Addis Ababa University

Addis Ababa Institute of Technology

Electrical and Computer Engineering Department

Observer based Speed Control of PMSM using TMS320F2812 DSP

By

Hamdihun Abdie

Advisor

Dr.-Ing. Mengesha Mamo

A Thesis Submitted to the Addis Ababa Institute of Technology, School of Graduate Studies, Addis Ababa University, in Partial Fulfillment of the Requirements for the Degree of Masters of Science in Electrical Engineering

October, 2012

Addis Ababa, Ethiopia

Addis Ababa University

Addis Ababa Institute of Technology

Electrical and Computer Engineering Department

Observer based Speed Control of PMSM using TMS320F2812 DSP

By

Hamdihun Abdie

Approval by Board of Examiners

_____	_____
Chairman, Dept. Graduate Committee	Signature
Dr.Ing. Mengesha Mamo	_____
Advisor's Name	Advisor Signature
_____	_____
Internal Examiner	Signature
_____	_____
External Examiner	Signature

Declaration

I, the undersigned, declare that this thesis is my original work, has not been presented for a degree in this or any other university, and all sources of materials used for the thesis have been duty acknowledged.

Hamdihun Abdie _____
Name Signature

Place: Addis Ababa

Date of Submission: _____

This thesis has been submitted for examination with my approval as a university advisor.

Dr.- Ing. Mengesha Mamo _____
Advisor's Name Signature

Dedicated to

My Mother

Acknowledgment

Next to Allah-The Almighty God, I would like to express my sincere gratitude to my advisor Dr. Mengesha Mamo for his continuous support of my thesis, for his patience, motivation, enthusiasm, and most importantly, his friendship. His guidance helped me in all time of the research and writing of this thesis.

Beside my advisor, I would like to thank all Stuff and Laboratory Assistants of Electrical and Computer Engineering Department for their assistance and encouragement throughout my work.

Last but not least, I would like to thank my family for their great support during my study.

Table of Contents

List of Table

List of Figures

List of Acronyms

PMSM	Permanent Magnet Synchronous Motor
DSP	Digital Signal Processor
ADC	Analog to Digital Converter
FOC	Field Oriented Control
EKF	Extended Kalman Filter
PWM	Pulse Width Modulation
SVPWM	Space Vector Pulse Width Modulation
SPWM	Sinusoidal Pulse Width Modulation
BLDM	Brush Less DC motor
EMF	Electro Motive Force
MRAS	Model Reference Adaptive Systems
JTAG	Joint Test Action Group
RAM	Random Access Memory
SRAM	Static Random Access Memory
CCS	Code Composer Studio
MIPS	Millions of Instruction Per Second
ISR	Interrupt Subroutine

Abstract

Problems due to mechanical position sensor of Permanent Magnet Synchronous Motor during speed control are alleviated by an observer for estimating instantaneous speed and position of the rotor. This needs a high performance speed and position estimator, i.e. Extended Kalman Filter (EKF), for computing the position and speed of the motor mathematically from current and voltage information using the nonlinear system model. EKF is a set of mathematical equation that provides an efficient computational means to estimate the state of a process, in a way that minimizes the mean of the squared error. It uses tedious mathematical computations recursively which is difficult to use in most digital control systems. EKF algorithm is derived from the input output state space model of the motor. It also considers system and output noise. The mathematical computation can be processed using a Texas Instrument TMS320F2812 DSP with 150MIPS processing speed which is high enough to support it. The system uses Proportional Integral (PI) controller since it is simple and popular in most industrial application. The value of the parameter gains were initially calculated by using pole and zero placements and some tuning was taken place. For the investigation of the problem, both simulation and experimental setups were done. In the simulation, EKF algorithm was written in M-file and embedded on simulation block. To make system control simple, vector control technique was used to control the torque and flux of the motor directly as DC motor. The overall system algorithm is also written in C programming language using code composer studio V3.1 for loading and running into TMS320F2812 DSP to control the motor. Our laboratory motor has no parameter information on it; hence it was difficult to run the motor in closed loop. But to check the functionality of the system Open loop motor running using TMS320F2812 DSP was done.

Keywords: PMSM, Extended Kalman Filter, Vector control, TMS320F2812 DSP, State Observer.

Chapter One

Introduction

1.1 Background

Permanent magnet synchronous motors (PMSM) are increasing interests in recent years for industrial drive applications. Their high efficiency, high steady state torque density compared to induction motor drives make them a good alternative in certain applications. Synchronous motors with permanent magnets are also becoming an attractive alternative for applications in high performance variable speed drives. Significant advantages arise from the simplification in construction, the reduction in losses and the improvement in efficiency by using digital control system for them. PMSM is a motor in which permanent magnets are rigidly fixed on the rotor to create a constant flux. When the stator windings are energized, a rotating magnetic field is created. To control the rotating magnetic field, it is necessary to control the stator currents. The interaction between the stator and rotor fluxes produces a torque. Since the stator is firmly fixed to the frame, the rotor is free to rotate for producing a useful mechanical output. The rotating stator field must rotate at the same frequency as the rotor permanent magnetic field: otherwise the rotor will experience rapidly alternating positive and negative torque. This will result in an excessive mechanical vibration, noise, and mechanical stress on the machine parts. For digital speed control of PMSM motor, electronic switching power supplies such as Inverters are needed. Determining the switching sequence and timing of such devices is the basic thing in digital control of PMSM. This can be done using digital signal processors.

Digital control of switching power supplies is becoming more and more common in industry today because of the availability of low cost, high performance DSP controller with enhanced and integrated power electronic peripherals such as analog-to-digital converters (ADC) and pulse width modulation (PWM). DSP based control allows for the implementation of more functional control schemes. In addition, digital controllers are less susceptible to aging and environmental variations and have better noise immunity. Modern 32-bit DSP controllers with processor speed up to 150MHz and enhanced peripheral such as, 12-bit ADC with conversion speed up to 80nSec

allows implementation of high band width, high frequency power supplies without losing performance. This thesis is about sensorless control of Permanent Magnet Synchronous Motor. The word sensorless might be misleading for someone who is not in the field of motor control. Sensorless control refers actually to a special type of motor control where the mechanical rotor position sensor has been replaced by a mathematical algorithm. This algorithm needs however to get information from other sensors such as current sensors and in some cases also voltage sensors. But these sensors are cheaper and reliable. They are sometimes required for control anyway. The mathematical algorithm that is used to replace the position sensor can be quit complex and needs a certain amount of computation power. With today's trend of faster and cheaper microprocessors the computational power is not an issue any more.

1.2 Problem Statement

PMSM basically uses mechanical position sensor for accessing actual rotor position and speed information for speed and torque control of the motor. This position sensor has its own disadvantage like increasing additional cost, and reducing system reliability. This thesis avoids the mechanical position sensor and substitutes it by mathematically computed state (speed, position) Estimator/Observer for controlling the speed. Specifically the thesis tries to answer the following questions:

- How the system state space equation of the PMSM is modeled for Observer based sensorless motor control?
- How rotor position and speed estimator is designed?
- How speed control of the system is designed?
- How the overall system algorithm is programmed in TMS320F2812 DSP?

1.3 Motivation and Objective

Digital signal processing technology is enabling cost effective and energy efficient control system design. In addition to the advantage of digital control system, running permanent magnet synchronous motor without position mechanical sensor has its own advantage over permanent magnet synchronous motor with position sensor:

- It minimizes the drive cost.
- It decreases mechanical size of the machine.

- It increases the reliability of the system.
- It improves the noise immunity efficiency of the system.
- It avoids the complex process of embedding the position sensor into the stator.

The above facts invite working in this area. Hence, this thesis has a general objective of developing an embedded controller for a permanent magnet synchronous motor using a Texas Instrument DSP of TMS320F2812 with specific objectives of:

- Studying model of the permanent magnet synchronous motor.
- Design an observer for position and speed of the motor.
- Design speed controller of the motor.
- Develop simulation and experimental setup for the system.

1.4 Methodology

For the accomplishment of observer based speed controlling of PMSM using DSP, the following methodologies have been followed:

- Gathering & survey related works from various literature and publication.
- Study and design the appropriate modelling of the system.
- Design an appropriate Observer & Controller for the system.
- Analyze the design with Matlab/Simulink simulation.
- Build an observer and controller algorithm to program the system in to DSP.
- Code the algorithm with C programming language in to the DSP using code composer studio.
- Develop necessary Hardware components for experimental investigation.

1.5 Scope and Limitation

The scope of this thesis is designing and developing speed control of permanent magnet synchronous motor on the TMS320F2812 DSP using an observer for removing the mechanical position sensor which is used for measuring instantaneous mechanical position of the rotor.

Our Laboratory Motor has no parameter specification and it needs some form of motor parameter identification. Simple parameter identification is done for Parameters like stator

Resistance, Inductance, No of pole pair but the Moment of inertia and Friction coefficient are difficult to investigate. And for experimental investigation only open loop system was tested on the DSP.

1.6 Literature Review

One of the most active areas of control development during recent years, involving PMSM motor types, have been the evolution of new techniques for eliminating the position and speed sensor. Elimination of the shaft-mounted sensor is required in many applications since it is often one of the most expensive and fragile components in the entire drive system. The approaches to sensorless drives vary depending on the rotor flux distribution. A motor with a trapezoidal rotor flux distribution (such as brushless dc motor, BLDM) provides an attractive candidate. Two out of three stator windings are excited at the same time and the unexcited winding can be used as a sensor. The control scheme as well as the position detection is relatively simple. The rotor speed and position can be determined by the electromagnetic field induced in the unexcited winding [1], [2]. This is usually done either by a zero crossing approach of the back-EMF or by a phase-locked loop technique to lock on to the back-EMF waveform in the unexcited winding. It is enough to detect the rotor position every 60° to obtain a proper switching sequence [22]. On the contrary, PMSM motor, having a sinusoidal flux distribution, excites all three windings at the same time. The information on the rotor position is required continuously for speed control of PMSM. Hence, both the control algorithm and the speed/position estimation become more complicated. PMSM motor may be salient or non-salient rotor type. Depending on this, their sensorless control strategy is different. Various schemes for sensorless control of permanent magnet synchronous motor ,such as State Observers [3], Model Reference Adaptive Systems (MRAS) [4], Sliding Mode Observer, and other nonlinear observer systems like estimating the position from the estimate of Sin(θ) and Cos(θ) [4],[5], Flux injecting method for non-salient pole PMSM motor, have been reported in the literatures. Model reference adaptive systems are simple but greatly depend on the accuracy of reference model. Methods based on linear state observers are robust but the problem is to linearise the nonlinear state equation of the motor. Sliding Mode observers has been used as back up and fault detection for surface mounted PMSM. The target motor for this thesis is a non-salient type, hence, this paper deals with the Extended Kalman Filter (EKF) to estimate the mechanical rotor speed and position for supporting the nonlinear state equation of the motor.

1.7 Thesis Outline

This thesis includes five chapters including this introduction. Chapter-2 describes and states vector control of the permanent magnet synchronous motor. It deals with vector control theory, mathematical modeling of the motor, observer and controller design. Chapter-3 deals with system experimental organization including hardware and software. In this chapter system software flowcharts and different components for experimental investigation are discussed. Chapter-4 states system simulation model, simulation results and their discussion. System experimental configuration and results also presented. Chapter-5 presents concluding remarks of the study with some recommendations. Appendices are added to the report as part of the thesis. Simple motor parameter identification technique and DSP based C programming of the overall system are presented.

Chapter Two

Control Theory of PMSM motor

2.1 Introduction

There are different techniques of speed control of permanent magnet synchronous motors. Some of the techniques are shown in the block diagram below. The speed control may be scalar based or vector based. The vector control based also can be direct torque control or field oriented control. This thesis follows the shaded path of the Figure.

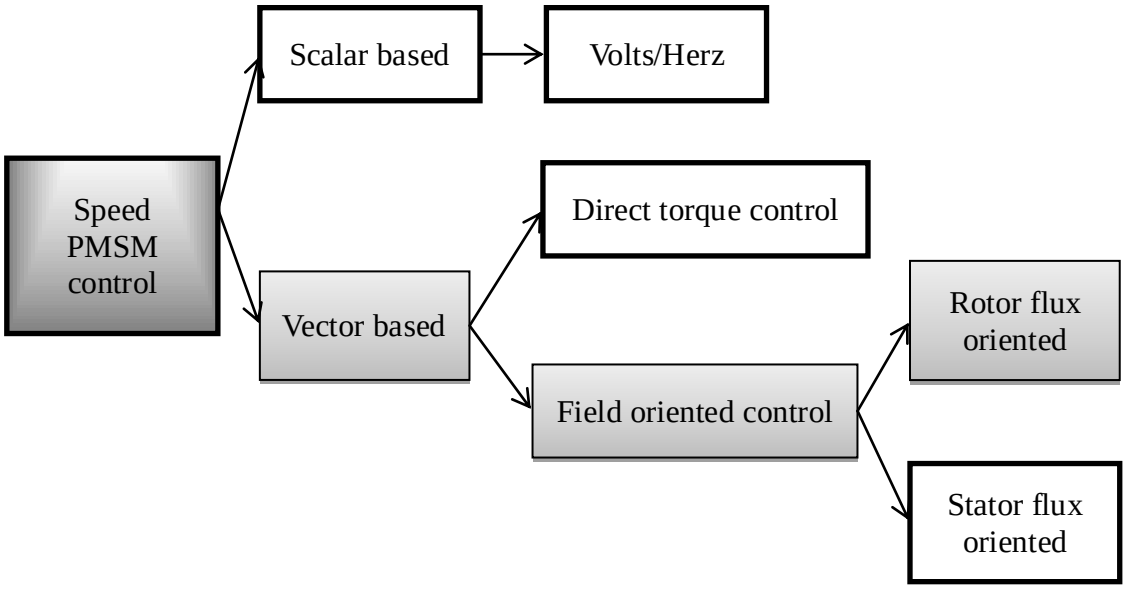


Figure 2.1: Different techniques of PMSM speed control

For vector control, a three phase voltage is generated and used as a vector to control three phase stator currents. By transforming physical phase currents into a rotational vector using Clarke and Park transformations, the flux and torque currents become time invariant, allowing direct control of the motor flux and torque, as for DC motors for achieving fast dynamic response and high performance. Vector controlled machines need two constants as input references: the torque component (aligned with the q-coordinate) and the flux component (aligned with d-coordinate). As it is simply based on projections, the control structure handles instantaneous electrical quantities. This makes the control accurate in all working operation (steady state and transient).

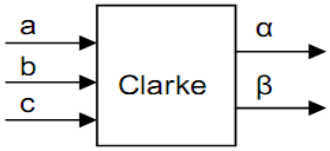
2.2 Vector Control Theory

Generally, vector control establishes three reference frames as shown in Figure 2.2.

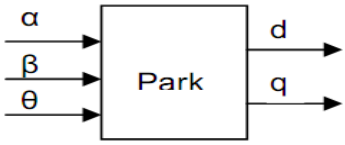
1. **Stator reference frame** in which the (a, b, c) are coplanar at 120° to each other.
2. **An orthogonal reference frame** (α, β) in the same plane as stator reference frame and the angle between the two axis is 90° instead of 120°, the α-axis is aligned with a-axis.
3. **A rotor reference frame**, known as the (d, q) frame, in which the d-axis is along the overall flux vector of the rotor and the q-axis is at 90° to the d-axis.

If (a, b, c) are the three axes of the three phase sinusoidal system, it needs transformation into two time invariant coordinate system. This transformation can be split into two steps[22].

- (a, b, c) → (α, β): The **Clarke transformation** which outputs a two coordinates time variant system.



- (a,b,c)→ (d,q,0): The **Park transformation** which outputs a two time invariant Coordinates. For Vector Motor Control Theory, the normalized Park transformation [(α, β) → (d, q)] is considered.



The Clarke transformation: [The (a, b, c) → (α, β) projection]

The space vector can be projected in another reference frame with only two orthogonal axis called (α, β) which is time variant. Assuming, axis-a and axis-α are aligned in the same direction.

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} i_a \\ i_b \\ i_c \end{bmatrix}$$

The Park transformation: [The (α, β) → (d, q) projection]

Assuming d-axis is θ degree shift to α-axis:

$$\begin{pmatrix} i_d \\ i_q \end{pmatrix} = \begin{pmatrix} \cos(\theta) & \sin(\theta) \\ -\sin(\theta) & cos(\theta) \end{pmatrix} \begin{pmatrix} i_\alpha \\ i_\alpha \end{pmatrix}$$

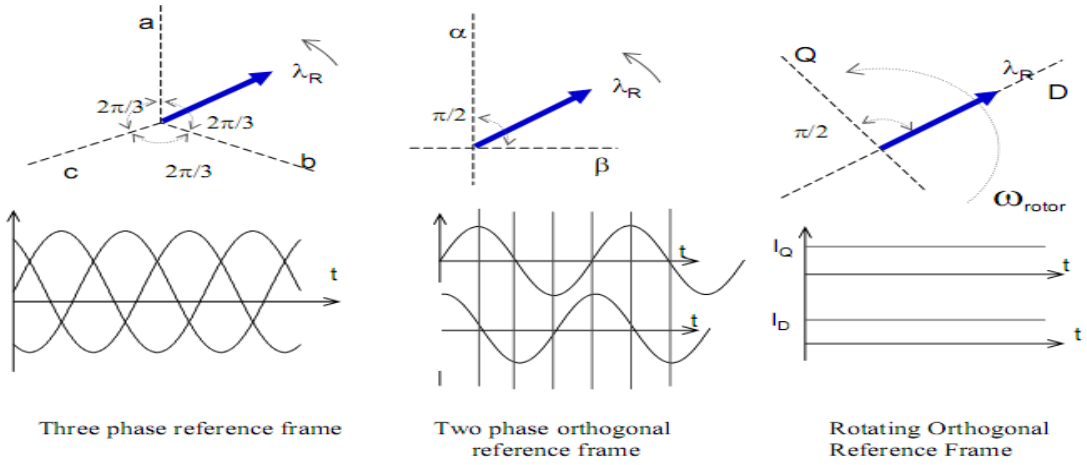


Figure 2.2: The three coordinate system and their wave forms

2.3 Mathematical Modeling of PMSM motor

The PMSM have an electromagnetic stator and a permanent magnet rotor. The magnetic flux of the rotor is caused by permanent magnets, rather than an alternating current, which is the case in DC motors. Since current isn't applied in the rotor, there is no need for mechanical contact between stator and rotor, hence described as brushless.

There are several design methods for building the rotor in a PMSM. The most notable difference in design is the number of magnets or pole pairs mounted. The number of pole pairs will affect the synchronous speed and the amount of torque the motor yields for a certain current. Another design difference is either the rotor magnets are mounted on the iron core (surface mounted PMSM) or the magnets are mounted inside the iron core (internal mounted PMSM). The type of mounting will affect the inductance created by the rotor. The inductance of stator winding is generally a function of rotor position. When a stator winding is energized, applying a DC voltage for a certain time, a magnetic field with a fixed direction will be established. The current response of the winding is different due to inductance variation. But if we assume that the rotor

has the surface mounted design, which is the case of most brushless motors, there is no saliency such that the stator self-inductances are independent of the rotor position.

The system model analysis is based on the following assumption for design simplification.

- The motor is not saturated or the effect of saturation in the magnetic paths of the machine is negligible.
- Stator inductance is assumed to be constant.
- Stator resistances and inductances of all the windings are equal.

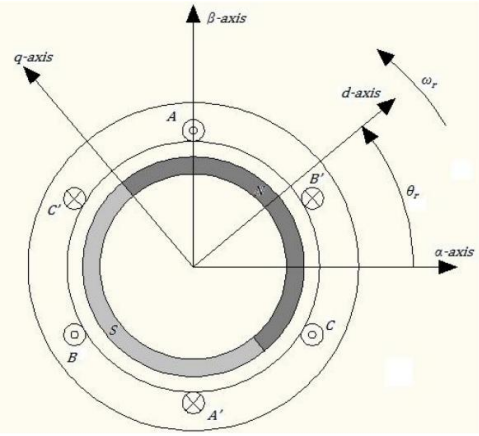


Figure 2.3: A typical three phase, two pole permanent magnet synchronous motor [9]

where ω_r- rotor flux speed, θ_r position of rotor flux from a-axis.

Model equations of the motor in stator reference frame (a, b, c)

Let, R – resistance, P – No of pole pairs; L- stator inductance; E_a, E_b, E_c– are back EMF in (a, b, c) axis respectively; λ_m - rotor flux linkage; θ_r - rotor position; ω_r-rotor speed; V_a, V_b , V_c & i_a, i_b, i_c are respectively stator voltages and currents in (a, b, c) axis; T_e-electrical torque; T_m-mechanical torque; T_L-load torque, J-moment of inertia, F-friction coefficient.

In (a, b, c) stationary reference frame, the model of a three phase PMSM can be described as:

$$V_{abc} = \Lambda\, i_{abc} + \frac{\partial \lambda_{abc}}{\partial t} \qquad (2.3)$$

Where, λ_{abc} is stator flux linkage in (a, b, c) reference frame respectively and

$$\Lambda = \begin{bmatrix} R & 0 & 0 \\ 0 & R & 0 \\ 0 & 0 & R \end{bmatrix} \quad (2.4)$$

The flux linkage equation can be expressed as:

$$\lambda_{abc} = \mathrm{L}\begin{pmatrix} i_a \\ i_b \\ i_c \end{pmatrix} + \lambda_m \begin{pmatrix} \sin(\theta_r) \\ \sin(\theta_r + \frac{2\pi}{3}) \\ \sin(\theta_r - \frac{2\pi}{3}) \end{pmatrix} \quad (2.5)$$

It can be also written as:

$$V_a = \mathrm{R}i_a + \mathrm{L}\frac{\mathrm{d}i_a}{\mathrm{dt}} + E_a;\ E_a = \omega_r \lambda_m \mathrm{Sin}(\theta_r) \quad (2.6)$$

$$V_b = \mathrm{R}i_b + \mathrm{L}\frac{\mathrm{d}b}{\mathrm{dt}} + E_b;\ E_b = \omega_r \lambda_m \mathrm{Sin}(\theta_r + \frac{2\pi}{3}) \quad (2.7)$$

$$V_c = \mathrm{R}i_c + \mathrm{L}\frac{\mathrm{d}c}{\mathrm{dt}} + E_c;\ E_c = \omega_r \lambda_m \mathrm{Sin}(\theta_r - \frac{2\pi}{3}) \quad (2.8)$$

The Electromagnetic torque:

$$T_e = (E_a i_a + E_a i_a + E_a i_a)/\omega_r \quad (2.9)$$

It can be written as:

$$T_e = \frac{\mathrm{P}}{2}\lambda_m ((i_a - \frac{1}{2} i_b - \frac{1}{2} i_c)\mathrm{Sin}(\theta_r) - \frac{\sqrt{3}}{2}(i_b - i_c)\mathrm{Cos}(\theta_r)) \quad (2.10)$$

And the Electromechanical torque

$$T_m = \mathrm{J}\frac{d\omega_r}{dt} + \mathrm{F}\omega_r + T_L \quad (2.11)$$

Model equations of the motor in the stationary orthogonal (α, β) reference frame [18],[8]

$$V_\alpha = Ri_\alpha + \mathrm{L}\frac{di_\alpha}{dt} - \lambda_m \omega_r \mathrm{Sin}(\theta_r) \quad (2.12)$$

$$V_\alpha = Ri_\beta + \mathrm{L}\frac{di_\beta}{dt} + \lambda_m \omega_r \mathrm{Cos}(\theta_r) \quad (2.13)$$

Electromagnetic torque

$$T_e = \lambda_m (-i_\alpha \text{ Sin}(\theta_r) + i_\beta \text{ Cos}(\theta_r)) \quad (2.14)$$

Electromechanical torque

$$T_m = J\frac{d\omega_r}{dt} + F\omega_r + T_L \quad (2.15)$$

As we can see, the above equations are non-linear and strongly coupled with each other. These factors make it very difficult to perform direct control of flux and torque by varying the currents in the stator coils. Therefore; transforming to another coordinate system is needed.

