2004

FPGA BASED IMPLEMENTATION OF A POSITION ESTIMATOR FOR CONTROLLING A SWITCHED RELUCTANCE MOTOR

Srilaxmi Pampana
University of Kentucky, spamp2@uky.edu

ABSTRACT OF THESIS

FPGA BASED IMPLEMENTATION OF A POSITION ESTIMATOR FOR CONTROLLING A SWITCHED RELUCTANCE MOTOR

Rotor Position information is essential in the operation of the Switched Reluctance Motor (SRM) for properly controlling its phase currents. This thesis uses Field Programmable Gate Array (FPGA) technology to implement a method to estimate the SRM's rotor position using the inverse inductance value of the SRM's phases. The estimated rotor position is given as input to the Commutator circuit, also implemented in the FPGA, to determine when torque-producing currents should be input in the SRM phase windings. The Estimator and Commutator design is coded using Verilog HDL and is simulated using Xilinx tools. This circuit is implemented on a Xilinx Virtex XCV800 FPGA system. The experimentally generated output is validated by comparing it with simulation results from a Simulink model of the Estimator. The performance of the FPGA based SRM rotor position estimator in terms of calculation time is compared to a digital signal processor (DSP) implementation of the same position estimator algorithm. It is found that the FPGA rotor position Estimator with a 5MHz clock can update its rotor position estimate every 7μs compared to an update time of 50μs for a TMS320C6701-150 DSP implementation using a commercial DSP board. This is a greater than 7 to one reduction in the update time.

KEYWORDS: Position Estimator, SRM, FPGA, Commutator, Verilog, Simulink, Xilinx.

Srilaxmi Pampana

October 26, 2004

FPGA BASED IMPLEMENTATION OF A POSITION ESTIMATOR FOR CONTROLLING A SWITCHED RELUCTANCE MOTOR

By

Srilaxmi Pampana

Dr. Arthur V. Radun
(Director of Thesis)

Dr. Yu Ming Zhang
(Director of Graduate Studies)

October 26, 2004

RULES FOR THE USE OF THESES

Unpublished theses submitted for the Master's degree and deposited in the University of Kentucky Library are as a rule open for inspection, but are to be used only with due regard to the rights of the authors. Bibliographical references may be noted, but quotations or summaries of parts may be published only with permission of the author, and with the usual scholarly acknowledgements.

Extensive copying or publication of the thesis in whole or in part also requires the consent of the Dean of the Graduate School of the University of Kentucky.

A library that borrows this thesis for use by its patrons is expected to secure the signature of each user.

Name Date

_____

_____

_____

_____

_____

_____

_____

_____

_____

_____

THESIS

Srilaxmi Pampana

The Graduate School

University of Kentucky

2004

FPGA BASED IMPLEMENTATION OF A POSITION ESTIMATOR FOR CONTROLLING A SWITCHED RELUCTANCE MOTOR

THESIS

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering at the University of Kentucky

By

Srilaxmi Pampana

Lexington, Kentucky

Director: Dr. Arthur Radun, Associate Professor Electrical Engineering, Lexington, Kentucky

2004

MASTER'S THESIS RELEASE

I authorize the University of Kentucky
Libraries to reproduce this thesis in
whole or in part for purposes of research

Signed: _____

Date: _____

ACKNOWLEDGEMENTS

I would like to express my sincere thanks and heartfelt gratitude to Dr. Arthur V Radun for his guidance and support throughout this thesis. I am very thankful for his constant encouragement and help in providing suggestions and insights towards my thesis.

I would also like to thank Dr. J. Robert Heath and Dr. William Dieter for serving on my thesis committee and providing me with invaluable comments and suggestions for improving this thesis.

I would like to express my deepest gratitude and thanks to my parents and my brother for their support and encouragement, they provided throughout my life and in finishing my Masters. I would also like to thank all my friends for all their help and encouragement.

TABLE OF CONTENTS

APPENDICES

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Introduction to Switched Reluctance Motor

The switched reluctance motor (SRM) is a doubly salient and singly excited machine with an unequal number of rotor and stator poles to avoid magnetic locking between the stator and rotor poles. The main advantages of Switched Reluctance motors are their simple construction due to the absence of magnets, rotor conductors, and brushes and high system efficiency over a wide speed range. However, the need for a direct rotor position sensor to commutate the current from phase to phase synchronously with rotor position has excluded the motor from many cost-sensitive applications.

For successful and reliable operation of the SRM, it is essential to know the rotor position accurately. For high performance SRM drives used in aircrafts, ships, and servo systems accurate rotor position is required to avoid initial starting hesitation. An encoder, resolver, or Hall sensor attached to the shaft is normally used to supply the rotor position, but the use of these sensors increases costs, decreases system reliability, and also increases the overall physical envelope of the motor drive and the number of motor wires. A variety of algorithms for sensorless control have been developed, most of which involve evaluation of the variation of magnetic circuit parameters that are dependent on the rotor position. These sensorless schemes use only terminal measurements and do not require additional hardware while maintaining reliable SRM operation over the entire speed and torque range with high resolution and accuracy.

In this thesis, a sensorless technique that has been developed to determine the SRM's rotor position is implemented using a field programmable gate array (FPGA) and the performance of the FPGA implementation is compared to a signal processor implementation.

1.2 Basic Structure and Principle of SRM Operation

A Switched Reluctance (SR) motor is a rotating electric machine where both the stator and rotor have salient poles, with windings only on the stator. Windings of diametrically opposite stator poles are connected in series or parallel to form one phase of the machine. The cross-section diagram of a 4-phase, 8/6 (# of stator poles/# of rotor poles) SRM is shown in figure 1.1.

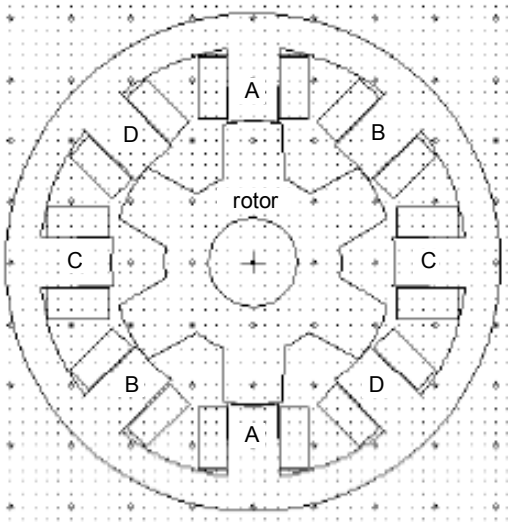

Figure 1.1 Cross-section diagram of an 8/6, four-phase SRM

The basic principle of a SRM operation is that when a stator phase is excited, the rotor of the SRM always rotates to the nearest position of minimum reluctance (aligned position), which corresponds to the minimum stored energy in the system.

When a stator pole pair is not aligned with a rotor pole pair, coils of the stator pole pair are excited by a sequence of current pulses applied to the phase and a magnetic flux path is created through the excited stator poles, air gap and the nearest rotor poles. Due to the tendency for the reluctance of the flux path to minimize, the rotor poles are attracted to the stator poles, producing torque. Then, when the stator pole pair becomes nearly aligned with the rotor pole pair, the excitation to the active coils is removed so that torque is not produced in the reverse direction; instead, coils of an adjacent stator pole pair are excited so that another rotor pole pair is attracted to the new stator pole pair since they are not aligned. By selectively exciting the stator pole pairs to attract rotor pole pairs,

synchronous continuous motion and continuous torque are produced. The current pulses need to be applied to the respective phase at the correct rotor position relative to the excited phase. Therefore, it is evident that the rotor position plays a critical role in determining which phase of the motor must be energized in order to produce the desired torque in the desired direction.

1.3 Contribution of the Thesis

The rotor position information in SRM drives is essential in determining the switching instants for proper control of speed, torque and torque pulsations. A shaft position transducer is usually employed to determine the rotor position. In inexpensive systems the rotor position sensor is comprised of a magnetized ring with Hall Effect sensors, or opto-interrupters with a slotted disk that produce discrete signals with no information between the pulses. In more expensive systems, a large number of pulses per revolution can be obtained from a resolver or optical encoder. Alternatively, a large number of pulses can be obtained by phase locking a high frequency oscillation to the pulses of discrete position sensors. Systems with such high resolution can work well down to zero speed. However, these sensors add complexity and cost to the system. Moreover, electromagnetic interference and temperature effects tend to reduce the reliability of the system. In order to avoid these difficulties some form of indirect position-sensing scheme is desirable. Several indirect position-sensing methods have been patented and published for sensorless control of SRM drives. The various indirect position-sensing techniques presented in the literature have their own advantages and disadvantages. Furthermore, the developed methods are application specific, depending on factors like, motor characteristics, converter topology, control strategy etc. The design considerations directly affect the type of indirect position scheme to be adopted for the drive.

The expected benefits of the indirect methods are: elimination of the electrical sensor connections, reduced SRM size, no maintenance, insusceptible to environmental factors and increased reliability. In addition, the expected features of desirable indirect methods

include: operating at zero speed and higher speed the same as conventional direct position sensors.

All of these indirect sensing methods use the instantaneous phase inductance variation information in some way to detect the rotor position at low speeds. This is possible with SRMs since the flux-angle-current characteristics vary significantly between the aligned and unaligned positions of the doubly salient stator and rotor poles.

In this thesis, a new method which estimates the rotor position by comparing the measured and estimated conductance values and calculating an error, which is input to a state estimator, is implemented using a FPGA. The estimated rotor position angle is used to control the electronic commutator, which controls when current is allowed in the machine's phases. The commutator is included in the FPGA with the state estimator and error calculation hardware. The rotor position estimator model is created and simulated in the Matlab/Simulink environment. The design is coded using the Verilog hardware design language and synthesized using Xilinx tools and simulated using Modelsim Simulator. The Simulated results obtained from both Simulink and Modelsim are compared. The design is then implemented on an FPGA chip and tested.

The block diagram of the entire motor drive system is as shown in figure 1.2.

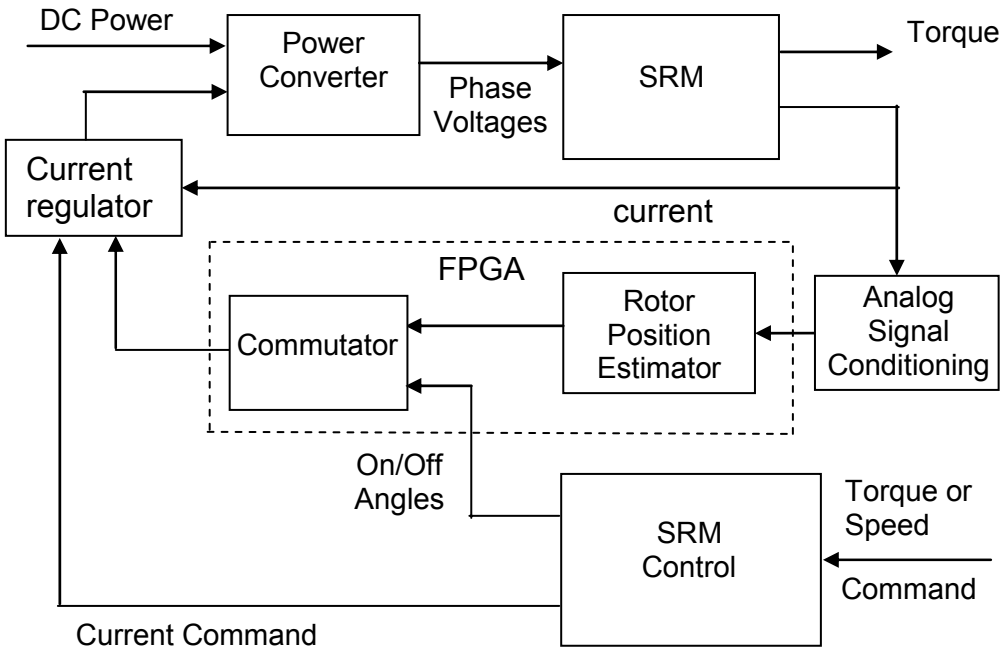

Figure 1.2: The Block diagram of the entire SRM based motor system

The Power Converter typically uses Silicon MOSFETs to control the voltage to the motor. The turning on and turning off of the MOSFETs is controlled by a current regulator circuit that forces the SRMs phase currents to be equal to the current command. The current command to the current regulator is the desired current in an SRM phase. The current command depends on the desired average torque. The rotor position estimator gives the rotor position which is given as input to the commutator, which controls the current pulses, that is determines which phase has to be excited to get the desired torque.

