UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ
À L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE ÉLECTRIQUE

PAR
ABDELRAHMAN YOUSIF ESHAG LESAN

COMMANDE VECTORIELLE SANS CAPTEUR PAR PROCESSEUR NUMÉRIQUE DE LA MACHINE ASYNCHRONE

MAI 2010

Abstract

The squirrel cage induction machine characterized by its simplicity of construction, robustness and low cost, is the workhorse of industry. However, the control of such machine is quite complex. This is due to the coupling between the torque and flux producing mechanisms. Recently, variable speed drives incorporating alternating current (AC) machines have been taking place over the direct current (DC) ones as a result of the progress in the field of power electronics and digital signal processing (DSP) technology where several control techniques emerged. These techniques, such as vector control, allow a precise and instantaneous torque and speed control of induction machines for high performance applications.

Generally, a speed sensor is used to measure the machine speed for vector control purposes. Speed sensors degrade system reliability particularly in hostile environments. In sensorless control technique, the speed sensor is replaced by a speed estimator, which eliminates the costs associated with the speed sensor and reduces the drive complexity.

The purpose of the present research work is to design and implement an induction machine sensorless vector control using a digital signal processor and to evaluate its performance. A digital filter is used to filter motor currents. The filter implementation reduces the drive cost since it is a part of the vector control algorithm.

To achieve the research objectives, modelling of the drive main components are carried out, basically the induction motor and the inverter. Both, steady state and dynamic models of the induction motor are discussed. Matlab-Simulink software is used to simulate a variable speed drive using the direct vector and sensorless control techniques. At the end, a typical electric drive using eZdspF2812 evaluation board is developed.

The simulation and experimental results obtained are good enough to say that this research project accomplished successfully its goals.

Résumé

La machine à induction à cage d'écureuil, qui se caractérise par sa simplicité de construction, robustesse et faible coût, est couramment utilisée dans l'industrie moderne. Ces avantages rendent ce type de machine un choix populaire pour réaliser des entraînements électriques performants où, traditionnellement, seules les machines (CC) étaient utilisées. Toutefois, la commande d'une machine à induction est très complexe [1]. Récemment, les entraînements à vitesse variable incorporant des machines asynchrones ont pris le dessus en raison du progrès dans le domaine de l'électronique de puissance et de la technologie de traitement numérique du signal où plusieurs techniques de commande ont émergé. Dans le passé, des telles techniques de commande n'auraient pas été possibles en raison de la complexité du matériel et des logiciels nécessaires pour résoudre le problème complexe de la commande [3].

En général, la vitesse de la machine est mesurée pour l'utiliser comme un signal de rétroaction dans la boucle de contrôle de vitesse de la commande vectorielle. Les capteurs de position et de vitesse diminuent la fiabilité du système particulièrement dans les environnements hostiles. Dans la technique de commande sans capteur, le capteur de vitesse est remplacé par un estimateur de vitesse qui élimine les coûts associés au capteur de vitesse et réduit la complexité de l'entraînement [5].

L'objectif de ce travail de recherche est de concevoir et de réaliser la commande vectorielle d'une machine asynchrone (sans capteur) en utilisant un processeur numérique de signal (DSP) et d'en évaluer la performance. Le travail vise aussi à concevoir et implémenter un filtre numérique dans l'algorithme de commande sans capteur pour filtrer les courants mesurés de la machine.

Le travail commence par une recherche bibliographique. Ensuite, nous effectuons la modélisation et la simulation de la commande vectorielle dans l'environnement Matlab/Simulink/SimPowerSystems. Par la suite, nous développons un algorithme de commande permettant d'introduire un filtre numérique. Enfin, la validation expérimentale est effectuée. [1]

[1] *Un abrégé en français du mémoire est présenté en annexe.*

Acknowledgements

I am heartily thankful to my supervisor, Mamadou Lamine DOUMBIA and my co-supervisor, Pierre SICARD for their encouragement, guidance and support throughout my research project.

It is a pleasure to thank Sebastien DULAC and Francois LABARRE for their cooperation.

I am grateful to my friend Brahim I for the valuable discussions and the time we spent together.

Finally, I owe my deepest gratitude to my family for their ultimate support.

Table of Contents

List of Figures

List of Symbols

Subscripts

s	*Stator*
r	*Rotor*
a ,b, c	*Stator phases*
A, B, C	*Rotor phases*
d, q	*dq-winding*
l	*Leakage*
m	*Magnetizing*
m	*Mechanical*
ag	*air gap*

Superscripts

a, b, c	*Stator phases*
A, B, C	*Rotor phases*
→	*Space vector*
^	*Estimated Value*

Symbols

V, v	Voltage
I, i	Current
R	Resistance
L	Inductance
X	Reactance
E, e	Back electromotive force
S	Slip
λ, ψ	Flux
θ	Angle or position

ω	Angular velocity, speed
σ	Leakage factor
T_{em}	Electromechanical Torque
T_L	Load torque
p	Number of pole pairs
T_r	Rotor time constant
s	Derivative
P_m	Mechanical Power
ω_s	Speed of revolving field, in rads
ω_r	Speed of rotor, in rads
k_i	Integral gain
k_p	Proportional gain

Chapter 1- Introduction

1.1 Background

An electric drive is an electromechanical system that performs the conversion of electrical energy into mechanical energy for the operation of technological processes. Today, in industrialized countries, over 60% of the electrical power destined to the industry, is consumed by electric drives. Therefore, electric drives that require precise control of torque, speed and position are widely spread. Such drives are characterized by fast torque response and controllability of torque and speed over a wide range of operating conditions [1]. Industrial electric drives are generally classified into constant speed and variable speed ones. Traditionally, Alternating Current (AC) machines were used in applications with constant speed, while separately excited Direct Current (DC) machines were preferred for variable speed applications, because their control is simple and has a very fast torque response. However, DC machines, in addition to their high cost, have also maintenance problems associated with commutators and brushes, which limit machine operation in dirty and explosive environments [2].

The squirrel cage induction machine, characterized by its simplicity of construction, robustness and low cost, is commonly used in modern industry. These advantages made this type of AC machine a popular choice to achieve modern high performance electric drives for applications where traditionally only DC motors were applied. However, the control of such induction machines is quite complex. This is due to the fact that the rotor current in an induction machine, which is responsible for the torque production, owes its origin to the stator current, which also contributes to the air-gap flux, resulting into a coupling between the torque and flux producing mechanisms [1]. Recently, variable speed drives incorporating alternating current (AC) motors have been taking place over the direct current (DC) ones as a result of the progress in the field of power electronics and digital signal processing (DSP) technology where several control techniques emerged. In the past, such control techniques were not possible because of the complexity of the hardware and software needed to solve the complex control problem [3].

The appropriate control method of an induction machine is chosen depending on the required performance level. When the dynamic performance is not crucial as in the case of pumps, fans and compressors, relatively simple control techniques are used, which are generally called scalar control. But when dynamic performance is essential, it is necessary to control the torque at low speeds and during transients. Basically, there are two types of instantaneous torque control used for high performance applications; direct torque control and field oriented control, named also vector control [4]. Vector control is one of the techniques used to precisely control torque and speed of induction machines used in high performance applications. Using space-vector theory, it is easy to show that, similarly to the expression of the electromagnetic torque of a separately excited DC machine, the instantaneous electromagnetic torque of an induction machine can be expressed as the product of a flux-producing current and a torque-producing current, if a special, flux-oriented reference frame is used [3].

1.2 Research problem

Generally, the machine speed is measured and used as a feedback signal in the speed control loop of vector or direct torque control. Speed or position sensors degrade system reliability particularly in hostile environments. In sensorless control technique, the speed sensor is replaced by a speed estimator, which eliminates the costs associated with the speed sensor and reduces the drive complexity [5]. The major drawbacks of sensorless control are the constrains associated with the estimation of non measurable quantities (stator and rotor flux) and / or quantities we do not want to measure for economic reasons (speed, position) [6]. These quantities could be estimated from the easily measured stator voltages and currents. There are many techniques presented in the literature to extract the speed from measured voltages and currents. However, the performance of these techniques at low speed remains uncertain. A lot of recent research is focusing on developing efficient speed estimation techniques in order to improve low-speed performance of sensorless drives.

The speed estimation techniques are generally classified into three main groups:

- The first includes estimators based on the dynamic model of the induction machine. In this case the flux is deduced from the dynamic equations of the machine [6]. This approach is straightforward, but it is very sensible to machine parameters and there are also problems associated with using integrators to obtain the flux [7]. Because of errors induced by the use of pure integrators, flux estimators whose designs are based on low pass filters have been proposed [8, 9]. However, the filters generate phase delays incompatible with the requested dynamics and lead to instability at high speed.
- The second uses the techniques of signal processing to extract the required information. Despite its simplicity, this technique requires complex filtering circuits and its dynamic performance is poor [10, 11].
- The third is based on techniques of artificial intelligence (neural networks, fuzzy logic and genetic algorithm). The neural networks estimators seem to have a better response, but their practical implementation remains a challenge [6].

Traditionally, motor control was designed with analog components which are easy to design and can be implemented with relatively inexpensive components. However, the analog components are affected by temperature and aging, therefore the control system needs to be adjusted periodically. Moreover, the design of analog motor control is hardware dependant so the upgrade of such system is quite complex. Digital motor control systems provide practical solutions to the analog control drawbacks. Upgrade is easily performed by software and most of the analog components could be replaced by functions. Despite the solutions offered by digital control, the motor control industry faces many challenges such as power consumption reduction and Electromagnetic interference (EMI) reduction issues imposed by governments. Furthermore reducing the cost is essential for the industry to remain competitive. The results of these constraining factors are the need of enhanced algorithms. DSP technology allows to achieve both, a high level of performance as well as a system cost reduction [12].

1.3 Objective

The main objective of this research work is to design and implement a reliable sensorless vector control of induction machine using a digital signal processor (DSP) and to assess its performance. The work aims also to introduce digital filters into the control structure. To achieve the research goals, several issues will be addressed such as: (a) modelling and simulation of electric drives using Matlab/Simulink; (b) fixed-point digital signal processors and their application in electric drives; (c) digital filtering; (d) real time implementation; (e) LabVIEW interface development.

1.4 Methodology

The work begins with a literature review (State of the Art) of vector control and the study of fundamental concepts of different estimation techniques applied to sensorless speed control of squirrel cage induction machines. Then, modeling and simulation of an induction machine as well as vector control will be carried out in Matlab / Simulink. Furthermore, digital filter will be introduced in the control structure. Finally, we will conduct an experimental validation of the vector control on a test bench.

1.5 Thesis overview

The thesis consists of six chapters. Chapter 1 introduces electric drives, research questions and provides an overview of the thesis. Chapter 2 presents the steady state and dynamic models of the squirrel-cage induction machines. The relevant equations and equivalent circuits are detailed. Dynamic model simulation results are shown where the response of the induction machine could be evaluated. In chapter 3, the different techniques of the induction machine control are discussed. First, scalar control and its various types are discussed. Then direct torque control is reviewed. Both mentioned control types are dealt with in a qualitative way. However, field oriented control and estimation techniques are discussed in details followed by the simulations results.

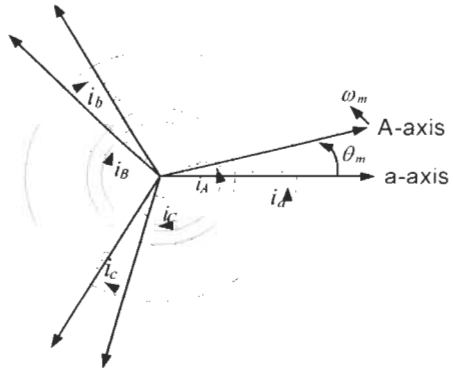
In chapter 4, digital filter types, characterization, analysis techniques and design aspects are addressed. Then, details of the filter used in the drive simulation and its design steps using Matlab filter design and analysis tool (FDAtool) are presented.

The experimental setup is detailed in the chapter 5, including the eZdsp board with the interface and current amplification circuits. A brief description of the Texas Instruments DSP platform (Code Composer Studio) is presented, in addition to the sensorless control implementation. Chapter 6 presents the conclusions of the research and gives recommendations for further research and development work.

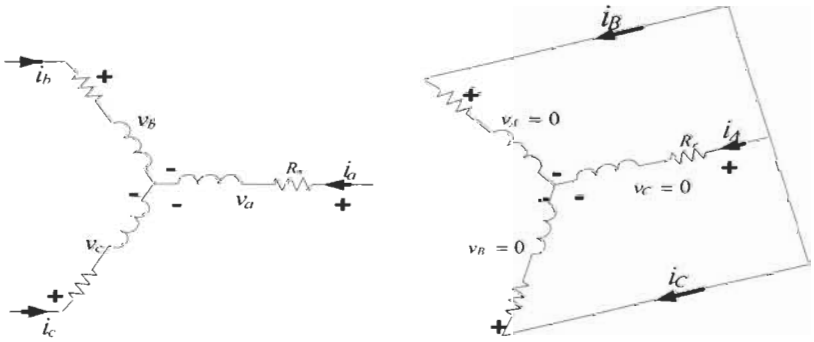
Chapter 2 - Induction motor modelling

2.1 Introduction

In this chapter, induction machine modelling is discussed, since obtaining the mathematical model that characterizes exactly the machine is primordial for motor control purposes. Both steady state and dynamic models are introduced. Finally, the developed Matlab-Simulink model and the simulation results are presented. For analysis purpose, the squirrel-cage on the rotor is replaced by a set of three sinusoidally distributed windings. Winding arrangement for a 2-pole, 3-phase, symmetrical squirrel-cage induction machine is shown in figure 2.1 [13] [14]. The stator and rotor are assumed to have the same number of turns.



a)



b)

Figure 2.1 Winding arrangement of a squirrel-cage induction motor

2.2 Steady state Model

When the stator of an induction machine is supplied from a balanced three phase sinusoidal power source and a constant load torque, the equivalent circuit of the induction machine is derived from the concept of a ''rotating transformer'' [15]. A diagram of the equivalent circuit with the rotor elements reflected into the stator for a polyphase induction machine is shown in figure 2.2. Core-loss resistance is neglected.

Stator and rotor phase voltage equations are drawn from figure 2.2 as follows:

$$V_{as} = R_s I_s + jX_{ls} I_s + E_{ag} \tag{2.1}$$

$$SE_{ag} = R_r I'_r + jSX_{lr} I'_r \tag{2.2}$$

Mechanical Power equation:

$$P_m = T_{em}\, \omega_r \tag{2.3}$$

Electromechanical Torque:

$$T_{em} = 3\frac{I'^2_r R'_r}{S\omega_s} \tag{2.4}$$

Where

$$S = \frac{\omega_s - \omega_r}{\omega_s} \tag{2.5}$$

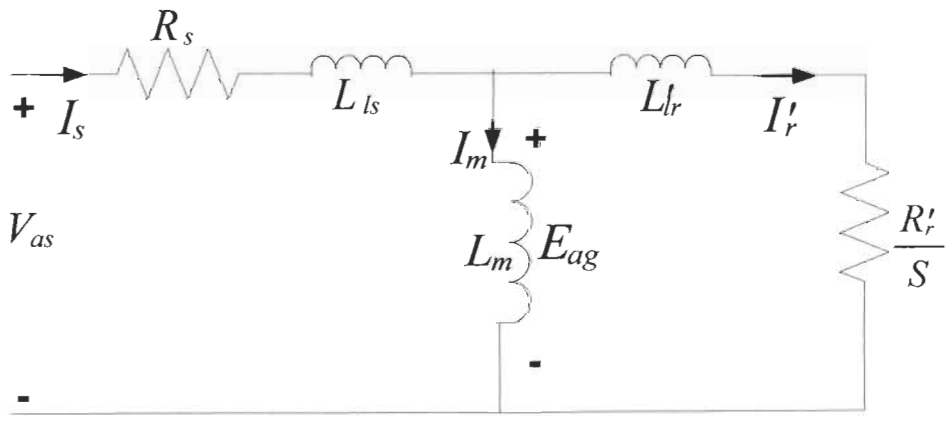


Figure 2.2 Per-phase equivalent circuit in steady state

2.3 Dynamic Model

The equivalent circuits, derived from steady state operation with sinusoidal voltages and currents, are inadequate when dealing with transients or when the motor is supplied from a switched converter [16]. Therefore the steady state equations written in phasor representation are not applicable under dynamic conditions. Instead space vector representation is used to write the machine equations.

Per-phase voltage equations are expressed as:

$$\vec{v}_s^{\,a}(t) = R_s \vec{i}_s^{\,a}(t) + \frac{d}{dt}\vec{\lambda}_s^{\,a}(t) \tag{2.6}$$

$$0 = R_r \vec{i}_r^{\,A}(t) + \frac{d}{dt}\vec{\lambda}_r^{\,A}(t) \tag{2.7}$$

Flux linkage equations:

$$\vec{\lambda}_s^{\,a}(t) = L_s \vec{i}_s^{\,a}(t) + L_m \vec{i}_r^{\,a}(t) \tag{2.8}$$

$$\vec{\lambda}_r^{\,A}(t) = L_m \vec{i}_s^{\,A}(t) + L_r \vec{i}_r^{\,A}(t) \tag{2.9}$$

Equations (2.6)-(2.9) constitute the space-phasor model of a squirrel cage induction machine.

We notice that in (2.8) the rotor current space vector is defined with respect to the stator phase-a axis, meanwhile the stator current in (2.9) is defined with respect to the rotor phase-A axis.

It is clear that the flux linkage depends on the rotor position for given values of the stator and the rotor currents at any instant of time. For this reason, the voltage equations in phase quantities, expressed in a space vector form by (2.6) and (2.7), which include the time derivatives of the flux linkage, are complicated to solve. This coupling could be avoided by using *dq*-analysis.

At any instant of time, the air gap *mmf* distribution produced by three phase windings can also be produced by a set of two orthogonal windings, one winding along the d-axis, and the other along the q-axis.

The *dq* winding analysis allows the torque and the flux in the machine to be controlled independently under dynamic conditions.

Any three-phase sinusoidal set of quantities can be transformed to an orthogonal reference frame by the following transformation matrix:

$$\begin{bmatrix} i_d(t) \\ i_q(t) \end{bmatrix} = \sqrt{\frac{2}{3}} \begin{bmatrix} \cos(\theta) & \cos\left(\theta - \frac{2\pi}{3}\right) & \cos\left(\theta - \frac{4\pi}{3}\right) \\ -\sin(\theta) & -\sin\left(\theta - \frac{2\pi}{3}\right) & -\sin\left(\theta - \frac{4\pi}{3}\right) \end{bmatrix} \begin{bmatrix} i_a(t) \\ i_b(t) \\ i_c(t) \end{bmatrix} \quad (2.10)$$

Now we can formulate stator and rotor windings equations along two orthogonal axes *d* and *q* moving at speeds ω_d and ω_{dA} respectively:

Stator windings:

$$v_{sd} = R_s i_{sd} + \frac{d}{dt}\lambda_{sd} - \omega_d \lambda_{sq} \quad (2.11)$$

$$v_{sq} = R_s i_{sq} + \frac{d}{dt}\lambda_{sq} + \omega_d \lambda_{sd} \quad (2.12)$$

Rotor windings:

$$0 = R_r i_{rd} + \frac{d}{dt}\lambda_{rd} - \omega_{dA} \lambda_{rq} \quad (2.13)$$

$$0 = R_r i_{rq} + \frac{d}{dt}\lambda_{rq} + \omega_{dA} \lambda_{rd} \quad (2.14)$$

Where ω_d and ω_{dA} are the instantaneous speed of the *dq*-winding set in the air gap with respect to the stator and rotor axis respectively. Assuming an arbitrary value for the *dq*-winding speed with respect to the stator axis, ω_d determines the reference frame, either stationary ($\omega_d = 0$), stator ($\omega_d = \omega_{syn}$) or rotor ($\omega_d = \omega_m$) [13].

Electromagnetic torque:

$$T_{em} = \frac{p}{2} L_m \left(i_{sq} i_{rd} - i_{sd} i_{rq} \right) \tag{2.15}$$

Mechanical speed:

$$\frac{d}{dt} \omega_{mech} = \frac{T_{em} - T_L}{J_{eq}} \tag{2.16}$$

$$\omega_{elec} = \omega_{mech} * p \tag{2.17}$$

The *dq*-equivalent circuit is obtained by substituting for flux linkage derivatives into voltage equations (2.11)-(2.14). These equations are in a general reference frame.

$$v_{sd} = R_s i_{sd} - \omega_d \lambda_{sq} + L_{ls} \frac{d}{dt} i_{sd} + L_m \frac{d}{dt} (i_{sd} + i_{rd}) \tag{2.18}$$

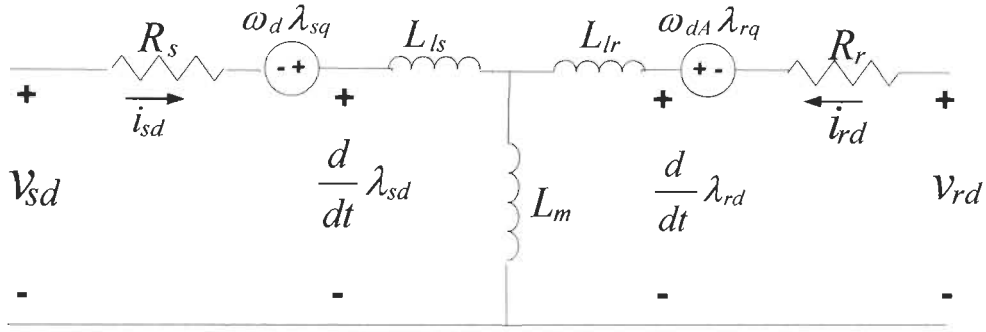
$$v_{sq} = R_s i_{sq} + \omega_d \lambda_{sd} + L_{ls} \frac{d}{dt} i_{sq} + L_m \frac{d}{dt} (i_{sq} + i_{rq}) \tag{2.19}$$

Rotor windings:

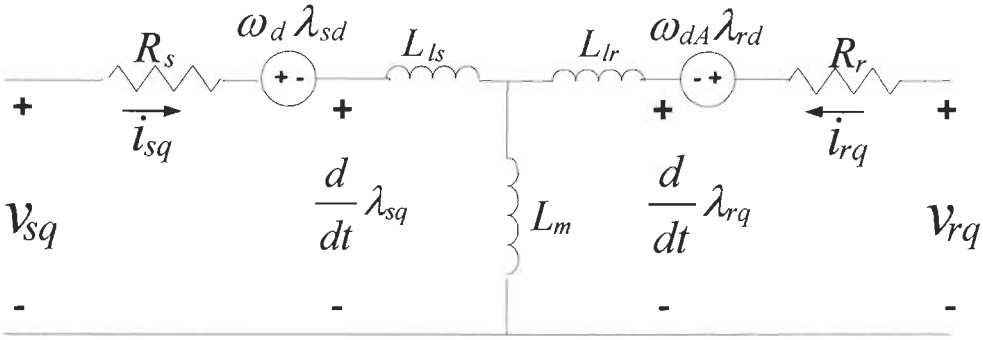
$$0 = R_r i_{rd} - \omega_{dA} \lambda_{rq} + L_{lr} \frac{d}{dt} i_{rd} + L_m \frac{d}{dt} (i_{rd} + i_{sd}) \tag{2.20}$$

$$0 = R_r i_{rq} + \omega_{dA} \lambda_{rd} + L_{lr} \frac{d}{dt} i_{rq} + L_m \frac{d}{dt} (i_{rq} + i_{sq}) \tag{2.21}$$

For each axis, the stator and the rotor windings are combined to result in the *dq* equivalent circuit shown in figures 2.3a and 2.3b [13].



a) *d*-axis



b) *q*-axis

Figure 2.3 *dq*-winding equivalent circuits

2.4 Simulation

The dynamic model equations (2.18) – (2.21) together with the motion equations (2.15) – (2.17) were used to develop a squirrel cage induction model in Matlab-Simulink.

The stator is selected as reference frame ($\omega_d = \omega_{syn}$), therefore ($\omega_{dA} = \omega_{syn} - \omega_m$). Figure 2.4 shows the developed model in Simulink.

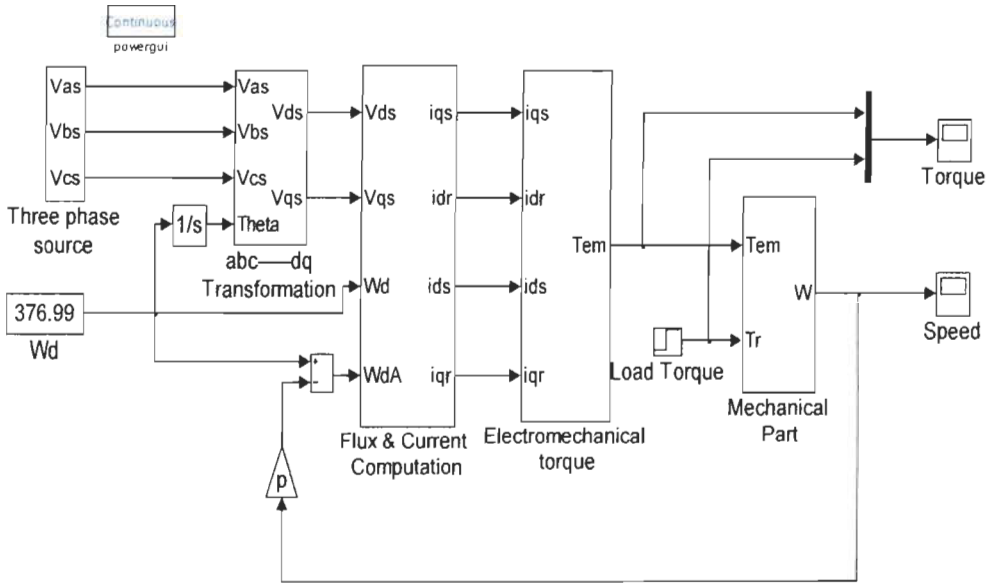


Figure 2.4 Induction Machine Model in Matlab/Simulink

Figures 2.5, 2.6 and 2.7 show the speed, torque and current responses of the induction machine dynamic model shown in figure 2.4. At t=0.5s a load torque is applied to the machine and its response is found satisfactory

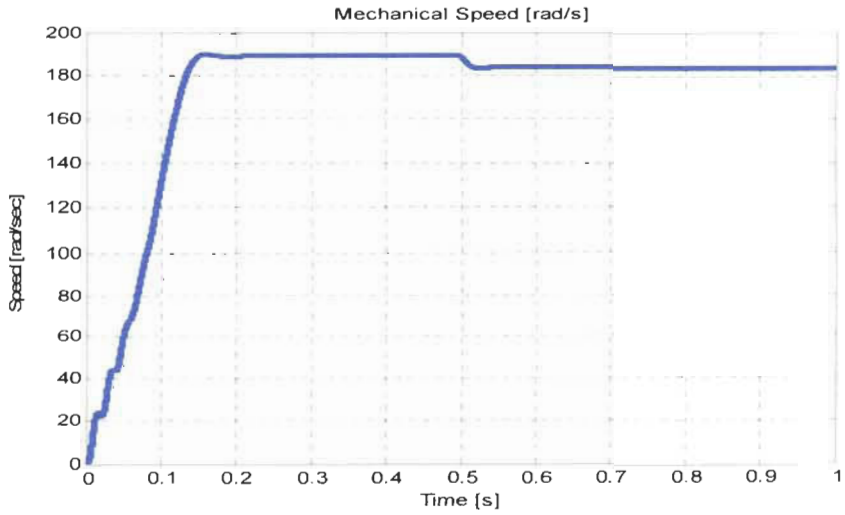


Figure 2.5 Motor speed

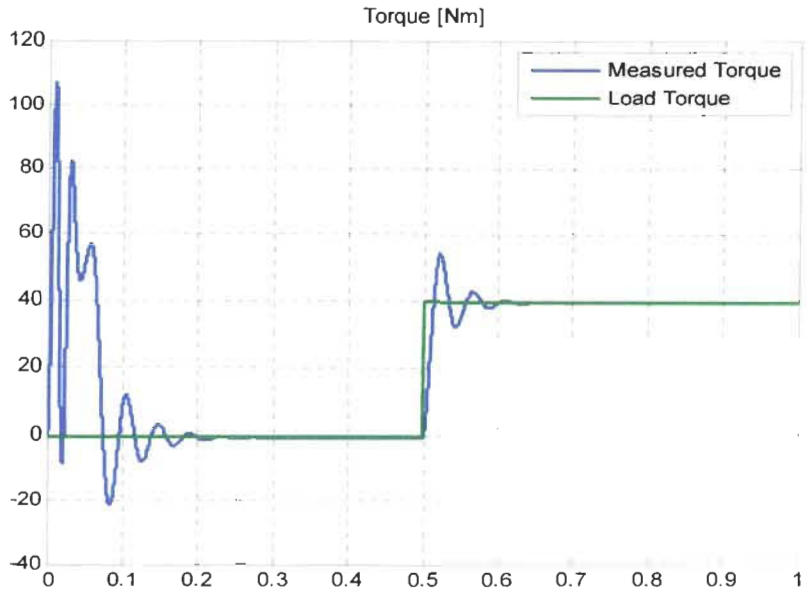


Figure 2.6 Electromagnetic Torque

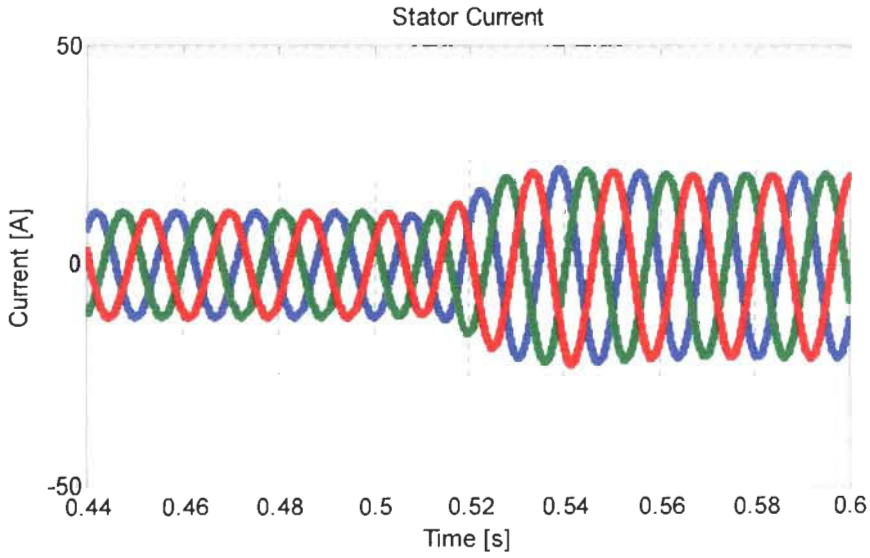


Figure 2.7 Stator current

2.5 Conclusion

Squirrel-cage induction machine modelling and simulation is described in this chapter. The traditional per-phase equivalent circuit is presented. Generally, this model is useful in analysing the steady state performance of the induction machine. A dynamic model of the induction machine is developed and the equivalent circuit is introduced. The dynamic model is simulated using Simulink. The machine response to a torque variation is investigated. The dynamic model will be used in the next chapter to develop a Simulink model of an electric drive.

Chapter 3 - Induction motor control

3.1 Introduction

This chapter describes various control methods used in induction machine variable speed drives. First, scalar control techniques based on the steady state model are discussed. Instantaneous torque control methods, mainly direct torque and field oriented control are presented. Inverter pulses generation method is also described. Finally, sensorless control and various estimation techniques are discussed in detail followed by simulation results.

3.2 Scalar control

The basic techniques of induction machine speed control are elaborated using the elementary model described in the steady state analysis. These techniques are viable when the motor operate at steady torque and when small errors in speed control are tolerated. Several open-loop (scalar) control methods are outlined in the literature. Some of popular scalar control methods are detailed below.

3.2.1 Stator Voltage Control

In this method of speed control, the magnitude of the terminal voltage is varied up to its rated value. A wide range of speed control cannot be accomplished by this technique. Generally, back-to-back thyristors are used for this purpose but this type of drive suffers from poor power and harmonic distortion factors when operated at low speed [17].

3.2.2 Voltage /frequency (V-f) control

In this scheme, induction motors are driven from a variable voltage supply with adjustable frequency. The combination of voltage amplitude and its frequency is necessary, since increasing the supply frequency alone increases the motor speed but reduces the maximum motor torque. Furthermore increasing the voltage amplitude alone increases motor maximum torque. The motor supply voltage amplitude is varied in proportion with its frequency in order to maintain the air gap flux to its rated value [17], [18].