Model equations of the motor in the Rotary Orthogonal reference (d, q) frame [18]

Finally, the model of the motor in rotary time invariant reference frame will be:

$$V_d = Ri_d + \frac{\partial\lambda_d}{\partial t} - \omega_r\lambda_q$$

$$V_q = Ri_q + \frac{\partial\lambda_q}{\partial t} + \omega_r\lambda_d$$

$$T_e = \frac{3P}{2}(\lambda_d i_q - \lambda_q i_d)$$

$$T_m = J\frac{d\omega_r}{dt} + F\omega_r + T_L$$

$$\Longrightarrow \begin{cases} \frac{di_d}{dt} = -\frac{R}{L}i_d + \omega_r i_q + \frac{V_d}{L} & (2.16) \\ \frac{di_q}{dt} = -\frac{R}{L}iq - \omega_r i_d - \omega_r\frac{\lambda_m}{L} + \frac{V_q}{L} & (2.17) \\ \frac{d\omega_r}{dt} = \frac{3}{2J}p^2\lambda_m i_q - \frac{F}{J}\omega_r - p\frac{T_L}{J} & (2.18) \\ \frac{d\theta_r}{dt} = \omega_r & (2.19) \end{cases}$$

Where, $\lambda_d = Li_d + \lambda_m$, $\lambda_q = Li_q$

V_d, V_q, i_d & i_q are d-axis and q-axis voltage and current component.

In order to produce maximum torque, optimal operation is achieved by vector control, which ensures that the stator- current space vector contains only a quadrature component by pushing d-axis current towards zero [13]. As a result the torque equation becomes:

$$T_e = \frac{3P}{2}\lambda_m i_q \quad (2.20)$$

Thus the torque can be controlled directly by the current i_q only. For development of Observer and Controller of the motor, the state space representation of the model is appropriate.

The Model State space representation in the d-q reference frame

Taking i_d, i_q, ω_r, θ_r as state vector, V_d, V_q and T_L as input vector and i_d, i_q as output vector, The input output state space representation of the system becomes:

$$\frac{d}{dt}\begin{pmatrix} i_d \\ i_q \\ \omega_r \\ \theta_r \end{pmatrix} = \begin{pmatrix} -\frac{R}{L} & 0 & i_q & 0 \\ 0 & -\frac{R}{L} & -i_d - \frac{\lambda_m}{L} & 0 \\ 0 & 1.5p^2\frac{\lambda_m}{J} & -\frac{F}{J} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix}\begin{pmatrix} i_d \\ i_q \\ \omega_r \\ \theta_r \end{pmatrix} + \begin{pmatrix} \frac{1}{L} & 0 & 0 \\ 0 & \frac{1}{L} & 0 \\ 0 & 0 & -\frac{P}{J} \\ 0 & 0 & 0 \end{pmatrix}\begin{pmatrix} V_d \\ V_q \\ T_L \end{pmatrix} \quad (2.21)$$

$$\begin{pmatrix} i_d \\ i_q \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix}\begin{pmatrix} i_d \\ i_q \\ \omega_r \\ \theta_r \end{pmatrix} \quad (2.22)$$

Discretization of the system model

The corresponding discrete time state and output model is done using the following discretization method [20]. Let the discrete for of the system be:

X (k+1) = A_dX(k) + B_dU(k)

Y (k) = C_dX(k)

The conversion is done by the following approximation;

$$\begin{cases} A_d = \exp(At) = I + AT_s \\ B_d = \int \exp(A\varepsilon)Bd\varepsilon = BT_s & \text{; where } T_s \text{ is sampling period} \\ C_d = C & \text{; A, B, C are state, input \& output matrix of the system.} \end{cases}$$

The discrete form of the state and output equation become:

$$\begin{pmatrix} \boldsymbol{i_d}(\mathrm{k}+1) \\ \boldsymbol{i_q}(\mathrm{k}+1) \\ \omega_{\boldsymbol{r}}(\mathrm{k}+1) \\ \boldsymbol{\theta_r}(\mathrm{k}+1) \end{pmatrix} = \begin{pmatrix} 1-T_s\frac{\mathrm{R}}{\mathrm{L}} & 0 & \mathrm{iq}T_s & 0 \\ 0 & 1-T_s\frac{\mathrm{R}}{\mathrm{L}} & -T_s(i_d+\frac{\lambda_m}{\mathrm{L}}) & 0 \\ 0 & 1.5T_s\mathrm{P}^2\frac{\lambda_m}{\mathrm{J}} & 1-T_s\frac{\mathrm{F}}{\mathrm{J}} & 0 \\ 0 & 0 & T_s & 0 \end{pmatrix} \begin{pmatrix} i_d(k) \\ i_q(k) \\ \omega_r(\mathrm{k}) \\ \theta_\mathrm{r}(\mathrm{k}) \end{pmatrix} + \begin{pmatrix} \frac{T_s}{L} & 0 & 0 \\ 0 & \frac{T_s}{L} & 0 \\ 0 & 0 & -T_s\frac{P}{J} \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} V_d \\ V_q \\ T_L \end{pmatrix} \quad (2.23)$$

$$\begin{pmatrix} \boldsymbol{i_d}(\mathrm{k}) \\ \boldsymbol{i_q}(\mathrm{k}) \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \begin{pmatrix} i_d(\mathrm{k}) \\ i_q(\mathrm{k}) \\ \omega_r(\mathrm{k}) \\ \theta_\mathrm{r}(\mathrm{k}) \end{pmatrix} \quad (2.24)$$

For the specific target motor used in simulation with parameters:

Stator resistance: R= 1.456Ω *Stator inductance: L=0.008H*

Rotor flux linkage: λ_m= 0.175 V.s *Rotor inertia: J=0.06JKgm²*

Friction coefficient: F= 0.001Nms *Number of pole pairs: P=3*

Flux distribution type: Sinusoidal *Voltage constant: K_v= 95.2445V/krpm*

Torque constant: K_T=7875Nm/A

If ***f(k +1)*** = [$\boldsymbol{i_d}$(k+1), $\boldsymbol{i_q}$(k+1), $\omega_{\boldsymbol{r}}$(k+1), $\boldsymbol{\theta_r}$(k+1)]^T and ***h(k)*** = [$\boldsymbol{i_d}$(k), $\boldsymbol{i_q}$(k)]^T, The corresponding state space model of the system as state and output equation with sampling period of 10μs can be written as:

$$\boldsymbol{f(k+)} = \begin{pmatrix} 0.9982 & 0 & 1\mathrm{e}-5i_d(\mathrm{k}) & 0 \\ 0 & 0.9982 & -1\mathrm{e}-5[i_d(\mathrm{k})+21.875] & 0 \\ 0 & 3.938\mathrm{e}-4 & 1 & 0 \\ 0 & 0 & 1\mathrm{e}-5 & 1 \end{pmatrix} \begin{pmatrix} i_d(k) \\ i_q(k) \\ \omega_r(k) \\ \theta_\mathrm{r}(k) \end{pmatrix} + \begin{pmatrix} 1.25\mathrm{e}-3 & 0 & 0 \\ 0 & 1.25\mathrm{e}-3 & 0 \\ 0 & 0 & -5\mathrm{e}-4 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} V_d \\ V_q \\ T_L \end{pmatrix} \quad (2.25)$$

$$\boldsymbol{h(k)} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.26)$$

2.4 Observer and Controller Design of the System

The general system block diagram in vector control method is as shown in Figure 2.4: The system basically includes Observer and Controllers. The observer estimates the motor position and speed using stator currents and voltages from mathematical computation based on state space model of the motor. The observer used in the work is Extended Kalman Filter as described before. The controller attains the required value of set point by using measured stator currents and estimated speed as feedback value. The controller is Proportional and Integral (PI) type which is mostly used in industrial application. The system also includes different co-ordinate transformation, such as Clarke, Park and IPark for the purpose of vector control. Space vector pulse width modulation (SVPWM) is also part of the system which calculates the operation time of the inverter gates from the controlled d-q space vector voltage value for proper application voltages on the stator windings.

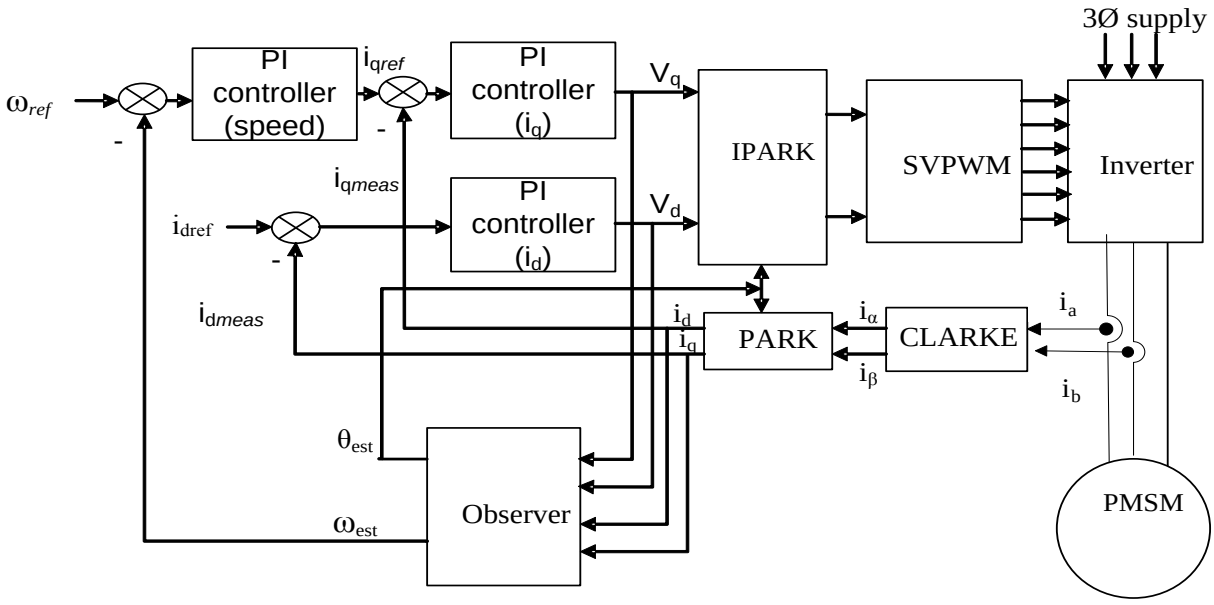


Figure 2.4: General system model for vector control

2.4.1 Observer Design

In speed control of PMSM, instantaneous value of rotor position and speed is needed as feedback for PI controller. The state observer is used to estimate these parameters using system model containing speed and position as state. The observablity of permanent Magnet Synchronous motor is studied in other papers which deal with this area. This paper considers PMSM as

observable by using these papers as references [9]. As explained previously the observer is an Extended Kalman Filter. It is a Kalman Filter which extends to support nonlinear system models. Kalman filter is a set of mathematical equations that provides an efficient computational means to estimate the state of a process in a way that minimizes the mean of the squared error. It is based on state space technique and recursive algorithms. It estimates the state of a dynamic system. This dynamic system can be disturbed by some noise, mostly assumed as white noise, to improve the estimated state. The Kalman filter uses measurements that are related to the state but disturbed as well.

The mathematical model of the PMSM is nonlinear. Hence, we use the Extended Kalman Filter for estimation of the states. Let the discrete time dynamic system with white additive noise be:

$f(k+1)=f(X(k),U(k))+v(k)$

$h(k)=C(k)X(k)+w(k)$; where *v* & *w* are system and measurement noises with covariance matrix of Q(k) and R(k) respectively.

For linearization process in the model, the partial derivative of the discrete state model is introduced.

$$F(k) = \frac{\partial f(x(k),u(k))}{\partial x(k)} \Big| x(k)=\hat{x}(k|k)$$

$$F(k) = \begin{pmatrix} 1-\frac{R}{L}T_s & \omega_r T_s & T_s i_q & 0 \\ -\omega_r T_s & 1-\frac{R}{L}T_s & T_s\left(i_d-\frac{\lambda_m}{L}\right) & 0 \\ 0 & -\frac{3}{2}P^2\frac{\lambda_m}{J}T_s & 1-\frac{F}{J}T_s & 0 \\ 0 & 0 & Ts & 1 \end{pmatrix} \quad (2.27)$$

$$H(k) = \frac{\partial h(x(k))}{\partial x(k)} \Big| x(k)=\hat{x}(k|k)$$

$$H(k) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{pmatrix} \quad (2.28)$$

Extended Kalman Filter has three basic stages [21] to estimate parameters of system model:

1. Prediction of state and error covariance matrix

   Assuming $X(0|0)$, $P(0|0)$ are the initial conditions of the state and its error covariance:

   $$\hat{X}(k+1|k) = f(x(k|k), u(k), k)$$

   $$P(k+1|k) = F(k|k)\,P(k|k)\,F^{T}(k|k) + Q^{T}(k)$$

2. Controller gain calculation

   $$\mathbf{K}(k+1) = P(k+1|k)\,H^{T}(k+1)\,[H(k+1)\,P(k+1|k)\,H^{T}(k+1) + R(k)]^{-1}$$

3. Updating the state and its error covariance

   $$\hat{X}(k+1|k+1) = \hat{X}(k+1|k) + \mathbf{K}(k+1)\,[Z(k+1) - h(x(k+1|k))]$$

   $$P(k+1|k+1) = P(k+1|k) - \mathbf{K}(k+1)\,H(k+1)\,P(k+1|K)$$

where $\mathbf{K}(k+1)$ is estimation gain and $Z(k)$-measured value of the output, $Q(k)$ and $R(k)$ are randomly taken based on the system and measurement noise since we don't have statistical information of the noise.

2.4.2 Controller Design

To control the system speed and torque, Proportional and integral (PI) controller is used. PI is a generic control loop feedback mechanism controller widely used in industrial control system. A PI controller calculates error values as the difference between a measured process variable and a desired set point. The controller attempts to minimize the error by adjusting the process control inputs.

Proportional Term: a high proportional gain results in a large change in the output for a given change in the error.

Integral Term: the contribution from the integral term is proportional to both the magnitude of the error and the duration of the error. The integral term in a PI controller is the sum of the instantaneous error over time and gives the accumulated offset that should have been corrected previously. The accumulated error then multiplied by the integral gain and added to the controller

output. The integral term accelerates the movement of the process towards the set points and eliminates the residual steady state error that occurs with a pure proportional controller.

Calculation of controller gains

For design of PI controllers, it is necessary to know the closed loop the transfer functions. The PI loops are used to control three interactive variables independently. The rotor speed, rotor flux, and rotor torque are each controlled by a separate PI module. K(1+ $\frac{1}{TiS}$) is the PI controller; where K-proportional gain, $\frac{K}{T_i}$ - is integral gain T_i- integral action time. The discrete form of the PI controller becomes:

C(z) = $K_p + K_i * \frac{Z}{Z-1}$; K_p = k, $K_i = \frac{K}{T_i} * sampling\ time$

Current controller gain

The d-axis current closed loop transfer function from equation (2.14) will be:

$$\frac{i_d}{V_d} = \frac{1}{Ls+R} + \omega_r L i_q \qquad (2.29)$$

The closed loop transfer function with the PI controller is as shown in the Figure 2.5:

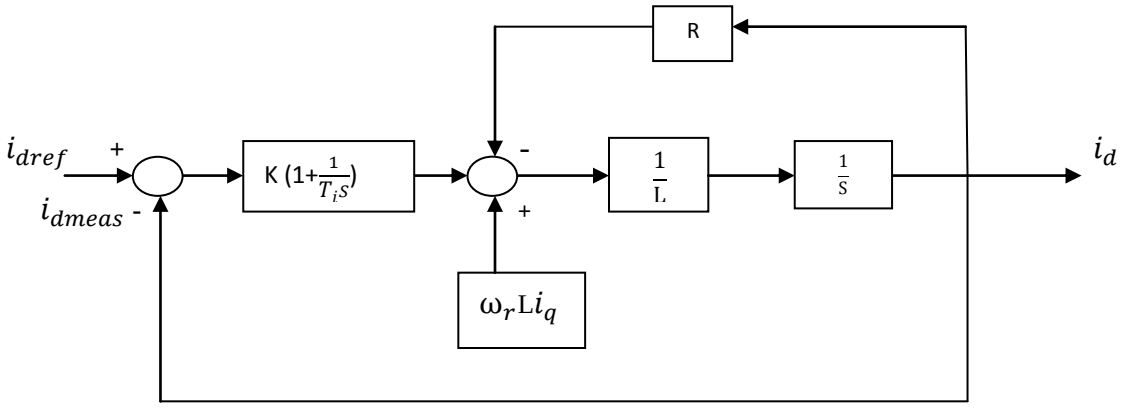


Figure 2.5: d –axis current closed loop transfer function with PI controller block diagram

As we can see from the above block diagram, $\omega_r L i_q$ block makes the closed loop transfer function complicated. For simplicity of PI controller parameter calculation, the $\omega_r L i_q$ part of the equation is ignored. Its effect can be compensated by tuning the parameters after calculation.

The closed loop current control equation:

$$\frac{i_{dmeas}}{i_{dref}} = \frac{K(T_iS+1)}{T_iS(LS+R)+K(T_iS+1)} \qquad (2.30)$$

To improve the response characteristics of the system we can use zero placement approach first by cancelling the zero of the closed loop equation.

Let T_i = L/R, then $\frac{i_{dmeas}}{i_{dref}} = \frac{K}{(T_i RS+K)} = \frac{1}{\frac{T_i R}{K}S+1}$

For the design of PI controller using pole placement method the transfer function of corresponding order is compared with denominator of closed loop transfer function $(S+\omega_0)^n$, where n-order of the system, ω_0 - minimal natural frequency obtained from Dodd's formula[10], which respects prescribed settling time Ts of corresponding loop.

T_s = 1.5(1+n).$\frac{1}{\omega_0}$

Comparing the ideal first order with the closed loop

K = 3 $\frac{L}{Ts}$; T_i = $\frac{L}{R}$; Hence substituting values, K = 34, T_i = 0.78

Similar calculation is done for q-axis current with closed loop transfer function:

$\frac{i_q}{V_q} = \frac{1}{Ls+R} + \omega_r L i_d$; and the gains become K = 21, T_i = 0.74

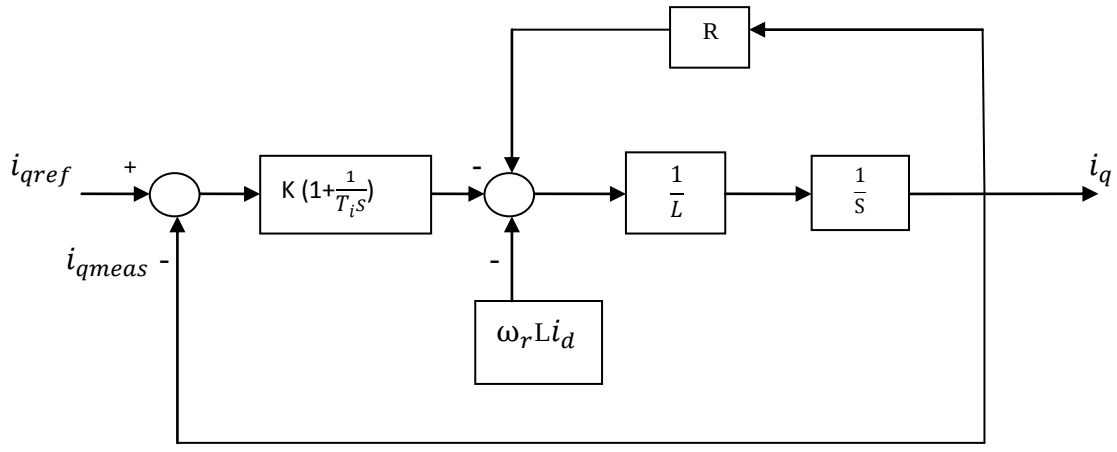


Figure 2.6: q–axis current closed loop transfer function with PI controller block diagram

Speed controller gain

For design of speed PI controller block diagram shown in the Figure 2.7 is used. Inner current control loop is replaced by its first order transfer function $\frac{1}{T_pS+1}$ where $T_p = \frac{T_{sq}}{3}$ and $K_m = \frac{3}{2} p^2 \lambda_m$.

Using Equation (2.18) the transfer function of the speed closed loop considering small friction coefficient can be simplified as:

$$\frac{\omega_{meas}}{\omega_{ref}} = \frac{\frac{KK_m}{JTp} S + \frac{KK_m}{JTpTi}}{S^3 + \frac{1}{T_p} S^2 + \frac{KK_m}{JT_p} S + \frac{KK_m}{JT_p T_i}} \quad (2.31)$$

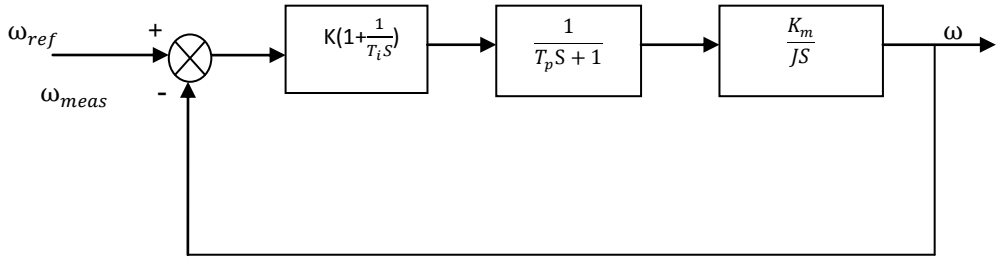


Figure 2.7: Speed closed loop transfer function with PI controller block diagram

For designing speed PI-controller by pole placement method, denominator of closed loop speed transfer function is compared with ideal transfer function of the same order which for order n = 3 is as follows:

$$s^3 + 3\omega_0 s^2 + 3\omega_0^2 s + \omega^3_0 \quad (2.32)$$

For calculation of proportional gain K and integral time constant Ti of speed PI controller, the above ideal transfer function is designed exploiting Dodd's formula which respects settling time of speed control loop. Then the ideal transfer function respecting settling time of speed control loop is as follows:

$$S^3 + \frac{18}{T_s\omega} S^2 + \frac{108}{T_s\omega^2} S + \frac{216}{T_s\omega^3} \quad (2.33)$$

By equating Equation (2.32) and Equation (2.33):

$K = \frac{108J\,T_p}{K_m T_s \omega^2}$ and $T_i = \frac{K_m T_s \omega^3}{216J\,T_p}$

For the proper function of speed control loop with chosen settling time Tsω, settling time of speed loop should be adjusted to satisfy following condition:

$$\frac{1}{T_p} = \frac{18}{T_s\omega} \rightarrow T_{sq} = \frac{T_{s\omega}}{6}$$

Hence, K =19.24; T_i= 0.07

For proper simulation of the system, both the Proportional and Integral gain values are tuned to the appropriate value.

2.5 Space Vector Pulse Width Modulation

Digital control techniques of AC motors, such as the space vector pulse width modulation (SVPWM) have been developed with wide range industrial applications. The SVPWM was brought forward in the 1980's, specifically for the AC motors. It controls the motor based on the switching of the space voltage vector, by which an approximate circular rotary magnetic field is obtained. In other words the inverter is controlled to output a sinusoidal voltage wave form with appropriate magnitude and frequency. This forms the basis for magnetic flux linkage tracking pulse width modulation. The basis of SVPWM is different from that of sine pulse width modulation (SPWM). SPWM aims to achieve symmetrical 3-phase sine voltage wave forms of adjustable voltage and frequency, while SVPWM takes the inverter and motor as a whole, using the eight fundamental voltage vectors to realize variable frequency of voltage and speed adjustment.