This thesis deals with the design of the blocks titled Rotor Position Estimator and Commutator in the FPGA block, in figure 1.2. The inputs and outputs of the FPGA are summarized in figure 1.3. The inputs to the estimator are used to compute the rotor position. The instantaneous position information is used as an input to the commutator to derive the instant of switching of the currents.

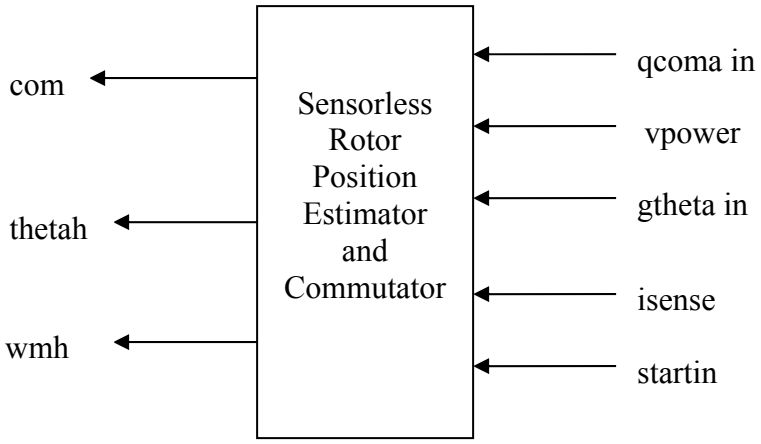

Figure 1.3 Block Diagram of Rotor Position Estimator and Commutator

1.4 Outline of the Thesis

This thesis is organized as follows:

Chapter 1 gives a general introduction of switched reluctance motors: its basic structure and principle of operation. Then the motivation of this thesis, which is the implementation of the rotor position estimator on a Field Programmable Gate Array (FPGA) is given.

Chapter 2 discusses the basic SRM model and the background issues related to the rotor position estimator and SRM control.

Chapter 3 gives the basic SRM drive system model created in Simulink and Matlab and important simulation results for the implementation of the rotor position estimator.

Chapter 4 introduces to the design of the rotor position estimator using the Verilog Hardware design language and its synthesis and simulation results using Xilinx tools and Modelsim Simulator. It also introduces to the Field Programmable Gate Array (FPGA).

Chapter 5 gives the comparison of the simulation results obtained using both Simulink and Modelsim. It also gives the implementation of the rotor position estimator on the FPGA. Experimental results from a programmed FPGA are presented to verify the correct operation of the rotor position estimator. The performance of the FPGA based SRM rotor position estimator in terms of calculation time is compared to a signal processor implementation of the same position estimator algorithm.

CHAPTER 2
DESIGN ANALYSIS OF ROTOR POSITION ESTIMATOR

2.1 Variation of inductance with rotor position

An efficient operation of the Switched Reluctance Motor (SRM) can be achieved only by proper determination of the rotor position. Rotor position measurement or estimation is an integral part of SRM control because of the nature of reluctance torque production. The excitation of the SRM phases needs to be properly synchronized with the rotor position for effective control of speed, torque and torque pulsation.

All SRM's possess a unique relationship between phase inductance, phase current, and rotor position, which makes prediction of rotor position possible. Since the rate of change of phase current is dictated by the incremental inductance of the phase circuit, and the incremental inductance is in turn a function of rotor position and phase current, rotor position can be deduced from knowledge of phase current and its rate of change.

At low speeds, to estimate the rotor position of the SRM, the variation of the phase inductance with rotor position can be used. But because the SRM operates with substantial iron saturation at torque producing currents, the phase inductance is a function of both rotor position and phase current.

The SRM motion is produced because of the variable reluctance in the air gap between the rotor and the stator. When a stator winding is energized, producing a single magnetic field, reluctance torque is produced by the tendency of the rotor to move to its minimum reluctance position. When a rotor pole is aligned with a stator pole, as shown in figure 2.1, there is no torque because field lines are orthogonal to the surfaces (considering a small gap). In this aligned position, the unsaturated inductance is a maximum since the reluctance is minimum. If one displaces the rotor from the aligned position, there will be torque production that will tend to rotate the rotor back to the aligned position.

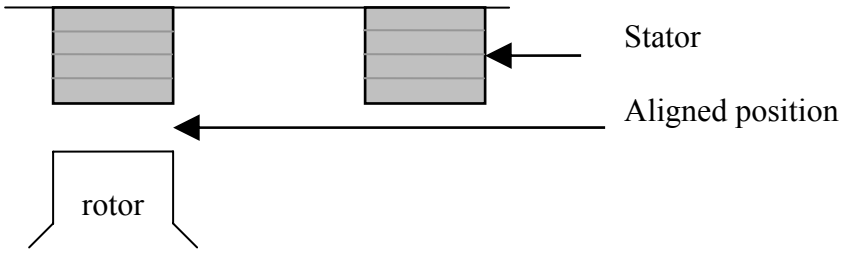

Figure 2.1 Aligned Position

If current is injected in the phase when the rotor is rotated to the unaligned position, as shown in figure 2.2, there will not be torque production (or very little). If one displaces the rotor from the unaligned position, then a torque is produced that tends to displace the rotor to the nearest aligned position.

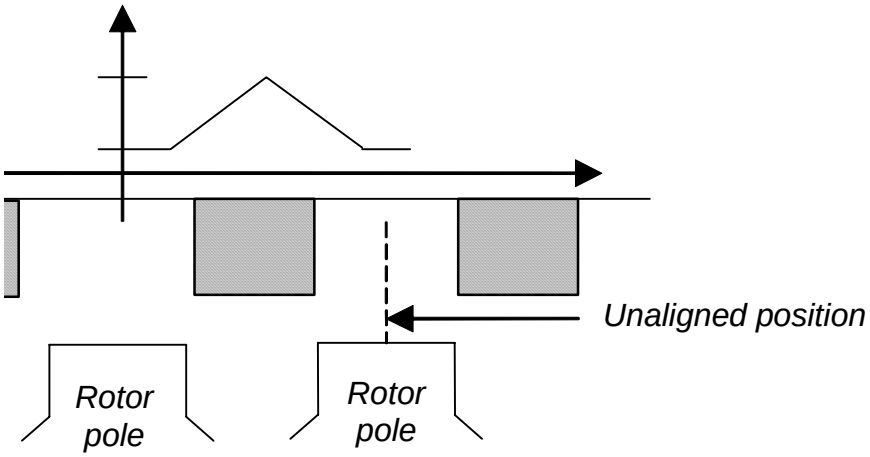

Figure 2.2 Unaligned Position

When rotor poles are aligned with the stator poles of a relevant phase the flux linkage, for a given phase current, is maximized. The flux linkage is a maximum when the unsaturated inductance is a maximum. However at this aligned position the relationship between flux linkage and phase current is extremely nonlinear in a well-designed machine, because the poles are magnetically saturated at the rated phase current.

The flux linked by a phase versus phase current for different rotor positions is plotted in figure 2.3. The slope of the flux curves at a fixed rotor position is the incremental phase inductance.

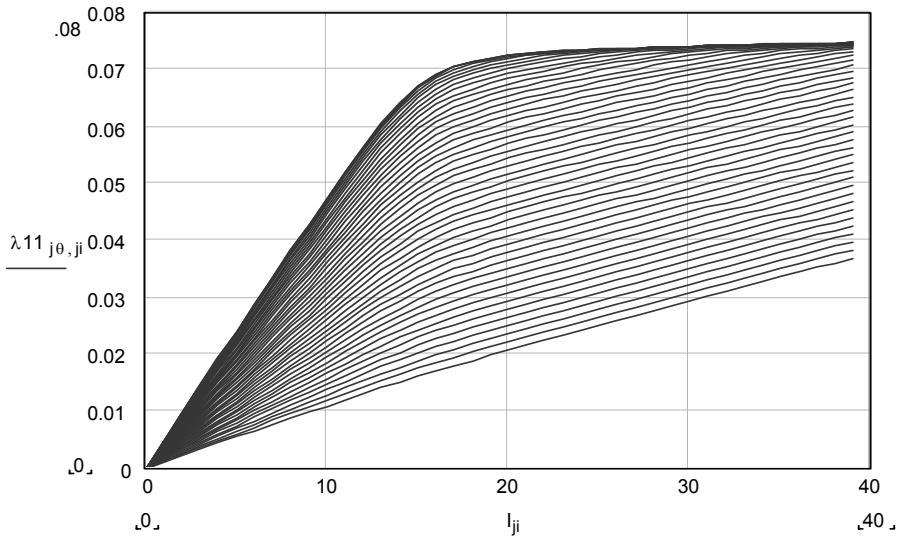

Figure 2.3 Plot of the SRM's phase flux versus current for different rotor positions showing the effect of iron saturation.

Because this slope changes with current so does the phase inductance as shown in figure 2.4 where the phase inductance is plotted versus rotor position for different currents.

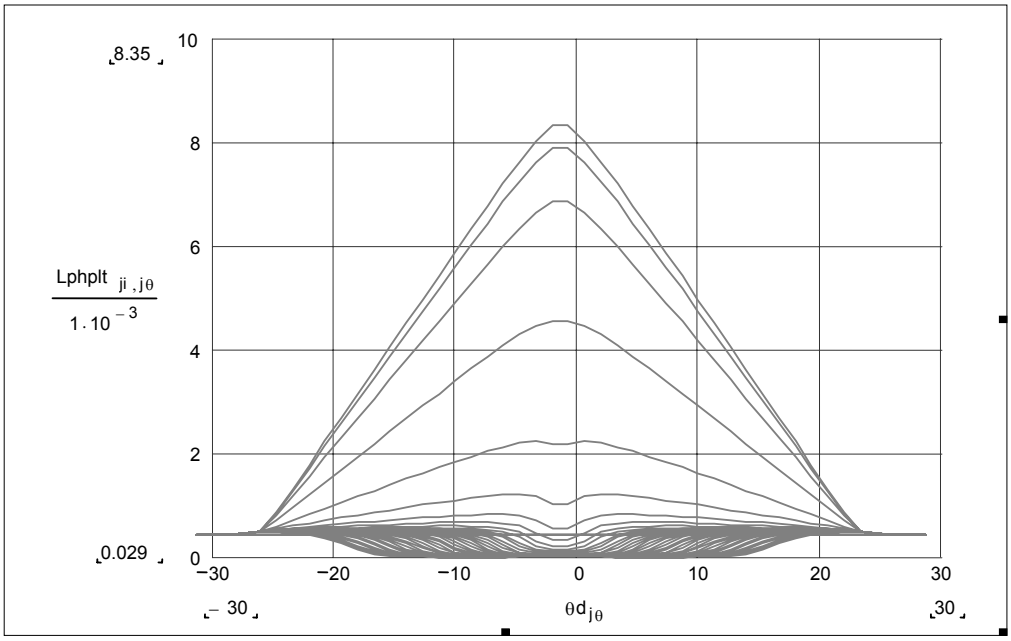

Figure 2.4 Plot of the SRM's phase inductance versus rotor position for different currents.

