1-1-2003

Speed sensorless control of 3-phase induction motor using MRAS speed estimator

Chaozheng Ma
Ryerson University

SPEED SENSORLESS CONTROL OF 3-PHASE INDUCTION MOTOR USING MRAS SPEED ESTIMATOR

by

CHAOZHENG MA

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

Instructions on Borrowers

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

Abstract

This project investigates the application of model reference adaptive system (MRAS) for the speed sensorless control of an induction motor. The rotor speed can be accurately estimated by employing the closed-loop observer named reactive MRAS. Therefore, this observer eliminates the need of a speed sensor for the control of the motor speed. The method is robust to stator and rotor resistance variations due to change of temperature. The dynamic system equations of the induction machine are formulated, and the motor control system performance is studied. Both scalar voltage-to-frequency (V/f) control and vector field oriented control (FOC) schemes, implemented using digital signal processor (DSP), are investigated. The design of the speed sensorless DSP-based controller is completed. Software packages have been developed to implement the design. An experimental system using the proposed controller has been built. Various tests have been conducted to verify the technical feasibility of the control technique. The experimental results confirm the feasibility of the proposed speed sensorless V/f control scheme using MRAS speed estimator. The designed V/f profile has been tested. Even with step change of the load or that of the command speed, the system can achieve the correct steady state after a short transient operation. The experimental results also confirm the feasibility of the proposed speed sensorless FOC control scheme using MRAS speed estimator. The current regulators meet the design requirements. Both the flux-producing current component and the torque-producing current component can be controlled separately. In the implementation, digital signal processor (DSP) TMS320 FL2407 and voltage source inverter (VSI) Skiip 342GD120-316CTV are employed. The modular strategy is adopted to develop the software package.

Acknowledgments

I would like to express my sincere gratitude and appreciation to Professor Richard Cheung, my supervisor, for his guidance throughout the entire period of this degree program. I also sincerely thank Professor Bin Wu, Dr. S. Wei, Dr. D. Xu and my classmates for their valuable suggestions and help. I thank Ryerson University for the Scholarship.

My special appreciation is extended to Ms Shuping Yang, my wife, for her support.

Contents

List of Symbols

[Motor parameters]

r_s: Stator resistance.

r_r: Rotor resistance refered to stator side

L_{ls}, L_{lr}: Stator and rotor leakage inductance

L_l: Total stator and rotor leakage inductance ($=L_{ls}+L_{lr}$)

L_m: Mutual inductance

L_s: Stator self inductance ($=L_m+L_{ls}$)

L_r: Rotor self inductance ($=L_m+L_{lr}$)

σ: Leakage coefficient ($=1-L_m^2/L_s L_r$)

τ_r: Rotor time constant ($=L_r/r_r$)

[Variables in an α-β stationary frame]

$v_{s\alpha}, v_{s\beta}$: α and β axis components of stator voltage

$i_{s\alpha}, i_{s\beta}$: α and β axis components of stator current

$i_{r\alpha}, i_{r\beta}$: α and β axis components of rotor current

$\psi_{s\alpha}, \psi_{s\beta}$: α and β axis components of stator flux

$\psi_{r\alpha}, \psi_{r\beta}$: α and β axis components of rotor flux

[Variables in an d-q rotating frame]

i_d, i_q: d and q axis components of stator current

θ, theta: Rotor flux position.

[Velocities, electric angle and torque]

n: Rotor speed (rpm)

ω_r: Rotor angular speed

ω_{syn}: Synchronous angular speed

ω_{sl}: Rotor Slip angular speed

[Subscripts]

p: Derivative operator(=d/dt)

s,r: Symbol of stator and rotor, respectively.

*, ref: Reference or setting value.

fdb: Feedback value in a closed loop.

est: Estimated value.

Chapter 1

Introduction

Induction motors are relatively economical and reliable machines because they are built without slip rings and commutators compared to dc machines. However, the use of induction motors may have several disadvantages, which include control difficulty due to the motor nonlinear behavior with saturation effects and the electrical parameter variations with temperature. To overcome this difficulty, much attention has been given to the control of induction motors for starting, braking, speed reversal, speed change, etc. Several techniques of controlling an induction motor have been proposed, such as voltage/frequency (V/f) control and field oriented control (FOC). V/f control is a scalar control, where the ratio of the supply voltage and frequency is maintained constant with the exception that at low speeds the machine flux level is kept constant. FOC control is a vector control, where the flux and the torque are controlled separately [1-5].

Usually a speed sensor is required in the closed loop operation for both V/f control and FOC control. The speed sensor or tachometer is mounted on the motor shaft. However, the shaft encoder may present problems. Delicate critical encodes with internal signal electronics are used, which lower the reliability of the motor system, especially in hostile environments, that requires careful arrangements with special attention to electrical noise. There are situations where the positional feedback is extremely difficult to obtain [3]. In addition, the sensor is cost factor as the provision of special motor-shaft and encode-mounting surfaces leads to more expensive machine control systems. The elimination of the speed sensor has been one of the important features in the modern motor control systems. This project presents a novel method of controlling the speed of induction motors without the speed sensor. The information of the rotor speed can be obtained by processing the stator voltages and currents measured at the motor terminals. The rotor

speed is estimated by employing a closed-loop observer named model reference adaptive systems (MRAS). In MRAS, a comparison is made between the outputs of two observers. One observer does not involve the rotor speed to be estimated, and one does. A scheme is based on the instantaneous reactive power that maintains the magnetizing current [8]. This approach is also called reactive power MRAS. The method is robust to stator and rotor resistance variations due to temperature changes. This project investigates the MRAS speed estimation technique in the implementation of speed sensorless system for scalar voltage-to-frequency (V/f) control and vector field orientation control (FOC) of induction motors.

This chapter provides an introduction of the control method developed in this project.
Section1.1 introduces the conventional speed control schemes for induction motors.
Section1.2 discussed the improved speed sensorless control schemes for induction motors using MRAS technique.
Section 1.3 presents the objectives of this project.
Section 1.4 provides the structure of this project.

1.1 Conventional Speed Control Schemes for Induction Motors

A typical V/f control scheme is shown in Figure1-1.

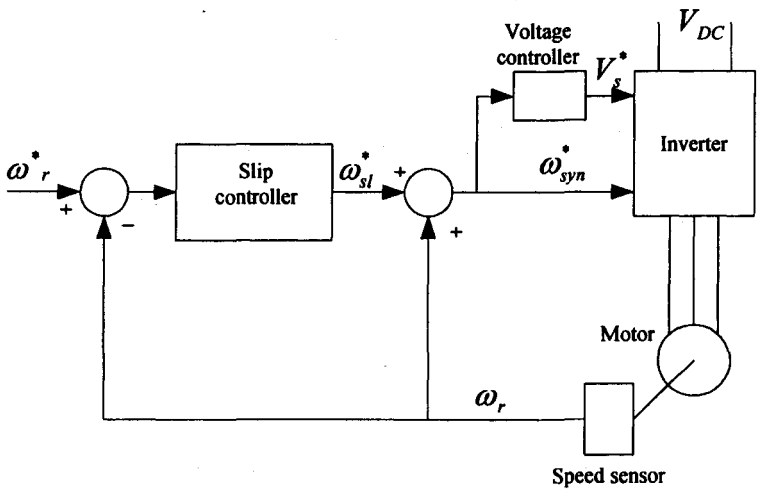


Figure1-1: Constant V/f Speed Control with a Speed Sensor

In Figure1-1, the speed of induction motor ω_r is sensored and compared with the reference speed ω_r^* to calculate the compensated slip. The synchronous speed ω_{syn}^* is computed from the compensated slip and the rotor speed. The relationship between the supply voltage and frequency is maintained linear except at low speeds in order to keep the machine flux level constant and close to its nominal values for various motor operating conditions. Constant V/f control techniques are based on static models of the induction motor for the constant flux operation. The V/f control is one of the scalar control techniques [1].

Vector control techniques have made induction motors suitable for high performance applications where traditionally only dc motors are used. The field orientation control is a vector control. In this control, the flux and the torque are controlled separately. A rotor flux oriented vector control scheme is shown in Figure 1-2 [9].

In Figure1-2, the measured induction motor current signals are converted to two co-ordinate current signals, i_d and i_q, using the Clark's and Park's transformations. The resulting two dc quantities are compared with the reference d-axis and q-axis components, i_{dref} and i_{qref}. The output of the inverse Park module "I-Park" is used to generate the pulse width modulation (PWM) signals for turning on/off the switches in the inverter that drives the motor. A speed sensor is used to measure the motor speed that provides the speed feedback information for the control of the induction motor.

1.2 Improved Speed Sensorless Control Schemes for Induction Motors

This project addresses the real-time control of induction motors combined with MRAS speed estimator. The speed of the induction motor is computed using MRAS estimator. This control scheme has improved the conventional speed control schemes discussed in Section 1.1. This control is implemented in the Power Electronic Laboratory at Ryerson University. The improved speed sensorless V/f and FOC control structures using MRAS technique are shown by Figure1-3 and Figure1-4, respectively.

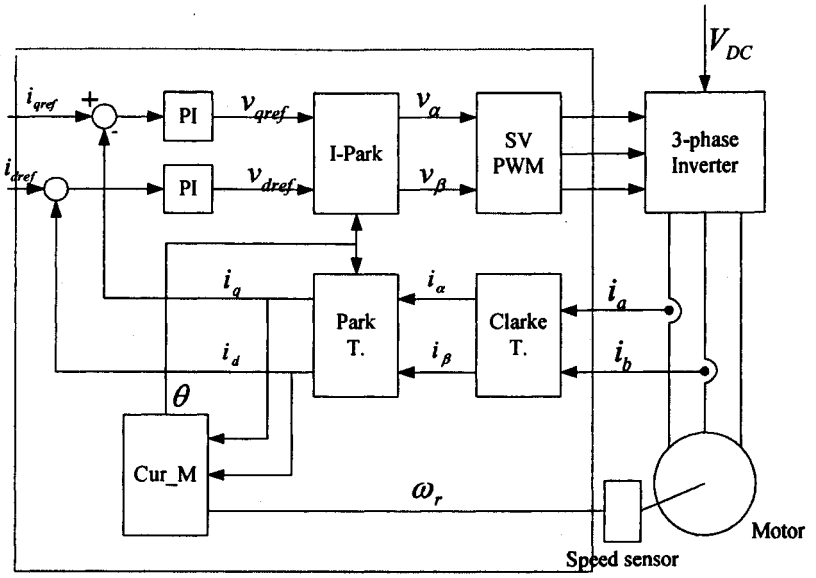


Figure 1-2: A Conventional FOC Control Scheme with a Speed Sensor

The main contribution of this project consists of achieving the sensorless control by using MRAS speed estimator in a digital signal processor (DSP) controller. A DSP is especially employed to implement the motor control schemes. The performance of an induction motor strongly depends on the characteristics of its control. DSP can be used to enhance the real time control of induction motors without the need of any electromechanical speed sensor. This reduces the number of components used in the conventional drives and optimizes the design to reduce overall drive system cost. A fixed point DSP TMS320LF2407 from Texas Instrument is used in the implementation. Structured software modules are used to build the control system. The software modules are written in assembly languages.

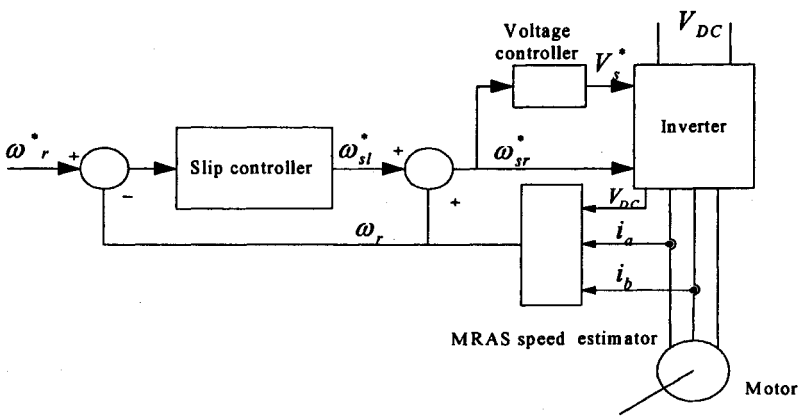


Fig.1-3: Sensorless Speed V/f Control Scheme Using MRAS Estimator

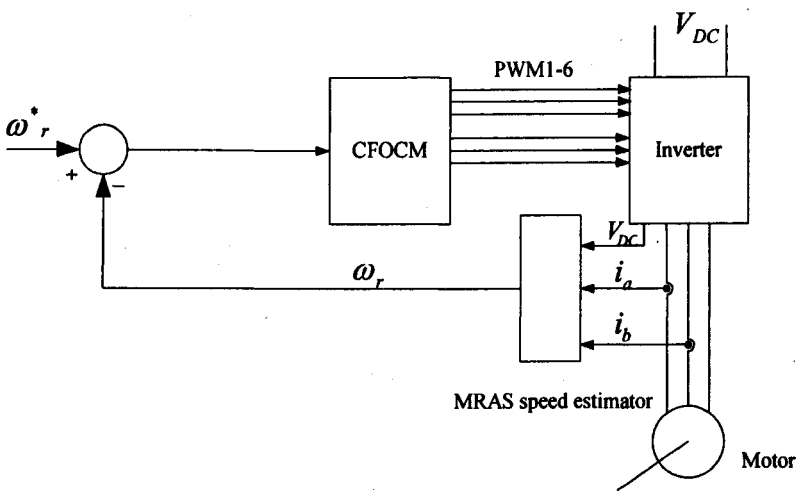


Figure 1-4: Sensorless Speed FOC Control Scheme Using MRAS Estimator

Note that CFOCM in Figure1-4 represents the conventional FOC module illustrated in the dot-lined part of Figure1-2.

1.3 Project Objectives

The main objective of this project is to use DSP to implement the improved speed sensorless control schemes for induction motor drive. Both scalar V/f control and vector FOC control schemes are investigated. MRAS technique is used to estimate the motor speed such that a speed sensorless three-phase motor control is achieved. More specifically, the following research is conducted:

1- The speed estimation technique MRAS is studied. An induction model is used to determine the performance of the induction motor. Differential equations are developed to describe the principle of the MRAS speed estimator. The relevant variables are studied to improve the performance of the MRAS speed estimator. The aim is to eliminate a speed sensor normally required in the control of an induction motor.
2- Both speed sensorless scalar V/f control and vector FOC control schemes are implemented.

3- Several different test conditions, including a step change in speed command and an abrupt change in load, are carried out to verify the proposed control method and to demonstrate the system performance.
4- An experimental system including a self-excited induction generator used as the load for the controlled prime motor is built.
5- Software written in assembly languages is developed for the control of induction motor. The software is reliable and flexible for future development.

1.4 Project outlines

The structure of this project is as follows:

Chapter 2 describes the modeling of an inductor motor. The MRAS technique is formulated, and the variables are determined.

Chapter 3 describes the details for the proposed V/f control using MRAS technique. The design considerations are presented, and the control strategy is discussed. Software package is developed, and software flow chart is illustrated. The experimental system for this improved V/f control is built.

Chapter 4 presents the details for the proposed FOC control using MRAS technique.

Chapter 5 shows the experimental results and demonstrates the achieved performance.

Chapter 6 provides the conclusions and suggestions for future studies.

Chapter 2

MRAS—Model Reference Adaptive System

MRAS is an approach to estimate induction motor speed from measured terminal voltage and currents for speed-sensorless motor control. It is based on computation of the instantaneous reactive power that maintains the motor magnetizing current [8,11]. The technique is simple, robust to variations of motor parameters due to operating conditions. This approach is not dependent upon the knowledge of the values of stator resistance, nor is it affected by stator resistance thermal variations. In this chapter, the 3-phase to 2-phase transformation is introduced first. Then a general description of the model of induction machines in the (α,β) stationary reference frame is presented. Equations of MRAS estimator of motor speed are derived, and its application to induction motors is discussed.

2.1 3-phase to 2-phase Transformation

The three-phase voltages, currents and fluxes of induction motors can be analyzed in terms of complex space vectors. With regard to the currents, the space vector can be defined as follows.

Assume that i_a, i_b and i_c are the instantaneous currents in the stator phases. Then the complex stator current vector $\bar{i}_s$ is defined by:

$$\bar{i}_s = i_a + \alpha i_b + \alpha^2 i_c \qquad (2\text{-}1)$$

where $\alpha = e^{j\frac{2}{3}\pi}$ and $\alpha^2 = e^{j\frac{4}{3}\pi}$ represent the spatial operators. Figure 2-1 shows the complex space vectors of stator currents:

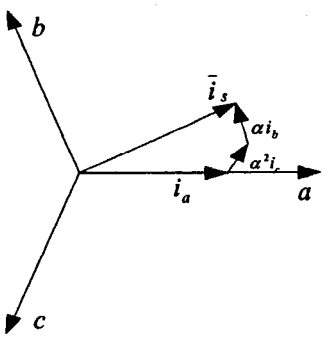


Figure 2-1: Stator Current Space Vector and its Components in (a,b,c)

where (a,b,c) are the three phase axes. This current space vector, which represents the three phase sinusoidal system, could be transformed into a two time invariant coordinate system by the following two steps.

- (a,b,c)⇒(α,β): Transform the three-phase system into a two-phase system where the orthogonal coordinate system is time varying. This transformation is crucial to the MRAS technique, which will be discussed in this chapter.
- (α,β)⇒(d,q): Transform the time-varying system into time-invariant system. This transformation is crucial to the field oriented control, which will be discussed in Chapter 4.

2.1.1 Clarke Transformation of (a,b,c) to (α,β)

The space vector can be expressed in a reference frame with only two orthogonal axes called (α,β). It is assumed that the axis "a" and the axis "α" are in the same direction as shown in Figure 2-2. This transformation is also called Clarke transformation.

Since in a three-phase balanced system, the sum of three phase currents $i_a + i_b + i_c = 0$. The two-phase (α,β) currents can derived as follows.

$$\begin{aligned} i_\alpha &= i_a \\ i_\beta &= (i_a + 2i_b)/\sqrt{3} \end{aligned} \qquad (2\text{-}2)$$

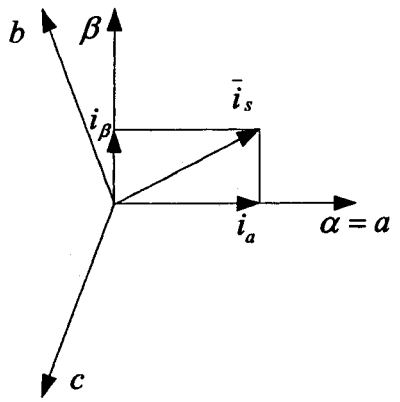


Figure 2-2: Clarke Transformation

If the three phase currents are assumed as:

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

then the output (α,β) currents are:

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

2.1.2 Park Transformation of (α,β) to (d,q)

Park transformation is used to change the two-phase orthogonal system (α,β) into (d, q) rotating reference frame. This is the most important transformation in the FOC control which will discussed in the Chapter 4. With the *d axis* aligned with the rotor flux, Figure 2-3 shows the relationship of the two reference frames:

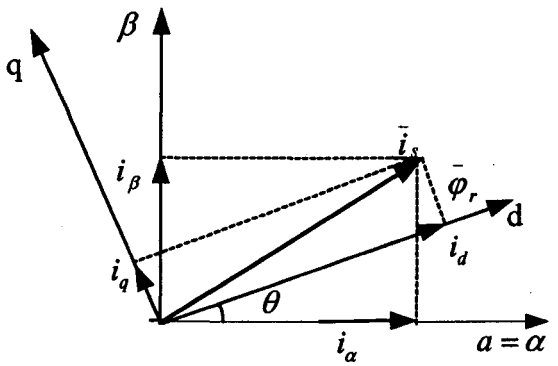


Figure 2-3: Park Transformation

In Figure2-3 θ is the rotor flux position. The flux and torque components of the current vector are determined by the following equations:

$$\begin{aligned} i_d &= i_\alpha \cos\theta + i_\beta \sin\theta \\ i_q &= -i_\alpha \sin\theta + i_\beta \cos\theta \end{aligned} \qquad (2\text{-}5)$$

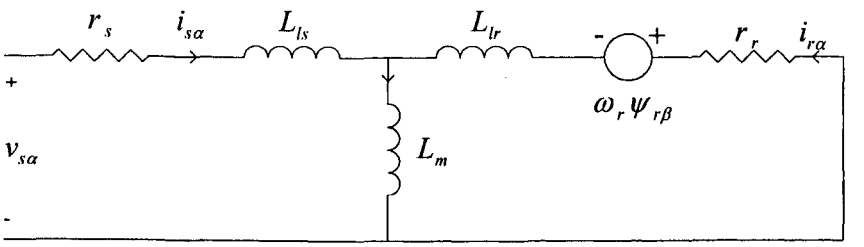
These components depend on the current vector (α,β) components and on the rotor flux position.

The (d,q) system has the following characteristics:

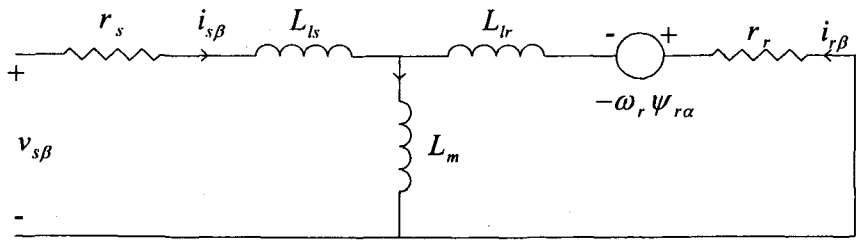
- It is a two-coordinate time-invariant system, where both i_d and i_q are DC quantities.
- The current i_d is a flux-related component and the current i_q is a torque-related component.

2.2 Dynamic Model of Induction Machine

To implement the MRAS, an appropriate model of induction motor is required. The model proposed in [10] is taken into consideration. Figure 2.4 shows the fundamental induction motor model in the (α,β) stationary reference frame. An induction machine model that is used by complex space vectors relies on the sinusoidal distribution. Space harmonics are not considered, and the model is called fundamental model [4].



(a)



(b)

Figure 2-4: The Equivalent Circuit of Induction Machine Model in (α,β) Stationary Reference Frame.

The related stator and rotor voltage equations in the stator frame are [4]:

$$v_{s\alpha} = r_s i_{sa} + \frac{d\psi_{s\alpha}}{dt} \qquad (2\text{-}6)$$

$$v_{s\beta} = r_s i_{s\beta} + \frac{d\psi_{s\beta}}{dt} \qquad (2\text{-}7)$$

$$0 = r_r i_{r\alpha} + \frac{d\psi_{r\alpha}}{dt} + \omega_r \psi_{r\beta} \qquad (2\text{-}8)$$

$$0 = r_r i_{r\beta} + \frac{d\psi_{r\beta}}{dt} - \omega_r \psi_{r\alpha} \qquad (2\text{-}9)$$

where ω_r is the angular velocity of the rotor, the flux linkage equation are:

$$\psi_{s\alpha} = L_s i_{s\alpha} + L_m i_{r\alpha} \qquad (2\text{-}10)$$

$$\psi_{s\beta} = L_s i_{s\beta} + L_m i_{r\beta} \qquad (2\text{-}11)$$

$$\psi_{r\alpha} = L_m i_{s\alpha} + L_r i_{r\alpha} \qquad (2\text{-}12)$$

$$\psi_{r\beta} = L_m i_{s\beta} + L_r i_{r\beta} \qquad (2\text{-}13)$$

where $L_s = L_{ls} + L_m$, and $L_r = L_{lr} + L_m$.

Equations (2-6) to (2-13) are used by MRAS to compute the reactive power in the reference model and that in the adaptive model of MRAS.

2.3 MRAS Speed Estimator

MRAS is based on the comparison between the outputs of two observers. The observers are used to calculate the instantaneous reactive power maintaining the magnetizing current [8, 11]. Figure 2-5 illustrates the structure of MRAS for speed estimation.

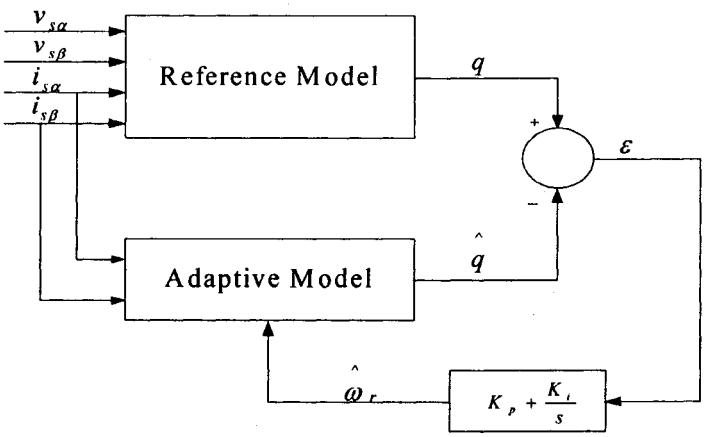


Figure 2-5: Simplified Block Diagram of Reactive Power MRAS Speed Estimator

The input data required for this model is the stator voltages and stator currents in the (α,β) stationary reference frame. Two sets of equations are developed to compare reactive power of the induction machine in the reference model and the adaptive model. The reference model does not involve the rotor speed, while the adaptive model needs the estimated rotor speed to adjust the computed reactive power to that computed from the reference model. The upper portion of Figure 2-5 serves as a reference model while the lower one as an adaptive model. The error between two models is used to drive a suitable proportional integrator (PI) controller, which generates an estimated rotor speed $\hat{\omega}_r$.

2.3.1 Reference Model

The active power in the reference model is computed from cross product of the stator currents and the counter electromotive force (emf). The counter emf is derived from Equations (2-6), (2-7), (2-10) and (2-11). They can be expressed as follow.

$$e_{m\alpha} = \frac{L_m}{L_r}\frac{d\psi_{r\alpha}}{dt} = v_{s\alpha} - r_s i_{s\alpha} - \sigma L_s \frac{di_{s\alpha}}{dt} \quad (2\text{-}14)$$

$$e_{m\beta} = \frac{L_m}{L_r}\frac{d\psi_{r\beta}}{dt} = v_{s\beta} - r_s i_{s\beta} - \sigma L_s \frac{di_{s\beta}}{dt} \quad (2\text{-}15)$$

$$\bar{e}_m = e_{m\alpha} + je_{m\beta} \quad (2\text{-}16)$$

where $\sigma = 1 - \frac{L_m^{\ 2}}{L_s L_r}$, leakage coefficient. The stator currents,

$$\bar{i}_s = i_{s\alpha} + ji_{s\beta} \quad (2\text{-}17)$$

The reactive power q, which represents the instantaneous reactive power that maintains the magnetizing current, can be computed from the following equation:

$$q = \bar{i}_s \times \bar{e}_m \quad (2\text{-}18)$$

The reactive power in (2-18) can be further derived as:

$$q = i_{s\alpha} v_{s\beta} - i_{s\beta} v_{s\alpha} - \sigma L_s (i_{sa}\frac{di_{s\beta}}{dt} - i_{s\beta}\frac{di_{sa}}{dt}) \quad (2\text{-}19)$$

2.3.2 Adaptive Model

The active power in the adaptive model is computed from cross product of the stator currents and the estimated counter electromotive force (emf). The estimated counter emf is derived from Eq. (2-8) to (2-13). They can be expressed as follow.

$$\hat{e}_{m\alpha} = \frac{L_m^{\ 2}}{L_r}\frac{di_{m\alpha}}{dt} = \frac{L_m^{\ 2}}{L_r\tau_r}(-\tau_r \hat{\omega} i_{m\beta} - i_{m\alpha} + i_{s\alpha}) \quad (2\text{-}20)$$

$$\hat{e}_{m\beta} = \frac{L_m^{\ 2}}{L_r}\frac{di_{m\beta}}{dt} = \frac{L_m^{\ 2}}{L_r\tau_r}(-\tau_r \hat{\omega} i_{m\alpha} - i_{m\beta} + i_{s\beta}) \quad (2\text{-}21)$$

$$\hat{e}_m = \hat{e}_{m\alpha} + j\hat{e}_{m\beta} \quad (2\text{-}22)$$

where $\tau_r = L_r / r_r$ is rotor time constant. The currents $i_{m\alpha}, i_{m\beta}$ can be computed from following equations:

$$\frac{di_{m\alpha}}{dt} = -\hat{\omega}_r i_{m\beta} - \frac{1}{\tau_r} i_{m\alpha} + \frac{1}{\tau_r} i_{s\alpha} \qquad (2\text{-}23)$$

$$\frac{di_{m\beta}}{dt} = \hat{\omega}_r i_{m\alpha} - \frac{1}{\tau_r} i_{m\beta} + \frac{1}{\tau_r} i_{s\beta} \qquad (2\text{-}24)$$

After the estimated counter emf, $\hat{e}_m$, is computed by using (2-20) to (2-21), the estimated reactive power can be computed as follows.

$$\hat{q} = \bar{i}_s \times \hat{e}_m = i_{s\alpha} \hat{e}_{m\beta} - i_{s\beta} \hat{e}_{m\alpha} \qquad (2\text{-}25)$$

The voltage model of equation (2-19) does not involve rotor speed ω_r, while the current model of equation (2-25) does .The speed can be estimated based on the output difference between these two models.

$$\hat{\omega}_r = (k_p + \frac{K_i}{s})\varepsilon \qquad (2\text{-}26)$$

where $\varepsilon = q - \hat{q}$.

The above PI controller tunes the estimated rotor speed $\hat{\omega}_r$ such that the reactive power $\hat{q}$, generated by adaptive model, matches that q generated by reference model. From equations (2-19), (2-25) and (2-26), it is evident that the speed estimation system of Figure2-3 is robust to the stator resistance, and thus is not affected by the stator resistance thermal variations.

2.4 Per Unit, Discrete Time Representation

For implementation on DSP-based system, the differential equations given in Section 2.3 need to be discretized. For generality, per unit (PU) values are used in all equations. The reactive power MRAS can be expressed in the following.

Reference Model:

According to equation (2-19), and using backward approximation with the sampling period T, then

$$q(k)=i_{s\alpha}(k)v_{s\beta}(k)-i_{s\beta}(k)v_{s\alpha}(k)-\sigma L_s\left(i_{sa}(k)\frac{i_{s\beta}(k)-i_{s\beta}(k-1)}{T}-i_{s\beta}(k)\frac{i_{sa}(k)-i_{s\alpha}(k-1)}{T}\right) \quad (2\text{-}27)$$

Equation (2-27) can be further simplified as follows:

$$q(k)=i_{s\alpha}(k)v_{s\beta}(k)-i_{s\beta}(k)v_{s\alpha}(k)-\frac{\sigma L_s}{T}\left(i_{s\beta}(k)i_{s\alpha}(k-1)-i_{s\alpha}(k)i_{s\beta}(k-1)\right) \quad (2\text{-}28)$$

Dividing equation (2-28) by base power of $V_b I_b$ and rearranging the resulting equation the per unit representation is as follows:

$$q(k)=i_{s\alpha}(k)(V_{s\beta}(k)-K_1 i_{s\beta}(k-1))-i_{s\beta}(k)(V_{s\alpha}(k)+K_1 i_{s\alpha}(k-1)) \quad (2\text{-}29)$$

where $K_1=\frac{\sigma L_s I_b}{TV_b}$, V_b is base voltage, and I_b is base current.

Adaptive Model:

In same way as in reference model, equation (2-25) is discretized as follows:

$$\hat{q}(k)=i_{s\alpha}(k)\hat{e}_{m\beta}(k)-i_{s\beta}(k)\hat{e}_{m\alpha}(k) \quad (2\text{-}30)$$

and equation (2-21) to (2-24) can re-written in per unit representation as:

$$\hat{e}_{m\alpha}(k)=K_2(-K_3\hat{\omega}_r(k)i_{m\beta}(k)-i_{m\alpha}(k)+i_{s\alpha}(k)) \quad (2\text{-}31)$$

$$\hat{e}_{m\beta}(k)=K_2(-K_3\hat{\omega}_r(k)i_{m\alpha}(k)-i_{m\beta}(k)+i_{s\beta}(k)) \quad (2\text{-}32)$$

$$i_{m\alpha}(k)=i_{m\alpha}(k-1)(-K_4(\hat{\omega}^2(k)+K_5)-i_{m\beta}(k-1)\hat{\omega}(k)K_6+i_{s\alpha}(k)K_7 \quad (2\text{-}33)$$

$$i_{m\beta}(k)=i_{m\beta}(k-1)(-K_4(\hat{\omega}^2(k)+K_5)-i_{m\alpha}(k-1)\hat{\omega}(k)K_6+i_{s\beta}(k)K_7 \quad (2\text{-}34)$$

where:

$$K_2 = \frac{{L_m}^2 I_b}{L_r \tau_r V_b}, K_3 = \tau_r \omega_b = \frac{L_r \omega_b}{r_r}, K_4 = \frac{{\omega_b}^2 T^2}{2}, K_5 = 1 - \frac{T}{\tau_r} + \frac{T^2}{2{\tau_r}^2}$$

$$K_6 = \omega_b (T - \frac{T^2}{\tau_r}), K_7 = \frac{T}{\tau_r} - \frac{T^2}{2{\tau_r}^2}$$

After $i_{ma}(k)$ and $i_{m\beta}(k)$ in per unit are calculated from (2-33) and (2-34), the counter emf can be computed by using (2-31) and (2-32). Then per unit estimated reactive power in adaptive model can be simply calculated from (2-30).

2.5 Implementation of MRAS

To implement the discussed MRAS technique in motor control drive, some motor parameters are needed. The parameters consist of:

- Number of poles
- Rotor resistance (r_r)
- Stator leakage inductance (L_{sl})
- Rotor leakage inductance (L_{rl})
- Magnetizing inductance (L_m)

The base quantities are:

- Base current (I_b)
- Base voltage (V_b)
- Base electrically angular velocity (ω_b)
- Sampling period (T)

In summary, the 3-phase to 2-phase transformation has been introduced. The general description of the model of induction machines in the (α,β) stationary reference frame has been presented. Equations of MRAS estimator of motor speed have been derived, and its application to induction motors has also been discussed in this chapter.