Techniques of SVPWM [12]

SVPWM is a technique of determining the switching sequence of the upper three transistors of three phase voltage source inverter. There are eight possible output voltage states. Two of the output states are null vectors ($\bar{V}0$ and $\bar{V}7$) whereas the other six output vectors are spatially spaced 60° apart as shown in the Figure 2.8.

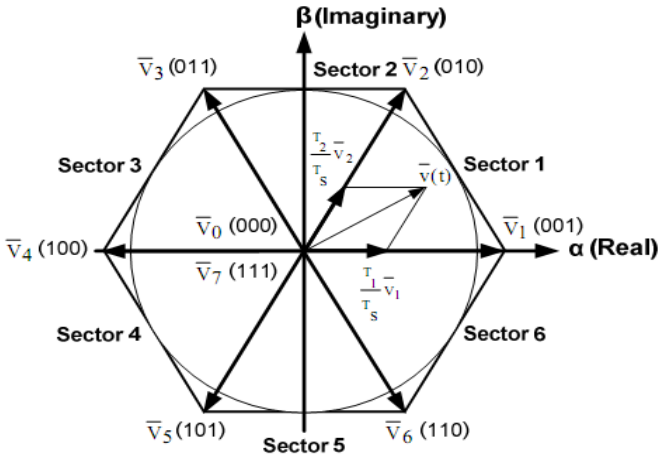


Figure 2.8: Voltage space vectors representing eight possible switching states

The voltage space vector is synthesized by time weighted averaging of the two adjacent basic non-zero voltage vectors that form the sector in which the reference voltage space vector to be synthesized lies. Thus if the reference voltage space vector lies in the first sector, voltage space vector $\bar{V}1$ is active for duration T1 and voltage space vector $\bar{V}2$ is active for duration T2 within the switching period Ts as shown in Figure 2.9. And the reference vector $\bar{v}(t)$ can be described as equation :

$$[\bar{v}(t)] = \frac{T_1}{T_s}\bar{V}1 + \frac{T_2}{T_s}\bar{V}2 + \frac{T_0}{T_s}\bar{V}0 + \frac{T7}{T_s}\bar{V}7$$

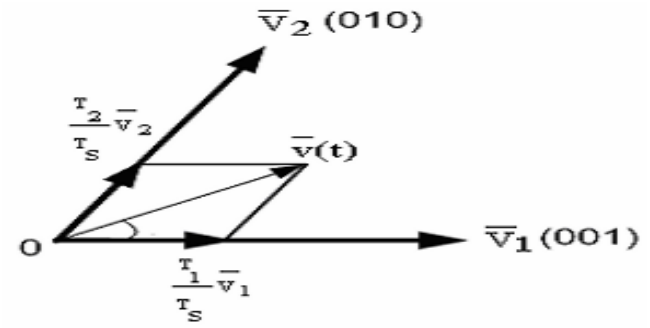


Figure 2.9: Synthesized reference voltage space vector in sector one

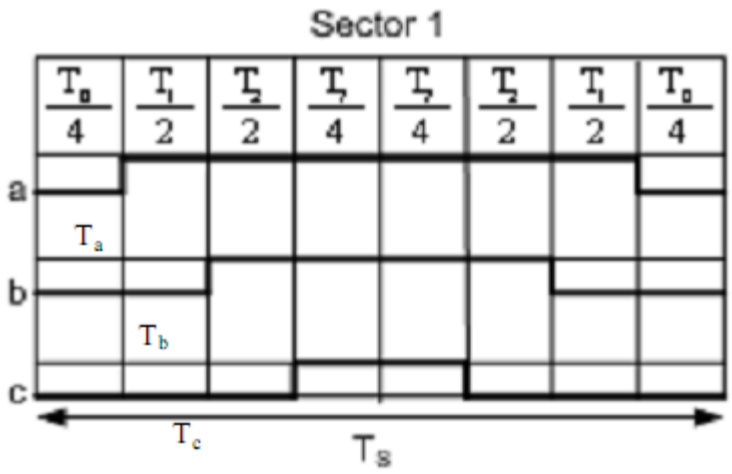


Figure 2.10: Switching pattern of sector-1

SVPWM aims to generate a voltage vector that is close to the reference circle through the various switching modes of inverter. Figure 2.11 is a typical diagram of a three phase voltage source inverter model. For the on-off state of the three phase inverter circuit, every phase can be considered as a switch S. here $S_A(t)$, $S_B(t)$, $S_C(t)$ are used as the switching functions for the three phases respectively.

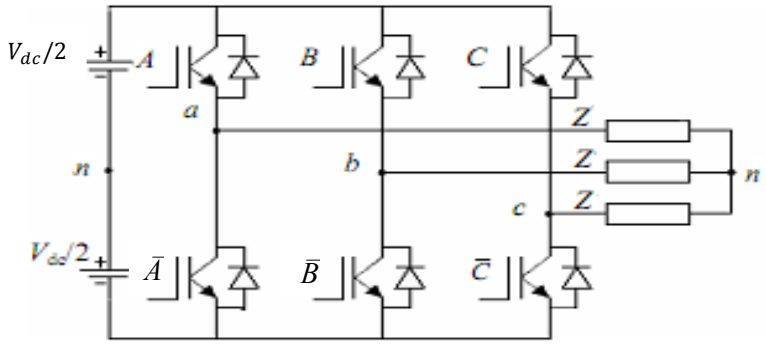


Figure 2.11: Typical diagram of a three-phase inverter

The space vector voltage of inverter can be expressed as $V(S_A, S_B, S_C) = V_{dc}\,(S_A+\alpha S_B+\alpha^2 S_C)/3$

where V_{dc} is the DC bus voltage of inverter and $\alpha=e^{j120}$. If we express the on state of the upper arm with ‘’1’’ and the off state with ‘’0’’, the on-off state of three phase have eight combinations, correspondingly forming eight voltage space vectors, as shown in Table 2.1. T refers to the operation times of two adjacent non-zero voltage space vectors in the same zone.

Both $\bar{V}_0$ (000) and $\bar{V}_7$ (111) are called the zero voltage space vector, and the other six vectors are called the effective vector with a magnitude of 2Vdc/3. For example, when the output voltage vector V is within the first sector, it is composed of $\bar{V}_4$, $\bar{V}_6$, $\bar{V}_7$ and $\bar{V}_0$ and can be obtained by

$$V_{out} = \frac{T_4 V_4}{T} + \frac{T_6 V_6}{T}$$

The eight on-off states of the inverter are listed in the Table 2.1 below

Table 2.1: eight on-off state of the inverter

Inverter State	$S_A S_B S_C$	$S_{\bar{A}} S_{\bar{B}} S_{\bar{C}}$	V_A/V_{dc}	V_B/V_{dc}	V_C/V_{dc}
0	000	111	0	0	0
1	001	110	−1/3	−1/3	2/3
2	010	101	−1/3	2/3	−1/3
3	011	100	−2/3	1/3	1/3
4	100	011	2/3	−1/3	−1/3
5	101	010	1/3	−2/3	1/3
6	110	001	1/3	1/3	−2/3
7	111	000	0	0	0

Based on the principle of SVPWM , the simulation model for generating SVPWM wave forms mainly include the sector judgment model, calculation of operation time of fundamental vectors, calculation of switching time (duty cycle) and generation model of SVPWM waveforms.

Sector Judgment

For implementing the technique of SVPWM, initially it is needed to determine the sector where the voltage vector, which is determined from controller output V_α and V_β, is within. Considering that the expression of vector in the α-β coordinate is suitable for controlling implementation, the following procedure is used for determining the sector.

When V_β>0, A = 1; when $\sqrt{3}V_\alpha$-V_β>0, B=1; when $\sqrt{3}V_\alpha$+V_β>0, C=1. Then the sector containing the voltage vector can be decided according to N = A+2B + 4C, as listed in the Table 2.2 below.

Table 2.2: The sector containing the voltage vector versus N

Sector	I	II	III	IV	V	VI
N	3	1	5	4	6	2

Calculation of operation times of fundamental vectors

The following table lists the operation times of fundamental vectors against N, where T1 and Tm refer to the operation times of two adjacent non-zero voltage space vectors in the same sector, where, Z=T$\frac{-\sqrt{3}V_\alpha+V_\beta}{\sqrt{V_{dc}}}$, Y= T$\frac{\sqrt{3}V_\alpha+V_\beta}{\sqrt{V_{dc}}}$, X= 2T$\frac{V_\beta}{\sqrt{2V_{dc}}}$. The sum of T_1 and T_m should be smaller than or equal to T (PWM period). The over saturation state must be judged: if T_1 +T_m>T, take T_1 = $T_1[\frac{T}{T_1+T_m}]$, T_m = T_m $[\frac{T}{T_1+T_m}]$.

Table 2.3: Operation time of fundamental vector

N	1	2	3	4	5	6
T_1	Z	Y	$-Z$	$-X$	X	$-Y$
T_m	Y	$-X$	X	Z	$-Y$	$-Z$

Generation of SVPWM wave form

The relation between N and switch operation times is shown in table below.

Where, T_a = $\frac{T-T_1-T_m}{4}$, T_b = T_a + $\frac{T_1}{2}$, T_c = T_b + $\frac{T_m}{2}$ and T_{cm1}, T_{cm2} and T_{cm3} are the operation times of the three phase respectively.

Table 2.4: Relation between N and operation time

N	1	2	3	4	5	6
T_{cm1}	T_b	T_a	T_a	T_c	T_c	T_b
T_{cm2}	T_a	T_c	T_b	T_b	T_a	T_c
T_{cm3}	T_c	T_b	Tc	T_a	T_b	T_a

By comparing the computed T*cm1*, T*cm2* and T*cm3* with the equilateral triangle diagram, a symmetrical SVPWM wave form can be generated.

Chapter Three

System Simulation and Results

3.1 System Simulation

The simulator for observer based speed control of PMSM is Matlab/Simulink R2009b as shown in Figure 3.1. The overall system simulation block includes different sub-functional blocks: PI controller blocks, Coordinate Transformation blocks, Observer block and SVPWM block are main sub-functional blocks. The simulation is based on discrete system with sampling period of 10μs.

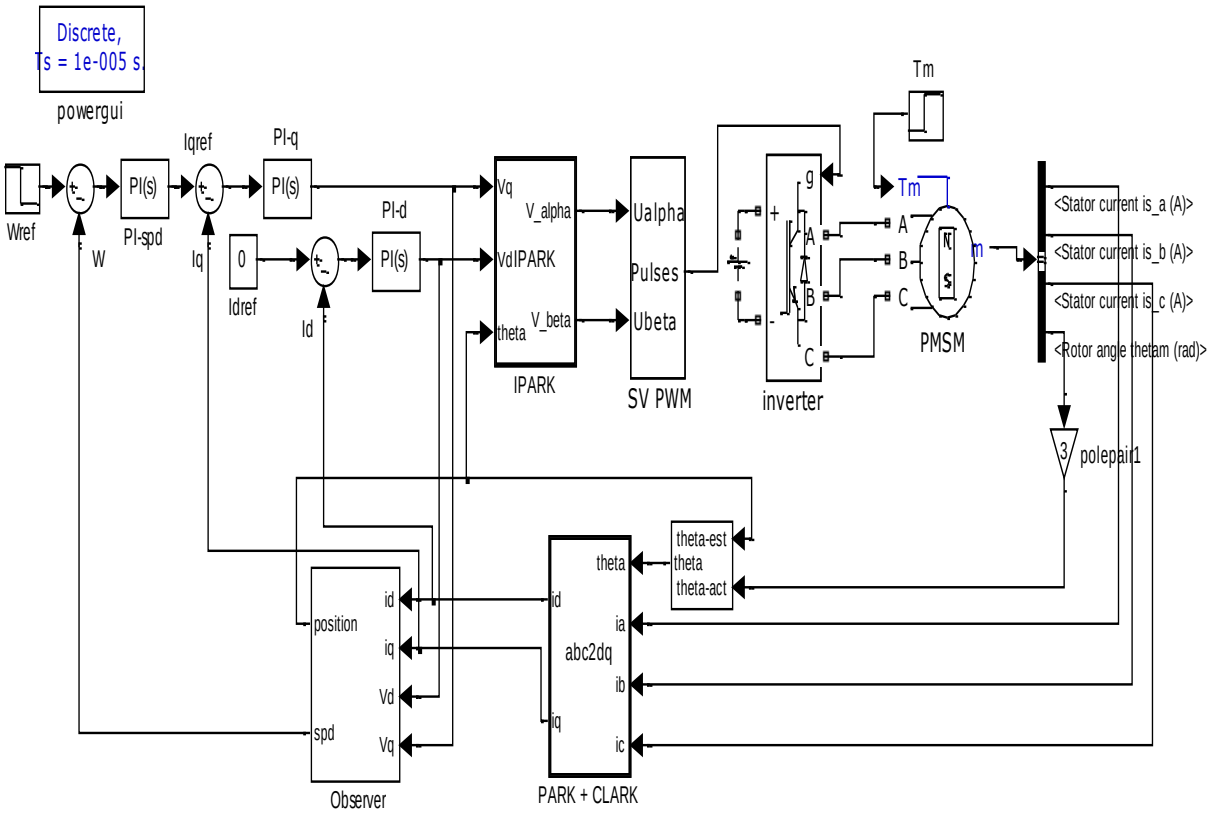


Figure 3.1: Overall observer based speed control of PMSM system simulation block diagram

The PI controller blocks are taken from Matlab Simulink library and adjusting the Proportional and Integral gain parameters calculated from the PI controller design as stated in Chapter Two.

The different coordinate transformations are written in Matlab source code and embedded in simulation blocks. Figure 3.2 shows simulation block diagram of Space Vector Pulse Width Modulated signal. SVPWM block which is a Matlab source code embedded on simulation block taking PWM period, Dc voltage, state voltage in alpha and beta direction as input, Tcm1, Tcm2, Tcm3 operation times as output. The output operation times are compared with a saw tooth repeating sequence signal with the same frequency as PWM to generate pulse width modulated signal. Finally the three output signals are passed through inverter logic operator to produce the inverse of them. These six pulse width modulated signals are used to drive a three phase voltage source inverter.

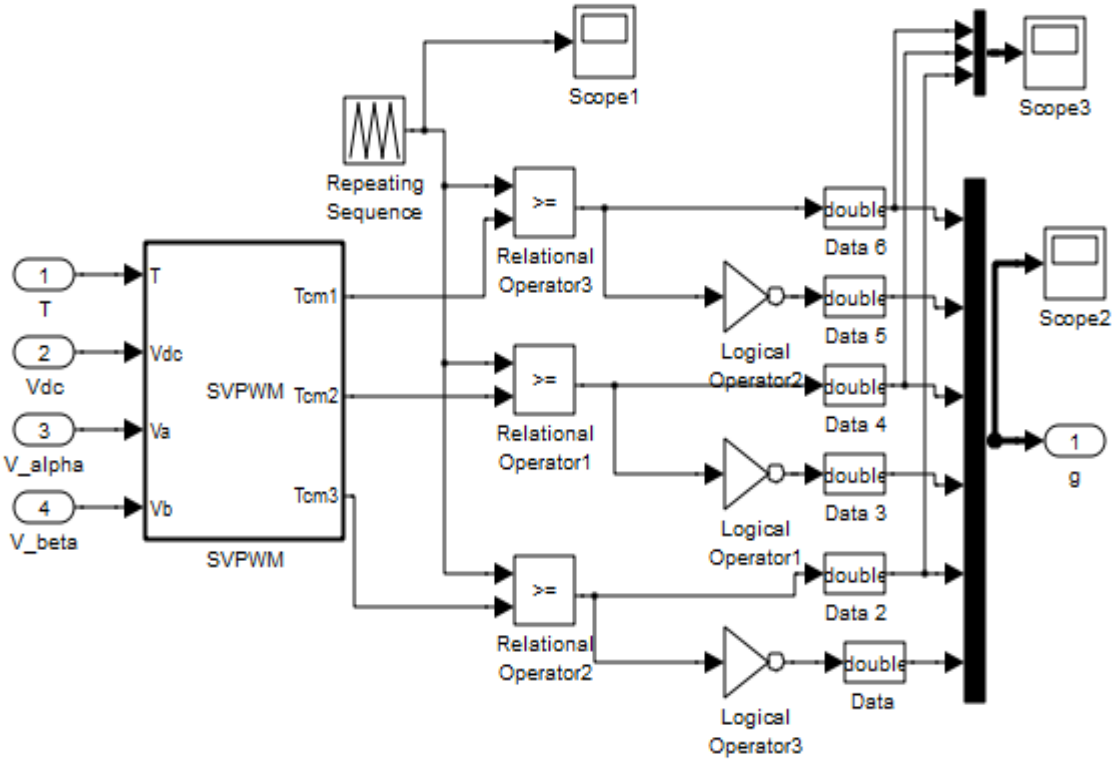


Figure 3.2: SVPWM generation simulation diagram

The main part in the system simulation is the Observer block which is an Embedded Matlab source code block diagram derived from Extended Kalman Filter algorithm. The block takes q and d axis current and voltages as input, the estimated speed and position as output as we can see

from Figure 3.3. The block works as recursive process which takes some of its outputs as inputs for the next computation process.

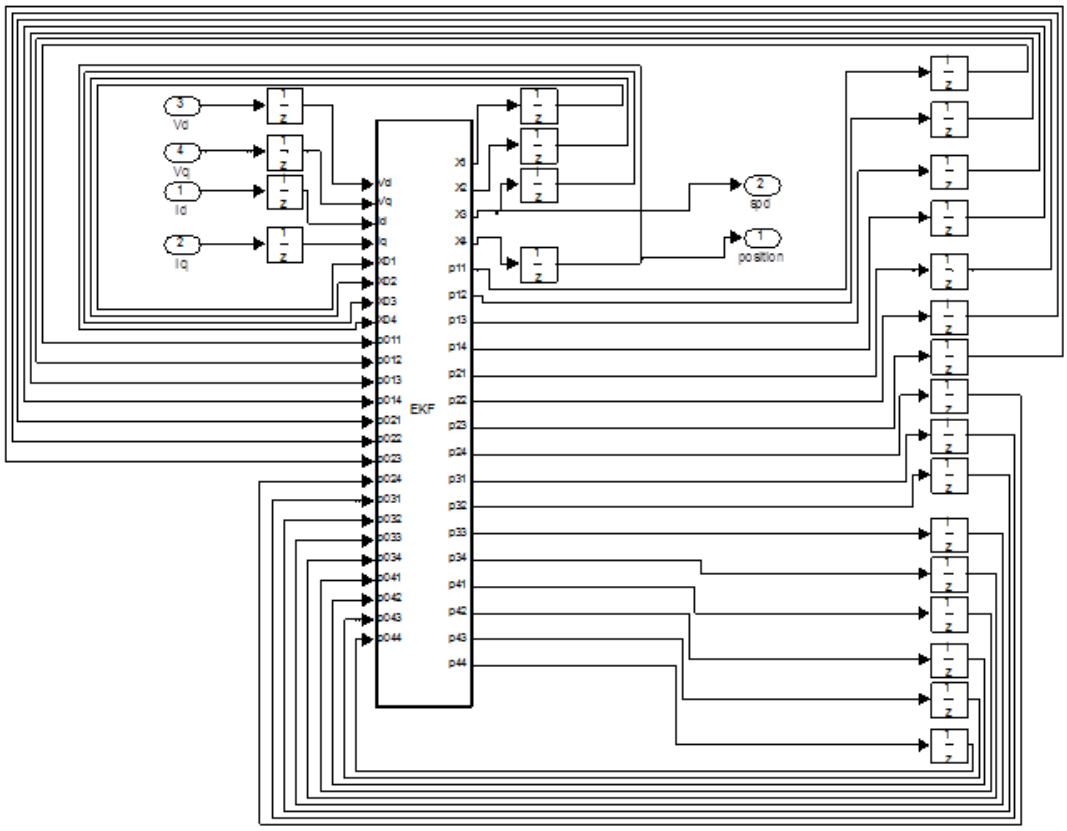


Figure 3.3: Extended Kalman Filter simulation block diagram

One of the basic things in sensorless control of PMSM is detecting the initial position of the rotor for startup. This problem is the current research area in motor control. In this paper, this problem is alleviated by running the motor initially with a known position for a certain time till the system stabilize using ramp angle generation method and switching to a closed loop sensorless condition. The rotor position is generated with a certain frequency which determines the frequency of three phase voltage applied to the motor. This indirectly determines the speed of the rotor. This is done until the motor reaches appropriate speed value.

3.2 Simulation Results

The first simulation result for the observer based speed control of PMSM is the three phase stator currents which are generated by the three phase inverter. This three phase inverter is controlled by SVPWM block for appropriate stator current generation. These three phase currents should be equal magnitude and 60° phase shift with each other for appropriate rotating flux generation as shown in Figure 3.4. The detail view it is also shown below for a certain time interval.

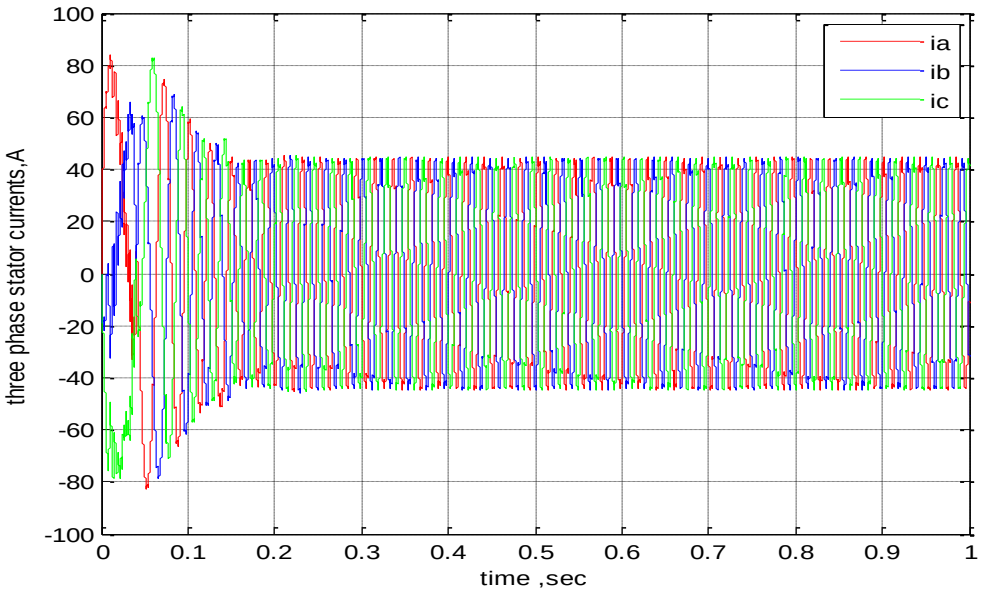


Figure 3.4: Three phase stator current in A vs. time

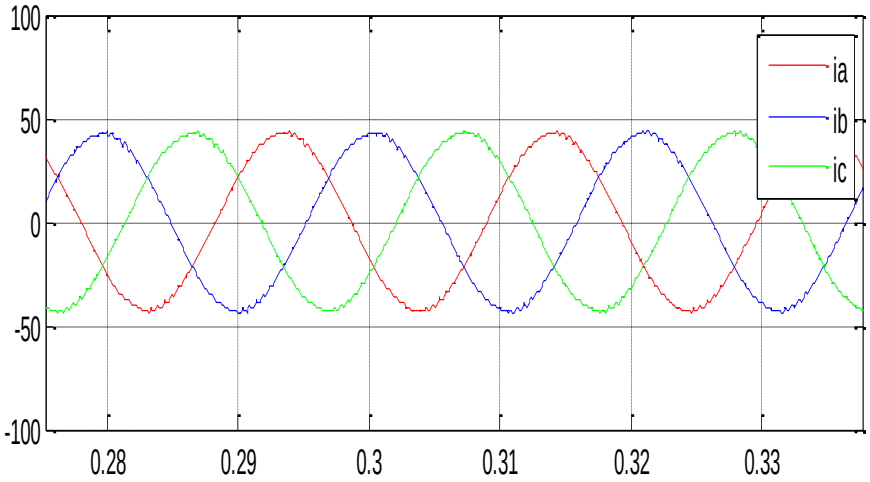


Figure 3.5: Three phase currents zoom out between 0.275 and 0.336 sec

As we see from in the above Figure, the appropriate stator phase current value is generated with good accuracy. Hence the system can feed the appropriate stator voltage to the motor. If the voltage applied to the motor is applied with appropriate magnitude and frequency, the speed of the motor is respected as set to the reference value. Figure 3.6 shows simulation result for an actual and estimated speed with reference speed of 300rad/sec (2865rev/min). The system can also follow the reference signal with a rise time of less than 0.2 second and settling time of 0.25second. The system also gives a good overshoot value with steady state error of 0.05%.