At low currents, the phase inductance has an essentially triangular shape versus rotor position while at high currents the phase inductance is far from triangular. This makes using the current in the torque-producing phase for rotor position estimation very complex. Thus, the normal practice is to stimulate the non-torque-producing phases that normally have zero current in them for some instantaneous rotor positions, with small sensing currents, that do not saturate these phases and produce little torque, to do the position estimation. Figure 2.5 shows the aligned incremental phase inductance versus current for an experimental SRM. For iron saturation to be ignored in this machine the sensing phase current must be less than about 2.0A. It has its maximum inductance when it is in an aligned position and minimum inductance when unaligned. When the voltage is applied to the stator phase the current in that phase increases and the SRM creates torque in the direction of increasing inductance.

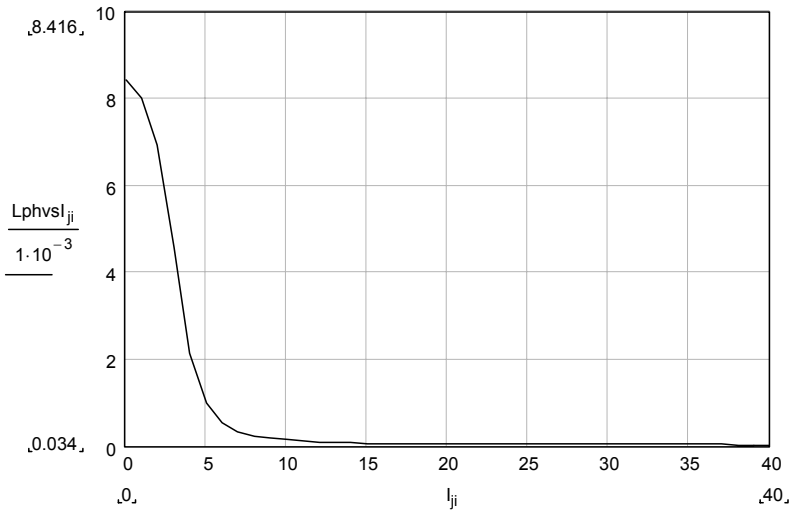

Figure 2.5 Plot of the phase inductance at the aligned position versus current for an SRM.

2.2 Theory behind the Rotor Position Estimator used in this thesis

When voltage is applied across the stator winding of the SRM as shown in the figure 2.6,

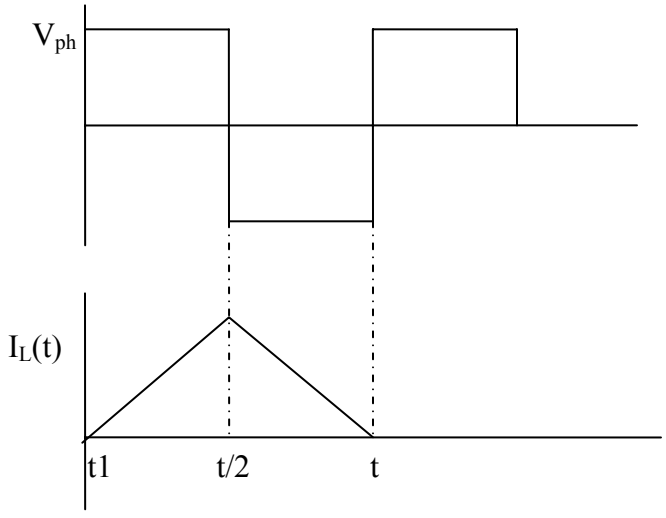

Figure 2.6 Current Profile across an inductor

the current in the inductor obeys the following equations

$$Vph(t) = Lph\big(\theta(t)\big)\frac{diph}{dt}, \qquad iph(t) = iph(t1) + \int_{t_1}^{t} \frac{Vph(\tau)}{Lph\big(\theta(\tau)\big)}d\tau \tag{2.1}$$

$V_{ph}(t)$ is a constant from t_1 to t with a value equal to the DC voltage, V_{DC}. It will be assumed that $i_{ph}(t_1) = 0$ since t1 coincides with a zero phase current. It will also be assumed that the modulation frequency F_{mod} is high enough that the inductance does not vary over the time period of one modulation cycle. In this case

$$iph(t) = \frac{1}{Lph\big(\theta(t1)\big)}\int_{t1}^{t} Vph(\tau)d\tau \equiv g\big(\theta(t1)\big)\int_{t1}^{t} Vph(\tau)d\tau \approx g\big(\theta(t1)\big)\cdot VDC\cdot t \tag{2.2}$$

where g(θ) has been defined as the inverse of the phase inductance. Equation 2 gives the current during the increasing current part of the waveform in figure 2.6 and is a positive ramp as expected. The peak phase current (at $t = t_1 + D_{mod}T_{mod}$) is modulated by the inverse phase inductance function $g_{ph}(\theta)$ which does not depend on the details of the phase current or voltage during one modulation cycle. Similarly the phase current ramps down during the decreasing current part of the waveform generating a triangle of current whose peak depends on $g_{ph}(\theta)$. Taking the average of the phase current over one modulation cycle gives

$$\big\langle iph\big(\theta(t_1)\big)\big\rangle_{Tmod} = \frac{1}{T_{mod}}\int_{t_1}^{t1+Tmod} g_{ph}\big(\theta(t_1)\big)\int_{t1}^{t} V_{ph}(\tau)d\tau dt = g_{ph}\big(\theta(t_1)\big)\frac{1}{T_{mod}}\int_{t_1}^{t1+Tmod}\int_{t1}^{t} V_{ph}(\tau)d\tau dt$$

$$\big\langle iph\big(\theta(t_1)\big)\big\rangle_{Tmod} \equiv i_{sense}(\theta) = g_{ph}(\theta)\frac{V_{DC}T_{mod}}{4} \tag{2.3}$$

Thus the average of the phase current over one modulation cycle is proportional to the inverse phase inductance function $g_{ph}(\theta)$.

The $g_{ph}(\theta)$ functions (one for each phase) can be used to estimate the SRM's rotor position since they are known ahead of time from the machine's characteristics and they can be measured by exciting the phases of the SRM as described above. For simplicity in

what follows, it will be assumed that the known machine inductance profile has a constant value equal to the unaligned inductance for rotor positions where the rotor and stator poles do not overlap and it varies linearly from the unaligned inductance value to the aligned inductance value when the poles overlap as shown in figure 2.7.

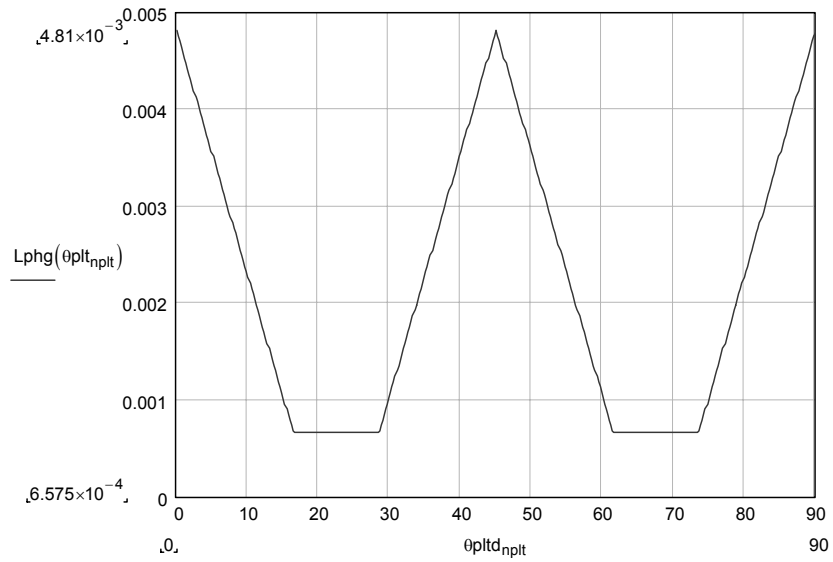

Figure 2.7 Ideal phase inductance profile versus rotor position.

An error between the estimated and measured angle cannot be computed directly since the rotor position is not measured. However a suitable error can be defined using the $g_{ph}(\theta)$ functions which can be measured using the average in equation 2.3. Let θ be the actual rotor position and α be the estimated rotor position, then the error for data from phases one and two is

$$error_1(\theta,\alpha) = g_2(\theta)g_1(\alpha) - g_1(\theta)g_2(\alpha) \tag{2.4}$$

Note that when θ = α, the error is zero. The total error is just the sum of the errors for each phase pair. The error for a four phase SRM is

$$error_{tot}(\theta,\alpha) = \left(g_2(\theta)g_1(\alpha) - g_1(\theta)g_2(\alpha)\right) + \left(g_3(\theta)g_2(\alpha) - g_2(\theta)g_3(\alpha)\right) + \left(g_4(\theta)g_3(\alpha) - g_3(\theta)g_4(\alpha)\right) + \left(g_1(\theta)g_4(\alpha) - g_4(\theta)g_1(\alpha)\right) \qquad (2.5)$$

The total error is also zero when θ = α. The total error depends on both θ and α and thus on both θ and α – θ. It can be verified that this error has one sign for α – θ positive and the opposite sign when α – θ is negative for any θ. This is shown in figure 2.8 for an experimental 4 phase SRM for α – θ = +/-1º, +/-5º, and +/-10º and rotor positions from 0º to 90º.

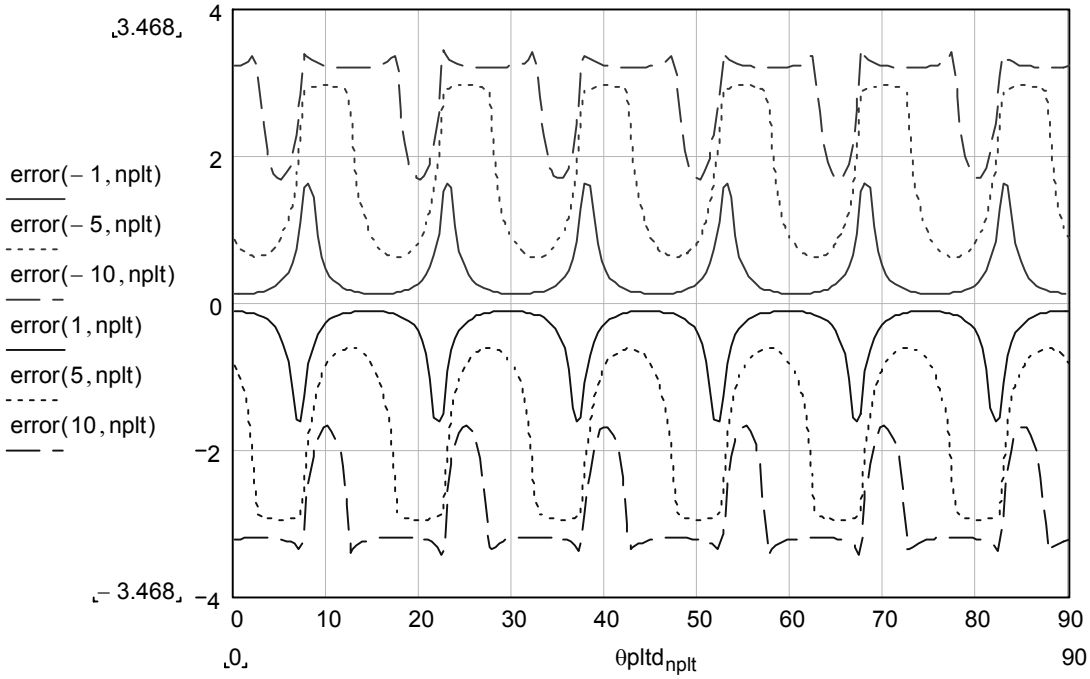

Figure 2.8 The computed error for an experimental 4 phase SRM for α – θ = +/-1º, +/-5º, and +/-10º and rotor positions from 0º to 90º.

