacc      _x+1
```

```
        ldp     #iSbe_est
*       sfr
*       sfr
        sacl    iSbe_est,1                      ;iSbeta estimated*2


        ldp     #kal_fil
        lacc    _x+2            ;_x+2 is the rotor flux alpha comp address
        ldp     #psiRal_est
        sacl    psiRal_est              ;f 1.15, used for the 1.15 division
*       sfr
*       sfr
*       sfr
*       sacl    psiRal_est              ;psiRalfa estimated, 4.12 format

        ldp     #kal_fil
        lacc    _x+3            ;_x+3 is the rotor flux beta comp address
        ldp     #psiRbe_est
        sacl    psiRbe_est
*       sfr
*       sfr
*       sfr
*       sacl    psiRbe_est              ;psiRbeta estimated, 4.12 format

        ldp     #kal_fil
        lacc    _x+4            ;_x+4 is the rotor speed variable address
        ldp     #n_est
*       sfr
*       sfr
        sacl    n_est,1                 ;n_est estimated*2
        ret
*** END Extended Kalman Filter **************
```

```
;=============================================================
; Filename:     pid.asm
;
; Module names:     pid_reg_id, pid_reg_iq
;
; Initialization routines: pid_reg_id_init, pid_reg_iq_init
;
;
; Description:  PI current regulator with integral correction
;               for d and q axes
;
;                 |~~~~~~~~~~~~|
; i_fdb     o------>|            |
; i_ref     o------>|  pid_reg   |----->o  u_out
; u_int     o------>|            |
; Kp              |              |
; Ki        o------>|____________|
; Kc
;
; Target dependency: 'c2xx core only
;=============================================================================
; History:
;-----------------------------------------------------------------------------
; 3-11-2003
;=============================================================
```

```
************************************************************
* D-Axis PI Current Regulator
************************************************************
;-----------------------------------------------------------
; Reference/Prototype
;-----------------------------------------------------------
;       .ref  pid_reg_id,pid_reg_id_init    ; function call
;       .ref  id_fdb,id_ref,Kp_d,Ki_d,Kc_d; Inputs
;       .ref  ud_int                        ; Input
;       .ref  ud_out                        ; Outputs

;-----------------------------------------------------------
; Global Definitions
;-----------------------------------------------------------
        .def  pid_reg_id,pid_reg_id_init    ; function call
        .def  id_fdb,id_ref,Kp_d,Ki_d,Kc_d; Inputs
        .def  ud_int                        ; Input
        .def  ud_out                        ; Outputs

;-----------------------------------------------------------
; Variable Definitions
;-----------------------------------------------------------
id_fdb    .usect "pid",1        ; current feedback
id_ref    .usect "pid",1        ; current reference
ud_out    .usect "pid",1        ; control voltage output

ud_int        .usect "pid",1        ; error integral
uintlo_d  .usect "pid",1

Kp_d          .usect "pid",1        ; proportional gain
Ki_d      .usect "pid",1        ; integral gain
Kc_d          .usect "pid",1        ; integral correction gain

id_error  .usect "pid",1        ; current error
uprsat_d  .usect "pid",1        ; control voltage prio saturation
saterr_d  .usect "pid",1        ; saturation error
tmp_d         .usect "pid",1        ; temp scrach

;-----------------------------------------------------------
; Default parameters
; Parameter spreadsheet: pid.xls
;-----------------------------------------------------------
Kp_d_         .set  783 ;      388      ; Q11, proportional gain 0.189d
Ki_d_         .set  5133;   2542; 1271  ; Q25, integral gain 0.37878*0.0002d
Kc_d_         .set  32767                    ; Q14, saturation correction gain 2d

Umax_d_ .set  07000h        ; maximum U(0.875PU)
Umin_d_ .set  08FFFh        ; minimum U(-0.875PU)

;-----------------------------------------------------------
; Initialization
;-----------------------------------------------------------
pid_reg_id_init
        ldp     #Kp_d                   ;
        SPLK  #Kp_d_,Kp_d               ;
        SPLK   #Ki_d_,Ki_d              ;
        SPLK   #Kc_d_,Kc_d              ;
        SPLK  #0,ud_int             ; zero integral term
        SPLK  #0,uintlo_d
    RET

;-----------------------------------------------------------
; Routine
```

```
;-------------------------------------------------------------
pid_reg_id
     setc  SXM                ; Allow sign extension
     setc  OVM                     ; Set overflow protection mode

     ldp   #id_ref                 ;
     LACC     id_ref,16       ; Use ACCH for OV protection
     SUB    id_fdb,16              ;
     SACH   id_error               ; Q15 id_ref - id_fdb

     lacl  uintlo_d        ;
     add   ud_int,16          ; 32-bit Q30
     spm   2                       ; product l/s 4 for accumulation
     LT    id_error           ;
     mpy   Kp_d                    ; Q15*Q11 -> 32-bit Q26
     apac                     ; 32-bit Q30 uint + id_error*Kp_d
     SACH     uprsat_d             ; save as Q14 uprsat_d
     sacl  tmp_d                   ;
     adds  tmp_d                   ;
     add   uprsat_d,16                  ; Q30 -> Q31 with OV protection
     sach  tmp_d                   ; save to tmp_d as Q15

     lacc  tmp_d                   ;
     sub   #Umin_d_                     ;
     bcnd  U_gmind,GEQ             ; Continue if tmp_d>=U_min
     lacc  #Umin_d_                ; otherwise, saturate
     B     Nextd
U_gmind
     lacc  tmp_d                   ;
     sub   #Umax_d_                ;
     BCND     U_lmaxd,LEQ               ; Continue if tmp_d<=U_max
     lacc  #Umax_d_                ; otherwise, saturate
     b     Nextd

U_lmaxd
     lacc  tmp_d                   ;
Nextd
     sacl  ud_out             ;

Int_termd
     lacc  ud_out,15          ; Use ACCH for OV protection
     SUB uprsat_d,16                ;
     sach  saterr_d           ; save as Q14 saterr_d = ud_out-uprsat_d

     lt    id_error           ;
     mpy   Ki_d               ; Q15*Q25 -> Q40
     pac                           ; Q40 -> Q44
     sach  tmp_d                   ;
     lacc  tmp_d                   ; Q44 -> Q28 (r/s 16 bits)

     LT    saterr_d                ;
     MPYKc_d                       ; Q14*Q14 -> Q28 saterr_d * Kc_d
     APAC                          ; Q28 Ki_d*id_error + Kc_d*saterr_d

     norm      *                   ;
     norm      *                   ; Q28 -> Q30 (with OV protection)

     ADDS      uintlo_d
     ADD       ud_int,16                ; uint + saterr_d*Kc_d + id_error*Ki_d
     SACH    ud_int
     SACL      uintlo_d            ; save as 32-bit Q30

     RET
```

```
***END D-Axis PI Current Regulator


************************************************************
* Q-Axis PI Current Regulator
************************************************************
;-----------------------------------------------------------
; Reference/Prototype
;-----------------------------------------------------------
;     .ref  pid_reg_iq,pid_reg_iq_init     ; function call
;     .ref  iq_fdb,iq_ref,Kp_q,Ki_q,Kc_q; Inputs
;     .ref  uq_int                         ; Input
;     .ref  uq_out                         ; Outputs

;-----------------------------------------------------------
; Global Definitions
;-----------------------------------------------------------
      .def  pid_reg_iq,pid_reg_iq_init     ; function call
      .def  iq_fdb,iq_ref,Kp_q,Ki_q,Kc_q; Inputs
      .def  uq_int                         ; Input
      .def  uq_out                         ; Outputs