3.3 *Vector control*

It is a control strategy for three phase ac motors that emulates the dc motor control by orienting the stator current so as to control torque and flux independently. The stator current components obtained are oriented in phase (flux component) and in quadrature (torque component) to the flux vector, which can be one of the three distinct flux space-phasors in the induction machine: airgap, stator and rotor flux. Vector control could be performed with respect to any of these flux space-phasors by attaching the reference system *d* axis to the respective flux linkage space-phasor direction and by keeping its amplitude under surveillance. The rotor flux orientation provides natural decoupling, fast torque response and all around stability. Stator flux orientation and airgap flux orientation are attractive due to ease of flux computation and for the purpose of wide range of field weakening, but a decoupler network is needed [1, 19].

Vector control technique allows a squirrel-cage induction motor to be driven with high dynamic performance, comparable to that of a dc motor. This type of control is suitable for applications that require accurate control of speed and position. In vector control, the dynamic model of the induction machine rather than steady-state model is used to design the controller [17, 19]. There are basically two different types of vector control techniques. Either direct or indirect field oriented control (FOC), depending upon the method of flux acquisition.

3.3.1 Direct FOC

In the direct method, the airgap flux is directly measured with the help of sensors, search coils or taped stator windings or estimated from machine terminal variables such as stator voltage, current and speed. Since it is not possible to directly sense rotor flux, it is synthesized from the directly sensed airgap flux. Rotor flux angle is directly computed from flux estimation or measurement.

A major drawback with the direct orientation scheme is the inherent problem at very low speeds when the machine IR drop dominates and the required integration of the signal to measure the airgap flux is difficult. Closed-loop stator flux observers based on the motor current, voltage and the measured rotor position have been found to overcome this difficulty [1].

To illustrate a rotor flux based direct vector-controlled drive, figure 3.1 shows the schematic diagram of an induction motor drive system with impressed stator currents employing direct rotor-flux-oriented control and using a current regulated pulse width modulation inverter (CRPWM) [3].

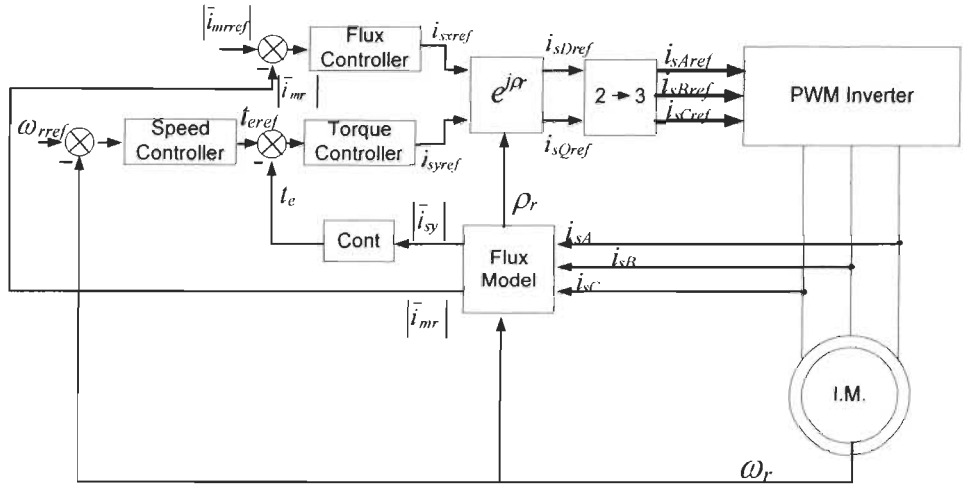


Figure 3.1 Direct FOC of an induction motor

3.3.2 Indirect FOC

In this method, the rotor flux angle is indirectly computed by summing a sensed rotor position signal with a commanded slip position. In contrast to direct methods, the indirect methods are highly dependent on machine parameters, which may vary with temperature, saturation level and frequency. Consequently, parameter adaptation schemes are required to have a better overall performance. Figure 3.2 shows the schematic diagram of an induction motor drive system with impressed stator currents employing indirect rotor-flux-orientation [1, 3]. The speed control loop is not shown, but it is identical to that illustrated in figure 3.1.

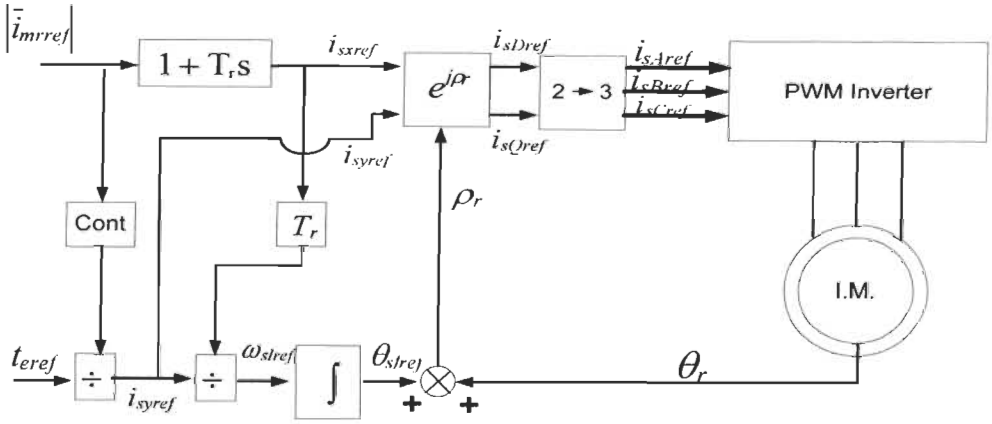


Figure 3.2 Indirect FOC of an induction motor

3.4 Direct torque control

In addition to vector control systems, instantaneous torque control yielding fast torque response can also be obtained by employing direct torque control (DTC) where flux linkage and electromagnetic torque are controlled directly and independently by the selection of optimum inverter switching modes. The selection is made to restrict the flux-linkage and electromagnetic torque errors within the respective flux and torque hysteresis band, to obtain fast torque response, low inverter switching frequency and low harmonic losses. Figure 3.3 shows the schematic of a simple form of direct torque control induction motor drive [3]. The speed control loop is not shown, but it is identical to that illustrated in figure 3.1.

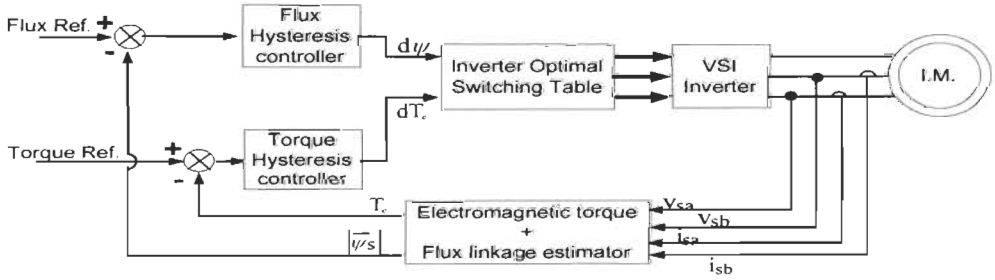


Figure 3.3 DTC of induction motor

3.5 Space Vector Pulse Width Modulation

The structure of a typical 3-phase power inverter is shown in Figure 3.4 where Vab, Vbc and Vca are the voltages applied to the motor windings. Vdc is the continuous inverter input voltage. The ON-OFF sequence of all these devices must respect the following conditions:

1- Three of the switches must always be ON and three always OFF.

2- The upper and the lower switches of the same leg are driven with two complementary pulsed signals. In this way no vertical conduction is possible, providing care is taken to ensure that there is no overlap in the power switch transitions [20].

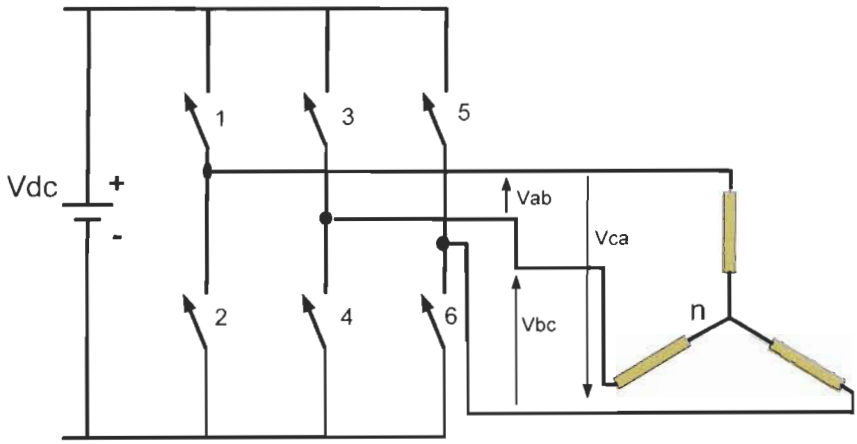


Figure 3.4 Three phase inverter connected to an induction motor

Space Vector Pulse Width Modulation (SVPWM) is one of the methods to determine switched pulse width. It is easy to implement and has a good performance concerning harmonic content reduction. Taking into consideration the two constraints quoted above, there are eight possible switch states. Two of these states ($\overline{V}_0$ and $\overline{V}_7$) correspond to a short circuit on the output, while the other six can be considered to form stationary vectors in the *d-q* complex plane as shown in figure 3.5 [21].

The reference vector, which represents three phase sinusoidal voltage, is generated using SVPWM by switching between two nearest active vectors and zero vector [22]. Each of the active six vectors has a magnitude of 4/3 V_{dc} .

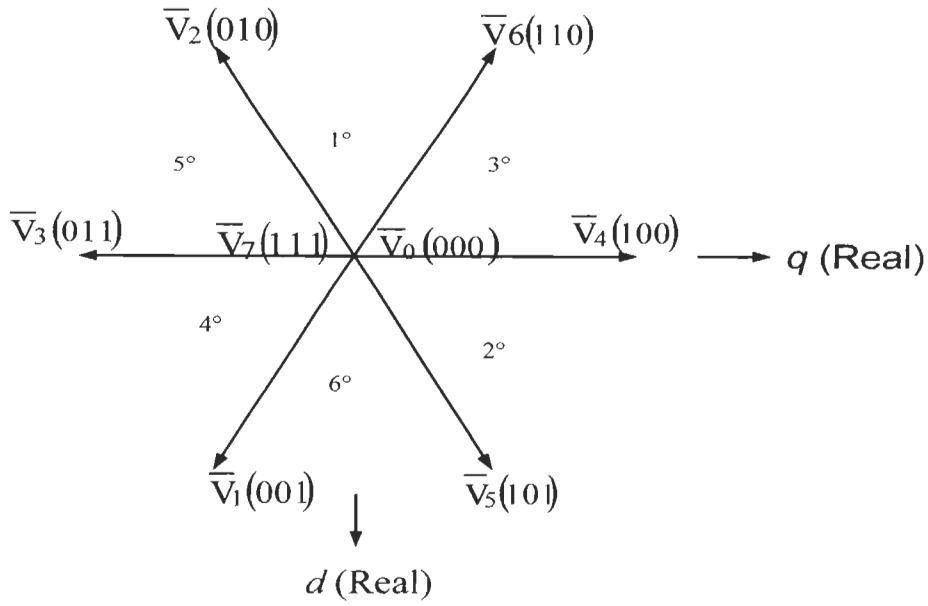


Figure 3.5 SVPWM Switch states

Assuming that the reference vector $\overline{V}_{ref}$ is in the 3° sector as shown in Figure 3.6, we have the following situation, where T4 and T6 are the times during which the vectors *V*4 and *V*6 are applied and T0 is the time during which the zero vectors are applied. When the reference voltage and the sample periods are known, the following system makes it possible to determine the switching times T4, T6 and T0.

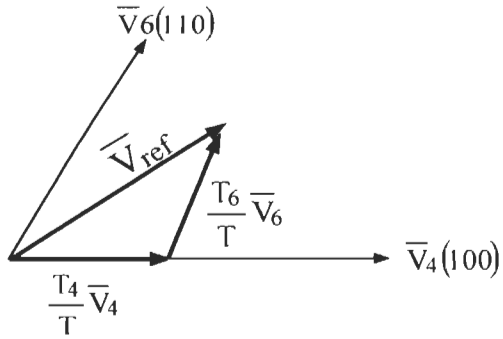


Figure 3.6 Reference vector V_{ref} is in the 3° sector

$$T = T_4 + T_6 + T_0 \tag{3.1}$$

$$\overline{V}_{ref} = \frac{T_4}{T}\overline{V}_4 + \frac{T_6}{T}\overline{V}_6 \tag{3.2}$$

A Matlab/Simulink model of SVPWM inverter developed in [22] is used in the vector control simulation. The switching time and corresponding switch state for each power switch is calculated using Matlab code. Figure 3.7 shows the complete inverter model in Simulink. The model is tested by supplying it with *d-q* reference voltage and the results are shown in figure 3.8.

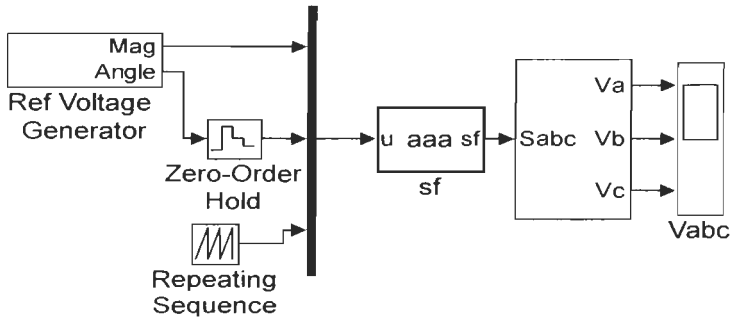


Figure 3.7 Matlab/Simulink model of SVWPM inverter

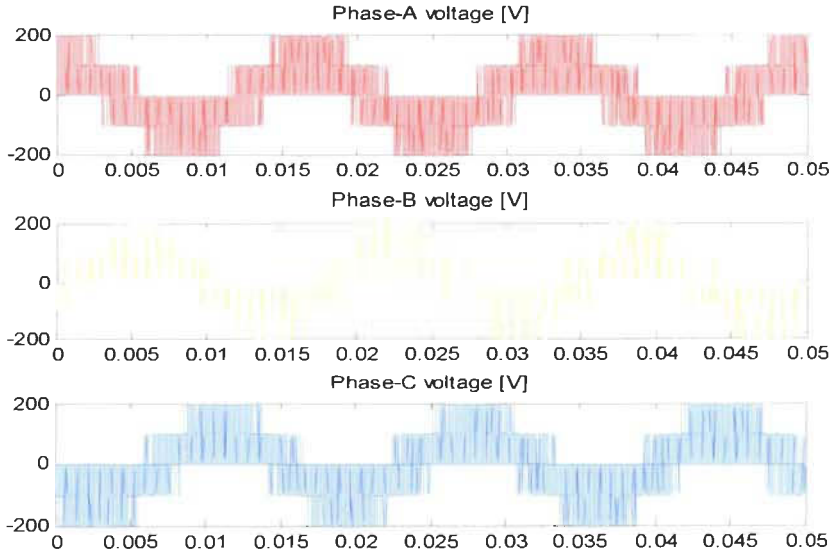


Figure 3.8 SVPWM Inverter output phase voltage

3.6 Simulation of direct vector control

The rotor flux oriented direct vector control technique described in subsection 3.3.1 is simulated using Simulink. The simulated control structure in shown in figure 3.9 where the induction machine subsystem contains the motor model developed in chapter two. The inverter subsystem consists of the SVPWM inverter model detailed in section 3.5 above.

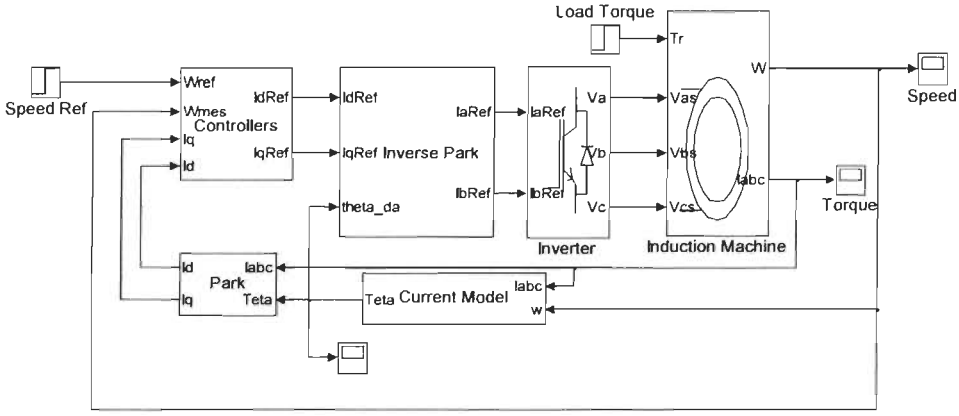


Figure 3.9 Direct vector control overall structure-Simulink model

The current model block calculates the rotor angle, which is derived from the rotor voltage equations expressed in the rotor flux oriented reference frame. In this case, the entire flux is aligned to the rotor flux, so the quadrature component of the rotor flux equals zero. Reformulating rotor equations (2.13-2.14), we obtain the new rotor equations shown in (3.3) and (3.5). The rotor angle is calculated using the following equations as shown in figure 3.10.

$$0 = \frac{R_r}{L_r}\lambda_{rd} + \frac{d}{dt}\lambda_{rd} - \frac{R_r \times L_m}{L_r} i_{sd} \tag{3.3}$$

$$\lambda_{rd} = \frac{L_m}{1 + T_r \cdot s} i_{sd} \tag{3.4}$$

$$0 = R_r i_{rq} + (\omega_{sys} - \omega_m)\lambda_{rd} \tag{3.5}$$

$$i_{rq} = -\frac{L_m}{L_r} i_{sq} \tag{3.6}$$

$$\omega_{sl} = \frac{R_r \times L_m}{L_r \times \lambda_{rd}} i_{sq} \tag{3.7}$$

$$\theta = \int \omega_m + \omega_{sl} \tag{3.8}$$

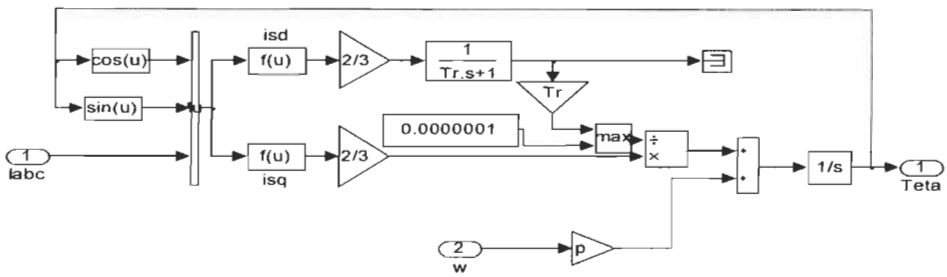


Figure 3.10 Current model block

The controller block illustrated in the direct vector control structure (figure 3.9) contains two parallel control loops having Proportional Integral (PI) controllers. The first loop is used to adjust the machine flux. The second one consists of two cascade loops, the inner current and outer speed control loops. The controllers are tuned by trial and error method. Figure 3.11 below shows the contents of the controller block.

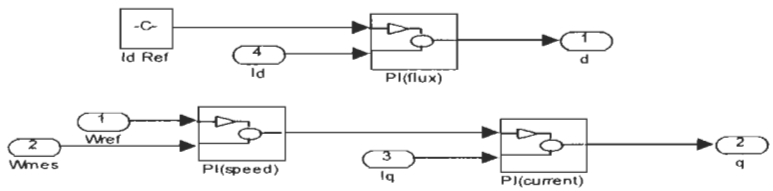


Figure 3.11 Controller block

Figure 3.12 through figure 3.15 show the simulation results of the direct vector control.

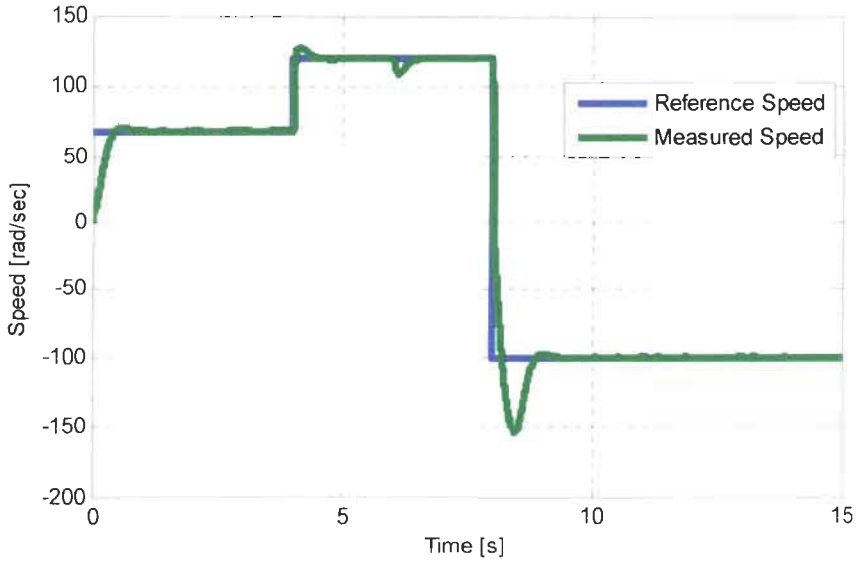


Figure 3.12 Direct vector control speed response

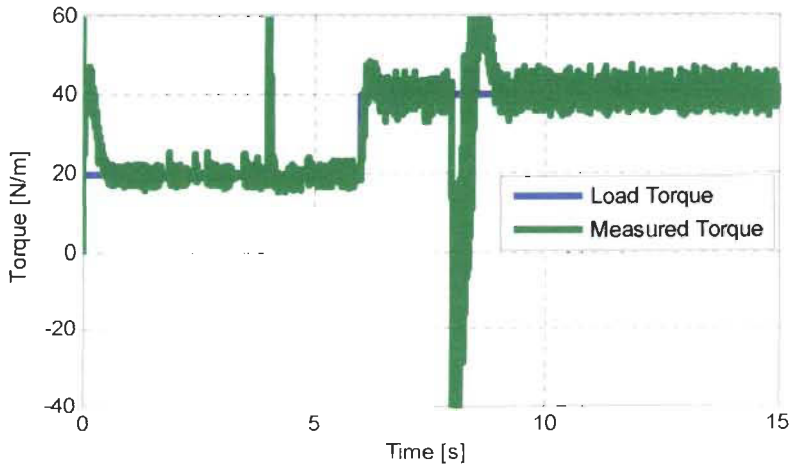


Figure 3.13 Direct vector control torque response

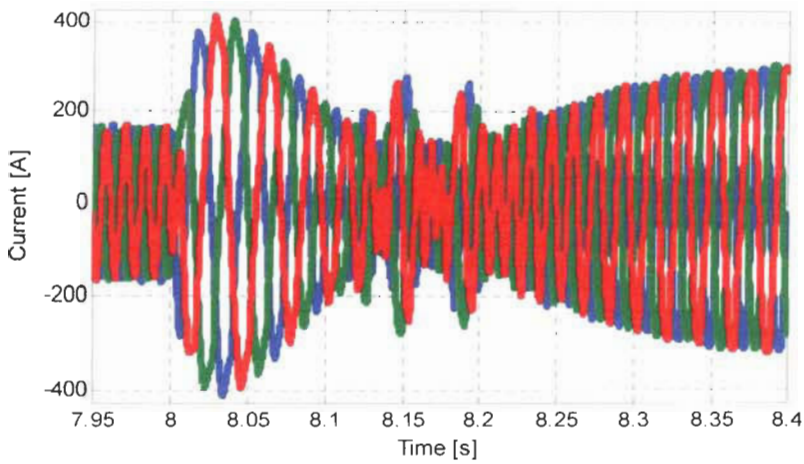


Figure 3.14 Direct vector control – stator current

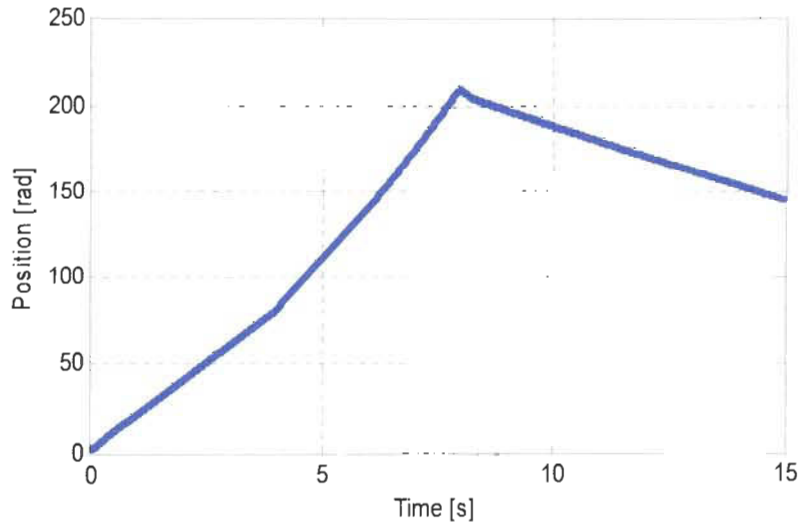


Figure 3.15 Direct vector control rotor position

3.7 Discussion:

Different operating conditions were investigated in order to validate the direct vector control model and to demonstrate the effectiveness of the system modeling and simulation. Figure 3.12 shows the reference and actual motor speeds, at t=4s, motor speed is increased from 70 to 120 rad/s. The speed dip at t=6s is due to load torque augmentation. Electromagnetic torque response is illustrated in figure 3.13. Load torque is increased at t=6s. Despite the presence of torque pulsations, the motor follows precisely the load torque value. At t=8s, motor speed is reduced from 120 to -100 rad/s We notice in both figures that in presence of disturbance the vector control responds adequately and brings back the controlled variable to its desired value. Figure 3.14 illustrates the stator current. The flux angle (position) is shown in figure 3.15.

3.8 Sensorless Control

Sensorless vector control is an instantaneous electromagnetic torque control technique applied to variable speed (AC) motor drives where speed estimators and observers are used instead of the costly speed or position sensors. Eliminating speed and position sensors reduces hardware complexity and cost, increases the mechanical robustness and reliability of the drive, and leads to a better noise immunity.

There are many speed or position extraction techniques, some of them are only valid in the steady state. Such techniques could be used in low-cost drive applications not requiring high dynamic performance. Other techniques are applicable for high-performance applications in vector- and direct-torque-controlled drives. Most of these techniques depend on machine parameters. Therefore various parameter adaptation schemes have been proposed in the literature. In an ideal sensorless drive, speed information and control is provided with an accuracy of 0.5 % or better, from zero speed to the highest speed, for all operating conditions and independently from saturation levels and parameter variations [3].

The estimation techniques can be classified into three main groups as detailed below.

The first includes estimators based on the dynamic model of the induction machine. The flux is deduced from the dynamic equations of the machine [6] using the monitored stator voltages and currents. The basic scheme is called open-loop estimator. This approach is straightforward, but there are problems associated with parameter mismatch and noise, especially at low speed [7]. Thus it is difficult to achieve stable, very low speed operation in a speed or position sensorless high-performance induction motor drive. Others improved schemes like model reference adaptive systems (MRAS), and observers (Kalman and Luenberger) have better accuracy.

The second uses techniques of signal processing to extract the required information. In one of such estimators, stator currents and spatial saturation third-harmonic voltage are measured and then the fundamental component of the magnetizing flux is determined using the third-harmonic flux. Despite its simplicity, this technique requires complex filtering circuits and its dynamic performance is poor [6].

The third is based on techniques of artificial intelligence (neural networks, fuzzy logic and genetic algorithm). The neural networks estimators seem to have a better response, but their practical implementation remains a challenge [6].

The most popular techniques of sensorless control for induction motor drives are presented as follows.

3.8.1 Open loop estimators

The monitored stator voltages/currents are used to estimate the flux linkage components, from which the speed is derived.

The mathematical expression for the rotor speed is given as:

$$\omega_r = \omega_{mr} - \omega_{sl} \tag{3.9}$$

Where ω_{mr} is the speed of the rotor flux–linkage space vector with respect to the rotor.

$$\omega_{mr} = \frac{d}{dt}\left[tan^{-1}\left(\frac{\lambda_{rq}}{\lambda_{rd}}\right)\right] = \frac{\lambda_{rd}\, d\lambda_{rq}/dt - \lambda_{rq}\, d\lambda_{rd}/dt}{\lambda_{rd}^2 + \lambda_{rq}^2} \tag{3.10}$$

The angular rotor-slip-frequency (ω_{sl}) is derived from rotor voltage equation of the induction machine expressed in the rotor flux oriented frame as follows:

$$\omega_{sl} = \frac{L_m}{T_r \left|\overline{\lambda_r}\right|^2}\left(- \lambda_{rq} i_{sd} + \lambda_{rd} i_{sq} \right) \tag{3.11}$$

Finally we obtain the speed as shown below:

$$\omega_r = \frac{\lambda_{rd}\, d\lambda_{rq}/dt - \lambda_{rq}\, d\lambda_{rd}/dt}{\left|\overline{\lambda_r}\right|^2} - \frac{L_m}{T_r\left|\overline{\lambda_r}\right|^2}\left(-\lambda_{rq} i_{sd} + \lambda_{rd} i_{sq}\right) \tag{3.12}$$

The implementation of (3.12) is shown in Figure 3.16. First the flux (λ) is deduced using (2.18) and (2.19) in a stationary reference frame ($\omega_d = 0$) as shown in (2.13, 2.14) then (3.12) is used to extract the speed.

$$\frac{L_m}{L_r}\frac{d}{dt}\lambda_{rd} = v_{sd} - R_s i_{sd} + L_s\left(1 - \frac{L_m^2}{L_r L_s}\right)\frac{d}{dt} i_{sd} \tag{2.13}$$

$$\frac{L_m}{L_r}\frac{d}{dt}\lambda_{rq} = v_{sq} - R_s i_{sq} + L_s\left(1 - \frac{L_m^2}{L_r L_s}\right)\frac{d}{dt} i_{sq} \tag{2.14}$$

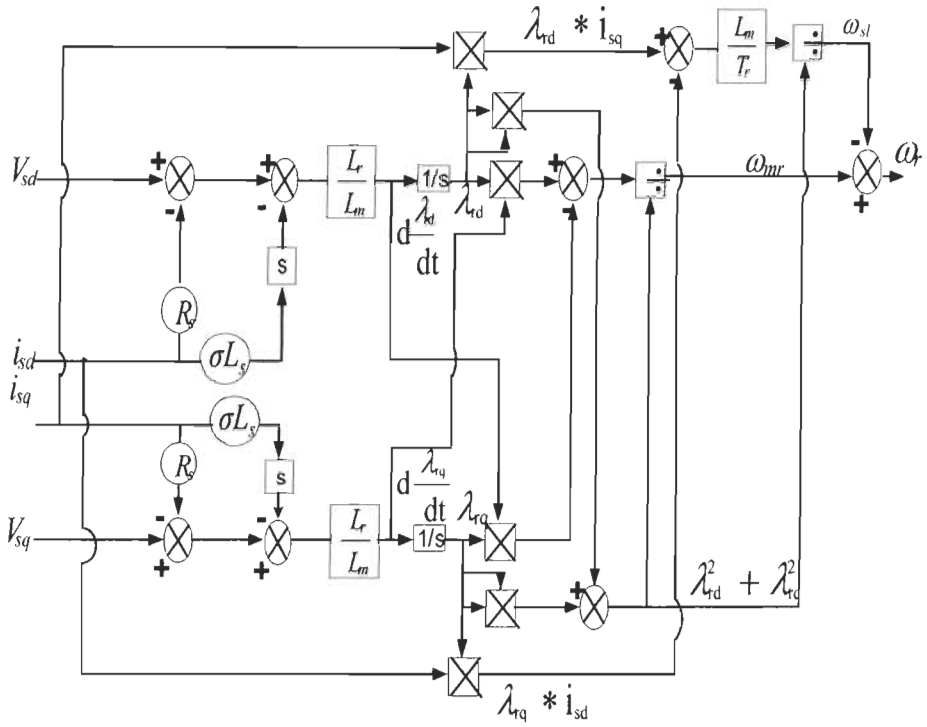


Figure 3.16 Open-loop rotor speed estimator

The estimator shown in Figure 3.16 is implemented using Matlab-Simulink as illustrated in figure 3.17, the motor is supplied from a three phase supply and the measured stator and rotor currents are fed to the estimator as inputs.