Thus, this error can be used in a state estimator to estimate the rotor position. The gain of the estimator depends on rotor position but because the sign of the error does not vary with rotor position, it will be possible to design a stable state estimator.

2.3 Rotor Position State Estimator Equations

Consider the physical system of a motor. The state equations for the physical system are of the form

$$X_p = A\, x_p + B\, u_p \tag{2.6}$$

$$Y_p = C\, x_p \tag{2.7}$$

Where the subscript p indicates the variables are for the physical system. The motor's acceleration is given by

$$\frac{J dwm}{dt} = Te - Tload \tag{2.8}$$

where, J= Moment of Inertia

$\frac{dwm}{dt}$ = acceleration

T_e= electrical Torque

T_{load} = mechanical Torque

When the motor runs at steady speed, the acceleration is zero because the load and electrical torques balance. Therefore

$$\frac{dwm}{dt} = 0 \tag{2.9}$$

The rate of change of rotor position is the rotor speed, w_m given by

$$\frac{d\theta}{dt} = \omega_m \tag{2.10}$$

where θ is the angular position of the rotor of the motor.

The equation (2.6) can be deduced from equations (2.9) and (2.10)

$$\frac{d}{dt}\begin{bmatrix} \theta \\ \omega_m \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \theta \\ \omega_m \end{bmatrix} \tag{2.11}$$

Since time rate of change of the rotor position is nothing but ω_m (the rate of change of position is speed), the first row of the matrix is [0 1] and since the acceleration is zero, the rotor speed ω_m is zero, hence the second row of matrix is [0 0].

The state equations for the state estimator in the linear case are

$$X_e = A\, x_e + B\, u_e + H\,(Y_e - Y_p) \qquad (2.11)$$

$$Y_e = C\, x_e \qquad (2.12)$$

Since the model system used to estimate the unmeasurable state is not exactly the same as the real physical system, an error between what is measured and the estimator's prediction of what is measured (Y) is input to the model estimator equations. In equation 2.11 this error input is given by $H\,(Y_e - Y_p)$, where H is a constant gain matrix to be determined by the designer such that the error between the estimated state and the actual state values decays to zero (thus $H\,(Y_e - Y_p)$ decays to zero) and thus the estimated state is the correct value. The gain matrix H is adjusted to see how fast and with what dynamics the state error decays to zero

The error is the difference between the actual and the estimated values and can be computed by

$$\begin{aligned} d\,\text{Error} / dt &= d\,(X_p - X_e) / dt \\ &= A\,X_p + B\,U_p - A\,X_e + B\,U_e + H\,(Y_e - Y_p) \\ &= A\,(X_p - X_e) - H(C\,X_e - C\,X_p) \\ &= (A+HC)\,(X_p - X_e) \\ &= (A+HC)\,\varepsilon \end{aligned} \qquad (2.13)$$

$$\varepsilon = \varepsilon_0\, e^{(A+HC)\,t}$$

For the error to decay to zero the real parts of the eigen values of the matrix (A+HC) must be negative. The matrix H is chosen so that the error decays with the desired dynamics.

For the SRM rotor position estimator the state vector and state vector error vectors are

$$X = \begin{bmatrix} \theta \\ \omega_m \end{bmatrix} \tag{2.14}$$

$$\varepsilon = \begin{bmatrix} \theta - \alpha \\ \omega_m - \omega_\alpha \end{bmatrix} \tag{2.15}$$

where α is the estimated rotor position angle and ω_α is the estimated speed of the rotor. Differentiating equation (2.14),

$$\frac{dX}{dt} = \frac{d}{dt}\begin{bmatrix} \theta \\ \omega_m \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \theta \\ \omega_m \end{bmatrix} + \begin{bmatrix} 0 \\ 1 \end{bmatrix}(Te - T_{load}) \quad \text{--- (from equation (2.8))}$$

But because of equation (2.9), (T_e- T_{load}) is equal to zero.

The equation for the state estimator is defined as

$$\frac{d}{dt}\begin{bmatrix} \alpha \\ \omega_\alpha \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ \omega_\alpha \end{bmatrix} + \begin{bmatrix} H_1 \\ H_2 \end{bmatrix}\Big(\big(g_2(\theta)g_1(\alpha) - g_1(\theta)g_2(\alpha)\big) + \big(g_3(\theta)g_2(\alpha) - g_2(\theta)g_3(\alpha)\big) + \big(g_4(\theta)g_3(\alpha) - g_3(\theta)g_4(\alpha)\big) + \big(g_1(\theta)g_4(\alpha) - g_4(\theta)g_1(\alpha)\big)\Big)$$

$$= \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix}\begin{bmatrix} \alpha \\ \omega_\alpha \end{bmatrix} + \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} error_{tot}(\theta, \alpha) \tag{2.16}$$

$$\frac{d\alpha}{dt} = \omega_\alpha + H_1 error_{tot}(\theta, \alpha) \tag{2.17}$$

where $error_{tot}(\theta,\alpha)$ was defined previously in equation 2.5. To compute the stability of the state error e the nonlinear error function must be linearized.

$$error_{tot}(\theta,\alpha) = error_{tot}(\theta,\theta-\alpha) = error_{tot}(\theta,\varepsilon_\theta) \approx \left(\frac{\partial}{\partial \theta e} \left(error_{tot}(\theta,\varepsilon_\theta) \right)_{\theta,0} \right) \cdot \varepsilon_\theta \qquad (2.18)$$

The partial derivative in equation 2.18 can be estimated from the results in figure 2.8. With the linearization of the error function the error dynamics are governed by

$$\frac{d}{dt}\begin{bmatrix} \varepsilon_\theta \\ \varepsilon_\omega \end{bmatrix} = \begin{bmatrix} H_1 \frac{\partial}{\partial \varepsilon_\theta} \left(error_{tot}(\theta,\varepsilon_\theta \right)_{\theta,0} & 1 \\ H_1 \frac{\partial}{\partial \varepsilon_\theta} \left(error_{tot}(\theta,\varepsilon_\theta \right)_{\theta,0} & 0 \end{bmatrix} \begin{bmatrix} \varepsilon_\theta \\ \varepsilon_\omega \end{bmatrix} \qquad (2.19)$$

The equations (2.16) and (2.17) can be represented by the block diagram shown in figure 2.9.

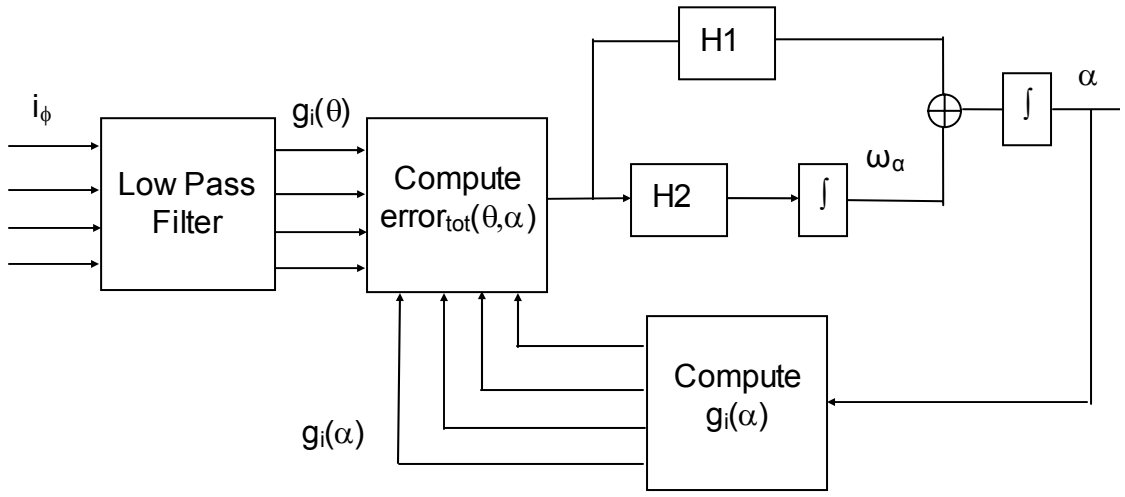

Figure2.9. Block diagram representation of the state estimator

This system is used to estimate the values of rotor position and rotor speed. In this thesis, this system is coded using the Verilog hardware design language and then is implemented

on an FPGA chip. The programmed FPGA chip would be suitable for use as part of the SRM motor control.

The block diagram of the control system of the Switched Reluctance Motor is as shown in figure 2.10.

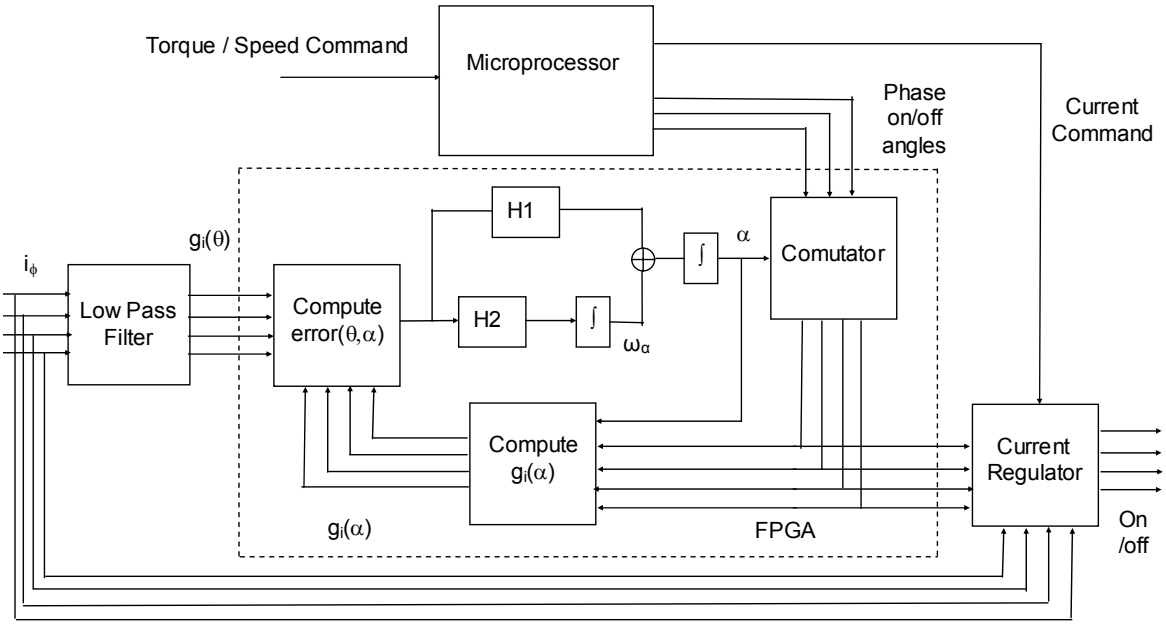

Figure 2.10 Block Diagram of the SRM control system

The block diagram of the block FPGA, which is actually the basis of this thesis, is as shown in the figure 2.11.