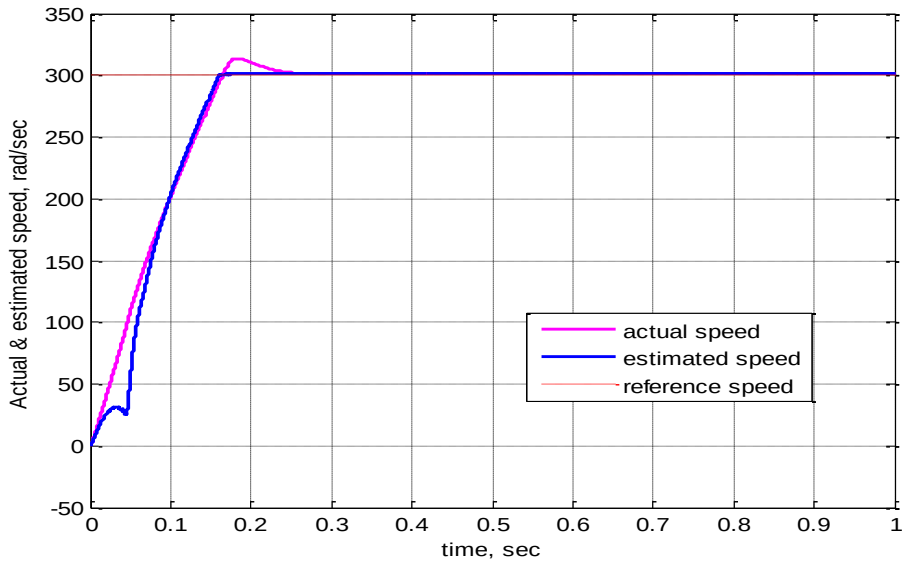


Figure 3.6: Actual and Estimated speed vs. time diagram

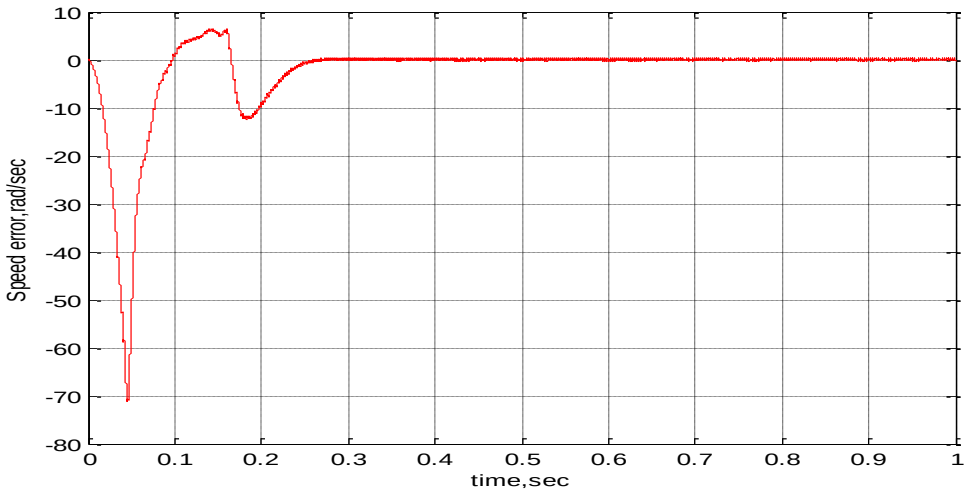


Figure 3.7: Error signal between actual and estimated speed in rad/sec vs. time

The above diagram shows the error signal between actual and estimated speed is ±0.2rad/sec which is tolerable. The actual and estimated rotor position of the motor is also shown in Figure 3.8 with respect to time in second. This figure indicates the Observer can estimate the rotor position with an error 2.4º of as shown in Figure 3.9.

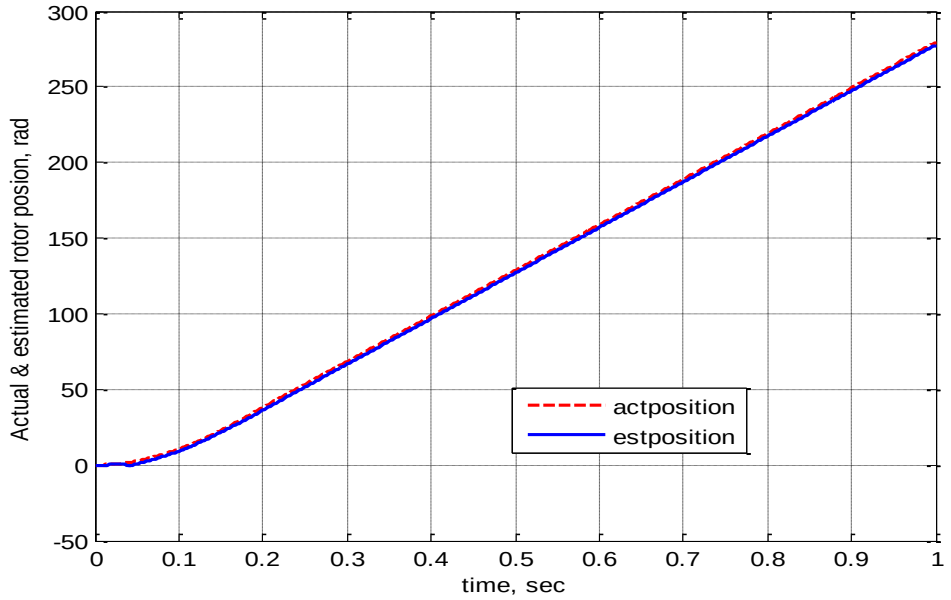


Figure 3.8: Actual and Estimated Position vs. time diagram

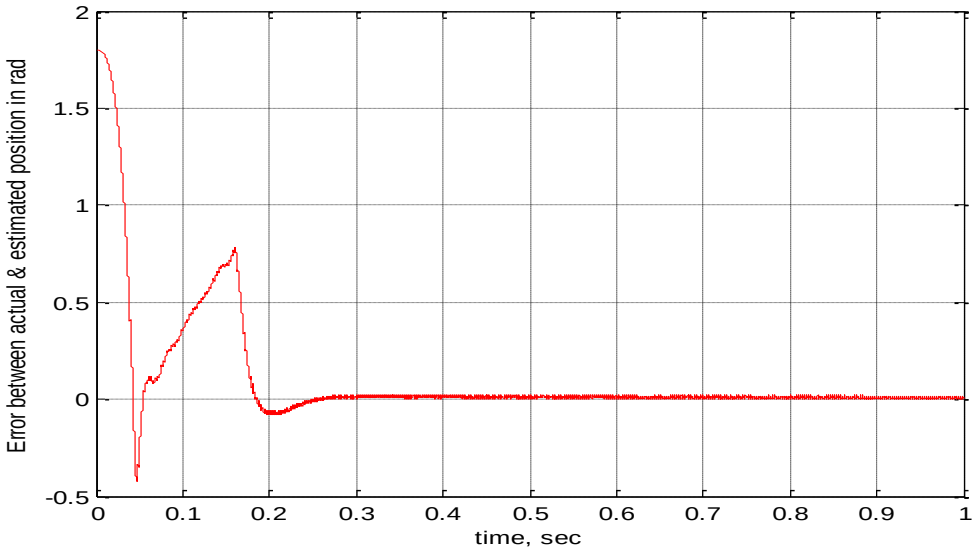


Figure 3.9: Error signal between actual and estimated position in rad vs. time

The system also works in forward and reverse condition. Figure 3.10 shows simulation result while the motor is running in 300rad/sec forward and reverse condition. This result indicates robustness of the system.

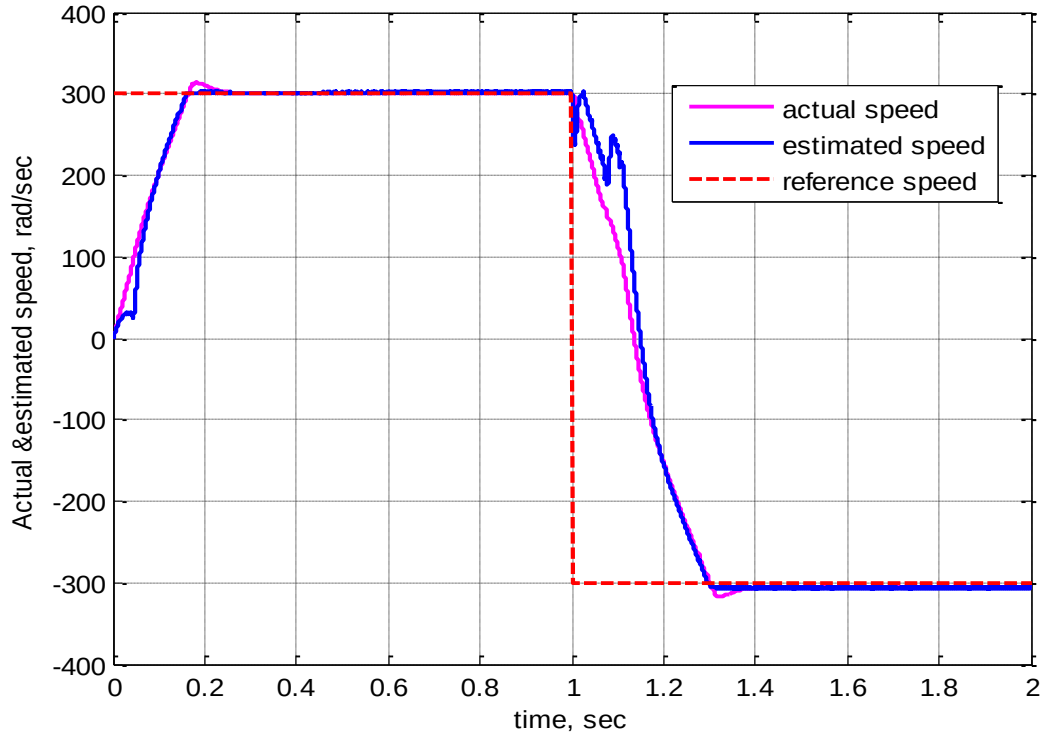


Figure 3.10: Actual and estimated speed (rad/s) at forward and reverse condition vs. time

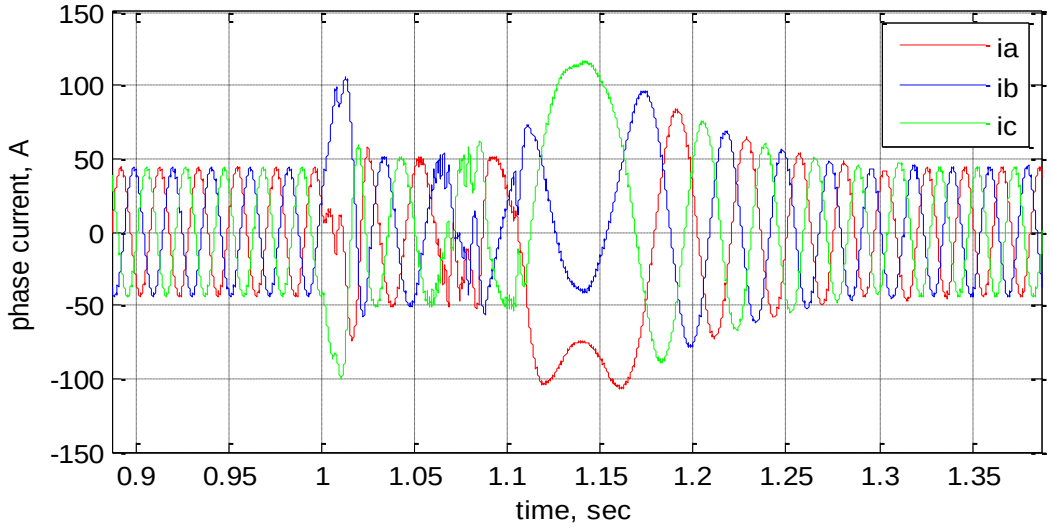


Figure 3.11: Stator currents in forward and reverse condition with respect to time

As stated in Chapter Two, the advantage of vector control is directly controlling the motor torque by using the stator current like DC motor control. Figure 3.10 and Figure 3.11 shows this fact in which the Electromagnetic torque is directly proportional with the q-axis current. The simulation was taken place with a load torque of 3Nm.

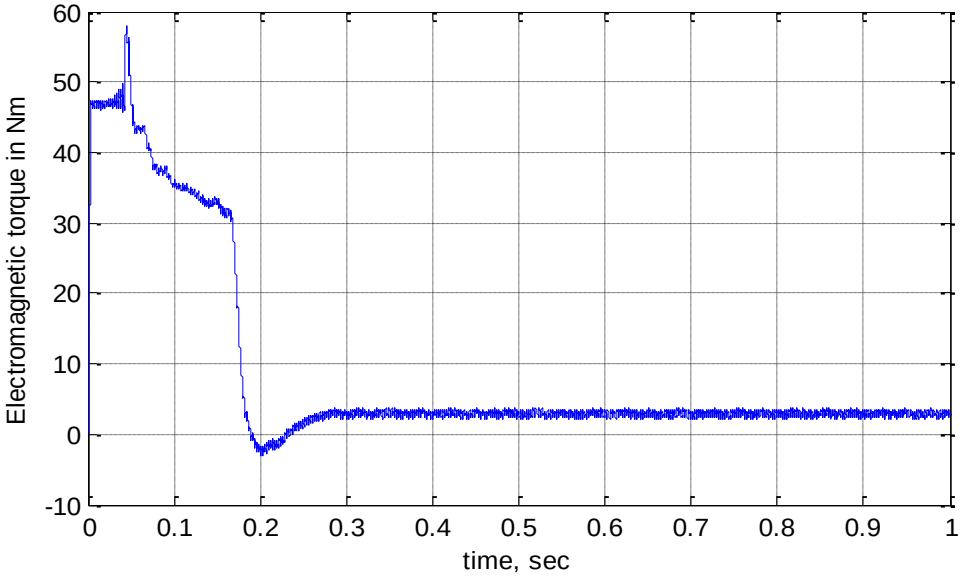


Figure 3.12: Motor electromagnetic torque in Nm with respect to time in sec

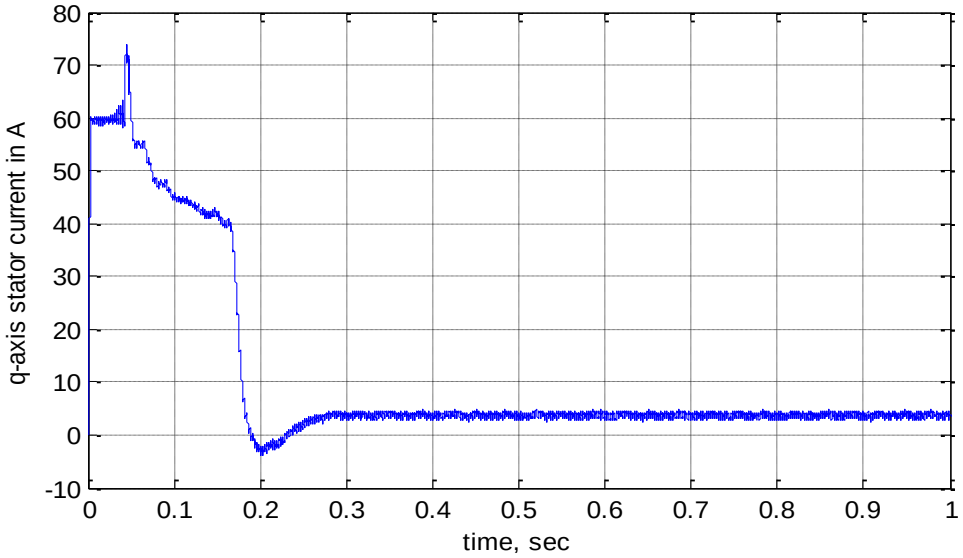


Figure 3.13: q-axis stator current in A with respect to time in sec

The system also works at different load torque condition. Figure 3.14 and Figure 3.15 shows actual and estimated speed of the motor at 0Nm (no-load) and 5Nm load torque condition

respectively. As we can see from the figures, the system has good steady state and transient performance.

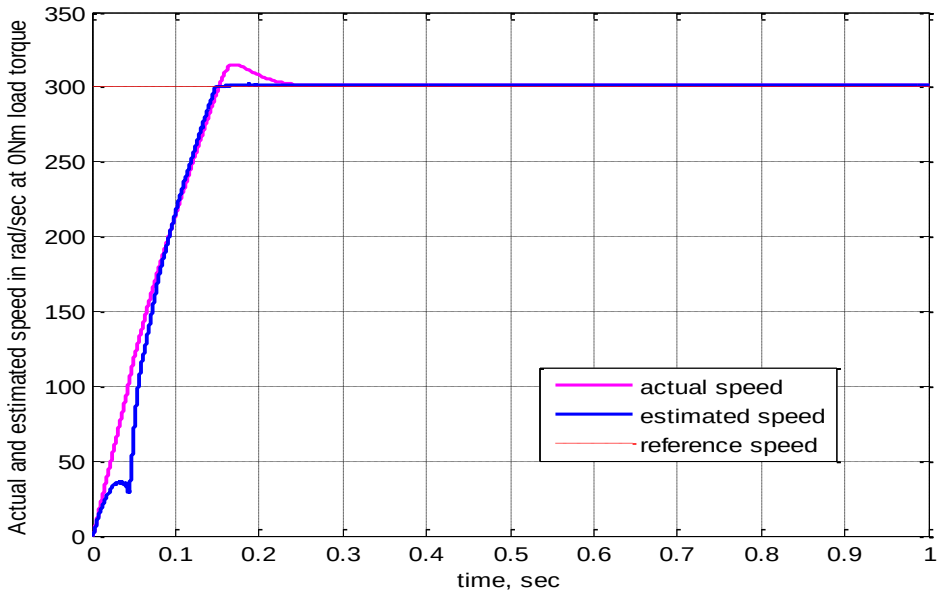


Figure 3.14: Actual and estimated speed in rad/sec at 0Nm load torque (No-load) condition

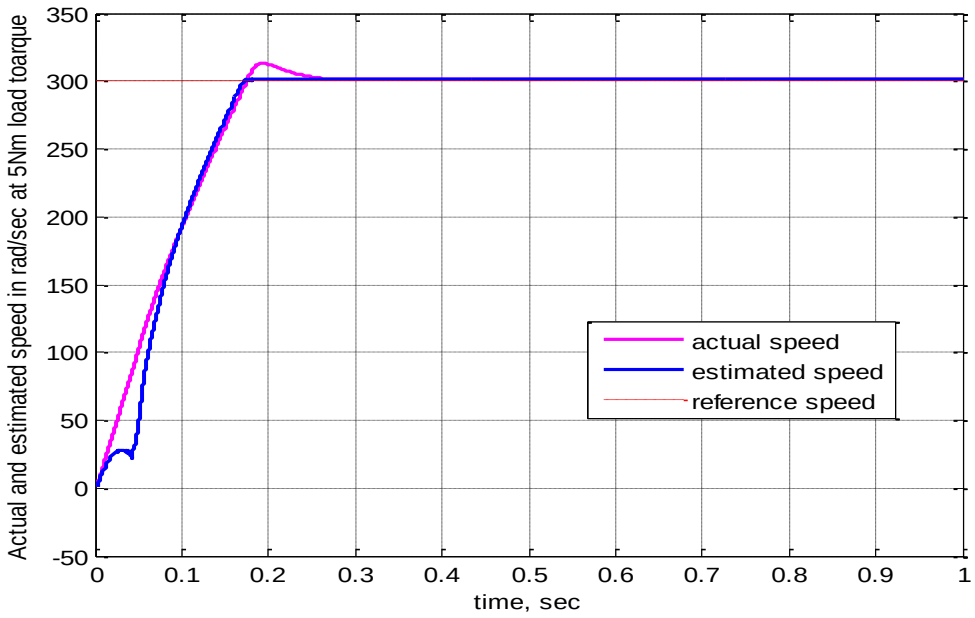


Figure 3.15: Actual and estimated speed in rad/sec at 5Nm load torque

The Electromagnet torque and Stator phase currents of the system at different load condition also have good transient and steady state response. The following figures show these properties. For instance Figure 3.16 and Figure 3.18 show the Electromagnetic torque steady state response of the system which is around 0Nm and 5Nm with good transient response. The corresponding three phase stator current also has good steady state and transient response.