Chapter 3

V/f Control of Induction Motor Using MRAS

Constant V/f control techniques are based on the static models of the induction motor for the constant flux operation. The V/f control is a scalar control. This chapter deals with the scalar V/f control of three-phase induction motors using DSP eZdsp^{TM} LF2407. This chapter explains in detail the hardware and software configuration. This chapter presents the techniques to implement the speed sensorless V/f control system for induction motor drive using MRAS speed estimator. An experimental system is built to validate the designed drive performance.

3.1 Background for V/f Control of Induction Machine

In the V/f control, the speed of induction motor is controlled by adjusting both the magnitude V and the frequency f of the stator voltages in such a way that the air gap flux is always maintained at the desired steady-state level [11]. Figure 3-1 shows the steady-state equivalent circuit of an induction machine. In the figure, x_{ls}, x_{lr} and x_m represent the stator leakage impedance, the rotor leakage impedance and the mutual impedance, respectively. r_s, $\frac{r_r}{s}$, v_s and E_m represent the stator resistance, equivalent rotor resistance, stator voltage and the counter electromotive force (*emf*), respectively.

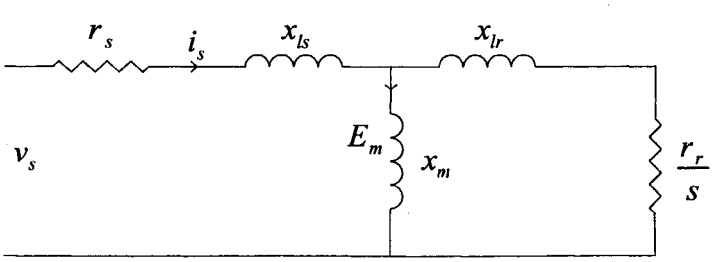


Figure 3-1: Equivalent Circuit of Induction Machine

If the voltage applied to a three-phase induction motor is sinusoidal, we have, at steady state, the *emf* and the air gap flux can be expressed as:

$$E_m = 4.44 N_s f_s \psi \qquad (3\text{-}1)$$

where N_s is number of turns, f_s is stator frequency, and ψ is air gap flux.

$$\psi = \frac{1}{4.44 N_s} \frac{E_m}{f_s} = k \frac{E_m}{f_s} \qquad (3\text{-}2)$$

In case of f_s is high, $x_m = 2\pi f_s L_m \rangle\rangle r_s$ and $x_m \rangle\rangle x_{ls}$, then

$$E_m = \left| v_s - (r_s + j x_{ls}) i_s \right| \approx V_s \qquad (3\text{-}3)$$

$$\psi = k \frac{E_m}{f_s} \approx k \frac{V_s}{f_s} \qquad (3\text{-}4)$$

where V_s is the magnitude of the stator voltage.

From (3-4), it follows that if the ratio V_s / f_s keeps constant for any change in the supply frequency f_s for the speed control of an induction motor, then the motor flux remains constant and the torque becomes independent of the supply frequency.

Since the stator flux is maintained to be constant, the torque developed depends only on the slip speed. This is shown in Figure 3-2. Therefore by regulating the slip speed, the torque and speed of an induction motor can be controlled using the constant V/*f* method.

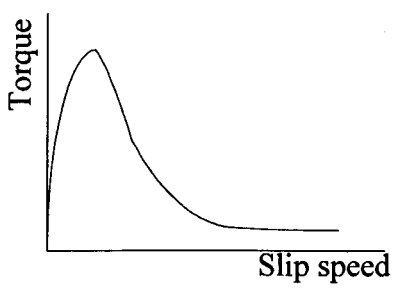


Figure 3-2: Torque vs. Slip Speed of an Induction Motor with Constant Flux

In a practical implementation, the ratio of the stator voltage magnitude to frequency is usually based on the motor rated parameters. However, in case of the frequency, and hence the stator voltage, is low, the voltage drop across the stator resistor and leakage inductance cannot be neglected and must be compensated. On the other hand, when the frequency is higher than the rated value, maintaining constant V_s / f_s means exceeding rated stator voltage and thereby causing the possibility of voltage insulation problem. In order to avoid this, constant V/f principle is also violated at such frequencies. Basically, there are three speed ranges in the V/f profile: stator voltage drop compensation region, linear region and field weakening region. A typical V/f profile is illustrated in Figure 3-3.

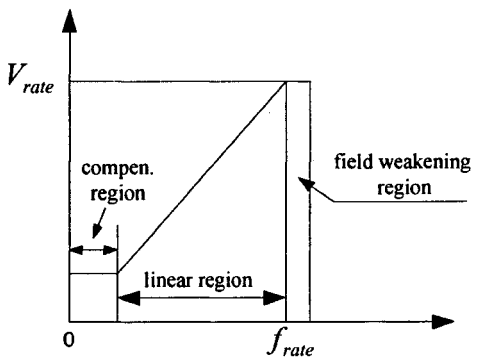


Figure3-3: Voltage vs. Frequency Profile

3.2 V/f Control

To precisely control the speed of the induction motor with a load, a closed-loop V/f control is needed. Since the frequency of the stator voltage represents the synchronous speed but not the real speed of the motor, a slip speed exists. At no load condition, the slip is very small. As the load increases, the slip also increases. The slip controller is needed for speed regulation. A typical V/f control with a speed sensor has been shown in Figure 1-1, and for convenience, it is re-shown in Figure 3-4.

In Figure3-4, the speed of induction motor ω_r is measured using a sensor and compared with the reference speed ω_r^* to calculate the compensated slip. The synchronous speed ω_{syn}^* is computed from the compensated slip and the rotor speed.

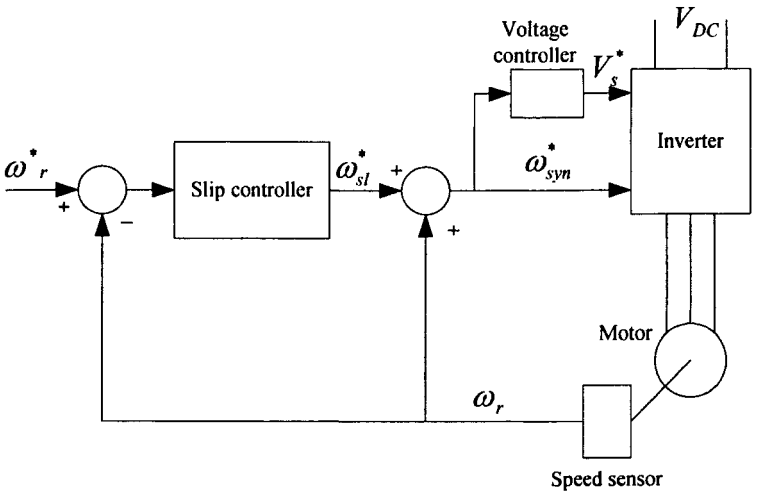


Figure 3-4: Constant V/f Speed Control Scheme with a Speed Sensor

The speed of the induction motor can be estimated by MRAS estimator. The synchronous speed ω^*_{syn} can be computed from the compensated slip and the reference rotor speed. As shown in Chapter 2, MRAS estimator consists of two models: the reference model and the adaptive model. The reference model does not involve the rotor speed, while the adaptive model needs the estimated rotor speed to adjust the computed reactive power to that computed from the reference model. The error between two models is fed to a PI controller, which generates an estimated rotor speed ω_r. Figure 3-5 shows the proposed speed sensorless V/f control scheme using MRAS technique. The detail will be discussed in this Chapter.

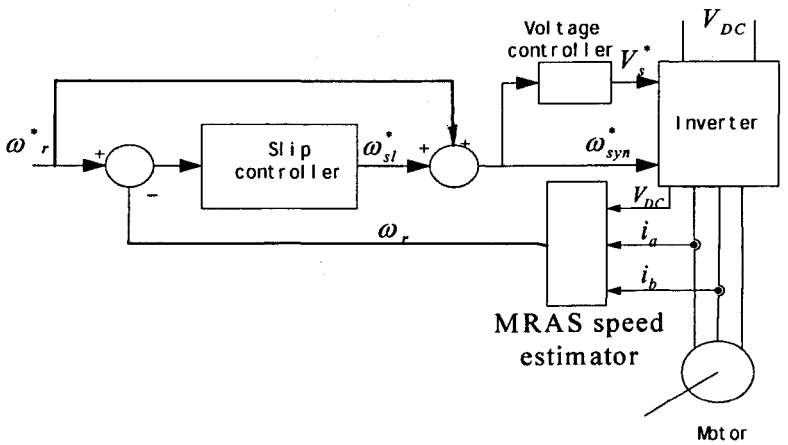


Figure 3-5: Sensorless Speed V/f Control Scheme Using MRAS Estimator

3.3 Implementation of V/f Control

This section discusses the implementation details, which include the organization of hardware and software. Debugging of the software is also addressed.

3.3.1 The Hardware Organization

Figure 3-6 shows an overview of the hardware required to implement the control scheme illustrated in Figure 3-5. DSP is the core of the control system. Three analog to digital converter (ADC) channels of DSP are used to sense stator currents (i_a, i_b) of the motor and the bus DC voltage V_{DC} of the voltage source inverter (VSI). The DSP uses their measured values to estimate the rotor speed. Six PWM output signals from the DSP are used to drive the VSI and implement the V/f control of an induction motor. An isolated transformer is employed to isolate the VSI from the utility. In the commissioning phase, a variac is also employed to increase the input AC voltage of VSI step by step.

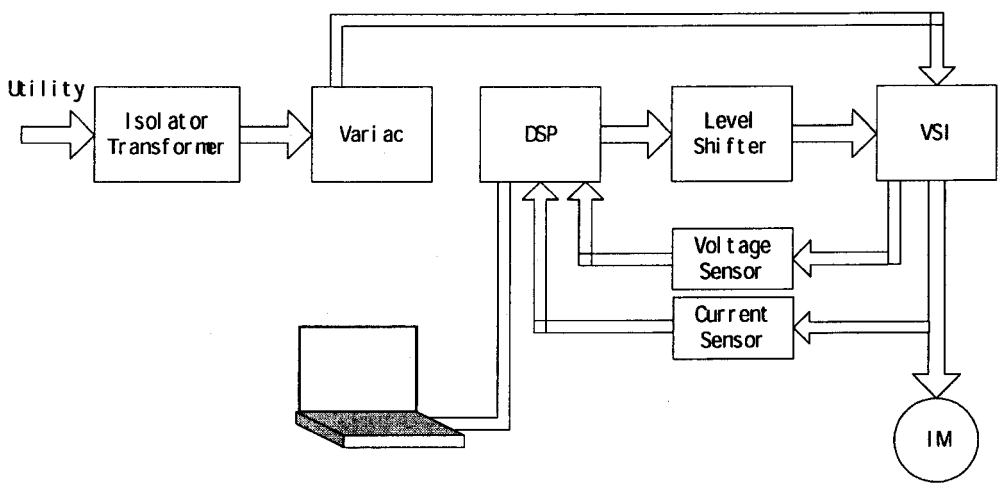


Figure 3-6 Experimental Setup for Sensorless Speed Control System

The major hardware components used in the experimental system are given as follows.

1- *eZdsp*^{TM} *LF*2407 **Board**

eZdsp™ LF2407 is the platform to develop and run software for the TMS320LF2407 digital signal processor (DSP). It allows the verification of LF2407 code. With 64K words of onboard program/data RAM, it can solve a variety of problems. Its three expansion connectors are provided for any necessary evaluation circuitry not provided on the configuration. The C2000 Code Composer driver is used to simplify code development and shorten debugging time. Figure 3-7 shows its block diagram.

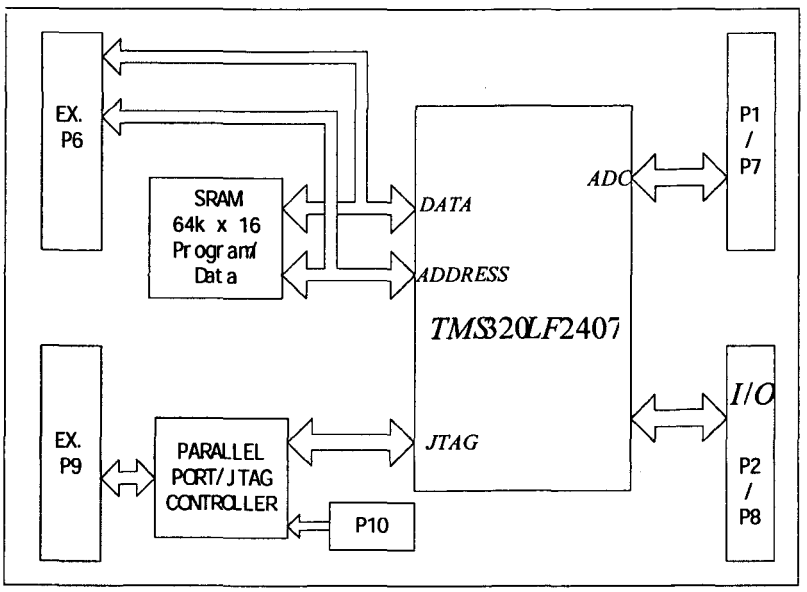


Figure3-7: Block Diagram *eZdsp™ LF2407*

The *eZdsp™ LF2407* board has the following features:

- Development/Emulation environment: Code Composer (CC) C'2000 supports real-time debugging.
- Onboard 7.3728-MHz oscillator for 30 MIPS operating speed.
- Power by a 5-volt power supply.
- 0 to 3.3V input and output signal
- 16 channels of Analog to Digital interface, no Digital to Analog interface.
- 16 channels of PWM interface connector

In the design of control scheme, DSP related settings are as follows.

- PWM frequency/period: 5kHz/200μs.

- PLL: 4.
- PWM mode: symmetrical with dead band 1.2μs.
- Interrupts: T1 underflow
- Peripheral Usage: Timer1, PWM 1-6, 3 ADC channels

2- Voltage Source Inverter (VSI)

The power electronic inverter module Skiip® 342GD120-316CTV is employed to supply power to the motor. The inverter switches are rated at an input voltage (V_{CES}) of 1200V and a maximum line current of 120A. The inverter module consists of two parts. The first part of the inverter circuit is used to rectify AC power to DC power. The second part of the inverter circuit converts the DC power to the required AC power for motor operation using six power IGBTs driven on/off by the DSP controller.

3-Isolation Transformer

The inverter is isolated from the utility power system using a transformer. The rated power of the transformer is 23.4kVA, and the rated current is 28A.

4- Personal Computer

The PC is connected with DSP board through RS232 port. It has been installed with Code Composer (CC) C'2000 which is used to build, compile, debug and run the required software.

5- The Voltage Sensor and Current Sensor

The V/f control needs measuring the VSI dc bus and the two phase currents as input signals to the DSP controller. In the implementation, the current transducers (LEM type, +/-10A) measure the two currents and the voltage transducer sense the DC voltage. Since the input and output signals of the DSP has the range of 0 to 3.3V, the output signal of

the sensors must therefore be adjusted to the same range. Since the output signal of the current sensors can be either positive or negative, 1.65V analog offset is required in order to allow the single voltage analog-to-digital converter (ADC) module to read both positive and negative values. To filter the harmonic components, a R-C filter is connected before AD conversion. The resistance is selected to 2kΩ, and the capacitance is 0.1 μF. The block diagram below shows of the implementation of current sensor.

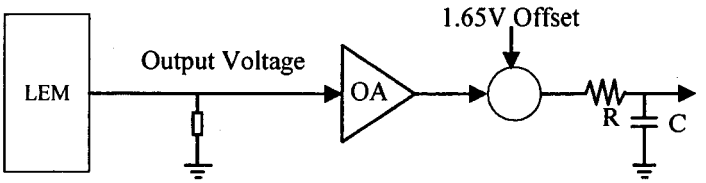


Figure 3-8: Current Sensing Interface Block Diagram

6- Level Shifter

The output PWM signals of the DSP used are 3.3V. Since the switching signal of the IGBT needs 15V, a voltage-level shifter is required to amplify PWM signals. The chip MC14504 is used to implement the amplification (voltage level shift).

7- Induction motor

The induction motor used in this project experiments is a squirrel cage three phase Y-connected motor from *TECO-WESTINGHOUSE*. The parameters and rated values of this motor are listed in Appendix A.

3.3.2 The Software Organization

The modular strategy is adopted in the development of the software package for the control of induction motor [15]. The standard modules are investigated and have been modified to meet the objective of this project [10]. DLOG_VIEW module [7] is employed to display two graphical waveforms in real time. The package consists of the

initialization module and the run module, which are written in assembly language. The former is performed only once at the beginning of operation. A typical incremental-system-build process is developed in this module to reduce the debugging time. There are five phases for debugging in the main program. The second module is based on a waiting loop interrupted by the PWM underflow. The complete V/f algorithm is computed within the PWM Interrupt Service Routine (ISR).

The DSP Controller Full Compare Unit is used to generate the necessary pulsed signals to the level shifter board. The phase lock loop (PLL) unit is set to 4, so that the CPUCLK runs at 30MHz. It is programmed to generate symmetrical complementary PWM signals at a frequency of 5kHz, with the timer TIMER1 as the time base and with the dead time of 1.2μs. The sampling period (T) of 200 μs can be established by setting the timer period T1PER to 3000 (PWMPRD=3000). Figure 3-9 illustrates the time diagram for the initialization and the operating system. Figure 3-10 illustrates the software flowchart. Figure 3-11 depicts the software modules for the V/f control system as follows. The software modules meet the design requirements: flexibility, compatibility and expandability, which are listed in Appendix B.

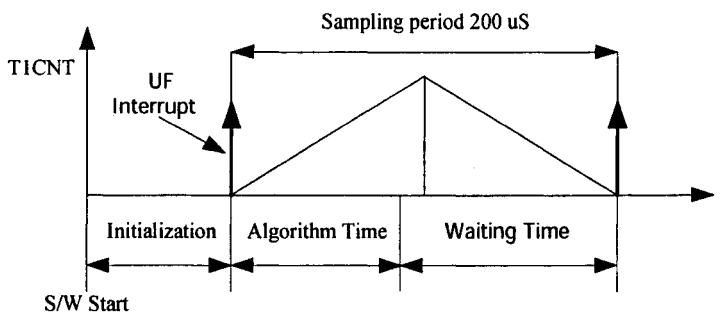


Figure 3-9 The Software Initialization and Operating System

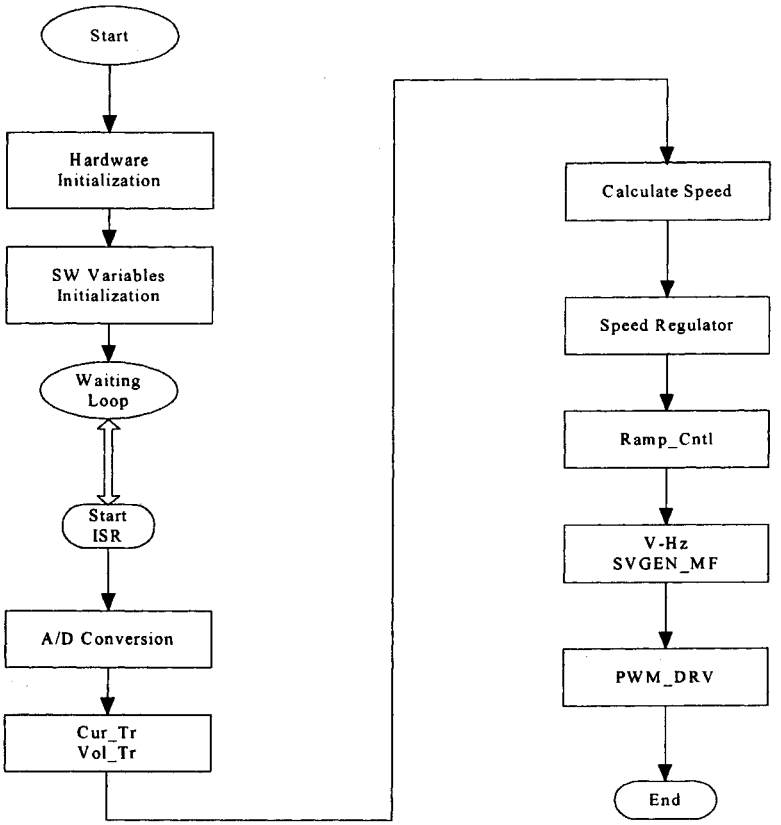


Figure 3-10: V/f Control Software Flowchart

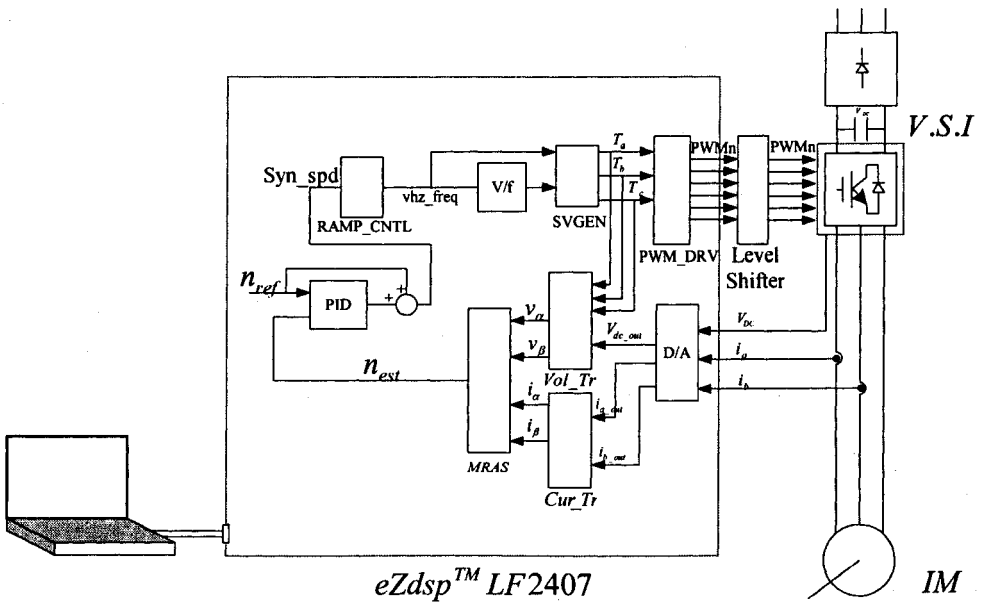


Figure 3-11: DSP Based Instrumentation for the V/f Control Scheme

The software modules for the V/f control system are as follows.

1- A/D ----ADC module converts the stator currents (i_a , i_b) of the induction motor, and the bus DC voltage V_{DC} of the inverter VSI into digital representations.2- Cur_Tr---- Clarke's transformation converts the balanced 3-phase currents into 2-phase (α,β) currents.

3- Vol_Tr----This module calculates three-phase voltages applied to the motor. The three phase voltages are reconstructed from the DC-bus voltage and three switching functions of the upper switching devices of the inverter. In addition, this module also includes the Clarke's transformation that converts three-phase voltages into two-phase stationary (α,β) voltage.

4- MRAS----This module is a speed estimator module for the 3-phase induction motor, based on the reactive power model reference adaptive system.

5- PID----Digital PID controller regulates the slip speed.

6- RAMP_CNTL----This module implements a ramp up and ramp down function to make the output equal to the input in designed ramp time.

7- V/f----This module generates an output command voltage for a specific input command frequency according to the specific V/f profile.

8- SVGEN---- This module calculates the appropriate duty ratios needed to generate a given stator reference voltage using space vector PWM technique. The stator voltage is described by its magnitude and frequency.

9- PWM_DRV---- This module generates the PWM outputs.

10- DATA_LOG---- This module stores the real-time values of two selected variables for displaying in two graphs.

11- SYS_INT----Initialization of system.

3.3.3 Debugging of the Software

In the debugging phase, the incremental-system-build process is used to decompose the debugging task into several small ones. There are totally five phases for debugging in the V/f control system described as follows.

Table3-1 Five Phases for Debugging

```
;************************************************************************

Phase1_inc_build      .set    0       ; V/Hz profile test
phase2_inc_build      .set    0       ; SV_GEN and FC_PWM tests
phase3_inc_build      .set    0       ; Currents and DC-bus volt measurement tests
phase4_inc_build      .set    0       ; Speed measurement and MRAS speed estimator tests
phase5_inc_build      .set    1       ; Sensorless closed-loop V/f system test

;************************************************************************
```

For simplicity, only Phase 4 is explained in the following as an example. The other phase debugging procedure can be found in the main program *vfmars.asm*

Goal of the Phase 4

Assuming phase 3 is completed successfully, the purpose of this section is to verify speed estimation by means of the MRAS technique.

Building the interconnections

Figure 3-12 shows the block diagram in this phase. An open loop V/f control method is employed. The speed of induction motor is controlled by adjusting the reference frequency (ω_{ref}) of the stator voltage.

The V/f module generates an output voltage command for a specific input command frequency according to the specified V/f profile. The module SVGEN calculates the appropriate duty ratios Ta, Tb and Tc needed to generate a given stator reference voltage, with the use of the space vector PWM technique. The stator reference voltage is described by its magnitude and frequency. The module PWM_DRV uses the duty ratio information to generate the PWM signals for the control of the power electronic inverter that drives the induction motor.

The digital-to-analog (D/A) module provides a 3-channel analog-to-digital conversion. The converted results represent the load currents i_a, i_b and the DC-bus voltage V_{DC}. The Clarke's transformation module of Cur_Tr converts the balanced three phase currents into balanced two phase (α,β) quarature quantities i_α and i_β. The module of Vol_Tr calculates the three phase voltages applied to the three-phase induction motor and also converts the three phase voltages into two stationary (α,β) voltages V_α and V_β. The (α,β) voltage and currents V_α, V_β, i_α and i_β, are fed into MRAS module to estimate the motor speed n_{est}. A tacheometer is used to measure the motor speed n_{meas}, which is used to compare with the estimated speed n_{est}. The DATA_LOG module provides a dual memory buffer to display two graphical waveforms in real time. This module can be used to show any interested variables such as i_α, i_β, V_α and V_β or PWM outputs.

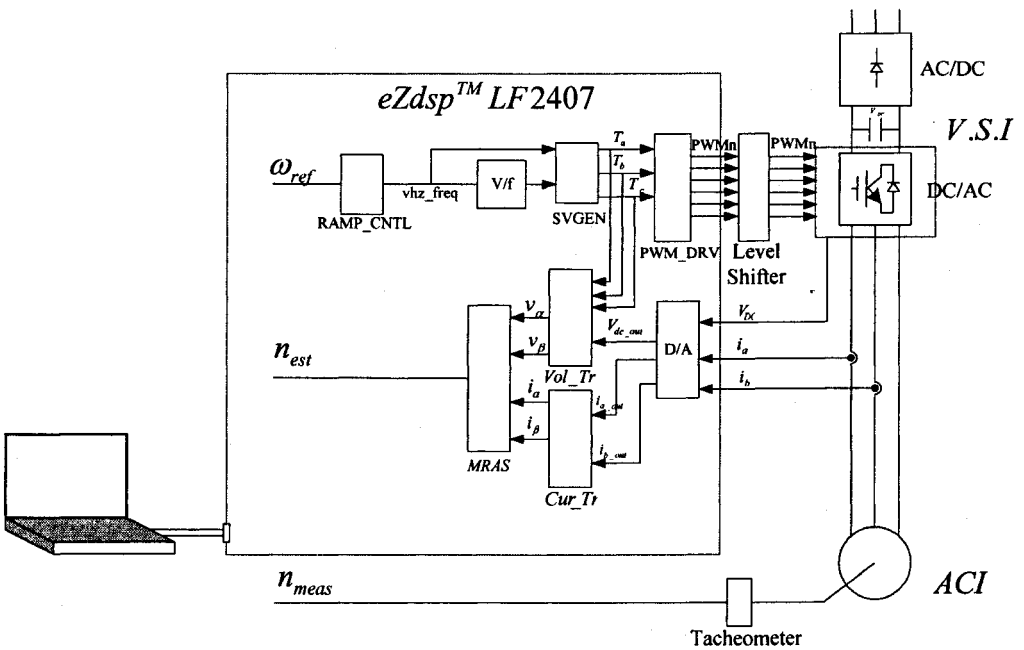


Figure3-12: Experimental System Blocks for the Speed Identification Using MRAS

Debugging in phase 4

- Set ω_{ref} to a suitable value, for example 0.3 pu.

- Set ref_model_ and adt_model_ to 1, and set pi_mras_ to 0.
- Compile/load/run program with real time mode and then increase input AC voltage to generate the appropriate DC-bus voltage.
- Set wr_hat_mras to 0.29 pu (assuming the slip is 0.01pu).
- Compare the reactive power q in the reference model with q_hat in the adaptive model. They should be equal if wr_hat_mras is the actual speed.
- Try different values of ω_{ref} to test q_hat and q.
- Reduce the input AC voltage to zero, halt the program and stop real time mode.
- Set pi_mras_ to 1, compile, load and run program with real time mode and then increase input AC voltage to generate the appropriate DC-bus voltage again.
- Adjust the gains of Kp, Ki_high and Ki_low such that the estimated speed n_{est} with the measured speed n_{mear}.

3.3.4 Other Considerations

3.3.4.1 Numerical Considerations

The numeric format between any two modules is such that 1 bit is dedicated to the integer part and 15 bits are dedicated to the fractional part. This numeric format is denoted by 1.15 or Q15. The resolution for this format is:

$$\frac{1}{2^{15}} = 0.000031$$

With the sign extension mode set, the link between the real quantity and its 1.15 representation is illustrated by Figure 3-13.

A generic x.y format uses x bits for the integer part and y bits for the fractional part. The resolution is 2^{-y}. If z is the per unit value to implement, then its software value is $z \cdot 2^{-y}$ in x.y format. In a specific module, the numeric formats for different variables may be different. The selection ensures that the software values can handle each drive control quantity, not only during the steady state operation but also during the transient operation.

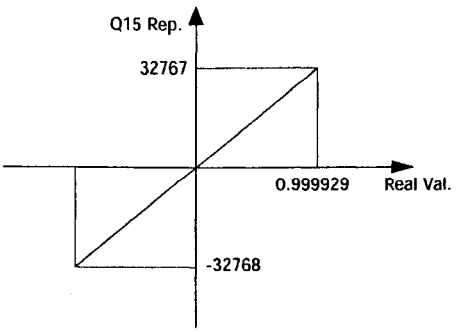


Figure 3-13: Q15 Format Correspondence Diagram

3.3.4.2 The V/f Profile Design

In the software implementation, the stator voltage to frequency ratio is calculated using the rated per unit values of these two variables. For the three speed ranges of the V/f profile, compensation region, linear region and field-weakening region, specific considerations are presented as follows.

The base frequency is specified at 1.1 times the rated frequency (60Hz). This will allow the motor running in the field-weaking region at a speed not more than 1.1times the rated frequency to avoid the possible over speed in the commissioning operation. The cutoff frequency is specified at 0.2 times the base value. Figure 3-14 shows the V/f profile used in the software module.

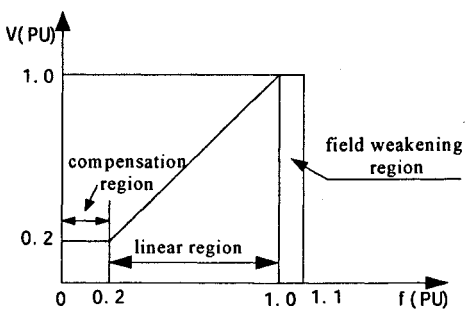


Figure 3-14 V/f Profile Configuration

Based on the V/f profile of Figure 3-14, the output voltages corresponding to a few frequency values are shown in Table 3-2. The maximum output voltage value is assigned to be 32765 to avoid the possible resulting value over 32767 that means a negative value.

Table 3-2 Resulting Output Voltage vs. Frequency for the V/f Profile

Frequency		Output Voltage	
Actual Value(pu)	Q15(Dec)	Actual Value(pu)	Q15(Dec)
0.1	3277	0.2	6553
0.2	6553	0.2	6553
0.3	9830	0.31	10298
0.9	29490	1.0	32765
1.0	32765	1.0	32765

3.3.4.3 Slip Speed PID Controller Design

A conventional propotional-integration-differention (PID) controller shown in Figure 3-15 is employed [10].

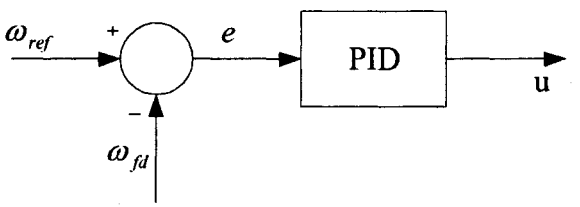


Figure3-15: PID Controller Block Diagram

The equation for PID controller is given as follows:

$$u(t) = K_p e(t) + K_I \int_0^t e(t)dt + K_D \frac{de(t)}{dt} \qquad (3\text{-}5)$$

where:

$u(t)$: the output of PID controller

$e(t)$: the error between the reference and feedback variables.

ω_{ref} : the reference variable

ω_{fd} : the feedback variable

K_P : the proportional gain of PID controller

K_I : the integral gain of PID controller

K_D : the derivative gain of PID controller

Applying the Laplace transform to equation (3-5) with zero initial condition, this yields:

$$U(s) = \left[K_P + \frac{K_I}{s} + K_D s \right] E(s) \qquad (3\text{-}6)$$

By substituting $s = \frac{1 - z^{-1}}{T}$, where T is the sampling period, Eq. (3-6) can be transformed to be a difference equation:

$$U(z) = \left[K_P + \frac{K_I T}{1 - z^{-1}} + \frac{K_D}{T}(1 - z^{-1}) \right] E(z) \qquad (3\text{-}7)$$

Eq. (3-7) can be rewritten in the discrete time-domain as,

$$u(k) = u(k-1) + \left(K_P + K_I T + \frac{K_D}{T} \right) e(k) - \left(K_P + 2\frac{K_D}{T} \right) e(k-1) + \frac{K_D}{T} e(k-2) \qquad (3\text{-}8)$$

where $u(k)$, $u(k-1)$ are the k^{th} and k-1 th output of PID controller, $e(k)$, $e(k-1)$ and $e(k-2)$ are the k^{th}, k-1th and k-2 th error between the reference and feedback variables, respectively.

3.3.4.4 Load Design

Experiments are performed to study and verify the steady state and transient performance of the complete control system designed in this project. During the tests, the motor has been mounted on to a test bench connecting with a set of self-excited generator as its load. Figure 3-16 shows the connection.