;-----------------------------------------------------------
; Variable Definitions
;-----------------------------------------------------------
iq_fdb    .usect "pid",1          ; current feedback
iq_ref    .usect "pid",1          ; current reference
uq_out    .usect "pid",1          ; control voltage output

uq_int          .usect "pid",1          ; error integral
uintlo_q  .usect "pid",1

Kp_q            .usect "pid",1          ; proportional gain
Ki_q      .usect "pid",1          ; integral gain
Kc_q            .usect "pid",1          ; integral correction gain

iq_error  .usect "pid",1          ; current error
uprsat_q  .usect "pid",1          ; control voltage prio saturation
saterr_q  .usect "pid",1          ; saturation error
tmp_q           .usect "pid",1          ; temp scrach

;-----------------------------------------------------------
; Default parameters
; Parameter spreadsheet: pid.xls
;-----------------------------------------------------------
Kp_q_           .set  1566;    776        ; Q11, proportional gain   0.3789d
Ki_q_           .set  7700;    3814       ;1907 ; Q25, integral gain   0.56833*0.0002d
Kc_q_           .set  24576               ; Q14, saturation correction gain 1.5d

Umax_q_ .set  07000h          ; maximum U
Umin_q_ .set  08FFFh          ; minimum U

;-----------------------------------------------------------
; Initialization
;-----------------------------------------------------------
pid_reg_iq_init
     ldp  #Kp_q                    ;
     SPLK     #Kp_q_,Kp_q               ;
     SPLK   #Ki_q_,Ki_q           ;
     SPLK   #Kc_q_,Kc_q           ;
     SPLK     #0,uq_int            ; zero integral term
     SPLK     #0,uintlo_q
```

```
    RET

;-----------------------------------------------------------
; Routine
;-----------------------------------------------------------
pid_reg_iq
    setc SXM            ; Allow sign extension
    setc OVM                ; Set overflow protection mode

    ldp  #iq_ref            ;
    LACC   iq_ref,16    ; Use ACCH for OV protection
    SUB    iq_fdb,16        ;
    SACH   iq_error         ; Q15 iq_ref - iq_fdb

    lacl uintlo_q       ;
    add  uq_int,16      ; 32-bit Q30
    spm 2                   ; product l/s 4 before accumulation
    LT   iq_error       ;
    mpy Kp_q                    ; Q15*Q11 -> 32-bit Q26
    apac                    ; 32-bit Q30 uint + iq_error*Kp_q
    SACH    uprsat_q        ; save as Q14 uprsat_q
    sacl tmp_q              ;
    adds tmp_q              ;
    add  uprsat_q,16            ; Q30 -> Q31 with OV protection
    sach tmp_q              ; save to tmp_q as Q15

    lacc tmp_q              ;
    sub  #Umin_q_               ;
    bcnd U_gminq,GEQ        ; Continue if tmp_q>=U_min
    lacc #Umin_q_           ; otherwise, saturate
    B    Nextq
U_gminq
    lacc tmp_q              ;
    sub  #Umax_q_           ;
    BCND    U_lmaxq,LEQ         ; Continue if tmp_q<=U_max
    lacc #Umax_q_           ; otherwise, saturate
    b    Nextq

U_lmaxq
    lacc tmp_q              ;
Nextq
    sacl uq_out             ;

Int_termq
    lacc uq_out,15      ; Use ACCH for OV protection
    SUB uprsat_q,16             ;
    sach saterr_q       ; save as Q14 saterr_q = uq_out-uprsat_q

    lt   iq_error       ;
    mpy Ki_q                ; Q15*Q26 -> Q40
    pac                     ; Q40 -> Q44
    sach tmp_q              ;
    lacc tmp_q              ; Q44 -> Q28 (r/s 16 bits)

    LT   saterr_q               ;
    MPYKc_q                     ; Q14*Q14 -> Q28 saterr_q * Kc_q
    APAC                        ; Q28 Ki_q*iq_error + Kc_q*saterr_q

    norm    *               ;
    norm    *               ; Q28 -> Q30 (with OV protection)

    ADDS    uintlo_q
    ADD     uq_int,16           ; uint + saterr_q*Kc_q + iq_error*Ki_q
```

```
    SACH    uq_int
    SACL    uintlo_q            ; save as 32-bit Q30

    RET

***END Q-Axis PI Current Regulator
```

```
;=============================================================
; Filename:     pid_spd.asm
;=============================================================================
; History:
;-----------------------------------------------------------------------------
; 3-11-2003
;=============================================================================
************************************************************
* Speed PI Regulator
************************************************************
;           .ref  pid_reg_spd,pid_reg_spd_init ; function call
;           .ref  spd_fdb,spd_ref          ; Inputs
;           .ref  spd_out                  ; Outputs
;-----------------------------------------------------------
; Global Definitions
;-----------------------------------------------------------
            .def  pid_reg_spd,pid_reg_spd_init ; function call
            .def  spd_fdb,spd_ref          ; Inputs
            .def  spd_out                  ; Outputs
;-----------------------------------------------------------
; Variable Definitions
;-----------------------------------------------------------

spd_ref    .usect "pid_spd",1
spd_fdb    .usect "pid_spd",1
spd_out    .usect "pid_spd",1


;other variables :
* PI regulators variable
upi         .usect "pid_spd",1    ;PI regulators (current and speed)          ;output
elpi  .usect "pid_spd",1    ;PI regulators (current and speed)          ;limitation error
Kcorn       .usect "pid_spd",1
Ki          .usect "pid_spd",1
Kcor.usect "pid_spd",1
Kpin.usect "pid_spd",1
Kin         .usect "pid_spd",1
Kpi         .usect "pid_spd",1
epin .usect "pid_spd",1    ;speed error (used in speed                 ;regulator)
xin         .usect "pid_spd",1    ;speed regulator integral                   ;component
epiq .usect "pid_spd",1    ;q-axis current regulator error
epid .usect "pid_spd",1    ;d-axis current regulator error
xiq         .usect "pid_spd",1    ;q-axis current regulator integral          ;component
xid         .usect "pid_spd",1    ;d-axis current regulator integral          ;component

*** vSqref and VdSr limitations
Vmin        .set  0ec00h    ;4.12 format=-1.25 pu
```

```
Vmax    .set  1400h    ;4.12 format=1.25 pu

*** iSqref limitations
Isqrefmin .set  -1310     ;4.12 format=-0.8 pu, I_nom=10A
Isqrefmax .set  1310;4.12 format=0.8 pu, I_nom=10A
vSqref    .usect "pid_spd",1
iSqref    .usect "pid_spd",1
iSd       .usect "pid_spd",1
iSq       .usect "pid_spd",1
n         .usect "pid_spd",1
n_ref     .usect "pid_spd",1
vSdref    .usect "pid_spd",1
iSdref    .usect "pid_spd",1


pid_reg_spd_init
      zac
      ldp       #spd_ref
      sacl  xin
      sacl  upi
      sacl  elpi
      sacl      spd_out
      sacl      upi
      sacl      elpi
      sacl      epin
      sacl      epiq
      sacl      epid
      sacl      xiq
      sacl      xid
      sacl      vSqref
      sacl      iSqref
      sacl      iSd
      sacl      iSq
      sacl      n
      sacl      n_ref
      sacl      vSdref
      sacl      iSdref