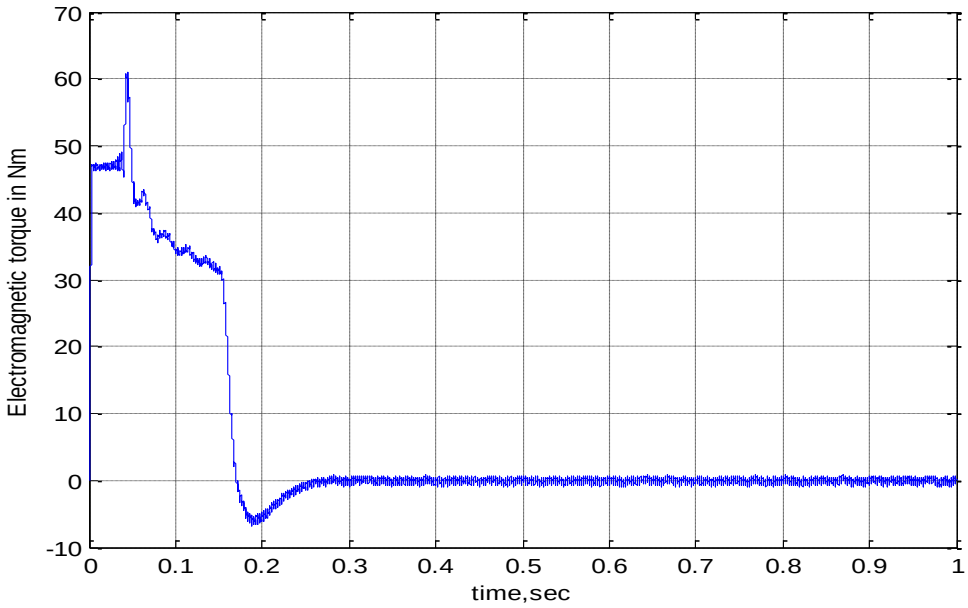


Figure 3.16: Electromagnetic torque vs. time while the motor is running at no-load

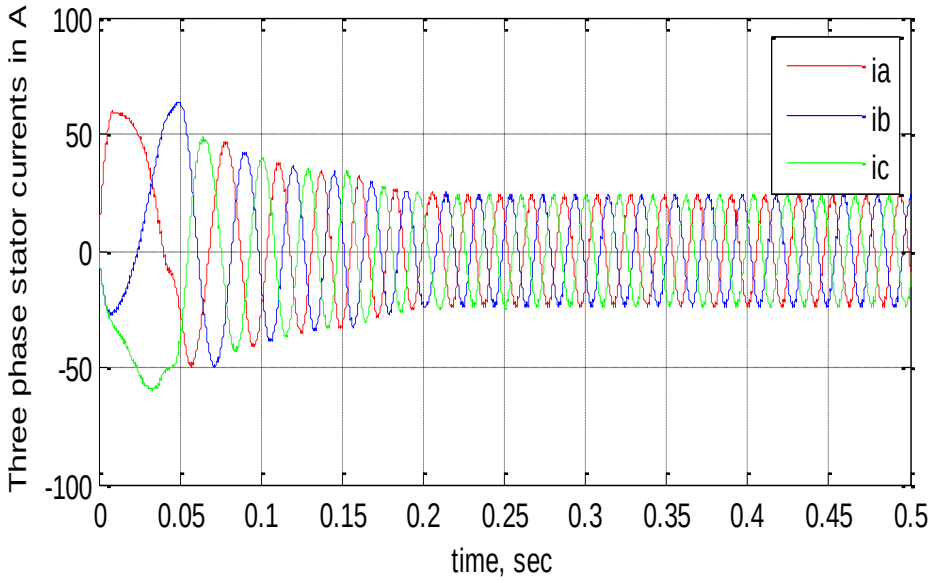


Figure 3.17: Three phase stator current vs. time while the motor is running at no-load

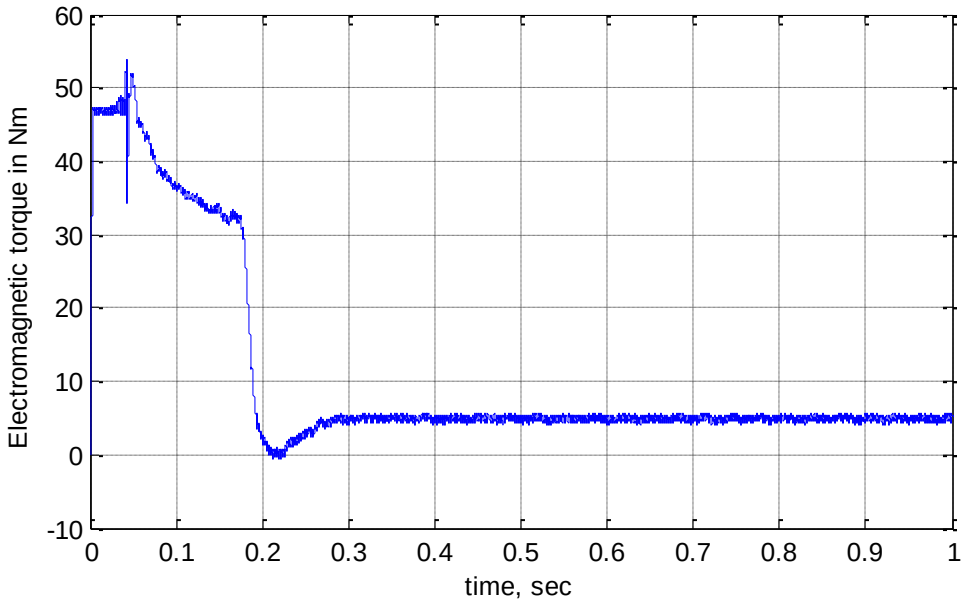


Figure 3.18: Electromagnetic torque vs. time while the motor is running in loaded condition (5Nm load torque)

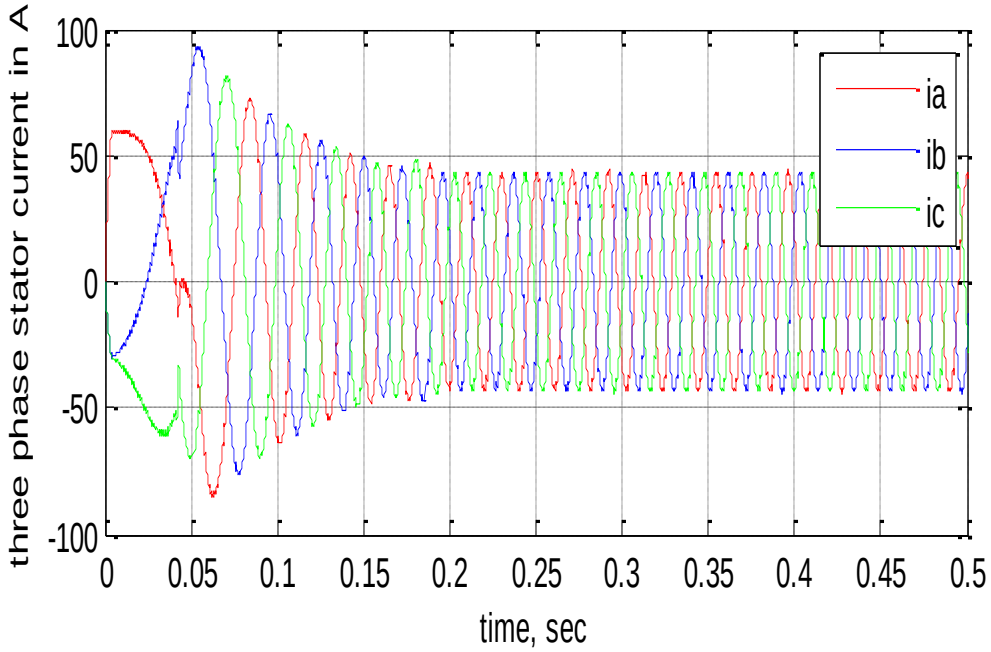


Figure 3.19: Three phase stator current vs. time while the motor is running in loaded condition (5Nm load torque)

Chapter Four

Experimental System and Results

4.1 Introduction

The overall experimental system contains both hardware and software. The hardware includes: TMS320F2812 DSP, Two current sensors, AC drive and Inverter, current conditioning circuit and the motor. The software includes different coordinate transformation algorithms, controller algorithms, estimator algorithm, space vector pulse width modulation algorithm (SVPWM) and others as shown in Figure 4.1.

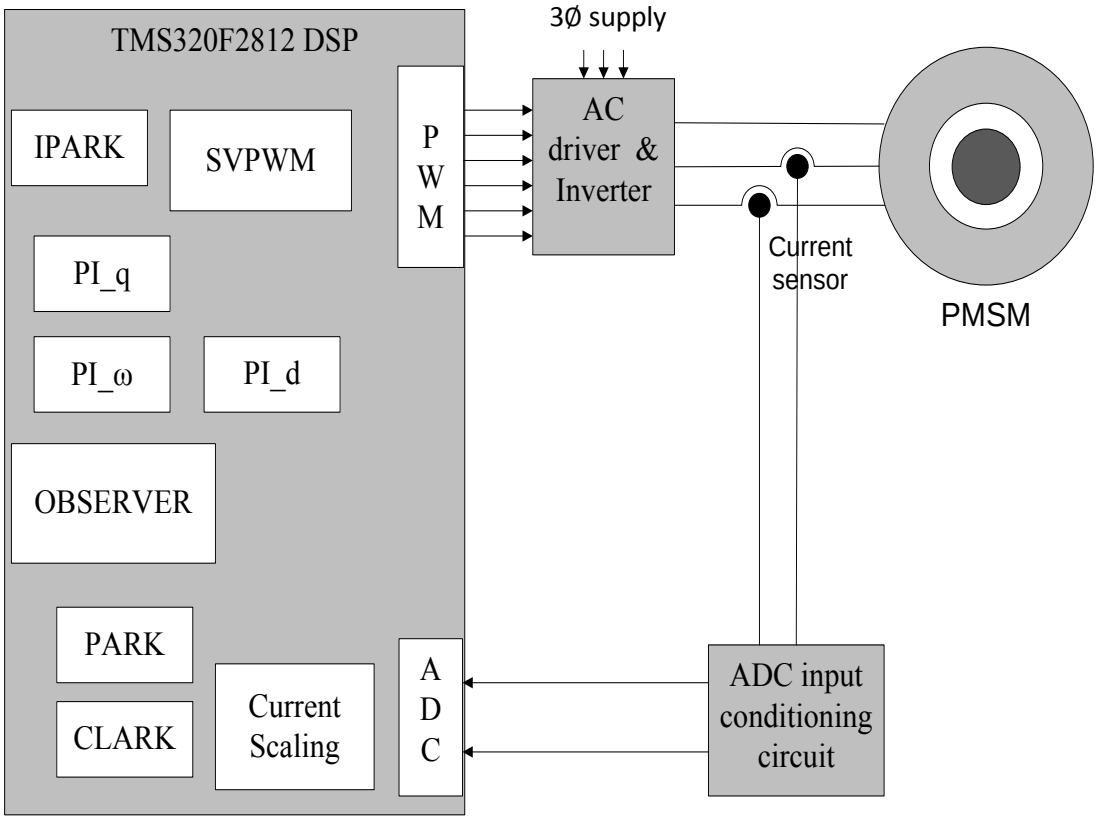


Figure 4.1: Overall experimental system organization

4.2 System Hardware

The system hardware contains eZdsp™F2812 module, Inverter, two current sensors, ADC input conditioning signal and the Permanent magnet synchronous motor for experimental investigation.

Overview of eZdsp™F2812

The eZdsp™F2812 is a standalone-card allowing evaluators to examine the TMS320F2812 digital signal processor to determine if it meets their application requirements. Furthermore, the module is an excellent platform to develop and run software for the TMS320F2812 processor. To simplify code development and shorten debugging time, a C2000 Tools Code Composer drive is provided. In addition, an on board JTAG connector provides interface to emulators, operating with other debuggers to provide assembly language and 'C' high level language debug.

Key Features of the eZdsp™F2812 [26]

It has the following features:

- Contains TMS320F2812 Fixed Point Digital Signal processor
- 150MIPS operating speed
- 18K words on chip RAM
- 128K words on-chip Flash memory
- 64K words off-chip SRAM memory
- 30MHz. Clock rate
- 2 expansion connectors (analog and I/O)
- Onboard IEEE 1149.1 JTAG Connector
- 5-volt only operation with supplied AC adapter
- TI F28xx code composer studio tools driver

Figure 4.2 shows the functional overview of the eZdsp™F2812. The major interfaces of eZdsp are the JTAG (Joint Test Action Group) interface, Analog input connector, I/O interface connector and expansion interface. JTAG connector uses to connect JTAG emulators to the DSP board. It comes in a variety of different ways to connect to the PC. The USB and Parallel Port are the most common interfaces used.

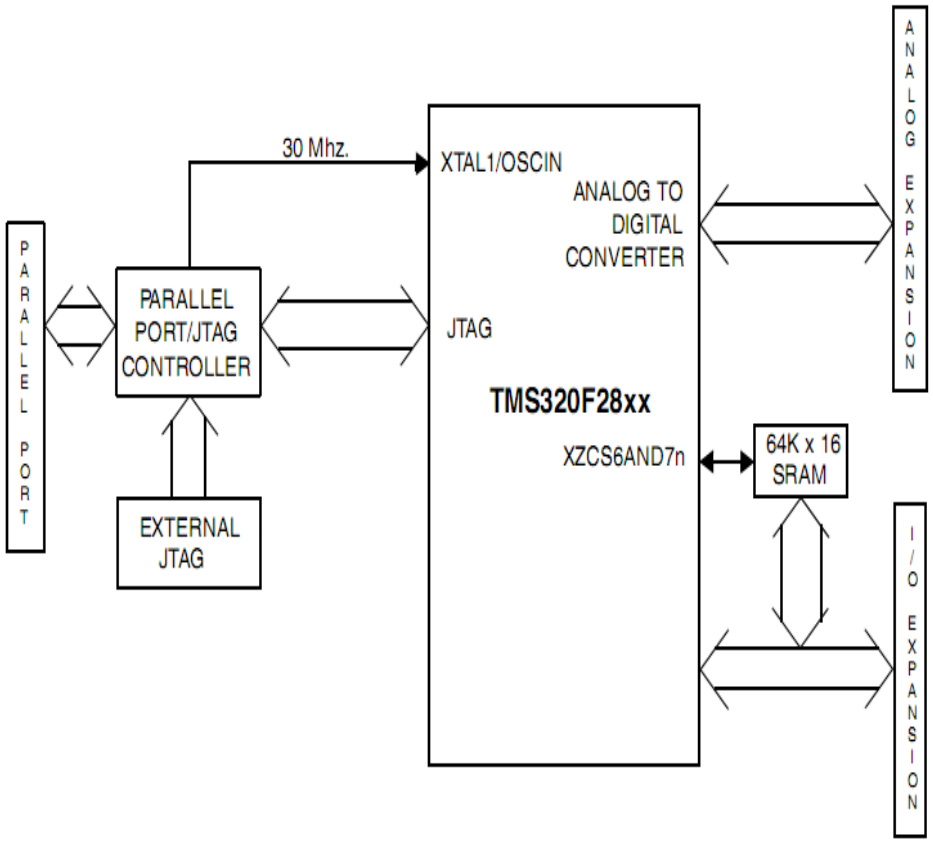


Figure 4.2: Functional overview block diagram of the eZdsp™F2812

TMS320F2812 is a new generation controller with main frequency of 150MHz. The speed and feature of this DSP make it an excellent choice for the digital control of the motor. It has 12-Pulse Width Modulation (PWM) output and 12-bit analog to digital converter (ADC). The fast conversion period, which is 80ns, is very suitable for real time sampling. Figure 4.3 shows the functional overview of the TMS320F2812 in detail including its different ports, different type internal memory with respected size, different data and address paths.

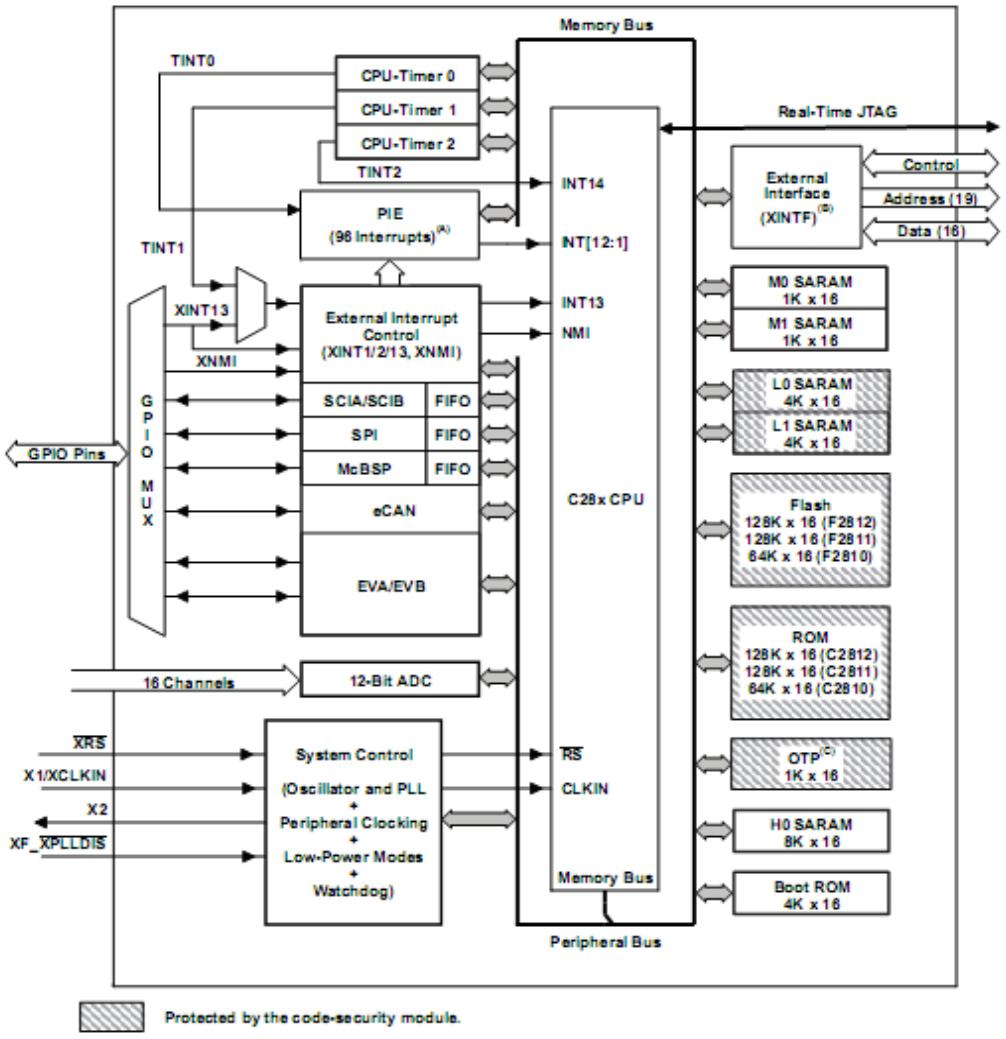


Figure 4.3: Functional overview of TMS320F2812, [23]

Processors can be divided into two groups, ‘’floating point’’ and ‘’fixed point’’. The core of a floating point processor is a hardware unit that supports floating point operations according to international IEEE 754. Intel’s x86 – family of Pentium processor is a typical example of this type. Floating point processors are very efficient when operating with floating point data and allow a high dynamics range for numerical calculations. They are not so efficient when it comes to control tasks of bit manipulation, input output manipulation, and interrupt response. They are also expensive.

Fixed point processors are based on internal hardware that supports operations with integer data. The arithmetic logic unit and, in case of digital signal processors, the hardware multiply unit,

expect data to be in the fixed-point types. They have limitations in the dynamic range of a fixed point processor, but they are inexpensive.

If we write a program for a fixed point processor in C and we declare a floating point data type ‘’float’’ or ‘’double’’, a number of library functions support this data type on a fixed data type on a fixed data machine. These standard ANSI-C functions consume a lot of computing power. Recalling the time constraints in a real time project, we just can’t afford to use these data types in most of embedded control applications. The solution in case of the C28x is ‘’IQ-math’’; the IQ-math library is a collection of highly optimized and high precision mathematical functions used to seamlessly port floating point algorithms in to fixed point code. In addition, by incorporating the ready to use high precision functions, the IQ-math library can shorten significantly an embedded control development time.

The TMS320F2812 have different module and signals, such as general purpose input output (GPIO), Analog to digital converter (ADC), Event manager (EV), Timer module, System control and interrupts, External interface (XINTF), Input/output signal etc. Among them for motor control system ADC and EV are the basics one.

Analog-to-Digital Converter

The TMS320F2812 ADC module is a 12-bit pipelined analog to digital converter (ADC). The analog circuits of this converter include the analog multiplexers (MUXs), sample-and-hold (S/H) circuit, the conversion core, voltage regulators, and other analog supporting circuit such as ADCRESULT Registers. The ADC module has 16 channels, configurable as two independent 8-channel modules to serve event managers A and B. The two independent 8 channel can be cascaded to form a 16 channel module.

The two 8-channel modules have the capability to auto sequence a series of conversions; each module has the choice of selecting any one of the respective eight channels available through an analog MUX. In the cascaded mode, the auto sequencer functions as a single 16-channel sequencer. On each sequencer, once the conversion is complete, the selected channel value is stored in its respective ADCRESULT register. Auto sequencing allows the system to convert the same channel multiple times, allowing the user to perform oversampling algorithms. This gives increased resolution over traditional single-sampled conversion results.

Features of the ADC module include [24]:

- 12-bit ADC core with built-in dual sample-and-hold (S/H).
- Simultaneous sampling or sequential sampling modes.
- Analog input: 0 V to 3 V.
- Fast conversion time runs at 25 MHz, ADC clock, or 12.5 MSPS.
- 16-channel, multiplexed inputs.
- Auto sequencing capability provides up to 16 "auto conversions" in a single session. Each conversion can be programmed to select any one of 16-input channels.
- Sequencer can be operated as two independent 8-state sequencers or as one large 16-state sequencer (i.e., two cascaded 8-state sequencers).
- Sixteen result registers (individually addressable) to store conversion values. The digital value of the input analog voltage is derived by:

The digital value of the input analog voltage is derived by:

$$\text{Digital value} = 4096 \times \frac{\text{Input Analog Voltage} - \text{ADCLO}}{3}$$

There are multiple triggers as source for the start of conversion (SOC) sequence:

- S/W – immediate start
- EVA – event manager A (multiple event sources within EVA)
- EVB – Event manager B (multiple event sources within EVB)
- External pin

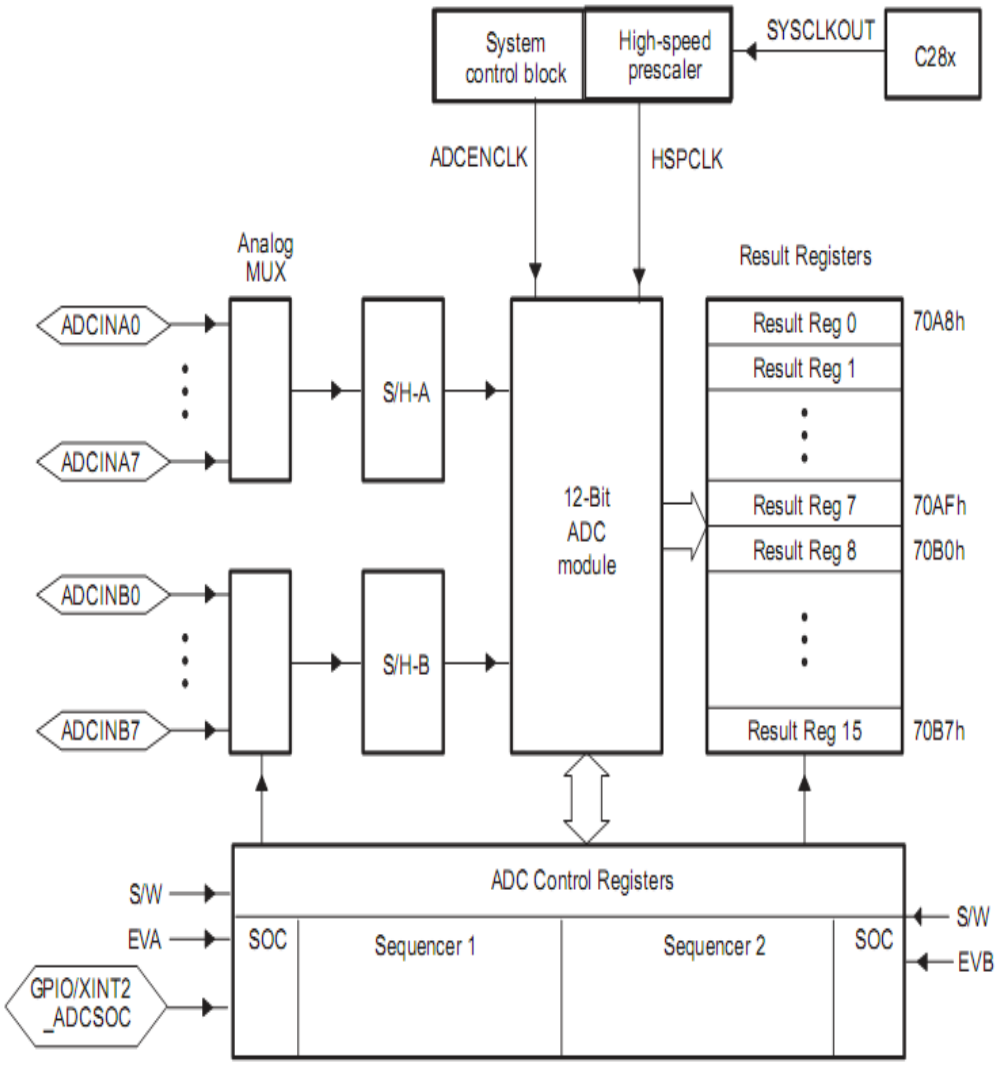


Figure 4.4: Block diagram of ADC module and its different registers [15]

Event Manager [25]

The Event Manager (EV) modules provide a broad range of functions and features that are particularly useful in motion control and motor control applications. The EV module includes general purpose (GP) timers, full-compare/PWM units, capture units, and quadrature-encoder pulse (QEP) circuits. The two EV modules, EVA and EVB are identical peripherals, intended for multi-axis motion control application. Each EV is capable of controlling three Half-H bridges, when each bridge requires a complementary PWM pair of control. Each EV also has two additional PWMs with no complimentary output.