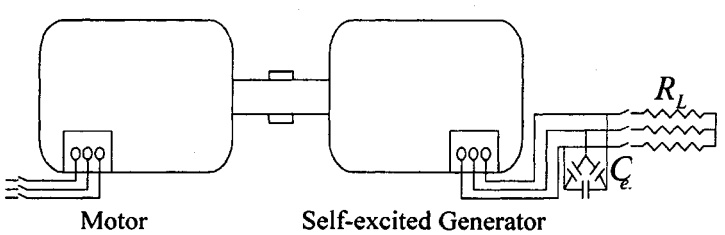


Figure3-16: Motor and Self-excited Generator Unit

The self-excited generator is the same type of induction machine as the prime motor to be controlled. The generator connects with a bank of capacitors and resistors and acts as a load to the prime motor. Its ratings are given in Appendix A.

When the induction machine is driven by the prime motor, the residual magnetism in the rotor produces a small voltage in the stator windings, which causes a current to flow into the capacitor. This resulting current provides a positive feedback that causes a further increase in the voltage. The voltage is eventually limited due to the magnetic saturation of the machine. The load of the prime motor will be changed when the resistance R_L is changed.

In Figure3-16, the bank of three-phase capacitors is delta-connected. The rating of each capacitor is selected as follows: C_{ex} is 56μF and its maximum line to line voltage is 370V.

In this chapter, the speed sensorless V/f control has been developed with the MRAS speed estimator. The hardware and the software have been designed to implement the control schemes.

Chapter 4

Field Oriented Control Using MRAS

The field oriented control (FOC) of induction motors is a vector control method based on transformations of a three-phase time-dependent motor model into a two-coordinate (d and q coordinates) time invariant model. The model rotates at the rotor flux speed or synchronous speed. These transformations lead to a structure similar to that of a self-excited DC motor drive control, of which the flux and the torque of the motor can be controlled separately.

In this chapter, the space vector PWM is introduced first. This vector is used to produce appropriate signals that drive the voltage source inverter (VSI). Then the vector FOC of induction motors using DSP eZdsp^{TM} LF2407 as controller is discussed in detail. An experimental system is built. This chapter presents the technique used to the implement the proposed FOC speed sensorless control scheme for induction motors using the MRAS estimator.

4.1 Space Vector Pulse Width Modulation

The space vector pulse-width-modulation (SVPWM) is employed to control the switching of the VSI controller. This special switching scheme for the power devices produces 3 pseudosinusoidal currents in the stator phases. The SVPWM modulation generates smaller harmonic distortion in the output voltages or currents in the windings of the motor and provides more efficient use of the DC supply voltages as compared with the direct sinusoidal modulation technique [11].

Figure 4-1 shows a basic 3-phase voltage source inverter circuit. Its input is a DC voltage V_{DC}, and its outputs are 3-phase AC voltages. The inverter has six IGBT switches, which follows the conditions.

- Three of the switches are always ON and other three are always OFF.
- The upper and the lower switches of the same leg are driven with two complementary gate signals with dead band to avoid any short-circuit condition occurred in the inverter.

There are eight (2^3) possible combinations for the switch commands, which determine eight phase voltage combinations. The results are eight basic vectors including six non–zero vectors and two zero vectors in (α,β) stationary reference frame as shown in Figure 4-2. For example in Figure 4-2, 100 represents the switch command, which means the status of the switches, **a**, **b** and **c**, are ON, OFF and OFF.

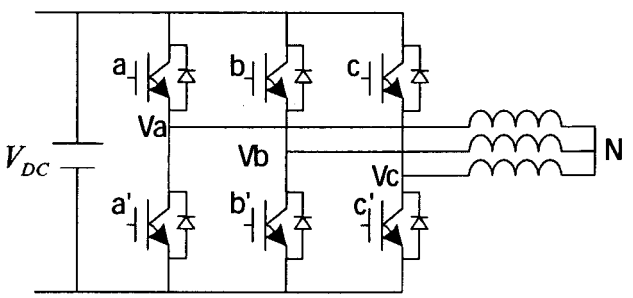


Figure 4-1 Three-phase Voltage Source Inverter

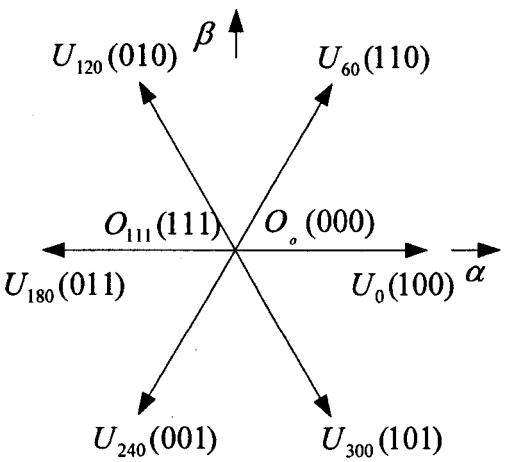


Figure 4-2: Basic Space Vectors

The objective of the space vector PWM technique is to represent a given stator reference voltage vector U_{out} by combination of the switching pattern corresponding to the basic space vectors. The vectors divide the plane into six sectors. Depending on the sector that the reference voltage is in, two adjacent vectors are chosen. The binary representations of two adjacent basic vectors differ in only one bit. Therefore, only one of the upper transistors switches when the switching pattern moves from one vector to the adjacent one. The two vectors are time weighted in a sample period T to produce the desired output voltage.

For example, assuming that the reference vector U_{out} in the sector shown in Figure 4-3, the output voltage U_{out} is represented by two vectors U_0 and U_{60}. Therefore,

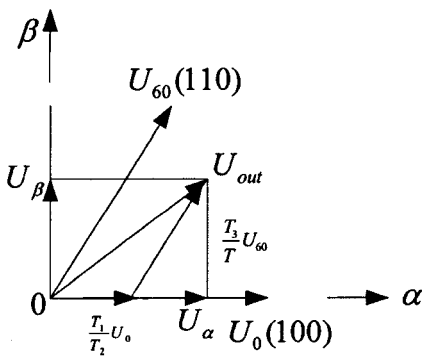


Figure 4-3: Reference Vector as a Combination of Adjacent Vectors

$$T = T_1 + T_3 + T_0 \qquad (4\text{-}1)$$

$$U_{out} = \frac{T_1}{T}U_0 + \frac{T_3}{T}U_{60} \qquad (4\text{-}2)$$

where T_1 and T_3 are the dwell times for the vectors U_0 and U_{60}, respectively. T_0 is the time duration for which the null vector is applied. When U_{out} and the sample period T are known, the time duration for each switching combination can be determined by follow equations:

$$T_1 = T\, m_a \sin(\frac{\pi}{3} - \theta) \qquad (4\text{-}3)$$

$$T_3 = T\, m_a \sin\theta \qquad (4\text{-}4)$$

$$T_0 = T - T_1 - T_3 \qquad (4\text{-}5)$$

where m_a is the modulation index. $m_a = \frac{\sqrt{3}U_{out}}{V_{DC}}$.

The generated space vector PWM waveforms are symmetrical with respect to the middle of each PWM period [8]. The diagram 4-4 shows a typical seven-segment switching sequence for the reference vector U_{out} in the example presented above.

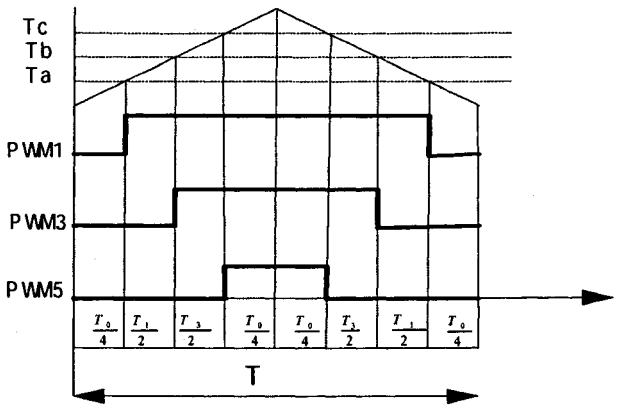


Figure 4-4: Pattern of SVPWM in the sector contained by U_0 and U_{60}

4.2 Background of FOC Control of Induction Motor

The Field Orientation Control (FOC) consists of controlling the stator currents of an induction motor. With Clarke and Park transformations, a three-phase time and speed dependent model can be transformed into a two co-ordinate (d and q co-ordinates) time invariant model. These transformations resolve the stator currents into two components: the flux producing component i_{sd} and the torque producing component i_{sq}.

As in the (d, q) reference frame, the rotor flux ψ_r and torque T_e have relationship with

$$\psi_r \propto i_{sd} \qquad (4\text{-}6)$$

$$T_e \propto \psi_r i_{sq} \qquad (4\text{-}7)$$

the flux producing component i_{sd} and the torque producing component i_{sq} as follows. By maintaining the amplitude of the rotor flux ψ_r at a fixed value, there is a linear relationship between the torque T_e and the torque component i_{sq}. The torque can be controlled by adjusting the torque component of the stator current vector. In other words,

if i_{sd} is kept constant, i_{sq} can be independently controlled. Therefore the torque of the induction motor can be controlled.

Both i_{sd} and i_{sq} have only DC components in steady state because the relative speed with respect that of the rotor field is zero. Therefore they are ideal for use as control variables. The field oriented control needs two constants as input references: the torque-producing component i_{qref} and the flux-producing component i_{dref}. As the FOC is based on the three-phase to two-phase transformation, the control structure handles instantaneous values. This achieves an accurate control in both the steady state and the transient operation.

4.3 The Basic Scheme for the FOC

Figure 4-5 shows the basic scheme of the motor torque control using the FOC control [13].

Two motor phase currents are measured and fed to the Clarke transformation module. The output of this module becomes input of the Park transformation module that produces the currents i_{sd} and i_{sq} in the (d, q) rotating reference frame. These (d,q) currents are compared to the references i_{dref} (the flux reference) i_{qref} (the torque reference). The outputs of the current regulators are applied to the inverse Park transformation. The outputs of this transformation are v_α and v_β which are the components of the stator vector voltage in α, β stationary orthogonal reference frame. v_α and v_β are also the inputs of the space vector PWM. The outputs of this control block provide the signals that drive the inverter. Both Park and inverse Park transformations need the data of the rotor flux position.

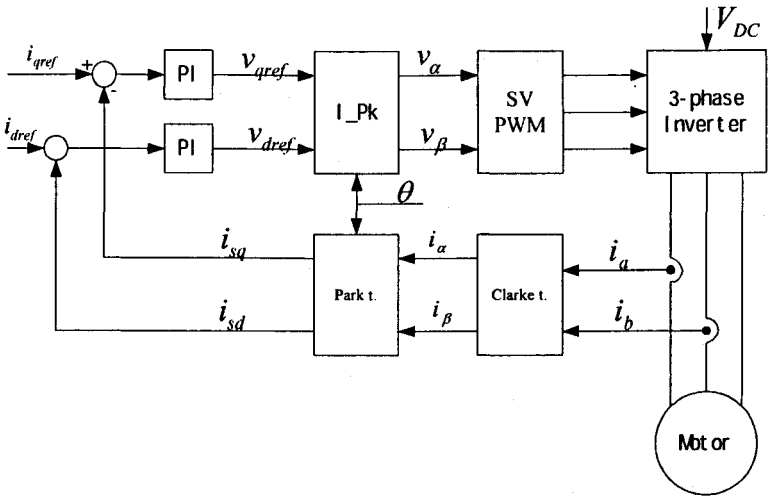
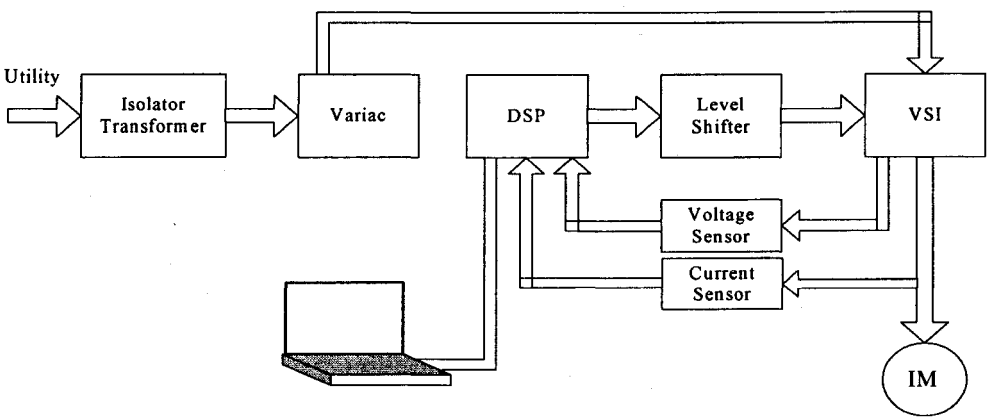


Figure 4-5: Basic Scheme of FOC for AC Motor

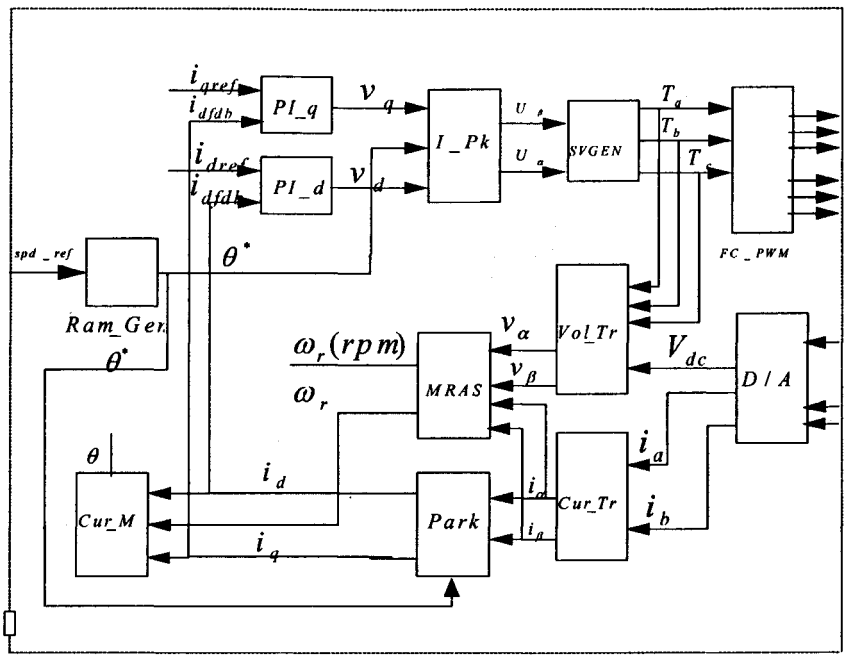
4.4 Proposed Speed Sensorless FOC Control Scheme Using MRAS

The conventional FOC control scheme with a speed sensor is shown in Figure1-2. In this scheme, a speed sensor is used to measure the rotor speed. The speed is fed to a speed regulator comparing with the user designed reference speed. The output is used as torque command i_{qref}. Rotor flux position θ is estimated by using the current module of Cur_M [11].

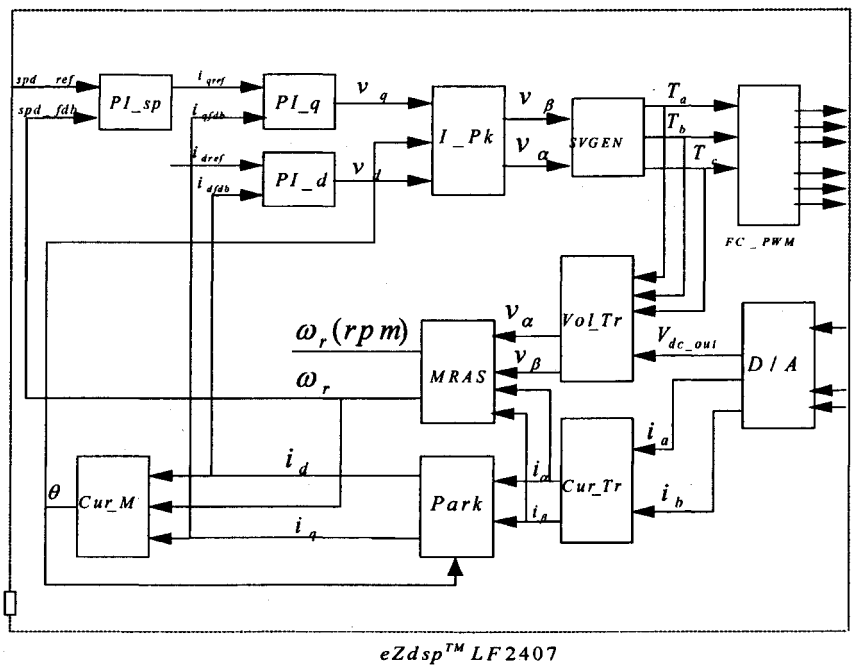
The speed of the induction motor can be computed using the MRAS estimator. Figure 4-6 shows the proposed speed sensorless FOC control scheme using the MRAS speed estimator.



(a) Hardware Configuration



(b) Software Module Configuration of Open loop FOC Control



(c) Software Module Configuration of Closed Loop FOC Control

Figure 4-6: FOC Control Scheme Using MRAS Speed Estimator

Figure 4-6 (a) shows the hardware configuration, which is similar to that discussed in Section 3.3.2. Figure 4-6 (b) shows the software module configuration of the open-loop FOC control. Figure 4-6(c) shows the proposed closed-loop FOC control of induction motors proposed in this project.

To implement the closed-loop control shown in Figure4-6(c), the two-phase currents i_a and i_b from the output of A/D module are fed to the Clarke transformation module *Cla_Tr* which is used to convert three phase stationary frame to two phase (α, β) stationary frame. The outputs are i_α and i_β.

These two α, β current components are fed to the input of the *MRAS* module that estimates the rotor speed. This module also need the two-phase (α,β) stationary voltage v_α and v_β from the outputs of the module *Vol_Tr*. The module *Vol_Tr* calculates the three phase voltages and converts them into two stationary (α-β) voltages using the DC-bus voltage V_{DC} and three switching functions of the upper power switching devices of the inverter.

The MRAS estimated speed ω_r is fed to *PI _sp* regulator module where it is compared with the reference speed *spd _ref*. The output of the speed regulator is the quadrature-axis stator current reference i_{qref} in the (d, q) frame.

The outputs, i_α and i_β, are also inputs of the Park transformation that produces the current in the (d, q) rotating reference frame. The components i_d and i_q are compared to the references i_{dref} (the flux reference) and i_{qref} (the torque reference) using *PI_q* and *PI_d* module respectively. The outputs of the PI controllers are applied to the inverse Park module *I _ Pk*.

The outputs of module I_Pk are v_α and v_β which are the components of the stator vector voltage in (α,β) stationary orthogonal reference frame. These are the inputs of the space vector PWM. The outputs of this block are the signals that drive the inverter.

Both Park and inverse Park transformations need the rotor flux position θ. The position is calculated by the module Cur_M. The estimated speed ω_r, the components i_d and i_q are inputs to this module.

Figure 4-6(b) shows the open-loop FOC control of rotor speed. This is a necessary commissioning phase to complete the proposed close-loop speed sensorless FOC control. In this phase, the estimated speed ω_r is not fed to the regulator module PI_sp. The reference rotor flux position θ^* is generated by the module Ram_Gen using a given reference speed.

4.5 Software Implementation of the Proposed FOC Scheme

The software organization is similar to that of V/f control strategy described in the Section 3.3.3. A modular strategy is adopted to develop the software package written in assembly language [15].

The DSP Controller Full Compare Unit is used to generate the necessary pulse switching signals to the level shifter board. It is programmed to generate symmetrical complementary PWM signals at a frequency of 5 kHz, with TIMER1 as the time base and with the dead time of 1.2μs. The sampling period (T) is 200 μs. Figure 4-7 illustrates the software flowchart. The program is listed in Appendices C.

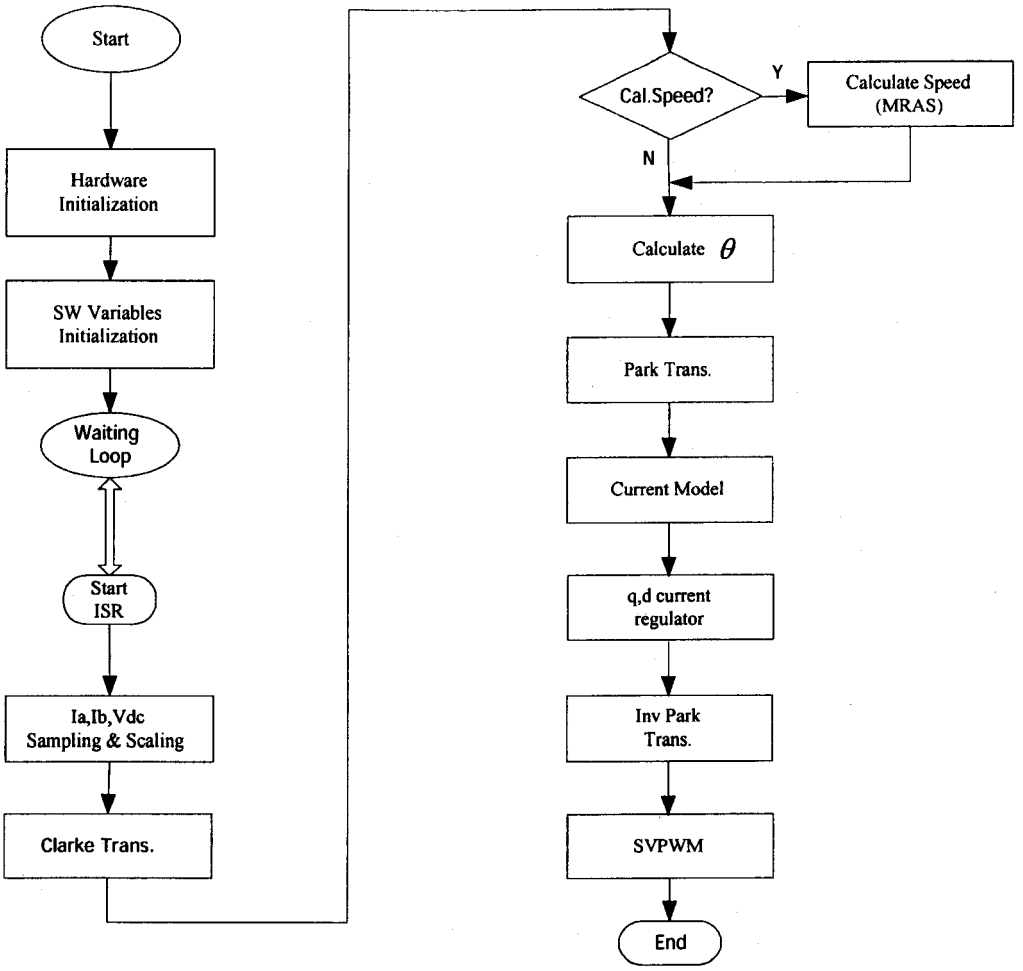


Figure 4-7: Sensorless Speed FOC Control Software Flowchart

In this chapter, the speed sensorless FOC control has been developed with the MRAS speed estimator. The hardware and the software have been designed to implement the control schemes.

Chapter 5

Experimental Results

In this chapter, experiments are carried out to verify the capability of the speed sensorless control schemes proposed in this project. Typical results are given to demonstrate the performance of the control system. First, the test of speed identification using MRAS estimator is presented in Section 5.1. Then the V/f control behavior in both open loop and closed loop operating are described in Section 5.2 and 5.3. Finally, the open loop FOC control performance is discussed in section 5.4.

5.1 Speed Identification Using the Proposed MRAS Estimator

In order to verify the proposed reactive power MRAS speed estimator, an experimental motor drive system shown in Figure3-12 was built and tested. The results are described in this section.

Result 1: No load test

Table 5-1 shows the relative speed error at different speeds. In this table, the relative speed error is defined as follows.

$$e(\%) = \frac{n_{ref} - n_{est}}{n_{ref}} 100\% \qquad (5\text{-}1)$$

In above expression, the measure speed n_{ref} is measured by a tachometer, and the estimated speed n_{est} is estimated by MRAS. Figure 5-1 (a) and Figure 5-1 (b) also show the results.

Table 5-1 Speed Identification Using MRAS

Reference speed n_{ref} (rpm)	Estimated speed n_{est} (rpm)	Relative Error e(%)
177	152	14.1
296	273	7.8
593	573	3.4
890	881	1.0
1067	1058	0.8
1187	1179	0.7
1306	1301	0.4
1484	1483	0.06
1662	1663	0.06
1722	1724	0.12
1752	1753	0.06

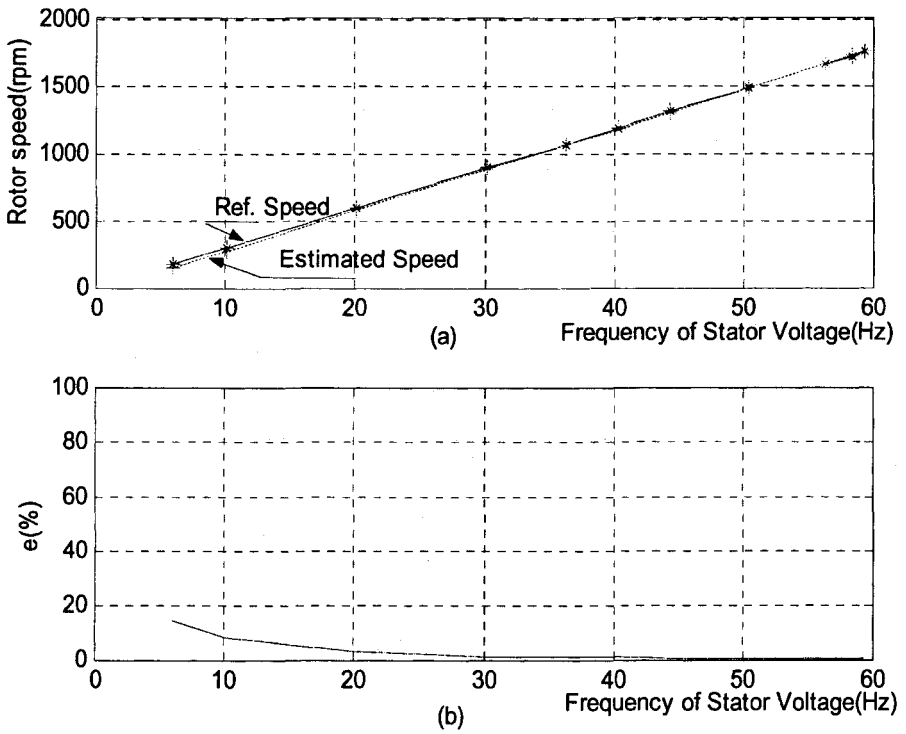


Figure 5-1: (a) Speed Identification Using MRAS (b) Relative Error

Results 2: Load test

In order to do the load test, the induction motor is mounted on to a test bench connecting with a set of self-excited generator as its load as shown in Figure 3-16. When the load

resistance R_L is changed, the load of the prime motor will change, and therefore the stator current will change. During the test, R_L was changed three times such that the stator current changes from 2.5A to 3A, then from 3A to 4.5A and from 4.5A to 6A. In each time the rotor speed is measured by a tachometer and is estimated by MRAS. The results are listed in Table 5-2, as well as in Figure 5-2 respectively.

Table 5-2: Speed Identification Test Result 2

Load Case (stator current, A)	Measure speed n_{meas} (rpm)	Estimated speed n_{est} (rpm)	Relative Error e(%)
2.5	1484	1483	0.07
3.0	1476	1477	0.07
4.5	1462	1460	0.14
6.0	1443	1449	0.42

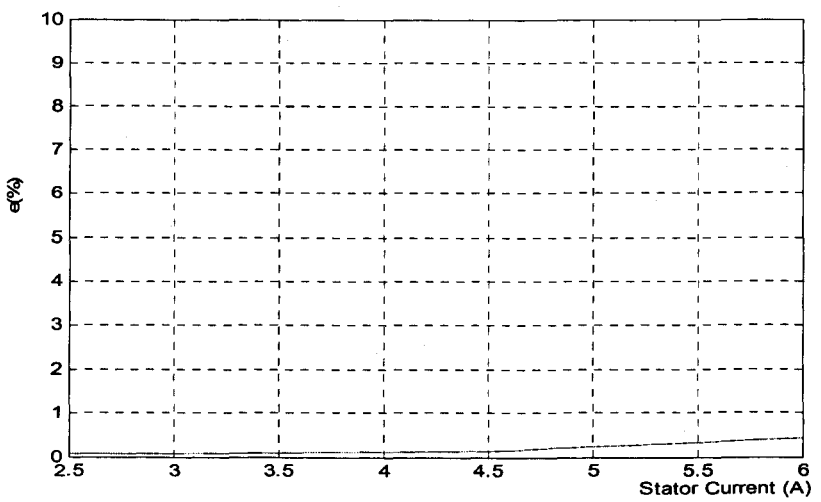


Figure5-2 Relative Error e(%) with Different Loads

The above results show that the MRAS speed estimator performed well in estimating the motor speed, especially when the frequency of stator voltage is above 50 Hz. Since a motor usually runs under the rated frequency, the MRAS speed estimator can be used to implement the speed sensorless motor control schemes.

5.2 Open Loop Speed V/f Control

With the experimental system shown in Figure3-12, an open-loop speed V/f control is carried out to check whether the relationship between the stator voltage and frequency agrees with the designed V/f profile of Figure 3-14. The software modules were checked and modified in this phase. During the tests, the stator current keeps constant except in the compensation region and the field-weakening region. The results are shown in Figure 5-3.

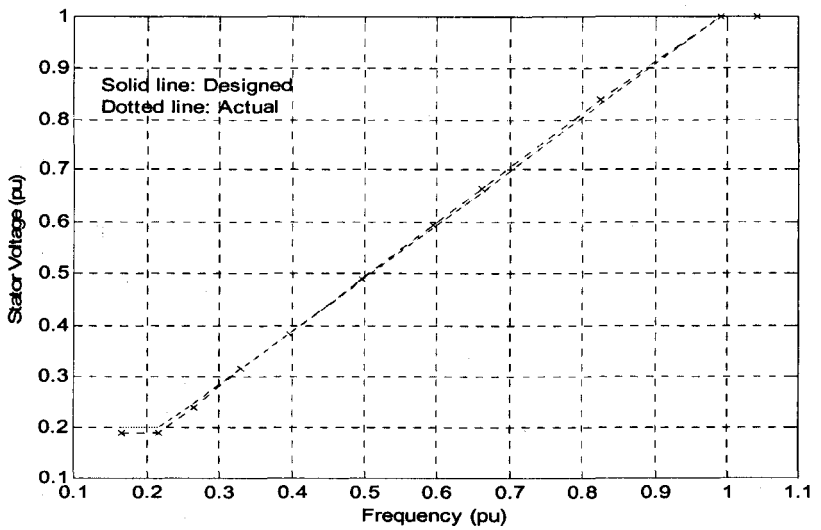


Figure5-3: Stator Voltage Vs Test Frequency

This result shows that designed V-Hz relationship has been verified. Both hardware and software performance are confirmed.

5.3 Closed-loop Speed V/f Control

The control was operated in the closed speed loop mode with the circuit configuration shown in Figure 3-11. Two types of tests were carried out to verify the performance of the control scheme using the speed sensorless MRAS estimator designed in this project. One is a step change in load and another is an abrupt change in reference speed.

Result 1: Step Change in Load

A test was conducted with a step change of load. At first, the motor runs at a reference speed with the self-excited generator unit disconnecting from the resistive load as shown

in Figure 3-16. Then the load is connected to the generator unit. After the motor runs at the speed in a steady state, the load is suddenly disconnected with the generator unit. Table 5-3 shows the motor speeds measured by speed sensor and by MRAS speed estimator in steady state in the test. Figure 5-4 shows the waveform of the stator current i_a in the dynamic state and Figure 5-5 shows the waveforms of the stator currents in the steady state. The results show that the MRAS has good performance in the closed loop speed V/f control.