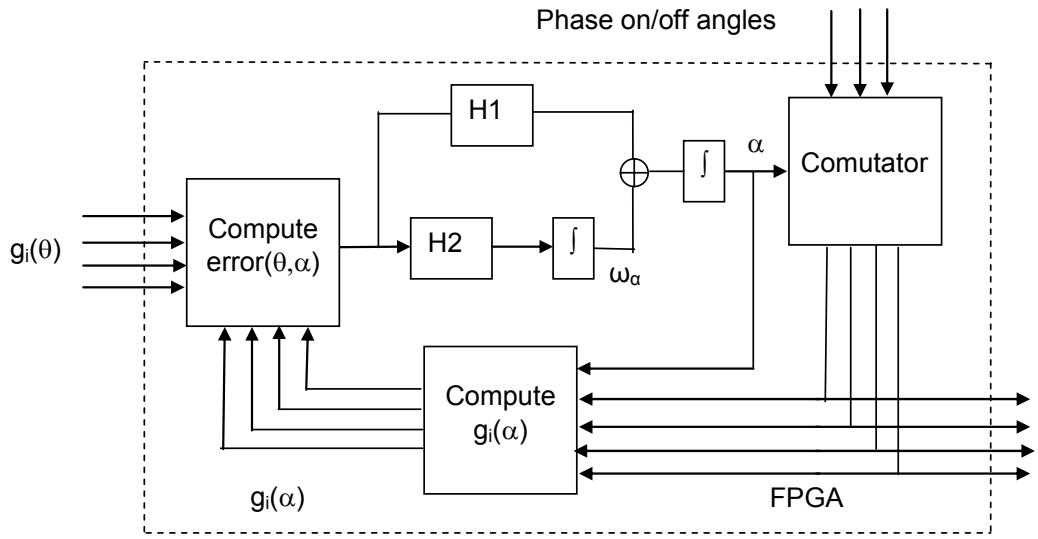

Figure 2.11 Block diagram of the FPGA that is the subject of this thesis

The estimated value of rotor position, α is given as input to the commutator circuit. The basic block diagram of the commutator is as shown in the figure 2.12.

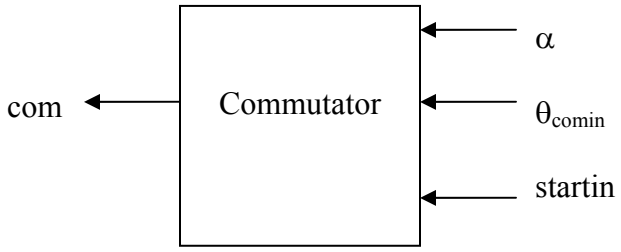

Figure2.12 Block diagram of Commutator

The commutator gets the estimated rotor position angle as input, which it compares with the desired angle range and determines which of the four phases is to be excited and then accordingly turns on the phase that is to be excited. Thus, the current is given as input to that phase, which will in turn produce the desired torque. The commutator compares the instantaneous estimated rotor position α with the values (θ_{sm}, θ_{lrg}) which are given as input through 'θ_{comin}' and commutates the current in the corresponding phase. The

commutator insures that the currents in the stator circuits are switched on and off in accordance with the rotor position. The torque can be controlled to give a resultant which is positive (i.e. motor action) or is negative (i.e. generator action) simply by switching the current in the coil on and off at appropriate instants.

A model of the circuitry in the FPGA and which is the subject of this thesis has been created in Simulink prior to this thesis research and was available to facilitate this research. This model will be discussed in detail in Chapter 3.

CHAPTER 3
SIMULINK MODEL OF THE CIRCUIT TO BE DESIGNED

3.1 Introduction to Simulink model

The simulation of a system is important in view of its design and experimental realization. Simulation using Matlab/Simulink allows a high flexible modeling environment to model power electronic systems containing electrical machinery, electronic controls, and power circuits. This thesis implements part of the control design for a power electronics controller for a SRM. During this thesis a complete Matlab/Simulink model of this power electronics system was available. It included a model of the part of the control being implemented here. All Simulink simulations are documented by their block diagrams, their corresponding special Matlab functions and their input parameters. A strong aspect of the SRM simulation using Simulink is the use of conventional blocks allowing easier understanding of the program's structure.

Figure 3.1 shows the block diagram of the Rotor Position Estimator and Commutator with all the inputs and outputs.

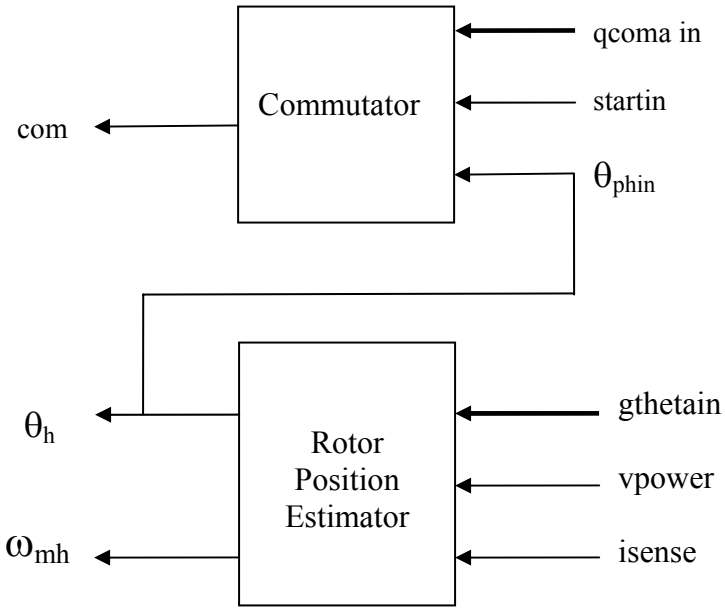

Figure 3.1 Block Diagram of Rotor Position Estimator and Commutator

Consider the block diagram of the Sensorless Rotor position estimator. The input gthetain represents the inverse inductance value of the actual rotor position g(θ). The input vpower is the voltage applied to the motor. The input isense is the select signal given as input to the estimator to choose between g(θ) and g(α). Recall that the error is defined as

$$error_{tot}(\theta,\alpha) = \left(g_2(\theta)g_1(\alpha) - g_1(\theta)g_2(\alpha)\right) + \left(g_3(\theta)g_2(\alpha) - g_2(\theta)g_3(\alpha)\right) + \left(g_4(\theta)g_3(\alpha) - g_3(\theta)g_4(\alpha)\right) + \left(g_1(\theta)g_4(\alpha) - g_4(\theta)g_1(\alpha)\right)$$

If the measured g(θ) from a particular phase is not available because the rotor is in a position where that phase is producing torque (as seen in figure 3.2), the calculated g(α) for that phase is used instead. The signal isense is high when the rotor is in a position where the given phase is not producing torque and thus is being energized with sense pulses to produce g(θ)and is low when the rotor is in a position where the given phase is producing torque as shown in figure 3.3. When the select signal isense is high, g(θ) is selected and thus used in the error calculation used to estimate the rotor position. When isense is low, g(α) for the given phase is selected because the g(θ) for that phase is not available, and thus error is computed without the g(θ) information from the given phase. The error is computed using only the g(θ)s from the other phases.

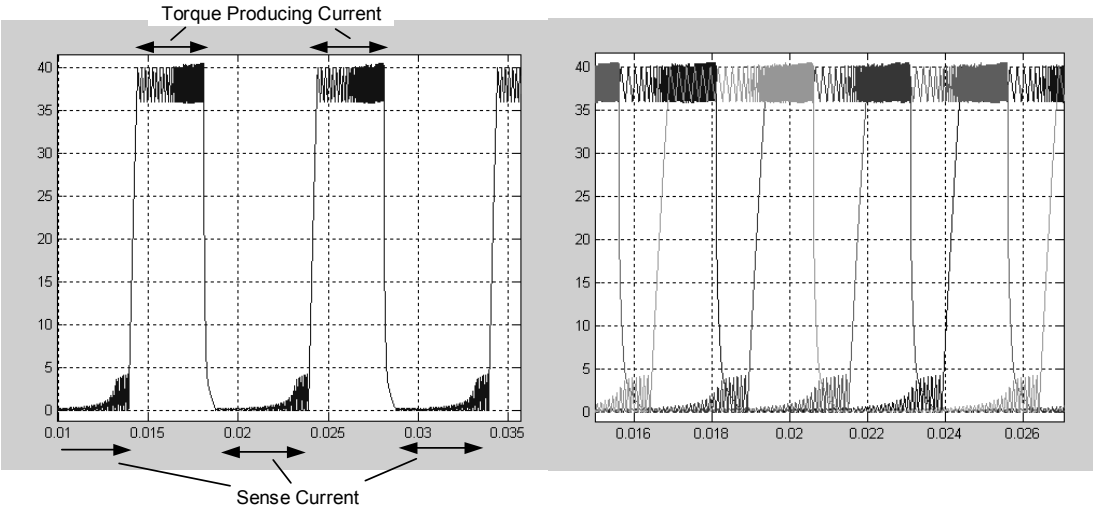

Fig. 3.2 Phase current showing the torque producing and sense phase currents

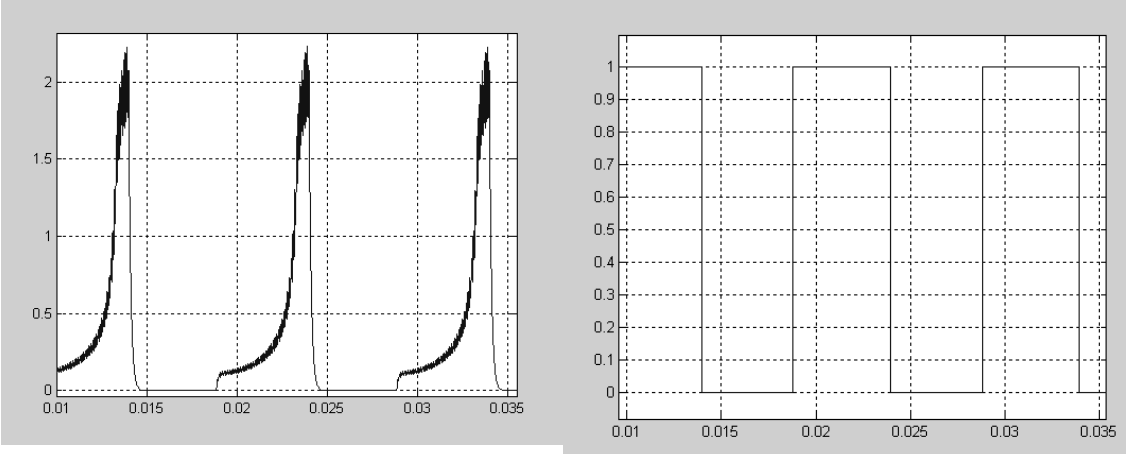

Figure 3.3 Plot of measured (from simulation) $g_1(\theta)$ and the select signal isense for phase one.