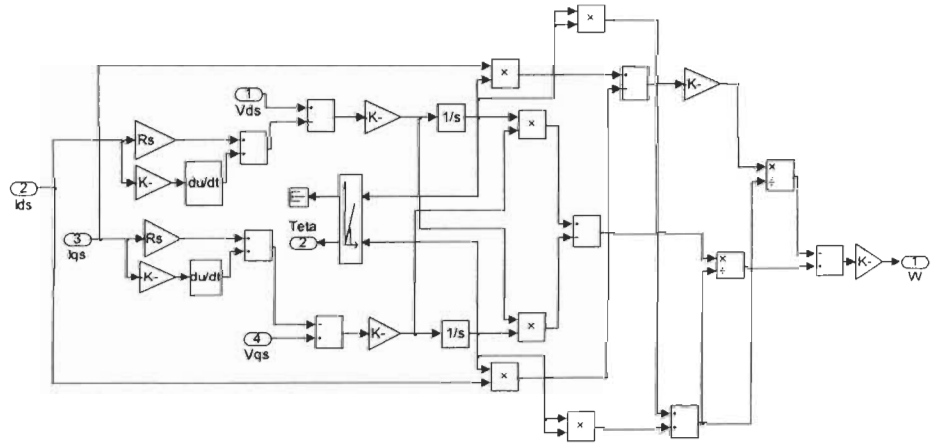


Figure 3.17 Open-loop estimator - Simulink model

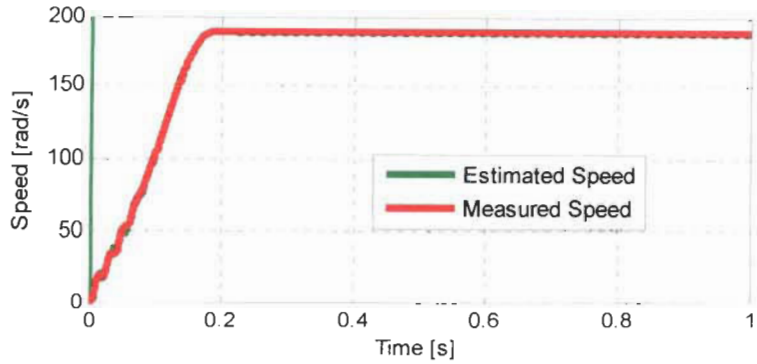


Figure 3.18 Open-loop estimator speed

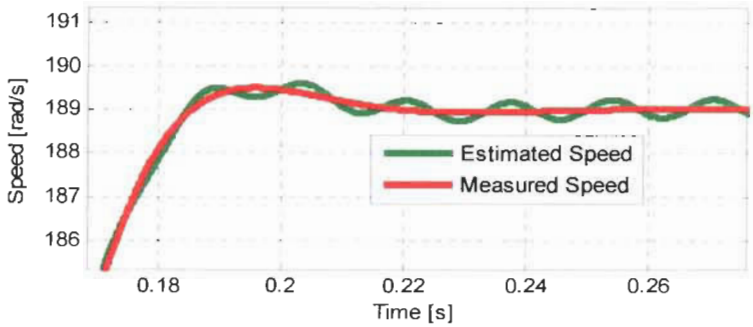


Figure 3.19 Zoomed Open-loop estimator speed

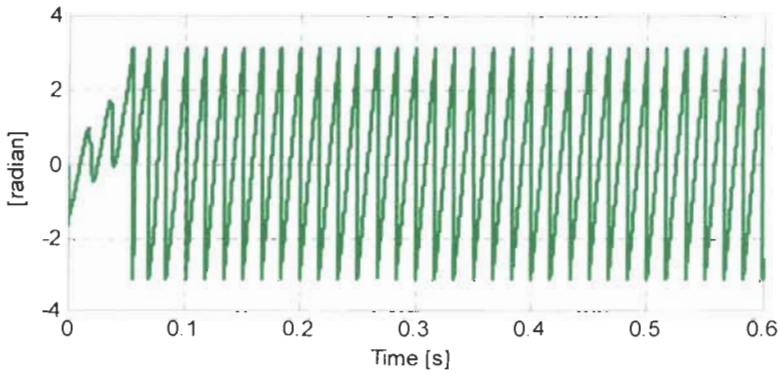


Figure 3.20 Open-loop estimator flux angle

The speed response of the open-loop estimator is shown in figure 3.18; at motor starting we notice a small overshoot in the estimated speed. Furthermore the zoomed speed response illustrated in figure 3.19 shows ripples in the estimated speed, which is a commonplace in such type of speed estimators. Figure 3.20 shows the estimated flux angle.

3.8.2 Model Reference Adaptive Systems

The accuracy of the open loop estimator discussed above depends strongly on the machine parameters. In closed loop estimators the accuracy can be increased. In the present section, a closed loop estimator using the Model Reference Adaptive System (MRAS) is described. In a MRAS system, some state variables, (e.g. rotor flux-linkage components, or back *emf,* etc.) of the induction machine are estimated in a reference model and are then compared with state variables estimated by using an adjustable model. The difference between these state variables is then used in an adaptation mechanism, which outputs the estimated value of the rotor speed and adjusts the adaptive model until satisfactory performance is obtained [3]. Figure 3.21 shows a complete scheme of MARS-based speed estimator using rotor flux linkages for the speed tuning signal [3].

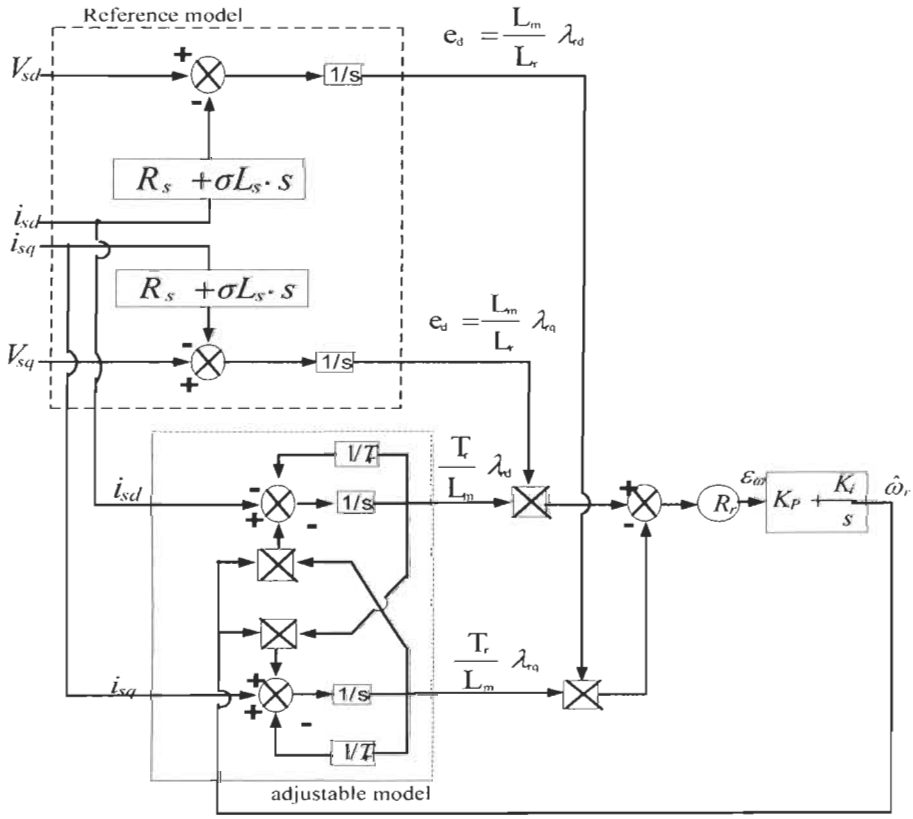


Figure 3.21 MRAS-based speed estimator

The MRAS speed estimator shown in Figure 3.21 is implemented using Matlab/Simulink as illustrated in figure 3.22, the motor is supplied from a three phase supply and the measured stator and rotor currents are fed to the estimator as inputs.

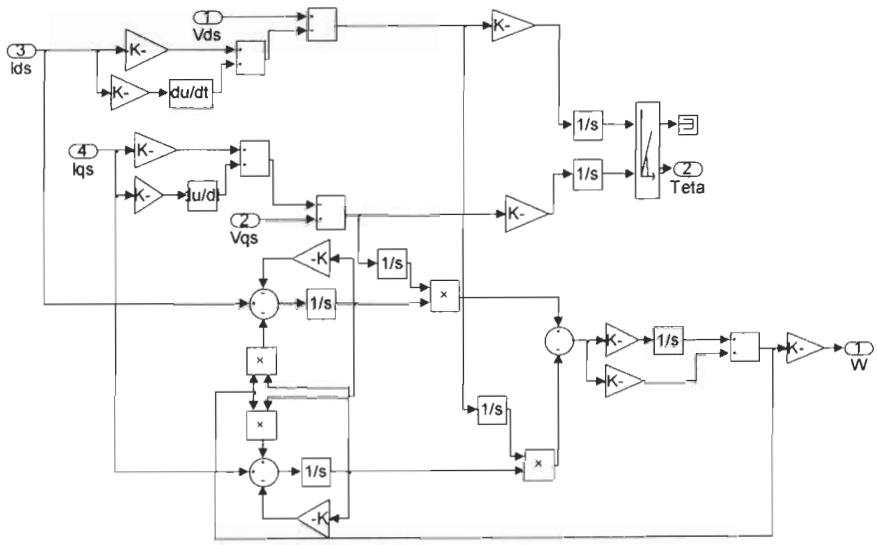


Figure 3.22 MRAS-based speed estimator- Simulink model

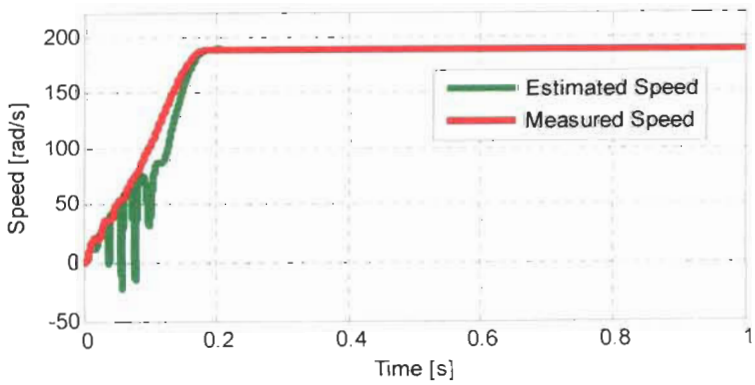


Figure 3.23 MRAS-based estimator speed

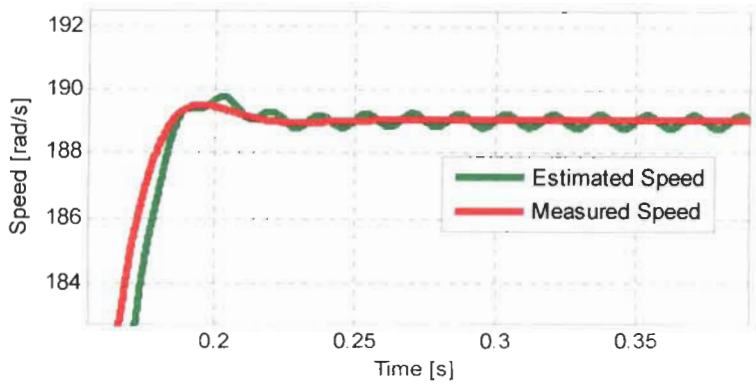


Figure 3.24 Zoomed MRAS-based estimator speed

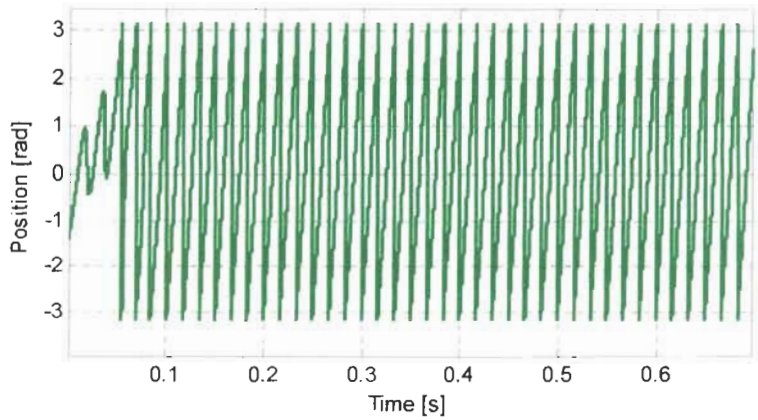


Figure 3.25 MRAS-based estimator flux angle

The MRAS estimator is simulated under the same conditions of the open loop estimator. Here we notice the absence of the starting speed overshoot found in open loop estimators. However, the accuracy of the estimated speed is compromised at low speed due to the use of pure integrators.

3.8.3 Adaptive Observer

A state observer is a model-based state estimator that can be used for the state estimation of a non-linear dynamic system in real time. In the calculations, the states are predicted by using a mathematical model, but the predicted states are continuously corrected by using a feedback correction scheme. Stator and rotor equations of the induction machine in stator coordinates are used to obtain a full-order speed observer [3]. The induction machine equations can be expressed as state variable equations:

$$\frac{d}{dt}\begin{vmatrix}\bar{i}_s\\ \bar{\lambda}_r\end{vmatrix} = \begin{vmatrix}A_{11} & A_{12}\\ A_{21} & A_{22}\end{vmatrix}\begin{vmatrix}\bar{i}_s\\ \bar{\lambda}_r\end{vmatrix} + \begin{vmatrix}B_1\\ 0\end{vmatrix}\left|\bar{v}_s\right| \tag{3.15}$$

$$\bar{i}_s = \begin{vmatrix} i_d & i_q \end{vmatrix}^T \quad ; \quad \bar{v}_s = \begin{vmatrix} v_d & v_q \end{vmatrix}^T \tag{3.16}$$

$$\bar{\lambda}_r = \begin{vmatrix} \lambda_{dr} & \lambda_{qr} \end{vmatrix}^T \tag{3.17}$$

$$A_{11} = -\frac{R_r}{\sigma L_s}\left(1 + \frac{L_m^2}{L_r^2}\right)\cdot I \tag{3.18}$$

$$A_{12} = \frac{L_m}{\sigma L_s L_r}\left(\frac{1}{T_r}\cdot I - \hat{\omega}_r \cdot I'\right) \tag{3.19}$$

$$A_{21} = \frac{L_m}{T_r}\cdot I \tag{3.20}$$

$$A_{22} = \frac{1}{T_r}\cdot I + \hat{\omega}_r \cdot I' \tag{3.21}$$

$$I = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \quad ; \; I' = \begin{bmatrix} 0 & -1 \\ 1 & 0 \end{bmatrix} \tag{3.22}$$

$$\sigma = 1 - \frac{L_m^2}{L_s L_r} \tag{3.23}$$

The state observer, which estimates the stator current and the rotor flux together, is written as

$$\frac{d}{dt}\hat{X} = \hat{A}\ \hat{X} + B\ V_s + G\left(\hat{i}_s - i_s\right) \tag{3.24}$$

Where

$$\hat{X} = \left|\hat{i}_{ds} \quad \hat{i}_{qs} \quad \hat{\lambda}_{dr} \quad \hat{\lambda}_{qr} \quad \right|^T \tag{3.25}$$

$$\hat{A} = \begin{vmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{vmatrix}, \qquad B = \frac{1}{\sigma L_s} I \tag{2.26}$$

The error between the model current and the machine current $(\hat{i}_s - i_s)$ is used to generate corrective inputs to the dynamic subsystems of the stator and rotor [23]. Hence the observer gain G is selected so that (3.24) would be stable.

The speed estimator is based on rotor flux $\hat{\bar{\lambda}}_r$ and $\hat{\bar{i}}_s$ estimators

$$\hat{\omega}_r = \left(K_p + \frac{K_i}{s}\right) \mathrm{Imag}\left[\left(\bar{i}_s - \hat{\bar{i}}_s\right)\hat{\bar{\lambda}}_r\right] \tag{3.27}$$

3.9 Simulation of sensorless vector control

The simulation of the sensorless vector control is implemented in Simulink. The structure of the sensorless electric drive is shown in figure 3.26. The speed observer subsystem contains the adaptive speed observer model illustrated in figure 3.27. Figure 3.28 through figure 3.32 show speed, torque, current and position responses of the drive. The motor parameters are assumed to be constant so parameter variation effects are not investigated.

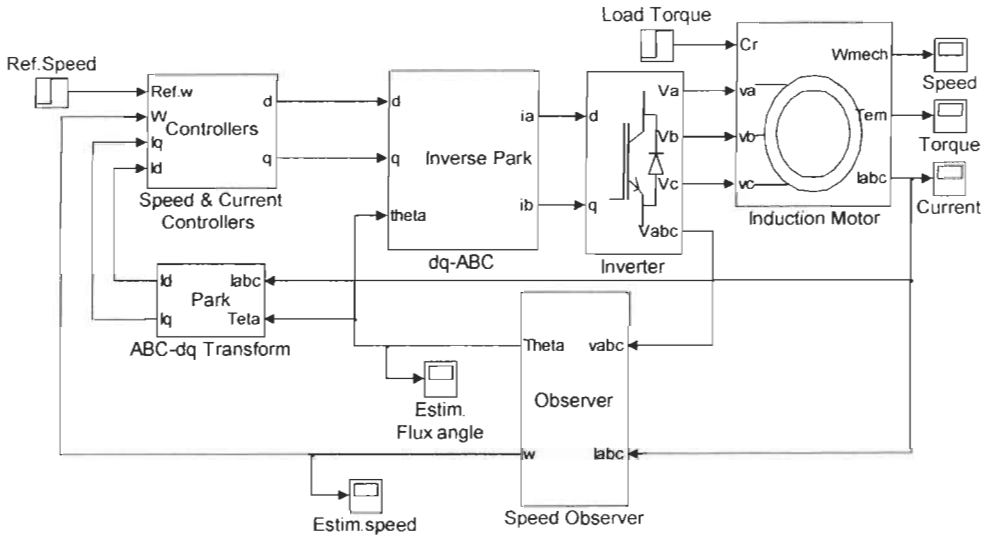


Figure 3.26 Overall sensorless control structure –Simulink model

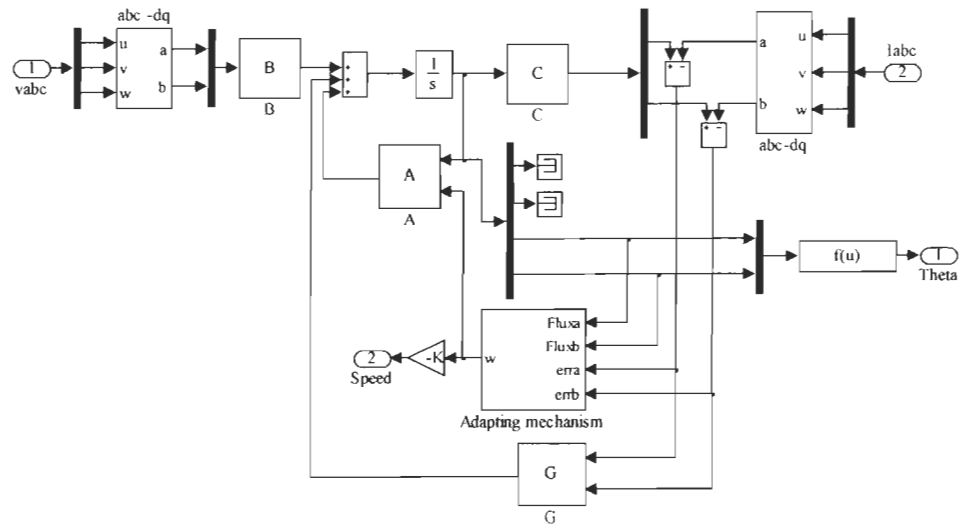


Figure 3.27 Adaptive speed observer-Simulink model

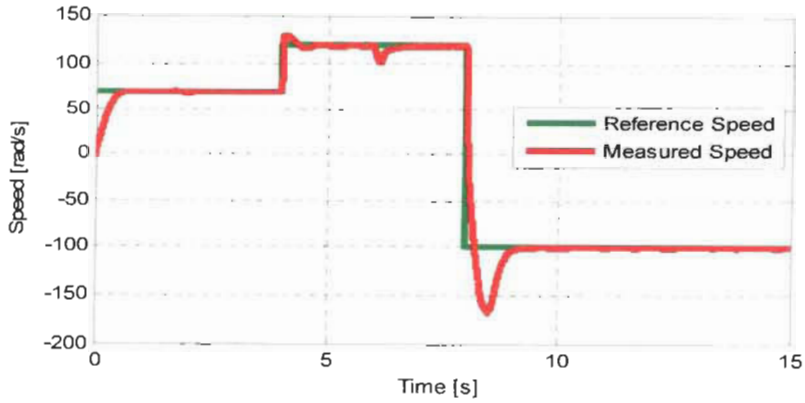


Figure 3.28 Sensorless vector control speed response

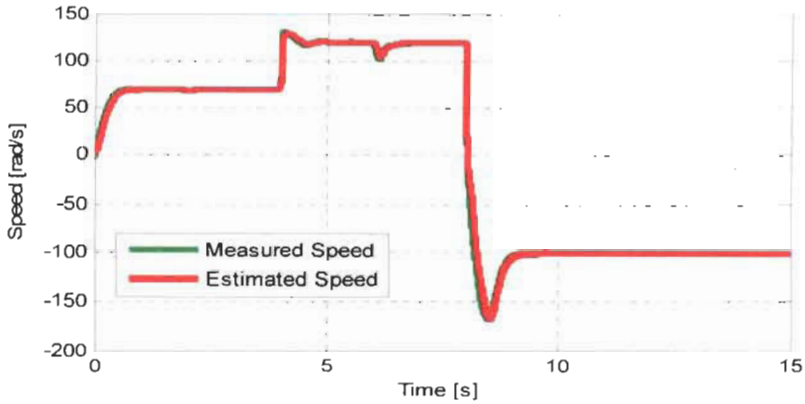


Figure 3.29 Sensorless vector control Estimated vs. Measured speed

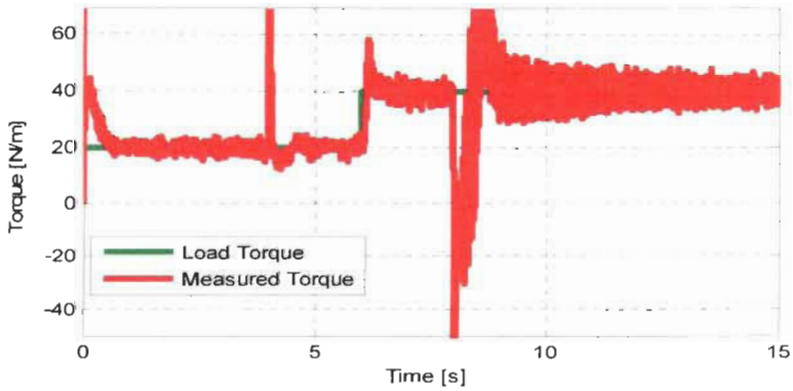


Figure 3.30 Sensorless vector control torque response

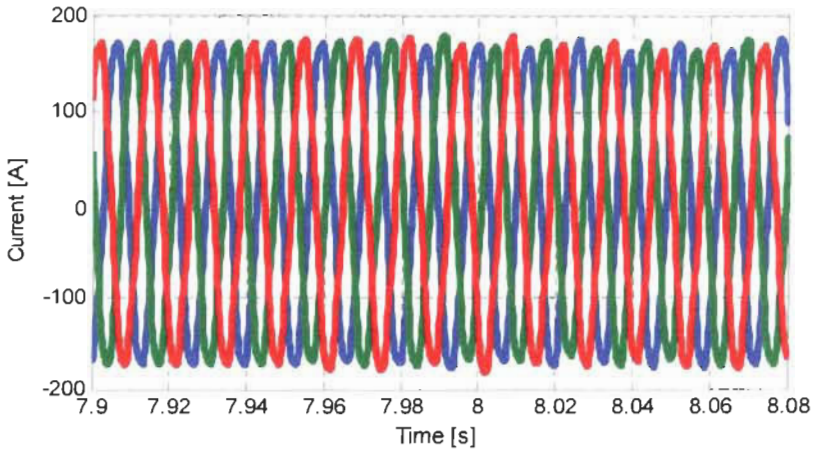


Figure 3.31 Sensorless vector control - Stator current

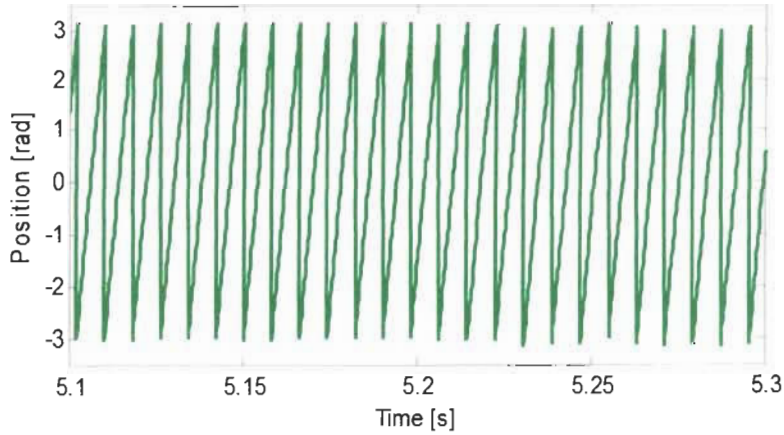


Figure 3.32 Sensorless vector control flux angle

The same operating conditions applied to the direct vector control in section 3.7 are applied to the sensorless vector control. The responses in both cases are identical.

3.10 Conclusion

This chapter presented several techniques used to control squirrel-cage induction machines. These techniques prove the controllability of the induction machine in terms of torque, speed and position. The appropriate control technique choice depends on the requested drive performance. If moderate performance fills the application requirements then scalar control techniques is quite enough. Otherwise vector control or direct torque control could be used. Both control techniques provide decoupled flux and torque control.

A senssorless vector control system for induction machine is presented. System modeling and simulation are detailed using Matlab/Simulink. Different flux and speed estimation techniques that could be used in high performance drives were described. Among the presented estimation techniques, the adaptive speed observer has better behaviour with respect to steady state error, parameter sensitivity and dynamic response although its performance at low speed remains problematic. Robust estimation techniques and parameter identification by self-commissioning or by online tuning have the potentiality of reducing the estimation errors. The majority of speed extraction techniques relay on the induction machine model.

Chapter 4 - Digital Filters

4.1 Introduction

An electrical filter is a system that can be used to modify, reshape or manipulate the frequency spectrum of an electrical signal according to some prescribed requirements. A filter may be used to amplify or attenuate a range of frequency components, reject or isolate one specific frequency component. Typically, an electrical filter receives an input signal or excitation and produces an output signal or response. Depending on the type of input, output and internal operating signals, three general types of filters can be identified, namely, continuous-time, sampled-data and discrete-time filters [24]. Depending on the format of the input, output and internal operating signals, filters can be classified either as analog or digital filters. In analog filters the operating signals are varying voltages and currents, whereas in digital filters they are encoded in some binary format. Digital filters are programmable, reliable and have better performance, in addition to the possibility of implementing complex combinations of filters with relatively simple hardware requirement. However, they are more expensive compared to analog ones. Continuous-time and sampled-data filters are always analog filters, but discrete-time filters can be analog or digital [24, 25]. A digital filter is used in the implementation of the sensorless vector control to eliminated measurement noise.

Figure 4.1 shows a digital filter where analog to digital converters are necessary to digitize the unfiltered analog inputs [26]. Then, signal processors are used to implement digital filters by performing the necessary mathematical operations.

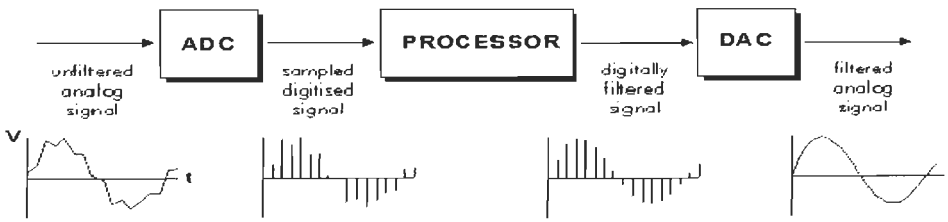


Figure 4.1 Digital filter block diagram

4.2 Types of digital filters

Mainly, digital filters are classified into two types according to filter characteristics as shown below [27].

4.2.1 Finite impulse-response (FIR)

It is a linear phase filter characterized by a unit-sample response (h_n) that has a finite duration.

$$\begin{aligned} h_n &= 0, \quad \text{for} \quad \infty > n > N_1 \quad \rightarrow N_1 < n < \infty \\ h_n &= 0, \quad \text{for} \quad -\infty < n < N_2 \quad \rightarrow -\infty < n < N_2 \\ &\text{where} \quad N_1 > N_2. \end{aligned} \tag{4.1}$$

4.2.2 Infinite impulse-response (IIR)

This term means that the duration of the filter impulse response (h_n), is infinite.

4.2.3 Recursive realization

This term describes the way a filter (either IIR or FIR) is realized. It means that the current filter output y_n is obtained explicitly in terms of past filter outputs (y_{n-1}, y_{n-2}, ...), as well as in terms of past and present filter inputs (x_n, x_{n-1}, ...). Thus the output of a recursive realization can be written as

$$y_n = F(y_{n-1}, y_{n-2}, \ldots\ldots\ldots, x_n, x_{n-1}, \ldots\ldots\ldots\ldots) \tag{4.2}$$

4.2.4 Non-recursive realization:

This term means that the current filter output y_n is obtained explicitly in terms of only past and present inputs; i.e., previous outputs are not used to generate the current output. The representation of a non-recursive realization can be written as

$$y_n = F(x_n, x_{n-1}, \ldots\ldots\ldots\ldots) \tag{4.3}$$

4.3 FIR advantages

In this application a FIR realized non-recursively is selected for the following reasons [27]:

- FIR filters can easily be designed to approximate a prescribed magnitude frequency response to arbitrary accuracy with an exactly linear phase characteristic.
- FIR filter realized non-recursively contains only zeros in the finite z plane, and hence is always stable.
- The coefficient accuracy problems inherent in sharp cut-off IIR filters can often be made less severe for realizations of equally sharp FIR filters.

In general, an Mth order FIR filter is described by a difference equation:

$$y(n) = b_0 x(n) + b_1 x(n-1) + ... + b_M x(n-M) \tag{4.4}$$

Or equivalently, by the system function:

$$H(z) = b_0 + b_1 z^{-1} + ... + b_M z^{-M} \tag{4.5}$$

The signal flow diagram of the FIR filter, representing the difference equation (4.4) is shown in figure 4.2.

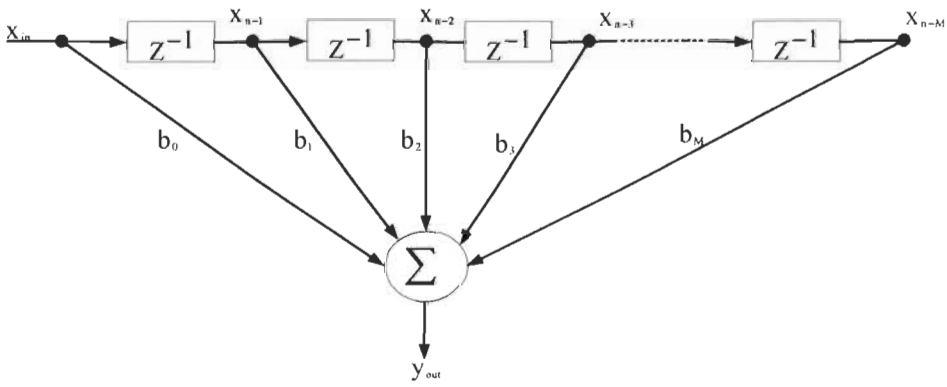


Figure 4.2 FIR filter diagram

The FIR filter is essentially a sum of products operating on an array of values maintained in a delay line. The number of filter coefficients will always be greater than the order of the filter by one [28].

4.4 Filter design

Basically, this part deals with the appropriate choice of coefficients in the filter's transfer function to approximate an ideal or desired response. The following three steps summarize the filter design process as detailed in Matlab documentation (Digital Filter Design). A flow chart that summarizes the design steps is shown in figure 4.3.

4.4.1 Response

In this application the aim is to eliminate the noise associated with the motor current by blocking frequencies greater that the fundamental frequency. Therefore a low pass response is selected.

4.4.2 Specification

This term is meant to identify the various design parameters, these parameters depend on the filter response. The general specifications for a low-pass filter are passband-edge frequency, stopband-edge frequency, passband ripple, stopband attenuation and filter order.

4.4.3 Design method

The design method (algorithm) selection depends on the filter response and the design parameters. It is clear that each algorithm requires a distinct specification string. In this case, a window algorithm is chosen for the low-pass response, thus the specification set consists of filter order (N), cut-off frequency (Fc) and stop frequency (Fs). The three most popular FIR filter design methods are listed as follows.

- **Parks-McClellan**

This is a general-purpose computer program, which is capable of designing a large class of optimum FIR linear phase digital filters [29].