Table 5-3: Load Test in Closed Loop Speed Control

Item		No-load	On-load	No-load
Motor Speed(rpm)	Tachometer	1365	1364	1365
	MRAS estimator	1365	1364	1365

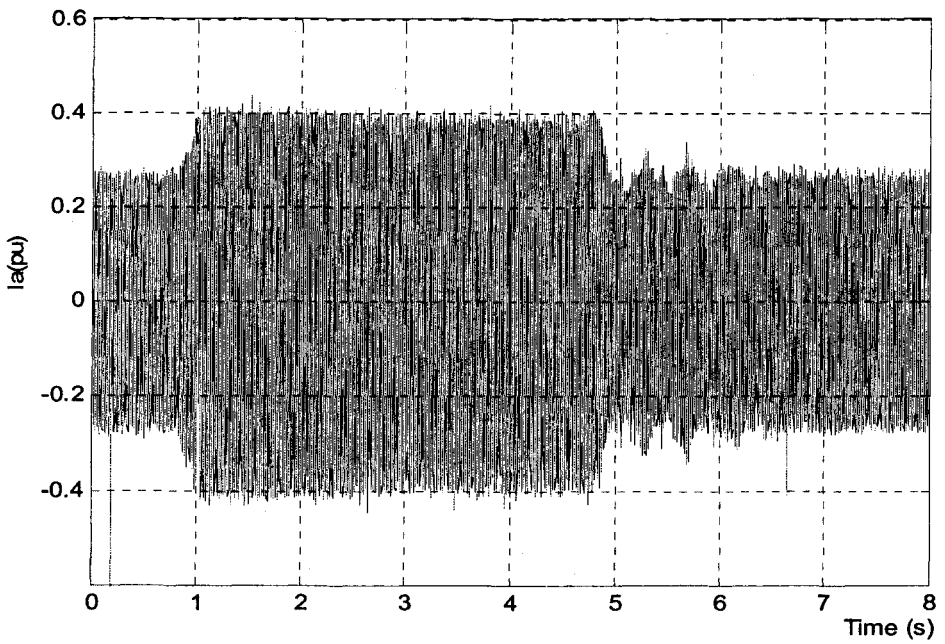
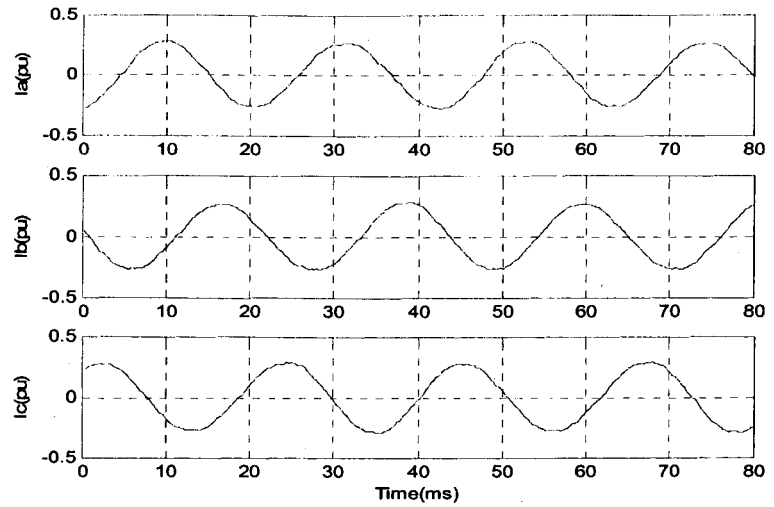
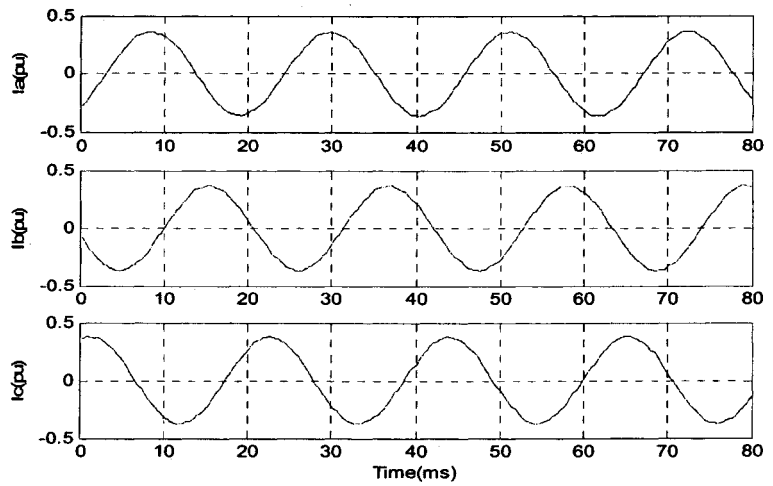


Figure5-4: The Waveform of Stator Current i_a in the Dynamic State



(a) No-load



(b) On-load

Figure 5-5: Three-phase Stator Steady Currents

Result 2: Step Change in Speed

Tests were conducted with several step changes of the reference speed. During this test, the motor runs at a reference speed with the self-excited generator unit shown in Figure3-16, and then the reference speed is changed. After the motor runs in a steady state, the reference speed is changed back again. Table 5-4 shows the motor speed changes during the test. Figure 5-6 shows the waveform of the stator current i_a in the dynamic state. Figure 5-7 shows the waveforms of three-phase stator currents in the steady states. The result shows even with 40.7% speed change the control can achieve steady state after a

period of dynamic state. Therefore, the control designed in this project has good performance under abrupt changes of the reference speed.

Figure 5-4 Speed Change Test in Closed-loop Mode

Item		No-load	On-load	No-load
Motor Speed(rpm)	Tachometer	881	1485	881
	MRAS estimator	881	1485	881

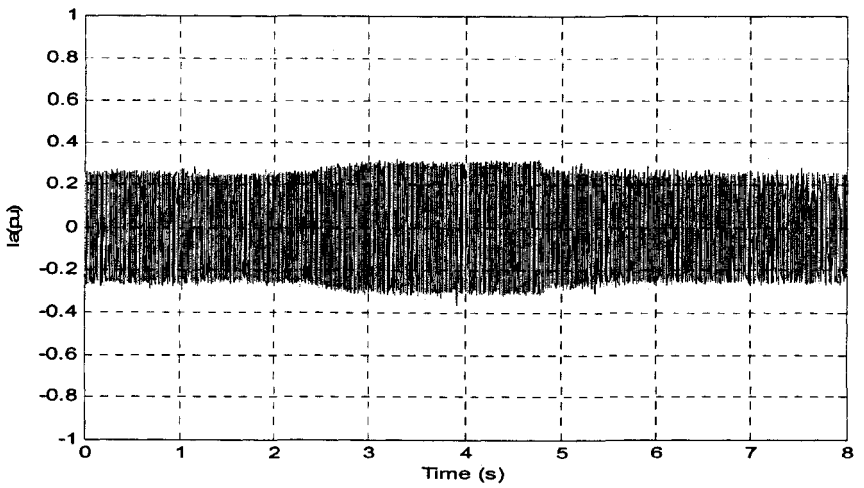
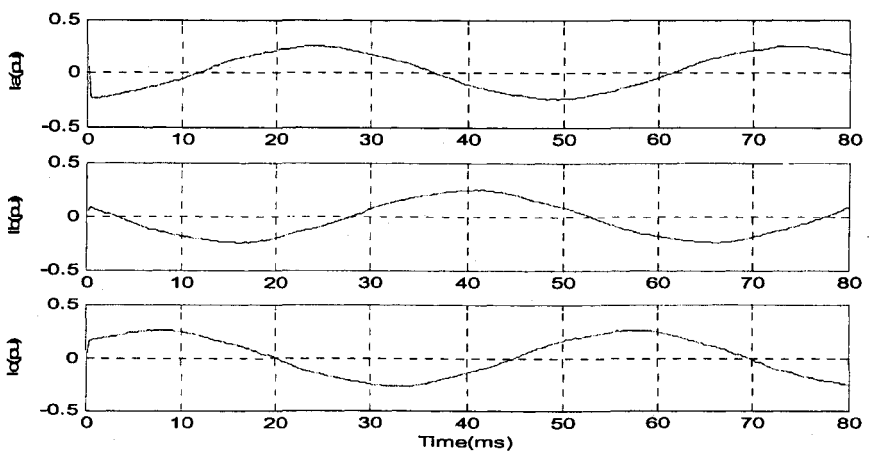


Figure 5-6: The Waveform of Stator Currents i_a in the Dynamic State When the Reference Speed Changes



(a)

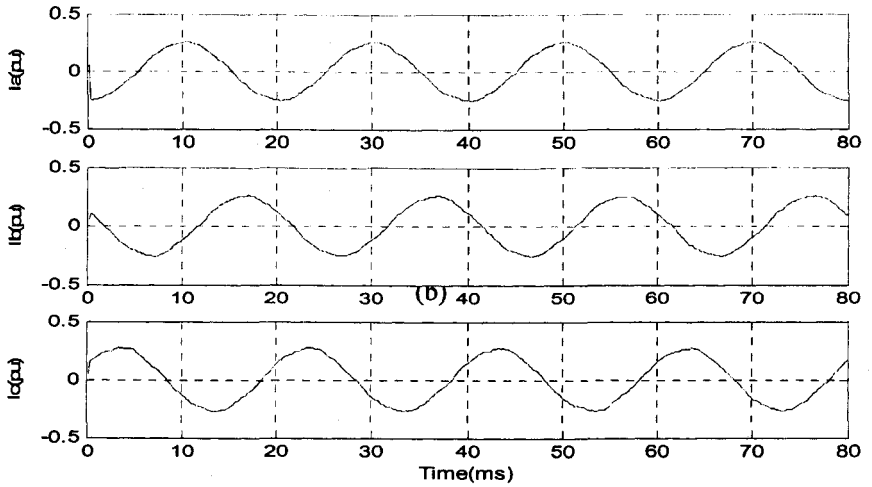


Figure 5-7: Three-phase Stator Steady Currents
(a) Rotor speed =881rpm (b) Rotor speed=1485 rpm

5.4 Open Loop Speed FOC Control

Tests were carried out to check the correct operation of *PWM* generator module *SVGEN*, and the current regulators PI_d and PI_q under no-load condition. The speed estimator designed in this project was also checked. The control scheme has been shown in Figure 4-6(b).

During the test, the reference frequency is changed to different values while the reference flux-producing component i_{dref} is kept at 0.152pu and the reference torque-producing component i_{qref} is kept 0. The results of a test are presented as follows.

Step 1:

The reference speed spd_ref is set to 0.776 (pu). In this case, the motor runs at a speed of 1358 rpm. Figure 5-8 shows the waveforms of the referent components of the stator current and their responding feedback. Figure 5-8 presents waveforms of actual rotor speed and that of estimated speed using the MRAS speed estimator designed in this project.

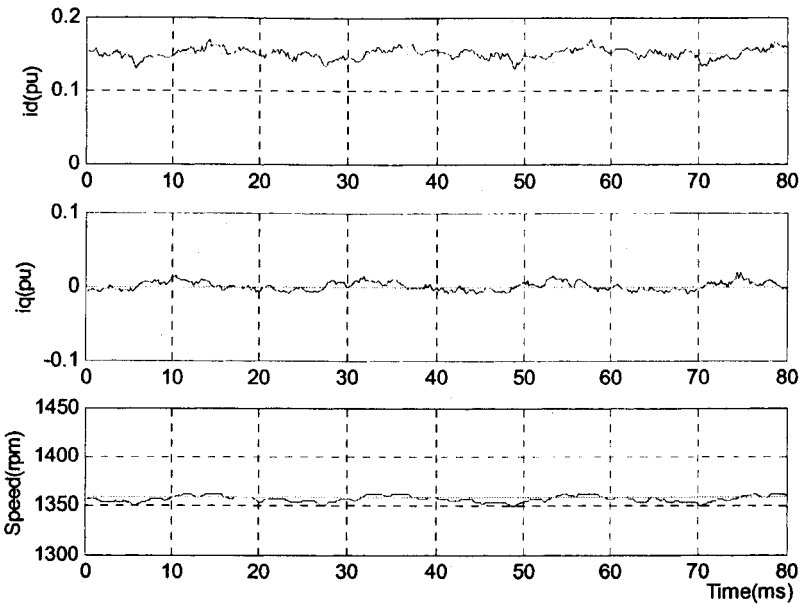


Figure 5-8: Waveforms of i_d, i_q and that of Rotor Speed in Case of spd_ref =0.776(pu)

Step 2:

The reference speed is changed from 0.776pu to 0.970pu. In this case, the motor runs at 1710rpm.

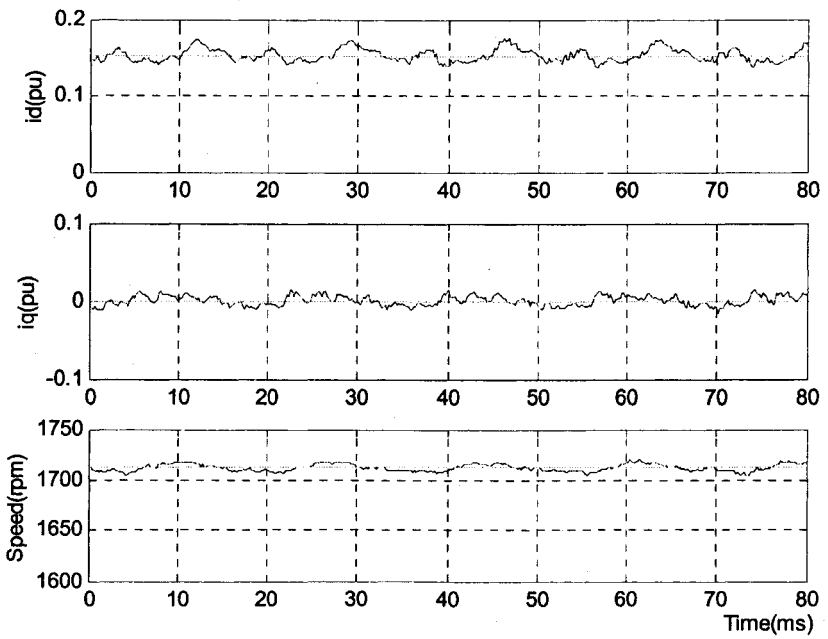


Figure 5-9: Waveforms of i_d, i_q and that of Rotor Speed in Case of spd_ref =0.970(pu)

The results are shown by Figure 5-9, which illustrates waveforms of referent components of the stator current and their responding feedback. This figure also shows the waveforms of actual rotor speed and that of estimated speed using MRAS speed estimator. Figure5-10 shows the duty ratios T_a, T_b and T_c of $PWM1$, $PWM3$ and $PWM5$, respectively.

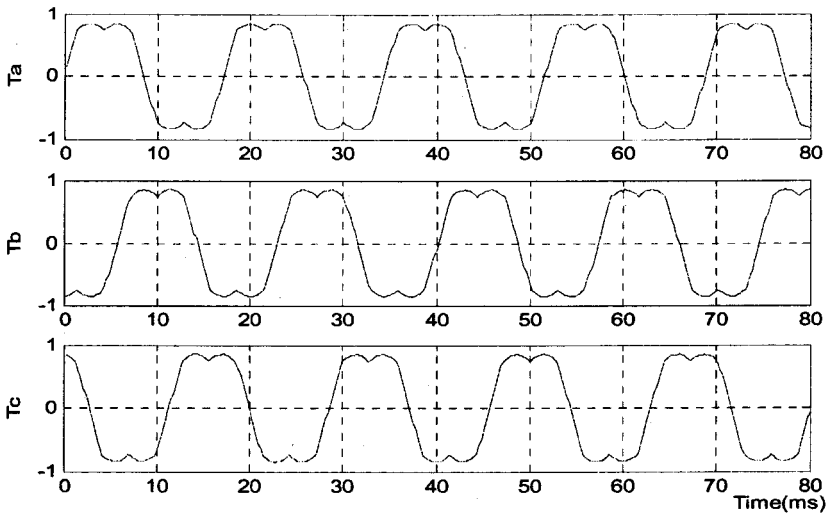


Figure10 duty ratios T_a, T_b and T_c

As shown in Figure 5-8 to Figure 5-10, the following results are obtained.

1) The feedback flux-producing component i_d and the feedback torque-producing component i_q are close to DC currents, which can be used to control the flux and the torque of the induction motor.
2) The performance of the speed control, the flux-producing component i_d control and the torque-producing component i_q are good.
3) The duty ratios T_a, T_b and T_c are correct. T_a leads T_b 120 degrees and T_b leads T_c 120 degrees. The outputs of PWM generator module $SVGEN$ can be used to drive the voltage source inverter.

5.5 Summary

The experimental results show the high performance of the designed reactive power MRAS speed estimator designed in this project and the speed sensorless V/f control scheme proposed in this research using the MRAS speed estimator. The designed V/f profile has been verified. Even with a large sudden change of the load or the reference speed, the system can achieve the expected steady state in a short period of time. This type of change is one of the worst conditions, as normally the maximum typical change is in the range of 10% to 15%.

The experimental results also show the high performance of current regulator design in the proposed speed sensorless FOC control scheme. Both the flux current component and the torque current component can be controlled separately. The experiments have shown the MRAS as an excellent speed estimator.

Chapter 6

Conclusions

This project report presents an efficient speed sensorless scalar V/f control and the speed sensorless vector FOC control of induction motors using MRAS speed estimator. The MRAS speed estimator shows very good performance. The speed sensor, which usually is needed in traditional variable speed drives of induction motors, is eliminated with the use of MRAS speed estimator.

A dynamic model is used to design the drive for the induction motor. Equations are developed to describe the principle of the MRAS speed estimator, which shows the speed estimation system is completely robust to the stator resistance variations due to change of temperature.

The modularity strategy used in this project design has been shown great flexibility to develop the software packages for the control of induction motors. The experimental system is built by using DSP eZdsp^{TM} LF2407 as controller and verified experimentally.

6.1 Main Work in This Project

The main work that has been accomplished in this project is summarized as follows:

1- The MRAS control method to induction motor drives has been developed.

   In this project, the reactive power MRAS technique has been successfully applied to induction motor drives as a speed estimator. An appropriate induction motor model for the design of the drive has been established. Equations have been developed for the design of the MRAS speed estimator. An experimental system has been built to

verify the performance of the MRAS as a speed sensor. The experimental results have shown the very good performance from the MRAS speed estimator developed in this project. The speed sensor, which usually is needed in traditional variable speed drives of induction motors, has been eliminated with the use of MRAS speed estimator.

2- The control structure for speed sensorless control of induction motors has been developed.

Both the speed sensorless V/f control and the FOC control schemes have been developed with the MRAS speed estimator. The hardware and the software have been designed to implement the control schemes. All required hardware components, including DSP control processor, VSI power electronic inverter, voltage sensor, current sensor, level shifter and the induction motor have been investigated or designed. These include

a). Made the outputs of voltage sensor box and current sensor box to match the input requirements of the DSP.
b). Designed a level shifter to amplify the DSP output PWM signals to the corresponding level for IGBT power module.
c). Conducted tests to determine the parameters of the induction motor to be used in the experimental verification.
d). Designed a self-excited generator unit as the load of the induction motor, which has shown great flexibility during the tests.

The DSP is the core of control system. All the control operation is implemented using software in the DSP TMS320 LF2407.

3- Program software modules have been developed with use of modularity strategy.

Following the modularity rule, the software has been programmed in modules to implement the software control. The modules are written in assembly language.

DLOG_VIEW module is employed to display two graphical waveforms in real time. The software modules meet the requirements of flexibility, compatibility and expandability.

4- The hardware has been fully checked and the software has been completely debugged with the use of incremental system builds process.

This process has decomposed the debugging task into several small ones. This has been greatly shorten the developing time and has reduced the chance of making mistake.

5- The experimental system has been built and various tests have been carried out to verify the control developed in this project.

The test items have included the speed identification by using the designed MRAS speed estimator, for both open-loop and closed-loop V/f control under on-load condition, and the open-loop FOC control. The experimental results have shown the high performance of the designed reactive power MRAS speed estimator and the proposed speed sensorless V/f control scheme using MRAS speed estimator. The designed V/f profile has been verified. Even with a large sudden change of the load or the reference speed, the system can achieve the expected steady state in a short period of time. The experimental results have also confirmed the feasibility of the proposed speed sensorless FOC control scheme using MRAS speed estimator. The current regulators have been successfully designed. Both the flux-producing current component and the torque-producing current component can be controlled separately.

6.2 Suggestion for the Future Work

This report has provided detailed information in the speed sensorless control of induction motors using the MRAS speed estimator. The following gives recommendations for future.

1- Improvement of the proposed close-loop speed sensorless FOC control scheme.

At present, the current regulators have been designed. Both the flux current component and the torque current component can be controlled separately. However, the estimated rotor flux position still shows a small difference with the referent rotor flux position that causes a small oscillation during the tests in the speed closed loop control mode. Therefore, a further study is needed to achieve an absolutely stable speed-close-loop control.

2- Low-speed sensorless control of induction motor with MRAS speed estimator.

The designed reactive power MRAS speed estimator has a very good performance in estimating the motor speed in rated speed range but slightly poor when the stator frequency is under 2Hz. Different self-adaptive sets of initial constants (i.e., K_1 to K_7) may be needed to improve this condition.

3- Simulation work.

Detailed simulation is needed to verify all aspects of the MRAS speed estimator designed in this project.

Bibliography

[1] Kaushik Rajashekara et " Sensorless Control of AC Motor Drives", IEEE Press, Piscataway, NJ, 1996

[2] T. Ohtami et "Vector control of induction motor without shaft encoder", IEEE Trans. Indus. Appli. Vol.28, No.1, 1992.

[3] Colin Schander , "Adaptive speed induction motors without rotor rotational transducers. IEEE Trans. Indus. Appli. Vol.28, No.1, 1992.

[4] Joachim Holtz, "Sensorless speed and position control of induction motors", 27'IECON, 2001.

[5] Y. Hori and T. Umeno, "Implementation of robust flux observer based field orientation controller for induction machines", In Proc. 1989 IAS.

[6] A.Abbondanti and M.B. Brennen, "Variable speed induction Motor Drives Use Electronic Slip Calcutator Based on Motor Voltage and Currents," IEEE Trans. Indus. Appli. Vol.IA-11, No.5, 1975.

[7] R. Jortten and G. Maeder, "Control Mthods for Good Dynamic Performance Induction Motor Drives Based on Current and Voltage as Measured Quantities," IEEE Trans. Indus. Appli. Vol.IA-19, No.3, 1983.

[8] F.Z Peng and T. Fukao, "Robust speed identification for speed –sensorless vector control of induction motors", IEEE Trans. Indus. Appli. Vol.28, No.1, 1994.

[9] TI documenta SPRU 444, "Sensored Field Oriented Control(FOC) of Three Phase AC induction motorl".

[10] Paul C. Krause et "Analysis of Electric Machinery", IEEE Press, Piscataway, NJ, 1996

[11] TI document SPRU 485, "Digital motor control software library".

[12] TI document SPRU 443, "sensorless variable speed 3-phase AC induction motor with closed loop speed control".

[13] TI document BPRA073, "Field Orientation Control of 3-Phase AC-Motors".

[14] TI document BPRA076, "Implementation of a speed field orientated control of three phase AC induction motor using TMS320F240".

[15] TI document SPRA701, "A software modularity strategy for digital control systems".

[16] L.Zhang, C. Wathanasarn, F. Hardan, "An efficient Microprocessor-Based Pulse Width Modulator using Space Vector Modulation Strategy", IEEE 1994.

Appendix A

The Parameters of the Induction Machine

The induction machine used for the experimental system is a squirrel cage machine with the following ratings:

Rated voltage	230Volts, rms, line-to-line
Rated current	12.7Amps, rms
Rated frequency	60Hz
Rated power	5 hp
Rated speed	1750 rpm
Number of Poles	4

By field test and compared with data of the manufacturer, the parameters are determined as follows.

Rotor resistance r_r	0.3097Ω
Stator leakage inductance L_{sl}	1.304mH
Rotor leakage inductance L_{rl}	1.6337mH
Magnetizing inductance L_m	74.38mH
Stator self inductance L_s	75.684mH
Rotor self inductance L_r	76.014mH

Appendix B

Software Package for V/f Control Scheme

```
;=================================================================================
; File Name:        vfmras.asm
; Description: Sensorless control of Induction Motor by Volts/Hertz technique
;              using MRAS speed estimator
; Originator:  Digital control systems Group - Texas Instruments
; Modifier:    Chaozheng Ma, Ryerson University,Toronto,Canada.
; Target dependency:    DSP eZdsp™ LF2407, use 5kHz sampling frequency
; May 15;2003
;=================================================================================
;                              SYSTEM OPTIONS
;*********************************************************************************
real_time              .set 1       ; 1 for real time mode, otherwise set 0
;*********************************************************************************
phase1_inc_build .set     0         ; V/Hz profile test
phase2_inc_build .set     0         ; SV_GEN and FC_PWM tests
phase3_inc_build .set     0         ; Currents and DC-bus volt measurement tests
phase4_inc_build .set     0         ; Speed measurement and MRAS speed estimator tests
phase5_inc_build .set     1         ; Sensorless closed-loop V/Hz system test

;---------------------------------------------------------------------------
; External references
;---------------------------------------------------------------------------
        .include  "x24x_app.h"
        .global   MON_RT_CNFG
        .ref      SYS_INIT
; Added by C. Ma
        .ref      DATA_LOG, DATA_LOG_INIT  ;function call
        .ref      dlog_iptr1, dlog_iptr2     ;Inputs
        .ref      trig_value                 ;Inputs
;done
.ref    RAMP_CNTL, RAMP_CNTL_INIT          ; function call
        .ref      target_value               ; Inputs
        .ref      rmp_dly_max, rmp_lo_limit  ; Input Parameters
        .ref      rmp_hi_limit               ; Input Parameter
        .ref      setpt_value, s_eq_t_flg    ; Outputs
        .ref      V_Hz_PROFILE, V_Hz_PROFILE_INIT  ;function call
        .ref      vhz_freq                   ;Inputs
        .ref      v_out                      ;Outputs

        .ref      SVGEN_MF, SVGEN_MF_INIT  ;function call
        .ref      sv_gain, sv_offset, sv_freq;Inputs
        .ref      Ta, Tb, Tc                 ;Outputs

        .ref      FC_PWM_DRV, FC_PWM_DRV_INIT          ;function call
        .ref      Mfunc_c1, Mfunc_c2, Mfunc_c3, Mfunc_p  ;Inputs
```

```
        .ref    CLARKE, CLARKE_INIT                     ;function call
        .ref    clark_a, clark_b        ;Inputs
        .ref    clark_d, clark_q        ;Outputs
;Changed by C. Ma
        .ref    ILEG2DRV, ILEG2DRV_INIT         ;function call
        .ref    Ia_gain,Ib_gain,Ia_offset,Ib_offset ;Inputs
        .ref    Ia_out, Ib_out,Ic_out                   ;Outputs
        .ref    Vdc_meas_gain, Vdc_meas_gain_1          ;Inputs
        .ref    Vdc_meas_offset         ;Inputs
        .ref    Vdc_meas, Vdc_meas_1
;done
        .ref    PHASE_VOLTAGE_CALC                      ;function call
        .ref    PHASE_VOLTAGE_CALC_INIT                 ;function call
        .ref    Mfunc_V1, Mfunc_V2                      ;Inputs
        .ref    Mfunc_V3, DC_bus                        ;Inputs
        .ref    Vphase_A,Vphase_B,Vphase_C              ;Outputs
        .ref    Vdirect,Vquadra                 ;Outputs
;Changed by C. Ma
        .ref    ACI_MRAS, ACI_MRAS_INIT                 ;function call
        .ref    ualfa_mras, ubeta_mras          ;Inputs
        .ref    ialfa_mras, ibeta_mras          ;Inputs
        .ref    wr_hat_mras, wr_hat_rpm_mras            ;Outputs
        .ref  wr_cal,wr_hat_mras1
;done
        .ref    PID_REG1, PID_REG1_INIT                 ;function call
        .ref    pid_fb_reg1, pid_ref_reg1       ;Inputs
        .ref    pid_out_reg1                            ;Output

        .ref    BC_INIT,BC_CALC                         ;function call
        .ref    BC_IN,BC_OUT                            ;Inputs/Outputs
;-------------------------------------------------------------------------
; Local Variable Declarations
;-------------------------------------------------------------------------
        .def    GPR0                    ;General purpose registers.
        .def    GPR1
        .def    GPR2
        .def    GPR3
        .def    GPR4
       .def     COMCON

        .bss    GPR0,1                  ;General purpose registers.
        .bss    GPR1,1
        .bss    GPR2,1
        .bss    GPR3,1
        .bss    GPR4,1

        .bss    freq_testing,1
        .bss    freq_testing_rpm,1
        .bss    speed_ref,1
        .bss    speed_ref_rpm,1
        .bss    syn_speed,1
        .bss    cl_flag,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

RESET   B       _c_int0         ; 00
INT1    B       PHANTOM         ; 02
INT2    B       T1UF_ISR        ; 04
INT3    B       PHANTOM         ; 06
INT4    B       PHANTOM         ; 08
INT5    B       PHANTOM         ; 0A
INT6    B       PHANTOM         ; 0C

        .include "rtvecs.h"
;=============================================================
; M A I N   C O D E  - starts here
;=============================================================
                .text
_c_int0:
        CALL    SYS_INIT
;added by C. Ma
        CALL    DATA_LOG_INIT
;done
        CALL    RAMP_CNTL_INIT
        CALL    V_Hz_PROFILE_INIT
        CALL    SVGEN_MF_INIT
        CALL    FC_PWM_DRV_INIT
        CALL    ILEG2DRV_INIT           ;changed by C. Ma
        CALL    PHASE_VOLTAGE_CALC_INIT
        CALL    CLARKE_INIT
        CALL    ACI_MRAS_INIT
        CALL    PID_REG1_INIT
        CALL    BC_INIT
;-------------------------------------------------------
; Initialise the Real time monitor
;-------------------------------------------------------
        CALL    MON_RT_CNFG             ;For Real-Time
;-------------------------------------------------------
; System Interrupt Init.
;-------------------------------------------------------
        POINT_EV        ;Event Manager
        SPLK    #0000001000000000b,IMRA ;Enable T1 Underflow Int (i.e. Period)
                ; 5432109876543210
        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
;changed by C. Ma
                POINT_PG0
        SPLK    #0000000001000010b,IMR          ;En Int lvl 2,7 (T1 ISR)
                 ;5432109876543210
        SPLK    #0FFFFh, IFR                    ;Clear any pending Ints
        EINT                                    ;Enable global Ints
            POINT_PF2
        LACC    OCRA
        AND             #0BFFFh
        SACL    OCRA                    ;Select Secondary function IOPB6
        LACC    PBDATDIR
        OR              #04000h
```

```
        SACL   PBDATDIR                 ;Set IOPB6 as output
        LACC   PBDATDIR
        AND        #0FFBFh     ;Set IOPB6 low, Enable PWM
        SACL   PBDATDIR
;done
;---------------------------------------------------------
;SYSTEM PHASE INCREMENTAL BUILD OPTIONS - Initialization
;---------------------------------------------------------
          POINT_B0
        .if (phase1_inc_build)
;changed by C. Ma
    SPLK        #15000, freq_testing    ; 45.45% of 66 Hz (30 Hz)
;done
        .endif
;---------------------------------------------------------
        .if (phase2_inc_build)
SPLK    #15000, freq_testing    ; 45.45% of 66 Hz (48 Hz)
LDP     #rmp_dly_max                ; for Ramp control
SPLK    #1,rmp_dly_max
 ;added by C. Ma
    LDP #dlog_iptr1;
    SPLK #Ta, dlog_iptr1;
    SPLK #Tc, dlog_iptr2
; done
        .endif
;---------------------------------------------------------
.if (phase3_inc_build)
    SPLK        #15000, freq_testing    ; 45.45% of 66 Hz (30 Hz)
    LDP         #rmp_dly_max            ; for Ramp control
    SPLK        #4,rmp_dly_max
;added by C. Ma
      LDP #dlog_iptr1;
      SPLK #Ta, dlog_iptr1;
*     SPLK #ipark_q, dlog_iptr2
*     SPLK #rmp_out, dlog_iptr2
      SPLK #Tc, dlog_iptr2
;done
        .endif
;---------------------------------------------------------
        .if (phase4_inc_build)
SPLK    #15000, freq_testing    ; 45.45% of 66 Hz (30 Hz)
LDP             #rmp_dly_max    ; for Ramp control
SPLK    #1,rmp_dly_max
;added by C. Ma
    LDP #dlog_iptr1;
    SPLK #Ta, dlog_iptr1;
    SPLK #rmp_out, dlog_iptr2
;done
        .endif
;---------------------------------------------------------
        .if (phase5_inc_build)
;Changed or added by C. Ma
    SPLK        #15000, freq_testing    ; 45.45% of 66 Hz (30 Hz)
    SPLK        #14850h, speed_ref      ; 40.905% of 66 Hz(27 Hz)
    SPLK        #1, cl_flag
```

```
    LDP                 #rmp_dly_max      ; for Ramp control
    SPLK        #2,rmp_dly_max

    LDP #dlog_iptr1;
    SPLK #Ta, dlog_iptr1;
    SPLK #rmp_out, dlog_iptr2
;done
   .endif
;=================================================================
MAIN:           ;Main system background loop
;=================================================================
M_1
        B       MAIN

;=================================================================
; Routine Name: T1UF_ISR
; Modifier:    Chaozheng Ma
; Last Update:     May 15,2003
;=================================================================
T1UF_ISR:
        MAR             *,AR1           ; Save context
        MAR             *+              ; point to a guaranteed unused location
        SST     #1, *+    ; save ST1
        SST     #0, *+    ; save ST0
        SACH    *+      ; save acc high
        SACL    *+                      ; save acc low
;changed by C. Ma
;       SAR     AR6,*+     ; save AR6 (used by DAC)
;done
        POPD    *+          ; save TOS to free h/w stack
        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 (phase1_inc_build)
LDP     #vhz_freq
        BLDD    #freq_testing,vhz_freq
        CALL    V_Hz_PROFILE
        .endif
;-----------------------------------------------------------
        .if (phase2_inc_build)
        LDP     #target_value
        BLDD    #freq_testing,target_value
CALL    RAMP_CNTL
LDP             #vhz_freq
        BLDD    #setpt_value,vhz_freq
CALL    V_Hz_PROFILE
        LDP             #sv_freq
        BLDD    #vhz_freq,sv_freq
        BLDD    #v_out,sv_gain
        CALL    SVGEN_MF
```

```
        LDP     #Mfunc_c1
        BLDD    #Ta,Mfunc_c1
        BLDD    #Tb,Mfunc_c2
        BLDD    #Tc,Mfunc_c3
        CALL    FC_PWM_DRV
; added by C. Ma
    CALL    DATA_LOG
; done
        .endif
;---------------------------------------------------------
        .if (phase3_inc_build)
POINT_B0
        LDP             #target_value
        BLDD    #freq_testing,target_value
  CALL          RAMP_CNTL
        LDP     #vhz_freq
        BLDD    #setpt_value,vhz_freq
  CALL V_Hz_PROFILE
        LDP             #sv_freq
        BLDD    #vhz_freq,sv_freq
        BLDD    #v_out,sv_gain
  CALL          SVGEN_MF
        LDP             #Mfunc_c1
        BLDD    #Ta,Mfunc_c1
        BLDD    #Tb,Mfunc_c2
        BLDD    #Tc,Mfunc_c3
 CALL           FC_PWM_DRV
 CALL  ILEG2DRV     ;ILEG2_DCBUS_DRV
; Clarke module
    LDP     #clark_a
    BLDD    #Ia_out,clark_a
    BLDD    #Ib_out,clark_b
CALL    CLARKE
    LDP         #DC_bus
;add by C. Ma
    BLDD        #Vdc_meas_1,DC_bus
    BLDD        #Ta,Mfunc_V1
    BLDD        #Tb,Mfunc_V2
    BLDD        #Tc,Mfunc_V3
CALL   PHASE_VOLTAGE_CALC
    LDP         #ualfa_mras
    BLDD        #Vdirect,ualfa_mras
    BLDD        #Vquadra,ubeta_mras
    BLDD        #clark_d,ialfa_mras
    BLDD        #clark_q,ibeta_mras
CALL   ACI_MRAS
;  LDP          #BC_IN
;  BLDD         #wr_hat_mras1,BC_IN
CALL   DATA_LOG
 ;  CALL        DAC_VIEW_DRV
        .endif
;---------------------------------------------------------
        .if (phase4_inc_build)
;---------------------------------------------------------
    POINT_B0
LDP             #target_value
```

```
                BLDD  #freq_testing,target_value
 CALL  RAMP_CNTL
        LDP             #vhz_freq
        BLDD  #setpt_value,vhz_freq
 CALL  V_Hz_PROFILE
        LDP             #sv_freq
        BLDD  #vhz_freq,sv_freq
        BLDD  #v_out,sv_gain
 CALL           SVGEN_MF
        LDP             #Mfunc_c1
        BLDD  #Ta,Mfunc_c1
        BLDD  #Tb,Mfunc_c2
        BLDD  #Tc,Mfunc_c3
 CALL           FC_PWM_DRV

SKIP_SPEED

     CALL       ILEG2DRV  ;ILEG2_DCBUS_DRV
     LDP        #clark_a
     BLDD       #Ia_out,clark_a
     BLDD       #Ib_out,clark_b

     CALL       CLARKE
     LDP        #DC_bus
     BLDD       #Vdc_meas_1,DC_bus
     BLDD       #Ta,Mfunc_V1
     BLDD       #Tb,Mfunc_V2
     BLDD       #Tc,Mfunc_V3

     CALL       PHASE_VOLTAGE_CALC
      LDP       #ualfa_mras
      BLDD      #Vdirect,ualfa_mras
      BLDD      #Vquadra,ubeta_mras
      BLDD      #clark_d,ialfa_mras
      BLDD      #clark_q,ibeta_mras

     CALL       ACI_MRAS
     LDP                #BC_IN
     BLDD       #wr_hat_mras1,BC_IN
; CALL BC_CALC
; LDP   #dlog_iptr1
     CALL   DATA_LOG
;  CALL          DAC_VIEW_DRV
        .endif
;---------------------------------------------------------
        .if (phase5_inc_build)
;---------------------------------------------------------
     POINT_B0
; checking the close-loop flag here
     LACC       cl_flag                         ; ready to be close-loop ?
     BCND       CLOSE_LOOP,GT                   ; Branch for the close-loop system
OPEN_LOOP
     SPLK       #7EB8h,GPR0             ; It is 0.99, assuming slip is 1%
     LT         freq_testing
     MPY                GPR0
     PAC
```

```
        SACH    speed_ref,1
        LDP     #pid_ref_reg1
        BLDD    #speed_ref,pid_ref_reg1
        BLDD    #wr_hat_mras1,pid_fb_reg1     ; for sensorless
;               BLDD  #BC_OUT,pid_fb_reg1     ; for sensorless (averaged)
        CALL    PID_REG1
        LDP     #pid_out_reg1
        LACC    pid_out_reg1                  ; pid_out_reg1 is compensating slip
        ADD     pid_ref_reg1          ; add reference speed here
        POINT_B0
        SACL    syn_speed             ; here is syn. speed
        LDP     #target_value
        BLDD    #freq_testing,target_value
CALL   RAMP_CNTL
        LDP     #vhz_freq
        BLDD    #setpt_value,vhz_freq
        B       SKIP_CL

CLOSE_LOOP
        LDP     #target_value
        BLDD    #speed_ref,target_value
        CALL    RAMP_CNTL
        LDP     #pid_ref_reg1
        BLDD    #setpt_value,pid_ref_reg1
        BLDD    #wr_hat_mras1,pid_fb_reg1     ; for sensorless
        ;       BLDD  #BC_OUT,pid_fb_reg1     ; for sensorless (averaged)
        ;   BLDD #speed_prd,pid_fb_reg1       ; for sensor
CALL   PID_REG1
        LDP     #pid_out_reg1
        LACC    pid_out_reg1                  ; pid_out_reg1 is compensating slip
        ADD     pid_ref_reg1                  ; add reference speed here
        POINT_B0
        SACL    syn_speed                     ; here is syn. speed
        SACL    freq_testing          ; update freq_testing for smoothly returning open-loop
        LDP     #vhz_freq
        BLDD    #syn_speed,vhz_freq
                POINT_B0
SKIP_CL
        CALL    V_Hz_PROFILE
        LDP     #sv_freq
        BLDD    #vhz_freq,sv_freq
        BLDD    #v_out,sv_gain
        CALL    SVGEN_MF
        LDP     #Mfunc_c1
        BLDD    #Ta,Mfunc_c1
        BLDD    #Tb,Mfunc_c2
        BLDD    #Tc,Mfunc_c3
        CALL    FC_PWM_DRV

SKIP_SPEED
      CALL      ILEG2DRV      ;ILEG2_DCBUS_DRV
      LDP       #clark_a
      BLDD      #Ia_out,clark_a
      BLDD      #Ib_out,clark_b
      CALL      CLARKE
      LDP               #DC_bus
```

```
        BLDD  #Vdc_meas_1,DC_bus
        BLDD  #Ta,Mfunc_V1
        BLDD  #Tb,Mfunc_V2
        BLDD  #Tc,Mfunc_V3
 CALL PHASE_VOLTAGE_CALC
        LDP   #ualfa_mras
        BLDD  #Vdirect,ualfa_mras
        BLDD  #Vquadra,ubeta_mras
        BLDD  #clark_d,ialfa_mras
        BLDD  #clark_q,ibeta_mras
 CALL ACI_MRAS
        LDP   #BC_IN
        BLDD  #wr_hat_mras1,BC_IN
; CALL BC_CALC
  CALL  DATA_LOG
        .endif
;-----------------------------------------------------------
; Change synchronous speed from pu value to rpm value (Q15 -> Q0)
        POINT_B0
        SPLK  #28800, GPR1          ; GPR1 = base speed (3600 rpm) (Q3)
        LT            GPR1          ; TREG = GPR1                  (Q3)
        MPY     freq_testing        ; PREG = GPR1*freq_testing     (Q18)
        PAC                         ; ACC = GPR1*freq_testing      (Q18)
        SFR                         ; ACC = GPR1*freq_testing      (Q17)
        SFR                         ; ACC = GPR1*freq_testing      (Q16)
        SACH  freq_testing_rpm      ; freq_testing_rmp = GPR1*freq_testing   (Q0)
; Change reference speed from pu value to rpm value (Q15 -> Q0)
        MPY     speed_ref       ; PREG = GPR1*speed_ref     (Q18)
        PAC                     ; ACC = GPR1*speed_ref      (Q18)
        SFR                     ; ACC = GPR1*speed_ref      (Q17)
        SFR                     ; ACC = GPR1*speed_ref      (Q16)
        SACH  speed_ref_rpm     ; speed_ref_rpm = GPR1*speed_ref     (Q0)
;=================================================================
;End main section of ISR
;=================================================================
;Context restore regs

END_ISR:
        POINT_PG0
  MAR       *, AR1          ; make stack pointer active
  MAR       *-              ; point to top of stack

        ;More context restore if needed
        PSHD  *-                                    ; restore TOS
;       LAR           AR6,*-                        ; restore AR6
        LACL  *-                                    ; Restore Acc low
        ADD           *-,16                         ; Restore Acc high
        LST           #0, *-          ; Restore ST0
        LST     #1, *-                ; Restore ST1 and pointer
        EINT
        RET

; I S R  -  PHANTOM

PHANTOM     B       PHANTOM
PHANTOM1    B       PHANTOM1
```

```
PHANTOM2    B       PHANTOM2
PHANTOM3    B       PHANTOM3
PHANTOM4    B       PHANTOM4
PHANTOM5    B       PHANTOM5
PHANTOM6    B       PHANTOM6
```