*** PI speed regulators parameters
      splk  #35h,Kin ;    0.012939 Q12?
      splk  #482bh,Kpin; 4.5105
      splk  #0bh,Kcorn ; 0.0026855
      ret

pid_reg_spd
;Interface avec les nouvelles variables
      ldp       #spd_fdb
      lacc  spd_fdb ;Q15, 6000rpm as pu
      sfr
      ldp       #n
      sacl  n           ;Q12, 1500rpm as pu

      ldp       #spd_ref  ;Q15 ref with 1pu =1500rpm
      lacc  spd_ref
      sfr
      sacl  n_ref           ;Q12 with 1500rpm as pu
**************************************************
* Speed regulator with integral component correction
**************************************************
      ldp       #n_ref
      lacc  n_ref
      sub       n
      sacl  epin ;epin=n_ref-n, 4.12 format
```

```
        lacc  xin,12
        lt          epin
        mpy         Kpin
        apac
        sach upi,4;upi=xin+epin*Kpin, 4.12 format
                    ;here we start to saturate
        bit         upi,0
        bcnd upimagzeros,NTC    ;If value >0 we branch
        lacc  #Isqrefmin        ;negative saturation
        sub         upi
        bcnd neg_sat,GT         ;if upi<ISqrefmin then branch to
                    ;saturate
        lacc  upi   ;value of upi is valid
        b     limiters
neg_sat
        lacc  #Isqrefmin        ;set acc to -ve saturated value
        b     limiters

upimagzeros     ;Value is positive
        lacc  #Isqrefmax        ;positive saturation
        sub   upi
        bcnd pos_sat,LT         ;if upi>ISqrefmax then branch to                ;saturate
        lacc  upi   ;value of upi valid
        b     limiters
pos_sat
        lacc  #Isqrefmax        ;set acc to +ve saturated value

limiters
        sacl  iSqref      ;Store the acc as reference value
;Interface avec la nouvelle variable de sortie
        setc  OVM
        sfl
        sfl
        sfl
        ldp         #spd_out
        sacl  spd_out
        sub   #7fffh
        clrc  OVM
        ldp         #iSqref
        lacc  iSqref
;
        sub   upi
        sacl  elpi  ;elpi=iSqref-upi, 4.12 format
        lt    elpi  ;if there is no saturation elpi=0
        mpy Kcorn
        pac
        lt    epin
        mpy Kin
        apac
        add   xin,12
        sach xin,4;xin=xin+epin*Kin+elpi*Kcorn, 4.12                    ;format
**************************************************************
* END Speed regulator with integral component correction
**************************************************************
        ret



;=============================================================================
; Filename:      pwm_drv.asm
;
; Module Name:      FC_PWM_DRV
;
```

```
; Initialization Routine: FC_PWM_DRV_INIT
;
; Description:  This module uses the duty ratio information and calculates
;               the compare values for generating PWM outputs. The compare
;               values are used in the full compare unit in 24x/24xx event
;               manager(EV). This also allows PWM period modulation.
;
;                        |~~~~~~~~~~~~~~~~~|
; Mfunc_c1      o------>|                   |----->o   CMPR1 (EV register)
; Mfunc_c2      o------>|   FC_PWM_DRV      |----->o   CMPR2 (EV register)
; Mfunc_c3      o------>|                   |----->o   CMPR3 (EV register)
; Mfunc_po------>|                        |----->o   T1PER (EV register)
; n_period   o------>|_____________________|
;
; Target:       x2407 Event Manager Timer1 & F Compares
; Other:        Dependent on PWM period selection
;
;===============================================================================
; History:
;-------------------------------------------------------------------------------
; 3-11-2003
;===============================================================================

;---------------------
; Reference/Prototype
;---------------------
;           .ref FC_PWM_DRV,FC_PWM_DRV_INIT         ;function calls
;           .ref Mfunc_c1,Mfunc_c2,Mfunc_c3,Mfunc_p ;Inputs
;           .ref n_period                   ;Input

;---------------------------------
; Define Related Peripherals
;---------------------------------
            .include "x24x_app.h"

;-------------------------
; Default PWM Period
;-------------------------
*PWM_PERIOD     .set  50        ; PWM period in uS (20KHz) ,different from foc1.asm?
PWM_PERIOD      .set  200       ; PWM period in uS (5Khz)
;-------------------------
; Global Definitions
;-------------------------
            .def FC_PWM_DRV,FC_PWM_DRV_INIT         ;function calls
            .def Mfunc_c1,Mfunc_c2,Mfunc_c3,Mfunc_p ;Inputs
            .def n_period                   ;Input

;------------
; Variables
;------------
Mfunc_c1.usect "pwm_drv",1  ; Phase 1 mod function Q15
Mfunc_c2.usect "pwm_drv",1      ; Phase 2 mod function Q15
Mfunc_c3.usect "pwm_drv",1      ; Phase 3 mod function Q15
Mfunc_p       .usect "pwm_drv",1 ; Period mod function Q15
n_period  .usect "pwm_drv",1    ; Norminal period/compare value
m_period .usect "pwm_drv",1     ; Modulated period

;------------------------------
; Configuration parameters
;------------------------------

            .if x2407
T1PER_  .set  PWM_PERIOD*15 ; *1000nS/(2*33nS)
```

```
T1CON_ .set  1000100001000000b      ; Symmetric PWM
DBTCON_     .set  09E8h             ; D/B = 1.18uS @ 33nS clk
ACTR_       .set  011001100110b     ; 1/3/5 Active Hi, 2/4/6 Active Lo
COMCON_     .set  1000001000000000b     ; Compare Cntl
            .endif
;---------------------------------------------------------------------
; Initialization
;---------------------------------------------------------------------
FC_PWM_DRV_INIT
            LDP #T1PER>>7
            SPLK    #T1PER_,T1PER
            SPLK    #T1CON_,T1CON
            SPLK    #DBTCON_,DBTCON
            SPLK    #ACTR_,ACTR
            SPLK    #COMCON_,COMCON
            .if x2407
            ldp  #OCRA>>7               ; Configure 6-pwm pins
            LACC    OCRA
            OR  #0000111111000000b
            SACL    OCRA
            .endif
            ldp  #n_period
            SPLK    #T1PER_,n_period
            SPLK    #7FFFh,Mfunc_p
            RET

;---------------------------------------------------------------------
; Driver Routine
;---------------------------------------------------------------------
FC_PWM_DRV:
            ldp  #Mfunc_p; modulate period
            LT   Mfunc_p
            MPY     n_period  ; Mfunc_p*n_period/2
            PAC           ;
            add  n_period,15    ; offset by n_period/2
            SACH    m_period ; save for later reference
            ldp  #T1PER>>7    ;
            sach T1PER          ; save

            ldp  #Mfunc_c1     ; Modulate channel one
            LT   Mfunc_c1
            MPY     m_period ; Mfunc_c1 x m_period/2
            PAC           ;
            add  m_period,15    ; offset by m_period/2
            ldp  #CMPR1>>7
            SACH    CMPR1         ; save

            ldp  #Mfunc_c2     ; Modulate channel two
            LT   Mfunc_c2
            MPY     m_period ; Mfunc_c2 x m_period/2
            PAC           ;
            add  m_period,15    ; offset by m_period/2
            ldp  #CMPR2>>7
            SACH    CMPR2         ; save

            ldp  #Mfunc_c3     ; modulate channel three
            LT   Mfunc_c3
            MPY     m_period ; Mfunc_c3 x m_period/2
            PAC           ;
            add  m_period,15    ; offset by m_period/2
            ldp  #CMPR3>>7
            SACH    CMPR3         ; save
            RET
```