- **Windowing**

Windowing is one of the earliest techniques for designing FIR filters. In this method, an initial impulse response is derived by taking the Inverse Discrete Fourier Transform (IDFT) of the desired frequency response. Then, the impulse response is refined by applying a data window to it. Several types of windows are available

- **The frequency-sampling method**

This method is fast and simple. It is useful for adaptive filters or for an intermediate stage in a more complicated algorithm where speed is important. For linear phase filters, direct design with the frequency-sampling method is possible by applying the Inverse Discrete Fourier Transform (IDFT) to equally spaced samples of the frequency response $H(\omega)$ [30].

Once the required filter parameters are carefully selected as detailed above, the filter design and analysis tool (FDATool) provided by Matlab could be used to obtain the filter coefficients and analyze the filter response. Figure 4.4 shows the graphical user interface (GUI) of the tool where the user is prompted to select the filter response, specifications and algorithm. The GUI is invoked by entering (fdatool) in the Matlab command window. The tool generates a C-header file containing the designed filter coefficients, which facilitates the implementation in C-program environment. The designed filter could be analyzed using various graphical analysis options provided by the tool to ensure that the filter complies with all design criteria. In figure 4.4, the magnitude response of a 60 Hz low pass filter is shown. Step and impulse responses shown respectively in figures 4.5 and 4.6, could be visualized from the analysis drop down menu.

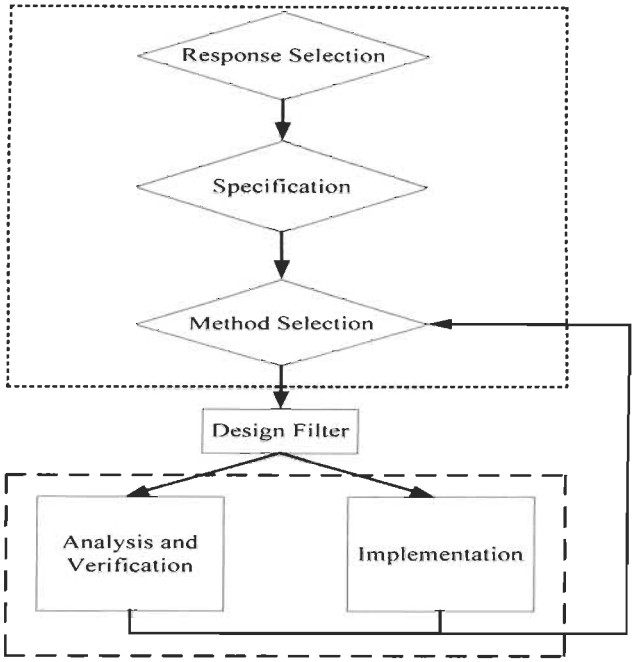


Figure 4.3 Filter design flow chart

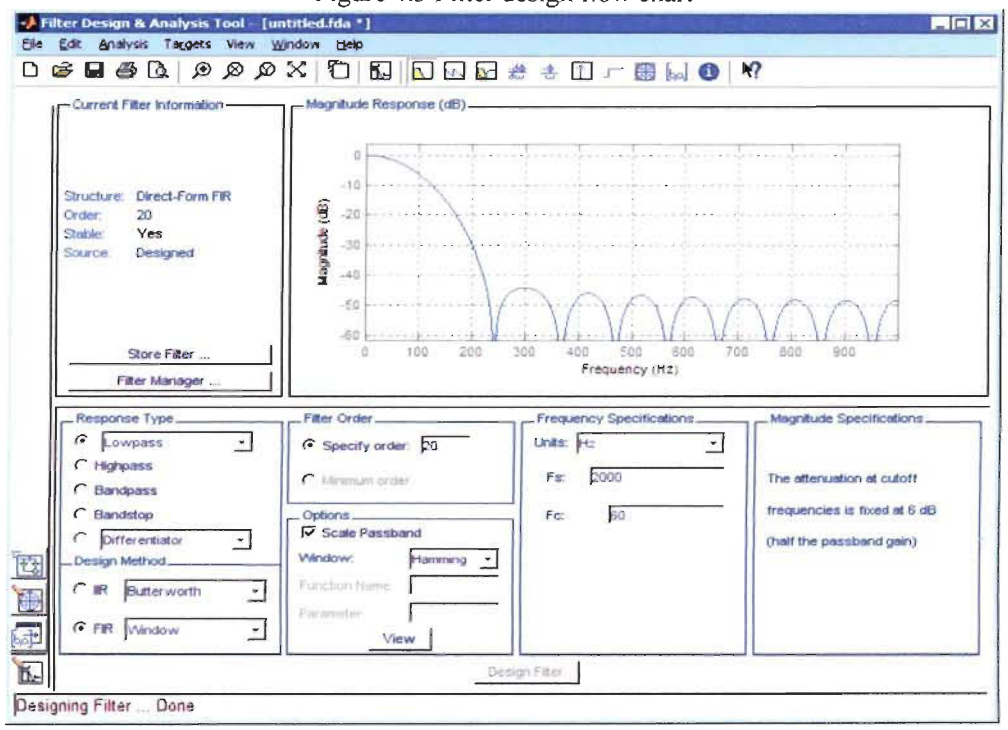


Figure 4.4 FDATool Graphical User Interface

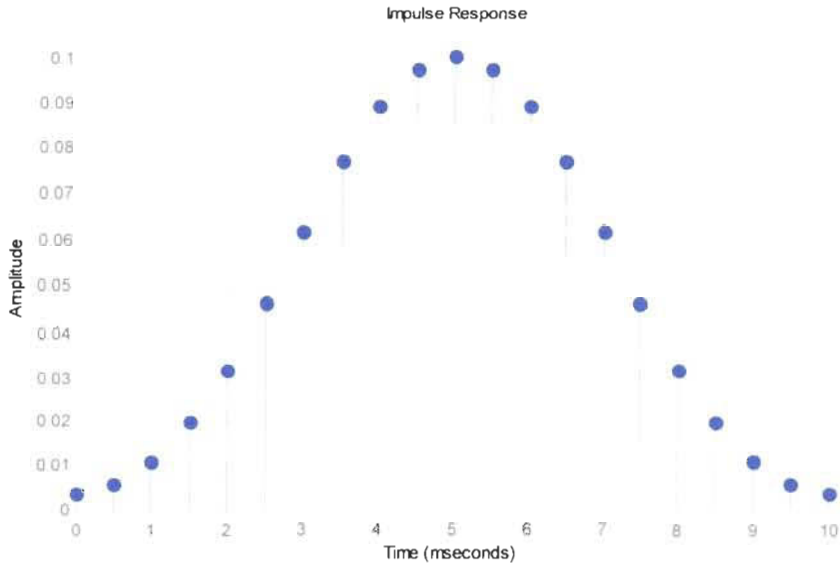


Figure 4.5 Impulse response

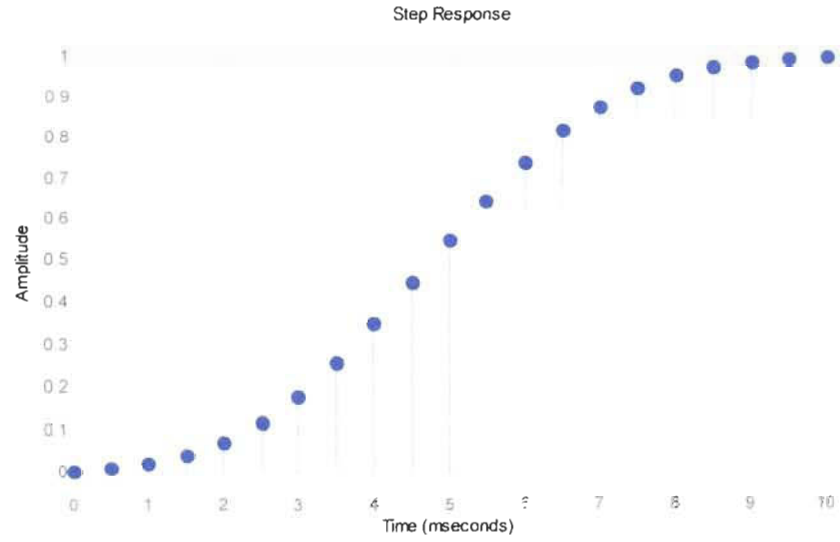


Figure 4.6 Step response

4.5 FIR Implementation

Texas Instrument provides cost-free resources to implement digital filters. These resources include target independent C based algorithms and optimized assembly language ones intended to be used in C28X processor family. The assembly coded functions could be used in a C-based program. Those functions could be called to the program as any ordinary C subroutine by using the interface structure. The filter library of Texas Instrument contains a 16-bit FIR filter module that implements FIR filter using Direct Memory Access Controller (DMAC) instructions that effectively executes two filter taps in a cycle. This module can support up to 55th order FIR filter [28].

- The first step to implement the FIR filter is to add the computation function written in assembly language (.asm) to the project as a source file, then put the coefficients calculated from the design process in the filter header file (.h) as shown below. A filter of 50^{th} order is used in this case, so only the first 26 coefficients are placed in the header file since the remaining 25 set of coefficients is a mirror image of the first 25 set.

```
#define FIR16_COEFF {\
        1210,6292661,6620328,7472274,8848501,10814543,13304867,\
        16188400,19530679,23266170,27263801,31523573,35979952,\
        40501866,45089317,49611233,54067616,58327395,62325034,\
        66060535,69402827,72286373,74776711,76677234,78119013,78970976}
```

- Secondly, memory sections should be created in the pre-configured linker command file (F2812_EzDSP_RAM_lnk.cmd). In this file, the peripheral memory regions defined and assigned to the individual peripheral. The following code lines create data sections for two filter modules (instances) and delay buffers.

```
DbufB           align(0x100)> LOL1RAM,    PAGE = 1
filB            : > LOL1RAM,      PAGE = 1
bufA            align(0x100)> RAMM1,      PAGE = 1
AfilA           : > RAMM1,      PAGE = 1
```

- Thirdly, the required number of filter variables (instances), the data sections and coefficient array are declared at the beginning of the program as shown in the following lines of code.

```
#define FIR_ORDER    50
 // Create Instances of FIRFILT_GEN module and
 // place the object in "AfilA" & "filB" section
#pragma DATA_SECTION(filtA,"AfilA");
   FIR16 filtA= FIR16_DEFAULTS;
#pragma DATA_SECTION(filtB, "filB");
   FIR16 filtB= FIR16_DEFAULTS;
 // Define the Delay buffers for the 50th order filter and
// place it in "bufA"& "DbufB" section
#pragma DATA_SECTION(dbuffer1,"bufA");
 long dbuffer1[(FIR_ORDER+2)/2 ];
#pragma DATA_SECTION(dbuffer2,"DbufB");
 long dbuffer2[(FIR_ORDER+2)/2 ];
// Define Constant Co-efficient Array  and
// place the .constant section in ROM memory
 const long coeff[(FIR_ORDER+2)/2]= FIR16_COEFF;
```

- Fourthly, the filter modules are initialized in the main program. The following code initializes two filter modules for phase (A) and phase (B) currents.

```
//--------Phase A filter
  filtA.coeff_ptr=(long *)coeff;
  filtA.dbuffer_ptr=dbuffer1;
  filtA.order= FIR_ORDER;;
  filtA.init(&filtA);
//--------Phase B filter
  filtB.coeff_ptr=(long *)coeff;
  filtB.dbuffer_ptr=dbuffer2;
  filtB.order= FIR_ORDER;
  filtB.init(&filtB);
```

- Finally, the outputs from the ADC converter are passed as inputs to the two filter modules and the filter computation functions are called as shown below.

```
//---------------Phase A filter-------------------
        filtA.input= ilg2_vdc1.ImeasA ;
        filtA.calc(&filtA);

//---------------Phase A filter-------------------
      filtB.input= ilg2_vdc1.ImeasB;
       filtB.calc(&filtB);
```

4.6 Conclusion

This chapter provided an introduction to digital filters with an insight to filter design and implementation problem. A practical filter design tool provided by Matlab is presented along with filter responses that contain complete information about the filter. A Texas Instrument module that helps implement finite impulse response filter is introduced. This module is used in the sensorless vector control algorithm to implement a low pass filter that filters the motor currents. The effect of frequency variation is not investigated.

Chapter 5 - DSP based Implementation

5.1 Introduction

The rapid advancements made in the field of microprocessors and digital signal processors (DSP) have contributed significantly to the development of various high-performance alternating current electric drives since it has become possible to introduce relatively complicated control algorithms in these drives [1]. Nowadays, manufacturers provide a wide range of enhanced application oriented digital signal processors along with development boards to evaluate certain characteristics of a specific DSP. An evaluation module named eZdspTM is used in this application. The eZdspTM F2812 is a stand-alone card that serves as an excellent platform to develop and run software for the TMS320F2812 processor.

The eZdspTM F2812 allows full speed verification of F2812 code. Two expansion connectors are provided for any necessary evaluation circuitry not included in the basic circuit configuration. To simplify code development and shorten debugging time, a C2000 Tools Code Composer driver is provided. In addition, an onboard JTAG connector provides interface to emulators, operating with other debuggers to provide assembly language and 'C/C++' high level language debug. Figure 5.1 shows the block diagram of the eZdspTM module [31].

5.1.1 Key Features of the eZdspTM F2812

The eZdspTM F2812 has the following features:

- TMS320F2812 Digital Signal Processor;
- 150 MIPS operating speed, 30 MHz. clock;
- 18K words on-chip RAM;
- 128K words on-chip Flash memory, 64K words off-chip SRAM memory;
- 2 Expansion Connectors (analog, I/O);
- Onboard IEEE 1149.1 JTAG Controller and emulation connector;
- 5-volt only operation with supplied AC adapter;
- TI F28xx Code Composer Studio tools driver;

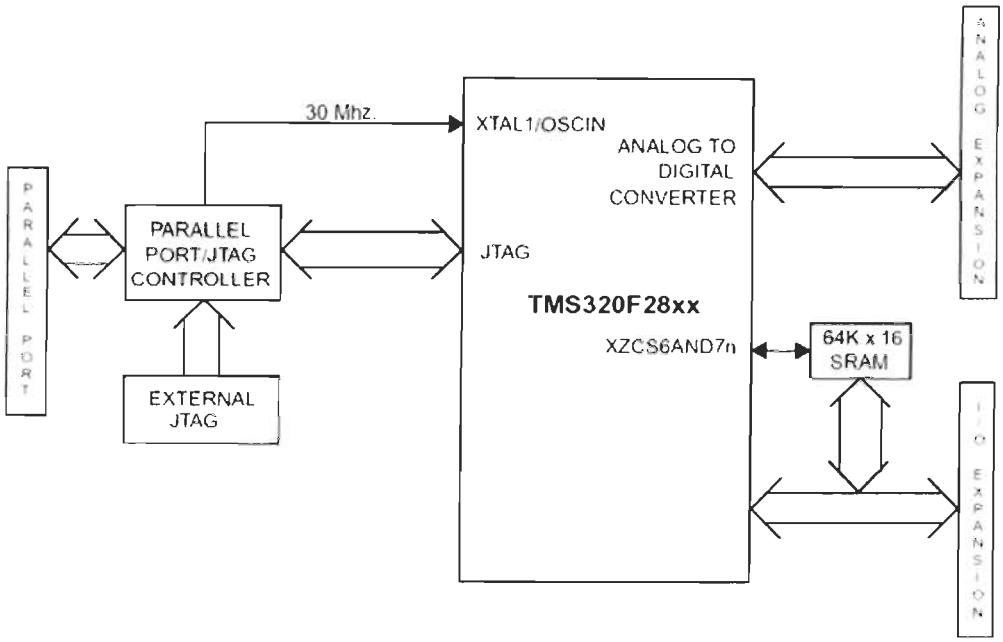


Figure 5.1 eZdsp F2812 block diagram

5.2 Code Composer Studio

Code composer studio (CCS) is an integrated development tool provided by Texas Instruments (TI) that allows the realization of some crucial steps of the digital signal processor based applications design process. Once the application is designed and the code is developed, code composer studio is used to code/ build, debug, analyze, tune, and on to test. There are various useful windows as shown in figure 5.2. Upon CCS start, project view and code windows are the default ones, the project view window displays the files added to the project in a tree format while the code window displays the text editor. Other debugging windows such as output and watch windows could be added to the development environment. The output window shows compile errors; meanwhile watch window allows variable viewing during program run. Figure 5.2 shows a screen shot of CCS development environment [32].

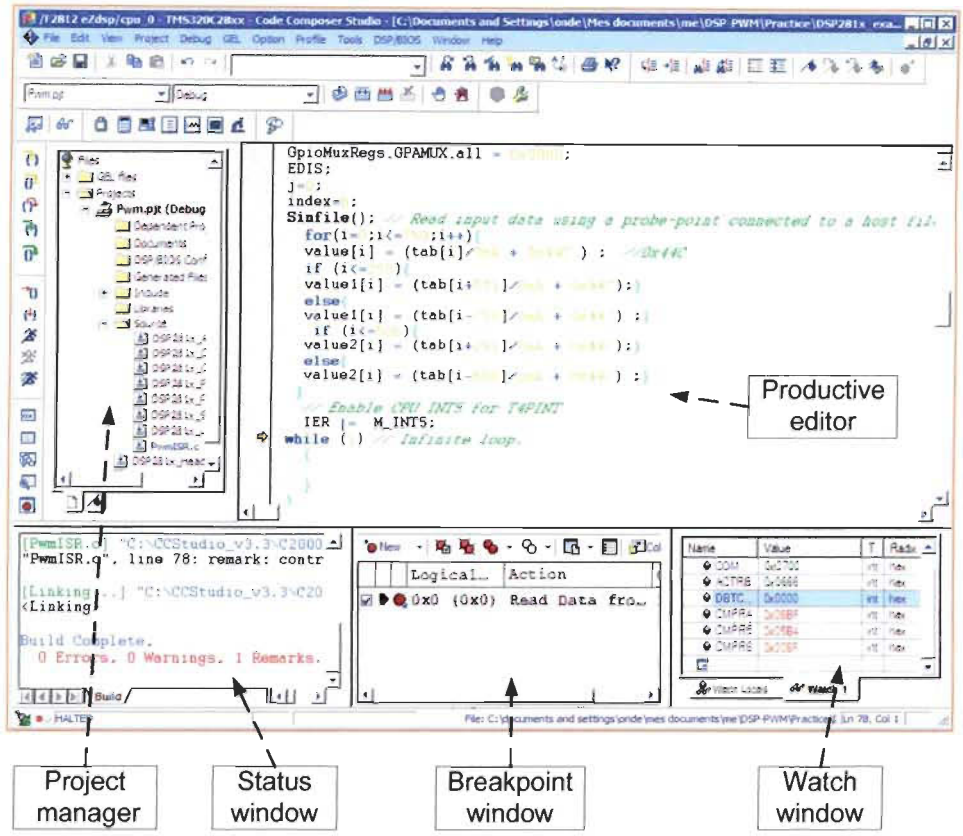


Figure 5.2 Code Composer Studio

Once the code composer studio installed in a personal computer (PC), the connection with the **eZdsp** board could be established via a standard parallel port since the F2812 implements the standard Joint Test Action Group (JTAG) interface (IEEE 1149.1 JTAG interface). Additionally, the F2812 supports real-time mode of operation whereby the contents of memory, peripheral, and register locations can be modified while the processor is running and executing code and servicing interrupts [31].

5.3 Generation of sine modulated pulse width modulation signals

Traditionally, three phase inverters are used to power ac drives. Three commonly used Pulse Width Modulation (PWM) techniques include sinusoidal, hysteresis (bang-bang), and space-vector (symmetrical or asymmetrical) implementations. Sinusoidal PWM is an effective method to generate sinusoidal currents. Sinusoidal PWM is obtained by comparing a high-frequency carrier with a low-frequency sinusoid, which is the modulating or reference signal. The carrier has a constant period; therefore, the switches have constant switching frequency. The switching instants are determined from the crossing of the carrier and the modulating signal [33]. The carrier can have a triangular (or saw tooth shaped) waveform. The three mentioned PWM techniques generate periodic rectangular waveforms with varying duty cycle. The basic idea is to control the duty cycle of a switch such that a load sees a controllable average voltage. To achieve this, the switching frequency (repetition frequency for the PWM signal) is chosen high enough so that the load cannot follow the individual switching events [34, 35].

Today's programmable DSPs provide capabilities that make generating advanced PWM signals easier than previous techniques. Using a DSP makes it easy to change carrier frequency and PWM scheme simply through reprogramming. In addition, they allow generation of three pairs of complementary PWM waveforms with programmable dead bands for three-phase voltage-source inverters. The combination of these features helps reduce the number of chips needed to implement the entire circuit, increasing reliability and lowering the overall cost of the system [36].

In this section a flexible DSP-based algorithm to generate sine wave PWM signals is introduced. Algorithm development process, flowchart and implementation results are presented. The algorithm implementation and the experimental setup serve as a platform to test the three phase inverter and the ancillary circuits that will be used later in the vector control implementation.

5.3.1 DSP SIGNAL GENERATION METHODOLOGY

The event-manager (EV) modules are the main DSP component for the PWM signal generation. They provide a broad range of functions and features that are particularly useful in motion control and motor control applications. The EV modules include 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 applications. Each EV is capable of controlling three half-H bridges, when each bridge requires a complementary PWM pair for control. Each EV also has two additional PWMs with no complementary outputs [37].

5.3.2 Code generation

A general purpose (GP) timer, which repeats a counting period equal to the PWM period, is used to generate the triangular carrier signal. A compare register is used to hold the modulating sine wave values read from a data file. The value of the compare register is constantly compared with the value of the timer counter. When the values match, a transition happens on the associated output. When a second match is made between the values, or when the end of a timer period is reached, another transition happens on the associated output. In this way, an output pulse is generated whose on (or off) duration is proportional to the value in the compare register. This process is repeated for each timer period with different values in the compare register. As a result, a PWM signal is generated at the associated output [8].

5.3.2.1 PWM modulating signal

A look-up table that contains the calculated sine values for one phase is stored in a data file (file.dat). Table length plays an important role in characterizing the inverter output waveforms therefore, more entries results in less deformed output waveforms. However, this choice is restricted by DSP memory capacity. Generation of the three sine wave modulating signals based on the read data file is detailed in sub-section DSP signal generation flow chart of this section. Figure 5.3 shows a graph of the three sine wave modulating signals generated by the program.

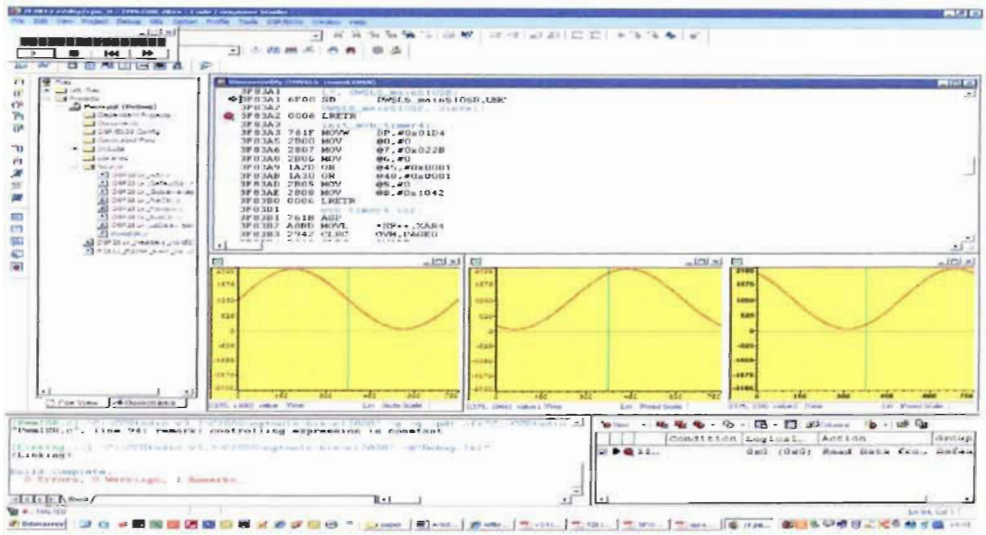


Figure 5.3 Three phase sine wave modulating signals

5.3.2.2 PWM carrier signal

In this application Event Manager Module (EVB) and General Purpose Timers 3 and 4 (GP timers 3 and 4) of the DSP were used to generate asymmetrical PWM waves. GP timer 3 is configured as follows [37]:

- The timer period register T3PR is set to 2500 (desired 10 kHz PWM frequency);
- The timer control register T3CON is set to 0x1042 (counting mode, clock source).

A continuous up-counting mode is generally used to generate asynchronous PWM.

The timer period value is calculated as follows:

$$Timer\,3\,Period = \frac{SYSCLKOUT}{2 \times Clock\ \ Prescaler \times PWM\ Frequency} \qquad 5.1$$

where: *SYSCLKOUT* is the CPU clock frequency (150 MHz for the TMS320F2812 DSP)

$$Timer\,3\,Period = \frac{150\,MHz}{2 \times 3 \times 10\,KHz} = 2500$$

For a 5 kHz PWM frequency, the period value for GP timer 3 is recalculated and the new value is written to the period register during the program run. It is also possible to modify the switching frequency by means of a digital input.

GP timer 4 is used to modify periodically the compare register in order to obtain the desired frequency of the inverter output. In this application the compare registers were updated each timer interrupt to obtain the 60 Hz inverter output voltage. The interrupt is generated upon period match. The period value for this timer is calculated as follows:

$$Timer4\ Period = \frac{SYSCLKOUT}{2 \times Clock\ Prescaler \times output\ frequency \times number\ of\ Sine values} \quad 5.2$$

$$Timer\ 4\ Period = \frac{150\ MHz}{2 \times 3 \times 60 \times 751} = 555$$

Timer 4 period register value represents the step required to load the next value from the sine table. Modifying this period register value results in changing the inverter output frequency without the need to change the frequency of the modulating signal as in the conventional sine PWM generation.

5.3.2.3 DSP signal generation flowchart

The implemented code flowchart is shown in Figure 5.4. The first part of the algorithm (Initialize DSP block) initializes System Control registers, Clocks and Peripheral Interrupt Expansion (PIE) control registers to their default state. This is achieved by calling the following functions:

```
InitSysCtrl ();
InitPieCtrl ();
```

The high speed peripheral clock prescaler is then set to 3 to adjust its frequency to 25 MHz.

```
EALLOW;
SysCtrlRegs.HISPCP.all = 0x3;
EDIS;
```

Action control settings of the three compare units of event manager B in addition to enabling its associated six PWM output pins are done as follows (Enable PWM block).

```
EvbRegs.ACTRB.all = 0x0666;
EvbRegs.COMCONB.all = 0xC600;
```

EALLOW;

GpioMuxRegs.GPBMUX.all = 0x00FF;

EDIS;

GP timer 3 period and control register are initialized as follows:

EvbRegs.T3PR =0x9C4;

EvbRegs.T3CON.all = 0x1042;

Modifying this timer's period value will change the generated PWM switching frequency. Up-counting mode is selected in both timers by setting the appropriate control register's bit.

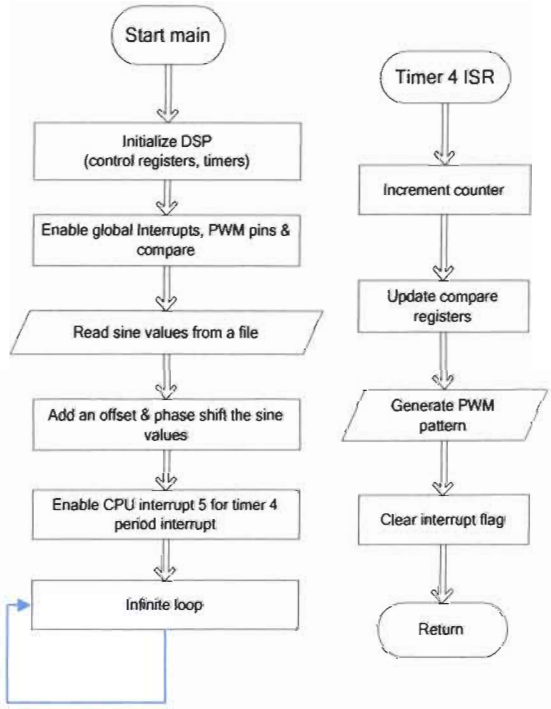


Figure 5.4 DSP signal generation flowchart

Inverter output voltage frequency is fixed by setting the Period for GP timer 4 and enabling Period interrupt bits.

EvbRegs.T4PR = 0x022B;

EvbRegs.T4CON.all = 0x1042;

```
EvbRegs.EVBIMRB.bit.T4PINT = 1;
EvbRegs.EVBIFRB.bit.T4PINT = 1;
```

A timer 4 interrupt service routine (ISR) is used to update compare registers, where a new sine value from the three different look-up tables is assigned to its corresponding compare register each interrupt. The interrupt is triggered at timer 4 period match. At the end, the interrupt is acknowledged to receive more interrupts from PIE group 5 and the flag is cleared. The counter (index) is reset every 751 interrupts where the number (751) defines the look-up table length.

The ISR is shown below:

```
Interrupt void evb_timer4_isr (void)
{
 index++;
 EvbRegs.EVBIFRB.all = BIT0;
 EvbRegs.CMPR4 = value [index];
 EvbRegs.CMPR5 = value1 [index];
 EvbRegs.CMPR6 = value2 [index];
 PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;
 if (index≥751)
  {
   index=0;
    }
 EvbRegs.EVBIFRB.bit.T4PINT = 1;
 }
```

Before building the project in code composer studio environment the source and General Extension Language (gel) files listed below were added to the program.

DSP28_SysCtrl.c

DSP28_PieCtrl.c

DSP28_PieVect.c

DSP28_usDelay.asm

f2812.gel

f2812_peripheral.gel

These files in addition to initialization and interrupt functions used in this application are provided by Texas Instruments.

After building and loading the project to the DSP, a break point is set at the beginning of the sine function. At first run, the program execution halts then breakpoint action is changed to ''read from file'' instead of ''halt program'', this is done in the breakpoint window. The data file location is specified in a dialogue message in addition to the starting address and the length of look-up table.

An offset is added to the sine values read from the data file and the result is considered as phase (A) values. The offset value depends on the amplitude of the sine wave. The modulating sine waves for the other two phases are obtained by shifting phase (A) values by 500 and 250 points for phase (B) and phase (C) respectively.

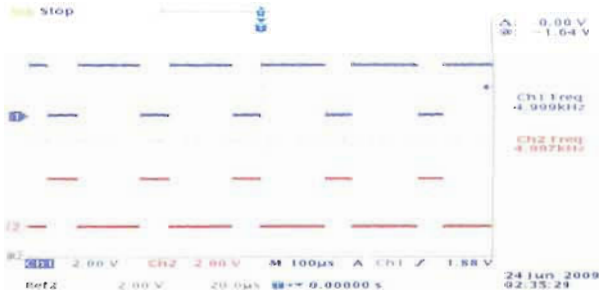
The following lines of code generate the three modulating signals.

```
for (i=0; i<=750; i++)
{
Value [i] = (Val [i]/0xA + 0x44C);
 if (i<=250)
  {
 value1 [i] = (Val [i+501]/0xA + 0x44C);
  }
else {
 value1 [i] = (Val [i-250]/0xA + 0x44C);
      }
if (i<=500)
    {
 value2 [i] = (Val [i+251]/0xA + 0x44C);
    }
else {
 value2 [i] = (Val [i-500]/0xA + 0x44C);
      }
}
```

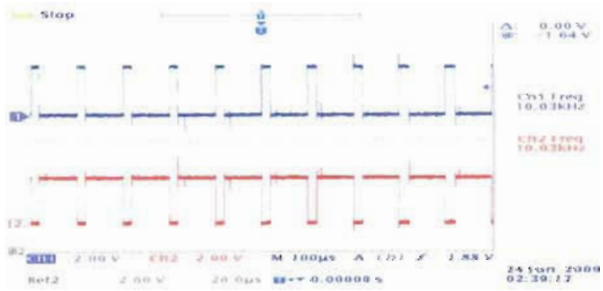
5.3.3 Experimental results

Fig. 5.5 illustrates a single sequence of the generated phase (A) PWM patterns for two different switching frequencies 5 kHz (Figure 5.5a) and 10 kHz (Figure 5.5b). The amplitude of the generated three phase signals is 3.3V so they are amplified and sent to the three phase inverter's IGBT driving circuit, which operates at 12V. Resistors (30Ω) in series with inductors (30 mH) are used as inverter load. The DC bus voltage is 115 V, and the load is delta connected. Fig. 5.6 and Figure 5.7 illustrate respectively the inverter output voltage and current waveforms for 5 kHz and 10 kHz frequencies. It is clear that by increasing the IGBT switching frequency (Figure 5.6b and Figure 5.7b), we obtain less deformed waveforms and greater output amplitudes for the same inverter dc voltage input, of course at the expense of increasing switching losses.