The output ω_{mh} represents the speed of the motor. The output θ_h represents the estimated rotor position, α. This rotor position angle, α is the input to the commutator as θ_{phin}. The rotor position angle is shifted to produce 3 additional angles. These shifted angles are input to the same commutator hardware to produce the commutator output for the other 3 phases.

3.2 Commutator Block

Now, consider the commutator block. The input startin is the select signal given to the commutator, which tells the commutator whether the user wants the SRM to operate normally and produce torque or to go into startup mode. If the startin select signal is high, the control and position estimator go into start up mode. In this mode the control continuously applies sense pulses to all four phases of the SRM so that the SRM's rotor position can be estimated without a net torque being produced and without the rotor rotating. When the user wants the SRM to produce torque, the input startin is made low taking the control and position estimator out of the startup mode. Now the control commutates the SRM so that it produces the commanded torque.

The input qcomin is a concatenation of the angles θ_{sm}, θ_{lrg} and a one bit signal $\theta_{lrg}>\theta_{sm}$, given as a single digital word.

The angle θ_{sm} is the smaller of θ_{sm} and θ_{lrg}. If the motor is rotating in the forward direction θ_{sm} is the turn-on angle and θ_{lrg} is the turn off angle and the reverse is true if the motor is rotating in the reverse direction. These angles are the rotor's position defined relative to a given stator. This rotor position is near the rotor's unaligned position with respect to the stator pole.

The commutator checks if the estimated rotor position angle α falls in between θ_{sm} and θ_{lrg}. If the estimated rotor position is between these two angles the commutator outputs a one, otherwise it outputs a zero as shown in figure 3.4. This is done for all the four phases of the stator and accordingly each phase of the stator is commutated enabling the SRM to produce torque.

The signal $\theta_{lrg}>\theta_{sm}$ is high when θ_{lrg} is greater than θ_{sm}, i.e. the normal case and low when θ_{lrg} is smaller than θ_{sm}, i.e. when the angles are outside the range of –30º to +30º and they must be wrapped to stay within this range as illustrated in figure 3.5 for forward rotor rotation.

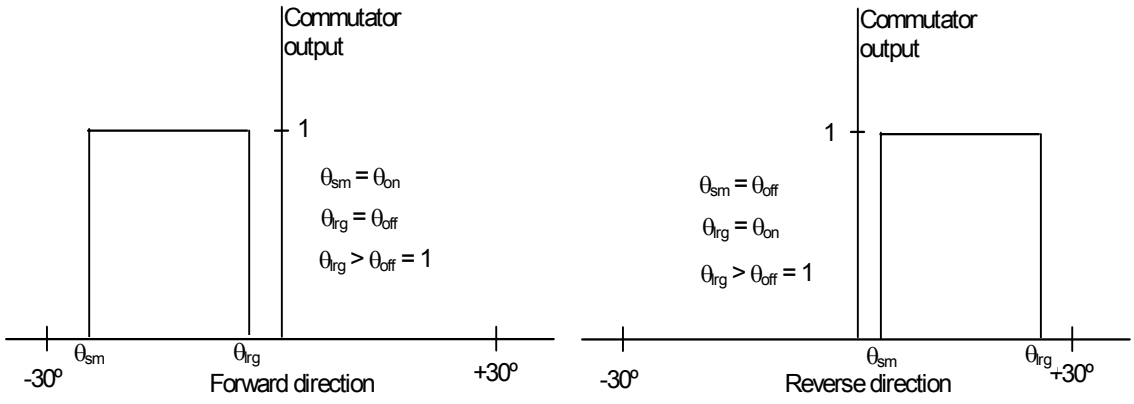

Figure 3.4 Commutator output for forward and reverse directions under normal conditions.

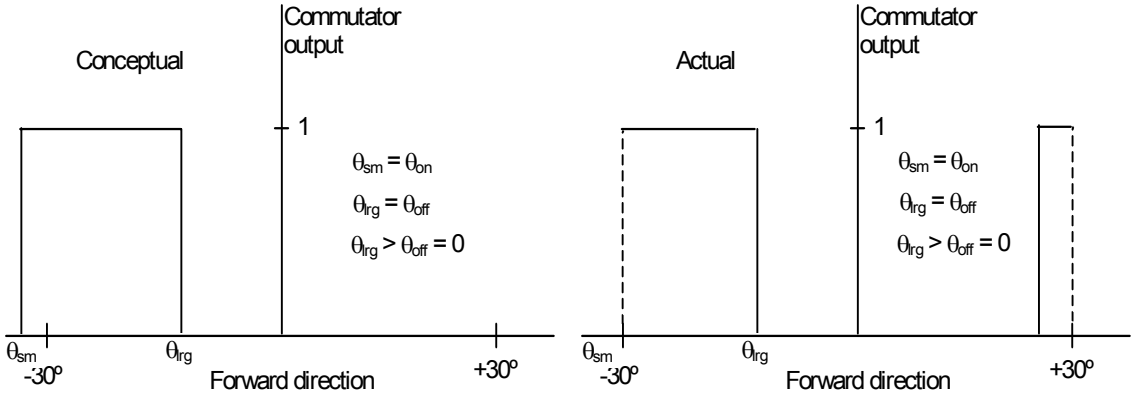

Figure 3.5 Commutator output for forward rotation when the turn on angle is advanced beyond the allowed range.

The output com is the four-bit output from the commutator corresponding to the four phases, each bit being high or low according to whether that phase is to produce torque or not.

The commutation for all possible combinations of α with respect to θ_{sm}, θ_{lrg} and $\theta_{lrg}>\theta_{sm}$ is discussed below.

$\theta_{ph}> \theta_{sm}$	$\theta_{ph}<\theta_{lrg}$	$\theta_{lrg}>\theta_{sm}$	com
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 3.1 Truth Table for the Commutator

As seen from the table 1.1, the commutator turns on for only three cases.

1) When the estimated rotor position θ_{ph} falls in between θ_{sm} and θ_{lrg}, i.e., θ_{ph} is greater than θ_{sm} and smaller than θ_{lrg} and θ_{lrg} is greater than θ_{sm}.

2) When qph is larger than both θ_{sm} and θ_{lrg} and θ_{lrg} is smaller than θ_{sm}.

3) When qph is smaller than both θ_{sm} and θ_{lrg} and θ_{lrg} is smaller than θ_{sm}.

This truth table is obtained from figures 3.4 and 3.5.

A Simulink model developed for the commutator block and available for this thesis showing all the four phases is as shown in figure 3.6. Note that in the Simulink model the letter q is used for θ and no subscripts are used. The figure shows the block named four wrapped angles in which the shifting of the rotor position angle by 45˚ takes place for each phase, and then wrapping also takes place such that the rotor position angle for each phase lies in between -30˚ and +30˚. The blocks named commutator1, commutator2 etc. represents the commutator for each phase.

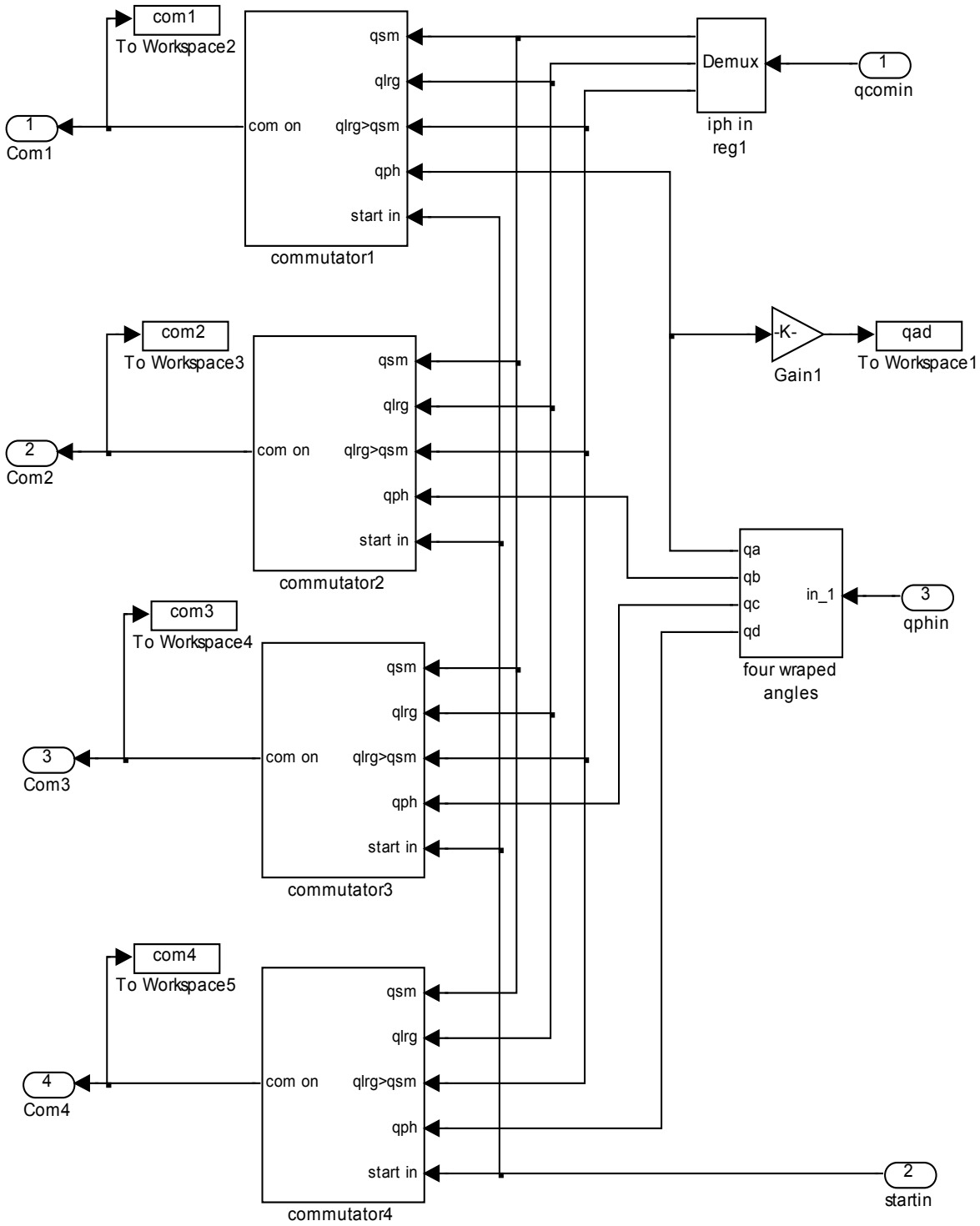

Figure 3.6 Simulink model of the commutator block.

Figure 3.7 shows the Simulink model of the block commutator1, which is a sub-block of the commutator model. As seen in the figure, the angle θ_{sm} (qsm), the angle θ_{lrg} (qlrg), the signal corresponding to $\theta_{lrg}>\theta_{sm}$ (qlrg>qsm), the startin signal and the estimated rotor position angle are the inputs. The model checks if the rotor angle is greater than θ_{sm} and also if it is less than θ_{lrg} and accordingly gives the 3-bit input to the look-up table which contains the corresponding truth-table shown in table 3.1. The model also checks the status of the startin signal. The output of the AND gate is the output of the commutator for that phase. If it is one, the phase is energized to produce torque and if it is zero, the phase is energized with sense pulses.