```
;=====================================================================
; File Name:    rampgen.asm
;
; Module Name:      RAMP_GEN
;
; Initialization Routine: RAMP_GEN_INIT
;
;
; Description:  This module generates ramp output of adjustable gain,
;           frequency and dc offset.
;                                   |~~~~~~~~~~~~~~~~~~|
;         rmp_gain      o------>|                  |
;         rmp_offset o------>|     RAMP_GEN     |----->o   rmp_out
;         rmp_freq      o------>|                  |
;                                   |__________________|
;
;Target dependency: c2xx core only
;=====================================================================
; History:
;---------------------------------------------------------------------------
; 3-11-2003
;=====================================================================
;(To use this Module, copy this section to main system file)
;           .ref  RAMP_GEN, RAMP_GEN_INIT             ;function call
;           .ref  rmp_gain, rmp_offset, rmp_freq      ;Inputs
;           .ref  step_angle_max                      ;Input
;           .ref  rmp_out                     ;Outputs
;=====================================================================
           .def  RAMP_GEN, RAMP_GEN_INIT             ;function call
           .def  rmp_gain, rmp_offset, rmp_freq      ;Inputs
           .def  step_angle_max                      ;Input
           .def  rmp_out                     ;Output
;=====================================================================

STEP_ANGLE_RG_MAX     .set  1000;Corresponds to 305.2Hz for fs=20kHz
                           ; or 152.6Hz for fs=10kHz, or 76.3Hz for fs=5KHz
                           ;See related doc for details

alpha_rg  .usect "rampgen",1
step_angle_rg  .usect "rampgen",1
step_angle_max     .usect "rampgen",1
rmp_gain .usect "rampgen",1
rmp_offset     .usect "rampgen",1
rmp_freq  .usect "rampgen",1
rmp_out        .usect "rampgen",1
rmp_out_abs    .usect "rampgen",1



RAMP_GEN_INIT:
          LDP #alpha_rg
          SPLK     #0, alpha_rg
          SPLK     #STEP_ANGLE_RG_MAX, step_angle_max

          SPLK     #3FFFh, rmp_gain
          SPLK     #3FFFh, rmp_offset
          SPLK     #3FFFh, rmp_freq
          RET

RAMP_GEN:
          SPM   0
          LDP #rmp_freq
          LT   rmp_freq        ;Normalized frequency rmp_freq
```

```
                        ;is in Q15
        MPY     step_angle_max          ;Q15 x Q0
        PAC             ;Q0 x Q15 = Q15 (32bit)
        SACH    step_angle_rg,1         ;Q0

        LACC    alpha_rg        ;Q0
        ADD     step_angle_rg       ;Q0+Q0
        SACL    alpha_rg        ;Q0

    ;scale the output with gain(user specified)
        LT  alpha_rg        ;Q0
        MPY     rmp_gain        ;Q0 x Q15
        PAC             ;P = rmp_gain * alpha_rg
        SACH    rmp_out_abs,1       ;Q0
        LACC    rmp_out_abs         ;Q0
        SACL    rmp_out             ;Q15**
**
;In the last two instructions the variables rmp_out_abs and rmp_out contain
;the same value which is the result of the preceeding multiply operation.
;Although they have the same value, by representing rmp_out with a
;different Q format(Q15) than rmp_out_abs(Q0), we have essentially performed
;an implicit normalization(division) operation. The normalized ramp output,
;rmp_out(in Q15), and the absolute ramp output, rmp_out_abs (in Q0), are
;related by,
;rmp_out = rmp_out_abs/7FFFh.
;The output of this module(rmp_out) is normalized (expressed in Q15) since
;in many other s/w modules, where this is used as input, require the input
;be provided in Q15 format.


    ;add offset value
        LACC    rmp_out             ;Q15
        ADD     rmp_offset          ;Q15+Q15
        SACL    rmp_out             ;Q15
        RET
```

```
;=============================================================================
; File:          SINTB360.asm
; Name:   SINTAB_360
; Type:          Table
; Description:   This is a program space lookup table for
;                Sin/Cos functions.
;
;
; Target:        C2xx core
;
; Last Update:  03-11-2003
;=============================================================================
;(To use this Module, copy this section to the calling file)
;          .ref  SINTAB_360
;=============================================================================
;Module definitions for external reference.
          .def  SINTAB_360
;=============================================================================
;-----------------------------------------------------
; Sine look-up table
; No. Entries:  256
; Angle Range: 360 deg
; Number format:    Q15 with range -1 < N < +1
;-----------------------------------------------------
                    ;SINVAL          ;     Index       Angle     Sin(Angle)
SINTAB_360  .word          0          ;         0           0  0.0000
            .word        804          ;         1        1.41     0.0245
            .word       1608          ;         2        2.81     0.0491
            .word       2410          ;         3        4.22     0.0736
            .word       3212          ;         4        5.63     0.0980
            .word       4011          ;         5        7.03     0.1224
            .word       4808          ;         6        8.44     0.1467
            .word       5602          ;         7        9.84     0.1710
            .word       6393          ;         8       11.25     0.1951
            .word       7179          ;         9       12.66     0.2191
            .word       7962          ;        10       14.06     0.2430
            .word       8739          ;        11       15.47     0.2667
            .word       9512          ;        12       16.88     0.2903
            .word      10278          ;        13       18.28     0.3137
            .word      11039          ;        14       19.69     0.3369
            .word      11793          ;        15       21.09     0.3599
            .word      12539          ;        16       22.50     0.3827
            .word      13279          ;        17       23.91     0.4052
            .word      14010          ;        18       25.31     0.4276
            .word      14732          ;        19       26.72     0.4496
            .word      15446          ;        20       28.13     0.4714
            .word      16151          ;        21       29.53     0.4929
            .word      16846          ;        22       30.94     0.5141
            .word      17530          ;        23       32.34     0.5350
            .word      18204          ;        24       33.75     0.5556
            .word      18868          ;        25       35.16     0.5758
            .word      19519          ;        26       36.56     0.5957
            .word      20159          ;        27       37.97     0.6152
            .word      20787          ;        28       39.38     0.6344
            .word      21403          ;        29       40.78     0.6532
            .word      22005          ;        30       42.19     0.6716
            .word      22594          ;        31       43.59     0.6895
            .word      23170          ;        32       45.00     0.7071
            .word      23731          ;        33       46.41     0.7242
            .word      24279          ;        34       47.81     0.7410
            .word      24811          ;        35       49.22     0.7572
            .word      25329          ;        36       50.63     0.7730
            .word      25832          ;        37       52.03     0.7883
```