Figure 5.8, 5.9 show the harmonic content of the inverter output voltage (Figure 5.8) and current (Figure 5.9) for 5 kHz and 10 kHz switching frequencies. Again we notice that the total harmonic distortion (THD) is substantially reduced upon switching frequency augmentation (Figure 5.10).

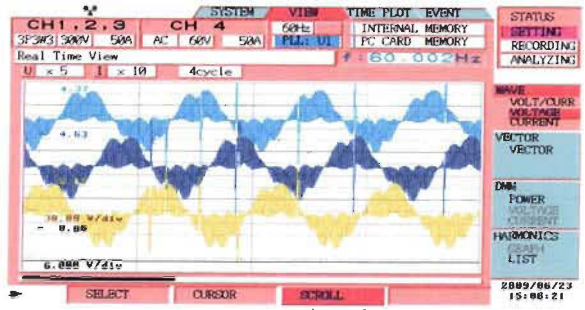


a) 5 kHz



b) 10 kHz

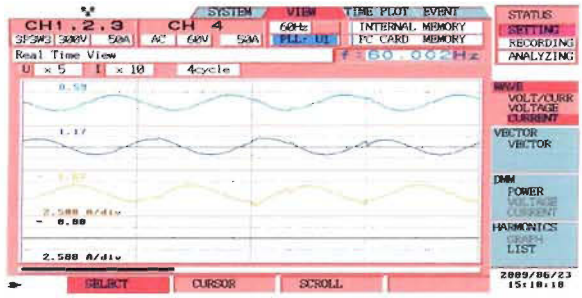
Figure 5.5 PWM patterns at 5 kHz and 10 kHz



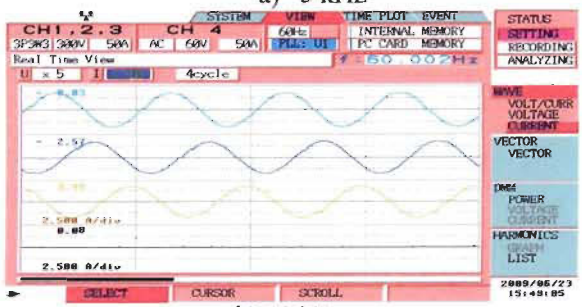
a) 5 kHz

b) 10 kHz

Figure 5.6 Voltage waveforms at 5 kHz and 10 kHz

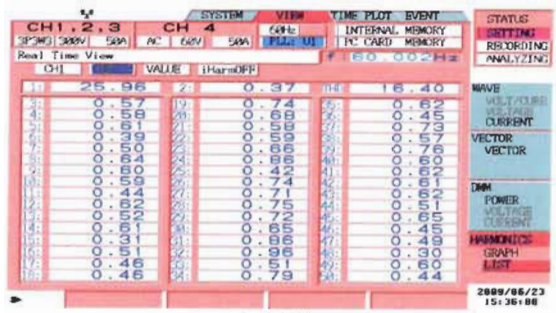


a) 5 kHz



b) 10 kHz

Figure 5.7 Current waveforms at 5 kHz and 10 kHz

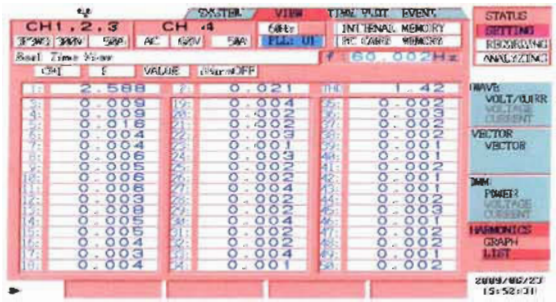


a) 5 kHz

b) 10 kHz

Figure 5.8 Voltage harmonics values at 5 kHz and 10 kHz

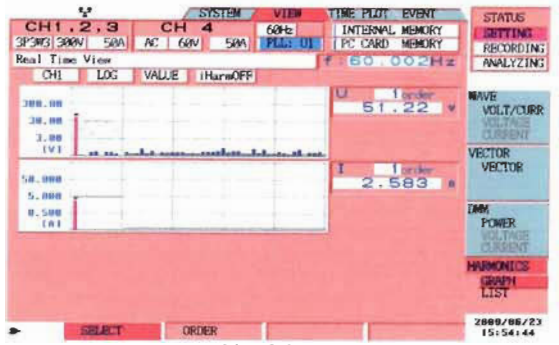
a) 5 kHz



b) 10 kHz

Figure 5.9 Current harmonics values at 5 kHz and 10 kHz

a) 5 kHz



b) 10 kHz

Figure 5.10 Voltage and current harmonics spectra at 5 kHz and 10 kHz

5.3.4 Discussion

This section presented an algorithm to generate sine modulated PWM signals for three phase inverters using Texas Instruments TMS320F2812 DSP. Step-by-step program development, algorithm and flowchart are detailed. The sinusoidal PWM generation algorithm is written in C language so it can be reused easily, in addition to the flexibility it provides in terms of changing the fundamental frequency of the inverter output voltage. Output voltage and current's total harmonic distortions were measured and compared for 5 kHz and 10 kHz IGBT switching frequency. Experimental results demonstrate the algorithm functionality and the validity of the experimental setup for three phase inverter applications including electric drives.

5.4 Sensorless vector control

This section discusses in detail the hardware and software used to implement sensorless vector control. This implementation is mainly based on the "C/C++" real control framework to demonstrate sensorless vector control demo provided by Texas Instruments[38]. The overall system is shown in figure 5.11. The eZdsp board generates six PWM signals by means of space vector technique. These PWM signals are adapted to the inverter driver circuit (12V) by using the interface circuit. The three phase inverter supplies the induction motor with a three phase AC voltage having the appropriate magnitude and frequency to control the motor speed. The DC-bus voltage is measured and used along with the switching states to reconstruct the three phase voltage. Motor two phase currents are measured using current sensors and then these currents are adjusted with respect to the analog to digital converter input limits using the current amplification circuit. The implemented structure consists of the following components:

1. TMS320F2812 eZdsp platform
2. Three phase induction machine (1/3 hp)
3. Semikron MINI8 three phase inverter
4. Current amplification circuit
5. PWM interface circuit
6. Lab-volt power supply (model 8525)
7. Auxiliary DC supply (5, 12, 24 VDC)
8. PC with code composer studio installed
9. Measurement instruments (multi meter, oscilloscope)

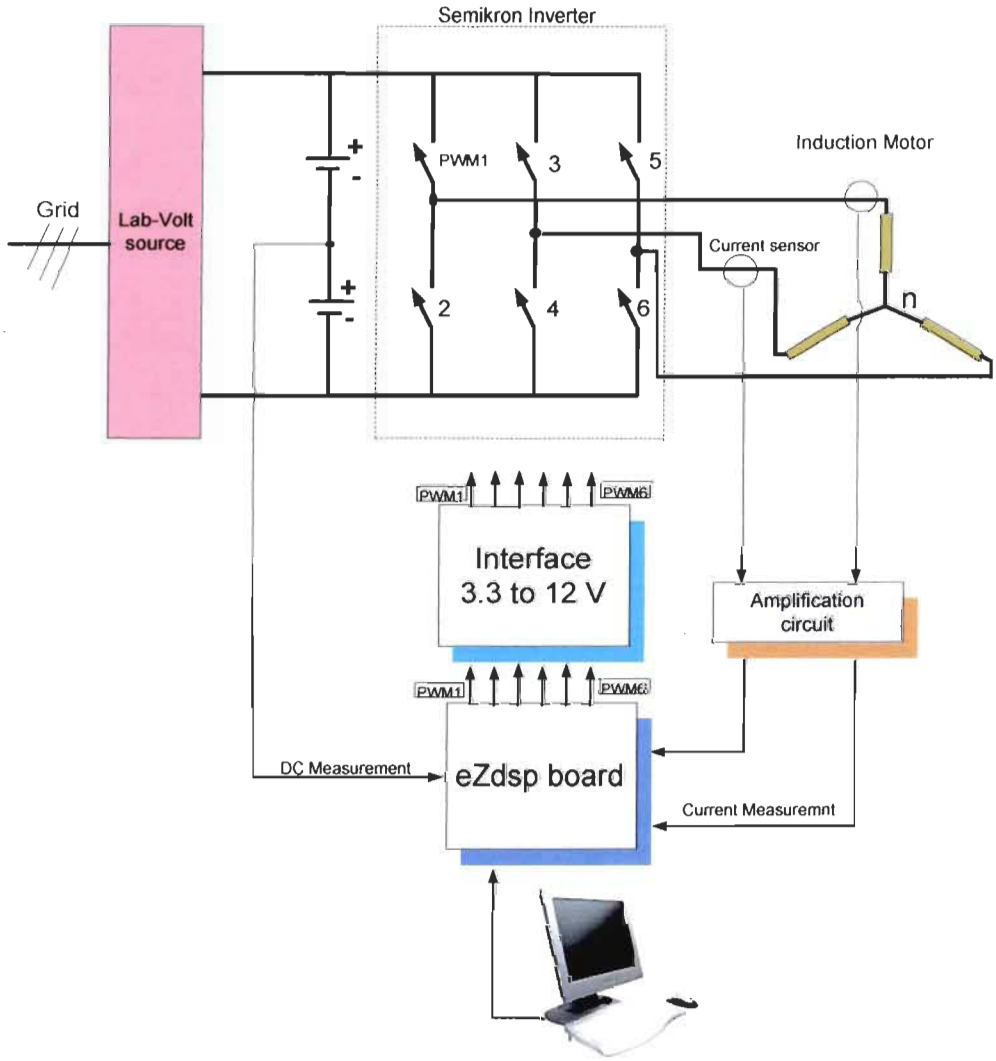


Figure 5.11 Hardware configuration

5.4.1 PWM interface circuit

The DSP generates six 3.3 V PWM signals, these signals could not be sent directly to the inverter since the IGBT gate driver control signals threshold is 12V. Therefore, an interface circuit to boost the voltage level is necessary. The interface circuit is obtained by cascading two NOT gate with two different voltage levels (3.3-12 V). A 7405 hex inverter integrated circuit is used for this purpose. The schematic drawing of the interface circuit is illustrated in figure 5.12.

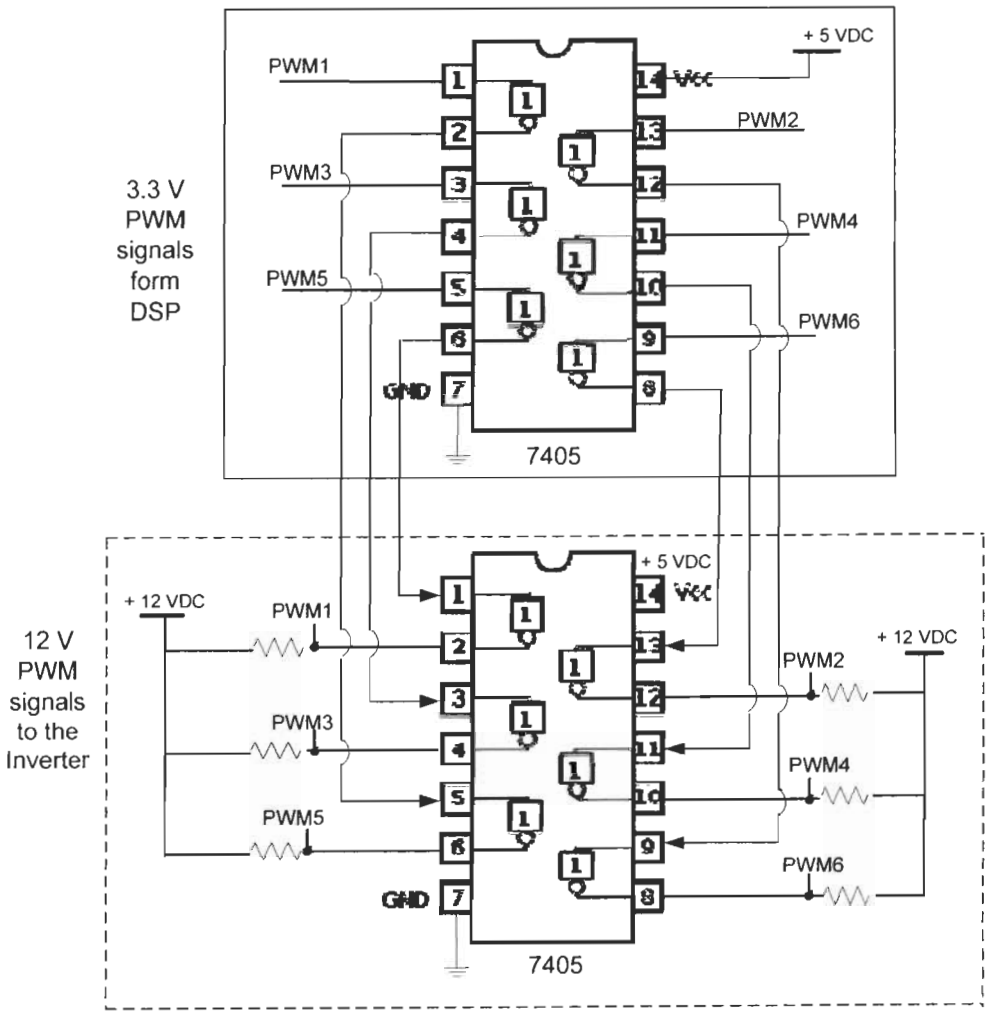


Figure 5.12 PWM interface circuit

5.4.2 Current amplification circuit

The F2812 DSP has two 8-channels analog to digital converter (ADC) modules, each module consists of a 12-bit ADC with a built-in sample-and-hold (S/H) circuit. It has a sampling frequency of maximum 25 MHz.

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

$$\text{Digital Value} = 0, \qquad \text{if input} \leq 0 \text{ V}$$

$$\text{Digital value} = 4096 \times \frac{\text{Input Analog Voltage} - \text{ADCLO}}{3}, \quad \text{if } 0\text{V} < \text{input} < 3\text{V}$$

$$\text{Digital Value} = 4095, \qquad \text{if input} \geq 3 \text{ V}$$

ADCLO equals zero when the pin (VREFLO) on the DSP is connected to the analog ground. The LEM sensors (HX 03-P) used to measure the motor currents give a maximum of ±4V as output. This output needs to be scaled since the DSP analog input signal must be within the range of zero to three volts (0,3) V. The full load motor current is 1.4 A, so this value is used as the maximum input current. A non-inverting operational amplifier is used to scale the current signal. The schematic drawing of the amplification circuit is illustrated in figure 5.13.

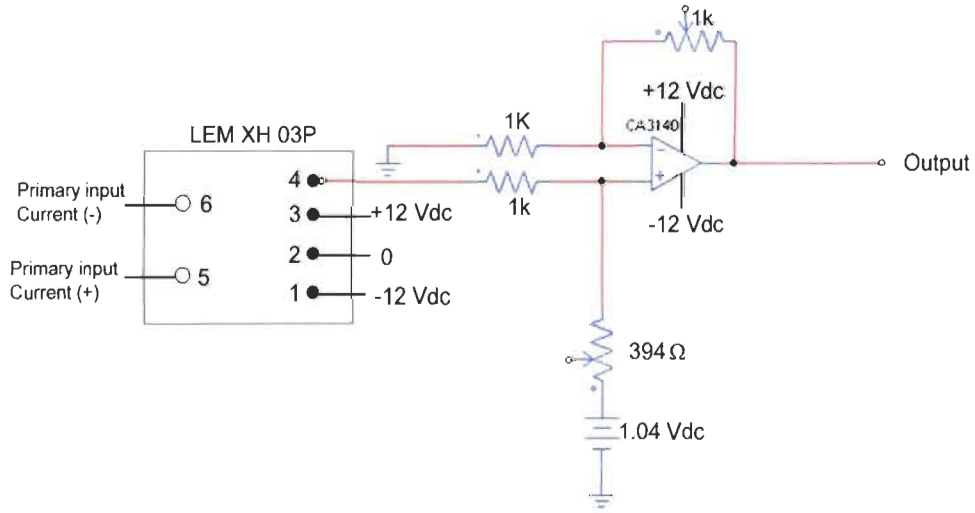


Figure 5.13 Current measurement and amplification circuit

PSIM is used to validate the amplification circuit parameters where the output and input of the amplifier are shown respectively in figures 5.14 - 5.15.

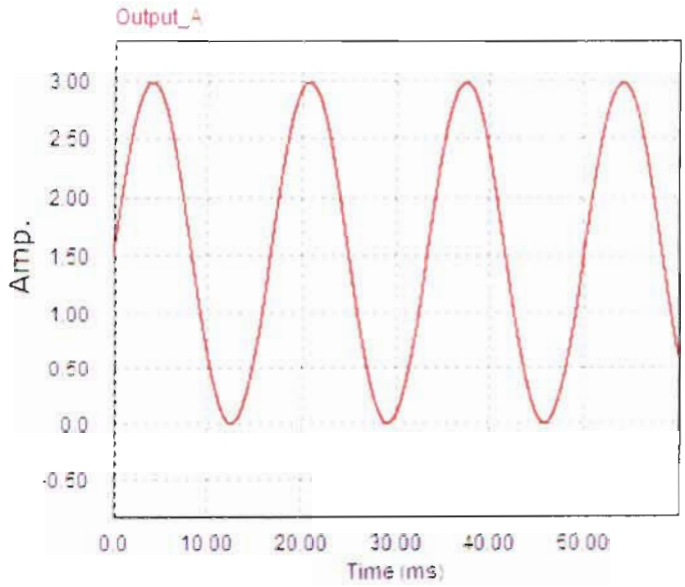


Figure 5.14 Amplifier output

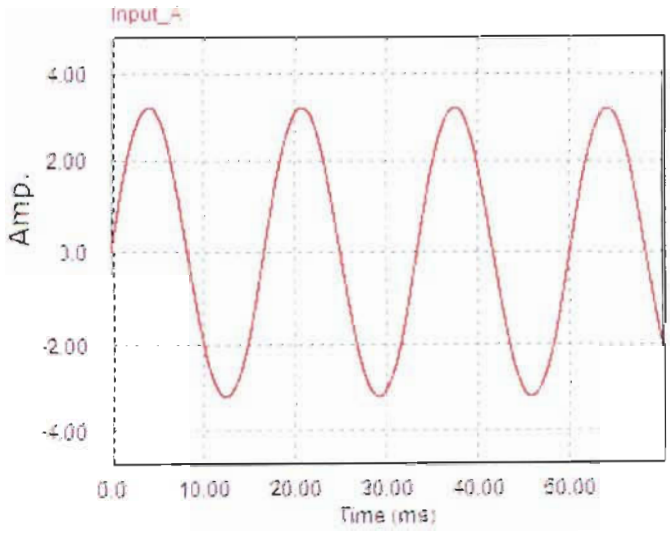


Figure 5.15 Amplifier input

5.4.3 Software Implementation

The general software flowchart incorporates a main routine where CPU, system control registers and modular software blocks initialization take place and interrupt routine where the vector control modules are executed every interrupt. The flowchart is shown in figure 5.16.

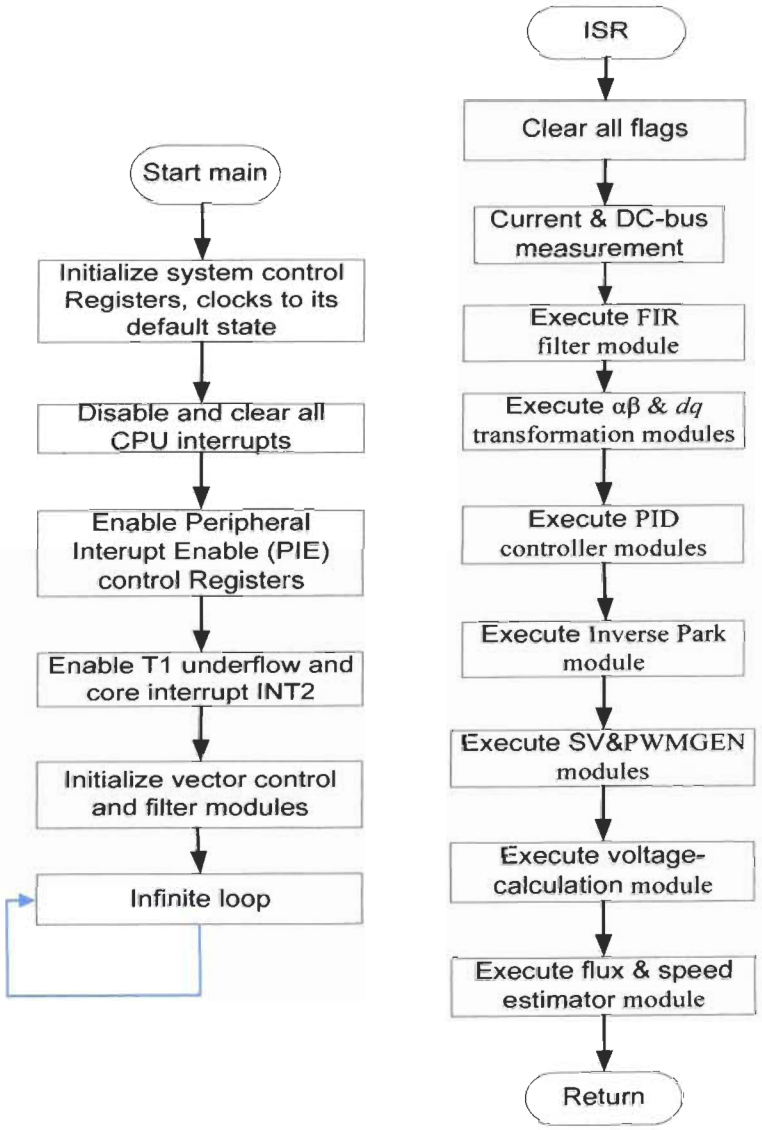


Figure 5.16 Algorithm flowchart

A fractional representation is used for all real quantities so the line current and DC-bus voltage variables are normalized with respect to their individual instantaneous maximum values and expressed as fractional numbers (Q15 format). The implemented control algorithm is an "IQmath" algorithm since the F2812 is a fixed point DSP. A global Q value (Q24) is used throughout the algorithm. The global Q value should be in the range from 19 to 26 in order to have a stable system. Machine parameters and base peak values are listed below.

```
// This machine parameters are based on 1/3-hp Leeson induction motor

#define RS          8.4                 // Stator resistance (ohm)
#define RR          3.819907              // Rotor resistance (ohm)
#define LS          0.297953              // Stator inductance (H)
#define LR          0.297953              // Rotor inductance (H)
#define LM          0.268079              // Magnatizing inductance (H)
#define P           2                     // Number of poles

#define BASE_VOLTAGE    173.2050        // Base peak phase voltage (volt)
#define BASE_CURRENT    1.979898          // Base peak phase current (amp)
#define BASE_TORQUE     0.6596088         // Base torque (N.m)
#define BASE_FLUX       0.4594407       // Base flux linkage (volt.sec/rad)
#define BASE_FREQ       60              // Base electrical frequency (Hz)
```

Peripheral specific C functions are used to initialize the peripherals. They are used by adding the appropriate .c file to the project a long side with linker command and filter files as shown in the project manager view below.

5.4.3.1 Interrupt Service Routine (ISR)

The vector control algorithm shown in Figure 5.17 is executed in the interrupt routine. Each block exists as a stand-alone C/C++ module, which can be interconnected to form the complete vector control algorithm. The ISR starts by calling the IIeg2-bus driver module which allows 3-channel analog-to-digital conversion with programmable gains and offsets. The converted results represent load currents and DC-bus voltage in the inverter. Then, the other vector control modules are executed sequentially.

The modular software blocks are tested by using an incremental development process.

The first incremental step helps verify motor currents and DC-bus voltage measurements in addition to the functionality of the other modules. In this step, reference voltages (*Vds*, *Vqs*) are applied to the inverse Park transform along with a flux angle generated from a ramp generator module.

In the second step, the current loop is closed and the PI controllers are tuned while the system is running in speed open-loop.

Finally, the speed loop is closed and the controller parameters are adjusted to obtain an optimum overall system response.

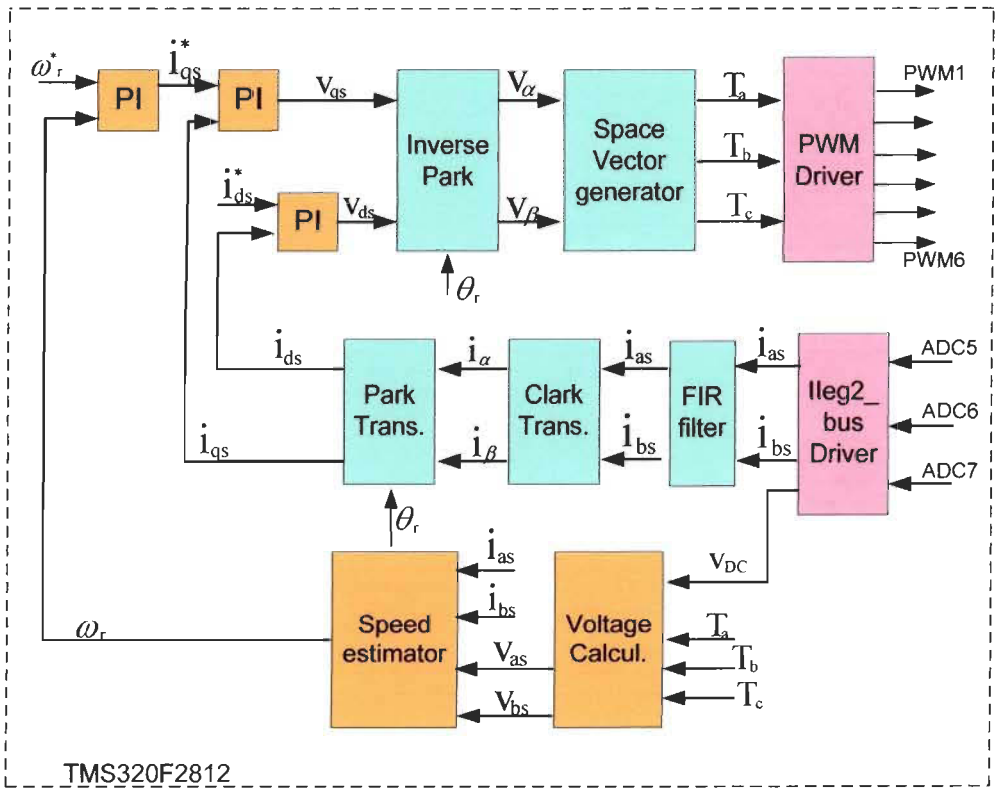


Figure 5.17 Vector control algorithm

The following code lines show how a Clarke transformation module is initialized and called. In the ISR, inputs are passed to the Clarke module then the computation function is called and finally the module outputs are assigned to a Park module.

```
// Instance a clarke transform objects
CLARKE clarke1 = CLARKE_DEFAULTS;

void main(void)
{

 while (1) // Infinite loop.
  {
  }
}
interrupt void MainISR(void)
{

 //.....................................................
 // Filter current outputs are connected to Clarke module
 // and the Clarke computation fuction is called
 //.....................................................

  clarke1.As = _IQ15toIQ((int32)filtA.output);
  clarke1.Bs = _IQ15toIQ((int32)filtB.output);
  clarke1.calc(&clarke1);

 //.....................................................
 // Clarke module outputs are connected to Park module
 // and the Park computation fuction is called
 //.....................................................

  park1.Alpha = clarke1.Alpha;
  park1.Beta = clarke1.Beta;
  park1.Angle = FluxAngle;
  park1.calc(&park1);

}
```

5.4.4 Speed measurement

A quadrature encoder is used to measure the motor speed so as to compare it with the estimated one and to verify whether the motor is capable of following the reference speed or not. LabVIEW software is used to develop a program that reads encoder output (A) pulses then calculates the motor speed and save the result in excel file. The saved speed measurements are plotted using Matlab. Once the number of pulses in a fixed time interval is measured the angular velocity of the quadrature encoder can be calculated using the following formula:

$$\text{Speed} = \frac{\dfrac{Encoder\ Pulses}{Pulses\ per\ revolution} \times \dfrac{60}{1\,min}}{Fixed\,Time\ Interval\,(sec)}\ (RPM) \tag{5.3}$$

The encoder (model E3-500-500-IHT) produces 500 pulses per revolution. Figures 5.18-5.19 below show the block diagram and front panel of the LabVIEW program.

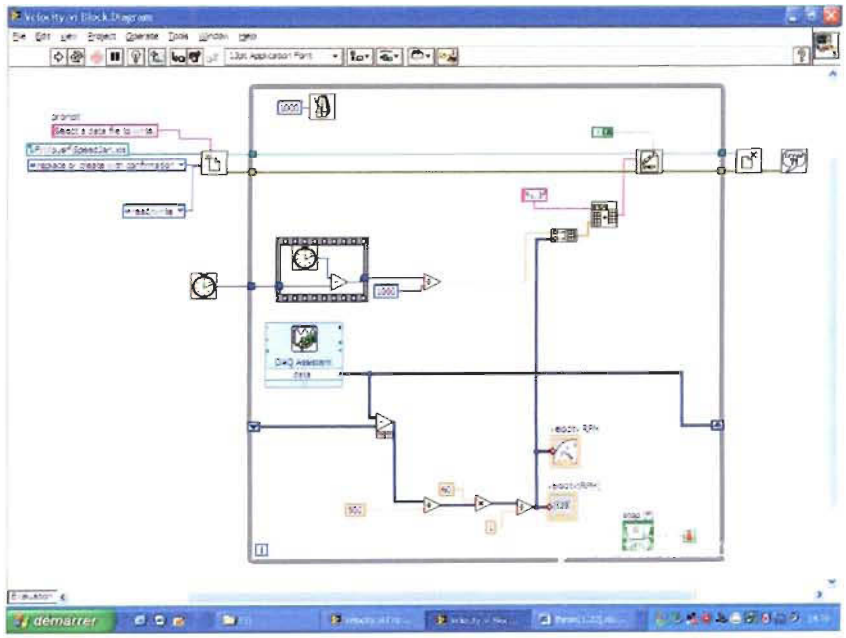


Figure 5.18 Speed measurement Block diagram

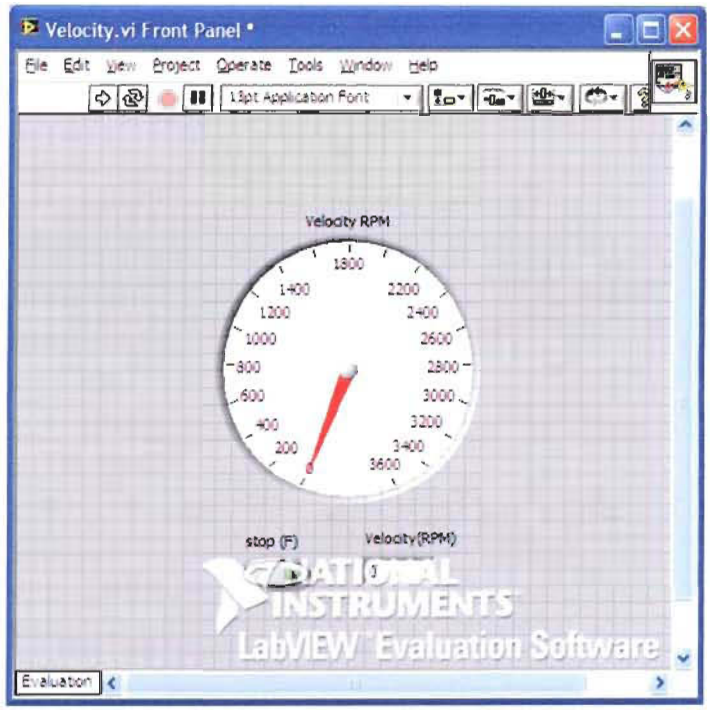


Figure 5.19 Speed measurement Front Panel

5.4.5 Experimental Results

The experimental test results are presented in this section. The electric drive is tested under no load condition for various reference speed values from stand still.

Figure 5.20 shows the measured and filtered phase (A) current. The estimated rotor flux angle is shown in figure 5.21.