Module Name: CLARKE

```
;=================================================================================
; Module Name:  CLARKE
; 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
;=================================================================================
                .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))
                SPM     0               ; SPM reset
                RET
```

Module Name: IV2DRV-M

```
;*************************************************************
; 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                  |
;=============================================================
; 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)
;   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:
; ___________________________________________________
; Global Definitions
;-------------------------------------------------------------
                .def    ILEG2DRV, ILEG2DRV_INIT          ;function call
                .def    Ia_gain,Ib_gain,Ia_offset,Ib_offset  ;Inputs
                .def    Ia_out, Ib_out,Ic_out            ;Outputs
;added by C Ma
                .def    Vdc_meas_gain           ;Inputs
                .def    Vdc_meas_gain_1
                .def    Vdc_meas_offset         ;Inputs
                .def    Vdc_meas, Vdc_meas_1
;done
*******************************************************
* 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
Ic_out          .usect "ileg2drv",1
I_temp          .usect "ileg2drv",1

Vdc_meas_gain           .usect "ileg2drv",1
Vdc_meas_offset         .usect "ileg2drv",1
Vdc_meas                .usect "ileg2drv",1
; added by C Ma
Vdc_meas_gain_1         .usect "ileg2drv",1
Vdc_meas_1              .usect "ileg2drv",1
```

```
; done
*******************************************************
* Configuration Parameters
*******************************************************
I_ch_sel_       .set    0C32h           ; Select Ch3(Phase B),2(Phase A)
                                        ; Select Ch13(Vdc)
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
Ia_offset_      .set    -200            ; Offset introduced by XOR
Ib_offset_      .set    -130            ;
Vdc_offset_     .set  0
Ia_gain_        .set    8215            ; gain=1.02 (Q13)
Ib_gain_        .set    8215            ; gain=1.02 (Q13)
Vdc_gain_       .set  8258
Vdc_gain_1_     .set  5500              ;gain=0.69(Q13)
;done

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   ;
        SPLK    #Vdc_offset_,Vdc_meas_offset    ; Set offsets
                SPLK    #Vdc_gain_,Vdc_meas_gain        ;
        SPLK    #Vdc_gain_1_,Vdc_meas_gain_1    ;
                ldp     #GPTCON>>7              ; Set T1UF as ADC trigger
                splk #0000h, GPTCON

                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
;changed by C Ma
                SPLK    #2,MAXCONV      ; 3 conversions
                bldd    #I_ch_sel,CHSELSEQ1     ; Configure channel select
;done
                SPLK #0010000000000000b,ADCL_CNTL2 ; Start the conversions from stop position
                ret

ILEG2DRV:
                ldp     #ADCL_CNTL2>>7          ; Check SEQ_BSY bit

                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
                LACC    Ia_out                  ;Calculated value for Imeas_c (i.e. Phase W)
                ADD     Ib_out                ;ACC = Imeas_a + Imeas_b
                NEG                                             ;ACC = - (Imeas_a + Imeas_b)
                SACL    Ic_out                ;Imeas_c = - (Imeas_a + Imeas_b)
; added by C Ma
            ;Read 3rd converted value  (Vdc_meas)
                LDP             #ADC_RESULT2>>7
                LACC    ADC_RESULT2 ;Ch2 (default) is for DC_bus voltage
                SFR                     ;Convert to Q15 (Unipolar signal)
                AND             #7FFFh          ;Convert to Q15 (Unipolar signal)
                LDP             #Vdc_meas
                SACL    Vdc_meas
                LT              Vdc_meas_gain   ;Vdc_meas_gain in Q13
                MPY             Vdc_meas        ;Q13 x Q15 = Q28
                PAC
                ADD             Vdc_meas_offset,13      ;add offset in Q28
                SACH    Vdc_meas,3                      ;convert final result to Q15
                LT              Vdc_meas_gain_1         ;Vdc_meas_gain in Q13
                MPY     Vdc_meas                ;Q13 x Q15 = Q28
                PAC
                SACH    Vdc_meas_1,3            ;convert final result to Q15
;done
                RET
```

Module Name: PHASE_VOLTAGE_CALC

```
;=================================================================
; Description:    Calculates the 3 Phase Motor voltages and stationary
;                 dq-axis voltages based on the PWM modulating function
;                 & DC bus voltage measurement.
;                          |                              |-----> Vphase_A
;       Mfunc_V1  o------->|                              |-----> Vphase_B
;       Mfunc_V2  o------->| PHASE_VOLTAGE_CALC           |-----> Vphase_C
;       Mfunc_V3  o------->|    Q15 / Q15                 |
;       DC_bus    o------->|                              |-----> Vdirect
;                                                        |-----> Vquadra
;=================================================================
;Module definitions for external reference.
                .def    PHASE_VOLTAGE_CALC              ;function call
                .def    PHASE_VOLTAGE_CALC_INIT         ;function call
                .def    Mfunc_V1, Mfunc_V2              ;Inputs
                .def    Mfunc_V3, DC_bus                ;Inputs
                .def    Vphase_A,Vphase_B,Vphase_C      ;Outputs
```

```
                .def    Vdirect,Vquadra                         ;Outputs
;=======================================================================
out_of_phase_           .set    0               ; set 1 for the out of phase correction if
; * Mfunc_V1 is out of phase with PWM1, Mfunc_V2 is out of phase with PWM3, * Mfunc_V3 is out of
phase with PWM5; otherwise, set 0 if their phases are correct.
                        .include x24x_app.h
Mfunc_V1                .usect  "volt_cal",1
Mfunc_V2                .usect  "volt_cal",1
Mfunc_V3                .usect  "volt_cal",1
DC_bus          .usect  "volt_cal",1
Vphase_A        .usect  "volt_cal",1
Vphase_B                .usect  "volt_cal",1
Vphase_C                .usect  "volt_cal",1
Vdirect         .usect  "volt_cal",1
Vquadra         .usect  "volt_cal",1
one_third               .usect  "volt_cal",1    ; 1/3
sqrt3inv        .usect  "volt_cal",1    ; 1/sqrt(3)
tmp_volt                .usect  "volt_cal",1    ; temporary variable
;=========================
PHASE_VOLTAGE_CALC_INIT:
;=========================
; 1/3 = 0.33333 => 0.33333*2^15 = 10923
                LDP     #one_third
                SPLK    #10923,one_third;7FFFh x 0.33333...  (Q15)
; 1/sqrt(3) = 0.577350269 => 0.577350269*2^15 = 18919
                SPLK #018919,sqrt3inv   ;7FFFh x 0.577350269...  (Q15)
                RET
;=========================
PHASE_VOLTAGE_CALC:
;=========================
                SETC    SXM
        .if (out_of_phase_)
                LACC    Mfunc_V1
                NEG
                SACL    Mfunc_V1
                LACC    Mfunc_V2
                NEG
                SACL    Mfunc_V2
                LACC    Mfunc_V3
                NEG
                SACL    Mfunc_V3
        .endif
;scale the incomming Modulation functions with the DC_bus value:
        ;Input 1
                LDP     #Mfunc_V1
                LT      Mfunc_V1                ;Mfunc_V1 is in Q15
                MPY     DC_bus          ;DC_bus is in Q15
                PAC                             ;P = Mfunc_V1 * DC_bus
                SACH    Mfunc_V1,1      ;shift 1 to restore Q15 format
        ;Input 2
                LT      Mfunc_V2                ;Mfunc_V1 is in Q15
                MPY     DC_bus          ;DC_bus is in Q15
                PAC                             ;P = Mfunc_V1 * DC_bus
                SACH    Mfunc_V2,1      ;shift 1 to restore Q15 format
        ;Input 3
                LT      Mfunc_V3                ;Mfunc_V1 is in Q15
```

```
                MPY     DC_bus          ;DC_bus is in Q15
                PAC                     ;P = Mfunc_V1 * DC_bus
                SACH    Mfunc_V3,1      ;shift 1 to restore Q15 format
;Calculate the 3 Phase voltages:
                SPM     1
        ;Phase A
                LT      one_third               ; TREG = one_third      (Q15)
                MPY     Mfunc_V1                ; PREG = one_third*Mfunc_V1  (Q15)
                PAC                             ; ACC = one_third*Mfunc_V1   (Q15)
                SFL                             ; ACC = 2*one_third*Mfunc_V1 (Q15)
                MPY     Mfunc_V2        ; PREG = one_third*Mfunc_V2  (Q15)
                SPAC                            ; ACC = (2/3)*Mfunc_V1-(1/3)*Mfunc_V2  (Q15)
                MPY     Mfunc_V3        ; PREG = one_third*Mfunc_V3  (Q15)
                SPAC            ; ACC = (2/3)*Mfunc_V1-(1/3)*Mfunc_V2-(1/3)*Mfunc_V3 (Q15)
                SACH    Vphase_A
        ;Phase B
                MPY     Mfunc_V2                ; PREG = one_third*Mfunc_V2  (Q15)
                PAC                             ; ACC = one_third*Mfunc_V2   (Q15)
                SFL                             ; ACC = 2*one_third*Mfunc_V2 (Q15)
                MPY     Mfunc_V1                ; PREG = one_third*Mfunc_V1  (Q15)
                SPAC                            ; ACC = (2/3)*Mfunc_V2-(1/3)*Mfunc_V1  (Q15)
                MPY     Mfunc_V3                ; PREG = one_third*Mfunc_V3  (Q15)
                SPAC            ; ACC = (2/3)*Mfunc_V2-(1/3)*Mfunc_V1-(1/3)*Mfunc_V3 (Q15)
                SACH    Vphase_B
        ;Phase C
                MPY     Mfunc_V3                ; PREG = one_third*Mfunc_V3  (Q15)
                PAC                             ; ACC = one_third*Mfunc_V3   (Q15)
                SFL                             ; ACC = 2*one_third*Mfunc_V3 (Q15)
                MPY     Mfunc_V1                ; PREG = one_third*Mfunc_V1  (Q15)
                SPAC                            ; ACC = (2/3)*Mfunc_V3-(1/3)*Mfunc_V1  (Q15)
                MPY     Mfunc_V2                ; PREG = one_third*Mfunc_V2  (Q15)
                SPAC            ; ACC = (2/3)*Mfunc_V3-(1/3)*Mfunc_V1-(1/3)*Mfunc_V2 (Q15)
                SACH    Vphase_C
; Voltage transformation (a,b,c)  ->  (Direct,Quadrature)
; Direct-axis
                LACC    Vphase_A        ; ACC = Vphase_A     (Q15)
                SACL    Vdirect         ; Vdirect = Vphase_A  (Q15)
; Quadrature-axis
                SFR                     ; ACC = Vphase_A/2          (Q15)
                ADD     Vphase_B            ; ACC = Vphase_A/2 + Vphase_B (Q15)
                SACL    tmp_volt            ; tmp_volt = Vphase_A/2 + Vphase_B (Q15)
                LT      tmp_volt                    ; TREG = Vphase_A/2 + Vphase_B (Q15)
                MPY     sqrt3inv        ; PREG = (Vphase_A/2 + Vphase_B)*(1/sqrt(3))  (Q30)
                PAC                     ; ACC = (Vphase_A/2 + Vphase_B)*(1/sqrt(3))   (Q30)
                SFL                     ; ACC = (Vphase_A/2 + Vphase_B)*(1/sqrt(3))   (Q31)
                SACH    Vquadra ; Vquadra = (Vphase_A + 2*Vphase_B)*(1/sqrt(3))     (Q15)
                SPM     0
                CLRC    SXM
                RET
```

; Module Name: PID_REG1

```
;=================================================================================
; Description:    Digital PID controller without anti-windup correction
;       pid_ref_reg1 o--------->|Q15  PID_REG1   Q15|----->o pid_out_reg1
;       pid_fb_reg1  o--------->|Q15                 |
```

```
;=============================================================================
                .def    PID_REG1, PID_REG1_INIT                         ;function call
                .def    pid_fb_reg1, pid_ref_reg1               ;Inputs
                .def    pid_out_reg1                                    ;Output
;=============================================================================
Kp_REG1_        .set    20866   ; for Kp_reg1
Ki_HI_REG1_     .set    80      ; for Ki_high_reg1  (Ki=0 for PD)
Ki_LO_REG1_     .set    -1000   ; for Ki_low_reg1  (Ki=0 for PD)
Kd_REG1_        .set    5997    ; for Kd_reg1       (Kd=0 for PI)
PID_OUT_MAX_    .set    300     ; for pid_out_max
PID_OUT_MIN_    .set    0000h   ; for pid_out_min
                .include "x24x_app.h"
;=============================================================================
;Variable Definitions for PID_REG1 module
;-----------------------------------------------------------------------------
Kp_reg1         .usect  "pid_reg1",1    ; Kp = Q15
Ki_low_reg1     .usect  "pid_reg1",1    ; Ki = Q31
Ki_high_reg1    .usect  "pid_reg1",1    ; Ki = Q31
Kd_reg1         .usect  "pid_reg1",1    ; Kd = Q15
K0_low_reg1     .usect  "pid_reg1",1    ; K0 = Q31
K0_high_reg1    .usect  "pid_reg1",1    ; K0 = Q31
K1_reg1         .usect  "pid_reg1",1    ; K1 = Q15
pid_fb_reg1     .usect  "pid_reg1",1
pid_ref_reg1    .usect  "pid_reg1",1
pid_out_reg1    .usect  "pid_reg1",1
pid_out1_reg1   .usect  "pid_reg1",1
pid_e0_reg1     .usect  "pid_reg1",1
pid_e1_reg1     .usect  "pid_reg1",1
pid_e2_reg1     .usect  "pid_reg1",1
tmp1_low_reg1   .usect  "pid_reg1",1
tmp1_high_reg1  .usect  "pid_reg1",1
tmp2_low_reg1   .usect  "pid_reg1",1
tmp2_high_reg1  .usect  "pid_reg1",1
tmp3_reg1       .usect  "pid_reg1",1
abs_e0_reg1     .usect  "pid_reg1",1
sign_reg1       .usect  "pid_reg1",1
PID_OUT_MAX     .usect  "pid_reg1",1
;=============================================================================
PID_REG1_INIT:
;=============================================================================
    LDP         #Kp_reg1

                SPLK    #Kp_REG1_,Kp_reg1
                SPLK    #Ki_LO_REG1_,Ki_low_reg1
                SPLK    #Ki_HI_REG1_,Ki_high_reg1
                SPLK    #Kd_REG1_,Kd_reg1
                SPLK    #0,pid_e1_reg1
                SPLK    #0,pid_e2_reg1
                SPLK    #0,pid_out1_reg1
    SPLK    #PID_OUT_MAX_, PID_OUT_MAX
    RET
;=============================================================================
PID_REG1:
;=============================================================================
        SETC    SXM             ; Sign extension mode
        SETC    OVM             ; Overflow mode
```

```
    SPM         0           ; Reset SPM
; Converting from Kp, Ki, Kd to K0, K1 (Note: K2 = Kd)
            LDP     #Kp_reg1
            LACC    Ki_high_reg1,16   ; ACC = Ki          (Q31)
            ADDS    Ki_low_reg1       ; ACC = Ki          (Q31)
            ADD     Kp_reg1,16   ; ACC = Kp + Ki                    (Q31)
            ADD     Kd_reg1,16   ; ACC = Kp + Ki + Kd               (Q31)
            SACH    K0_high_reg1      ; K0 = Kp + Ki + Kd           (Q31)
            SACL    K0_low_reg1       ; K0 = Kp + Ki + Kd           (Q31)
            LACC    Kd_reg1,16        ; ACC = Kd                    (Q15)
            SFL                       ; ADD = 2*Kd                  (Q15)
            ADD     Kp_reg1,16              ; ACC = 2*Kd+Kp         (Q15)
            SACH    K1_reg1                 ; K1 = 2*Kd+Kp          (Q15)
; e(k) = ref(k)-fb(k) => Q15 = Q15 - Q15
            LACC    pid_ref_reg1    ; ACC = pid_ref_reg1          (Q15)
            SUB     pid_fb_reg1     ; ACC = pid_ref_reg1 - pid_fb_reg1 (Q15)
            SACL    pid_e0_reg1     ; e(k) = pid_ref_reg1 - pid_fb_reg1 (Q15)
; tmp1 = -K1*e(k-1)+K2*e(k-2) => Q31 = -Q15*Q15 + Q15*Q15
            LT      Kd_reg1         ;TREG = K2  (Q15)
            MPY     pid_e2_reg1     ; PREG = K2*e(k-2)              (Q30)
            PAC                     ; ACC = K2*e(k-2)               (Q30)
            LT      K1_reg1         ; TREG = K1                     (Q15)
            MPY     pid_e1_reg1     ; PREG = K1*e(k-1)              (Q30)
            SPAC                    ; ACC = -K1*e(k-1)+K2*e(k-2)    (Q30)
    SACH    tmp1_high_reg1,1        ; tmp1 = -K1*e(k-1)+K2*e(k-2)   (Q31)
    SACL    tmp1_low_reg1,1         ; tmp1 = -K1*e(k-1)+K2*e(k-2)   (Q31)
; tmp2 = K0*e(k)  => Q31 = Q31*Q15
; check sign for "error" only
            LACC    pid_e0_reg1     ; ACC = e(k)
            SACL    sign_reg1
; take absolute for "pid_e2_reg1" only because "K0" is always positive
            ABS                     ; ACC = |e(k)|
            SACL    abs_e0_reg1     ; |e(k)| = ACC low
; now they're positive.
            LT          abs_e0_reg1         ; TREG = |e(k)|
            MPYU    K0_low_reg1     ; PREG = K0_low*|e(k)|
            SPH         tmp2_low_reg1   ; tmp2_low = PREG high
            MPYU    K0_high_reg1        ; PREG = K0_high*|e(k)|
            PAC                                 ; ACC = K0_high*|e(k)|
            ADDS    tmp2_low_reg1       ; ACC = K0_high*|e(k)| + tmp2_low
            SACH    tmp2_high_reg1,1        ; tmp2_high = ACC high  (Q31)
            SACL    tmp2_low_reg1, 1            ; tmp2_low = ACC low (Q31)
; check the sign condition
            LACC    sign_reg1                   ; ACC = sign
            BCND    DONE_REG1, GT       ; Check sign = positive ?
            LACC    tmp2_high_reg1,16   ; ACC high= tmp2_high
            ADDS    tmp2_low_reg1       ; ACC low = tmp2_low
            NEG                                 ; Make the result negative
            SACH    tmp2_high_reg1      ; tmp2_high = ACC high
            SACL    tmp2_low_reg1       ; tmp2_low = ACC low
DONE_REG1
; tmp2 + tmp1 = tmp3  => Q31 + Q31 = Q15
            LACC    tmp1_high_reg1,16   ; ACC high = tmp1_high      (Q31)
            ADDS    tmp1_low_reg1       ; ACC low = tmp1_low        (Q31)
            ADDS    tmp2_low_reg1       ; ACC = tmp1_low+tmp2_low   (Q31)
            ADDH    tmp2_high_reg1      ; ACC = tmp1_high+tmp2_high (Q31)
```

```
                SACH   tmp3_reg1                          ; tmp3 = tmp1_high+tmp1_high    (Q15)
; u(k) = u(k-1) + tmp3  => Q15 = Q15 + Q15
                LACC   pid_out1_reg1,16; ACC = u(k-1)    (Q15)
                ADD    tmp3_reg1,16          ; ACC = u(k-1)+K0*e(k)-K1*e(k-1)+K2*e(k-2)  (Q15)
                SACH   pid_out_reg1          ; u(k) = u(k-1)+K0*e(k)-K1*e(k-1)+K2*e(k-2)   (Q15)
; If u(k) > u_max, u(k) = u_max. If u(k) < u_min, u(k) = u_min.
                LACC   pid_out_reg1              ; ACC = u(k)          (Q15)
  ;   SUB                #PID_OUT_MAX_         ; ACC = u(k)-u_max    (Q15)
    SUB    #PID_OUT_MAX_
    BCND       SAT_MAX,GT            ; Branch if saturated at max
    LACC       pid_out_reg1          ; ACC = u(k)                  (Q15)
    SUB                #PID_OUT_MIN_; ACC = u(k)-u_min           (Q15)
    BCND       SAT_MIN,LT                    ; Brnch if saturated at min
                B                  REG1_END
SAT_MIN
     SPLK      #PID_OUT_MIN_,pid_out_reg1    ; u(k) = u_min          (Q15)
                B                  REG1_END
SAT_MAX
     SPLK   #PID_OUT_MAX_,pid_out_reg1        ; u(k) = u_max          (Q15)
REG1_END
; Updating the errors e(k-1), e(k-2) and output u(k-1)
                LACC   pid_e1_reg1           ; ACC = e(k-1)           (Q15)
                SACL   pid_e2_reg1           ; e(k-2) = e(k-1)        (Q15)
                LACC   pid_e0_reg1           ; ACC = e(k)             (Q15)
                SACL   pid_e1_reg1           ; e(k-1) = e(k)          (Q15)

               LACC    pid_out_reg1        ; ACC = u(k)              (Q15)
                SACL   pid_out1_reg1       ; u(k-1) = u(k)           (Q15)
                CLRC   SXM
                RET
```