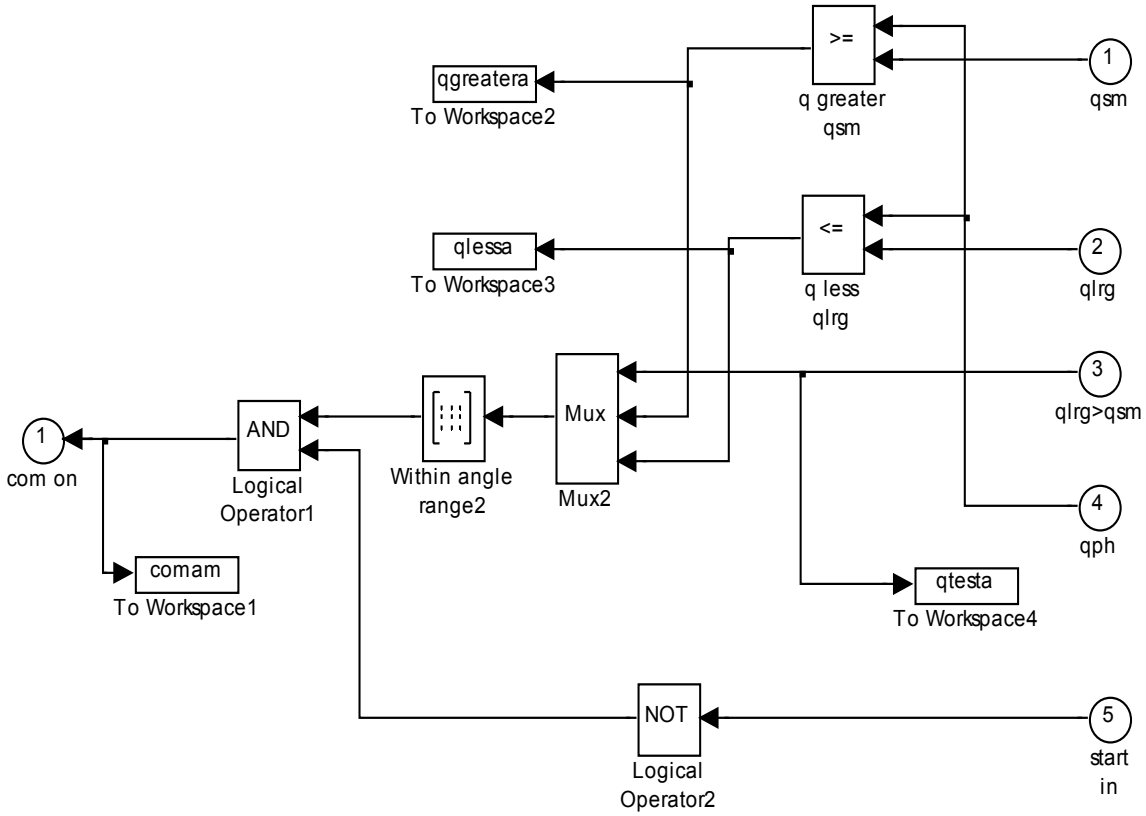

Figure 3.7 Simulink model of the commutator block for one phase

3.3 Rotor Position Estimator

Now consider the block diagram of the Sensorless rotor position estimator in figure 3.1 shown in figure 3.8. The signals g(θ) (gthetain), V_{DC} (vpower) and i_{sense} (isense) are inputs to the error computing block. Based on these inputs and also the shifted and wrapped estimated rotor position α (alpha), the error is calculated.

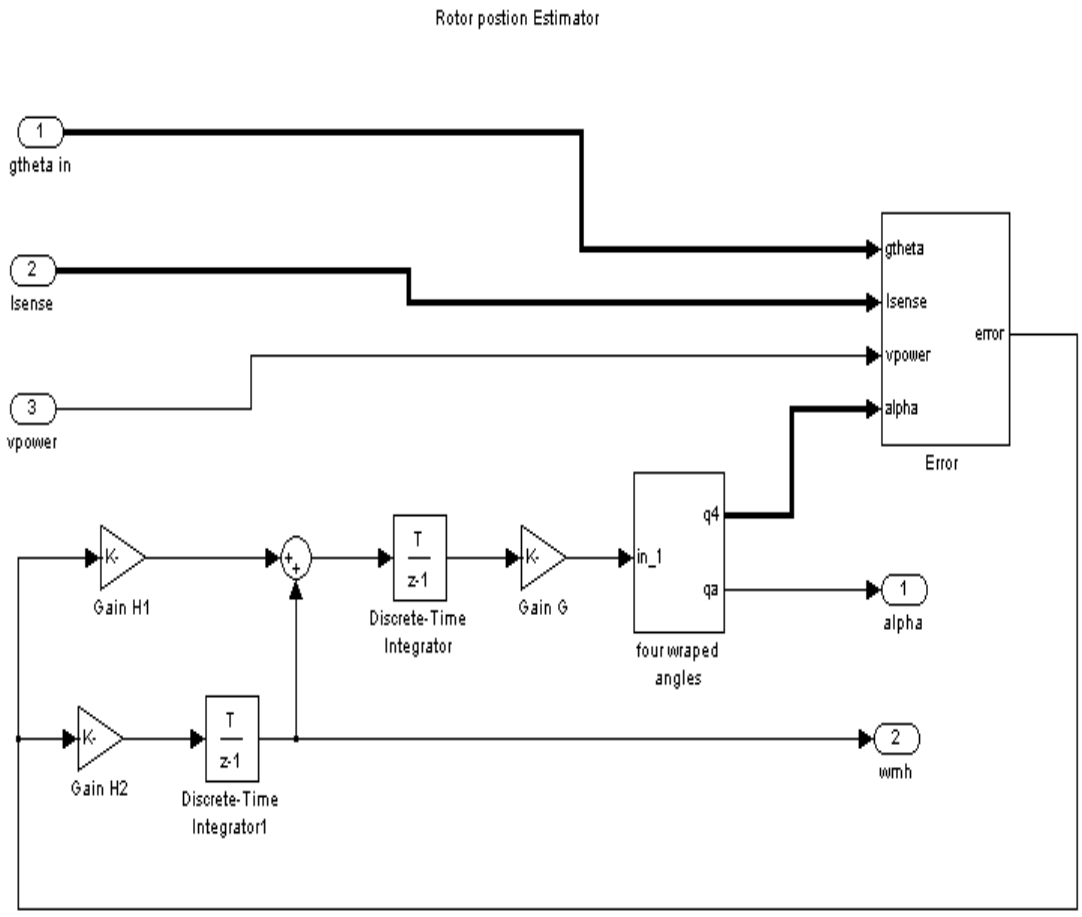

Figure 3.8 Simulink model of the rotor position estimator.

As discussed in chapter 2, the error is calculated using the measured inverse inductance values g(θ) and the computed inverse inductance values g(α) using equation 2.5. To compute the error, the values of g(α) for all four phases must be computed.

Figure 3.9 shows the Simulink model for calculating g(α) for each SRM phase.

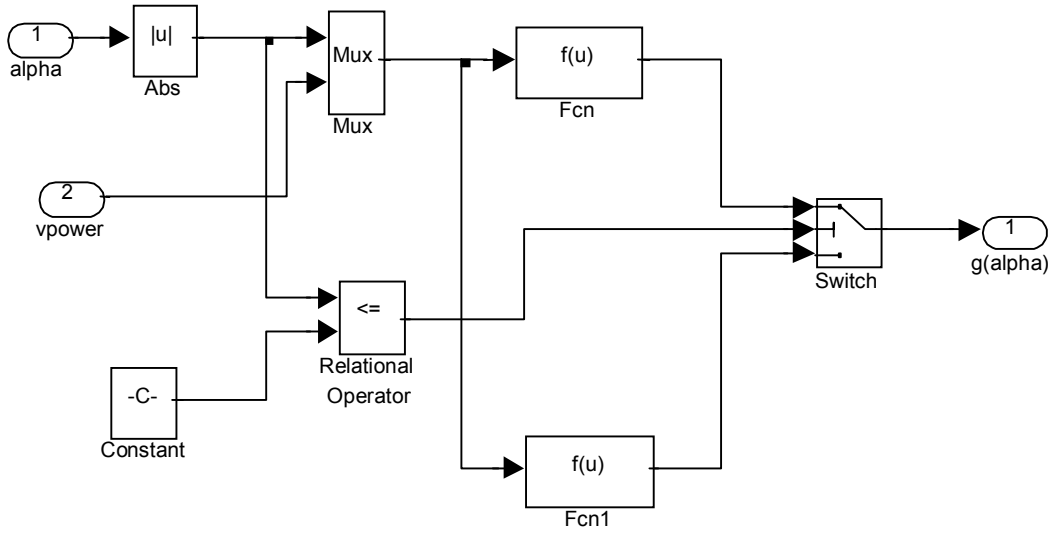

Figure 3.9 Simulink model of the block galpha to calculate g(α)

While calculating g(α), the absolute value of α is used since the function is an even function of α. The inverse inductance function g(α) is a function of α and should be equal to the measured g(θ) function obtained by demodulating (low pass filtering) the sense pulse currents. The sense pulse currents are triangular in shape with a peak value determined by the SRM's phase inductance. Thus g(α) is calculated by using the following equation.

$$g(\alpha) = \frac{\text{vpower} * (D\,\text{mod})^2}{100^2 * F\,\text{mod} * (\text{Laideal} - \text{mideal} * \text{alpha})} \qquad (3.1)$$

where, vpower is the input power.

D_{mod} is the modulation duty cycle in % = 40

F_{mod} is the modulation frequency =1000Hz

Laideal is the inductance value in the aligned position= 0.0084

mideal= (Laideal-Lpideal)/thetaTm

Lpideal is the phase inductance at a rotor position of +/- 24º

thetaTm is the torque producing angle range, and 100 is to convert the % duty cycle to its decimal value.

In the Simulink model, equation 3.1 is computed by the block labeled Fcn, if alpha <thetaTm (α<24˚), otherwise equation 3.2 computed by Fcn1 is used.

$$g(\alpha) = \frac{\text{vpower} * (\text{D mod})^2}{100^2 * \text{F mod} * (\text{Lpideal} - \text{mpideal} * (\text{alpha} - \text{thetaTm})} \qquad (3.2)$$

mpideal= (Lpideal-Luideal)/ (thetaum-thetaTm);

thetaum is the unaligned rotor position angle in radians

It is known that the maximum inductance occurs at the position of minimum reluctance when the rotor pole aligns with the stator poles, and the minimum inductance occurs when the rotor and stator poles are completely unaligned. Therefore the inverse inductance g(α) is maximum when the rotor and stator align with each other and is minimum when unaligned.

Figure 3.10 shows the alpha profile varying between -30˚ to +30˚ and figure 3.11 shows the corresponding g(α) profile obtained by simulating the block galpha.