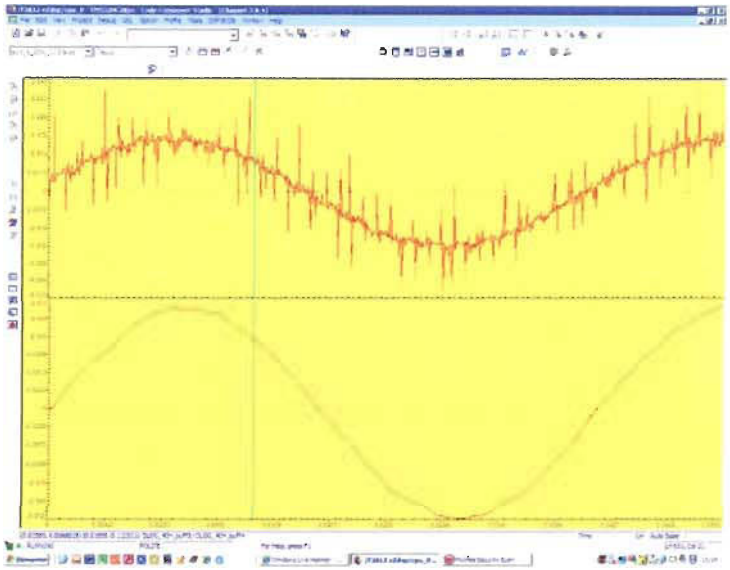


Figure 5.20 Phase (A) current

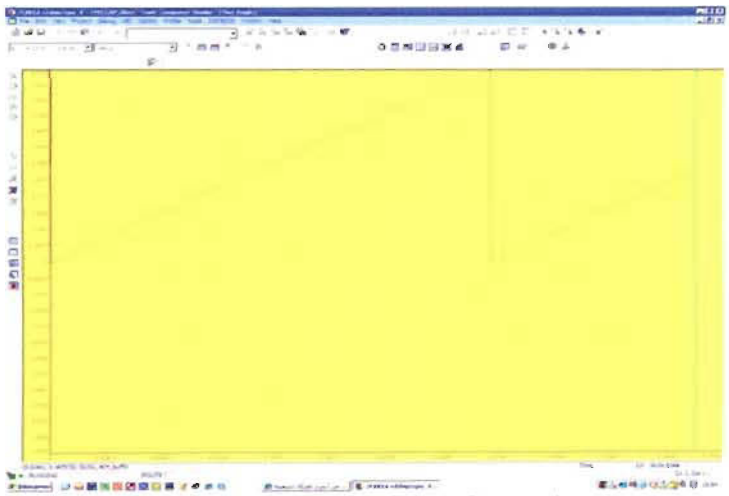


Figure 5.21 Estimated rotor flux angle

Figures 5.22-5.23 illustrate the reference and estimated speed for 60% and 30% of rated speed. The values are expressed in Per Unit.

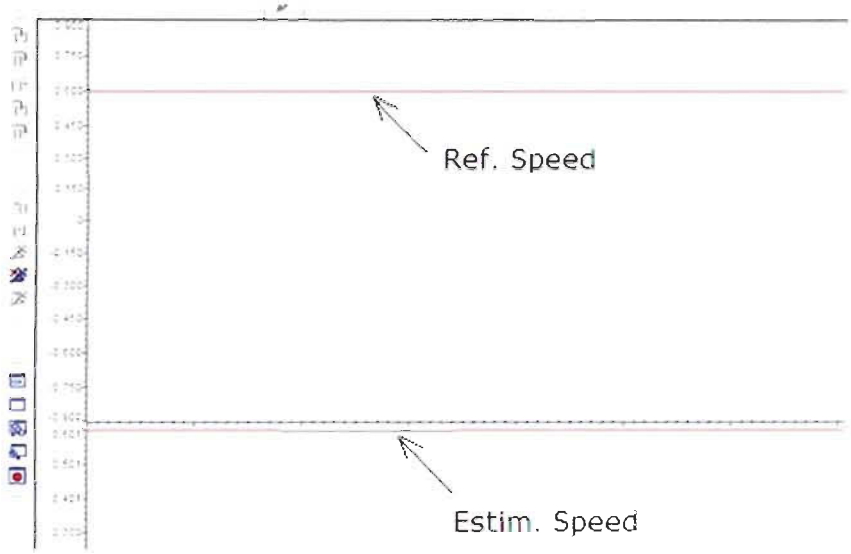


Figure 5.22 Reference and Estimated speed - 0.6 P.U

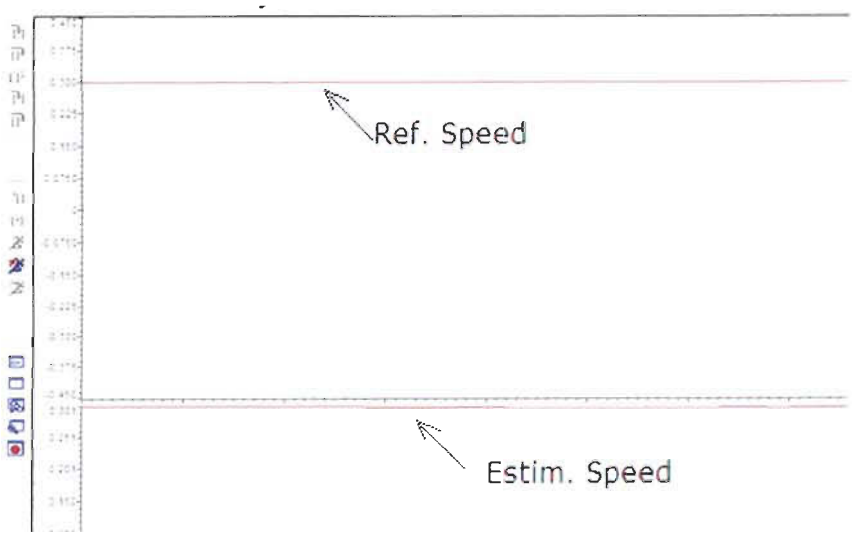


Figure 5.23 Reference and Estimated speed - 0.3 P.U

Figure 5.24 shows the motor speed response to two different reference values, 0.3 and 0.6 of rated speed.

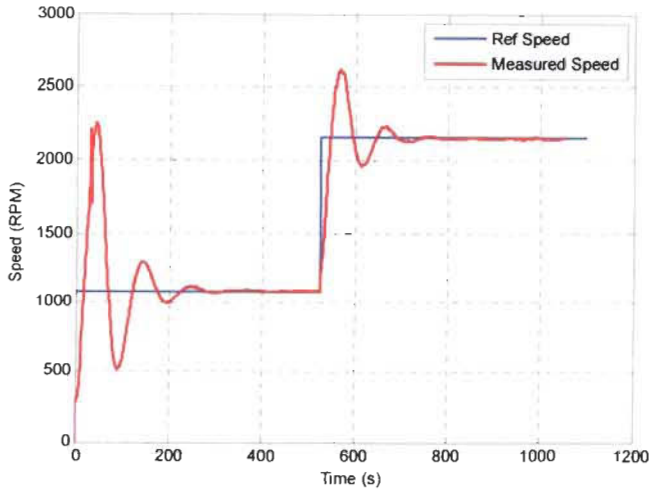


Figure 5.24 Speed response

Figure 5.25 illustrates the motor speed response to different reference values, 0.6, 0.3 and -0.3 of rated speed. Initially the motor is running at 60% of rated speed then the reference speed is changed to 0.3 and -0.3 respectively.

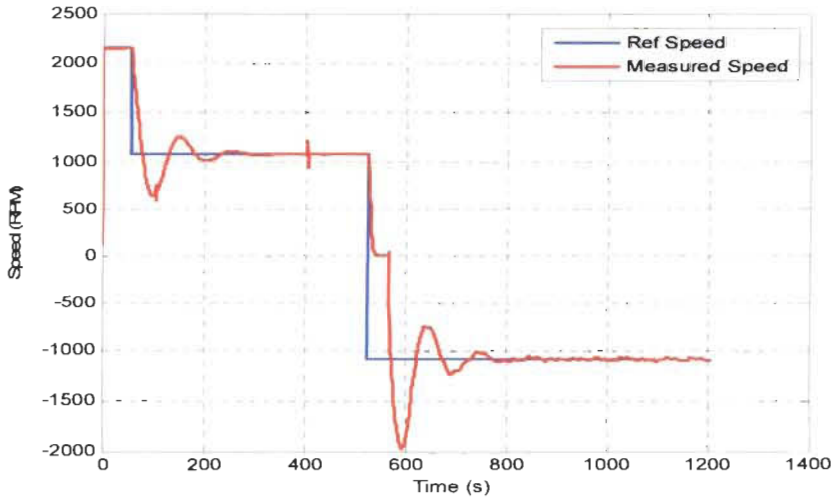


Figure 5.25 Reference and measured speed

Figure 5.26 shows the three phase motor terminal voltage.

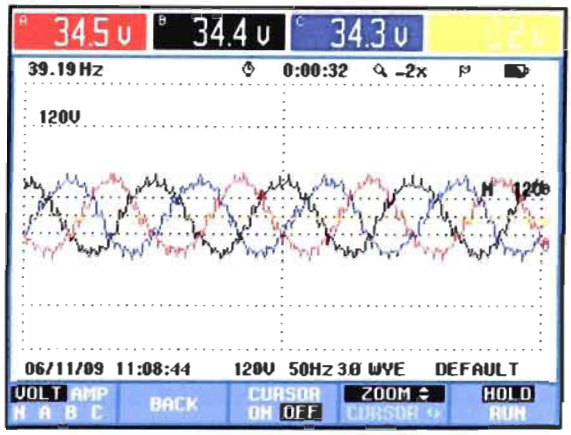


Figure 5.26 Motor 3-phase voltages

5.5 Discussion

The implementation of the sensorless vector control using a digital signal processor (F2812) in addition to a PWM signal generation algorithm are presented in this chapter. The different hardware parts of the experimental setup are outlined. The setup represents a traditional variable speed induction motor drive. However, the current amplification circuit is designed to be used specifically with the selected motor (1/3 hp).

The sensorless vector control algorithm is coded in C language using software modules developed by TI. These modules are provided in form of free libraries available for user to evaluate TI products. The algorithm flowchart that describes the implemented algorithm is illustrated. Finally, the experimental results are shown, thanks to the graph feature of code composer studio that allows the real time display of different variables.

The motor current signals show that the digital filter works properly. The estimated flux angle and motor speed demonstrates that the sensorless vector control algorithm is functioning as desired. These results demonstrate also the validity of the overall hardware structure of the illustrated electric drive as well as the control algorithm.

Chapter 6 - Conclusion

The goal of this thesis is to design a variable speed induction motor drive based on a sensorless vector control technique and then to realize the electric drive using a digital signal processor. The work aims also to include the design and implementation of a digital filter within the sensorless algorithm to filter measured motor currents.

The research work started by a thorough literature review of alternate current (AC) electric drives, drive components, types, the various vector control and speed estimation techniques. Then, modeling of the drive main components is carried out, basically the induction motor and the inverter. Both, steady state and dynamic models of the induction motor are discussed. The well-known Matlab-Simulink software is used to simulate the induction motor based on the motor dynamic model. A Space Vector Pulse Width Modulation (SVPWM) based voltage source inverter is also simulated using Simulink. The results obtained from the simulation of these two major components of the electric drive demonstrate the validity of the developed Simulink models. Therefore, these two components could be used in the simulation of an electric drive. Next, the work is headed towards studying the different induction machine control techniques, specifically the vector control technique.

Once the various control techniques are explored, a simulation of a variable speed drive is achieved using the direct vector control technique where the measured motor current, voltage and speed are used to estimate the flux angle. The main problem that I faced at this point was the tuning of the Proportional Integral (PI) controller parameters since I used the trial and error method, which is tedious and time-consuming for the first time. As soon as the controllers are tuned the electric drives performed well and the results were found satisfactory. Then, speed and flux estimation techniques are discussed in details where the famous estimation techniques are simulated. The advantages and disadvantages of each technique are outlined. Next, the simulation of the sensorless vector control based electric drive is carried out by replacing the speed sensor of the direct vector control simulation model mentioned earlier with a speed and flux estimator.

After the successful finishing of the electric drive simulation, the work is devoted for a while to studying digital filters and their implementation using the F2812 DSP. Design

issues are addressed where a step-by-step design procedure is introduced. Matlab Filter Design and Analysis Tool (FDATool) is used to obtain the digital filter coefficient in a form of a header file. The header file is used alongside with a TI filter module to implement the digital filter. The results show that motor currents are filtered correctly without the need to add more hardware to the electric drive.

The work achieved up to now lays the groundwork for the experimental validation. The test bench represents a typical electric drive where an industrial power inverter is used. Ancillary circuits are designed for motor current amplification and interfacing the DSP signals to suit the selected inverter. A flexible sinusoidal PWM signal generation algorithm is developed to test the experimental setup. The algorithm is written in C++ language where the switching frequency and the output signal frequency could be varied simply by writing new values to their corresponding registers while the program is running. Formulas to calculate these frequencies are introduced. The algorithm is implemented and satisfactory results are obtained. The sensorless vector control algorithm is implemented with help of TI C coded modules. There is a software module for each building block of the control structure. Coding the control algorithm using C language is time-consuming and requires profound programming skills, so I do recommend rapid control prototyping platforms for similar project. The experimental results obtained are good enough to say that this research project accomplished successfully its goals.

References

[1] A.K. Chattopadhyay, "Advances in vector control of ac motor drives – A review," Sadhana, vol.22, 1997.

[2] P. Vas, Vector control of AC machines. Oxford: Clarendon Press, 1990.

[3] P. Vas, Sensorless vector and direct torque control. New York: Oxford University Press, 1998.

[4] Jean Bonal and G. Séguier, Entraînement électrique à vitesse variable. Vol. 2. Lavoisier TEC & DOC 1998.

[5] J. S. Thongam, "Commande de haute performance sans capteur d'une machine asynchrone," UQAC, 2006.

[6] J. Ghouili, "Commande sans capteur d'une machine asynchrone avec estimation de la vitesse par réseaux de neurones,". UQTR, 2005.

[7] F. Blaabjerg, R. Teodorescu, M. Zelechowski, P. Lach, and P. Rozanski, "Rotor flux and speed estimators for induction motor drives. A performance evaluation," IEEE ISIE, 2005, pp. 19-24 vol.1.

[8] J. Holtz and J. Quan, "Drift and parameter compensated flux estimator for persistent zero stator frequency operation of sensorless controlled induction motors," in Industry Applications Conference, 2002. 37th IAS Annual Meeting. Conference Record of the, 2002, pp. 1687-1694 vol.3.

[9] H. Jun and W. Bin, "New integration algorithms for estimating motor flux over a wide speed range," Power Electronics, IEEE Transactions on, vol. 13, pp. 969-977, 1998.

[10] M. Ishida and K. Iwata, "A New Slip Frequncy Detector of an Induction Motor Utilizing Rotor Slot Harmonics," Industry Applications, IEEE Transactions on, vol. IA-20, pp. 575-582, 1984.

[11] F. Profumo, G. Griva, M. Pastorelli, J. Moreira, and R. De Doncker, "Universal field oriented controller based on air gap flux sensing via third harmonic stator voltage," Industry Applications, IEEE Transactions on, vol. 30, pp. 448-455, 1994.

[12] T. Instruments, "Digital Signal Processing Solution for AC Induction Motor," 1996.

[13] N. Mohan, Advanced Electric Drives - Analysis, Control and modeling using Simulink. Minneapolis: MNPERE, 2001.

[14] P. C. Krause, O. Wasynczuk, and S. D. Sudhoff, Analysis of electric machinery and drive systems. Piscataway, NJ New York: IEEE Press ; Wiley-Interscience, 2002.

[15] M. L. Barnes and C. A. Gross, "Comparison of Induction Machine Equivalent Circuit Models," IEEE, vol. System Theory, 1995., Proceedings of the Twenty-Seventh Southeastern Symposium pp. 14 - 17, 12-14 March 1995.

[16] W. Leonhard, Control of electrical drives. Berlin: Springer, 2001.

[17] M. H. Rashid, Power electronics handbook : devices, circuits, and applications. Amsterdam: Academic Press, 2006.

[18] M. A. El-Sharkawi, Fundamentals of electric drives. Pacific Grove, Calif.: Brooks/Cole, 2000.

[19] I. Boldea and S.A. Nasar, Electric drives. 2nd ed. 2006, Boca Raton, Fla.: Taylor & Francis. xvii, 522 p.
[20] T. Instruments Europe, " Field Oriented Control of 3-phase AC-motors," February 1998.
[21] D.G. Holmes and T.A. Lipo, Pulse Width Modulation for Power Converters:Principles and Practice. IEEE Press Series on Power Engineering, Wiley-IEEE. Press. 2003
[22] A. Iqbal, A. Lamine, I. Ashra, and Mohibullah, "Matlab/Simulink Model of Space Vector PWM for Three-Phase Voltage Source Inverter," in Universities Power Engineering Conference, 2006. UPEC '06. Proceedings of the 41st International, 2006, pp. 1096-1100.
[23] K. Rajashekara, A. Kawamura, and K. Matsuse, Sensorless control of AC motor drives : speed and position sensorless operation. Piscataway, N.J.: IEEE Press, 1996.
[24] W.-K. Chen, The Circuits and filters handbook. The electrical engineering handbook series. Boca Raton, New York: CRC Press, IEEE Press, 1995.
[25] L. Thede, Practical analog and digital filter design. Artech House microwave library. 2005, Boston: Artech House.
[26] www.dsptutor.freeuk.com/dfilt1.htm.
[27] L. Rabiner, "Techniques for Designing Finite-Duration Impulse-Response Digital Filters," Communication Technology, IEEE Transactions on, vol. 19, pp. 188-195, 1971.
[28] T. Instruments, "Filter library," 2002.
[29] T. Parks and J. McClellan, "A program for the design of linear phase finite impulse response digital filters," Audio and Electroacoustics, IEEE Transactions on, vol. 20, pp. 195-199, 1972.
[30] T. W. Parks and C. S. Burrus, Digital filter design. New York: J. Wiley, 1987.
[31] Spectrum Digital, "eZdspTM F2812: Technical Reference," 2002.
[32] Texas, I. (2006) Code Composer Studio Development Tools v3.3: Getting Started Guide (SPRU509H).
[33] Skvarenina, T., The Power Electronics Handbook. 2002: CRC Press.
[34] M. A. Boost and P. D. Ziogas, "State-of-the-art carrier PWM techniques: a critical evaluation," IEEE Transactions on Industry Applications, vol. 24, pp. 271-280, 1988.
[35] B. K. Bose, Power electronics and ac drives. Englewood Cliffs, N.J.: Prentice-Hall, 1986.
[36] Electronic Design, "Generate Advanced PWM Signals Using DSP," May 01, 1998.
[37] Texas, Instruments, TMS320x281x DSP Event Manager (EV) Reference Guide, Literature Number: SPRU065D, 2006.
[38] Texas Instruments, "Sensor-less Direct Flux Vector Control of 3-phas ACI". 2005.

Appendices

A- Résumé du travail de recherche

Introduction

Un entraînement électrique est un système électromécanique qui effectue la conversion de l'énergie électrique en énergie mécanique pour le fonctionnement de processus technologiques. Aujourd'hui, dans les pays industrialisés, plus de 60% de l'énergie électrique destinée à l'industrie est consommée par des entraînements électriques. Par conséquent, les entraînements électriques qui nécessitent un contrôle précis de couple, vitesse et position sont largement répandus. Ces entraînements sont caractérisés par la réponse rapide du couple et la possibilité de commande de couple et de vitesse sur une large plage [1]. Généralement, il y a deux types d'entraînements électriques industriels : ceux qui tournent à vitesse fixe et ceux fonctionnant à vitesse variable. Traditionnellement, les machines à courant alternatif (CA) ont été utilisées dans des applications à vitesse fixe, tandis que les machines à courant continu (CC) à excitation séparée ont été préférées pour les applications à vitesse variable, parce que ce type de machine a une réponse très rapide de couple et facile à commander. Toutefois, les machines à courant continu, en plus de leur coût élevé, ont aussi des problèmes de maintenance liés à des collecteurs et des balais qui limitent aussi le fonctionnement de la machine (CC) dans un environnement poussiéreux et explosif [2].

La machine à induction à cage d'écureuil, qui se caractérise par sa simplicité de construction, robustesse et faible coût, est couramment utilisée dans l'industrie moderne. Ces avantages rendent ce type de machine un choix populaire pour réaliser des entraînements électriques performants où, traditionnellement, seules les machines (CC) étaient utilisées. Toutefois, la commande d'une machine à induction est très complexe. Cette complexité est due au fait que le courant rotorique, responsable de la production de couple, dépend du courant statorique qui contribue également au flux d'entrefer, créant ainsi un couplage entre les mécanismes de production de flux et de couple [1]. Récemment, les entraînements à vitesse variable incorporant des machines asynchrones ont pris le dessus en raison du progrès dans le domaine de l'électronique de puissance et de la technologie de traitement numérique du signal où plusieurs techniques de

commande ont émergé. Dans le passé, des telles techniques de commande n'auraient pas été possibles en raison de la complexité du matériel et des logiciels nécessaires pour résoudre le problème complexe de la commande [3].

En général, la vitesse de la machine est mesurée pour l'utiliser comme un signal de rétroaction dans la boucle de contrôle de vitesse de la commande vectorielle. Les capteurs de position et de vitesse diminuent la fiabilité du système particulièrement dans les environnements hostiles. Dans la technique de commande sans capteur, le capteur de vitesse est remplacé par un estimateur de vitesse qui élimine les coûts associés au capteur de vitesse et réduit la complexité de l'entraînement [5]. Les inconvénients majeurs de la commande sans capteur sont les contraintes associées à l'estimation des quantités non mesurables (flux statorique et rotorique) et / ou les quantités que nous ne voulons pas mesurer pour des raisons économiques (vitesse, position) [6]. Ces quantités peuvent être estimées à partir des tensions et courants statoriques faciles à mesurer. Il existe de nombreuses techniques présentées dans la littérature pour extraire la vitesse à partir de tensions et des courants mesurés. Toutefois, la performance de ces techniques à faible vitesse reste incertaine. De nombreuses recherches récentes se concentrent sur le développement des techniques efficaces d'estimation de la vitesse afin d'améliorer le rendement à faible vitesse des entraînements électriques sans capteur.

Les techniques d'estimation de la vitesse sont généralement classées en trois groupes principaux:

• Le premier inclut les estimateurs basés sur le modèle dynamique de la machine asynchrone. Dans ce cas, le flux est déduit à partir des équations dynamiques de la machine [6]. Cette approche est simple, mais elle est très sensible aux paramètres de la machine et il y a aussi des problèmes liés à l'utilisation des intégrateurs afin d'estimer le flux [7]. À cause des erreurs induites par l'utilisation des intégrateurs purs, des estimateurs de flux dont la conception est basée sur des filtres passe-bas ont été proposés [8, 9]. Cependant, les filtres génèrent des retards de phase incompatibles avec la dynamique demandée et mènent à l'instabilité à grande vitesse.

• Le second utilise les techniques de traitement du signal pour extraire les informations requises. Malgré sa simplicité, cette technique nécessite des circuits de filtrage complexes et ses performances dynamiques sont faibles [10, 11].

• Le troisième est basé sur des techniques d'intelligence artificielle (réseaux de neurones, logique floue et algorithmes génétiques). Les estimateurs de réseaux de neurones semblent avoir une meilleure réponse, mais leur réalisation pratique reste un défi [6].

Traditionnellement, les circuits de commande de moteur ont été conçus avec des composants analogiques qui sont faciles à concevoir et peuvent être réalisés avec des composants relativement peu coûteux. Toutefois, les composants analogiques sont affectés par la température et le vieillissement, donc le système de contrôle doit être ajusté périodiquement. D'ailleurs, la mise à niveau d'un tel système est tout à fait complexe, car la conception est dépendante du matériel. Les systèmes numériques de commande offrent des solutions pratiques aux inconvénients de la commande analogique. La mise à niveau est facilement réalisée par un logiciel et la plupart des composants analogiques, pourrait être remplacée par des fonctions. Malgré les solutions offertes par la commande numérique, il reste encore plusieurs défis à relever comme la réduction de la consommation d'énergie et la réduction d'interférence électromagnétique (EMI) imposée par les gouvernements. Le développent d'algorithmes améliorés peut donner des solutions à tous ces facteurs contraignants. La technologie des processeurs numériques de signal (DSP) permet d'obtenir à la fois, un haut niveau de performance ainsi qu'une réduction des coûts de système [12].

L'objectif de ce travail de recherche est de concevoir et de réaliser la commande vectorielle d'une machine asynchrone (sans capteur) en utilisant un processeur numérique de signal (DSP) et d'en évaluer la performance. Le travail vise aussi à concevoir et implémenter un filtre numérique dans l'algorithme de commande sans capteur pour filtrer les courants mesurés de la machine.

Le travail commence par une recherche bibliographique et l'étude des concepts fondamentaux des différentes techniques de commande vectorielle appliquées à la machine asynchrone à cage. Ensuite, nous effectuons la modélisation et la simulation de la commande vectorielle dans l'environnement Matlab/Simulink/SimPowerSystems.

Par la suite, nous développons un algorithme de commande permettant d'introduire un filtre numérique. Enfin, la validation expérimentale est effectuée.

Modélisation de la machine asynchrone

Dans le premier chapitre, la modélisation de la machine à induction est présentée, car l'obtention du modèle mathématique qui caractérise exactement la machine asynchrone est nécessaire pour la commander. Les équations de modèle en régime permanent et de modèle dynamique sont présentées. Enfin, le logiciel Matlab - Simulink est utilisé pour simuler le modèle dynamique développé et les résultats de simulation sont présentés. La figure 1 montre le modèle Simulink de la machine asynchrone sur la référence statorique.

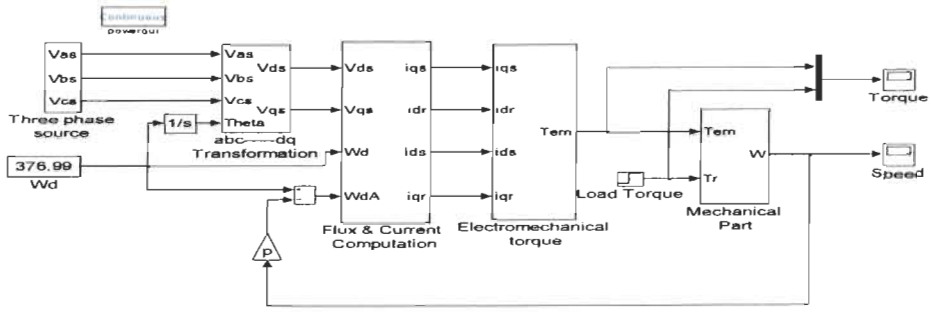


Figure 1 Modèle Simulink de la machine asynchrone.

Les réponses de la vitesse et de couple sont illustrées sur les figures 2 et 3. À l'instant t = 0.5s un couple de charge est appliqué à la machine et sa réponse est jugée satisfaisante.

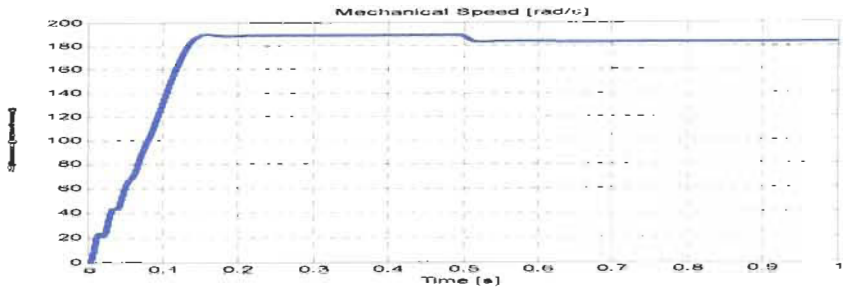


Figure 2 Vitesse de la machine.

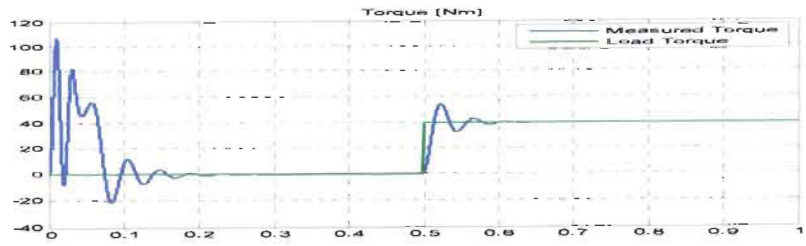


Figure 3 Couple de la machine.

Commande de la machine asynchrone

Le deuxième chapitre décrit les différentes méthodes de commande appliquées à des entrainements électriques à vitesse variable. Tout d'abord, les techniques de commande scalaire basée sur le modèle en régime permanent de la machine asynchrone sont examinées. Puis, les méthodes de commande instantanée de couple sont abordées, principalement la commande directe de couple et la commande vectorielle. La méthode de commande par modulation de largeur d'impulsions par vecteurs spatiaux « Space Vector Pulse Width Modulation » (SVPWM) est également décrite. Enfin, la commande sans capteur, et les différentes techniques d'estimation de vitesse sont discutées en détail suivi par des simulations.

Aujourd'hui, il existe plusieurs types de commande pour la machine asynchrone. On peut choisir la commande appropriée selon la performance demandée.

Lorsque les performances dynamiques demandées ne sont pas contraignantes, ce qui est le cas des entraînements de pompes, ventilateurs et compresseurs, on peut utiliser des commandes relativement simples que l'on qualifie souvent de commandes de type scalaire. Mais lorsqu'on est plus exigeant sur les performances dynamiques, il est nécessaire de contrôler le couple à faible vitesse et pendant les régimes transitoires. On utilise alors des commandes qui permettent de contrôler les courants statorique et rotorique et donc le couple. Fondamentalement, il existe deux types de commande instantanée de couple pour les applications de haute performance : la commande directe de couple et la commande de flux orienté (commande vectorielle) [4]. En utilisant la théorie de vecteurs spatiaux, il est facile de démontrer que le couple électromagnétique instantané d'une machine asynchrone peut être exprimé comme le produit d'un courant lié au flux et un courant lié au couple, si le flux est orienté sur un repère donné [3].

Il existe essentiellement deux types de techniques de commande vectorielle. Soit la commande directe ou indirecte.

Dans la méthode directe, le flux d'entrefer est directement mesuré à l'aide de capteur, ou estimé à partir de la vitesse, tension et courant statorique.

L'inconvénient majeur de la méthode directe est le problème lié à l'intégration dans l'établissement du flux à basse vitesse lorsque la chute ohmique (IR) de la machine devient dominante.

Dans la méthode directe, l'angle du flux du rotor est indirectement calculé en additionnant la position du rotor mesurée avec la position de glissement. Contrairement à la méthode directe, la méthode indirecte est fortement dépendante des paramètres de la machine.

La commande vectorielle directe est simulée avec Simulink. La figure 4 montre la structure de contrôle simulée où le sous-système intitulé machine à induction contient le modèle de la machine asynchrone développé dans le premier chapitre.

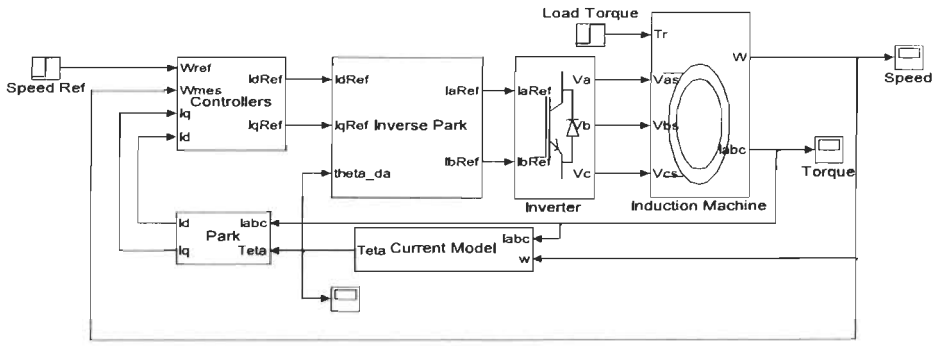


Figure 4 Modèle Simulink de la commande vectorielle directe

Le sous-système onduleur contient un modèle d'onduleur SVPWM développé dans [22]. Le temps de commutation et l'état de chaque interrupteur sont calculés à partir de code du programme MATLAB. La figure 5 montre le modèle Simulink complet de l'onduleur. Le modèle est testé en lui fournissant la tension de référence *dq* et les résultats sont présentés dans la figure 6.