; Module Name: ACI_MRAS

```
;=================================================================================
; Description:  Reactive Power Model Reference Adaptive System (MRAS)
;                       Speed Estimator of Induction Motor
;       ualfa_mras      o----->|Q15
;       ubeta_mras      o----->|Q15   ACI_MRAS  Q15|----->o  wr_hat_mras
;       ialfa_mras      o----->|Q15                Q0|----->o  wr_hat_rpm_mras
;       ibeta_mras      o----->|Q15                 |
;Module definitions for external reference.
                .def    ACI_MRAS, ACI_MRAS_INIT             ;function call
                .def    ualfa_mras, ubeta_mras              ;Inputs
                .def    ialfa_mras, ibeta_mras              ;Inputs
                .def    wr_hat_mras, wr_hat_rpm_mras        ;Outputs
                .def    wr_cal,wr_hat_mras1
;=================================================================================
ref_model_      .set    1                       ; set 1 to activate reference model, otherwise set 0
adt_model_      .set    1                       ; set 1 to activate adaptive model, otherwise set 0
pi_mras_        .set    1                        ; set 1 to activate PI controller, otherwise set 0
motor4          .set    0                       ; 5-Hp motor (WH)
motor5          .set    1                       ; 5-Hp motor (WH)

 .if (motor4)
K1_                     .set    6000            ; for K1 (Q10)
K2_                     .set    1500            ; for K2 (Q15)
```

```
K3_                     .set    23684           ; for K3 (Q8)
K4_                     .set    7               ; for K4 (Q15)
K5_                     .set    32740           ; for K5 (Q15)
K6_                     .set    600             ; for K6 (Q15)
K7_                     .set    28              ; for K7 (Q15)
 .endif
.if (motor5)
K1_                     .set    2350            ; for K1 (Q10)
K2_                     .set    1538            ; for K2 (Q15)
K3_                     .set    23684           ; for K3 (Q8)
K4_                     .set    93              ; for K4 (Q15)
K5_                     .set    32740           ; for K5 (Q15)
K6_                     .set    2475            ; for K6 (Q15)
K7_                     .set    35              ; for K7 (Q15)
wr_cal_                 .set  30200         ;(Q15)
 .endif
BASE_RPM_  .set         14400                   ; for base_rpm (Q3)
                        .include x24x_app.h
ualfa_mras              .usect  "mras_aci",1    ; Measured alfa-axis voltage at k (pu)
ubeta_mras              .usect  "mras_aci",1    ; Measured beta-axis voltage at k (pu)
ialfa_mras          .usect"mras_aci",1      ; Measured alfa-axis current at k (pu)
ibeta_mras              .usect  "mras_aci",1    ; Measured beta-axis current at k (pu)
wr_hat_mras          .usect     "mras_aci",1    ; Estimated rotor speed at k (pu)
wr_hat_mras1        .usect"mras_aci",1
wr_hat_rpm_mras         .usect  "mras_aci",1    ; Estimated rotor speed at k (rpm)
ialfa_old          .usect   "mras_aci",1      ; Measured alfa-axis current at k-1 (pu)
ibeta_old               .usect  "mras_aci",1    ; Measured beta-axis current at k-1 (pu)
ealfa                   .usect  "mras_aci",1    ; Alfa-axis back emf (pu)
ebeta                   .usect  "mras_aci",1    ; Beta-axis back emf (pu)
imalfa_high             .usect  "mras_aci",1    ; Alfa-axis magnetizing current at k (pu)
imalfa_low              .usect  "mras_aci",1    ; Alfa-axis magnetizing current at k (pu)
imbeta_high             .usect  "mras_aci",1    ; Beta-axis magnetizing current at k (pu)
imbeta_low              .usect  "mras_aci",1    ; Beta-axis magnetizing current at k (pu)
imalfa_old_high .usect  "mras_aci",1    ; Alfa-axis magnetizing current at k-1 (pu)
imalfa_old_low  .usect  "mras_aci",1    ; Alfa-axis magnetizing current at k-1 (pu)
imbeta_old_high .usect  "mras_aci",1    ; Beta-axis magnetizing current at k-1 (pu)
imbeta_old_low  .usect  "mras_aci",1    ; Beta-axis magnetizing current at k-1 (pu)
q                               .usect  "mras_aci",1    ; Reactive power in reference model (pu)
q_hat                   .usect  "mras_aci",1    ; Reactive power in adaptive model (pu)
error                       .usect      "mras_aci",1    ; Reactive error (pu)
K1                              .usect  "mras_aci",1    ; Constant using in reference model
K2                              .usect  "mras_aci",1    ; Constant using in adaptive model
K3                              .usect  "mras_aci",1    ; Constant using in adaptive model
K4                              .usect  "mras_aci",1    ; Constant using in adaptive model
K5                              .usect  "mras_aci",1    ; Constant using in adaptive model
K6                              .usect  "mras_aci",1    ; Constant using in adaptive model
K7                              .usect  "mras_aci",1    ; Constant using in adaptive model
Kp                              .usect  "mras_aci",1    ; PI proportionnal constant
Ki_high                 .usect  "mras_aci",1    ; PI integral constant (high 16 bit)
Ki_low                  .usect  "mras_aci",1    ; PI integral constant (low 16 bit)
tmp1_high               .usect  "mras_aci",1    ; 32-bit temporary variable (high 16 bit)
tmp1_low                    .usect      "mras_aci",1    ; 32-bit temporary variable (low 16 bit)
tmp2_high               .usect  "mras_aci",1    ; 32-bit temporary variable (high 16 bit)
tmp2_low                    .usect      "mras_aci",1    ; 32-bit temporary variable (low 16 bit)
tmp3_high               .usect  "mras_aci",1    ; 32-bit temporary variable (high 16 bit)
tmp3_low                .usect  "mras_aci",1    ; 32-bit temporary variable (low 16 bit)
```

```
tmp4                    .usect    "mras_aci",1      ; 16-bit temporary variable
tmp5                    .usect    "mras_aci",1      ; 16-bit temporary variable
sign1                   .usect    "mras_aci",1      ; Checking sign variable for Q31 multiplication
sign2                   .usect    "mras_aci",1      ; Checking sign variable for Q31 multiplication
sign3                   .usect    "mras_aci",1      ; Checking sign variable for Q31 multiplication
base_rpm                .usect    "mras_aci",1      ; Base motor speed in rpm (Q3 signed)
wr_cal      .usect      "mras_aci",1
;=========================
ACI_MRAS_INIT:
;=========================
        LDP             #K1
        SPLK    #K1_,K1                           ; K1 = (Ls-Lm^2/Lr)*Ib/(T*Vb)  (Q11)
        SPLK    #K2_,K2                           ; K2 = Lm^2*Ib/(Lr*Tr*Vb)      (Q15)
        SPLK    #K3_,K3                           ; K3 = Tr*Wb                   (Q8)
        SPLK    #K4_,K4                           ; K4 = (Wb*T)^2/2              (Q15)
  SPLK #K5_,K5                  ; K5 = 1-T/Tr+T^2/(2*Tr^2)    (Q15)
        SPLK    #K6_,K6         ; K6 = Wb*(T-T^2/Tr)          (Q15)
  SPLK #K7_,K7                  ; K7 = T/Tr-T^2/(2*Tr^2)      (Q15)
  SPLK #BASE_RPM_,base_rpm              ; Base motor speed in rpm     (Q3)
  SPLK #wr_cal_,wr_cal
        SPLK    #0000h,q                          ; Initial value of q in reference model
        SPLK    #0000h,q_hat                      ; Initial value of q_hat in adaptive model
  SPLK #0000h,ialfa_old         ; Initial value of ialfa
  SPLK #0000h,ibeta_old         ; Initial value of ibeta
  SPLK #0000h,imalfa_old_low    ; Initial value of imalfa (low 16 bit)
  SPLK #0000h,imalfa_old_high   ; Initial value of imalfa (high 16 bit)
  SPLK #0000h,imbeta_old_low    ; Initial value of imbeta (low 16 bit)
  SPLK #0000h,imbeta_old_high   ; Initial value of imbeta (high 16 bit)
  SPLK #0000h,wr_hat_mras       ; Initial value of angular speed
        SPLK    #0000h,error              ; Initial value of error of reactive power
        SPLK    #25,Kp                    ; PI proportionnal constant         (Q15)
  SPLK #30,Ki_high              ; PI integral constant (high 16 bit)  (Q31)
        SPLK    #0DEADh,Ki_low            ; PI integral constant (low 16 bit)   (Q31)
  RET
;=========================
ACI_MRAS:
;=========================
        SPM             0                         ; Reset product mode
        SETC    SXM                               ; Set sign extension mode
        SETC    OVM                               ; Set overflow mode
        LDP             #ibeta_old
        .if (ref_model_)
;=========================
;REF_MODEL:
;=========================
; ------------------------ Start reference model section ----------------------------
; K1*(ibeta(k)*ialfa(k-1)-ialfa(k)*ibeta(k-1)) = tmp1  => Q10*(Q15*Q15-Q15*Q15) = Q31
     LT         ibeta_old                 ; TREG = ibeta(k-1)                 (Q15)
     MPY        ialfa_mras                ; PREG = ialfa(k)*ibeta(k-1)        (Q30)
        PAC                               ; ACC  = ialfa(k)*ibeta(k-1)        (Q30)
        NEG                               ; Make result negative
        LT      ialfa_old                 ; TREG = ialfa(k-1)             (Q15)
        MPY     ibeta_mras                ; PREG = ibeta(k)*ialfa(k-1)        (Q30)
        APAC                              ; ACC = ibeta(k)*ialfa(k-1)-ialfa(k)*ibeta(k-1)       (Q30)
        SACH    sign1                     ; Keep sign for ibeta(k)*ialfa(k-1)-ialfa(k)*ibeta(k-1)
        ABS                               ; ACC = |ibeta(k)*ialfa(k-1)-ialfa(k)*ibeta(k-1)|     (Q30)
```

```
        SACH   tmp1_high,1              ; tmp1_high = |ibeta(k)*ialfa(k-1)-ialfa(k)*ibeta(k-1)| (Q31)
        SACL   tmp1_low,1               ; tmp1_low = |ibeta(k)*ialfa(k-1)-ialfa(k)*ibeta(k-1)|  (Q31)
; now they're positive.
        LT             K1                                          ; TREG = K1                 (Q10)
        MPYU   tmp1_low                 ; PREG = K1*tmp1_low
        SPH            tmp4             ; tmp4 = PREG high (save partial result)
        SPL            tmp5             ; tmp5 = PREG low
        MPYU   tmp1_high                ; PREG = K1*tmp1_high
        PAC                                         ; ACC = K1*tmp1_high
        ADDS   tmp4                                 ; ACC = K1*tmp1_high + tmp4
        SACH   tmp1_high,6                          ; tmp1_high = ACC high      (Q31)
        SACL   tmp1_low,6               ; tmp1_low      = ACC low               (Q31)
        LACC   tmp5                                 ; ACC = tmp5
        AND            #0FC00h          ; Pick MSB 6 bits in low 16 bits of K1*tmp1_low
        SACH   tmp5,6                   ; tmp5 = MSB 6 bits in low 16 bits of K1*tmp1_low
        LACL   tmp5                                 ; ACC = tmp5
        ADDS   tmp1_low         ; tmp1_low = MSB 6 bits in low 16 bits of K1*tmp1_low + tmp1_low
        SACL   tmp1_low                             ; tmp1_high = ACC low
; check the sign condition
        LACC   sign1                    ; ACC = sign1
        BCND   DONE0, GT                ; Check sign1 = positive ?
        LACC   tmp1_high,16             ; ACC high = tmp1_high
  ADDS          tmp1_low                ; ACC = tmp1_high tmp3_low
        NEG                             ; Make the result negative
        SACH   tmp1_high                ; tmp1_high = ACC high
        SACL   tmp1_low                 ; tmp1_low = ACC low
DONE0
; ialfa(k)*ubeta(k)-ibeta(k)*ualfa(k) = tmp2  => Q15*Q15-Q15*Q15 = Q31
        LT             ibeta_mras       ; TREG = ibeta(k)           (Q15)
        MPY    ualfa_mras       ; PREG = ibeta(k)*ualfa(k)      (Q30)
        PAC                                         ; ACC  = ibeta(k)*ualfa(k)      (Q30)
        NEG                                         ; Make result negative
        LT             ialfa_mras       ; TREG = ialfa(k)           (Q15)
        MPY    ubeta_mras       ; PREG = ialfa(k)*ubeta(k)      (Q30)
        APAC                                        ; ACC  = ialfa(k)*ubeta(k)-ibeta(k)*ualfa(k)  (Q30)
        SACL   tmp2_low,1               ; tmp2 = ialfa(k)*ubeta(k)-ibeta(k)*ualfa(k) (L 16 bit)  (Q31)
        SACH   tmp2_high,1              ; tmp2 = ialfa(k)*ubeta(k)-ibeta(k)*ualfa(k) (H 16 bit)  (Q31)
; tmp4 = q(k) = tmp2 - tmp1  => Q15 = Q31 - Q31
Q
        LACC   tmp2_high,16             ; ACC high = tmp2_high              (Q31)
        ADDS   tmp2_low                             ; ACC low = tmp2_low                (Q31)
        SUBS   tmp1_low                             ; ACC = tmp2_low-tmp1_low           (Q31)
        SUBH   tmp1_high                            ; ACC = tmp2_high-tmp1_high         (Q31)
        SACH   tmp4                                 ; tmp4 or q(k) = tmp2_high-tmp1_high   (Q15);
;Averaging the reactive power => (q(k)+q(k-1))/2
        LACC   tmp4                                 ; ACC = q(k)                (Q15)
        ADD    q                                    ; ACC = q(k)+q(k-1)         (Q15)
        SFR                                         ; ACC = (q(k)+q(k-1))/2     (Q15)
        SACL   q                                    ; q(k) = (q(k)+q(k-1))/2    (Q15)
; Update ialfa(k-1) = ialfa(k) and ibeta(k-1) = ibeta(k)
        LACC   ialfa_mras                   ; ACC low = ialfa(k)        (Q15)
        SACL   ialfa_old                    ; ialfa(k-1) = ialfa(k)     (Q15)
        LACC   ibeta_mras                   ; ACC low = ibeta(k)        (Q15)
        SACL   ibeta_old                            ; ibeta(k-1) = ibeta(k)     (Q15)
; ------------------------ End reference model section ----------------------------
        .endif
```

```
        .if (adt_model_)
;=========================
;ADT_MODEL:
;=========================
; ------------------------ Start adaptive model section ----------------------------
IM_ALFA
; K5-K4*wr_hat*wr_hat = tmp1  =>  Q15-Q15*Q15*Q15 = Q31
        LT      wr_hat_mras                     ; TREG = wr_hat(k)          (Q15)
        MPY     wr_hat_mras                     ; PREG = wr_hat(k)*wr_hat(k)    (Q30)
        PAC                                     ; ACC  = wr_hat(k)*wr_hat(k)    (Q30)
        SACH    tmp1_high,1                     ; tmp1_high = wr_hat(k)*wr_hat(k) (Q15)
        LT              tmp1_high               ; TREG = wr_hat(k)*wr_hat(k)    (Q15)
        MPY     K4                              ; PREG = K4*wr_hat(k)*wr_hat(k)  (Q30)
        PAC                                     ; ACC  = K4*wr_hat(k)*wr_hat(k)  (Q30)
        NEG                                     ; ACC  = -K4*wr_hat(k)*wr_hat(k)  (Q30)
        SFL                             ; rotate to left 1 bit (SXM=1)   (Q31)
        ADDH    K5                      ; ACC = K5-K4*wr_hat(k)*wr_hat(k) (Q31)
        SACH    tmp1_high               ; tmp1_high = K5-K4*wr_hat(k)*wr_hat(k)  (Q31)
        SACL    tmp1_low                        ; tmp1_low = K5-K4*wr_hat(k)*wr_hat(k)   (Q31)
; wr_hat(k)*K6 = tmp4  =>  Q15*Q15 = Q15
        LT              wr_hat_mras             ; TREG = wr_hat(k)              (Q15)
        MPY             K6                      ; PREG = K6*wr_hat(k)           (Q30)
        PAC                                     ; ACC  = K6*wr_hat(k)           (Q30)
        SACH    tmp4,1                          ; tmp4 = K6*wr_hat(k)           (Q15)
; tmp1*imalfa(k-1) = tmp2  =>  Q31*Q31 = Q31
; check sign for "imalfa_old" and "tmp1"
  LACC          tmp1_high                       ; ACC = tmp1_high
  XOR           imalfa_old_high                 ;
  SACLsign1                                     ; Sign (0=+,1=-) for tmp1*imalfa(k-1)
; check sign for "imbeta_old" and "tmp1"
        LACC    tmp1_high                               ; ACC = tmp1_high
        XOR             imbeta_old_high         ;
        SACL    sign2                                   ; Sign (0=+,1=-) for tmp1*imbeta(k-1)
; check sign for "imalfa_old" and "tmp4"
        LACC    tmp4                                    ; ACC = tmp4
        XOR             imalfa_old_high         ;
        SACL    sign3                                   ; Sign (0=+,1=-) for tmp4*imalfa(k-1)
; take absolute for "imalfa_old" and "tmp1"
        LACC    tmp1_high,16                            ; ACC high = tmp1_high
        ADDS    tmp1_low                                ; ACC = tmp1_high tmp1_low
        ABS
        SACL    tmp1_low                                ; tmp1_low = ACC low
        SACH    tmp1_high               ; tmp1_high = ACC high
        LACC    imalfa_old_high,16      ; ACC high = imalfa(k-1)_high
        ADDS    imalfa_old_low          ; ACC = imalfa(k-1)_high imalfa(k-1)_low
        ABS
        SACL    imalfa_old_low          ; imalfa(k-1)_low = ACC low
        SACH    imalfa_old_high         ; imalfa(k-1)_high = ACC high
; now they're positive.
        LT              imalfa_old_low  ; TREG = imalfa(k-1)_low
        MPYU    tmp1_low                ; PREG = imalfa(k-1)_low*tmp1_low
        SPH             tmp5            ; tmp5 = PREG high (save partial result)
        MPYU    tmp1_high               ; PREG = imalfa(k-1)_low*tmp1_high
        LTP             imalfa_old_high ; TREG = imalfa(k-1)_high, ACC = PREG
        MPYU    tmp1_low                        ; PREG = imalfa(k-1)_high*tmp1_low (PREG_old)
        ADDS    tmp5                            ; ACC = imalfa(k-1)_low*tmp1_high + tmp5
```

```
        MPYA  tmp1_high          ; ACC = ACC + PREG_old, PREG_new = imalfa(k-
                                 ;1)_high*tmp1_high
        SACH  tmp5               ; tmp5 = ACC high
        SPLK  #1h,tmp2_low               ; tmp2_low = 1 (carry bit)
        BCND  NO_C1,NC                   ; Carry bit = 0 ?
        B             CHECK1             ; Branch to CHECK1 if carry bit = 1
NO_C1
        SPLK  #0h,tmp2_low               ; tmp2_low = 0 (no carry bit)
CHECK1
        PAC                              ; ACC = imalfa(k-1)_high*tmp1_high
        ADDS  tmp5                       ; ACC = imalfa(k-1)_high*tmp1_high + tmp5
        ADDH  tmp2_low           ; ACC = imalfa(k-1)_high*tmp1_high + tmp5 + tmp2_low
(Carry bit)
        SACH  tmp2_high,1                ; tmp2_high = ACC high  (Q31)
        SACL  tmp2_low,1                 ; tmp2_low = ACC low    (Q31)
; check the sign condition
        LACC  sign1                      ; ACC = sign1
        BCND  DONE1, GT                  ; Check sign1 = positive ?
        LACC  tmp2_high,16               ; ACC high = tmp2_high
        ADDS  tmp2_low                   ; ACC  = tmp2_high tmp2_low
        NEG                              ; Make the result negative
        SACH  tmp2_high                  ; tmp2_high = ACC high
        SACL  tmp2_low                   ; tmp2_low = ACC low
DONE1
; imbeta_old*tmp4 = tmp3  =>  Q31*Q15 = Q31
; check sign for "imbeta_old" and "tmp4"
        LACC  tmp4                       ; ACC = tmp4
        XOR           imbeta_old_high                ;
        SACL  sign1                      ; Sign (0=+,1=-) for tmp4*imbeta(k-1)
; take absolute for "imbeta_old" and "tmp4"
        LACC  tmp4                       ; ACC = tmp4
        ABS                              ; ACC = |tmp4|
        SACL  tmp4                       ; tmp4 = ACC low
        LACC  imbeta_old_high,16         ; ACC high= imbeta(k-1)_high
        ADDS  imbeta_old_low             ; ACC = imbeta(k-1)
        ABS                              ; ACC = |imbeta(k-1)|
        SACL  imbeta_old_low             ; imbeta(k-1)_low = ACC low
        SACH  imbeta_old_high            ; imbeta(k-1)_high = ACC high
; now they're positive.
        LT            tmp4               ; TREG = tmp4
        MPYU  imbeta_old_low     ; PREG = imbeta(k-1)_low*tmp4
        SPH           tmp3_low           ; tmp3_low = PREG high
        MPYU  imbeta_old_high    ; PREG = imbeta(k-1)_high*tmp4
        PAC                              ; ACC = imbeta(k-1)_high*tmp4
        ADDS  tmp3_low                   ; ACC = imbeta(k-1)_high*tmp4+tmp3_low
        SACH  tmp3_high,1                ; tmp3_high = ACC high  (Q31)
        SACL  tmp3_low,1                 ; tmp3_low = ACC low    (Q31)
; check the sign condition
        LACC  sign1                      ; ACC = sign1
        BCND  DONE2, GT                  ; Check sign1 = positive ?
        LACC  tmp3_high,16       ; ACC high= tmp3_high
        ADDS  tmp3_low                   ; ACC = tmp3_low
        NEG                              ; Make the result negative
        SACH  tmp3_high                  ; tmp3_high = ACC high
        SACL  tmp3_low                   ; tmp3_low = ACC low
DONE2
```

```
; tmp2 - tmp3 = tmp3   =>  Q31 - Q31 = Q31
        LACC   tmp2_high,16             ; ACC high = tmp2_high
        ADDS   tmp2_low                         ; ACC low = tmp2_low
        SUBS   tmp3_low                         ; ACC = tmp2_low-tmp3_low
        SUBH   tmp3_high                        ; ACC = tmp2_high-tmp3_high
        SACL   tmp3_low                         ; tmp3_low = tmp2_low-tmp3_low   (L 16-bit)
        SACH   tmp3_high                        ; tmp3_high = tmp2_high-tmp3_high (H 16-bit)
; ialfa(k)*K7 = tmp2   =>   Q15*Q15 = Q31
        LT              ialfa_mras                              ; TREG = ialfa(k)          (Q15)
        MPY             K7                                      ; PREG = ialfa(k)*K7       (Q30)
        PAC                                                     ; ACC = ialfa(k)*K7        (Q30)
        SACH   tmp2_high,1                              ; tmp2_high = ACC high   (Q31)
        SACL   tmp2_low,1                               ; tmp2_low = ACC low     (Q31)
; imalfa(k) = tmp3 + tmp2    =>   Q31 = Q31 + Q31
        LACC   tmp3_high,16             ; ACC high = tmp3_high
        ADDS   tmp3_low                         ; ACC low = tmp3_low
        ADDS   tmp2_low                         ; ACC = tmp3_low+tmp2_low
        ADDH   tmp2_high                        ; ACC = tmp3_high+tmp2_high
        SACL   imalfa_low                       ; imalfa_low = tmp3_low+tmp2_low   (L 16-bit)
        SACH   imalfa_high                      ; imalfa_high = tmp3_high+tmp2_high (H 16-bit)
IM_BETA
; tmp1*imbeta(k-1) = tmp2   =>   Q31*Q31 = Q31
; already checked sign for "imbeta_old" and "tmp1" kept in sign2
; already took absolute for "imbeta_old" and "tmp1"
; now they're positive.
        LT      imbeta_old_low          ; TREG = imbeta(k-1)_low
        MPYU    tmp1_low                ; PREG = imbeta(k-1)_low*tmp1_low
        SPH     tmp5                    ; tmp5 = PREG high  (save partial result)
  MPYU          tmp1_high               ; PREG = imbeta(k-1)_low*tmp1_high
        LTP             imbeta_old_high ; TREG = imbeta(k-1)_high, ACC = PREG
        MPYU    tmp1_low                        ; PREG = imbeta(k-1)_high*tmp1_low (PREG_old)
        ADDS    tmp5                            ; ACC = imbeta(k-1)_low*tmp1_high + tmp5
        MPYA    tmp1_high               ; ACC = ACC + PREG_old, PREG_new = imbeta(k-
                                        ;1)_high*tmp1_high
        SACH    tmp5                    ; tmp5 = ACC high
        SPLK    #1h,tmp2_low            ; tmp2_low = 1 (carry bit)
        BCND    NO_C2,NC                ; Carry bit = 0 ?
        B               CHECK2          ; Branch to CHECK2 if carry bit = 1
NO_C2
        SPLK    #0h,tmp2_low            ; tmp2_low = 0 (no carry bit)
CHECK2
        PAC                             ; ACC = imbeta(k-1)_high*tmp1_high
        ADDS    tmp5                    ; ACC = imbeta(k-1)_high*tmp1_high + tmp5
        ADDH    tmp2_low                ; ACC = imbeta(k-1)_high*tmp1_high + tmp5 + tmp2_low
(Carry bit)
        SACH    tmp2_high,1                     ; tmp2_high = ACC high   (Q31)
        SACL    tmp2_low,1                      ; tmp2_low = ACC low     (Q31)
; check the sign condition
        LACC    sign2                           ; ACC = sign2
        BCND    DONE3, GT                       ; Check sign2 = positive ?
        LACC    tmp2_high,16                    ; ACC high = tmp2_high
  ADDS          tmp2_low                        ; ACC  = tmp2_high tmp2_low
        NEG                                     ; Make the result negative
        SACH    tmp2_high                       ; tmp2_high = ACC high
        SACL    tmp2_low                        ; tmp2_low = ACC low
DONE3
```

```
; imalfa_old*tmp4 = tmp3  =>  Q31*Q15 = Q31
; already checked sign for "imalfa_old" and "tmp4" kept in sign3
; already took absolute for "imalfa_old" and "tmp4"
; now they're positive.
        LT      tmp4            ; TREG = tmp4
        MPYU    imalfa_old_low  ; PREG = imalfa(k-1)_low*tmp4
        SPH     tmp3_low        ; tmp3_low = PREG high
        MPYU    imalfa_old_high ; PREG = imalfa(k-1)_high*tmp4
        PAC                     ; ACC = imalfa(k-1)_high*tmp4
        ADDS    tmp3_low        ; ACC = imalfa(k-1)_high*tmp4+tmp3_low
        SACH    tmp3_high,1     ; tmp3_high = ACC high    (Q31)
        SACL    tmp3_low,1      ; tmp3_low = ACC low      (Q31)
; check the sign condition
        LACC    sign3           ; ACC = sign3
        BCND    DONE4, GT       ; Check sign3 = positive ?
        LACC    tmp3_high,16    ; ACC high= tmp3_high
        ADDS    tmp3_low        ; ACC = tmp3_low
        NEG                     ; Make the result negative
        SACH    tmp3_high       ; tmp3_high = ACC high
        SACL    tmp3_low        ; tmp3_low = ACC low
DONE4
; tmp2 + tmp3 = tmp3  => Q31 + Q31 = Q31
        LACC    tmp3_high,16    ; ACC high = tmp3_high
        ADDS    tmp3_low        ; ACC low = tmp3_low
        ADDS    tmp2_low        ; ACC = tmp3_low+tmp2_low
        ADDH    tmp2_high       ; ACC = tmp3_high+tmp2_high
        SACL    tmp3_low        ; tmp3_low = tmp3_low+tmp2_low   (L 16-bit)
        SACH    tmp3_high       ; tmp3_high = tmp3_high+tmp2_high (H 16-bit)
; ibeta(k)*K7 = tmp2  =>  Q15*Q15 = Q31
        LT      ibeta_mras      ; TREG = ibeta(k)         (Q15)
        MPY     K7              ; PREG = ibeta(k)*K7      (Q30)
        PAC                     ; ACC = ialfa(k)*K7       (Q30)
        SACH    tmp2_high,1                     ; tmp2_high = ACC high      (Q31)
        SACL    tmp2_low,1                      ; tmp2_low = ACC low        (Q31)
; imbeta(k) = tmp3 + tmp2   =>   Q31 = Q31 + Q31
        LACC    tmp3_high,16    ; ACC high = tmp3_high
        ADDS    tmp3_low                ; ACC low = tmp3_low
        ADDS    tmp2_low                ; ACC = tmp3_low+tmp2_low
        ADDH    tmp2_high               ; ACC = tmp3_high+tmp2_high
        SACL    imbeta_low              ; imbeta_low = tmp3_low+tmp2_low   (L 16-bit)
        SACH    imbeta_high             ; imbeta_high = tmp3_high+tmp2_high (H 16-bit)
; update imbeta(k-1)
        LACC    imbeta_high,16  ; ACC high = imbeta(k)_high
        ADDS    imbeta_low      ; ACC = imbeta(k)
        SACL    imbeta_old_low  ; imbeta(k-1)_low = imbeta(k)_low
  SACH          imbeta_old_high ; imbeta(k-1)_high = imbeta(k)_high
; update imalfa(k-1)
        LACC    imalfa_high,16  ; ACC high = imalfa(k)_high
        ADDS    imalfa_low      ; ACC = imalfa(k)
  SACLimalfa_old_low            ; imalfa(k-1)_low = imalfa(k)_low
        SACH    imalfa_old_high ; imalfa(k-1)_high = imalfa(k)_high
E_ALFA
; ialfa(k) - imalfa(k) = tmp1_low  => Q15 - Q31 = Q15
        LACC    ialfa_mras,16   ; ACC high = ialfa
        SUBS    imalfa_low      ; ACC low = ialfa_low-imalfa_low
        SUBH    imalfa_high     ; ACC high = ialfa_high-imalfa_high
```

```
        SACH   tmp1_low                 ; tmp1_low = ialfa_high-imalfa_high  (H 16-bit)
; ibeta(k) - imbeta(k) = tmp1_high  => Q15 - Q31 = Q15
        LACC   ibeta_mras,16            ; ACC high = ibeta
        SUBS   imbeta_low               ; ACC low = ibeta_low-imbeta_low
        SUBH   imbeta_high              ; ACC high = ibeta_high-imbeta_high
        SACH   tmp1_high                ; tmp1_high = ibeta_high-imbeta_high (H 16-bit)
; wr_hat(k)*imbeta(k) = tmp4  =>  Q15*Q31 = Q15
; check sign for "wr_hat" and "imbeta" kept in sign1
        LACC   wr_hat_mras                          ; ACC = wr_hat
        XOR            imbeta_high                  ;
        SACL   sign1                                ; Sign (0=+,1=-) for wr_hat*imbeta(k)
; take absolute for "imbeta" and "wr_hat"   (tmp5 = |wr_hat|)
        LACC   wr_hat_mras                          ; ACC = wr_hat
        ABS                                         ; ACC = |wr_hat|
        SACL   tmp5                                 ; tmp5 = |wr_hat|
        LACC   imbeta_high,16                   ; ACC high= imbeta(k)_high
        ADDS   imbeta_low                       ; ACC = imbeta(k)
        ABS
        SACL   imbeta_low                       ; imbeta(k)_low = ACC low
        SACH   imbeta_high                      ; imbeta(k)_high = ACC high
; now they're positive
        LT             tmp5                     ; TREG = |wr_hat(k)|          (Q15)
        MPYU   imbeta_low                       ; PREG = wr_hat(k)*imbeta(k)_low
        SPH            tmp4                                 ; tmp4 = PREG high
        MPYU   imbeta_high                      ; PREG = wr_hat(k)*imbeta(k)_high
        PAC                                     ; ACC = wr_hat(k)*imbeta(k)_high
        ADDS   tmp4                             ; ACC = wr_hat(k)*imbeta(k)_high+tmp4
        SACH   tmp4,1                           ; tmp4 = wr_hat(k)*imbeta(k) (Q15)
; check the sign condition
        LACC   sign1                            ; ACC = sign1
        BCND   DONE5, GT                        ; Check sign1 = positive ?
        LACC   tmp4                             ; ACC = tmp4
        NEG                                     ; Make the result negative
        SACL   tmp4                             ; tmp4 = ACC low    (Q15)
DONE5
; K3*tmp4 = tmp4  =>  Q8*Q15 = Q9
        LT             K3                       ; TREG = K3              (Q8)
        MPY            tmp4                     ; PREG = tmp4*K3         (Q23)
        PAC                                     ;ACC = tmp4*K3           (Q23)
        SACH   tmp4,2                           ; tmp4 = tmp4*K3          (Q9)
; tmp1_low - tmp4 = tmp4  =>  Q15 - Q9 = Q9
        LACC   tmp1_low,10                      ; ACC high = tmp1_low       (Q9)
        SUBH   tmp4                             ; ACC = tmp1_low-tmp4       (Q9)
        SACH   tmp4                             ; tmp4 = tmp1_low-tmp4       (Q9)
; ealfa(k) = tmp4*K2  =>  Q15 = Q9*Q15
        LT             K2                       ; TREG = K2              (Q15)
        MPY            tmp4                     ; PREG = tmp4*K2         (Q24)
        PAC                                     ;ACC  = tmp4*K2               (Q24)
        SACH   ealfa,7                          ; ealfa = tmp4*K2         (Q15)
E_BETA
; wr_hat(k)*imalfa(k) = tmp4  =>  Q15*Q31 = Q15
; already took absolute for "wr_hat" kept in tmp5
; check sign for "wr_hat" and "imalfa" kept in sign2
        LACC   wr_hat_mras                          ; ACC = wr_hat
        XOR            imalfa_high                  ;
        SACL   sign2                                ; Sign (0=+,1=-) for wr_hat*imalfa(k)
```

```
; take absolute for "imalfa"  (tmp5 = |wr_hat|)
        LACC   imalfa_high,16          ; ACC high= imalfa(k)_high
        ADDS   imalfa_low                       ; ACC = imalfa(k)
        ABS
        SACL   imalfa_low                       ; imalfa(k)_low = ACC low
        SACH   imalfa_high                      ; imalfa(k)_high = ACC high
; now they're positive
        LT             tmp5                     ;REG = |wr_hat(k)|          (Q15)
        MPYU   imalfa_low                       ; PREG = wr_hat(k)*imalfa(k)_low
        SPH            tmp4                     ;tmp4 = PREG high
        MPYU   imalfa_high                      ; PREG = wr_hat(k)*imalfa(k)_high

        PAC                                     ; ACC = wr_hat(k)*imalfa(k)_high
        ADDS   tmp4                             ;ACC = wr_hat(k)*imalfa(k)_high+tmp4
        SACH   tmp4,1                           ; tmp4 = wr_hat(k)*imalfa(k)   (Q15)
; check the sign condition
        LACC   sign2                            ; ACC = sign2
        BCND   DONE6, GT                        ;Check sign2 = positive ?
        LACC   tmp4                             ;ACC = tmp4
        NEG                                     ; Make the result negative
        SACL   tmp4                     ; tmp4 = ACC low    (Q15)
DONE6
; K3*tmp4 = tmp4  =>  Q8*Q15 = Q9
        LT             K3               ; TREG = K3             (Q8)
        MPY    tmp4                     ; PREG = tmp4*K3        (Q23)
        PAC                             ; ACC = tmp4*K3         (Q23)
        SACH   tmp4,2                   ; tmp4 = tmp4*K3        (Q9)
; tmp1_high + tmp4 = tmp4  =>  Q15 + Q9 = Q9
        LACC   tmp1_high,10             ; ACC high = tmp1_high  (Q9)
        ADDH   tmp4                     ; ACC = tmp1_high+tmp4  (Q9)
        SACH   tmp4                     ; tmp4 = tmp1_high+tmp4 (Q9)
; ebeta(k) = tmp4*K2  =>  Q15 = Q9*Q15
        LT             K2               ; TREG = K2             (Q15)
        MPY            tmp4             ; PREG = tmp4*K2        (Q23)
        PAC                             ; ACC = tmp4*K2         (Q23)
        SACH   ebeta,7                  ; ebeta = tmp4*K2       (Q15)
Q_HAT
; q_hat(k) = ialfa(k)*ebeta(k)-ibeta(k)*ealfa(k) => Q15 = Q15*Q15-Q15*Q15
        LT             ibeta_mras       ; TREG = ibeta(k)       (Q15)
        MPY            ealfa            ; PREG = ibeta(k)*ealfa(k)   (Q30)
        PAC                             ; ACC  = ibeta(k)*ealfa(k)   (Q30)
        NEG                             ; Make result negative
        LT             ialfa_mras       ; TREG = ialfa(k)       (Q15)
        MPY            ebeta            ; PREG = ialfa(k)*ebeta(k)   (Q30)
        APAC                            ; ACC = ialfa(k)*ebeta(k)-ibeta(k)*ealfa(k)  (Q30)
        SACH   q_hat,1                  ; q_hat = ialfa(k)*ebeta(k)-ibeta(k)*ealfa(k)  (Q15)
; ------------------------ End adaptive model section ----------------------------
        .endif
        .if (pi_mras_)
; ---------------- Start conventional PI controller section ----------------------
;=============================
;PI_MRAS:
;=============================
; tmp1 = Kp*error(k-1)  => Q31 = Q15*Q15
        LT             Kp               ; TREG = Kp             (Q15)
        MPY            error            ; PREG = Kp*error(k-1)  (Q30)
```

```
        PAC                                     ; ACC = Kp*error(k-1)          (Q30)
        SACH  tmp1_high,1                       ; tmp1_high = Kp*error(k-1)    (Q31)
        SACL  tmp1_low,1                        ; tmp1_low = Kp*error(k-1)     (Q31)
; error(k) = q - q_hat  =>  Q15 = Q15 - Q15
        LACC  q                         ; ACC = q                   (Q15)
        SUB   q_hat                     ; ACC = q - q_hat           (Q15)
        SACL  error                     ; error = q - q_hat         (Q15)
; Kp + Ki = tmp2  =>  Q15 + Q31 = Q31
        LACC  Ki_high,16                ; ACC high = Ki_high
        ADDS  Ki_low                    ; ACC = Ki_low                (Q31)
        ADDH  Kp                        ; ACC high = Ki_high + Kp     (Q31)
        SACH  tmp2_high                 ; tmp2_high = Ki_high + Kp    (Q31)
        SACL  tmp2_low                  ; tmp2_low = Ki_low + Kp      (Q31)
; tmp2*error = tmp2  =>  Q31*Q15 = Q31
; check sign for "error" only
        LACC  error                     ; ACC = error
        SACL  sign1                     ; Sign (0=+,1=-) for (Kp+Ki)*error
; take absolute for "error" only because "tmp2" is always positive
        ABS                             ; ACC = |error|
        SACL  error                     ; error = ACC low
; now they're positive.
        LT              error           ; TREG = error
        MPYU  tmp2_low                  ; PREG = tmp2_low*error
        SPH             tmp2_low        ; tmp2_low = PREG high
        MPYU  tmp2_high                 ; PREG = tmp2_high*error
        PAC                             ; ACC = tmp2_high*error
        ADDS  tmp2_low                  ; ACC = tmp2_high*error + tmp2_low
        SACH  tmp2_high,1               ; tmp2_high = ACC high   (Q31)
        SACL  tmp2_low,1                ; tmp2_low = ACC low     (Q31)
; check the sign condition
        LACC  sign1                     ; ACC = sign1
        BCND  DONE7, GT                 ; Check sign1 = positive ?
        LACC  tmp2_high,16              ; ACC high= tmp2_high
        ADDS  tmp2_low                  ; ACC low = tmp2_low
        NEG                             ; Make the result negative
        SACH  tmp2_high                 ; tmp2_high = ACC high
        SACL  tmp2_low                  ; tmp2_low = ACC low
DONE7
; tmp2 - tmp1 = tmp1_high  =>  Q31 - Q31 = Q15
        LACC  tmp2_high,16              ; ACC high = tmp2_high       (Q31)
        ADDS  tmp2_low                  ; ACC low = tmp2_low         (Q31)
        SUBS  tmp1_low                  ; ACC = tmp2_low-tmp1_low    (Q31)
        SUBH  tmp1_high                 ; ACC = tmp2_high-tmp1_high  (Q31)
        SACH  tmp1_high                 ; tmp1_high = tmp2-tmp1      (Q15)
; wr_hat(k) = wr_hat(k-1) + tmp1_high => Q15 = Q15 + Q15  (update wr_hat)
        LACC  wr_hat_mras               ; ACC = wr_hat(k-1)      (Q15)
        ADD   tmp1_high                 ; ACC = wr_hat(k-1)+tmp1_high  (Q15)
        SACL  wr_hat_mras               ; wr_hat(k) = wr_hat(k-1)+tmp1_high (Q15)
        .endif
; ---------- End conventional PI controller section ----------
; Change motor speed from pu value to rpm value (Q15 -> Q0 signed)
        LT      base_rpm                ; TREG = base_rpm (Q3)
        MPY     wr_hat_mras             ; PREG = base_rpm*wr_hat_mras (Q18)
        PAC                             ; ACC = base_rpm*wr_hat_mras (Q18)
        SFR                             ; ACC = base_rpm*wr_hat_mras (Q17)
        SFR                             ; ACC = base_rpm*wr_hat_mras (Q16)
```

```
        SACH    wr_hat_rpm_mras         ; wr_hat_rpm_mras = base_rpm*wr_hat_mras (Q0)
        LT      wr_cal                  ; TREG = wr_cal          (Q15)
        MPY     wr_hat_mras             ; PREG =     (Q30)
        PAC                             ; ACC =      (Q30)
        SACH    wr_hat_mras1,1
        RET
```