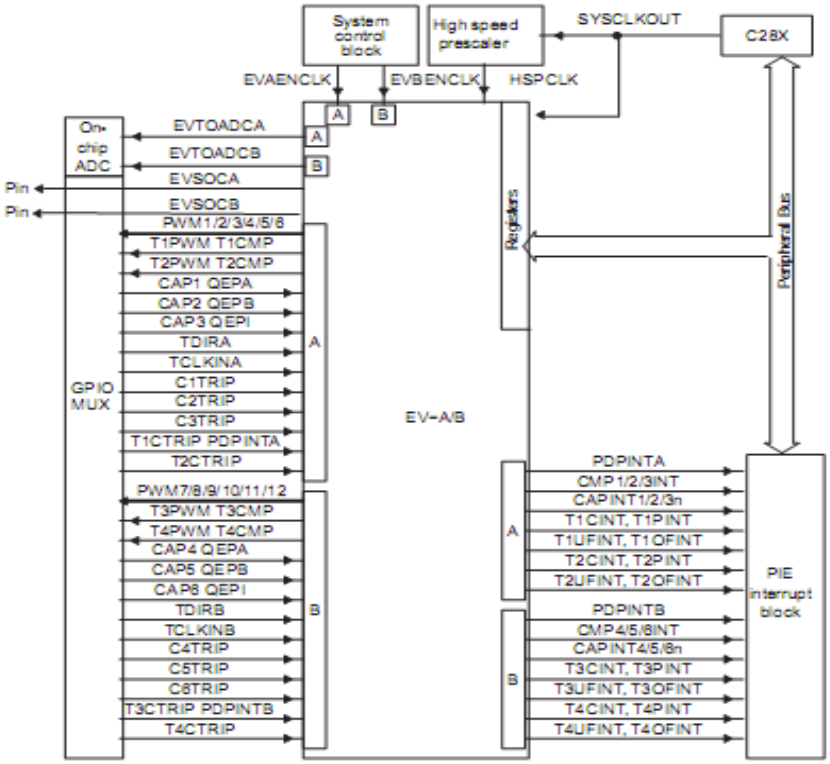


Figure 3.5: EV module and connection with other peripherals [17]

Input/output signal

The numbers of input/output pins are 56 in total. Most of these pins have multiple functions set by flags in registers. Each pin can be set as a standard I/O pin, reading or writing digital data. Special function such as Pulse Width Modulation and Analog to Digital Conversion can also be assigned to specific predefined pins.

Timer Module

The DSP is equipped with four independent timers (two for each EVx). The Timer is useful for time critical applications. It works by counting up/down a 16 bit value and generates an interrupt at a predefined value indicating ‘’Time is up’’.

Code Composer Studio (CCS)

The code development is carried out through an application suite called code composer studio (CCS v3.1). It is made by Texas instruments and is free of charge. It supports all their families and variations of DSPs. The key feature of the CCS is to serve the whole development chain. It

supports programming in both C and Assembler languages. The CCS communicates with the DSP via the computers standard parallel port. In Figure 4.6 the outline of the programming interface is stated. CCS consists of four main parts: Project window, Program window, Watch window and Output window.

In the Project window all files in an open project is shown in a directory structure. A double click on a file will open it in Program window and make it editable. When choosing the compile command, potential errors will show up in the Output window. The Watch window enables tracking of variable values while running the program.

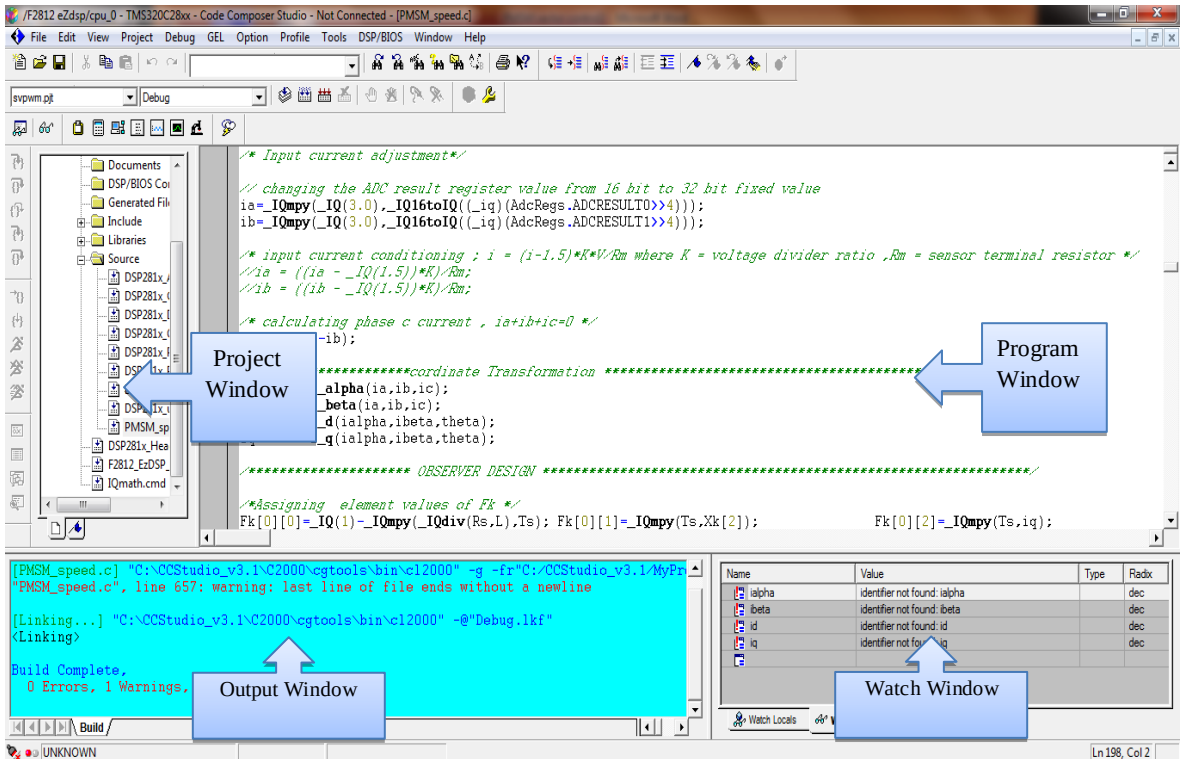


Figure 4.6: Snapshot of CCS programming interface

ADC Input Signal Conditioning Circuit

Since the DSP input-output specification is different from the external environment, some ADC input-output signal conditioning circuit is needed. The DSP input-output specification is:

- DSP Analog interface level in voltage [ADC port] – [0 – 3.3V]
- DSP input-output interface level in voltage [PWM port] – [0 – 5V]

But the external environment specification is different from this. For instance the output of the current sensors contain both positive and negative values and the inverter needs a voltage value greater than 5V to switch its gate power transistors. The DELERENZO DL2646 frequency converter, as shown in Figure 4.7, has its own voltage driver circuit to convert the 5V PWM output to the appropriate gate drive voltage value.

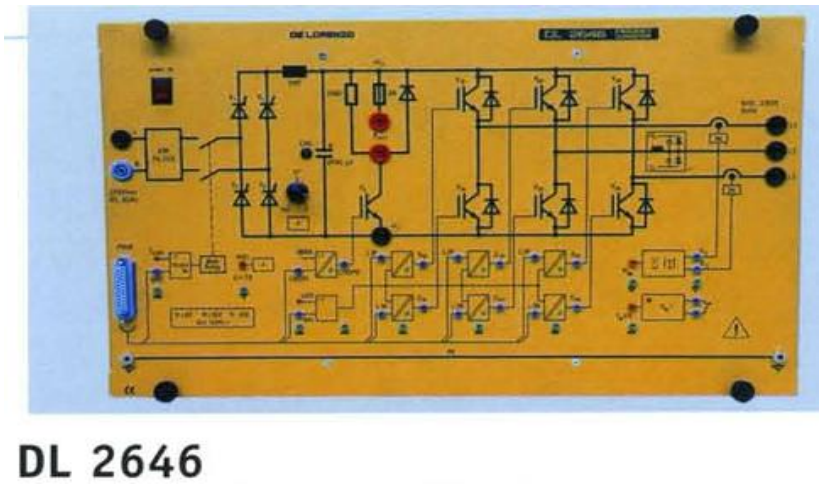


Figure 4.7: DELERENZO frequency converter DL 2646

Figure 4.8 shows the current conditioning circuit using operational amplifier and resistors.

The circuit does the following tasks:

- Change the current sensor output voltage value in the range of [-1.65V, 1.65V]. This is done using voltage divider circuit by selecting appropriate voltage divider constant, K.
- Adding 1.65V DC offset voltage to make the voltage in the range of [0, 3.3V] which is suitable for DSP analog interface using the operational amplifier as voltage adder.
- The operational amplifier also used as a buffer for not loading the circuit.

Then the conditioned voltage value on the ADCRESULT registers is changed to the corresponding current value and into its normal range using multiplication and subtraction algorithms. For instance, let phase-a stator current is I_a, then the current value on the DSP can be calculated as:

$I_a = \frac{(V_{out}-1.65)*K}{0.3}$; where V_{out}Vout is voltage on ADCRESULT register and voltage-current relation of the current transducer is: V_{input} = 0.3*I_a.

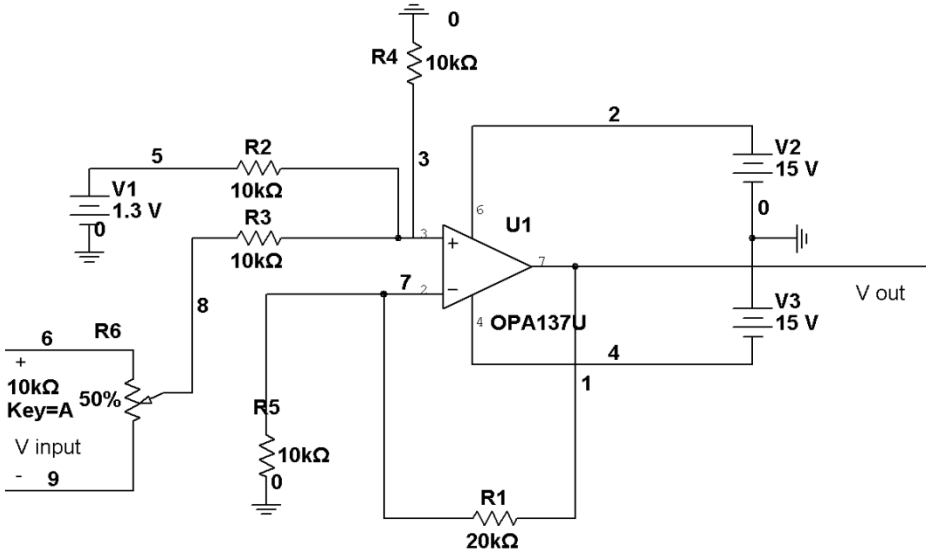


Figure 4.8: ADC input interface signal conditional circuit

4.3 System Software

The overall system software is based on two modules: The initialization module and the interrupt subroutine module.

Initialization module: After a DSP processor reset the initialization module performs the following tasks as described in the Figure 4.9.

- DSP setup: core, clocks, ADC, GPIO, Event manager, etc. initialization.
- Variable initialization.
- Interrupt source selection and enable: it describes enabling the DSP modules to accept interrupt and from where the interrupt is originated.
- Waiting loop: It is a loop which waits for indefinite time length until the interrupt sub module sends the interrupt starting signal.

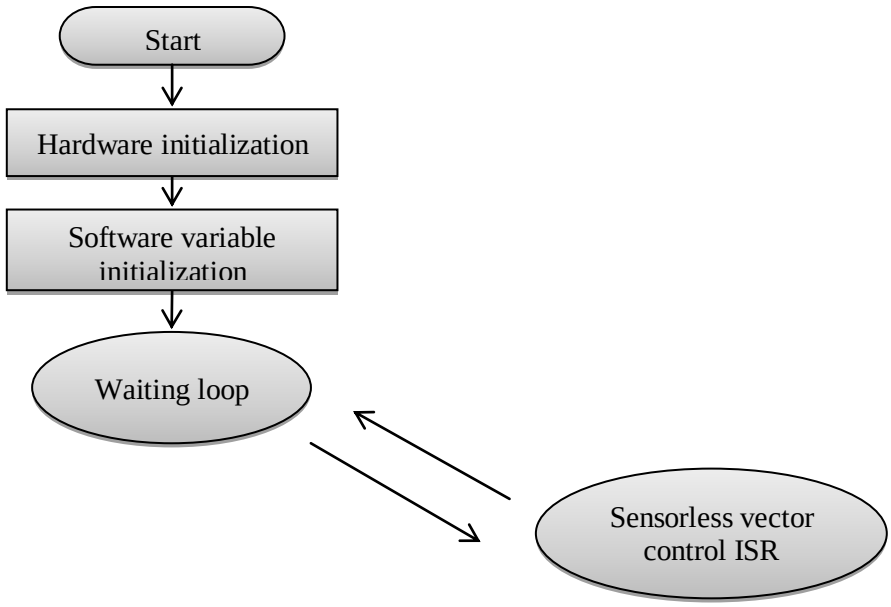


Figure 4.9: System software organization

Interrupt sub routine module description: the interrupt module handles the whole field oriented control (FOC) algorithm. It is periodically computed according to a fixed PWM period of 10µs. The choice of PWM frequency depends on the motor electrical constant L/R. The goal of interrupt module is to update the stator voltage reference value and to ensure the regulation of stator currents and rotor mechanical speed to their reference value. The module contains the following algorithms as we can see from Figure 4.10:

- Reading ADC output current value and scaling up it: The stator current sensor output is connected to the ADC input of the digital signal processor. The DSP processor reads this current value from the ADC result register.
- Coordinate transformation: Different coordinate transformations for FOC are computed.
- Speed and position estimation algorithm: It is an Extended Kalman filter algorithm as observer.
- Speed and current control algorithm: It is Proportional Integral (PI) controller for controlling the speed and the torque of the motor.
- SVPWM algorithm: it is an algorithm used to generate a nearly sinusoidal stator current by the inverter using q-axis and d-axis stator voltages.

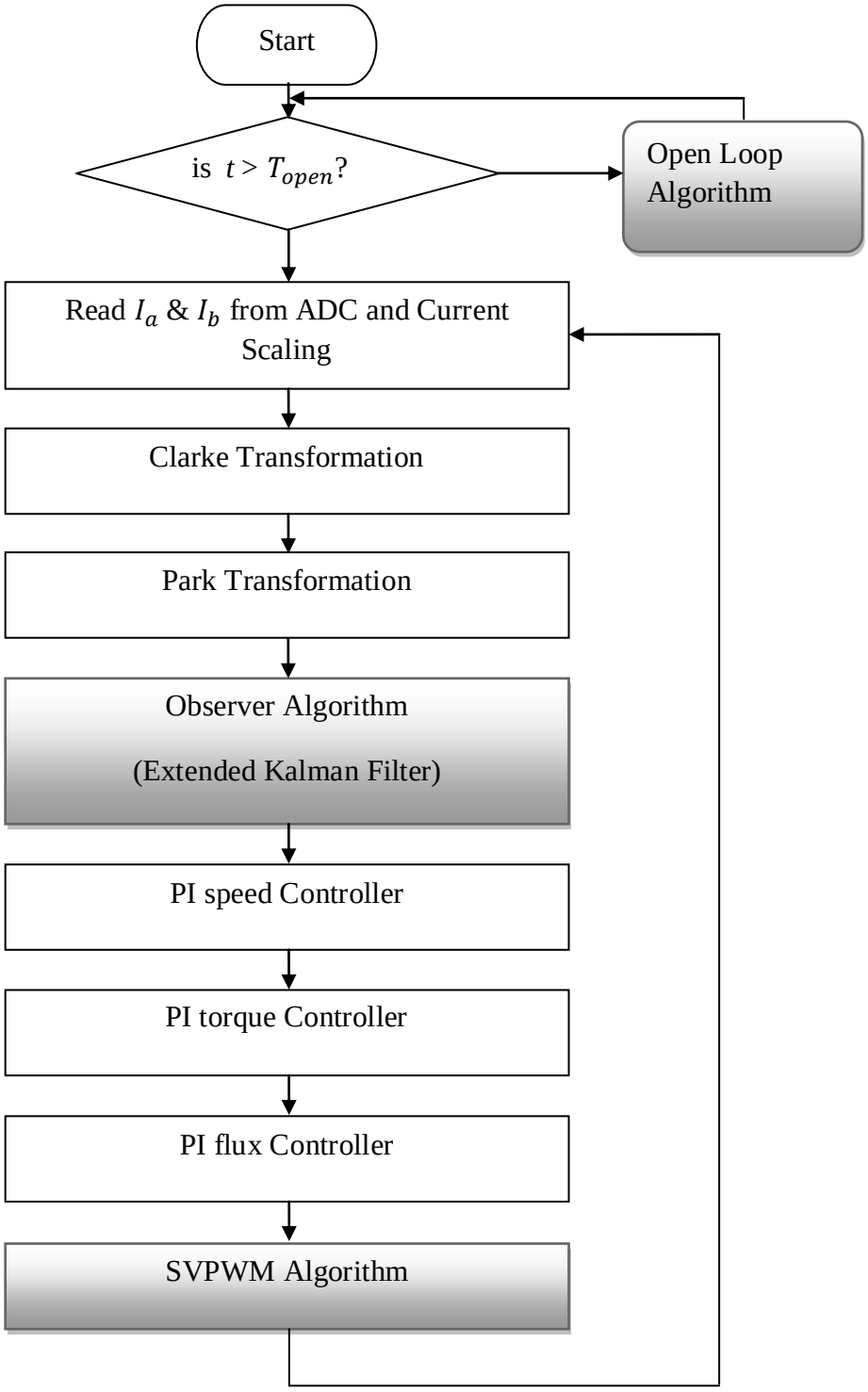


Figure 4.10: Interrupt subroutine system flowchart

Since estimating the rotor position at standstill is difficult because the extended Kalman filter may diverge, the motor initially run with open loop for few seconds. The open loop algorithm is makes the motor to rotate in certain known frequency by generating ramp like rotor position in the DSP processor as shown in Figure 4.11.

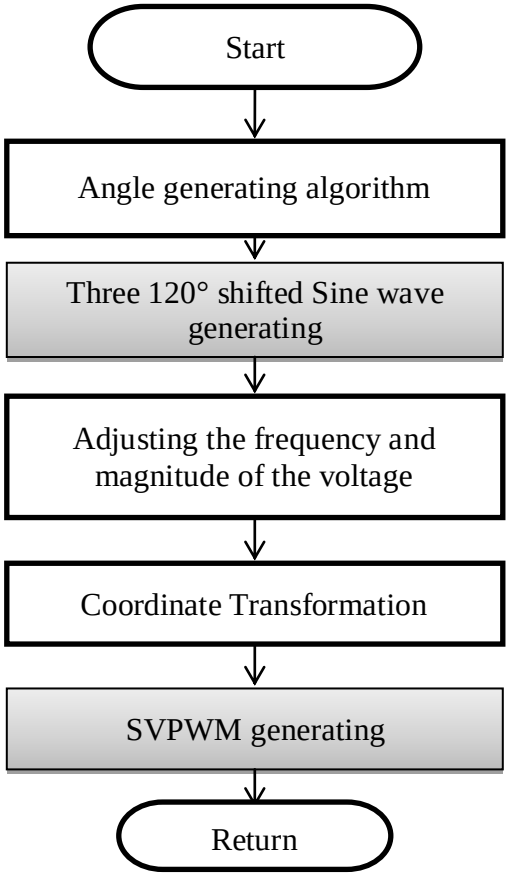


Figure 4.11: Open loop algorithm

The observer algorithm block describes the EKF algorithm for estimation of angular speed and position of the rotor while the motor is running. The algorithm is directly derived from the mathematical relations of prediction and estimation of EKF shown in the previous chapter. It is a recursive process computed periodically.

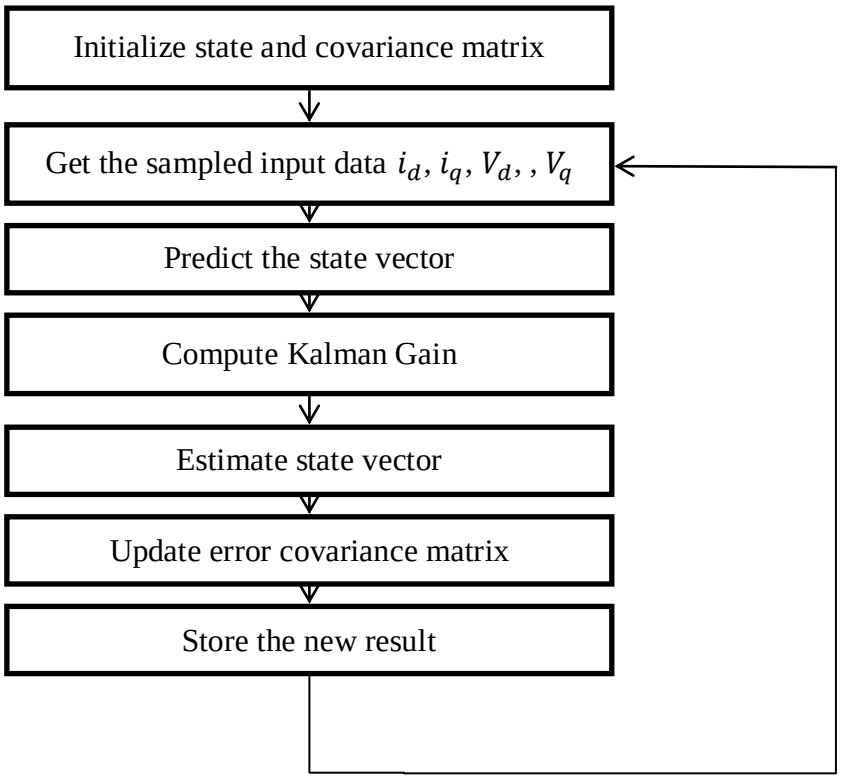


Figure 4.12: Extended Kalman Filter algorithm

The SVPWM algorithm describes the technique of space vector pulse width modulation described previously. The algorithm takes voltage outputs of the current controllers and it converts these voltages to d-q reference frame. After coordinate transformation, sector judgment is done. It is online determination of resultant space vector and in which sector it is found for appropriate operation-time calculation. From this operation time duty cycle is generated. Finally, this newly generated duty cycle is stored in Compare Registers (TxCMPR) of the DSP for appropriate pulse width modulation.

The entire interrupt sub-module algorithm is executed with in one PWM period which is equal to twice of Timer Period Register (TxPR). Hence the total execution time of the module should adjust with PWM period.