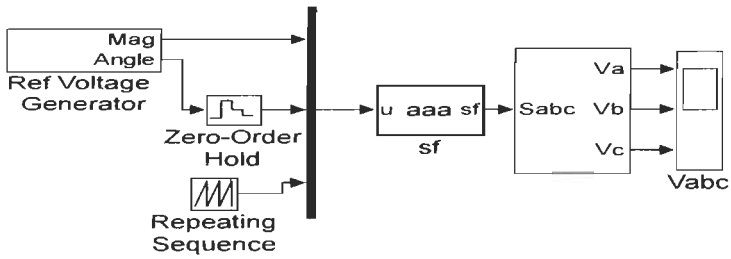


Figure 5 Modèle d'onduleur SVPWM

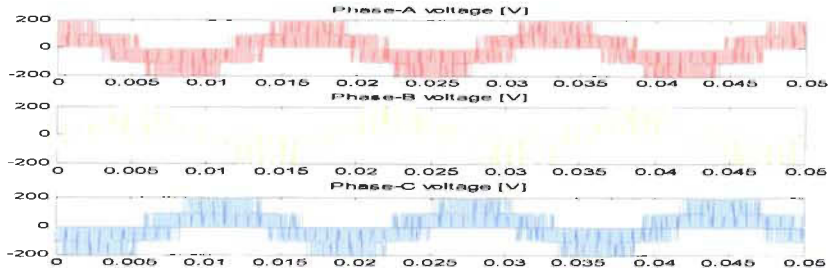


Figure 6 Tension aux bornes de l'onduleur

Le calcul d'angle du rotor est fait dans le sous-système modèle de courant, l'angle est dérivé des équations de tension de rotor exprimée dans la référence rotorique. Dans ce cas, le flux du rotor est entièrement aligné sur l'axe direct, donc la composante en quadrature du flux rotorique est égale à zéro. L'angle du rotor est calculé à partir des équations suivantes comme illustré dans figure 7.

$$0 = \frac{R_r}{L_r}\lambda_{rd} + \frac{d}{dt}\lambda_{rd} - \frac{R_r \times L_m}{L_r} i_{sd}$$

$$\lambda_{rd} = \frac{L_m}{1 + T_r.s} i_{sd}$$

$$0 = R_r i_{rq} + (\omega_{sys} - \omega_m)\lambda_{rd}$$

$$i_{rq} = -\frac{L_m}{L_r} i_{sq}$$

$$\omega_{sl} = \frac{R_r \times L_m}{L_r \times \lambda_{rd}} i_{sq}$$

$$\theta = \int \omega_m + \omega_{sl}$$

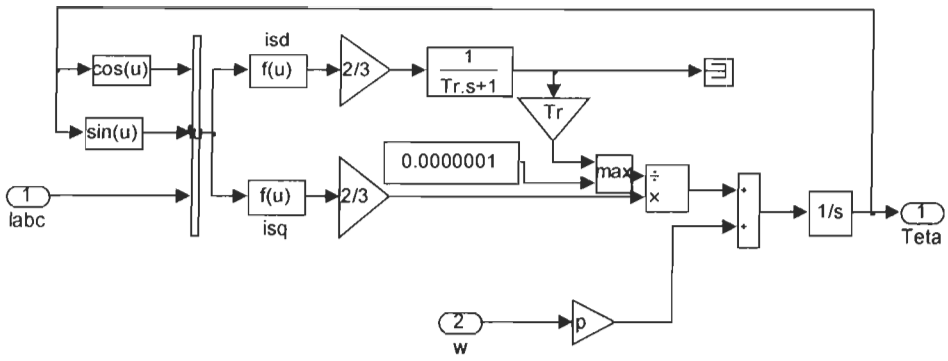


Figure 7 Calcul de l'angle du rotor

Le sous-système de contrôleur contient deux boucles parallèles de contrôle. La première boucle est utilisée pour ajuster le flux de la machine. La seconde se compose de deux boucles en cascade : une boucle intérieure de courant et une autre extérieure de vitesse. Des contrôleurs Proportionnel Intégral (PI) sont utilisés, les paramètres de ces contrôleurs sont ajustés par essai et erreur. La figure 8 montre le contenu du bloc contrôleur.

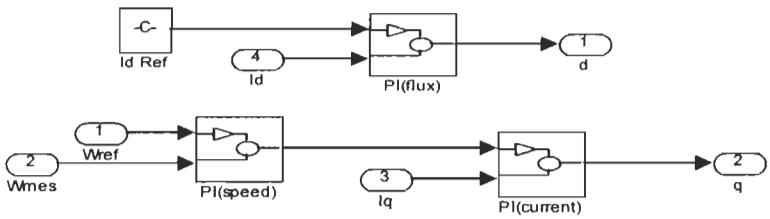


Figure 8 Boucle de régulation

Différentes conditions de fonctionnement ont été étudiées afin de valider le modèle de la commande vectorielle directe et de démontrer l'exactitude de la modélisation et simulation. Figure 9 montre la consigne de vitesse et la vitesse mesurée, à t = 4s, la vitesse du moteur est augmentée de 70 à 100 rad/s. La réponse de couple électromagnétique est illustrée sur la figure 10. Le couple de charge est augmenté à l'instant t = 8s. On voit que le moteur suit précisément la valeur de la consigne de couple.

Nous remarquons que dans les deux figures, la commande vectorielle répond de manière adéquate et ramène les variables commandées à leur valeur désirée.

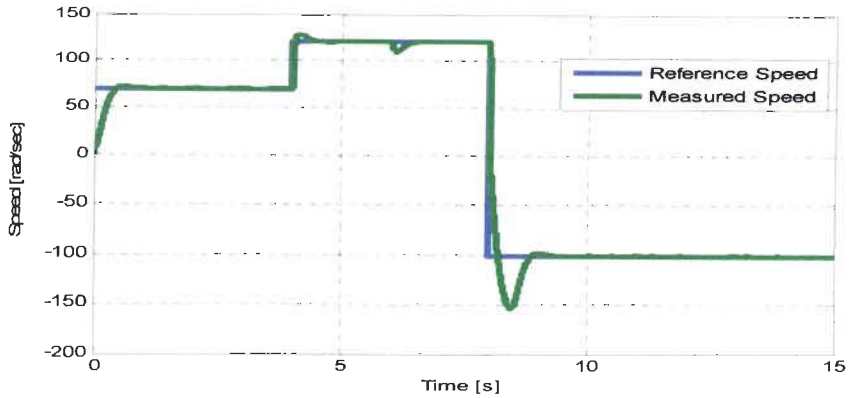


Figure 9 Vitesse du moteur

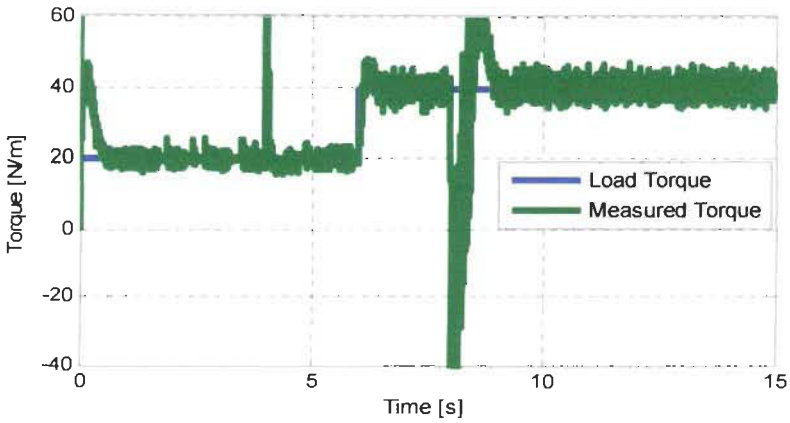


Figure 10 Couple électromagnétique

Commande sans capteur

Dans cette section, les techniques d'estimation de vitesse les plus populaires ont été examinées. Commençant par l'estimateur en boucle ouverte, où la vitesse est déduite à partir des équations du modèle dynamique de la machine. L'utilisation de ce type d'estimateurs est limitée à des applications à faible performance parce que leur précision dépend fortement des paramètres de la machine. Puis, l'estimateur en boucle fermée par

un système adaptatif avec modèle de référence (SAMR) est décrit où certaines variables d'état (flux rotorique, force contre-électromotrice, etc.) de la machine asynchrone sont estimées dans un modèle de référence et sont ensuite comparées avec les variables d'état estimées en utilisant un modèle ajustable. Finalement, un observateur adaptatif de vitesse basé sur le modèle dynamique est présenté. Cet observateur permet de reconstituer l'état d'un système observable à partir de la mesure des entrées et sorties. Ces différents types d'estimateurs sont simulés en boucle ouverte à l'aide de Simulink.

Les figure 11 et 12 montrent les résultats de simulation de ces deux estimateurs.

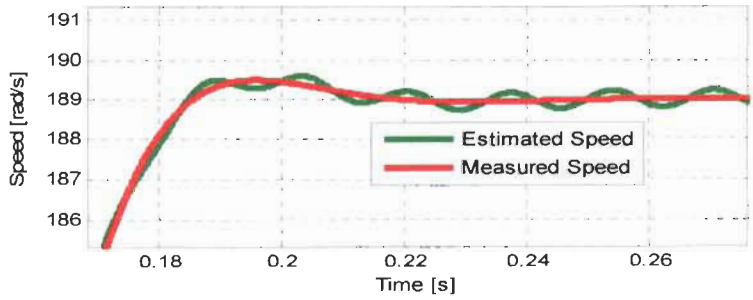


Figure 11 Estimateur en boucle ouverte

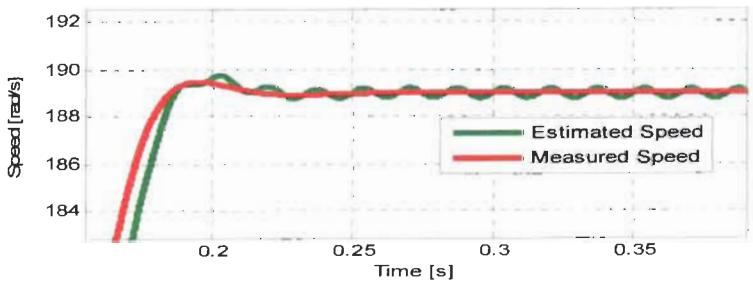


Figure 12 Estimateur SAMR

La commande vectorielle sans capteur est simulée dans Simulink. La structure de l'entraînement électrique sans capteur est montrée dans la figure 13.

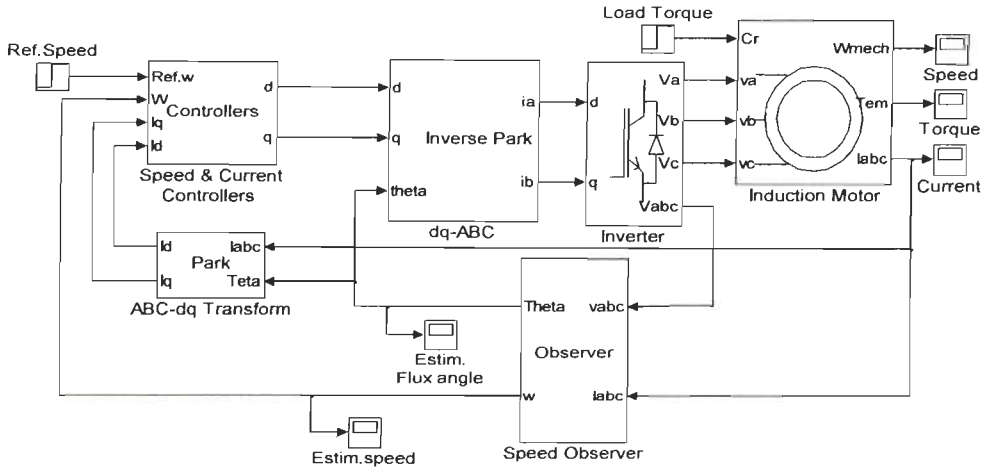


Figure 13 Modèle Simulink de la commande vectorielle sans capteur

Le sous-système d'observateur contient le modèle de l'observateur adaptatif illustré sur la figure 14.

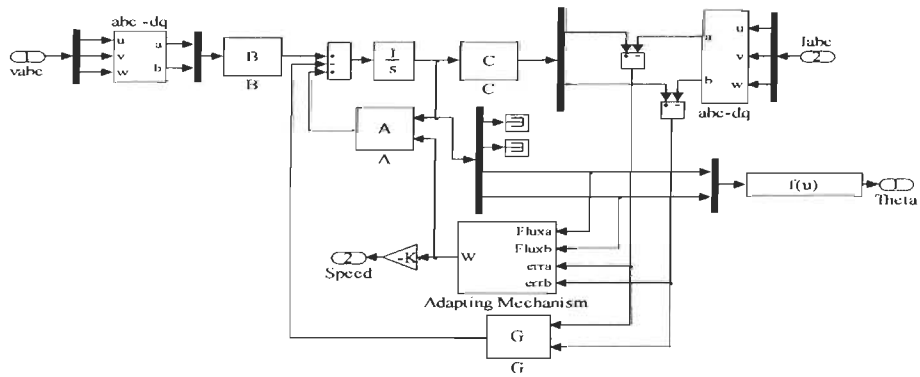


Figure 14 Modèle Simulink de l'observateur adaptatif

Résultats de simulation

Figures 15 à 18 montrent la vitesse, le couple, le courant et la position de rotor. Les paramètres du moteur sont supposés constants, donc l'effet de variation des paramètres n'est pas étudié.

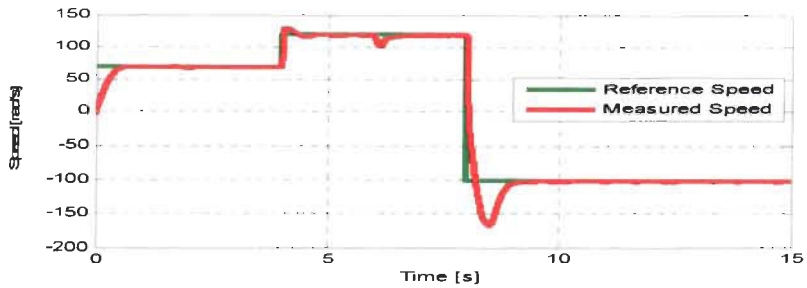


Figure 15 La vitesse du moteur

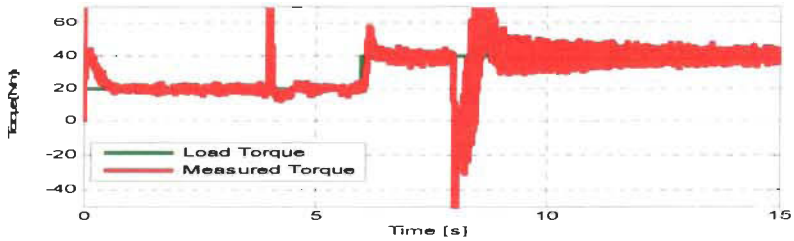


Figure 16 Couple électromagnétique

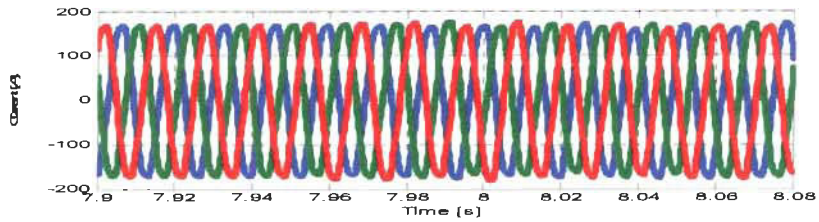


Figure 17 Courant statorique du moteur

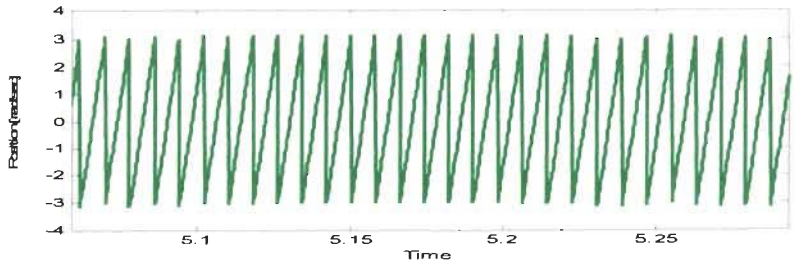


Figure 18 Angle du rotor

Filtrage numérique

Ce chapitre présente une introduction aux filtres numériques avec un aperçu de la conception de filtres et le problème de leur implémentation. Un outil pratique de conception de filtre fourni par Matlab est présenté ainsi que les étapes de design et les réponses qui permettent d'analyser le fonctionnement d'un filtre numérique. Un module logiciel développé par Texas Instrument qui permet de réaliser un filtre à réponse impulsionnelle finie est introduit. Ce module est utilisé dans l'algorithme de commande vectorielle sans capteur pour implémenter un filtre passe-bas afin d'extraire le composant fondamental (60Hz) de courant de la machine.

Filtre à réponse impulsionnelle finie (FIR)

Un filtre à réponse impulsionnelle finie est un filtre dont la transformée en z de sa réponse impulsionnelle est de la forme

$$H(z) = b_0 + b_1 z^{-1} + ... + b_M z^{-M} \qquad 4.1$$

Un filtre à réponse impulsionnelle finie peut être réalisé de façon non récursive, cela signifie que la sortie dépend uniquement de l'entrée du signal. Ce genre de filtre est stable et peut facilement être conçu pour rapprocher une réponse de fréquence donnée.

Conception du filtre

Fondamentalement, le but de processus de conception de filtre est de calculer les coefficients du filtre. Les trois étapes suivantes résument ce processus de conception.

1- Déterminer la réponse du filtre, dans ce cas un filtre passe-bas.
2- Identifier les différents paramètres de conception, ces paramètres dépendent de la réponse du filtre. Les spécifications générales d'un filtre passe-bas sont la bande passante, la fréquence de coupure et l'ordre du filtre.
3- Sélectionner la méthode de calcul (algorithme). Dans cette application, la méthode de la fenêtre est choisie.

Une fois les paramètres du filtre soigneusement choisis, l'outil de conception et d'analyse du filtre (FDATool) fourni par Matlab pourrait être utilisé pour obtenir les coefficients afin de filtrer et d'analyser sa réponse. La figure 19 montre l'interface

graphique de l'outil où l'utilisateur sélectionne la réponse du filtre, les spécifications et l'algorithme.

L'interface graphique est invoquée en écrivant (FDATool) dans la fenêtre de commandes Matlab. L'outil génère un fichier (code C) qui contient les coefficients du filtre conçu, ce qui facilite l'implémentation dans un environnement de programmation C. Les figures 20 et 21 montrent les réponses d'un filtre passe-bas (60 Hz) du vingtième ordre.

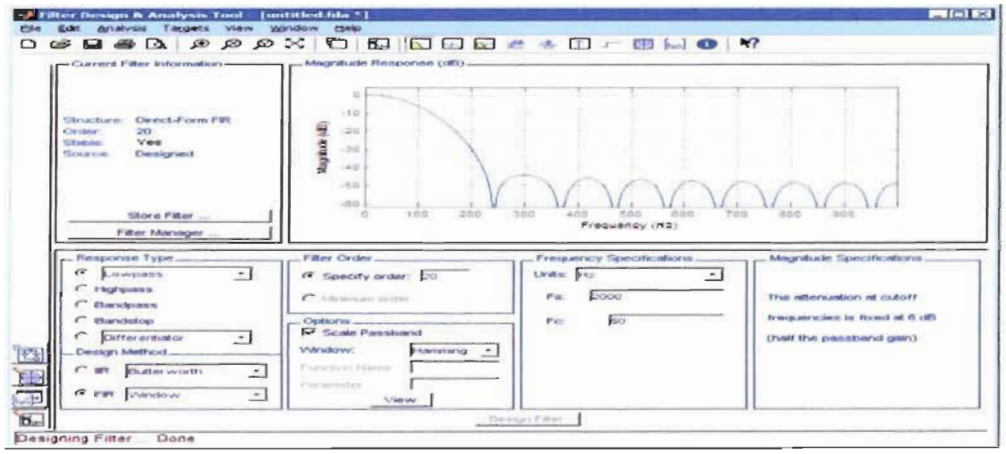


Figure 19 Interface graphique de FDATool

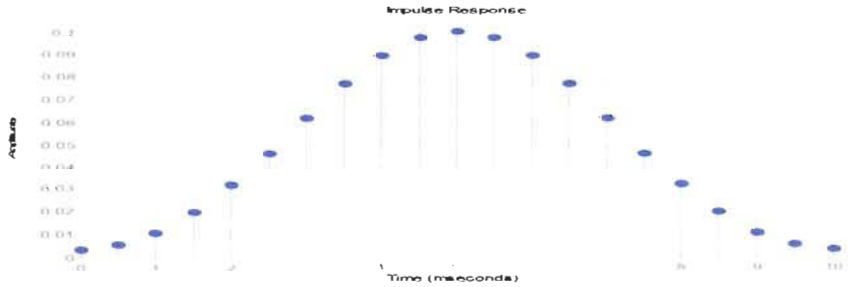


Figure 20 Réponse impulsionnelle du filtre

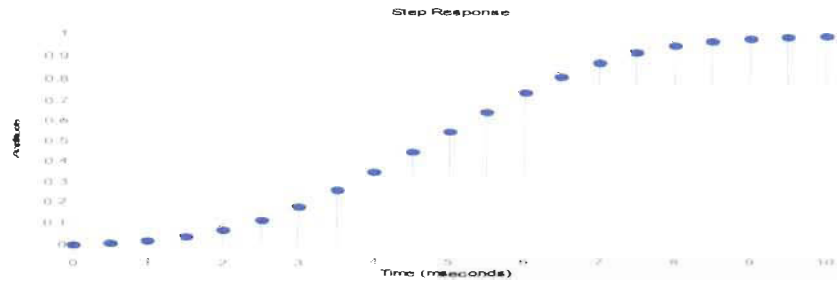


Figure 21 Réponse indicielle du filtre

Une fois les coefficients calculés, nous déclarons les variables de filtre au début du programme, comme indiqué dans les lignes de code suivant.

```
#define FIR_ORDER   50
 // Create Instances of FIRFILT_GEN module and
 // place the object in "AfilA" & "filB" section
#pragma DATA_SECTION(filtA,"AfilA");
   FIR16 filtA= FIR16_DEFAULTS;
#pragma DATA_SECTION(filtB, "filB");
   FIR16 filtB= FIR16_DEFAULTS;
 // Define the Delay buffers for the 50th order filter and
// place it in "bufA"& "DbufB" section
#pragma DATA_SECTION(dbuffer1,"bufA");
 long dbuffer1[(FIR_ORDER+2)/2 ];
#pragma DATA_SECTION(dbuffer2,"DbufB");
 long dbuffer2[(FIR_ORDER+2)/2 ];
// Define Constant Co-efficient Array  and
// place the .constant section in ROM memory
 const long coeff[(FIR_ORDER+2)/2]= FIR16_COEFF;
```

Ensuite, les modules de filtrage sont initialisés dans le programme principal. Le code suivant initialise deux modules de filtrage pour phase (A) et phase (B) du courant.

```
//--------Phase A filter
   filtA.coeff_ptr=(long *)coeff;
   filtA.dbuffer_ptr=dbuffer1;
   filtA.order= FIR_ORDER;;
   filtA.init(&filtA);
//--------Phase B filter
   filtB.coeff_ptr=(long *)coeff;
   filtB.dbuffer_ptr=dbuffer2;
   filtB.order= FIR_ORDER;
   filtB.init(&filtB);
```

Enfin, les sorties du convertisseur analogique numérique sont acheminées vers les entrées de deux modules de filtres ensuite les fonctions de calcul des filtres sont appelées comme indiqué ci-dessous

```
//---------------Phase A filter-------------------
        filtA.input= ilg2_vdc1.ImeasA ;
        filtA.calc(&filtA);

//---------------Phase A filter-------------------
       filtB.input= ilg2_vdc1.ImeasB;
        filtB.calc(&filtB);
```

Mise en ouvre expérimentale

Les avancées technologiques dans le domaine des microprocesseurs et des processeurs numériques de signal (DSP) ont contribué de manière significative au développement des entraînements électriques à haute performance, car il est devenu possible d'introduire des algorithmes de contrôle relativement complexes dans ces entraînements [1]. De nos jours, les fabricants offrent une large gamme dont des modules de développement pour évaluer certaines caractéristiques d'un DSP spécifique. Un module d'évaluation nommé eZdspTM est utilisé dans cette application. eZdspTM F2812 est une carte qui sert d'excellente plateforme pour développer des applications pour le processeur TMS320F2812.

Texas Instruments (TI) fournit un outil de développement (Code Composer Studio-CCS) qui permet la réalisation de certaines étapes cruciales du processus de conception des applications de processeur numérique de signal. Figure 22 montre les différentes fenêtres de CCS.

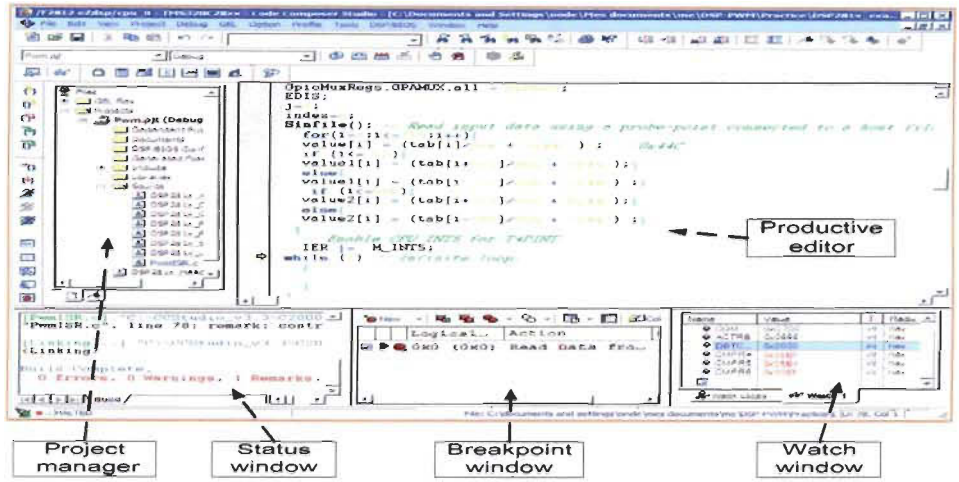


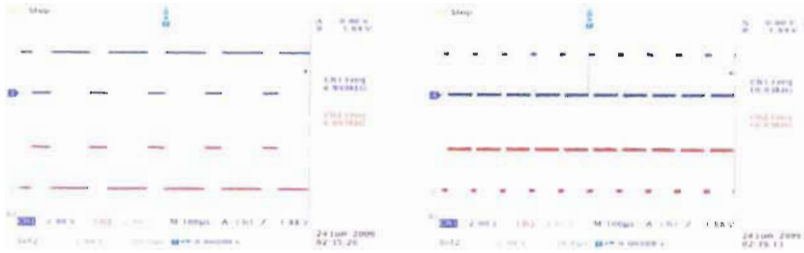
Figure 22 Code Composer Studio

Dans un premier temps, un algorithme flexible pour générer des signaux de commande pour un onduleur de tension triphasé est présenté. L'algorithme est basé sur la technique MLI sinusoïdale codée en C++. La flexibilité de l'algorithme réside dans la facilité de modifier la fréquence fondamentale et la fréquence de commutation. Il suffit de changer la valeur d'un registre d'horloge pour obtenir la fréquence désirée.

Le processus de développement de l'algorithme, l'organigramme et les résultats expérimentaux sont présentés. L'implémentation de l'algorithme et la plateforme expérimentale visent à tester l'onduleur triphasé et les circuits auxiliaires qui seront utilisés ultérieurement dans la réalisation de la commande vectorielle.

Résultats expérimentaux

La figure 23 illustre une séquence des signaux MLI (phase A) pour deux différentes fréquences de commutation, 5 kHz (Figure 23a) et 10 kHz (Figure 23b). Des résistances (30Ω) en série avec des inducteurs (30mH) sont utilisées comme charge de l'onduleur. La valeur de tension de bus continu est 115V, et la charge est connectée en triangle. Figure 24 et Figure 25 illustrent respectivement les formes d'ondes de la tension et du courant de l'onduleur pour les fréquences de 5 kHz et 10 kHz.



a) 5 kHz b) 10 kHz

Figure 23 Signaux MLI

Figure 24 Tension

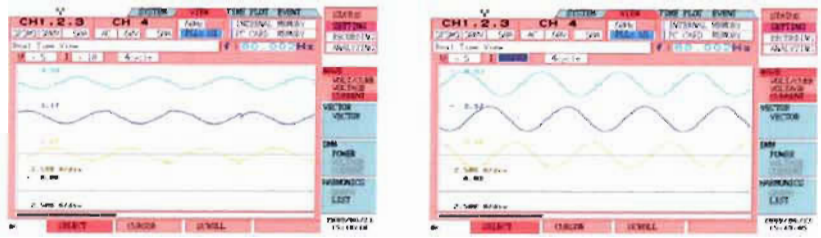


Figure 25 Courant

Commande sans capteur

La figure 26 montre la représentation synoptique de la plate forme expérimentale.

La carte eZdsp génère six signaux MLI. Ces signaux sont adaptés au circuit de commande de l'onduleur (12V) à l'aide du circuit d'interface. L'onduleur triphasé alimente le moteur asynchrone d'une tension triphasée à courant alternatif de l'ampleur et de la fréquence appropriées pour contrôler la vitesse du moteur. La tension de bus CC est mesurée et utilisée en conjonction avec les états des interrupteurs pour reconstruire la tension triphasée. Les courants de deux phases sont mesurés en utilisant des capteurs de courant et ensuite, ces courants sont adaptés à l'entrée du convertisseur analogique numérique en utilisant le circuit d'amplification du courant.

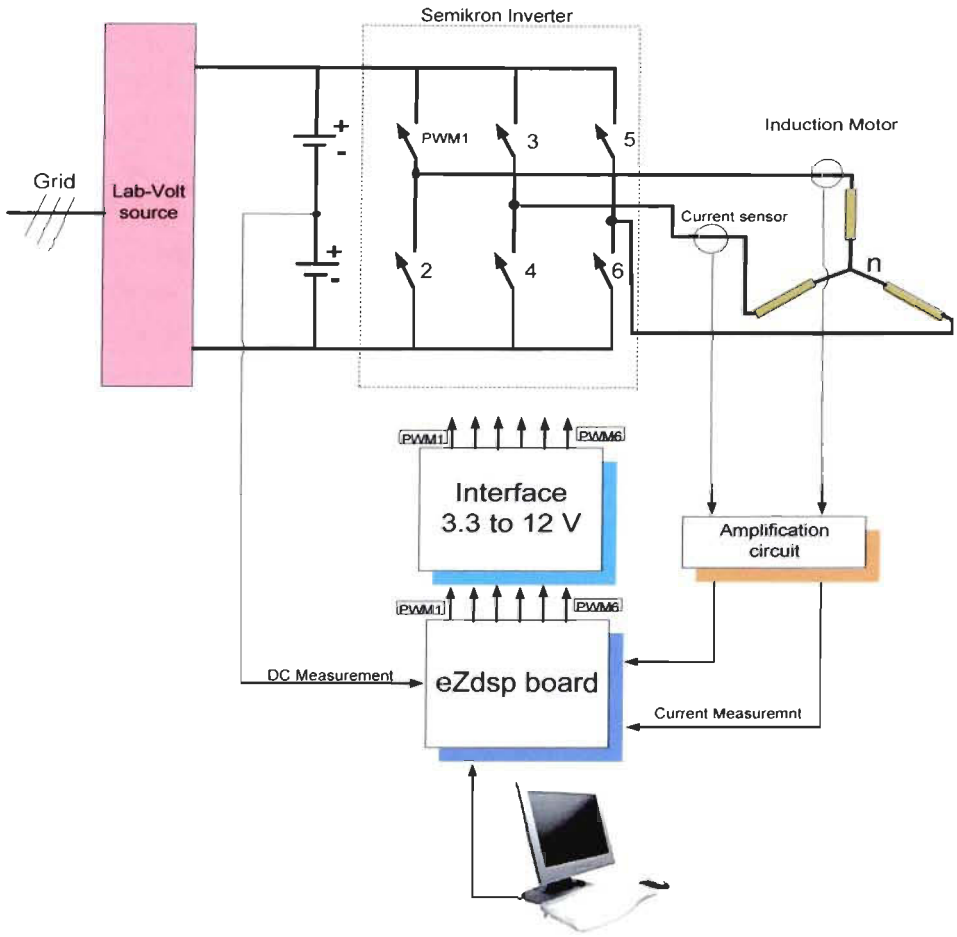


Figure 26 Plate forme expérimentale.

La figure 27 illustre l'organigramme de logiciel qui se compose d'une routine principale où les registres de contrôle et les blocs logiciels modulaires sont initialisés et une routine d'interruption où les modules de commande vectorielle sont exécutés à chaque interruption.