```
.word   26319   ;   38   53.44   0.8032
.word   26790   ;   39   54.84   0.8176
.word   27245   ;   40   56.25   0.8315
.word   27683   ;   41   57.66   0.8449
.word   28105   ;   42   59.06   0.8577
.word   28510   ;   43   60.47   0.8701
.word   28898   ;   44   61.88   0.8819
.word   29268   ;   45   63.28   0.8932
.word   29621   ;   46   64.69   0.9040
.word   29956   ;   47   66.09   0.9142
.word   30273   ;   48   67.50   0.9239
.word   30571   ;   49   68.91   0.9330
.word   30852   ;   50   70.31   0.9415
.word   31113   ;   51   71.72   0.9495
.word   31356   ;   52   73.13   0.9569
.word   31580   ;   53   74.53   0.9638
.word   31785   ;   54   75.94   0.9700
.word   31971   ;   55   77.34   0.9757
.word   32137   ;   56   78.75   0.9808
.word   32285   ;   57   80.16   0.9853
.word   32412   ;   58   81.56   0.9892
.word   32521   ;   59   82.97   0.9925
.word   32609   ;   60   84.38   0.9952
.word   32678   ;   61   85.78   0.9973
.word   32728   ;   62   87.19   0.9988
.word   32757   ;   63   88.59   0.9997
.word   32767   ;   64   90.00   1.0000
.word   32757   ;   65   91.41   0.9997
.word   32728   ;   66   92.81   0.9988
.word   32678   ;   67   94.22   0.9973
.word   32609   ;   68   95.63   0.9952
.word   32521   ;   69   97.03   0.9925
.word   32412   ;   70   98.44   0.9892
.word   32285   ;   71   99.84   0.9853
.word   32137   ;   72   101.25   0.9808
.word   31971   ;   73   102.66   0.9757
.word   31785   ;   74   104.06   0.9700
.word   31580   ;   75   105.47   0.9638
.word   31356   ;   76   106.88   0.9569
.word   31113   ;   77   108.28   0.9495
.word   30852   ;   78   109.69   0.9415
.word   30571   ;   79   111.09   0.9330
.word   30273   ;   80   112.50   0.9239
.word   29956   ;   81   113.91   0.9142
.word   29621   ;   82   115.31   0.9040
.word   29268   ;   83   116.72   0.8932
.word   28898   ;   84   118.13   0.8819
.word   28510   ;   85   119.53   0.8701
.word   28105   ;   86   120.94   0.8577
.word   27683   ;   87   122.34   0.8449
.word   27245   ;   88   123.75   0.8315
.word   26790   ;   89   125.16   0.8176
.word   26319   ;   90   126.56   0.8032
.word   25832   ;   91   127.97   0.7883
.word   25329   ;   92   129.38   0.7730
.word   24811   ;   93   130.78   0.7572
.word   24279   ;   94   132.19   0.7410
.word   23731   ;   95   133.59   0.7242
.word   23170   ;   96   135.00   0.7071
.word   22594   ;   97   136.41   0.6895
.word   22005   ;   98   137.81   0.6716
.word   21403   ;   99   139.22   0.6532
.word   20787   ;   100   140.63   0.6344
```

```
.word   20159   ;   101   142.03   0.6152
.word   19519   ;   102   143.44   0.5957
.word   18868   ;   103   144.84   0.5758
.word   18204   ;   104   146.25   0.5556
.word   17530   ;   105   147.66   0.5350
.word   16846   ;   106   149.06   0.5141
.word   16151   ;   107   150.47   0.4929
.word   15446   ;   108   151.88   0.4714
.word   14732   ;   109   153.28   0.4496
.word   14010   ;   110   154.69   0.4276
.word   13279   ;   111   156.09   0.4052
.word   12539   ;   112   157.50   0.3827
.word   11793   ;   113   158.91   0.3599
.word   11039   ;   114   160.31   0.3369
.word   10278   ;   115   161.72   0.3137
.word   9512    ;   116   163.13   0.2903
.word   8739    ;   117   164.53   0.2667
.word   7962    ;   118   165.94   0.2430
.word   7179    ;   119   167.34   0.2191
.word   6393    ;   120   168.75   0.1951
.word   5602    ;   121   170.16   0.1710
.word   4808    ;   122   171.56   0.1467
.word   4011    ;   123   172.97   0.1224
.word   3212    ;   124   174.38   0.0980
.word   2410    ;   125   175.78   0.0736
.word   1608    ;   126   177.19   0.0491
.word   804     ;   127   178.59   0.0245
.word   0       ;   128   180.00   0.0000
.word   64731   ;   129   181.41   -0.0245
.word   63927   ;   130   182.81   -0.0491
.word   63125   ;   131   184.22   -0.0736
.word   62323   ;   132   185.63   -0.0980
.word   61524   ;   133   187.03   -0.1224
.word   60727   ;   134   188.44   -0.1467
.word   59933   ;   135   189.84   -0.1710
.word   59142   ;   136   191.25   -0.1951
.word   58356   ;   137   192.66   -0.2191
.word   57573   ;   138   194.06   -0.2430
.word   56796   ;   139   195.47   -0.2667
.word   56023   ;   140   196.88   -0.2903
.word   55257   ;   141   198.28   -0.3137
.word   54496   ;   142   199.69   -0.3369
.word   53742   ;   143   201.09   -0.3599
.word   52996   ;   144   202.50   -0.3827
.word   52256   ;   145   203.91   -0.4052
.word   51525   ;   146   205.31   -0.4276
.word   50803   ;   147   206.72   -0.4496
.word   50089   ;   148   208.13   -0.4714
.word   49384   ;   149   209.53   -0.4929
.word   48689   ;   150   210.94   -0.5141
.word   48005   ;   151   212.34   -0.5350
.word   47331   ;   152   213.75   -0.5556
.word   46667   ;   153   215.16   -0.5758
.word   46016   ;   154   216.56   -0.5957
.word   45376   ;   155   217.97   -0.6152
.word   44748   ;   156   219.38   -0.6344
.word   44132   ;   157   220.78   -0.6532
.word   43530   ;   158   222.19   -0.6716
.word   42941   ;   159   223.59   -0.6895
.word   42365   ;   160   225.00   -0.7071
.word   41804   ;   161   226.41   -0.7242
.word   41256   ;   162   227.81   -0.7410
.word   40724   ;   163   229.22   -0.7572
```

```
.word   40206   ;   164   230.63   -0.7730
.word   39703   ;   165   232.03   -0.7883
.word   39216   ;   166   233.44   -0.8032
.word   38745   ;   167   234.84   -0.8176
.word   38290   ;   168   236.25   -0.8315
.word   37852   ;   169   237.66   -0.8449
.word   37430   ;   170   239.06   -0.8577
.word   37025   ;   171   240.47   -0.8701
.word   36637   ;   172   241.88   -0.8819
.word   36267   ;   173   243.28   -0.8932
.word   35914   ;   174   244.69   -0.9040
.word   35579   ;   175   246.09   -0.9142
.word   35262   ;   176   247.50   -0.9239
.word   34964   ;   177   248.91   -0.9330
.word   34683   ;   178   250.31   -0.9415
.word   34422   ;   179   251.72   -0.9495
.word   34179   ;   180   253.13   -0.9569
.word   33955   ;   181   254.53   -0.9638
.word   33750   ;   182   255.94   -0.9700
.word   33564   ;   183   257.34   -0.9757
.word   33398   ;   184   258.75   -0.9808
.word   33250   ;   185   260.16   -0.9853
.word   33123   ;   186   261.56   -0.9892
.word   33014   ;   187   262.97   -0.9925
.word   32926   ;   188   264.38   -0.9952
.word   32857   ;   189   265.78   -0.9973
.word   32807   ;   190   267.19   -0.9988
.word   32778   ;   191   268.59   -0.9997
.word   32768   ;   192   270.00   -1.0000
.word   32778   ;   193   271.41   -0.9997
.word   32807   ;   194   272.81   -0.9988
.word   32857   ;   195   274.22   -0.9973
.word   32926   ;   196   275.63   -0.9952
.word   33014   ;   197   277.03   -0.9925
.word   33123   ;   198   278.44   -0.9892
.word   33250   ;   199   279.84   -0.9853
.word   33398   ;   200   281.25   -0.9808
.word   33564   ;   201   282.66   -0.9757
.word   33750   ;   202   284.06   -0.9700
.word   33955   ;   203   285.47   -0.9638
.word   34179   ;   204   286.88   -0.9569
.word   34422   ;   205   288.28   -0.9495
.word   34683   ;   206   289.69   -0.9415
.word   34964   ;   207   291.09   -0.9330
.word   35262   ;   208   292.50   -0.9239
.word   35579   ;   209   293.91   -0.9142
.word   35914   ;   210   295.31   -0.9040
.word   36267   ;   211   296.72   -0.8932
.word   36637   ;   212   298.13   -0.8819
.word   37025   ;   213   299.53   -0.8701
.word   37430   ;   214   300.94   -0.8577
.word   37852   ;   215   302.34   -0.8449
.word   38290   ;   216   303.75   -0.8315
.word   38745   ;   217   305.16   -0.8176
.word   39216   ;   218   306.56   -0.8032
.word   39703   ;   219   307.97   -0.7883
.word   40206   ;   220   309.38   -0.7730
.word   40724   ;   221   310.78   -0.7572
.word   41256   ;   222   312.19   -0.7410
.word   41804   ;   223   313.59   -0.7242
.word   42365   ;   224   315.00   -0.7071
.word   42941   ;   225   316.41   -0.6895
.word   43530   ;   226   317.81   -0.6716
```