; Module Name: PID_REG1

```
;=============================================================================
; Description:    Digital PID controller without anti-windup correction
;       pid_ref_reg1 o--------->|Q15  PID_REG1    Q15|----->o pid_out_reg1
;       pid_fb_reg1  o--------->|Q15                 |
;=============================================================================
                .def    PID_REG1, PID_REG1_INIT                 ;function call
                .def    pid_fb_reg1, pid_ref_reg1               ;Inputs
                .def    pid_out_reg1                            ;Output
;=============================================================================
Kp_REG1_        .set    20866       ; for Kp_reg1
Ki_HI_REG1_     .set    80          ; for Ki_low_reg1  (Ki=0 for PD)
Ki_LO_REG1_     .set    -1000       ; for Ki_high_reg1  (Ki=0 for PD)
Kd_REG1_        .set    5997            ; for Kd_reg1      (Kd=0 for PI)
PID_OUT_MAX_    .set    300             ; for pid_out_max
PID_OUT_MIN_    .set    0000h       ; for pid_out_min
                .include "x24x_app.h"
;=============================================================================
;Variable Definitions for PID_REG1 module
;-----------------------------------------------------------------------------
Kp_reg1         .usect  "pid_reg1",1        ; Kp = Q15
Ki_low_reg1     .usect  "pid_reg1",1        ; Ki = Q31
Ki_high_reg1    .usect  "pid_reg1",1        ; Ki = Q31
Kd_reg1         .usect  "pid_reg1",1        ; Kd = Q15
K0_low_reg1     .usect  "pid_reg1",1        ; K0 = Q31
K0_high_reg1    .usect  "pid_reg1",1        ; K0 = Q31
K1_reg1         .usect  "pid_reg1",1        ; K1 = Q15
pid_fb_reg1     .usect  "pid_reg1",1
pid_ref_reg1    .usect  "pid_reg1",1
pid_out_reg1    .usect  "pid_reg1",1
pid_out1_reg1   .usect  "pid_reg1",1
pid_e0_reg1     .usect  "pid_reg1",1
pid_e1_reg1     .usect  "pid_reg1",1
pid_e2_reg1     .usect  "pid_reg1",1
tmp1_low_reg1   .usect  "pid_reg1",1
tmp1_high_reg1  .usect  "pid_reg1",1
tmp2_low_reg1   .usect  "pid_reg1",1
tmp2_high_reg1  .usect  "pid_reg1",1
tmp3_reg1       .usect  "pid_reg1",1
abs_e0_reg1     .usect  "pid_reg1",1
sign_reg1       .usect  "pid_reg1",1
PID_OUT_MAX     .usect  "pid_reg1",1
;=============================================================================
PID_REG1_INIT:
;=============================================================================
    LDP             #Kp_reg1
            SPLK    #Kp_REG1_,Kp_reg1
            SPLK    #Ki_LO_REG1_,Ki_low_reg1
```

```
                SPLK    #Ki_HI_REG1_,Ki_high_reg1
                SPLK    #Kd_REG1_,Kd_reg1
                SPLK    #0,pid_e1_reg1
                SPLK    #0,pid_e2_reg1
                SPLK    #0,pid_out1_reg1
        SPLK    #PID_OUT_MAX_,PID_OUT_MAX
        RET
;=================================================================================
PID_REG1:
;=================================================================================
            SETC    SXM             ; Sign extension mode
        SETC        OVM             ; Overflow mode
        SPM             0               ; Reset SPM
; Converting from Kp, Ki, Kd to K0, K1 (Note: K2 = Kd)
                LDP             #Kp_reg1
                LACC    Ki_high_reg1,16         ; ACC = Ki              (Q31)
                ADDS    Ki_low_reg1             ; ACC = Ki              (Q31)
                ADD             Kp_reg1,16              ; ACC = Kp + Ki                 (Q31)
                ADD             Kd_reg1,16              ; ACC = Kp + Ki + Kd            (Q31)
                SACH     K0_high_reg1           ; K0 = Kp + Ki + Kd                     (Q31)
                SACL    K0_low_reg1             ; K0 = Kp + Ki + Kd                     (Q31)
                LACC    Kd_reg1,16              ; ACC = Kd                              (Q15)
                SFL                             ; ADD = 2*Kd                            (Q15)
                ADD     Kp_reg1,16      ; ACC = 2*Kd+Kp         (Q15)
                SACH    K1_reg1         ; K1 = 2*Kd+Kp          (Q15)
; e(k) = ref(k)-fb(k) => Q15 = Q15 - Q15
                LACC    pid_ref_reg1    ; ACC =  pid_ref_reg1       (Q15)
                SUB     pid_fb_reg1     ; ACC = pid_ref_reg1 - pid_fb_reg1  (Q15)
                SACL    pid_e0_reg1     ; e(k) = pid_ref_reg1 - pid_fb_reg1 (Q15)
; tmp1 = -K1*e(k-1)+K2*e(k-2) => Q31 = -Q15*Q15 + Q15*Q15
                LT      Kd_reg1         ; TREG = K2                             (Q15)
                MPY     pid_e2_reg1     ; PREG = K2*e(k-2)                      (Q30)
                PAC                     ; ACC = K2*e(k-2)                       (Q30)
                LT      K1_reg1         ; TREG = K1                             (Q15)
                MPY     pid_e1_reg1     ; PREG = K1*e(k-1)                      (Q30)
                SPAC                    ; ACC = -K1*e(k-1)+K2*e(k-2)            (Q30)
        SACH    tmp1_high_reg1,1        ; tmp1 = -K1*e(k-1)+K2*e(k-2)           (Q31)
        SACL    tmp1_low_reg1,1         ; tmp1 = -K1*e(k-1)+K2*e(k-2)           (Q31)
; tmp2 = K0*e(k)   => Q31 = Q31*Q15
; check sign for "error" only
                LACC    pid_e0_reg1     ; ACC = e(k)
                SACL    sign_reg1       ; Sign (0=+,1=-) for K0*e(k)
; take absolute for "pid_e2_reg1" only because "K0" is always positive
                ABS                     ; ACC = |e(k)|
                SACL    abs_e0_reg1     ; |e(k)| = ACC low
; now they're positive.
                LT      abs_e0_reg1             ; TREG = |e(k)|
                MPYU    K0_low_reg1             ; PREG = K0_low*|e(k)|
                SPH     tmp2_low_reg1           ; tmp2_low = PREG high
                MPYU    K0_high_reg1            ; PREG = K0_high*|e(k)|
                PAC                             ; ACC = K0_high*|e(k)|
                ADDS    tmp2_low_reg1           ; ACC = K0_high*|e(k)| + tmp2_low
                SACH    tmp2_high_reg1,1        ; tmp2_high = ACC high      (Q31)
                SACL    tmp2_low_reg1,1         ; tmp2_low = ACC low        (Q31)
; check the sign condition
                LACC    sign_reg1               ; ACC = sign
```

```
                BCND  DONE_REG1, GT        ; Check sign = positive ?
                LACC  tmp2_high_reg1,16    ; ACC high= tmp2_high
                ADDS  tmp2_low_reg1        ; ACC low = tmp2_low
                NEG                        ; Make the result negative
                SACH  tmp2_high_reg1       ; tmp2_high = ACC high
                SACL  tmp2_low_reg1        ; tmp2_low = ACC low
DONE_REG1
; tmp2 + tmp1 = tmp3  => Q31 + Q31 = Q15
                LACC  tmp1_high_reg1,16    ; ACC high = tmp1_high        (Q31)
                ADDS  tmp1_low_reg1        ; ACC low = tmp1_low          (Q31)
                ADDS  tmp2_low_reg1        ; ACC = tmp1_low+tmp2_low     (Q31)
                ADDH  tmp2_high_reg1       ; ACC = tmp1_high+tmp2_high   (Q31)
                SACH  tmp3_reg1            ; tmp3 = tmp1_high+tmp1_high  (Q15)
; u(k) = u(k-1) + tmp3  =>  Q15 = Q15 + Q15
                LACC  pid_out1_reg1,16; ACC = u(k-1)   (Q15)
                ADD   tmp3_reg1,16     ; ACC = u(k-1)+K0*e(k)-K1*e(k-1)+K2*e(k-2)  (Q15)
                SACH  pid_out_reg1         ; u(k) = u(k-1)+K0*e(k)-K1*e(k-1)+K2*e(k-2)  (Q15)
; If u(k) > u_max, u(k) = u_max. If u(k) < u_min, u(k) = u_min.
                LACC  pid_out_reg1         ; ACC = u(k)  (Q15)
  ;  SUB                #PID_OUT_MAX_; ACC = u(k)-u_max  (Q15)
     SUB    #PID_OUT_MAX_
    BCND       SAT_MAX,GT                 ; Branch if saturated at max
    LACC       pid_out_reg1               ; ACC = u(k)   (Q15)
    SUB                #PID_OUT_MIN_      ; ACC = u(k)-u_min  (Q15)
    BCND       SAT_MIN,LT                 ; Brnch if saturated at min
       B               REG1_END
SAT_MIN
    SPLK       #PID_OUT_MIN_,pid_out_reg1 ; u(k) = u_min    (Q15)
       B               REG1_END
SAT_MAX
    SPLK   #PID_OUT_MAX_,pid_out_reg1     ; u(k) = u_max            (Q15)
REG1_END
; Updating the errors e(k-1), e(k-2) and output u(k-1)
    LACC       pid_e1_reg1     ; ACC = e(k-1)          (Q15)
    SACL       pid_e2_reg1     ; e(k-2) = e(k-1)       (Q15)
   LACC        pid_e0_reg1     ; ACC = e(k)            (Q15)
   SACL        pid_e1_reg1     ; e(k-1) = e(k)         (Q15)
   LACC        pid_out_reg1    ; ACC = u(k)            (Q15)
   SACL        pid_out1_reg1   ; u(k-1) = u(k)         (Q15)
               CLRC  SXM
               RET
```

; Module Name: RAMP_CNTL

```
;=============================================================================
; Description:  This module implements a ramp up and ramp down function.
;               The output flag variable s_eq_t_flg is set to 7FFFh when the
;               output variable setpt_value equals the input variable target_value.
;=============================================================================
;Module definitions for external reference.
               .def   RAMP_CNTL, RAMP_CNTL_INIT      ; function call
               .def   target_value                   ; Inputs
               .def   rmp_dly_max, rmp_lo_limit      ; Input Parameters
               .def   rmp_hi_limit                   ; Input Parameter
               .def   setpt_value, s_eq_t_flg        ; Outputs
```

```
;=================================================================
target_value    .usect "rmp_cntl",1
setpt_value     .usect "rmp_cntl",1
s_eq_t_flg      .usect "rmp_cntl",1
rmp_delay_cntr  .usect "rmp_cntl",1
rmp_dly_max     .usect "rmp_cntl",1
rmp_lo_limit    .usect "rmp_cntl",1
rmp_hi_limit    .usect "rmp_cntl",1
RAMP_CNTL_INIT:
                LDP     #setpt_value
                SPLK    #0h,setpt_value
                SPLK    #0h,rmp_delay_cntr
                SPLK    #1,rmp_dly_max
                SPLK    #0300h,rmp_lo_limit
                SPLK    #07500h,rmp_hi_limit
                RET
RAMP_CNTL:
                LDP     #target_value
                LACC    target_value
                SUB     setpt_value
                BCND    SET_FLG, EQ             ; If Set point = target
                                                ; set s_eq_t_flg = 7FFFh then exit
                LACC    rmp_delay_cntr
                ADD     #1
                SACL    rmp_delay_cntr
                SUB     rmp_dly_max
                BCND    SRC_EXIT, LT
CHNG_VALUE:
                LACC    target_value
                SUB     setpt_value
                BCND    INC_VALUE, GT
DEC_VALUE       LACC    setpt_value
                SUB     #1
                SACL    setpt_value
                SUB     rmp_lo_limit
                BCND    SRC_1, GEQ
                LACC    rmp_lo_limit
                SACL    setpt_value
                B       SRC_1
INC_VALUE       LACC    setpt_value
                ADD     #1
                SACL    setpt_value
                SUB     rmp_hi_limit
                BCND    SRC_1, LEQ
                LACC    rmp_hi_limit
                SACL    setpt_value
SRC_1:
                SPLK    #0, rmp_delay_cntr
SRC_EXIT
                RET
SET_FLG
                SPLK    #7FFFh, s_eq_t_flg
                RET
```

; Module Name: V_Hz_PROFILE

```
;=============================================================================
; Description:    This module generates the output command voltage for a
;                 specific input command frequency according to specified
;                 volts/hertz profile. This is used for variable speed
;                 implementation of AC induction motor drives.
;        vhz_freqo------>| V_Hz_PROFILE  |----->o v_out
;=============================================================================
;Module definitions for external reference.
                .def    V_Hz_PROFILE, V_Hz_PROFILE_INIT         ;function call
                .def    vhz_freq                        ;Inputs
                .def    v_out                                   ;Outputs
                .def    FL, FH, Fmax, Vmax, Vmin, vf_slope      ;Parameters
;=============================================================================
; Config Information:
;       For FL = 20%, FL = .2 x 32767 = 6553
;       For FH = 90%, FH = .9 x 32767 = 29490
;       For Vmin = 20%, Vmin = .2 x 32767 = 6553
;       VF_SLOPE = (Vmax - Vmin)/(FH - FL) x 4096
;                = (1 - 0.2)/(0.9 - 0.2) x 4097 = 4681
                .include x24x_app.h
;User configurable default parameter values
FL_             .set    6553            ;Low Freq point on profile(Q15)
FH_             .set    29490           ;High Freq point on profile(Q15)
Fmax_           .set    7FFFh   ;Max value (i.e. 0.999.. in Q15)
VF_SLOPE_       .set    4681            ;Volts/Hz slope 2.67 in Q12 format
Vmax_           .set    32765           ;0.9999  in Q15
Vmin_           .set    6553            ;0.200..  in Q15 (This is also the offset)
;User configurable default parameter values
;FL_            .set    6553            ;Low Freq point on profile(Q15)
;FH_            .set    16384           ;High Freq point on profile(Q15)
;Fmax_          .set    7FFFh           ;Max value (i.e. 0.999.. in Q15)
;VF_SLOPE_      .set    10243           ;Volts/Hz slope 2.67 in Q12 format
;Vmax_          .set    31129           ;0.95  in Q15
;Vmin_          .set    6553            ;0.200..  in Q15 (This is also the offset)
;-----------------------------------------------------------------------------
vhz_freq        .usect  "vhz_prof",1
v_out           .usect  "vhz_prof",1
vf_slope        .usect  "vhz_prof",1
FL                      .usect  "vhz_prof",1
FH                  .usect      "vhz_prof",1
Fmax            .usect  "vhz_prof",1
Vmax            .usect  "vhz_prof",1
Vmin            .usect  "vhz_prof",1
GPR0_vhz        .usect  "vhz_prof",1
;-----------------------------------------------------------------------
V_Hz_PROFILE_INIT:
;-----------------------------------------------------------------------
                LDP     #vhz_freq
                SPLK    #VF_SLOPE_, vf_slope
                splk    #FL_, FL
                splk    #FH_, FH
                splk    #Fmax_, Fmax
                splk    #Vmax_, Vmax
                splk    #Vmin_, Vmin
```

```
                RET
;-----------------------------------------------------------------
V_Hz_PROFILE:
;-----------------------------------------------------------------
                LDP     #vhz_freq
PROFILE1        LACC    vhz_freq
                SUB     FL                      ;Is Freq<=FL
                BCND    PROFILE2, GT
                LACC    Vmin
                SACL    v_out                   ;V is in Q15
                B       V_Hz_END
PROFILE2        LACC    vhz_freq
                SUB     FH
                BCND    PROFILE3, GT
                LACC    vhz_freq        ;Acc = FREQ_IN
                SUB     FL                      ;Acc = FREQ_IN - FL
                SACL    GPR0_vhz
                LT      GPR0_vhz
                MPY     vf_slope        ;P = vf_slope * (FREQ_IN - FL)
                PAC                             ;Q12 * Q15 --> Q27
                SACH    v_out,4         ;convert result to Q15 format
                LACC    v_out
                ADD     Vmin                    ;Offset is in Q15
                SACL    v_out                   ;v_out = vf_slope * (FREQ_IN - FL) + Vmin
                B       V_Hz_END
PROFILE3        LACC    Vmax
                SACL    v_out                   ;v_out is in Q15
V_Hz_END:
                RET
```

; Module Name: SVGEN_MF

```
;=================================================================================
; 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
; magnitude and frequency.
;       sv_gain o------>|                |----->o Ta
;       sv_offseto----->|  SVGEN_MF      |----->o Tb
;       sv_freqo------->|                |----->o Tc
;=================================================================================
;Module definitions for external reference.
                .def    SVGEN_MF, SVGEN_MF_INIT         ;function call
                .def    sv_gain, sv_offset, sv_freq     ;Inputs
                .def    Ta, Tb, Tc                      ;Outputs
                .def    sv_freq_max
;=================================================================================
;STEP_ANGLE_SV_MAX      .set    2359    ;corresponds to 120Hz frequency.
                                        ;(7FFFh = 120Hz) 20kHz PWM
STEP_ANGLE_SV_MAX       .set    5190    ;corresponds to 66Hz frequency.
                                        ;(7FFFh = 66Hz)  5kHz PWM
                .include x24x_app.h
ALPHA_SV        .usect  "svgen_mf",1
STEP_ANGLE_SV   .usect  "svgen_mf",1
ENTRY_NEW       .usect  "svgen_mf",1
ENTRY_OLD       .usect  "svgen_mf",1
```

```
SR_ADDR         .usect   "svgen_mf",1
SECTOR_PTR      .usect   "svgen_mf",1
dx              .usect   "svgen_mf",1
dy              .usect   "svgen_mf",1
T               .usect   "svgen_mf",1
Ta              .usect   "svgen_mf",1
Tb              .usect   "svgen_mf",1
Tc              .usect   "svgen_mf",1
sv_gain         .usect   "svgen_mf",1
sv_offset       .usect   "svgen_mf",1
sv_freq         .usect   "svgen_mf",1
sv_freq_max     .usect   "svgen_mf",1
;=================================================================
SVGEN_MF_INIT:
;=================================================================
                ldp     #ALPHA_SV
                SPLK    #7FFFh, T               ;T = 100%
                SPLK    #0, ALPHA_SV  ;Start at 0 deg
                SPLK    #0, ENTRY_NEW           ;Clear Sine Table Pointer
                SPLK    #0, SECTOR_PTR          ;Clear Sector Pointer
                SPLK    #STEP_ANGLE_SV_MAX, sv_freq_max
                SPLK    #3FFFh, sv_gain         ;Init amplitude to 0.5
                SPLK    #0h, sv_offset          ;Init offset to 0
                SPLK    #3FFFh, sv_freq         ;Init freq to 50%
                RET
;=================================================================
SVGEN_MF:
;=================================================================
        ;Normalise the freq input to appropriate step angle
                ldp     #sv_freq
                LT      sv_freq                 ;sv_freq is in Q15
                MPY     sv_freq_max             ;sv_freq_max is in Q0
                PAC                             ,P = Q0 x Q15 = Q15 (in 32bit word)
                SACH    STEP_ANGLE_SV,1         ;shift 1 to restore Q0 format
        ;Calculate new angle ALPHA
                LACC    ENTRY_NEW
                SACL    ENTRY_OLD
                LACC    ALPHA_SV
                ADD     STEP_ANGLE_SV           ;Inc angle.
                SACL    ALPHA_SV
                LACC    ALPHA_SV,8
                SACH    ENTRY_NEW
                LACC    #STABLE60
                ADD     ENTRY_NEW
                TBLR    dy                      ;dy=Sin(ALPHA)
                LACC    #0FFh                   ;ACC=60 deg
                SUB     ENTRY_NEW
                ADD     #STABLE60
                TBLR    dx                      ;dx=Sin(60-ALPHA)
        ;Determine which Sector
                LACC    ENTRY_NEW
                SUB     ENTRY_OLD
                BCND    BRNCH_SR, GEQ           ;If negative need to change Sector
                                                ;If positive continue
MODIFY_SEC_PTR:
                LACC    SECTOR_PTR
```

```
                SUB     #05h                    ;Check if at last sector (S6)
                BCND    PISR1,EQ                ;If yes, re-init AR1= 1st Sector (S1)
                LACC    SECTOR_PTR              ;If no, select next Sector (Sn->Sn+1)
                ADD     #01h
                SACL    SECTOR_PTR              ;i.e. inc SECTOR_PTR
                B       BRNCH_SR
PISR1           SPLK    #00, SECTOR_PTR         ;Reset Sector pointer to 0
BRNCH_SR:
                LACC    #SECTOR_TBL
                ADD     SECTOR_PTR
                TBLR    SR_ADDR
                LACC    SR_ADDR
                BACC
;---------------------------------------------------------------
;Sector 1 calculations - a,b,c --> a,b,c
;---------------------------------------------------------------
SECTOR_SR1:
                LACC    T               ;Acc = T
                SUB     dx              ;Acc = T-dx
                SUB     dy              ;Acc = T-dx-dy
                SFR                     ;Acc = Ta = 1/2(T-dx-dy)  <A>
                SACL    Ta
                ADD     dx      ;Acc = Tb = dx+Ta          <B>
                SACL    Tb
                LACC    T       ;ACC = T
                SUB     Ta      ;ACC = T-Ta
                SACL    Tc      ;ACC = Tc = T-Ta           <C>
                B       SV_END
;---------------------------------------------------------------
;Sector 2 calculations - a,b,c --> b,a,c  &  dx <--> dy
;---------------------------------------------------------------
SECTOR_SR2:
                LACC    T       ;Acc = T
                SUB     dx              ;Acc = T-dx
                SUB     dy              ;Acc = T-dx-dy
                SFR                     ;Acc = Tb = 1/2(T-dx-dy)  <A>
                SACL    Tb
                ADD     dy      ;Acc = Ta = dy+Tb          <B>
                SACL    Ta
                LACC    T       ;ACC = T
                SUB     Tb              ;ACC = T-Tb
                SACL    Tc      ;ACC = Tc = T-Tb           <C>
                B       SV_END
;---------------------------------------------------------------
;Sector 3 calculations - a,b,c --> c,a,b
;---------------------------------------------------------------
SECTOR_SR3:
                LACC    T       ;Acc = T
                SUB     dx              ;Acc = T-dx
                SUB     dy              ;Acc = T-dx-dy
                SFR                     ;Acc = Tc = 1/2(T-dx-dy)  <A>
                SACL    Tb
                ADD     dx              ;Acc = Ta = dx+Tc          <B>
                SACL    Tc
                LACC    T               ;ACC = T
                SUB     Tb              ;ACC = T-Tc
```

```
                SACL    Ta              ;ACC = Tb = T-Tc          <C>
                B       SV_END
;---------------------------------------------------------------
;Sector 4 calculations - a,b,c --> c,b,a  &  dx <--> dy
;---------------------------------------------------------------
SECTOR_SR4:
                LACC    T       ;Acc = T
                SUB     dx              ;Acc = T-dx
                SUB     dy              ;Acc = T-dx-dy
                SFR                     ;Acc = Tc = 1/2(T-dx-dy)  <A>
                SACL    Tc
                ADD     dy      ;Acc = Tb = dx+Ta         <B>
                SACL    Tb
                LACC    T       ;ACC = T
                SUB     Tc              ;ACC = T-Tc
                SACL    Ta      ;ACC = Ta = T-Tc          <C>
                B       SV_END
;---------------------------------------------------------------
;Sector 5 calculations - a,b,c --> b,c,a
;---------------------------------------------------------------
SECTOR_SR5:
                LACC    T       ;Acc = T
                SUB     dx              ;Acc = T-dx
                SUB     dy              ;Acc = T-dx-dy
                SFR                     ;Acc = Tb = 1/2(T-dx-dy)  <A>
                SACL    Tc
                ADD     dx      ;Acc = Tc = dx+Ta         <B>
                SACL    Ta
                LACC    T       ;ACC = T
                SUB     Tc              ;ACC = T-Tb
                SACL    Tb      ;ACC = Ta = T-Tb          <C>
                B       SV_END
;---------------------------------------------------------------
;Sector 6 calculations - a,b,c --> a,c,b  &  dx <--> dy
;---------------------------------------------------------------
SECTOR_SR6:
                LACC    T       ;Acc = T
                SUB     dx              ;Acc = T-dx
                SUB     dy              ;Acc = T-dx-dy
                SFR                     ;Acc = Ta = 1/2(T-dx-dy)  <A>
                SACL    Ta
                ADD     dy      ;Acc = Tc = dx+Ta         <B>
                SACL    Tc
                LACC    T       ;ACC = T
                SUB     Ta              ;ACC = T-Ta
                SACL    Tb      ;ACC = Tb = T-Ta          <C>
SV_END:
        ;Multiply by 2 & modify Ta output with input gain & offset
                LACC    Ta
                SUB     #3FFFh
                SACL    Ta,1
                LT      Ta                      ;Ta is in Q15
                MPY     sv_gain         ;sv_gain is in Q15
                PAC                             ;P = sg_gain * Ta
                SACH    Ta,1                    ;shift 1 to restore Q15 format
        ;add offset value to Ta
```

```
                LACC   Ta
                ADD    sv_offset
                SACL   Ta
        ;Multiply by 2 & modify Tb output with input gain & offset
                LACC   Tb
                SUB    #3FFFh
                SACL   Tb,1
                LT     Tb                    ;Tb is in Q15
                MPY    sv_gain               ;sv_gain is in Q15
                PAC                          ;P = sg_gain * Tb
                SACH   Tb,1                  ;shift 1 to restore Q15 format
        ;add offset value to Tb
                LACC   Tb
                ADD    sv_offset
                SACL   Tb
        ;Multiply by 2 & modify Tc output with input gain & offset
                LACC   Tc
                SUB    #3FFFh
                SACL   Tc,1
                LT     Tc                    ;Tc is in Q15
                MPY    sv_gain               ;sv_gain is in Q15
                PAC                          ;P = sg_gain * Tc
                SACH   Tc,1                  ;shift 1 to restore Q15 format
        ;add offset value to Tb
                LACC   Tc
                ADD    sv_offset
                SACL   Tc
                RET
;-------------------------------------------------------
;SVPWM Sector routine jump table - used with BACC inst.
;-------------------------------------------------------
SECTOR_TBL:
SR0     .word   SECTOR_SR1
SR1     .word   SECTOR_SR2
SR2     .word   SECTOR_SR3
SR3     .word   SECTOR_SR4
SR4     .word   SECTOR_SR5
SR5     .word   SECTOR_SR6
;-------------------------------------------------------
;-------------------------------------------------------
;Sine table (0 - 60 deg) used for Space Vector Generator.
;No. Samples    256     Angle Range     60
;-------------------------------------------------------
;       SINVAL          ;       Index   Angle Sin(Angle)
STABLE60:
    .word       0       ;           0   0           0.00
    .word       134     ;           1   0.23        0.00
    .word       268     ;           2   0.47        0.01
    .word       402     ;           3   0.70        0.01

.....
    .word       28106   ;       252     59.06   0.86
    .word       28175   ;       253     59.30   0.86
    .word       28243   ;       254     59.53   0.86
    .word       28311   ;       255     59.77   0.86
```

```
; Module Name:      FC_PWM_DRV
;=================================================================================
; 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_p           o------>|                  |----->o T1PER (EV register)
; n_period      o------>|_____________________|
;=================================================================================
; Define Related Peripherals
;-------------------------------
                .include "x24x_app.h"
; Default PWM Period
;-------------------------
;PWM_PERIOD .set     50              ; PWM period in uS (20KHz)
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

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 x240
                SPLK    #COMCON_+8000h,COMCON
                .endif
                .if x243|x2407
```

```
                ldp     #OCRA>>7                ; Configure 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
```

Module Name: S Y S _ I N I T

```
;===============================================================
; Description:  Initializes F24x/xx devices
                .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.
                SPM     0
                LAR     AR1, #stack_start       ;Init s/w stack pointer
                MAR     *,AR1
                POINT_B0
                SPLK    #00C0h, GPR0    ;Set 1 wait states for I/O space
                OUT     GPR0, 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
```

; Module Name: DATA_LOG

```
;=============================================================================
; Description:  Stores the realtime values of two user selectable s/w variables in the external data RAM
; provided on the 24x/24xx EVM. Two s/w variables are selected by configuring two module inputs,
;dlog_iptr1 and dlog_iptr2, point to the address of the two variables. The starting addresses of the two
;RAM locations, where the data values are stored, are set        to 8000h and 8400h. Each section allows
;logging of 400 data values.
;       dlog_iptr1      o------>|                               |
;       dlog_iptr2      o------>|  DATA_LOG/ Data RAM           |
;                       o------>|                               |
;=============================================================================
;Module definitions for external reference.
```

```
                .def        DATA_LOG, DATA_LOG_INIT          ;function call
                .def        dlog_iptr1, dlog_iptr2           ;Inputs
                .def        trig_value                                ;Inputs
;=================================================================================
                .include x24x_app.h
DLOG_PRESCALE   .set    1                       ;1 gives 50uS log rate
DL_BUFFER1_ADR  .set    08000h
DL_BUFFER2_ADR  .set    08400h
DLOG_CNTR_MAX   .set    400
dlog_iptr1      .usect "data_log",1
dlog_iptr2      .usect "data_log",1
dlog_skip_cntr  .usect "data_log",1
dlog_cntr       .usect "data_log",1
task_ptr .usect "data_log",1
graph_ptr1      .usect "data_log",1
graph_ptr2      .usect "data_log",1
trig_value      .usect "data_log",1
;---------------------------------------------------------------------------
DATA_LOG_INIT:
;---------------------------------------------------------------------------
                LDP     #task_ptr
                SPLK    #POS_TRIG_S1, task_ptr
                SPLK    #DL_BUFFER1_ADR, graph_ptr1
                SPLK    #DL_BUFFER2_ADR, graph_ptr2
                SPLK    #300h, dlog_iptr1
                SPLK    #300h, dlog_iptr2
                SPLK    #0h, trig_value         ;Set trig point to 0h
                SPLK    #0h, dlog_skip_cntr     ;clear Data log skip counter
                SPLK    #0h, dlog_cntr          ;clear Data log counter
                RET
;---------------------------------------------------------------------------
DATA_LOG:
;---------------------------------------------------------------------------
                LDP     #task_ptr
                MAR     *, AR5
                SETC    SXM
                LACC    task_ptr
                BACC
POS_TRIG_S1:    ;(Positive trigger, detect Negative signal)
                LAR     AR5, dlog_iptr1
                LACC    *,1
                SUB     trig_value,1
                BCND    DL_EXIT, GEQ                    ;If not Neg return
                SPLK    #POS_TRIG_S2, task_ptr
                B       DL_EXIT
POS_TRIG_S2:    ;(Positive trigger, detect Positive signal)
                LAR     AR5, dlog_iptr1
                LACC    *,1
                SUB     trig_value,1
                BCND    PTS2_1, LT                      ;If Neg, return to S1
                SPLK    #DL_TRIGGERED, task_ptr
                B       DL_EXIT
PTS2_1:         SPLK    #POS_TRIG_S1, task_ptr
                B       DL_EXIT
DL_TRIGGERED:
                LACC    dlog_skip_cntr          ;Check if time to Log
```

```
                ADD     #1
                SACL    dlog_skip_cntr
                SUB     #DLOG_PRESCALE
                BCND    DL_EXIT, NEQ
        ;Data is logged here
                LAR     AR5, dlog_iptr1
                LACC    *
                LAR     AR5, graph_ptr1
                SACL    *+
                SAR     AR5, graph_ptr1
                LAR     AR5, dlog_iptr2
                LACC    *
                LAR     AR5, graph_ptr2
                SACL    *+
                SAR     AR5, graph_ptr2
                SPLK    #0, dlog_skip_cntr
                LACC    dlog_cntr                       ;Check if Data buffer full
                ADD     #1
                SACL    dlog_cntr
                SUB     #DLOG_CNTR_MAX
                BCND    DL_EXIT, NEQ
                SPLK    #0, dlog_cntr
                SPLK    #POS_TRIG_S1, task_ptr
                SPLK    #DL_BUFFER1_ADR, graph_ptr1
                SPLK    #DL_BUFFER2_ADR, graph_ptr2
DL_EXIT
                CLRC    SXM
                RET
```