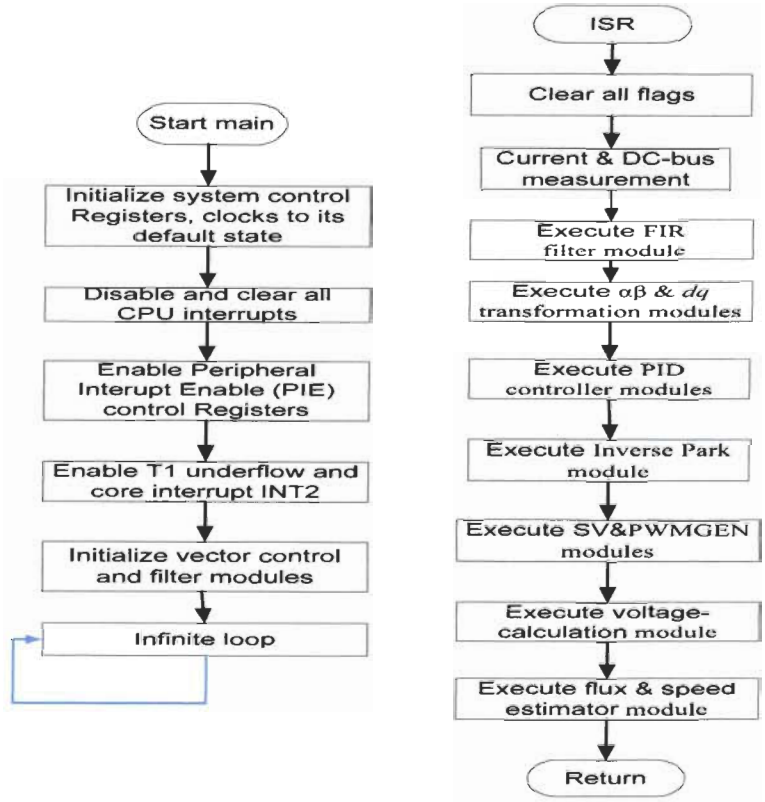


Figure 27 Organigramme du logiciel

Un codeur optique est utilisé pour mesurer la vitesse du moteur afin de la comparer avec celle estimée et de vérifier si le moteur est capable de suivre la vitesse de référence ou non. Un programme LabVIEW est développé pour compter les pulsations du codeur, calculer la vitesse du moteur et enregistrer le résultat dans un fichier Excel. Les mesures de vitesse enregistrées sont tracées en utilisant Matlab.

Résultats expérimentaux

La figure 28 illustre le courant avant et après le filtrage d'une phase du moteur.

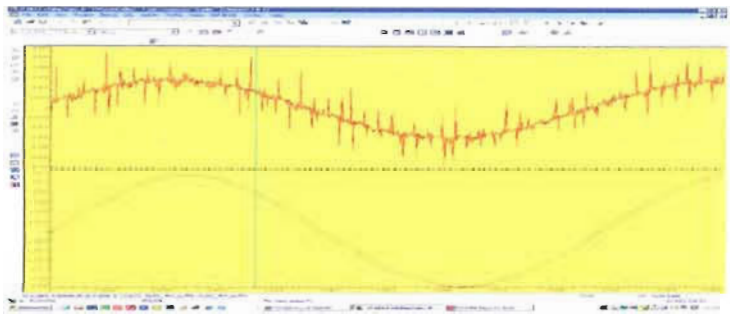


Figure 28 Courant d'une phase du moteur

La figure 29 montre la vitesse mesurée et la consigne de vitesse. On voit que la vitesse du moteur suit la référence demandée.

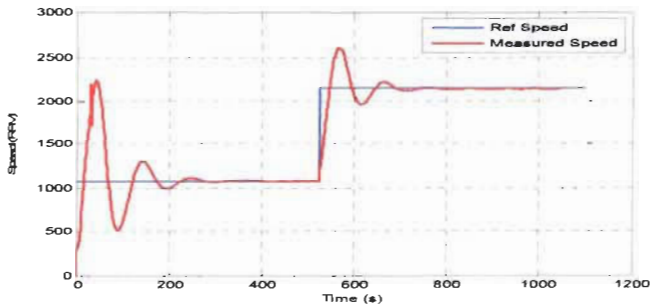


Figure 29 Vitesse du moteur

La figure 30 illustre la tension statorique aux bornes du moteur.

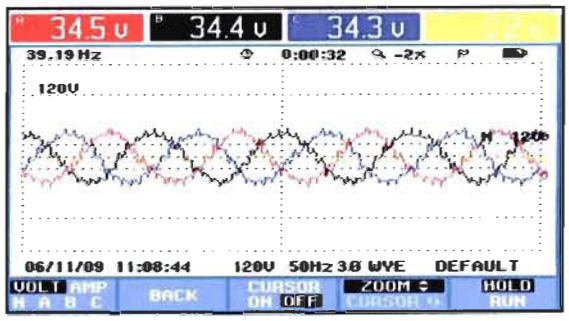


Figure 30 Tension statorique du moteur

Conclusion

Ce mémoire présente la conception et la mise en ouvre d'un variateur de vitesse pour un moteur asynchrone en utilisant un processeur numérique de signal. La commande repose sur une technique de commande vectorielle sans capteur. Le travail inclut également la conception et l'application d'un filtre numérique dans l'algorithme de commande pour filtrer les courants du moteur.

Les différentes étapes pour atteindre les objectifs du projet sont détaillées. Le travail commence par une recherche bibliographique. Puis l'étude des concepts de base et des percées récentes dans le domaine des entraînements électriques, des filtres numériques et des processeurs numériques du signal. En suite la modélisation, et la simulation d'un entraînement électrique. Enfin, la programmation et l'implémentation.

Les résultats expérimentaux obtenus démontrent la validité du montage de l'entraînement électrique ainsi que l'algorithme de commande.

B- Sinusoidal PWM Code

```
// Creating a Sine Modulated PWM Signal

#include "DSP281x_Device.h"
#include "DSP281x_Examples.h"
#include "volume.h"
int  tab[751],value[751],value1[751],value2[751];
int i, j,k,m, index;
long tmp;
static void Sinfile(); //function to read sine value

interrupt void evb_timer4_isr(void);
void init_evb_timer4(void);
Uint32  EvbTimer4InterruptCount;

void main(void)
{

// Initialize System Control registers, PLL, WatchDog, Clocks to default state:
InitSysCtrl();
EALLOW;
SysCtrlRegs.HISPCP.all = 0x3; // HSPCLK = SYSCLKOUT/6
EDIS;
// Disable and clear all CPU interrupts:
DINT;

// Initialize Pie Control Registers To Default State:
InitPieCtrl();

// Disable CPU interrupts and clear all CPU interrupt flags:
IER = 0x0000;
IFR = 0x0000;
// Initialize the PIE Vector Table To a Known State:
InitPieVectTable();

EALLOW;
PieVectTable.T4PINT = &evb_timer4_isr;
EDIS;
//Init_evb_timer4();
 init_evb_timer4();
 // Initialize count values to 0
  EvbTimer4InterruptCount = 0;
 // Enable PIE group 5 interrupt 1 for T4PINT
  PieCtrlRegs.PIEIER5.all = M_INT1;

 // Enable global Interrupts and higher priority real-time debug events:
  EINT;   // Enable Global interrupt INTM
  ERTM;   // Enable Global realtime interrupt DBGM
```

```
// Initialize EVB Timer3
EvbRegs.T3PR =0x09C4 ; //0x5555;0x9C4 //0EA6; // Timer3 period
EvbRegs.T3CNT = 0x0000; // Timer3 counter
EvbRegs.T3CON.all = 0x1042; // Timer enable
// Compare action control.
EvbRegs.ACTRB.all = 0x0666; // Action control register
EvbRegs.DBTCONB.bit.EDBT2 = 0; // Disable deadband
EvbRegs.COMCONB.all = 0xC600; //Compare operation modes
EALLOW; // Enable PWM pins.
GpioMuxRegs.GPBMUX.all = 0x00FF;
GpioMuxRegs.GPAMUX.all = 0x0000;
EDIS;
j=0;
index=0;
Sinfile(); // Read input data using a probe-point connected to a host file.
  for(i=0;i<=750;i++){
  value[i] = (tab[i]/0xA + 0x44C ) ;  //0x44C
  if (i<=250){
  value1[i] = (tab[i+501]/0xA + 0x44C);}
  else{
  value1[i] = (tab[i-250]/0xA + 0x44C) ;}
   if (i<=500){
  value2[i] = (tab[i+251]/0xA + 0x44C);}
  else{
  value2[i] = (tab[i-500]/0xA + 0x44C) ;}
 }
  // Enable CPU INT5 for T4PINT
  IER |=  M_INT5;
while (1) // Infinite loop.
  {

  }
}

// read input signal
static void Sinfile()
{
return;
 }
```

```
void init_evb_timer4(void)
{
    // Initialize EVB Timer 4:
    // Setup Timer 4 Registers (EV B)
    EvbRegs.GPTCONB.all = 0;

    // Set the Period for the GP timer 4 to 0x0200;
    EvbRegs.T4PR = 0x022B; //0x4E2; //0x1000;        // Period
    EvbRegs.T4CMPR = 0x0000;      // Compare Reg

    // Enable Period interrupt bits for GP timer 4
    // Count up, x128, internal clk, enable compare, use own period
    EvbRegs.EVBIMRB.bit.T4PINT = 1;
    EvbRegs.EVBIFRB.bit.T4PINT = 1;

    // Clear the counter for GP timer 4
    EvbRegs.T4CNT = 0x0000;
    EvbRegs.T4CON.all = 0x1042; //0x1042; //0x1742;

    // Start EVA ADC Conversion on timer 4 Period interrupt
    //EvbRegs.GPTCONB.bit.T4TOADC = 2;
}

interrupt void evb_timer4_isr(void)
{
   EvbTimer4InterruptCount++;
   index++;
   // Note: To be safe, use a mask value to write to the entire
   // EVBIFRB register.  Writing to one bit will cause a read-modify-write
   // operation that may have the result of writing 1's to clear
   // bits other then those intended.
   EvbRegs.EVBIFRB.all = BIT0;

   // Enable compare
   EvbRegs.CMPR4 = value[index];
   EvbRegs.CMPR5 = value1[index];
   EvbRegs.CMPR6 = value2[index];

   // Acknowledge interrupt to receive more interrupts from PIE group 5
   PieCtrlRegs.PIEACK.all = PIEACK_GROUP5;

    if (index>751){
     index=0;      }
     EvbRegs.EVBIFRB.bit.T4PINT = 1; //Clear T4INT flag

}
```

C- Sensorless vector control Code

```
// Include header files used in the main function
#include "target.h"
#include "DSP281x_Device.h"
#include "IQmathLib.h"
#include "aci3_4.h"
#include "parameter.h"
#include "build.h"
//#include "filter.h"
//#include "fir.h"
#include <math.h>
#include <fir.h>
  // ---------------start filter-----------------------------------------------------------
// FIR16 filtA,filtB;
  //FIR16 filtA= FIR16_DEFAULTS;
  //FIR16 filtB= FIR16_DEFAULTS;
  // Filter Symbolic Constants
#define FIR_ORDER   50
 // Create an Instance of FIRFILT_GEN module and place the object in "firfilt"
section
#pragma DATA_SECTION(filtA,"AfilA");
 FIR16 filtA= FIR16_DEFAULTS;
#pragma DATA_SECTION(dbuffer1,"bufA");
long dbuffer1[(FIR_ORDER+2)/2 ];
#pragma DATA_SECTION(filtB, "filB");
 FIR16 filtB= FIR16_DEFAULTS;
#pragma DATA_SECTION(dbuffer2,"DbufB");
 long dbuffer2[(FIR_ORDER+2)/2 ];
// Define the Delay buffer for the 50th order filterfilter and place it in "firldb"
section
// Define Constant Co-efficient Array  and place the .constant section in ROM
memory
  const long coeff[(FIR_ORDER+2)/2]= FIR16_COEFF;         //FIR16_COEFF;
 //long const coefB[(FIR_ORDER+0)/2]= FIR16_LPF20;
//////////:::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::::

   //-------------------End filter-------------------------------------------------------
```

```
interrupt void MainISR(void);

// Global variables used in this system
float32 VdTesting = 0.25;        // Vd testing (pu)
float32 VqTesting = 0.0;         // Vq testing (pu)
float32 IdRef = 0.203; //0.15          // Id reference (pu)
float32 IqRef = 0.05; //0.05          // Iq reference (pu)
float32 SpeedRef = 0.3;          // Speed reference (pu)
float32 T = 0.001/ISR_FREQUENCY;  // Samping period (sec), see
parameter.h0.001
Uint16 IsrTicker = 0;
Uint16 BackTicker = 0;

int16 PwmDacCh1 = 0;
int16 PwmDacCh2 = 0;
int16 PwmDacCh3 = 0;

int16 DlogCh1 = 0;
int16 DlogCh2 = 0;
int16 DlogCh3 = 0;
int16 DlogCh4 = 0;


volatile Uint16 EnableFlag = FALSE;

Uint16 SpeedLoopPrescaler = 20;     // Speed loop prescaler
Uint16 SpeedLoopCount = 1;              // Speed loop counter

// Instance rotor flux and speed estimations
ACIFE fe1 = ACIFE_DEFAULTS;
ACISE se1 = ACISE_DEFAULTS;

// Instance a few transform objects
CLARKE clarke1 = CLARKE_DEFAULTS;
PARK park1 = PARK_DEFAULTS;
IPARK ipark1 = IPARK_DEFAULTS;

// Instance PID regulators to regulate the d and q synchronous axis currents,
// and speed
PIDREG3 pid1_id = PIDREG3_DEFAULTS;
PIDREG3 pid1_iq = PIDREG3_DEFAULTS;
PIDREG3 pid1_spd = PIDREG3_DEFAULTS;

// Instance a PWM driver instance
PWMGEN pwm1 = PWMGEN_DEFAULTS;

// Instance a PWM DAC driver instance
PWMDAC pwmdac1 = PWMDAC_DEFAULTS;
```

```
// Instance a Space Vector PWM modulator. This modulator generates a, b and
c
// phases based on the d and q stationery reference frame inputs
SVGENDQ svgen_dq1 = SVGENDQ_DEFAULTS;

// Instance a Capture interface driver
CAPTURE cap1 = CAPTURE_DEFAULTS;

// Instance a speed calculator based on capture
SPEED_MEAS_CAP speed1 = SPEED_MEAS_CAP_DEFAULTS;

// Instance a enable PWM drive driver (only for DMC1500)
DRIVE drv1 = DRIVE_DEFAULTS;

// Instance a ramp controller to smoothly ramp the frequency
RMPCNTL rc1 = RMPCNTL_DEFAULTS;

//      Instance a ramp generator to simulate an Anglele
RAMPGEN rg1 = RAMPGEN_DEFAULTS;

//      Instance a phase voltage calculation
PHASEVOLTAGE volt1 = PHASEVOLTAGE_DEFAULTS;

// Create an instance of the current/dc-bus voltage measurement driver
ILEG2DCBUSMEAS ilg2_vdc1 = ILEG2DCBUSMEAS_DEFAULTS;

// Instance the constant calculations for rotor flux and speed estimations
ACIFE_CONST fe1_const = ACIFE_CONST_DEFAULTS;
ACISE_CONST se1_const = ACISE_CONST_DEFAULTS;

// Create an instance of DATALOG Module
DLOG_4CH dlog = DLOG_4CH_DEFAULTS;


void main(void)
{

// Initialize System Control registers, PLL, WatchDog, Clocks to Alphafault
state:
      // This function is found in the DSP281x_SysCtrl.c file.
       InitSysCtrl();

// HISPCP prescale register settings, normally it will be set to Alphafault values
   EALLOW;   // This is neeAlphad to write to EALLOW protected registers
   SysCtrlRegs.HISPCP.all = 0x0000;     // SYSCLKOUT/1
   EDIS;   // This is neeAlphad to disable write to EALLOW protected registers
```

```
// Disable and clear all CPU interrupts:
        DINT;
        IER = 0x0000;
        IFR = 0x0000;

// Initialize Pie Control Registers To default State:
      // This function is found in the DSP281x_PieCtrl.c file.
        InitPieCtrl();

// Initialize the PIE Vector Table To a Known State:
      // This function is found in DSP281x_PieVect.c.
        // This function populates the PIE vector table with pointers
      // to the shell ISR functions found in DSP281x_DefaultIsr.c.
        InitPieVectTable();

// User specific functions, Reassign vectors (optional), Enable Interrupts:

// Initialize EVA Timer 1:
   // Setup Timer 1 Registers (EV A)
   EvaRegs.GPTCONA.all = 0;

  // Waiting for enable flag set
  while (EnableFlag==FALSE)
   {
     BackTicker++;
   }

// Enable Underflow interrupt bits for GP timer 1
   EvaRegs.EVAIMRA.bit.T1UFINT = 1;
   EvaRegs.EVAIFRA.bit.T1UFINT = 1;

// Reassign ISRs.
      // Reassign the PIE vector for T1UFINT to point to a different
      // ISR then the shell routine found in DSP281x_DefaultIsr.c.
      // This is done if the user does not want to use the shell ISR routine
      // but instead wants to use their own ISR.

        EALLOW;     // This is neeAlphad to write to EALLOW protected registers
        PieVectTable.T1UFINT = &MainISR;
        EDIS;   // This is neeAlphad to disable write to EALLOW protected
registers

// Enable PIE group 2 interrupt 6 for T1UFINT
   PieCtrlRegs.PIEIER2.all = M_INT6;

// Enable CPU INT2 for T1UFINT:
        IER |= M_INT2;
```

```
// Initialize PWM module
   pwm1.PeriodMax = SYSTEM_FREQUENCY*1000000*T/2;  // Perscaler X1
(T1), ISR period = T x 1
        pwm1.init(&pwm1);

// Initialize PWMDAC module
   pwmdac1.PeriodMax = (SYSTEM_FREQUENCY*200/(30*2))*5;   // PWMDAC
Frequency = 30 kHz
   pwmdac1.PwmDacInPointer0 = &PwmDacCh1;
   pwmdac1.PwmDacInPointer1 = &PwmDacCh2;
   pwmdac1.PwmDacInPointer2 = &PwmDacCh3;
        pwmdac1.init(&pwmdac1);

// Initialize DATALOG module
   dlog.iptr1 = &DlogCh1;
   dlog.iptr2 = &DlogCh2;
   dlog.iptr3 = &DlogCh3;
   dlog.iptr4 = &DlogCh4;
   dlog.trig_value = 0x0;
   dlog.size = 0x400;
   dlog.prescalar = 1;
   dlog.init(&dlog);

// Initialize capture module
   cap1.init(&cap1);

// Initialize enable drive module (FOR DMC1500 ONLY)
   drv1.init(&drv1);

// Initialize ADC module
   ilg2_vdc1.init(&ilg2_vdc1);

// Initialize the SPEED_PR module

   // x128-T2, 150MHz, 1000-teeth sprocket
   speed1.InputSelect = 0;
        speed1.BaseRpm = 120*BASE_FREQ/P;
        speed1.SpeedScaler =
60*(SYSTEM_FREQUENCY*1000000/1000)*1/(128*speed1.BaseRpm);

// Initialize RAMPGEN module
   rg1.StepAngleMax = _IQ(BASE_FREQ*T);
```

```
// Initialize the ACI constant module
        fe1_const.Rs = RS;
        fe1_const.Rr = RR;
        fe1_const.Ls = LS;
        fe1_const.Lr = LR;
        fe1_const.Lm = LM;
        fe1_const.Ib = BASE_CURRENT;
        fe1_const.Vb = BASE_VOLTAGE;
        fe1_const.Ts = T;
        fe1_const.calc(&fe1_const);

// Initialize the ACI module
        fe1.K1 = _IQ(fe1_const.K1);
        fe1.K2 = _IQ(fe1_const.K2);
        fe1.K3 = _IQ(fe1_const.K3);
        fe1.K4 = _IQ(fe1_const.K4);
        fe1.K5 = _IQ(fe1_const.K5);
        fe1.K6 = _IQ(fe1_const.K6);
        fe1.K7 = _IQ(fe1_const.K7);
        fe1.K8 = _IQ(fe1_const.K8);
    fe1.Kp = _IQ(0.05);
    fe1.Ki = _IQ(T/0.45);

// Initialize the ACI constant module
        se1_const.Rr = RR;
        se1_const.Lr = LR;
        se1_const.fb = BASE_FREQ;
        se1_const.fc = 3;
        se1_const.Ts = T;
        se1_const.calc(&se1_const);

// Initialize the ACI module
        se1.K1 = _IQ(se1_const.K1);
        se1.K2 = _IQ21(se1_const.K2);
        se1.K3 = _IQ(se1_const.K3);
        se1.K4 = _IQ(se1_const.K4);
        se1.BaseRpm = 120*BASE_FREQ/P;

// Initialize the PID_REG3 module for Id
        pid1_id.Kp = _IQ(0.3125);
        pid1_id.Ki = _IQ(T/5);
        pid1_id.Kd = _IQ(0/T);
        pid1_id.Kc = _IQ(0.0625);
    pid1_id.OutMax = _IQ(0.7071);
    pid1_id.OutMin = _IQ(-0.7071);
```

```
// Initialize the PID_REG3 module for Iq
        pid1_iq.Kp = _IQ(0.3125);
        pid1_iq.Ki = _IQ(T/5);
        pid1_iq.Kd = _IQ(0/T);
        pid1_iq.Kc = _IQ(0.0625);
    pid1_iq.OutMax = _IQ(0.7071);
    pid1_iq.OutMin = _IQ(-0.7071);

// Initialize the PID_REG3 module for speed
    pid1_spd.Kp = _IQ(0.375);
        pid1_spd.Ki = _IQ(T*SpeedLoopPrescaler/6);//0.1666
        pid1_spd.Kd = _IQ(0/(T*SpeedLoopPrescaler));

        pid1_spd.Kc = _IQ(0.0625);
    pid1_spd.OutMax = _IQ(0.5);
    pid1_spd.OutMin = _IQ(-0.5);

// Enable global Interrupts and higher priority real-time debug events:
        EINT;   // Enable Global interrupt INTM
        ERTM;         // Enable Global realtime interrupt DBGM

// FIR Generic Filter Initialisation

     //--------Phase A filter
          filtA.coeff_ptr=(long *)coeff;
          filtA.dbuffer_ptr=dbuffer1;
          filtA.order= FIR_ORDER;;
     filtA.init(&filtA);
       //--------Phase B filter
          filtB.coeff_ptr=(long *)coeff;
          filtB.dbuffer_ptr=dbuffer2;
          filtB.order= FIR_ORDER;
     filtB.init(&filtB);

// IDLE loop. Just sit and loop forever:
        for(;;) BackTicker++;
}
```

```
interrupt void MainISR(void)
{

// Verifying the ISR
   IsrTicker++;

// ---------------------------------------------------------------------------
//   Call the ILEG2_VDC read function.
// ---------------------------------------------------------------------------
   ilg2_vdc1.read(&ilg2_vdc1);

//..... filter implementation.....
//---------------Phase A filter--------------------


         filtA.input= ilg2_vdc1.ImeasA ;
         filtA.calc(&filtA);

//---------------Phase A filter--------------------
        filtB.input= ilg2_vdc1.ImeasB;
         filtB.calc(&filtB);

// ---------------------------------------------------------------------------
//   Connect inputs of the CLARKE module and call the clarke transformation
//   calculation function.
// ---------------------------------------------------------------------------
        clarke1.As = _IQ15toIQ((int32)filtA.output);
        clarke1.Bs = _IQ15toIQ((int32)filtB.output);
        clarke1.calc(&clarke1);

// ---------------------------------------------------------------------------
//   Connect inputs of the PARK module and call the park transformation
//   calculation function.
// ---------------------------------------------------------------------------
   park1.Alpha = clarke1.Alpha;
   park1.Beta = clarke1.Beta;
   park1.Angle = fe1.ThetaFlux;
   park1.calc(&park1);

// ---------------------------------------------------------------------------
```

```
//   Connect inputs of the PID_REG3 module and call the PID speed controller
//   calculation function.
// ---------------------------------------------------------------------------
  if (SpeedLoopCount==SpeedLoopPrescaler)
    {
     pid1_spd.Ref = _IQ(SpeedRef);
     pid1_spd.Fdb = se1.WrHat;
          pid1_spd.calc(&pid1_spd);
     SpeedLoopCount=1;
    }
   else SpeedLoopCount++;

// ---------------------------------------------------------------------------
//   Connect inputs of the PID_REG3 module and call the PID IQ controller
//   calculation function.
// ---------------------------------------------------------------------------
   pid1_iq.Ref = pid1_spd.Out;
        pid1_iq.Fdb = park1.Qs;
        pid1_iq.calc(&pid1_iq);

// ---------------------------------------------------------------------------
//   Connect inputs of the PID_REG3 module and call the PID ID controller
//   calculation function.
// ---------------------------------------------------------------------------
   pid1_id.Ref = _IQ(IdRef);
        pid1_id.Fdb = park1.Ds;
        pid1_id.calc(&pid1_id);

// ---------------------------------------------------------------------------
//   Connect inputs of the INV_PARK module and call the inverse park
transformation
//   calculation function.
// ---------------------------------------------------------------------------
   ipark1.Ds = pid1_id.Out;
   ipark1.Qs = pid1_iq.Out;
   ipark1.Angle = fe1.ThetaFlux;
   ipark1.calc(&ipark1);

// ---------------------------------------------------------------------------
//   Connect inputs of the SVGEN_DQ module and call the space-vector gen.
//   calculation function.
// ---------------------------------------------------------------------------
        svgen_dq1.Ualpha = ipark1.Alpha;
        svgen_dq1.Ubeta = ipark1.Beta;
        svgen_dq1.calc(&svgen_dq1);

// ---------------------------------------------------------------------------
```

```
//   Connect inputs of the PWM_DRV module and call the PWM signal
generation
//   update function.
// -----------------------------------------------------------------------
   pwm1.MfuncC1 = (int16)_IQtoIQ15(svgen_dq1.Ta); // MfuncC1 is in Q15
   pwm1.MfuncC2 = (int16)_IQtoIQ15(svgen_dq1.Tb); // MfuncC2 is in Q15
   pwm1.MfuncC3 = (int16)_IQtoIQ15(svgen_dq1.Tc); // MfuncC3 is in Q15
       pwm1.update(&pwm1);
// -----------------------------------------------------------------------
//   Connect inputs of the VOLT_CALC module and call the phase voltage
//   calculation function.
// -----------------------------------------------------------------------
   volt1.DcBusVolt = _IQ15toIQ((int32)ilg2_vdc1.VdcMeas);
   volt1.MfuncV1 = svgen_dq1.Ta;
   volt1.MfuncV2 = svgen_dq1.Tb;
   volt1.MfuncV3 = svgen_dq1.Tc;
       volt1.calc(&volt1);

// -----------------------------------------------------------------------
//   Connect inputs of the ACI module and call the flux estimation
//   calculation function.
// -----------------------------------------------------------------------
       fe1.UDsS = volt1.Valpha;
       fe1.UQsS = volt1.Vbeta;
       fe1.IDsS = clarke1.Alpha;
       fe1.IQsS = clarke1.Beta;
       fe1.calc(&fe1);

// -----------------------------------------------------------------------
//   Connect inputs of the ACI module and call the speed estimation
//   calculation function.
// -----------------------------------------------------------------------
       se1.IDsS = clarke1.Alpha;
       se1.IQsS = clarke1.Beta;
       se1.PsiDrS = fe1.PsiDrS;
       se1.PsiQrS = fe1.PsiQrS;
       se1.ThetaFlux = fe1.ThetaFlux;
       se1.calc(&se1);

// -----------------------------------------------------------------------
```

```
//    Connect inputs of the SPEED_PR module and call the speed calculation
function
// ---------------------------------------------------------------------------


   if((cap1.read(&cap1))==0)            // Call the capture read function
   {
      speed1.TimeStamp=(int32)(cap1.TimeStamp);    // Read out new time
stamp
      speed1.calc(&speed1);               // Call the speed calulator
   }
// ---------------------------------------------------------------------------
//    Connect inputs of the PWMDAC module
// ---------------------------------------------------------------------------
   PwmDacCh1 = (int16)_IQtoIQ15(svgen_dq1.Ta);
   PwmDacCh2 = (int16)_IQtoIQ15(clarke1.As);
   PwmDacCh3 = (int16)_IQtoIQ15(fe1.ThetaFlux);

// ---------------------------------------------------------------------------
//    Connect inputs of the DATALOG module
// ---------------------------------------------------------------------------
   DlogCh1 = (int16)_IQtoIQ15(svgen_dq1.Ta);
   DlogCh2 = (int16)_IQtoIQ15(fe1.ThetaFlux);
   DlogCh3 = (int16)_IQtoIQ15(pid1_spd.Ref);
   DlogCh4 = (int16)_IQtoIQ15(pid1_spd.Fdb);

// ---------------------------------------------------------------------------
//    Connect inputs of the EN_DRV module and call the enable/disable PWM
signal
//    update function. (FOR DMC1500 ONLY)
// ---------------------------------------------------------------------------
   drv1.EnableFlag = EnableFlag;
   drv1.update(&drv1);



// ---------------------------------------------------------------------------
//    Call the PWMDAC update function.
// ---------------------------------------------------------------------------
       pwmdac1.update(&pwmdac1);

// ---------------------------------------------------------------------------
```

```
//   Call the DATALOG update function.
// ---------------------------------------------------------------------------
   dlog.update(&dlog);
// Enable more interrupts from this timer
        EvaRegs.EVAIMRA.bit.T1UFINT = 1;
           EvaRegs.EVAIFRA.all = BIT9;

// Acknowledge interrupt to recieve more interrupts from PIE group 2
        PieCtrlRegs.PIEACK.all = PIEACK_GROUP2;
}
```

D- SVPWM Matlab Code

```
%Matlab Code to generate Switching functions
% Inputs are magnitude u1(:),angle u2(:)
% and ramp time signal for comparison u3(:)
function sf    = aaa(u)
ts=0.0002;Vdc= 1 ;peak_phase_max= Vdc/sqrt(3);
x=u(2); y=u(3); mag=(u(1)/peak_phase_max)*ts;
% sector I
sa=0; sb=0; sc=0;
if(x>=0)&&(x<pi/3)
   ta = mag*sin(pi/3-x);tb= mag*sin(x);
   t0 =(ts-ta-tb);
   t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
   v1=[0 1 1 1 1 1 0];v2=[0 0 1 1 1 0 0];v3=[0 0 0 1 0 0 0];
    for j=1:7
       if(y<t1(j))
       break
       end
    end
   sa=v1(j);sb=v2(j);sc=v3(j);
end
% sector II
if (x>=pi/3)&&(x<2*pi/3)
    adv=x-pi/3;
    tb = mag*sin(pi/3-adv);ta= mag*sin(adv);
    t0 =(ts-ta-tb);
    t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
    v1=[0 0 1 1 1 0 0];v2=[0 1 1 1 1 1 0];v3=[0 0 0 1 0 0 0];
    for j=1:7
        if(y<t1(j))
            break
        end
    end
    sa=v1(j);sb=v2(j);sc=v3(j);
end
 %sector III
   if(x>=2*pi/3)&&(x<pi)
      adv=x-2*pi/3;
      ta=mag*sin(pi/3-adv); tb=mag*sin(adv);
      t0 =(ts-ta-tb);
   t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
   v1=[0 0 0 1 0 0 0];v2=[0 1 1 1 1 1 0];v3=[0 0 1 1 1 0 0];
     for j=1 :7
         if(y<t1(j))
             break
         end
     end
     sa=v1(j);sb=v2(j);sc=v3(j);
   end
   %sector IV
    if(x>=-pi)&&(x<-2*pi/3)
    adv = x + pi;
    tb=mag*sin(pi/3 - adv);ta=mag*sin(adv);
    t0 =(ts-ta-tb);
    t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
```

```
v1=[0 0 0 1 0 0 0];v2=[0 0 1 1 1 0 0];v3=[0 1 1 1 1 1 0];
 for j=1:7
     if(y<t1(j))
         break
     end
 end
 sa=v1(j);sb=v2(j);sc=v3(j);
end
%sector V
if(x>=-2*pi/3)&&(x<-pi/3)
 adv = x + 2*pi/3;
ta= mag*sin(pi/3 - adv);tb= mag*sin(adv);
t0 =(ts-ta-tb);
t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
v1=[0 0 1 1 1 0 0];v2=[0 0 0 1 0 0 0];v3=[0 1 1 1 1 1 0];
 for j=1:7
     if(y<t1(j))
         break
     end
 end
sa=v1(j);sb=v2(j);sc=v3(j);
end
%sector VI
if(x>= -pi/3)&&(x<0)
 adv = x + pi/3;
tb= mag*sin(pi/3 - adv);ta= mag*sin(adv);
t0 =(ts-ta-tb);
t1=[t0/4 ta/2 tb/2 t0/2 tb/2 ta/2 t0/4];t1=cumsum(t1);
v1=[0 1 1 1 1 1 0];v2=[0 0 0 1 0 0 0];v3=[0 0 1 1 1 0 0];
  for j=1:7
     if(y<t1(j))
         break
     end
  end
 sa=v1(j);sb=v2(j);sc=v3(j);
end
sf = [sa, sb, sc];
```

E- Simulated Machine parameters

```
%**********Simulated machine Parameters************************

Lm= 0.0403;          % Mutual inductance
Rr=0.25;             % Rotor resistance
Lr= 0.0412;          % Rotor inductance
Rs=0.3;              % Stator resistance
Ls=0.0415;           % Stator inductance
Beq=0.02;            % Friction coefficient
Jeq=0.1;             % Inertia coefficient
Tr=Lr/Rr;            % Rotor time constant
Vdc=360;             % Inverter DC bus voltage
```