```
        .word   44132   ;       227     319.22  -0.6532
        .word   44748   ;       228     320.63  -0.6344
        .word   45376   ;       229     322.03  -0.6152
        .word   46016   ;       230     323.44  -0.5957
        .word   46667   ;       231     324.84  -0.5758
        .word   47331   ;       232     326.25  -0.5556
        .word   48005   ;       233     327.66  -0.5350
        .word   48689   ;       234     329.06  -0.5141
        .word   49384   ;       235     330.47  -0.4929
        .word   50089   ;       236     331.88  -0.4714
        .word   50803   ;       237     333.28  -0.4496
        .word   51525   ;       238     334.69  -0.4276
        .word   52256   ;       239     336.09  -0.4052
        .word   52996   ;       240     337.50  -0.3827
        .word   53742   ;       241     338.91  -0.3599
        .word   54496   ;       242     340.31  -0.3369
        .word   55257   ;       243     341.72  -0.3137
        .word   56023   ;       244     343.13  -0.2903
        .word   56796   ;       245     344.53  -0.2667
        .word   57573   ;       246     345.94  -0.2430
        .word   58356   ;       247     347.34  -0.2191
        .word   59142   ;       248     348.75  -0.1951
        .word   59933   ;       249     350.16  -0.1710
        .word   60727   ;       250     351.56  -0.1467
        .word   61524   ;       251     352.97  -0.1224
        .word   62323   ;       252     354.38  -0.0980
        .word   63125   ;       253     355.78  -0.0736
        .word   63927   ;       254     357.19  -0.0491
        .word   64731   ;       255     358.59  -0.0245
        .word   65535   ;       256     360.00   0.0000
```

```
;=======================================================================
; Filename:     svgen_dq.asm
;
; Module Name:      SVGEN_DQ
;
; Initialization Routine: SVGEN_DQ_INIT
;
;
; Description:  This module calculates the appropriate duty ratios needed
;           to generate a given stator reference voltage using space
;           vector PWM technique. The stator reference voltage is
;           described by it's (a,b) components, Ualfa and Ubeta.
;
;                   |~~~~~~~~~~~~~~~~|
; Ualfa     o---->|                  |----->o  Ta
;                   |   SVGEN_DQ       |----->o  Tb
; Ubeta     o---->|                  |----->o  Tc
;                   |__________________|
;
;=======================================================================
; History:
```

```
;-------------------------------------------------------------------------
; 3-11-2003
;=========================================================================

;-------------------------------------------------------------------------
; Reference/Prototype
;-------------------------------------------------------------------------
;       .ref  SVGEN_DQ,SVGEN_DQ_INIT            ;function call
;       .ref  Ualfa,Ubeta                       ;Inputs
;       .ref  Ta,Tb,Tc                          ;Outputs

;-------------------------------------------------------------------------
; Select Processor and Define Related Peripherals
;-------------------------------------------------------------------------
        .include "x24x_app.h"

;-------------------------------------------------------------------------
; Global Definitions
;-------------------------------------------------------------------------
        .def  SVGEN_DQ,SVGEN_DQ_INIT            ;function call
        .def  Ualfa,Ubeta                       ;Inputs
        .def  Ta,Tb,Tc                          ;Outputs

out_of_phase_           .set    0
;-------------------------------------------------------------------------
; Variables
;-------------------------------------------------------------------------
Ualfa       .usect "svgen_dq",1
Ubeta       .usect "svgen_dq",1
Va      .usect "svgen_dq",1
Vb      .usect "svgen_dq",1
Vc      .usect "svgen_dq",1
Ta      .usect "svgen_dq",1
Tb      .usect "svgen_dq",1
Tc      .usect "svgen_dq",1
sector      .usect "svgen_dq",1     ;SVPWM sector
t1      .usect "svgen_dq",1     ;SVPWM T1
t2      .usect "svgen_dq",1     ;SVPWM T2
half_sqrt3.usect "svgen_dq",1   ;SQRT(3) * 0.5

;Alias Variable declaration (to conserve .bss locations)
X       .set  Va
Y       .set  Vb
Z       .set  Vc
SR_ADDR     .set  sector
;-----------------------------------------------------

;=============================
SVGEN_DQ_INIT:
;=============================
        ldp  #half_sqrt3
        SPLK    #28378,half_sqrt3 ; Set constant sqrt(3)*0.5 in Q15 format
        RET

;=============================
SVGEN_DQ:
;=============================
    ;INV_CLARKE:
    ;-----------
        SPM 1               ; SPM set for Q15 multiplication
        SETC    SXM             ; Sign extension mode on

    ;Va = Ubeta
```

```
            ldp  #Ubeta
            LACC    Ubeta                ; ACC = Ubeta
            SACL    Va              ; Va = Ubeta

      ;Vb = (-Ubeta + sqrt(3) * Ualfa) / 2
            LT   Ualfa               ; TREG = Ualfa
            MPY     half_sqrt3       ; PREG = Ualfa * half_sqrt3
            PAC                          ; ACC high = Ualfa * half_sqrt3
            SUB Ubeta,15        ; ACC high = Ualfa * half_sqrt3 +Ubeta/2
            SACH    Vb                   ; Vb = Ualfa * half_sqrt3 +Ubeta/2

      ;Vc = (-Ubeta - sqrt(3) * Ualfa) / 2
            PAC                          ; ACC high = Ualfa * half_sqrt3
            NEG                          ; ACC high = - Ualfa * half_sqrt3
            SUB Ubeta,15        ; ACC high = - Ualfa * half_sqrt3 -Ubeta/2
            SACH    Vc                   ; Vc = - Ualfa * half_sqrt3 -Ubeta/2

;-------------------------------------------------------
; 60 degrees sector determination
; sector = r1 + 2*r2 + 4*r3
; r1=1 if Va>0
; r2=1 if Vb>0
; r3=1 if Vc>0
;-------------------------------------------------------
            SPLK    #0,sector
            LACC    Va
            BCND    vref1_neg,LEQ    ;If Va<0 do not set bit 1 of sector
            LACC    sector           ;
            OR  #1                ;
            SACL    sector           ;
vref1_neg
            LACC    Vb
            BCND    vref2_neg,LEQ    ;If Vb<0 do not set bit 2 of sector
            LACC    sector           ;
            OR  #2                ;
            SACL    sector           ;
vref2_neg
            LACC    Vc
            BCND    vref3_neg,LEQ    ;If Vc<0 do not set bit 3 of sector
            LACC    sector           ;
            OR  #4                ;
            SACL    sector           ;
vref3_neg

;-------------------------------------------------------
;X,Y,Z calculation:
;-------------------------------------------------------
XYZ_CALC:
      ;X = Ubeta
            LACC    Ubeta
            SACL    X