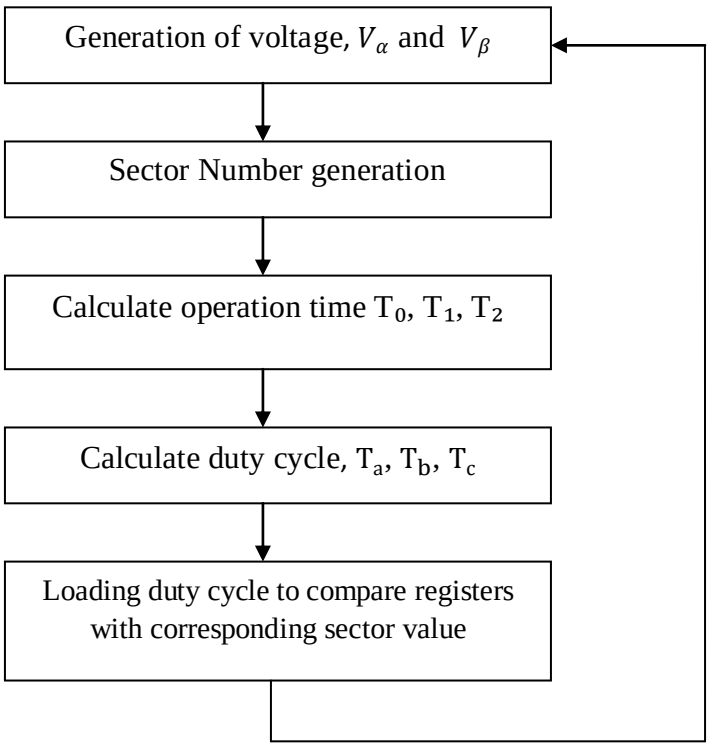


Figure 4.13: SPWM technique algorithm

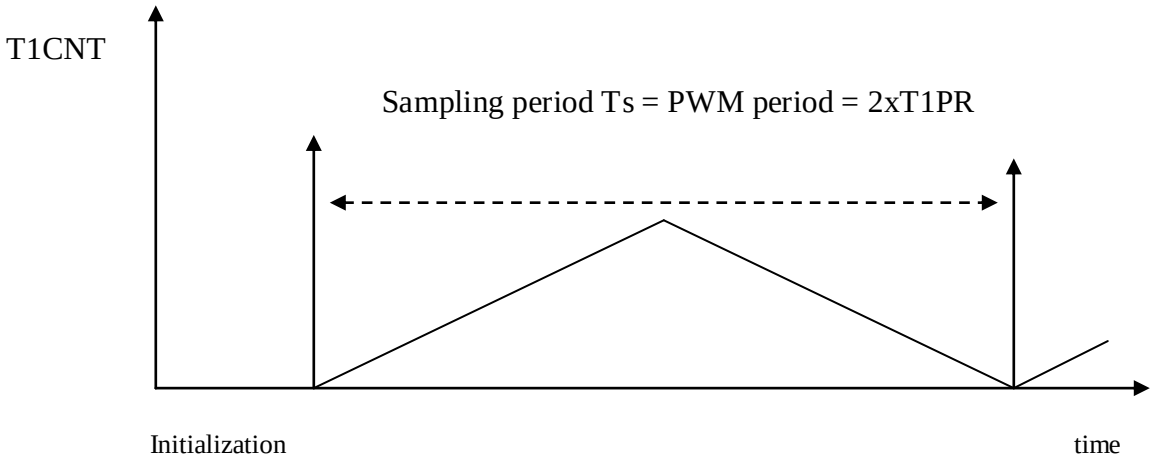


Figure 4.14: Software flow timing description

There are different counting modes of Timer counter (TxCN) of the DSP. For SVPWM, the DSP Timer counting mode should be at continuous up/down counting. The reloading condition of the Compare Registers is determined at the initialization stage. It may be Underflow- beginning of counting, Timer period-when counter is equal to period of the timer and Overflow-when counter

equals 0xFFFF. For this problem, the reloading condition of Compare Registers or updating a new value to these registers for determining the active phase size of PWM is Under Flow.

The length of the active phase (output pulse width) in Up/Down counting mode for one PWM period is given by:

$$(\text{TXPR}) - (\text{TxCMPR})_{up} + (\text{TxPR}) - (\text{TxCMPR})_{down}$$

TxPR is General Purpose Timer period value, $(\text{TxCMPR})_{up}$ is compare Register value during counting up of the counter, $(\text{TxCMPR})_{down}$ is compare Register value during counting down.

When the value in the TxCMPR is zero, the General Purpose timer compare output is active for the whole period and when the TxCMPR is equal or greater than TxPR, the General Purpose Timer compare output is inactive for the whole period. Hence updating the Compare Register values (TxCMPR) to their appropriate value to control the width of PWM active signal implies indirectly controlling the amount of voltage and its frequency applied on the stator winding. This also means controlling the speed of the rotor.

4.4 Experimental Setup

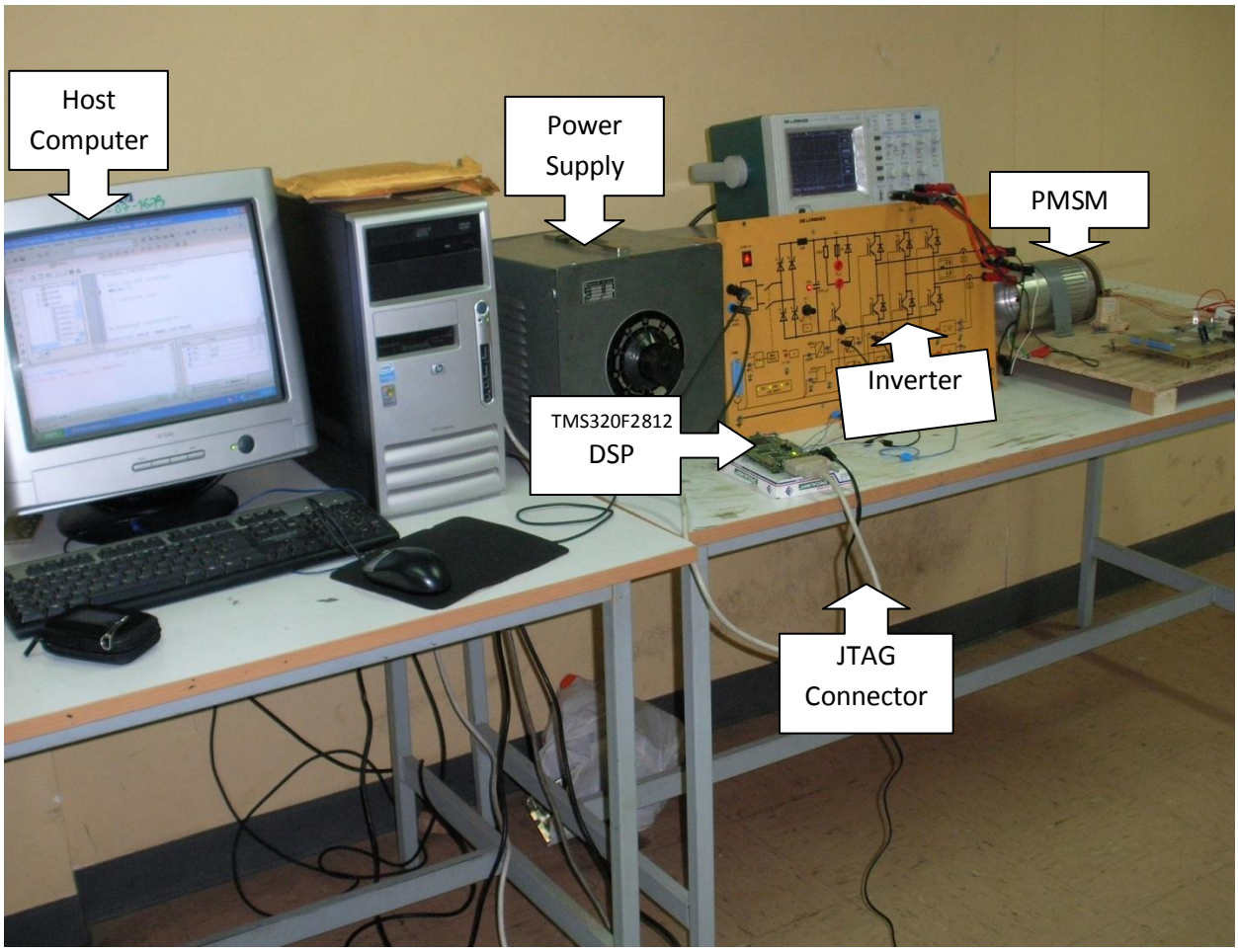


Figure 4.15: Experimental setup while the motor was running in open loop condition

The experimental setup includes the host computer which is used to program the system algorithm through code composer studio v3.1. The DSP is connected with the host computer through JTAG connector. The source code on the host computer is debugged and loaded to the memory of the DSP through this cable. Other DSP - host computer communication also takes place through this cable.

4.5 Expermental Results

The main target in the expermental investigation is genarating the appropraite PWM signal which genarates the appropraite sinusoidal voltage on the stator of the motor throught the DSP. This makes the motor to rotate at the requiered speed.

Figure 4.16: sample SVPWM output from the DSP

In the experimental investigation, open loop operation of the system is done using TMS320F2812 DSP. The open loop investigation is done by using a ramp angle generation technique and feeding this generated angle value to the system algorithm as a rotor position as shown in Figure 4.17. For this experimental investigation the angle is generated with 10Hz frequency. Using frequency speed relation of the motor which is $\omega_m = {}^{2\Pi f}/_{p}$, the motor rotates around 300 rpm as shown in the Figure 4.18.

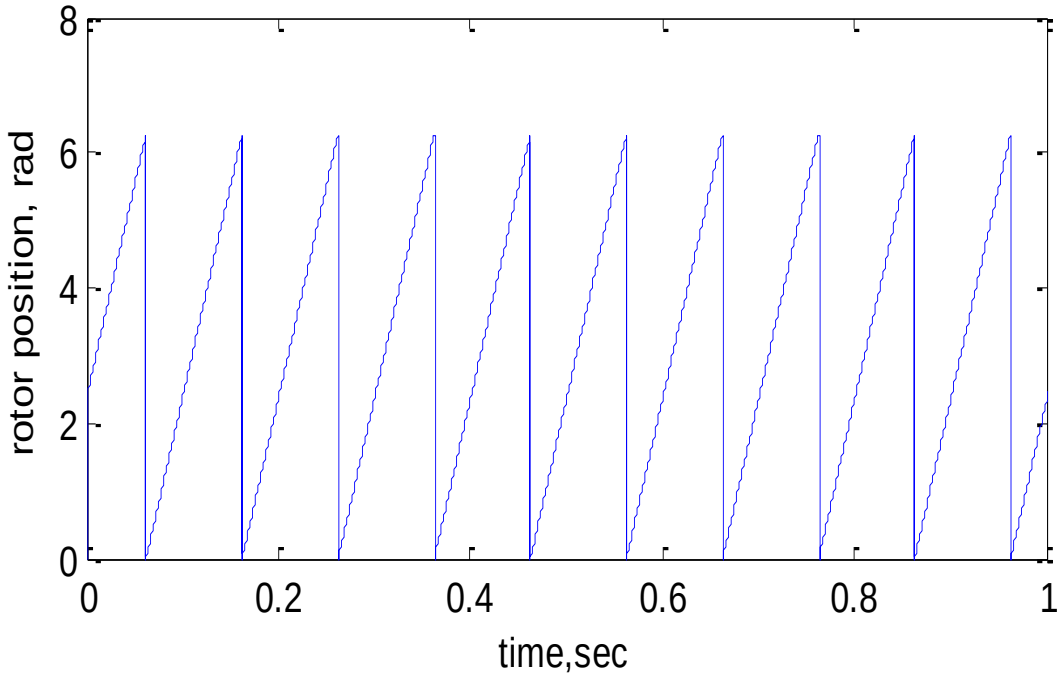


Figure 4.17: Rotor position while the motor is running in open loop

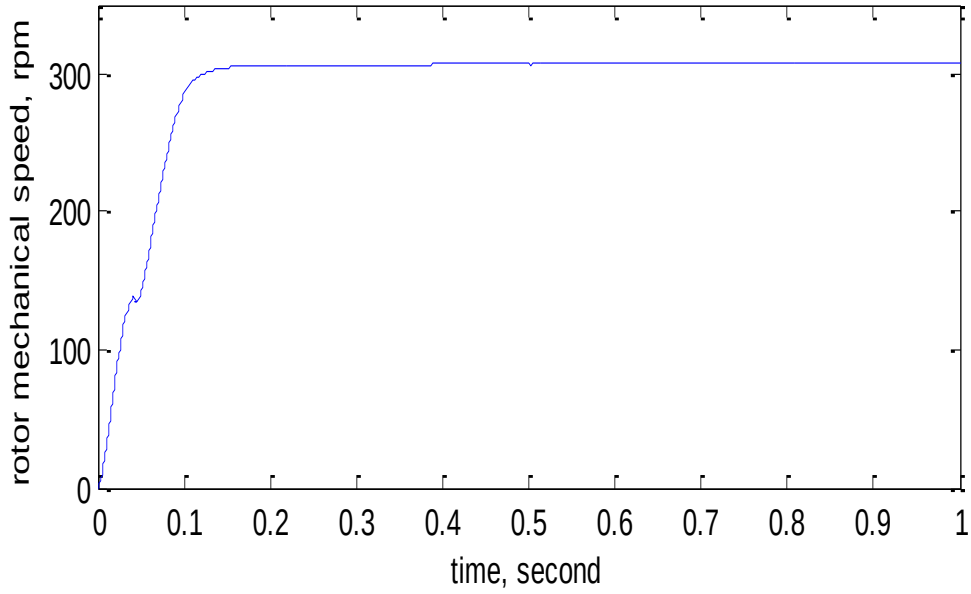


Figure 4.18: Rotor mechanical speed in rpm while the motor is running in open loop

Chapter Five

Conclusion and Recommendation

5.1 Conclusion

In this paper, design and implementation of Extended Kalman Filter to alleviate the problem due to mechanical position sensor for speed control of PMSM has been investigated. The computational techniques used to simplify the filter equation and their implementation in fixed point arithmetic is discussed. The filter was tuned by varying the parameters Q, R, P_0, X_0 to meet the desired transient and steady state performance. The discrete Extended Kalman Filter has been found to be well suited to the speed and rotor position estimation of a PMSM. The simulation results show Extended Kalman Filter can estimate the rotor position and speed with good performance for speed control of PMSM. As it is shown from Figure 4.6, EKF can estimate the actual speed of the motor with a steady state error of 0.067% with a good transient response time of less than 0.2 sec. The filter also estimates the actual rotor position with an average steady state error of 0.35%. The system also gives good performance at no-load (0Nm load torque condition) and loaded condition (5Nm load torque). Hence it can work with different load torque condition. The open loop experimental investigation also shows the overall system can be implemented using fixed point digital signal processor.

5.2 Recommendation

To complete the closed loop system control of the problems and improve the system performance the following can be done:

- If high performance initial rotor estimation technique is applied or good performance of open loop motor starting is applied for switching to closed loop.
- If the appropriate parameter identification is done for the motor for appropriate operation of the closed loop.
- If intelligent estimation and control technique is applied, the system performance can be improved.

References

[1] Alphonsa Roshin Paul, Prof May George, Brushless DC motor control using Digital PWM Technoques, Rajiv Gandhi Institute technology, Kottazam, Kerela.

[2] K.H.kim, C.Baik, K.chung, J.youn, Robust speed control of brushless DC motor using adaptive input-output linearization technique,. IEEE proc. Electr. Power app. Vol 144, November 1997.

[3] Erwan Simon, Implementation of speed field oriented control of three phase PMSM using TMS320F240, Digital control system, Texas Instrument, application report SPRA588, September, 1999.

[4] Mihia Cheles, sensorless field oriented control (FOC) for a permanent magnet synchronous motor (PMSM) using a PLL estimator and Field Weakening (FW), Microchip Technology Inc.AN1292, 2009.

[5] Speed sensorless Direct torque control of a PMSM drive using space vector modulation based MRAS and stator resistance estimator, A.Amuer, B.Mokhtari, N.Essounbouli, L.Mokrani, world academy of Science, Engineering and Technology66, 2012.

[6] Fredrik Petersson, Sensorless control of a Permanent Magnet Synchronous Motor, department of electrical engineering Linkopings University, 2009.

[7] Abraham Belay, DSP based vector control of induction motor, department of Electrical and Computer Engineering, Addis Ababa University, Ethiopia, September 2007.

[8] Salih Baris Ozturk, modeling, Simulation and Analysis of low cost direct torque control of PMSM using Hall Effect sensor, Texas A &M University, December 2005.

[9] Dalila Zaltni and Mohamed Naceur Abdlkrim, Permanent Magnet Synchronous Motor Observablity Analysis for sensor less control Design, J. Automation and System Engineering

[10] Minarech peter, Makys Pavol and Vittek Jan, PI controllers determination for Vector Control motion.

[11] E.Prasad, B.Suresh and K.Raghuveer , Field Oreinted control of PMSM using SVPWM technique, Global journal of Advanced Engineering Technologies, Vol1, Issue 2, 2012.

[12] E.Prasad, B.Suresh, K.Raghuveer, Field Oreinted Control of PMSM Using SVPWM Techniques, Global Journal of Advanced Engineering Technologies, Vol. 1, issue 2-2012.

[13] A.Ramasamy, V.Kumaran, R.Krishina Iyer and L.Zahan liu, Control of Dynamic Voltage Restorer using TMS320F2812, power quality research group, Electrical Engineering, University Tenaga Nasional, Malaysia; October 2007.

[14] Hu Xuezhi and Nan Guangqun, Research of vector variable frequency system based on TMS320F2812, Huangshi Institute of technology, 435003, P.R china , International conference on intelligent computation technology and automation, 2008.

[15] Jorge Zambada, Sensorless Field Oriented control of PMSM motors, Microchip Technology Inc. AN1078, 2007.

[16] Zheng-Guang Wang, Jian-Xun Jin You-Guang Guo and Jian-Guo Zuo, SVPWM Techniques and Application in HTS PMSM machines control, Journal of electrical science and technology of China vol.6, No.2, June 2008.

[17] David Vindel Munzo, Design, Simulation and Implementation of a PMSM Drive System, Department of Energy and Environment, Chamlers University Technology, Sweden, 2011.

[18] Andria G.M. Cilio and Henk Corporaal, Floating point to Fixed Point conversion of C code, Delft University of technology, computer Architecture and digital techniques Dept, Netherland.

[19] Siva Balaji T, Vargil Kumar.E and Teja.T, Modeling and Performance of Permanent Magnet synchronous motor Drive with Space Vector Modulation, Gudlavalleru Engineering College, Gudlava-India, International Journal of Advanced Engineering science and Technologies Vol.9, issue No.2.

[20] Discrete State Space Model of Permanent Magnet Synchronous Motor with Voltage source PWM Inverter.

[21] Maria Isabel Ribeiro, Kalman and Extended Kalman Filters: Concept, Derivation and Properties, Institute for system and Robotics, February, 2004.

[22] Coordinate Transform, 32-Bit Microcontroller MB9B100A/MB9B300A/MB9B500A Series

[23] TMS320x281x DSP System control and Interrupts Reference Guide, Literature Number-SPRU078F, April 2002-revised May 2009.

[24] TMS320x281x DSP Analog-to-digital converter (ADC) Reference Guide, Literature Number-SPRU060D, June 2002-Revised July 2005.

[25] TMS320x281x DSP Event Manager (EV) Reference Guide, Literature Number-SPRU065E, November 2004-Revised June 2007.

[26] TMS320x281x DSP External Interface (XINTF) reference Guide, Literature Number-SPRU067C, May 2002-Revised November 2004.

[27] eZdsp™ F2812 Technical Reference, Spectrum Digital Inc.506265 Rev.F September 2003.

[28] Pavel Grasblum, Sensorless BLDC Motor Control Using MC9S08AW60, Freescale Czech Systems Center, September, 2007.

7. Appendices

Appendix A: Simplified Actual Motor Parameter identification

The Permanent magnet synchronous motor in the control and Drive Lab has not parameter since it is a prototype motor. Hence, parameter identification of the motor should be done for closed loop control.

Resistance and Inductance:

The resistance and inductance of the motor is calculated by direct line-to-line measurement of the motor using inductance meter and multi-meter. The calculation is done by making different measurement and taking the average value.

Table 7.1: different measurement of line-to-line resistance of the motor

	L1-L2(Ω)	L1-L3(Ω)	L2-L3(Ω)	Average
Test1	0.45	0.35	0.35	0.35
Test2	0.40	0.35	0.40	0.35
Test3	0.35	0.40	0.35	0.35

If the motor is delta connected the Resistance of the motor becomes considering all phases are equal:

R=0.35Ω

If the motor is star connected

$R= {}^{0.35}/_{2} = 0.175\Omega$

Table 7.2: different measurement of line-to-line inductance of the motor

	L1-L2(mH)	L1-L3(mH)	L2-L3(mH)	Average
Test1	0.694	0.673	0.677	0.6813
Test2	0.695	0.673	0.674	0.68067
Test3	0.693	0.674	0.675	0.68067

If the motor is delta connected the Inductance of the motor becomes considering all phases are equal:

L=0.68088mH

If the motor is star connected

L= $0.68088/_2$ = 0.34044mH

Number of Pole Pair (P):

The number of pole pair is calculated by running the motor in open loop with a known frequency of 10Hz and measuring the actual mechanical speed (ωm) of the motor using taco meter. The motor runs around 300rpm with this input voltage frequency.

$$\omega m = \frac{2\pi f}{P}, \; P = \frac{2\pi f}{\omega m} = \frac{2\pi * 10}{300 * 2\pi/60} = 2$$

Flux Linkage (λm):

The applied voltage and induced voltage relation for the motor is:

$E_f = V - IR$

At steady state the effect of stator resistance is neglected and the flux produced by the stator in total is equal to the flux produced by the permanent magnets. Hence the relation becomes:

$$E_f = E_{pm} = \frac{2\pi f s \phi m N s K \omega s}{\sqrt{2}} = 2\pi f_s \lambda_m$$

$E_f = V = 2\pi f_s \lambda_m$

$$\lambda_m = \frac{V}{2\pi f s} = \frac{40}{2\pi * 10} = 0.647$$

Moment of Inertia (J):

The moment of inertia can be calculated from the torque-acceleration relation. But in my case the motor is rotated in steady state condition. Hence it is necessary to know the motor condition at starting to calculate its change of speed.