Appendix C

Software Package for FOC Control Scheme

```
;=====================================================================
;
; File Name:   FOC_MRAS.ASM
;
; Description: Sensorless control of Induction Motor by FOC technique
;              using MRAS speed estimator
; Originator:  Chaozheng Ma, Ryerson University, Toronto, Canada.
; Target dependency: DSP eZdsp™ LF2407, use 5kHz sampling frequency
;---------------------------------------------------------------------
; May15, 2003
;=====================================================================
;*********************************************************************
;                          SYSTEM OPTIONS
;*********************************************************************
real_time               .set        1    ; 1 for real time mode, otherwise set 0

phase_commissioning1    .set    0    ; Ramp and sine_wave signal generation
phase_commissioning2    .set    0    ; open_loop start up
phase_commissioning3    .set    0    ; current measurement chain checking
phase_commissioning31   .set    0    ; current measurement chain checking
phase_commissioning32   .set    0    ; current measurement chain checking
phase_commissioning3c   .set    0    ; current regulation implementation
phase_commissioning4    .set    0    ; current regulation implementation
phase_commissioning5    .set    1    ; Mras estimation
phase_commissioning5a   .set    0    ; Mras estimation
phase_commissioning5b   .set    0    ; Mras estimation
phase_commissioning5c   .set    0    ; use RAMP

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
*Park_and_speed         .set   0   ; check of park_D/park_Q and speed_frq
      .endif
      .if (phase_commissioning4)
constant_current        .set   0   ; Theta/Theta_ip switch (0 or rmp_out).
park_sign               .set   0   ; check of park_D/park_Q sign
      .endif
      .if (phase_commissioning5)
current_mod             .set   0 ; Theta/Theta_ip switch (rmp_out or cur_mod).
                                 ; 0--rmp_out 1--theta_cur_mod
;constant_current       .set   0   ; 1--iq_ref=0,0--iq_ref=my_iq_ref
      .endif
```

```
        .if (phase_commissioning5b)
loop_cnt1_      .set    10000 ;
loop_cnt2_      .set    100   ;
        .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
; Add by C. Ma
                .ref    DATA_LOG, DATA_LOG_INIT         ;function call
                .ref    dlog_iptr1, dlog_iptr2          ;Inputs
                .ref    trig_value                      ;Inputs

                .ref    PHASE_VOLTAGE_CALC              ;function call
                .ref    PHASE_VOLTAGE_CALC_INIT         ;function call
                .ref    Mfunc_V1, Mfunc_V2              ;Inputs
                .ref    Mfunc_V3, DC_bus                ;Inputs
                .ref    Vphase_A,Vphase_B,Vphase_C      ;Outputs
                .ref    Vdirect,Vquadra                 ;Outputs
;done
                .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, Ic_out          ;Outputs
; add by C. Ma
                .ref    Vdc_meas_offset, Vdc_meas_gain, Vdc_meas
                .ref    Vdc_meas_gain_1, Vdc_meas_1
;done
                .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    CURRENT_MODEL,CURRENT_MODEL_INIT; function call
                .ref    i_cur_mod_D,i_cur_mod_Q                 ; Inputs
                .ref    spd_cur_mod                                     ; Input
                .ref    theta_cur_mod                           ; Outputs
;add by C. Ma
                .ref    ACI_MRAS, ACI_MRAS_INIT                 ;function call
                .ref    ualfa_mras, ubeta_mras                  ;Inputs
                .ref    ialfa_mras, ibeta_mras                  ;Inputs
                .ref    wr_hat_mras, wr_hat_rpm_mras            ;Outputs
;done
                .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
                .ref    ramp_theta               ; add

                .ref    I_CLARKE, I_CLARKE_INIT     ;function call
                .ref    Iclark_d, Iclark_q          ;Inputs
                .ref    Iclark_a, Iclark_b, Iclark_c        ;Outputs
; add by C Ma
                .ref    BC_INIT,BC_CALC             ;function call
                .ref    BC_IN,BC_OUT                ;Inputs/Outputs
;done
;-------------------------------------------------------------------------
; Variable Declarations
;-------------------------------------------------------------------------
                .def    GPR0                    ;General purpose registers.
 ; add by C Ma
                .def  loop_cnt1,loop_cnt2
*               .def  theta_est_mod  ;1--rmp_out / 0-- theta_est
                .bss  loop_cnt1,1
                .bss  loop_cnt2,1
;done
                .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
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"

;=============================================================================
; 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
                CALL   DATA_LOG_INIT
        .if (phase_commissioning1)
                CALL   RAMP_GEN_INIT
                CALL   I_PARK_INIT
;add by C Ma
      LDP #dlog_iptr1
      SPLK #ipark_d, dlog_iptr1;
      SPLK #rmp_out, dlog_iptr2
;done
        .endif
        .if (phase_commissioning2)
                CALL   RAMP_GEN_INIT
                CALL   I_PARK_INIT
                CALL   SVGEN_DQ_INIT
      LDP #dlog_iptr1
      SPLK #Ta, dlog_iptr1;
      SPLK #ipark_q, dlog_iptr2
        .endif
        .if (phase_commissioning3)
                CALL   RAMP_GEN_INIT
                CALL   I_PARK_INIT
                CALL   SVGEN_DQ_INIT
                 CALL   CLARKE_INIT
                CALL   PARK_INIT
;changed by C Ma
                CALL   ILEG2DRV_INIT
                CALL   PHASE_VOLTAGE_CALC_INIT
       LDP #dlog_iptr1
       SPLK #Ta, dlog_iptr1;
       SPLK #Ib_out, dlog_iptr2
        .endif
        .if (phase_commissioning31)
                CALL   RAMP_GEN_INIT
                CALL   I_PARK_INIT
                CALL   SVGEN_DQ_INIT
```

```
                CALL   ILEG2DRV_INIT
                CALL   CLARKE_INIT
                 CALL   PARK_INIT
                CALL   PHASE_VOLTAGE_CALC_INIT
          LDP #dlog_iptr1
          SPLK #Ta, dlog_iptr1;
          SPLK #Ib_out, dlog_iptr2
          .endif

         .if (phase_commissioning32)
                CALL   RAMP_GEN_INIT
                CALL   I_PARK_INIT
                CALL   SVGEN_DQ_INIT
                CALL   ILEG2DRV_INIT
                CALL   CLARKE_INIT
                CALL   PARK_INIT
        CALL   PHASE_VOLTAGE_CALC_INIT
        CALL   ACI_MRAS_INIT
        CALL   BC_INIT
        LDP #dlog_iptr1;
        SPLK #Ta, dlog_iptr1;
        SPLK #Ib_out, dlog_iptr2
  ;done
         .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   pid_reg_id_init
                CALL   pid_reg_iq_init
      LDP #dlog_iptr1; add by C Ma
      SPLK #rmp_out, dlog_iptr2;
      splk #Ia_out, dlog_iptr1
        .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   pid_reg_id_init
                CALL   pid_reg_iq_init
                CALL   CURRENT_MODEL_INIT
;add by C. Ma
    CALL   PHASE_VOLTAGE_CALC_INIT
    CALL   ACI_MRAS_INIT
    CALL   BC_INIT
LDP #dlog_iptr1
    SPLK #rmp_out, dlog_iptr2;
    splk #Ia_out, dlog_iptr1
;done
        .endif
```

```
        .if (phase_commissioning5a)
                CALL   RAMP_GEN_INIT
                CALL   I_PARK_INIT
                CALL   SVGEN_DQ_INIT
                CALL   ILEG2DRV_INIT
                CALL   CLARKE_INIT
                CALL   PARK_INIT
                CALL   pid_reg_id_init
                CALL   pid_reg_iq_init
                CALL   CURRENT_MODEL_INIT
;add by C. Ma
    CALL   PHASE_VOLTAGE_CALC_INIT
    CALL   ACI_MRAS_INIT
    CALL   BC_INIT
    CALL   pid_reg_spd_init
    LDP #dlog_iptr1;
    SPLK #Ia_out, dlog_iptr2;
    splk #Ta, dlog_iptr1
;done
   .endif
.if (phase_commissioning5b)
                CALL   RAMP_GEN_INIT
                CALL   I_PARK_INIT
                CALL   SVGEN_DQ_INIT
                CALL   ILEG2DRV_INIT
                CALL   CLARKE_INIT
                CALL   PARK_INIT
                CALL   pid_reg_id_init
                CALL   pid_reg_iq_init
                CALL   CURRENT_MODEL_INIT
;add by C. Ma
    CALL   PHASE_VOLTAGE_CALC_INIT
    CALL   ACI_MRAS_INIT
    CALL   BC_INIT
    CALL   pid_reg_spd_init
    LDP #dlog_iptr1
    SPLK #rmp_out, dlog_iptr2;
    splk #Ia_out, dlog_iptr1
;done
   .endif
.if (phase_commissioning5c)
                CALL   RAMP_GEN_INIT
                CALL   I_PARK_INIT
                CALL   SVGEN_DQ_INIT
                CALL   ILEG2DRV_INIT
                CALL   CLARKE_INIT
                CALL   PARK_INIT
                CALL   pid_reg_id_init
                CALL   pid_reg_iq_init
                CALL   CURRENT_MODEL_INIT
;add by C. Ma
    CALL   PHASE_VOLTAGE_CALC_INIT
    CALL   ACI_MRAS_INIT
    CALL   BC_INIT
    CALL   pid_reg_spd_init
```

```
      LDP #dlog_iptr1;
      SPLK #Ia_out, dlog_iptr2;
      splk #Ta, dlog_iptr1
;done
   .endif
;---Real Time option----------
         .if (real_time)
         CALL   MON_RT_CNFG                    ;For Real-Time
         .endif
;---------------------------------
;   Variables initialization
;---------------------------------
          LDP   #rmp_freq
          SPLK  #5000,rmp_freq; 60Hz frequency for RAMPGEN(5kHz)/=25770d
*         SPLK  #12884,rmp_freq; 60Hz frequency for RAMPGEN(10kHz)/=25770d
*         ldp   #theta_est_mod
*         splk  #1,theta_est_mod ;1--rmp_out / 0-- theta_est
       .if (phase_commissioning5b)
     ldp  #loop_cnt1
     splk #loop_cnt1_, loop_cnt1
     ldp  #loop_cnt2
     splk #loop_cnt2_,loop_cnt2
     .endif

     LDP   #my_iq_ref
     SPLK  #0000h,my_iq_ref
     LDP   #my_id_ref
     SPLK  #5000,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    #0000000000000100b,IMRC ;Enable CAP3 int (i.e. QEP index pulse)
                           ;5432109876543210
                  SPLK    #0000000000000000b,IMRC ;Disable CAP3 int (i.e. QEP index pulse)
                  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  #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: Chaozheng Ma - Ryerson University
; Last Update:    11-Mar 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_commissioning31)
; 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
;add by C. Ma
                LDP             #DC_bus
                BLDD  #Vdc_meas_1,DC_bus
                BLDD  #Ta,Mfunc_V1
                BLDD  #Tb,Mfunc_V2
                BLDD  #Tc,Mfunc_V3
                CALL  PHASE_VOLTAGE_CALC
;done
   .endif
*****************************************
   .if (phase_commissioning32)
; 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
;add by C. Ma
          LDP         #DC_bus
                BLDD  #Vdc_meas_1,DC_bus
                BLDD  #Ta,Mfunc_V1
        BLDD  #Tb,Mfunc_V2
        BLDD  #Tc,Mfunc_V3
         CALL  PHASE_VOLTAGE_CALC

         LDP          #ualfa_mras
                BLDD  #Vdirect,ualfa_mras
                BLDD  #Vquadra,ubeta_mras
        BLDD  #clark_d,ialfa_mras
        BLDD  #clark_q,ibeta_mras
                CALL  ACI_MRAS
       LDP            #BC_IN
                BLDD  #wr_hat_mras,BC_IN
      CALL    BC_CALC
;done
   .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   ; check the internal phase_commissioning option
     .else
     BLDD    #rmp_out,theta_p
     .endif
     CALL    PARK
; 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
     .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
; PARK module
    LDP    #park_d
    BLDD    #clark_d,park_d
    BLDD    #clark_q,park_q
    .if current_mod
    SPLK    #theta_cur_mod,theta_p
    .else
    BLDD    #rmp_out,theta_p
    .endif
    CALL    PARK
; D-axis current regulator
    LDP    #id_ref
    BLDD    #my_id_ref,id_ref ; 5000d gives a current of 1.52A rms
    BLDD    #park_D,id_fdb
    CALL    pid_reg_id
; Q-axis current regulator
    LDP    #iq_ref
      ; check the internal phase_commissioning option
    BLDD    #my_iq_ref,iq_ref
    BLDD    #park_Q,iq_fdb
    CALL    pid_reg_iq
; Ramp generation module
   .if current_mod
    .else
```

```
      CALL   RAMP_GEN
      .endif
; Inverse-Park module
                  LDP             #ipark_D
                  BLDD   #ud_out,ipark_D
                  BLDD   #uq_out,ipark_Q    ; here we apply 0 or rmp_out
            .if   current_mod   ; check the internal phase_commissioning option
                  SPLK   #theta_cur_mod,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
;add by C. Ma
            LDP           #DC_bus
                  BLDD   #Vdc_meas_1,DC_bus
                  BLDD   #Ta,Mfunc_V1
                  BLDD   #Tb,Mfunc_V2
                  BLDD   #Tc,Mfunc_V3
      CALL        PHASE_VOLTAGE_CALC
          LDP           #ualfa_mras
                  BLDD   #Vdirect,ualfa_mras
                  BLDD   #Vquadra,ubeta_mras
                  BLDD   #clark_d,ialfa_mras
                  BLDD   #clark_q,ibeta_mras
                  CALL   ACI_MRAS
            LDP           #BC_IN
                  BLDD   #wr_hat_mras,BC_IN

        ;   CALL          BC_CALC
;done
* CURRENT MODEL
      LDP     #spd_cur_mod
      BLDD    #wr_hat_mras,spd_cur_mod
;     BLDD    #BC_OUT,spd_cur_mod
      BLDD    #park_D,i_cur_mod_D
      BLDD    #park_Q,i_cur_mod_Q
      CALL    CURRENT_MODEL
    .endif
*****************************************
    .if (phase_commissioning5a)
; Current leg measurement, Ileg2drv module
      CALL    ILEG2DRV
; Clarke module
      LDP     #clark_a
      BLDD    #Ia_out,clark_a
```

```
    BLDD    #Ib_out,clark_b
    CALL    CLARKE
;Loop control
; PARK module
    LDP     #park_d
    BLDD    #clark_d,park_d
    BLDD    #clark_q,park_q
    BLDD    #theta_cur_mod,theta_p
CALL    PARK
* CURRENT MODEL
    LDP     #spd_cur_mod
;   BLDD    #wr_hat_mras,spd_cur_mod
    BLDD    #BC_OUT,spd_cur_mod
    BLDD    #park_D,i_cur_mod_D
    BLDD    #park_Q,i_cur_mod_Q
    CALL    CURRENT_MODEL
; SPEED REGULATION
    LDP     #spd_ref
    bldd    #speed_reference,spd_ref
    bldd    #BC_OUT,spd_fdb
;    bldd    #wr_hat_mras,spd_fdb
    CALL    pid_reg_spd
; D-axis current regulator
    LDP     #id_ref
    BLDD    #my_id_ref,id_ref ; 5000d gives a current of 1.52A rms
    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
; Ramp generation module
;    CALL   RAMP_GEN
; Inverse-Park module
                LDP             #ipark_D
                BLDD    #ud_out,ipark_D
                BLDD    #uq_out,ipark_Q
                BLDD    #theta_cur_mod,theta_ip
        CALL    I_PARK
; Space-Vector DQ module
                LDP             #Ualfa
                BLDD    #ipark_d,Ualfa
                BLDD    #ipark_q,Ubeta
                CALL    SVGEN_DQ
;add by C. Ma
                LDP             #DC_bus
                BLDD    #Vdc_meas_1,DC_bus
                BLDD    #Ta,Mfunc_V1
                BLDD    #Tb,Mfunc_V2
                BLDD    #Tc,Mfunc_V3
        CALL    PHASE_VOLTAGE_CALC
        LDP             #ualfa_mras
        BLDD    #Vdirect,ualfa_mras
        BLDD    #Vquadra,ubeta_mras
        BLDD    #clark_d,ialfa_mras
```

```
        BLDD  #clark_q,ibeta_mras
        CALL  ACI_MRAS

          LDP          #BC_IN
               BLDD  #wr_hat_mras,BC_IN
          CALL         BC_CALC

 ; PWM driver
                LDP             #Mfunc_c1
                BLDD  #Ta,Mfunc_c1
                BLDD  #Tb,Mfunc_c2
                BLDD  #Tc,Mfunc_c3
                CALL  FC_PWM_DRV
          .endif
    .if (phase_commissioning5b)
; Current leg measurement, Ileg2drv module
    CALL    ILEG2DRV
; Clarke module
    LDP    #clark_a
    BLDD    #Ia_out,clark_a
    BLDD    #Ib_out,clark_b
    CALL    CLARKE
;Loop control
        ldp       #loop_cnt2              ;
                lacl      loop_cnt2
                bcnd      loop_loop,LEQ
        ldp       #loop_cnt1              ;
                lacl      loop_cnt1
                bcnd      rmp_loop1,LEQ
                sub       #1
                sacl      loop_cnt1
    b rmp_loop2
rmp_loop1
    ldp #loop_cnt1
    splk #loop_cnt1_,loop_cnt1
        ldp       #loop_cnt2
                lacl      loop_cnt2
                bcnd      loop_loop,LEQ
                sub       #1
                sacl      loop_cnt2
rmp_loop2
; PARK module
    LDP    #park_d
    BLDD    #clark_d,park_d
    BLDD    #clark_q,park_q
    BLDD    #rmp_out,theta_p
    CALL    PARK
; D-axis current regulator
    LDP    #id_ref
    BLDD    #my_id_ref,id_ref  ; 5000d gives a current of 1.52A rms
    BLDD    #park_D,id_fdb
    CALL    pid_reg_id
; Q-axis current regulator
    LDP    #iq_ref
        ; here we apply 0 or my_iq_ref (user)
    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    ; here we apply 0 or rmp_out
                  BLDD   #rmp_out,theta_ip
      CALL    I_PARK
      b    svpwm
loop_loop
       ; PARK module
      LDP     #park_d
      BLDD    #clark_d,park_d
      BLDD    #clark_q,park_q
      BLDD    #theta_cur_mod,theta_p
   ;   BLDD    #rmp_out,theta_p
      CALL    PARK
     ; CALL       BC_CALC
;done
* CURRENT MODEL
      LDP     #spd_cur_mod
      BLDD    #wr_hat_mras,spd_cur_mod
;       BLDD    #BC_OUT,spd_cur_mod
      BLDD    #park_D,i_cur_mod_D
      BLDD    #park_Q,i_cur_mod_Q
      CALL    CURRENT_MODEL
; D-axis current regulator
      LDP     #id_ref
      BLDD    #my_id_ref,id_ref  ; 5000d gives a current of 1.52A rms
      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    ; here we apply 0 or rmp_out
                  BLDD   #theta_cur_mod,theta_ip
     CALL    I_PARK

; SPEED REGULATION
      LDP    #spd_ref
      bldd   #speed_reference,spd_ref
*     bldd   #spd_cur_mod,spd_fdb
      bldd   #wr_hat_rpm_mras,spd_fdb
      CALL   pid_reg_spd
svpwm
; Space-Vector DQ module
                  LDP             #Ualfa
                  BLDD   #ipark_d,Ualfa
```

```
                BLDD  #ipark_q,Ubeta
                CALL  SVGEN_DQ
;add by C. Ma
          LDP         #DC_bus
                BLDD  #Vdc_meas_1,DC_bus
                BLDD  #Ta,Mfunc_V1
                BLDD  #Tb,Mfunc_V2
                BLDD  #Tc,Mfunc_V3
          CALL  PHASE_VOLTAGE_CALC
                LDP           #ualfa_mras
                BLDD  #Vdirect,ualfa_mras
                BLDD  #Vquadra,ubeta_mras
                BLDD  #clark_d,ialfa_mras
                BLDD  #clark_q,ibeta_mras
                CALL  ACI_MRAS
          LDP         #BC_IN
                BLDD  #wr_hat_mras,BC_IN
; PWM driver
                LDP           #Mfunc_c1
                BLDD  #Ta,Mfunc_c1
                BLDD  #Tb,Mfunc_c2
                BLDD  #Tc,Mfunc_c3
                CALL  FC_PWM_DRV
   .endif
.if (phase_commissioning5c)
; Current leg measurement, Ileg2drv module
     CALL    ILEG2DRV
; Clarke module
     LDP    #clark_a
     BLDD    #Ia_out,clark_a
     BLDD    #Ib_out,clark_b
     CALL    CLARKE
;Loop control
* Ramp_Gen Module
     LDP    #rmp_freq
     BLDD    #ramp_theta,rmp_freq
;    BLDD    #BC_OUT,spd_cur_mod
     CALL   RAMP_GEN
; PARK module
     LDP    #park_d
     BLDD    #clark_d,park_d
     BLDD    #clark_q,park_q
     BLDD    #rmp_out,theta_p
     CALL    PARK
; SPEED REGULATION
     LDP    #spd_ref
     bldd   #speed_reference,spd_ref
*    bldd   #spd_cur_mod,spd_fdb
     bldd   #wr_hat_mras,spd_fdb
     CALL   pid_reg_spd
; D-axis current regulator
     LDP    #id_ref
     BLDD    #my_id_ref,id_ref  ; 5000d gives a current of 1.52A rms
     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
; Ramp generation module
;     CALL    RAMP_GEN
; Inverse-Park module
                LDP             #ipark_D
                BLDD    #ud_out,ipark_D
                BLDD    #uq_out,ipark_Q
                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
  ;add by C. Ma
                LDP             #DC_bus
                BLDD    #Vdc_meas_1,DC_bus
                BLDD    #Ta,Mfunc_V1
                BLDD    #Tb,Mfunc_V2
                BLDD    #Tc,Mfunc_V3
     CALL       PHASE_VOLTAGE_CALC
                LDP             #ualfa_mras
                BLDD    #Vdirect,ualfa_mras
                BLDD    #Vquadra,ubeta_mras
               BLDD     #clark_d,ialfa_mras
               BLDD     #clark_q,ibeta_mras
    CALL        ACI_MRAS
          LDP           #BC_IN
                BLDD    #wr_hat_mras,BC_IN
   ; CALL       BC_CALC
;done
; PWM driver
                LDP     #Mfunc_c1
                BLDD    #Ta,Mfunc_c1
                BLDD    #Tb,Mfunc_c2
                BLDD    #Tc,Mfunc_c3
                CALL    FC_PWM_DRV
  .endif
;=================================================================
;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
PHANTOM         B       PHANTOM
```

Module Name: I_CLARKE

```
;=================================================================
; Description:    Converts balanced two phase quadrature quantities into balanced
;                 three phase quantities.
;                                 (d,q) -> (a,b,c) Transformation
;                                 Iclark_a = Iclark_d
;                                 Iclark_b = (-Iclark_d + sqrt(3) * Iclark_q) / 2
;                                 Iclark_c = (-Iclark_d - sqrt(3) * Iclark_q) / 2
;                          |~~~~~~~~~~~~~~~~~~~|
;       Iclark_d o------>|                     |----->o Iclark_a
;                        |    I_CLARKE         |----->o Iclark_b
;       Iclark_q o------>|                     |----->o Iclark_c
;                        |_____________________|
;=================================================================
;Module definitions for external reference.
                .def    I_CLARKE, I_CLARKE_INIT     ;function call
                .def    Iclark_d, Iclark_q          ;Inputs
                .def    Iclark_a, Iclark_b, Iclark_c ;Outputs
;=================================================================
                .include x24x_app.h
Iclark_d .usect "I_clarke",1
Iclark_q .usect "I_clarke",1
Iclark_a .usect "I_clarke",1
Iclark_b .usect "I_clarke",1
Iclark_c .usect "I_clarke",1
half_sqrt3      .usect "I_clarke",1
;================================
I_CLARKE_INIT:
;================================
                ldp     #half_sqrt3         ; Variables data page
                SPLK    #28377,half_sqrt3 ; Set constant sqrt(3)*0.5 in Q15 format
                RET
;================================
I_CLARKE:
;================================
                ldp     #Iclark_d           ; Variables data page
                SPM     1                   ; SPM set for Q15 multiplication
                SETC    SXM                 ; Sign extension mode on
        ;Iclark_a = Iclark_q
                LACC    Iclark_q            ; ACC = Iclark_q
                SACL    Iclark_a            ; Iclark_a = Iclark_q
        ;Iclark_b = (-Ubeta + sqrt(3) * Ualfa) / 2
                LT      Iclark_d            ; TREG = Iclark_d
                MPY     half_sqrt3          ; PREG = Iclark_d * half_sqrt3
                PAC             ; ACC high = Iclark_d * half_sqrt3
                SUB     Iclark_q,15         ; ACC high = Iclark_d * half_sqrt3 + Iclark_q/2
                SACH    Iclark_b            ; Iclark_b = Iclark_d * half_sqrt3 + Iclark_q/2
        ;Iclark_c = (-Ubeta - sqrt(3) * Ualfa) / 2
                PAC             ; ACC high = Iclark_d * half_sqrt3
                NEG                         ; ACC high = - Iclark_d * half_sqrt3
                SUB     Iclark_q,15         ; ACC high = - Iclark_d * half_sqrt3 - Iclark_q/2
                SACH    Iclark_c            ; Iclark_c = - Iclark_d * half_sqrt3 - Iclark_q/2
                SPM     0                   ; SPM reset
                CLRC    SXM                         ; Sign extension mode off
                RET
```

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 )
;=============================================================================
;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
;=============================================================================
High_precision  .set    1       ;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
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

        .if (negtive_sign)
                LACC    park_D
                BGEZ    D_neg
                NEG
                SACL    park_D
    SPLK #1,D_sign
D_neg
```

```
                SPLK  #0,D_sign
                LACC   park_Q
    BGEZ   Q_neg
                NEG
                SACL   park_Q
    SPLK  #1,Q_sign
Q_neg
    SPLK  #0,Q_sign
                .endif
                SPM    0        ; SPM reset
                RET
```

Module Name: I_PARK

```
;=============================================================================
; 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  )
;=============================================================================
;(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
                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
                SPM     1                       ; 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
                SPM     0               ; SPM reset
                RET
```

Module Name: SVGEN_DQ

```
;=================================================================================
; 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
;=================================================================================
; 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  1
;---------------------------------------------------------------------------------
; 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           SPM     0               ; 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

```
;=================================================================
; Description: Initializes F24x/xx devices
;   AR1 is used as stack pointer
;=================================================================
                .include x24x_app.h
                .def    SYS_INIT
                .ref    GPR0
stack_size      .set    20h
stack_start     .usect  "stack",stack_size
SYS_INIT:
                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.
                SPM     0
                LAR     AR1, #stack_start       ;Init s/w stack pointer
                MAR     *,AR1

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

Module names: pid_reg_id, pid_reg_iq

```
;=============================================================================
; 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
;=============================================================================
*****************************************************************
* 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;554 ;783 ;  388  ; Q11, proportional gain 0.189d
Ki_d_           .set    2542;3630;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
  MPY   Kc_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    1108 ;1566;  776 ; Q11, proportional gain  0.3789d
Ki_q_           .set    5446 ;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
  MPY   Kc_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
```

Module Name: FC_PWM_DRV

```
;=================================================================================
; 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_p         o------>|                       |----->o  T1PER (EV register)
;         n_period  o------>|_____________________|
;
; Target:         x2407 Event Manager Timer1 & F Compares
;=================================================================================
; 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     100    ; PWM period in uS (10KHz) ,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
```

```
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
```

; Module Name: CURRENT_MODEL

```
;=============================================================
; Description:    Current model for field oriented control of an AC induction machine
; i_cur_mod_d ->|                    |
; i_cur_mod_q ->| CURRENT_MODEL |-----> theta_cur_mod
; spd_cur_mod-->|                    |
;-------------------------------------------------------------
; Global Definitions
;-------------------------------------------------------------
                        .def    CURRENT_MODEL,CURRENT_MODEL_INIT; function call
                        .def    i_cur_mod_D,i_cur_mod_Q          ; Inputs
                        .def    spd_cur_mod                              ; Input
                        .def    theta_cur_mod                    ; Outputs
                        .def  fs
;-------------------------------------------------------------
; Variable Definitions
;-------------------------------------------------------------
;public variables
i_cur_mod_D     .usect "cur_mod",1
i_cur_mod_Q .usect "cur_mod",1
spd_cur_mod .usect "cur_mod",1
theta_cur_mod .usect "cur_mod",1
;private variables
iSd                     .usect "cur_mod",1              ;stator current flux component
iSq                     .usect "cur_mod",1              ;stator current torque component
n                       .usect "cur_mod",1              ;rotor mechanical speed
Teta_cm         .usect "cur_mod",1              ;rotor flux position
i_mr            .usect "cur_mod",1              ;magnetizing current
fs      .usect "cur_mod",1              ;rotor flux electrical speed
tetaincr  .usect "cur_mod",1            ;electrical angle variation within
Kr      .usect "cur_mod",1      ;Q12
Kt      .usect "cur_mod",1      ;Q12
K       .usect "cur_mod",1      ;Q0
myK       .usect "cur_mod",1
my_fs     .usect "cur_mod",1
p       .usect "cur_mod",1
                                                        ;one sampling period
```

```
tmp             .usect "cur_mod",1              ;temporary variable
tmp1        .usect "cur_mod",1                  ;temporary variable
Teta_cm1            .usect "cur_mod",1                  ;vizualisation variable
;-------------------------------------------------------
; Default parameters
;-------------------------------------------------------
Pole_pairs_number  .set  2

; Parameters of the motor used with the Encoder

;Kr_     .set      0eh  ;must be calculated according to the
;Kt_     .set      1b0h ;motor parameters (check out .xls sheet)
;K_              .set      148h

; Parameters of the motor used with the Tachometer

;Kr_             .set      3h    ;must be calculated according to the
;Kt_             .set      62h   ;motor parameters (check out .xls sheet)
;K_              .set      148h

Kr_             .set      3    ;must be calculated according to the
Kt_             .set      46   ;motor parameters (check out .xls sheet)
K_              .set      786
;-------------------------------------------------------
; Initialization
;-------------------------------------------------------
CURRENT_MODEL_INIT

                ldp             #n
                splk    #0000h,n
                splk    #0000h,i_mr
                splk  #Kr_,Kr
                splk  #Kt_,Kt
                splk  #K_,K
                splk  #3FFFh,theta_cur_mod
                splk  #0000h,i_cur_mod_D
                splk  #0000h,i_cur_mod_Q
                splk  #0000h,spd_cur_mod
                splk  #0000h,iSd
                splk  #0000h,iSq
                splk  #0000h,Teta_cm
                splk  #0000h,fs
                splk  #0000h,tetaincr
                splk  #0000h,tmp
                splk  #0000h,tmp1
                splk  #0000h,Teta_cm1
                splk  #Pole_pairs_number,p
                ret
; Current model with i_mr on 16 bit
CURRENT_MODEL
; Tuning from Q15 (input variables format) to Q12 (working variables)
        ldp             #i_cur_mod_D
        lacc    i_cur_mod_D
        sfr
        sfr
        sfr
```

```
        ldp             #iSd
        sacl    iSd
        ldp             #i_cur_mod_Q
        lacc    i_cur_mod_Q
        sfr
        sfr
        sfr
        ldp             #iSq
        sacl    iSq
        ldp             #spd_cur_mod
        lacc    spd_cur_mod ;Q15, 1800rpm as pu
  sfr  ; add by C.MA
  sfr
  sfr       ; check this conversion depending on your nominal speed
        ldp             #n
        sacl    n               ;Q12, 225rpm as pu
************************************************
* Current Model
************************************************
        ldp             #iSd
        lacc    iSd
        sub     i_mr
        sacl    tmp
        lt      tmp
        mpy     #Kr
        pac
        sach    tmp,4  ;add by C. Ma
        lacc    tmp
        add       i_mr
        sacl    i_mr    ;i_mr=i_mr+Kr*(iSd-i_mr), 4.12 f
        bcnd    i_mrnotzero,NEQ
        lacc    #0
        sacl    tmp     ;if i_mr=0 then tmp=iSq/i_mr=0
        b       i_mrzero
i_mrnotzero
*** division (iSq/i_mr)
        lacc    i_mr
        bcnd    i_mrzero,EQ
        sacl    tmp1
        lacc    iSq
        abs
        sacl    tmp
        lacc    tmp,12
        rpt     #15
        subc    tmp1
        sacl    tmp     ;tmp=iSq/i_mr
        lacc    iSq
        bcnd    iSqpos,GT
        lacc    tmp
        neg
        sacl    tmp     ;tmp=iSq/i_mr, 4.12 format
iSqpos
i_mrzero
*** END division ***
        lt      tmp
        mpy     #Kt
```

```
        pac
        sach    tmp,4    ;slip frequency, 4.12 format
        lacc    tmp      ;load tmp in low ACC
        add     n
; RPT  #(Pole_pairs_number-1)     changed by Ma
 ;sfr                    changed by Ma
        sacl    fs       ;rotor flux speed, 4.12 format,
                ;fs=n+Kt*(iSq/i_mr)
*** rotor flux position calculation ***
        lacc    fs
        abs
        sacl    tmp
        lt      tmp
        mpy     #K
        pac
        sach    tetaincr,4
        bit     fs,0
        bcnd    fs_neg,TC
        lacl    tetaincr
        adds    Teta_cm
        sacl    Teta_cm
        b       fs_pos
fs_neg
        lacl    Teta_cm
        subs    tetaincr
        sacl    Teta_cm
                ;Teta_cm=Teta_cm+K*fs=Teta_cm+tetaincr
                ;(0;360)<->(0;65535)
fs_pos
        ldp             #theta_cur_mod
        sfr    ; add by Ma
        and             #7FFFh
        sacl    theta_cur_mod
************************************************
*       END Current Model
************************************************
        ret
```

Module Name: theta_estimation_model

```
;=============================================================================
; Description:    Theta estimation model for field oriented control of an AC induction machine
;    psiRal ->|                    |--->sinTeta_est
;             | theta_est          |
;   psiRbe ->|                    |--->cosTeta_est
;             |  model             |
;=============================================================================
; 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
      splk   #psiRal,psiRal_est1
      splk   #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
        saclpsiR_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
```

; Module Name: RAMP_GEN

```
;=============================================================================
; 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------>|                    |
;=============================================================================
                .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
```