      ;Y = (0.5 * Ubeta) + (sqrt(3) * 0.5 * Ualfa)
            LT   Ualfa               ; TREG = Ualfa
            MPY     half_sqrt3       ; PREG = Ualfa * half_sqrt3
            PAC                          ; ACC high = Ualfa * half_sqrt3
            ADD     Ubeta,15         ; ACC high = Ualfa * half_sqrt3 + Ubeta/2
            SACH    Y                ; Y = Ualfa * half_sqrt3 + Ubeta/2

      ;Z = (0.5 * Ubeta) - (sqrt(3) * 0.5 * Ualfa)
            PAC                          ; ACC high = Ualfa * half_sqrt3
            NEG                          ; ACC high = - Ualfa * half_sqrt3
            ADD     Ubeta,15         ; ACC high = - Ualfa * half_sqrt3 + Ubeta/2
```

```
        SACH    Z                       ; Z = - Ualfa * half_sqrt3 + Ubeta/2

;---------------------------------------------------------------
;Sector calculations ("case statement")
;---------------------------------------------------------------
        LACC    #SECTOR_TBL
        ADD     sector
        TBLR    SR_ADDR
        LACC    SR_ADDR
        BACC

SECTOR_SR1:
;----------
    ;sector 1: t1=Z and t2=Y, (abc --> Tb, Ta, Tc)
        lacc Z
        sacl  t1
        lacc  Y
        sacl  t2

        lacc  #7FFFh        ;Load 1 (Q15)
        sub    t1
        sub    t2                   ;taon=(1-t1-t2)/2
        sfr
        sacl  Tb
        add    t1                   ;tbon=taon+t1
        sacl  Ta
        add    t2                   ;tcon=tbon+t2
        sacl  Tc
        B    SV_END

SECTOR_SR2:
;----------
    ;sector 2: t1=Y and t2=-X, (abc --> Ta, Tc, Tb)
        lacc Y
        sacl  t1
        lacc  X
        neg
        sacl  t2

        lacc  #7FFFh        ;Load 1 (Q15)
        sub    t1
        sub    t2                   ;taon=(1-t1-t2)/2
        sfr                       ;
        sacl  Ta
        add    t1                   ;tbon=taon+t1
        sacl  Tc
        add    t2                   ;tcon=tbon+t2
        sacl  Tb
        B    SV_END

SECTOR_SR3:
;----------
    ;sector 3: t1=-Z and t2=X, (abc --> Ta, Tb, Tc)
        lacc Z
        neg
        sacl  t1
        lacc  X
        sacl  t2

        lacc  #7FFFh        ;Load 1 (Q15)
        sub    t1
        sub    t2                   ;taon=(1-t1-t2)/2
        sfr                       ;
```

```
            sacl   Ta
            add    t1                    ;tbon=taon+t1
            sacl   Tb
            add    t2                    ;tcon=tbon+t2
            sacl   Tc
            B      SV_END

SECTOR_SR4:
;----------
      ;sector 4: t1=-X and t2=Z, (abc --> Tc, Tb, Ta)
            lacc  X
            neg
            sacl   t1
            lacc   Z
            sacl   t2

            lacc   #7FFFh        ;Load 1 (Q15)
            sub    t1
            sub    t2                    ;taon=(1-t1-t2)/2
            sfr                        ;
            sacl   Tc
            add    t1                    ;tbon=taon+t1
            sacl   Tb
            add    t2                    ;tcon=tbon+t2
            sacl   Ta
            B      SV_END

SECTOR_SR5:
;----------
      ;sector 5:  t1=X and t2=-Y, (abc --> Tb, Tc, Ta)
            lacc  X
            sacl   t1
            lacc   Y
            neg
            sacl   t2

            lacc   #7FFFh        ;Load 1 (Q15)
            sub    t1
            sub    t2                    ;taon=(1-t1-t2)/2
            sfr                        ;
            sacl   Tb
            add    t1                    ;tbon=taon+t1
            sacl   Tc
            add    t2                    ;tcon=tbon+t2
            sacl   Ta
            B      SV_END

SECTOR_SR6:
;----------
      ;sector 6:  t1=-Y and t2=-Z, (abc --> Tb, Tc, Ta)
            lacc  Y
            neg
            sacl   t1
            lacc   Z
            neg
            sacl   t2

            lacc   #7FFFh        ;Load 1 (Q15)
            sub    t1
            sub    t2                    ;taon=(1-t1-t2)/2
            sfr                        ;
            sacl   Tc
            add    t1                    ;tbon=taon+t1
```

```
            sacl   Ta
            add    t2                    ;tcon=tbon+t2
            sacl   Tb

SV_END:
      ;Multiply Ta by 2 & offset by 1/2
            LACC    Ta
            SUB #3FFFh
            SACL    Ta,1          ;mpy by 2

      ;Multiply Tb by 2 & offset by 1/2
            LACC    Tb
            SUB #3FFFh
            SACL    Tb,1          ;mpy by 2

      ;Multiply Tc by 2 & offset by 1/2
            LACC    Tc
            SUB #3FFFh
            SACL    Tc,1          ;mpy by 2

      .if (out_of_phase_)
            LACC    Ta
            NEG
            SACL    Ta
            LACC    Tb
            NEG
            SACL    Tb
            LACC    Tc
            NEG
            SACL    Tc
      .endif

DUMMY       SPM0                  ; SPM reset
            RET

;-------------------------------------------------------
;SVPWM Sector routine jump table - used with BACC inst.
;-------------------------------------------------------
SECTOR_TBL:
SR00          .word     DUMMY
SR0       .word     SECTOR_SR1
SR1       .word     SECTOR_SR2
SR2       .word     SECTOR_SR3
SR3       .word     SECTOR_SR4
SR4       .word     SECTOR_SR5
SR5       .word     SECTOR_SR6
```

```
;=============================================================================
; Module Name: S Y S _ I N I T
;
; File Name:    Sys_init.asm
;
; Description:   Initializes F24x/xx devices
;
;
;     AR1 is used as stack pointer
;
; Last Update: 3-11-2003
;=============================================================================
          .include  x24x_app.h
          .def  SYS_INIT
          .ref  GPR0


stack_size.set  20h
stack_start     .usect     "stack",stack_size

SYS_INIT:

;---target dependancy----------
     .if (x2407)
          POINT_PG0
          SETC     INTM              ;Disable interrupts
          SPLK     #0h, IMR      ;Mask all Ints
          SPLK     #0FFh, IFR        ;Clear all Int Flags

;Init PDP interrupt flag after reset
          LDP #PIRQR0>>7
          LACC     PIRQR0       ; Clear pending PDP flag
          AND      #0FFFEh
          SACL     PIRQR0
          LACC     PIRQR2       ; Clear pending PDP flag
          AND      #0FFFEh
          SACL     PIRQR2
          POINT_EV
          LACC     EVAIFRA           ; Clear PDPINTA flag
          OR  #0001h
          SACL     EVAIFRA
          LDP #EVBIFRA>>7
          LACC     EVBIFRA           ; Clear PDPINTB flag
          OR  #0001h
          SACL     EVBIFRA

          POINT_PG0
          CLRC     SXM               ;Clear Sign Extension Mode
          CLRC     OVM               ;Reset Overflow Mode
          CLRC     CNF          ;Config Block B0 to Data mem.
          SPM0
          LAR AR1, #stack_start          ;Init s/w stack pointer
          MAR      *,AR1

          POINT_B0
          SPLK     #00C0h, GPR0 ;Set 3 wait states for I/O space
          OUTGPR0, WSGR