$\frac{d\omega}{dt} = \frac{1}{J}$ (Te-T_L)

If we consider no-load condition of the motor

$\frac{d\omega}{dt} = \frac{1}{J}$ Te

J= Te/$\frac{d\omega}{dt}$

Friction Coefficient (F): this parameter also difficult to investigate in open loop condition

Appendix B: C- programming source code for DSP implementation

Main source code:

```
// Project : Obsrver based speed control of PMSM using Texas Instrument TMS320F2812
// Author : Hamdihun Abdie
// Year : 2011/2012
// Faculity : Addis Ababa institute of technology _AAiT, Ethiopia
// Stream : Control Engineering
//###########################################################################
//   The project sets up the PLL in x10/2 mode, divides SYSCLKOUT
//   by ten to reach a 15Mhz HSPCLK (assuming a 30Mhz XCLKIN). The
//   clock divider in the ADC is not used so that the ADC will see
//   the 15Mhz on the HSPCLK. Interrupts are enabled and the EVA
//   is setup to generate a periodic ADC SOC on SEQ1. two channels
//   are converted, ADCINA2, ADCINA1.
//###########################################################################
#include "DSP281x_Device.h"     // DSP281x Headerfile Include File
#include "DSP281x_Examples.h"   // DSP281x Examples Include File
#include "IQmathLib.h"
#include "parametrs.h"
```

```
#include "stdio.h"
#include "math.h"
/*prototype function for ISR */
interrupt void svpwm_isr(void);
/* Coordinate Transformation function   */
// Clark Transformation
_iq i_alpha(_iq iaa,_iq ibb,_iq icc )
    {
    return (iaa-_IQmpy(_IQ(0.5),(ibb+icc)));
    }
_iq i_beta(_iq iaa,_iq ibb,_iq icc )
    {
    return( _IQmpy(_IQdiv(_IQsqrt(3.0),2),(ibb-icc)) );
    }
// Park Transformation
_iq i_d(_iq ialph,_iq ibet,_iq thet)
    {
    return (_IQmpy(ialph,_IQcos(thet))+_IQmpy(ibet,_IQsin(thet)));
    }
_iq i_q(_iq ialph,_iq ibet,_iq thet )
    {
    return (_IQmpy(-ialph,_IQsin(thet))+_IQmpy(ibet,_IQcos(thet)));
    }
// Inverse Park Transformation
_iq V_alpha(_iq vdd,_iq vqq,_iq theta )
    {
    return (_IQmpy(vdd,_IQcos(theta))- _IQmpy(vqq,_IQsin(theta)));
    }
_iq V_beta(_iq vdd,_iq vqq,_iq thet )
    {
    return (_IQmpy(vdd,_IQsin(thet))+_IQmpy(vqq,_IQcos(thet)));
    }
```

```
/*PI controller function */
_iq PI_controller (_iq Kpp,_iq Kii, _iq ref_value,_iq meas_value)
    {
     err_k = ref_value - meas_value;
     Uk = Uk_1 + _IQmpy((Kpp + _IQmpy(Kii,Ts)) , err_k) + _IQmpy(Kpp , err_k_1);
   /* Limitation of the actuating signal for integral windup */
     if (Uk > Umax)
                Uk = Umax;
              if (Uk < Umin)
                Uk = Umin;
    /* Storing previous errors and actuating result */
     Uk_1    = Uk;
     err_k_1 = err_k;
     return (Uk);
    }
int v=0;
_iq Um = _IQ(15);
_iq Ux;
_iq Uy;
_iq Uz;
_iq T = _IQ(0.2);
main()
{
// Initialize System Control:
// PLL, WatchDog, enable Peripheral Clocks
  InitSysCtrl();
// For this project, set HSPCLK to SYSCLKOUT / 10 (15Mhz assuming 150Mhz SYSCLKOUT)
  EALLOW;
  SysCtrlRegs.HISPCP.all = 0x5;  // HSPCLK = SYSCLKOUT/10
//  Initialize GPAMUX:
  // Enable PWM pins
  GpioMuxRegs.GPAMUX.all = 0x00FF; // EVA PWM 1-6  pins
```

```
   GpioMuxRegs.GPBMUX.bit.PWM7_GPIOB0 = 0;
   GpioMuxRegs.GPBDIR.bit.GPIOB0 = 0;
   EDIS;
// Clear all interrupts and initialize PIE vector table:
// Disable CPU interrupts
   DINT;
// Initialize the PIE control registers to their default state.
// The default state is all PIE interrupts disabled and flags
// are cleared.
   InitPieCtrl();
// Disable CPU interrupts and clear all CPU interrupt flags:
   IER = 0x0000;
   IFR = 0x0000;
// Initialize the PIE vector table with pointers to the shell Interrupt
// Service Routines (ISR).
   InitPieVectTable();
// Interrupts that are used in this project are re-mapped to
// ISR functions found within this file.
   EALLOW;  // This is needed to write to EALLOW protected register
   PieVectTable.T1UFINT = &svpwm_isr;
   EDIS;    // This is needed to disable write to EALLOW protected registers
//  Initialize the Device Peripherals:
   InitAdc();  // For this project, init only the ADC
//  User specific code, enable interrupts:
// Enable ADCINT in PIE
   PieCtrlRegs.PIEIER2.bit.INTx6 = 1;
   IER |= M_INT2; // Enable CPU Interrupt 1
   EINT;          // Enable Global interrupt INTM
   ERTM;          // Enable Global realtime interrupt DBGM
// Configure ADC for the project
   AdcRegs.ADCTRL1.bit.CONT_RUN=1;
   AdcRegs.ADCTRL3.bit.SMODE_SEL=1;       // simultaneous mode conversion
```

```
   AdcRegs.ADCMAXCONV.all = 0x0000;       // Setup 2 conv's on SEQ1
   AdcRegs.ADCCHSELSEQ1.bit.CONV00 = 0x0; // Setup ADCINA0 and ADCINB0 as SEQ1
  // AdcRegs.ADCCHSELSEQ1.bit.CONV01 = 0x1; // Setup ADCINA2 as 2nd SEQ1 conv.
   AdcRegs.ADCTRL2.bit.EVA_SOC_SEQ1 = 1;  // Enable EVASOC to start SEQ1
//   AdcRegs.ADCTRL2.bit.INT_ENA_SEQ1 = 1;  // Enable SEQ1 interrupt (every EOS)
// Configure EVA for SVPWM
   EvaRegs.T1PR = 0x1D4C; // 7500 period register as the required 1KHz PWM frequency
   EvaRegs.GPTCONA.bit.T1TOADC = 1; // Enable EVASOC in EVA on timer counter under //flow condition
   EvaRegs.EVAIMRA.bit.T1UFINT = 1;       // Enable EVA timer 1 unred flow interrupt
   EvaRegs.T1CON.all = 0x0842;        // Enable timer 1 compare (up/Down count mode),TPS=1
   EvaRegs.T1CNT = 0x0000;               // Initialize counter register to start at 0x0000
// Enable compare for PWM1-PWM6
  EvaRegs.CMPR1 = 0x0C00;
  EvaRegs.CMPR2 = 0x3C00;
  EvaRegs.CMPR3 = 0xFC00;
// Compare action control.  Action that takes place on a cmpare event
  // output pin 1 CMPR1 - active high    output pin 2 CMPR1 - active low
  // output pin 3 CMPR2 - active high     output pin 4 CMPR2 - active low
  // output pin 5 CMPR3 - active high      output pin 6 CMPR3 - active low
  EvaRegs.ACTRA.all   = 0x0666;
  EvaRegs.DBTCONA.all = 0x0000; // Disable deadband
  EvaRegs.COMCONA.all = 0x8200;// SVPWM disabled
// Wait for ADC interrupt
   while(1)
   {}
}
/* Interrupt subroutine */
Interrupt void  svpwm_isr(void)
{
/********************************** reading voltage values from ADC*****************************************/
if( v>100000)// for 10 sec
{
```

```
if(theta > _IQmpy(_IQ(2),pi))

{theta=0;}

else

{theta=theta+_IQdiv(_IQmpy(_IQ(2),pi),200);}

v=v+1;

Ux=_IQmpy(Um,_IQsin(theta));

Uy=_IQmpy(Um,_IQsin(theta+_IQmpy(pi,_IQ(0.6666667))));

Uz=_IQmpy(Um,_IQsin(theta+_IQmpy(pi,_IQ(1.3333333))));

/*Park transformation of actuating voltage*/

  Valpha = i_alpha(Ux,Uy,Uz);

  Vbeta  = i_beta(Ux,Uy,Uz);

}

else

{

// changing the ADC result register value from 16 bit to 32 bit fixed value

ia=_IQmpy(_IQ(3.0),_IQ16toIQ((_iq)(AdcRegs.ADCRESULT0>>4)));

ib=_IQmpy(_IQ(3.0),_IQ16toIQ((_iq)(AdcRegs.ADCRESULT1>>4)));

/* input current conditioning ; i = (i-1.5)*K*V/Rm where K = voltage divider ratio ,Rm = sensor terminal resistor */

ia = ((ia - _IQ(1.5))*K)/Rm;

ib = ((ib - _IQ(1.5))*K)/Rm;

/* calculating phase c current , ia+ib+ic=0 */

ic =  (-ia-ib);

/*************************************Coordinate Transformation ****************************************/

ialpha = i_alpha(ia,ib,ic);

ibeta  = i_beta(ia,ib,ic);

id     = i_d(ialpha,ibeta,theta);

iq     = i_q(ialpha,ibeta,theta);

/************************************ OBSERVER DESIGN ****************************************/

/*Assigning  element values of Fk */

Fk[0][0]=_IQ(1)-_IQmpy(_IQdiv(Rs,L),Ts); Fk[0][1]=_IQmpy(Ts,Xk[2]);              Fk[0][2]=_IQmpy(Ts,iq);
Fk[0][3]=0;

Fk[1][0]=_IQmpy(Ts,-Xk[2]);             Fk[1][1]=_IQ(1)-_IQmpy(_IQdiv(Rs,L),Ts); Fk[1][2]=_IQmpy(Ts,(-id-_IQdiv(Am,L)));
Fk[1][3]=0;
```

```
Fk[2][0]=0;                Fk[2][1]=_IQmpy(_IQdiv(_IQmpy(_IQmpy(_IQmpy(P,P),Am),_IQ(1.5)),J),Ts);        Fk[2][2]=
_IQ(1);Fk[2][3]=0;

Fk[3][0]=0;                Fk[3][1]=0;                Fk[3][2]=Ts;                Fk[3][3]=_IQ(1);

/*Assigning element values of Xk_k or prediction of it */

Xk_k[0] = _IQmpy((_IQ(1)-_IQmpy(_IQdiv(Rs,L),Ts)),Xk[0])+_IQmpy(Xk[1],_IQmpy(Xk[2],Ts))+_IQmpy(_IQdiv(Vd,L),Ts);

Xk_k[1] = _IQmpy((_IQ(1)-_IQmpy(_IQdiv(Rs,L),Ts)),Xk[1])-
_IQmpy(_IQmpy(Ts,(Xk[0]+_IQdiv(Am,L))),Xk[2])+_IQmpy(_IQdiv(Vq,L),Ts);

Xk_k[2] = _IQmpy(_IQdiv(_IQmpy(_IQmpy(_IQmpy(P,P),Am),_IQ(1.5)),J),Ts) + Xk[3];

Xk_k[3] = _IQmpy(Xk[2],Ts) + Xk[3];

/* Measured output */

Zk[0]=id;

Zk[1]=iq;

/* estimated output */

hk_k[0]=Xk[0];

hk_k[1]=Xk[1];

/* Transpose of Fk   */

for (i=0;i<4;i++)
   {
   for (j=0;j<4;j++)
      {
      Fk_T[i][j]=Fk[j][i];
      }}

/*  Transpose of Hk  */

for (i=0;i<4;i++)
   {
   for (j=0;j<2;j++)
      {
      Hk_T[i][j]=Hk[j][i];
      }}

/* Prediction of state error covariance

  Pk_k = Fk * Pk * Fk' + Q          */

for (i=0;i<4;i++)
   {
```

```
    for (j=0;j<4;j++)
        {
        for (k=0;k<4;k++)
          {
           c1[i][j]=c1[i][j]+_IQmpy(Fk[i][k],Pk[k][j]);
          } }}
for (i=0;i<4;i++)
    {
    for (j=0;j<4;j++)
        {
        for (k=0;k<4;k++)
          {
           c2[i][j]=c2[i][j]+_IQmpy(c1[i][k],Fk_T[k][j]);
          }}}
for (i=0;i<4;i++)
    {
    for (j=0;j<4;j++)
        {
            Pk_k[i][j]=c2[i][j]+Q[i][j];
         }}
/*Kalman gain calculation
   K = Pk_k * H' * inv(H * Pk_k * H' +R) */
for (i=0;i<4;i++)
    {
    for (j=0;j<2;j++)
        {
        for (k=0;k<4;k++)
          {
           c3[i][j]=c3[i][j]+_IQmpy(Pk_k[i][k],Hk_T[k][j]);
          }}}
for (i=0;i<2;i++)
    {
```

```
   for (j=0;j<4;j++)
      {
      for (k=0;k<4;k++)
        {
         c4[i][j]=c4[i][j]+_IQmpy(Hk[i][k],Pk_k[k][j]);
        }}}
for (i=0;i<2;i++)
   {
   for (j=0;j<2;j++)
      {
      for (k=0;k<4;k++)
        {
         c5[i][j]=c5[i][j]+_IQmpy(c4[i][k],Hk_T[k][j]);
        }}}
for (i=0;i<2;i++)
   {
   for (j=0;j<2;j++)
      {
         c6[i][j]=_IQmpy(c5[i][j],R[i][j]);
       }}
det=_IQmpy(c6[0][0],c6[1][1])-_IQmpy(c6[1][0],c6[0][1]);
m1=c6[1][1];
m2=c6[0][1];
m3=c6[1][0];
m4=c6[0][0];
c66[0][0]= _IQdiv( m1,det);
c66[0][1]= _IQdiv((-m2),det);
c66[1][0]= _IQdiv((-m3),det);
c66[1][1]= _IQdiv(m4,det);
for (i=0;i<4;i++)
   {
   for (j=0;j<2;j++)
```

```
        {
        for (k=0;k<2;k++)
          {
           Kk[i][j]=Kk[i][j]+_IQmpy(c3[i][k],c66[k][j]);
          }}}
/*State error covariance matrix estimation
   Pk = Pk_k * (I-Kk * Hk)              */
for (i=0;i<4;i++)
   {
   for (j=0;j<4;j++)
      {
      for (k=0;k<2;k++)
        {
         c7[i][j]=c7[i][j]+_IQmpy(Kk[i][k],Hk[k][j]);
        }}}
for (i=0;i<4;i++)
   {
   for (j=0;j<4;j++)
      {
       c8[i][j]=I[i][j]-c7[i][j];
       }}
for (i=0;i<4;i++)
   {
   for (j=0;j<4;j++)
      {
      for (k=0;k<4;k++)
        {
         Pk[i][j]=Pk[i][j]+_IQmpy(c8[i][k],Pk_k[k][j]);
        }}}
/*   State estimation
  Xk = Xk_k + Kk * (Zk-hk_k)   */
for(i=0;i<2;i++)
```

```
  {
   c9[i]=Zk[i]-hk_k[i];
  }
//int i,j,k;
for (i=0;i<4;i++)
  {
  for (j=0;j<1;j++)
     {
     for (k=0;k<2;k++)
       {
        c10[i]=c10[i]+_IQmpy(Kk[i][k],c9[k]);
       }}}
for (i=0;i<4;i++)
  {
   Xk[i]=Xk_k[i]+c10[i];
  }
/* Accessing speed and Position */
speed = Xk[2];
theta = Xk[3];
/* only for graphing speed vs time purpose */
if(count < 500){spd[count] = speed;}
count=1 + count;
/****************************************** CONTROLLER DESIGN ************************************/
/*speed controller*/
i_q_ref = PI_controller(kp_speed,ki_speed,speed_ref,speed);
/*q _axis current controller*/
Vd = PI_controller(kp_q,ki_q,i_q_ref,iq);
/*d_axis current controller*/
Vq = PI_controller(kp_d,ki_d,i_d_ref,id);
/************************* SPASE VECTOR PULSE WIDTH MODULATION _SVPWM************************/
/*Park transformation of actuating voltage*/
 Valpha = V_alpha(Vd,Vq,theta);
```

```
 Vbeta  = V_beta(Vd,Vq,theta);
}
if (Vbeta > 0)
{A=1;}
else {A=0;}
if ((_IQmpy(_IQsqrt(3),Valpha)-Vbeta)>0)
{B=1;}
else{B=0;}
if ((_IQmpy(_IQsqrt(3),Valpha)+Vbeta)<0)
{C=1;}
else{C=0;}
N=A + 2 * B + 4 * C;
z=_IQdiv(_IQmpy(Ts,(_IQmpy(_IQ(-1.73205),Valpha)+Vbeta)),_IQmpy(_IQ(1.41421),Vdc));
y=_IQdiv(_IQmpy(Ts,(_IQmpy(_IQ(1.73205),Valpha)+Vbeta)),_IQmpy(_IQ(1.41421),Vdc));
x=_IQdiv(_IQmpy(_IQmpy(Ts,Vbeta),_IQ(2)),_IQmpy(_IQ(1.41421),Vdc));
if (N==1)
 {T1 = z;
 T2 = y;}
 else if (N==2)
 {T1 = y;
 T2 = -x;}
 else if (N==3)
 {T1 = -z;
 T2 = x;}
 else if (N==4)
 {T1 = -x;
 T2 = z;}
 else if (N==5)
 {T1 = x;
 T2 = -y;}
 else if (N==6)
 {T1 = -y;
```

```
 T2 = -z;}
 // Negative duty cycle protection
if ((T1 + T2) > Ts)
{T1=_IQmpy(T1,_IQdiv(Ts,(T1+T2)));
T2=_IQmpy(T2,_IQdiv(Ts,(T1+T2)));}
T0=Ts-T1-T2;
Ta=_IQdiv(T0,_IQ(4));
Tb=Ta+_IQdiv(T1,_IQ(2));
Tb=Tb+_IQdiv(T2,_IQ(2));
// Duty cycle calculation
Ta=_IQdiv(Ta,Ts);
Tb=_IQdiv(Tb,Ts);
Tc=_IQdiv(Tc,Ts);
// change duty cycle time in Secs to number of clock cycle for TPS=128 & HISPCP=10
/* Fixed to float conversion */
Taa = _IQtoF( _IQmpy((_IQ(7500)),(1-Ta)));
Tbb = _IQtoF( _IQmpy((_IQ(7500)),(1-Tb)));
Tcc = _IQtoF( _IQmpy((_IQ(7500)),(1-Tc)));
/* Loading the updated duty cycle interms of operation time to the compare registers for different sector condition*/
if (N==1)
  { EvaRegs.CMPR1=Tbb;
  EvaRegs.CMPR2=Taa;
  EvaRegs.CMPR3=Tcc;}
else if (N==2)
  {EvaRegs.CMPR1=Taa;
  EvaRegs.CMPR2=Tcc;
  EvaRegs.CMPR3=Tbb;}
else if (N==3)
  {EvaRegs.CMPR1=Taa;
  EvaRegs.CMPR2=Tbb;
  EvaRegs.CMPR3=Tcc;}
else if (N==4)
```

```
  {EvaRegs.CMPR1=Tcc;
  EvaRegs.CMPR2=Tbb;
  EvaRegs.CMPR3=Taa;}
else if (N==5)
  {EvaRegs.CMPR1=Tcc;
  EvaRegs.CMPR2=Taa;
  EvaRegs.CMPR3=Tbb;}
else if (N==6)
  {EvaRegs.CMPR1=Tbb;
  EvaRegs.CMPR2=Tcc;
  EvaRegs.CMPR3=Taa;}
/********** **********Reinitialize registers for next interrupt sequence **********/
  EvaRegs.EVAIFRA.bit.T1UFINT=1;          // clear the interupt flag
  PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;   // Acknowledge interrupt to PIE
  return;
}
/********** **********/
```

System parameter declaration and initialization

```
/* Header file for the declaration and initialization of parametrs included in the motor control system*/
#include"IQmathLib.h"
#ifndef PARAMETRS_H
#define PARAMETRS_H
/*Parametrs for SVPWM */
int A, B, C;
int n;
_iq z;
_iq y;
_iq x;
_iq Vdc=_IQ(150);
_iq pi =_IQ(3.14159);
_iq T1=0;
_iq T2=0;
```

```
_iq T0=0;
int  N;
_iq alpha;
_iq Ts = _IQ(0.0001);// period in second for 1 KHz frequency of SVPWM=1ms period
_iq Ta;
_iq Tb;
_iq Tc;
_iq Valpha;
_iq Vbeta;
/*parametrs for EKF*/
_iq Xk[4]    =  {0,0,0,0};
_iq Pk[4][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}};
_iq I[4][4]  = {{_IQ(1),0,0,0},{0,_IQ(1),0,0},{0,0,_IQ(1),0},{0,0,0,_IQ(1)}};
_iq Q[4][4]  = {{_IQ(0.23),0,0,0},{0,_IQ(0.023),0,0},{0,0,_IQ(0.063),0},{0,0,0,_IQ(0.0183)}};
_iq R[2][2]  = {{_IQ(10.1),0},{0,_IQ(10.1)}};
_iq Hk[2][4]  = {{_IQ(1),0,0,0},{0,_IQ(1),0,0}};
_iq Kk[4][4]  = {{0,0},{0,0},{0,0},{0,0}};
/*   Temporary array declaration and initialization */
_iq c1[4][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}};
_iq c2[4][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}};
_iq c3[4][2] = {{0,0},{0,0},{0,0},{0,0}};
_iq c4[2][4] = {{0,0,0,0},{0,0,0,0}};
_iq c5[2][2] = {{0,0},{0,0}};
_iq c6[2][2];
_iq c66[2][2];
_iq c7[4][4] = {{0,0,0,0},{0,0,0,0},{0,0,0,0},{0,0,0,0}};
_iq c8[4][4];
_iq c9[2];
_iq c10[4] = {0,0,0,0};
int i,j,k;
_iq det;
_iq m1 ,m2 ,m3, m4;
```

```
_iq speed;
_iq theta=0;
_iq theta_pre=0;
int t=0;
_iq speed_ref=_IQ(300);// reference speed
_iq i_d_ref=0;// d_axis reference current
_iq i_q_ref;// q_axis reference current
// PI_controller gain constants
_iq kp_d=_IQ(34.8);
_iq ki_d=_IQ(0.879);
_iq kp_q=_IQ(4.8);
_iq ki_q=_IQ(0.879);
_iq kp_speed=_IQ(12.466);
_iq ki_speed=_IQ(0.7963);
// three phase currents
_iq ia;
_iq ib;
_iq ic;
_iq ialpha;
_iq ibeta;
// Array declaration for extended kalman filter
_iq Fk[4][4],Hk_T[4][2],Fk_T[4][4],Pk_k[4][4],Zk[2],hk_k[2],Xk_k[4];
/* d & q _axis voltage and current*/
_iq Vd = 0;
_iq Vq = 0;
_iq id;
_iq iq;
_iq  err_k,err_k_1= 0;/* error value for PI_controller */
_iq  Uk,Uk_1      = 0; /* PI_controller output */
_iq Umax          = _IQ(1);
_iq Umin          = _IQ(-1);
float Taa;
```

```
float Tbb;
float Tcc;
/*The motor parameters*/
_iq Rs = _IQ(0.175); /*Motor stator winding resistance in ohom*/
_iq L  = _IQ(0.34044); /*Motor stator winding inductance in Henery */
_iq Am = _IQ(0.647); /*Rotor flux linkage in V.s*/
_iq P  = _IQ(2);     /*No of pole pairs*/
_iq J  = _IQ(0.06);  /* Rotor inertia JKg metresquare*/
_iq F  = _IQ(0.00001);  /* friction cofficient Nms*/
#endif
//*************************************************************************************************
```

Linker commands file source code

```
MEMORY
{
PAGE 0 :
   /* For this example, H0 is split between PAGE 0 and PAGE 1 */
   /* BEGIN is used for the "boot to HO" bootloader mode      */
   /* RESET is loaded with the reset vector only if           */
   /* the boot is from XINTF Zone 7.  Otherwise reset vector  */
   /* is fetched from boot ROM. See .reset section below      */
   RAMM0      : origin = 0x000000, length = 0x000400
   BEGIN      : origin = 0x3F8000, length = 0x000002
   PRAMH0     : origin = 0x3F8002, length = 0x0012B5
   RESET      : origin = 0x3FFFC0, length = 0x000002
PAGE 1 :
   /* For this example, H0 is split between PAGE 0 and PAGE 1 */
   RAMM1    : origin = 0x000400, length = 0x000400
  /*DRAMH0   : origin = 0x3F92B7, length = 0x001000*/
   DRAMH0   : origin = 0x3F92B7, length = 0x000D49
}
SECTIONS
{
```

```
   /* Setup for "boot to H0" mode:
      The codestart section (found in DSP28_CodeStartBranch.asm)
      re-directs execution to the start of user code.
      Place this section at the start of H0 */
   codestart       : > BEGIN,       PAGE = 0
   ramfuncs        : > PRAMH0       PAGE = 0
   .text           : > PRAMH0,      PAGE = 0
   .cinit          : > PRAMH0,      PAGE = 0
   .pinit          : > PRAMH0,      PAGE = 0
   .switch         : > RAMM0,       PAGE = 0
   .reset          : > RESET,       PAGE = 0, TYPE = DSECT /* not used, */
   .stack          : > RAMM1,       PAGE = 1
   .ebss           : > DRAMH0,      PAGE = 1
   .econst         : > DRAMH0,      PAGE = 1
   .esysmem        : > DRAMH0,      PAGE = 1
}
//***************************************The End**************************************************
```