          POINT_PF1

          .if (x4_PLL)
          SPLK     #0085h, SCSR1       ; x4 PLL, ADC en, EV1 en, clr Ill Addr flg
          .endif
```

```
        .if (x2_PLL)
        SPLK    #0285h, SCSR1       ; x2 PLL, ADC en, EV1 en, clr Ill Addr flg
        .endif

    ;Comment out if WD is to be active
        SPLK    #006Fh, WD_CNTL     ;Disable WD if VCCP=5V
        KICK_DOG
        RET
    .endif
;-----------------------------------
```

```
;=============================================================================
; File Name:    theta_est.asm
;
; Module Name:      theta_estimation_model
;
; Initialization Routine: theta_est_init
;
;
; Description:  Current model for field oriented control
;                       of an AC induction machine
;
;                 |~~~~~~~~~~~~~~~~~~|
;          psiRal ->|                  |--->sinTeta_est
;                   |   theta_est      |
;          psiRbe ->|                  |--->cosTeta_est
;                   |      model       |
;                   |__________________|
;
;
; Target:       x2407
;
;=============================================================================
; History:
;-----------------------------------------------------------------------------
; 3-18-2003
;-------------------------------------------------------------
; Reference/Prototype
;-------------------------------------------------------------

;           .ref  theta_est,theta_est_init      ; function call
;           .ref  psiRal,psiRbe                 ; Inputs
;           .ref  sinTeta_est,cosTeta_est       ; Outputs ;

;-------------------------------------------------------------
; Global Definitions
;-------------------------------------------------------------

    .def  theta_est,theta_est_init      ; function call
        .def  psiRal,psiRbe                 ; Inputs
        .def  sinTeta_est,cosTeta_est       ; Outputs ;
;-------------------------------------------------------------
        .ref    sqrt_tab    ; square root table
```

```
; Variable Definitions
;------------------------------------------------------------
;public variables
psiRal      .usect "theta_es",1
psiRbe      .usect "theta_es",1
sinTeta_est .usect "theta_es",1
cosTeta_est .usect "theta_es",1


;private variables
tmp             .usect "theta_es",1
tmp1            .usect "theta_es",1
Index           .usect "theta_es",1
psiR_est        .usect "theta_es",1
psiR_est1       .usect "theta_es",1
psiRal_est1 .usect "theta_es",1
psiRbe_est1 .usect "theta_es",1
;------------------------------------------------------------
; Default parameters
;------------------------------------------------------------
Ksqrt               .set   2751
Pole_pairs_number   .set    2


;------------------------------------------------------------
; Initialization
;------------------------------------------------------------
theta_est_init

CURRENT_MODEL_INIT

        ldp     #psiRal
        splk #0000h,sinTeta_est
        splk #7FFFh,cosTeta_est
        ret



*********************************************
* Rotor flux amplitude calculation
* psiR_est
*********************************************
theta_est
        ldp     #psiRal
        bldd    #psiRal,psiRal_est1
        bldd    #psiRbe,psiRbe_est1
        lacc    psiRal_est1
        sfr
        sfr                             ;change to 4.12 format
        sfr
        sacl    psiRal
        lacc    psiRbe_est1
        sfr
        sfr
        sfr
        sacl    psiRbe
        mar     *,AR5           ;ARP->AR5
        spm     2           ;automatic <<2 for 4.12 multiplications
        mpy     #0          ;Preg is cleared to zero
        zac                 ;Acc is cleared to zero
        sqra    psiRal ;Preg=(psiRal_est)2
        pac                 ;Acc=(psiRal_est)2 in 4.12 format (spm=2)
```

```
        sach    tmp         ;tmp=(psiRal_est)2
        sqra    psiRbe      ;Preg=(psiRbe_est)2
        pac                 ;Acc=(psiRbe_est)2 in 4.12 format (spm=2)
        sach    tmp1        ;tmp1=(psiRbe_est)2
        lacc    tmp1
        add     tmp
        sacl    tmp         ;tmp=(psiRal_est)2 + (psiRbe_est)2 still in 4.12f
        spm     0           ;no automatic shift anymore
        lt      tmp         ;Treg=tmp
        mpy     #Ksqrt      ;Preg=tmp*Ksqrt
        pac                 ;Acc=Preg, the two index digits are located in the high Acc
        sach    Index       ;store the two interesting digits into Index (->8.8f)
        lacc    Index       ;Acc=Index
        add     #sqrt_tab   ;Acc=Index+sqrt table address
        sacl    tmp         ;tmp= Index+sqrt table address
        lar     AR5,tmp     ;load in AR5 the content of tmp
        lacc    *           ;Acc= sqrt(psiRal_est2+psiRbe_est2) in 8.8 format
        sacl    psiR_est,4  ;psiR_est=sqrt(psiRal_est2+psiRbe_est2) 4 left shift -> 4.12f
        sacl    psiR_est1,7 ;psiR_est1=sqrt(psiRal_est2+psiRbe_est2) 7 left shift -> 1.15f

*********************************************
* divisions psiRal_est1/psiR_est1
*           psiRbe_est1/psiR_est1
* input in f 1.15
* output (cosTeta_est, sinTeta_est) in f 4.12
*********************************************
    lacc    psiR_est        ;Acc=psiR_est
    bcnd    modnotzero,NEQ
    lacc    #0
    sacl    tmp
    b       modzero
modnotzero                  ;If rotor flux module <> 0
    lacc    psiRal_est1     ;Acc=psiRal_est1 a comp of the rotor flux in 1.15 format
    abs                     ;Acc=abs(siRal_est1), unsigned numbers division
    sacl    tmp             ;tmp=abs(psiRal_est1)
    lacc    tmp,15          ;store abs(psiRal_est1) in high side of Acc
    rpt     #15
    subc    psiR_est1       ;AccL=psiRal_est1/psiR_est1, res in 1.15 format
*   sfr
*   sfr
*   sfr                     ;three right shifts -> division result in 4.12 format
    sacl    cosTeta_est     ;store 1.15 res (=psiRal_est1/psiR_est1) in cosTeta_est
    lacc    psiRal_est1
    bcnd    cospos,GT
    lacc    cosTeta_est     ;If psiRal_est1<0 then cosTeta_est<0
    neg
modzero
    sacl    cosTeta_est
cospos
    lacc    psiR_est        ;cosTeta_est=psiRal_est1/psiR_est1, 4.12 format
    bcnd    modnotzero1,NEQ
    lacc    #0
    sacl    tmp
    b       modzero1
modnotzero1                 ;If rotor flux module <> 0
    lacc    psiRbe_est1     ;Acc=psiRbe_est1 b comp of the rotor flux in 1.15 format
    abs                     ;Acc=abs(psiRbe_est1), unsigned numbers division
    sacl    tmp             ;tmp=abs(psiRbe_est1)
    lacc    tmp,15          ;store abs(psiRbe_est1) in high side of Acc
    rpt     #15
    subc    psiR_est1       ;AccL=psiRbe_est1/psiR_est1, res in 1.15 format
*   sfr
*   sfr
```

```
*     sfr                              ;three right shifts -> division result in 4.12 format
      sacl      sinTeta_est      ;store 1.15 res (=psiRbe_est1/psiR_est1) in sinTeta_est
      lacc      psiRbe_est1
      bcnd      sinpos,GT
      lacc      sinTeta_est      ;If psiRbe_est1<0 then sinTeta_est<0
      neg
modzero1
      sacl      sinTeta_est      ;sinTeta_est=psiRbe_est1/psiR_est1, 4.12 format
sinpos

*******************************
* END divisions          *
*******************************
      ret
```