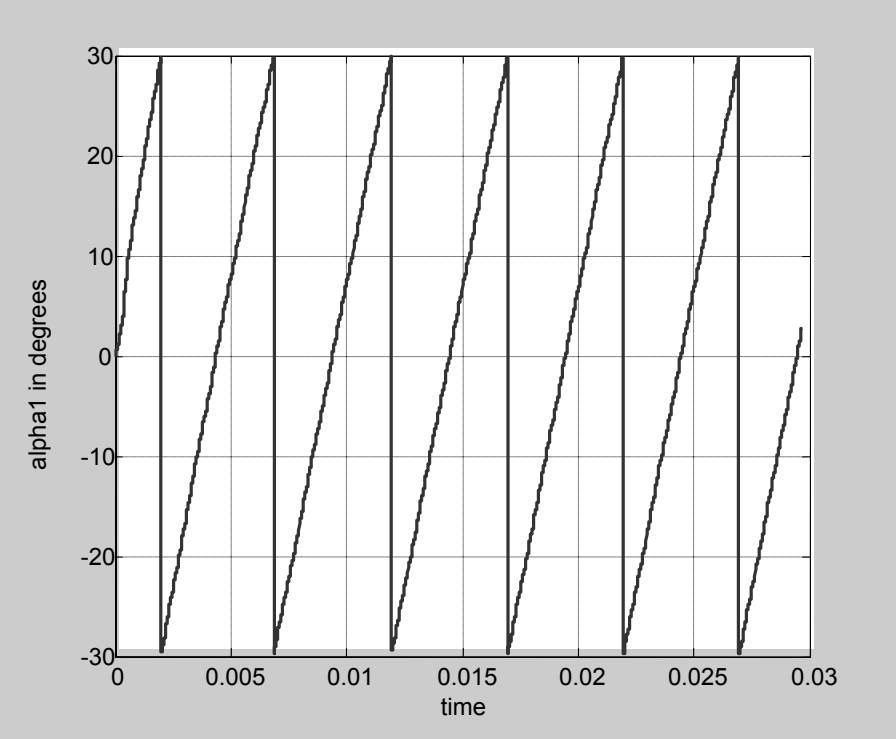

Figure 3.10 The rotor position angle profile.

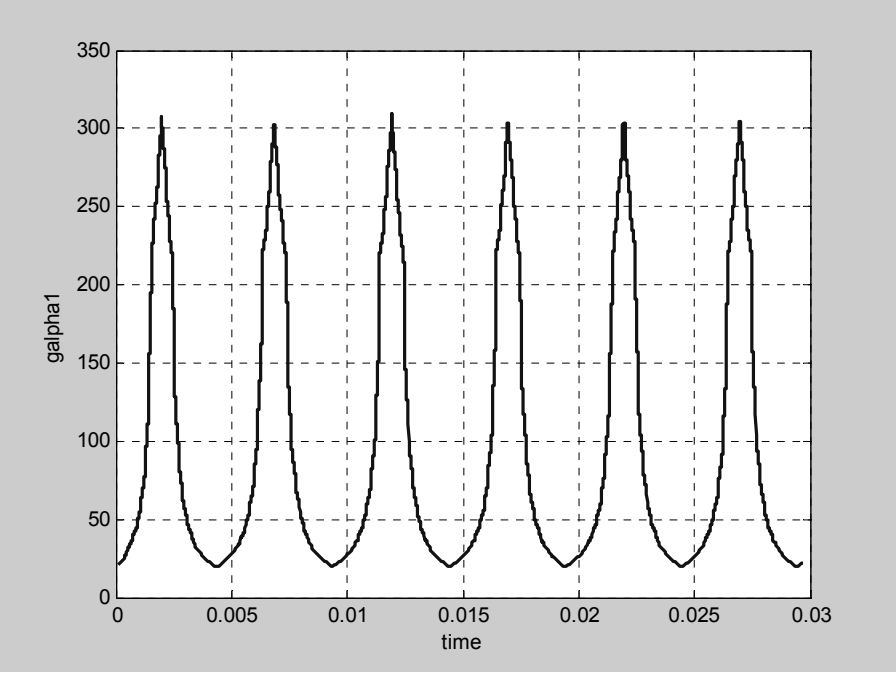

Figure 3.11 The inverse inductance value profile.

Therefore, as seen in these figures, when a rotor pole is in complete alignment with a stator pole i.e., α is 0˚, the inductance is at its maximum, and therefore the inverse of inductance, g(α) is minimum. As α increases from 0˚ to 30˚, inductance decreases thus increasing the value of g(α) to the maximum value and as α decreases from 30˚ to -30˚, g(α) also decreases to the minimum value.

Now, consider the Simulink model of the block for computing the error. Four of the galpha blocks shown in figure 3.9, one for each phase, are instantiated in the error block. The values of α for each phase are the input to each of these blocks and they compute outputs g(α) for all four phases. The definition of the error requires all four values of the computed g(α)s and all four values of the measured g(θ)s, one for each phase. If a phase is producing torque there is no measured g(θ) from that phase. In this case g(α) is substituted for g(θ). Depending on the input signal isense, a switch is used to select if g(α) should be substituted for g(θ). Four switches are used, one for each of the four phases. Then the values of g(α) and g(θ) of all the four phases are connected through multiplier blocks and then are added or subtracted accordingly to calculate the error. The simulation result from the error block for values of α and θ varying as shown in figure 3.10 is given in figure 3.12. As seen in the result, the average error goes to zero as the estimated value of the rotor position (α) becomes equal to the actual value of the rotor position (θ). The average value is zero though there are fluctuations.

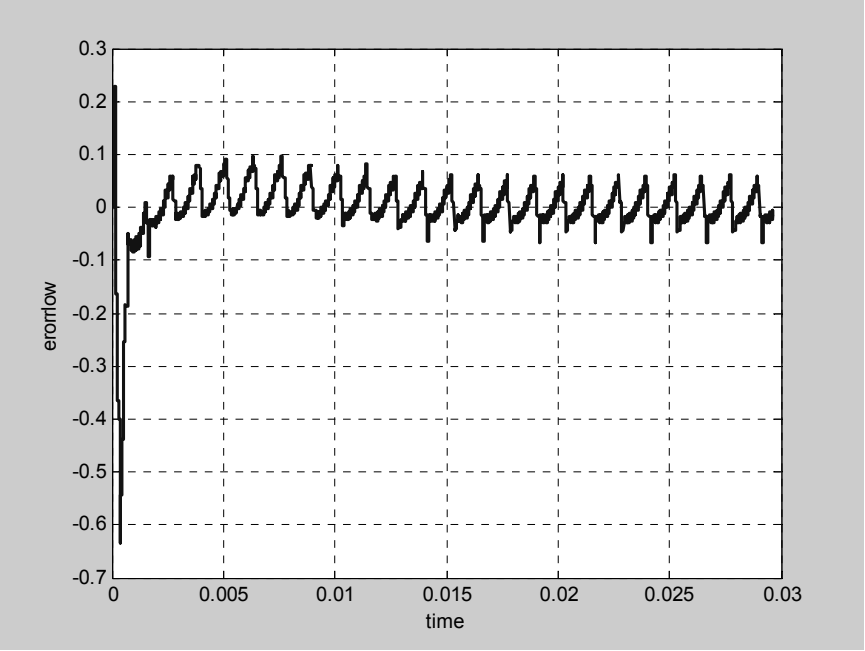

Figure 3.12 The error calculated for the actual and estimated inverse inductance profile.

Now consider the Simulink model of the rotor position estimator shown in figure 3.8. The sub-block galpha and the error block are instantiated in this main block.

The error calculated by the error block is used as the input to a state estimator to estimate the rotor position. This is done using the equations in chapter 2. The gains in the state estimator are chosen such that the state estimator is stable and its output converges to the correct values rapidly. The error multiplied with a gain factor is given as an input to an integrator whose output is the angular speed of the motor. Figure 3.13 shows the predicted estimated SRM speed using the Simulink model when the actual speed is constant and there is an initial error is the estimated rotor position. From the figure, we observe that the speed reaches its required constant value as the error between the actual and estimated rotor position tends to zero. Figure 3.14 shows the estimated rotor position for the SRM. The estimated angle has been wrapped such that it falls in between -30˚ and +30˚.

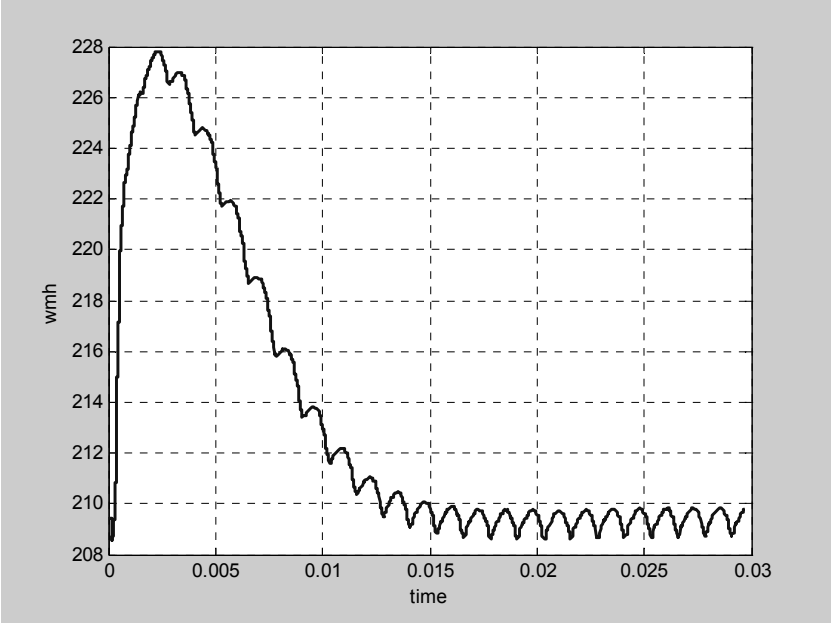

Figure 3.13 Estimated rotor speed for the SRM in radian/second (for a constant speed of 2000rpm).

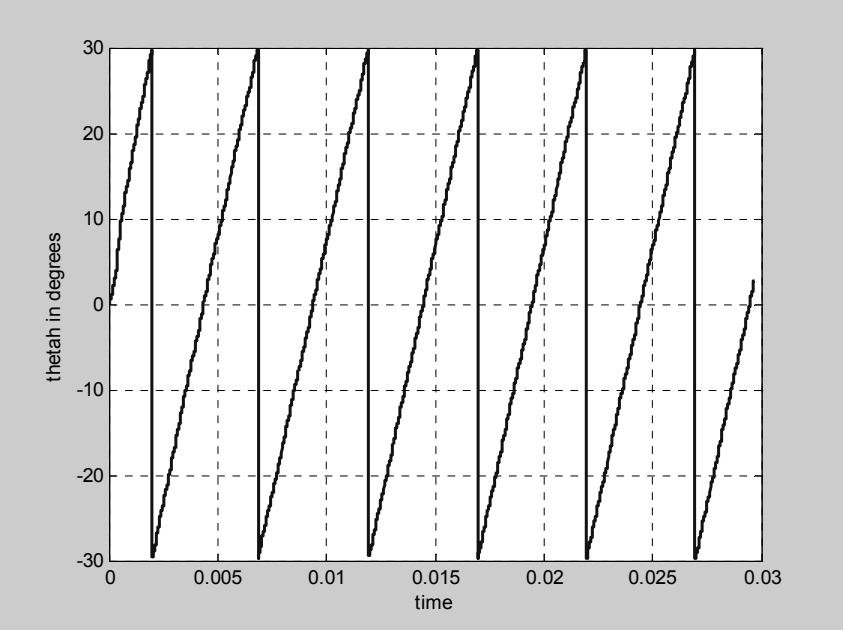

Figure 3.14 Estimated rotor position for the SRM

The next chapter describes the design and implementation of the commutator and position estimator design described in this chapter using an FPGA.

CHAPTER 4

IMPLEMENTATION OF THE ROTOR POSITION ESTIMATOR ON AN FPGA

4.1 Block Diagram of the system

The block diagram of the rotor position estimator and commutator system summarizing the inputs and outputs is as shown in the figure 4.1.

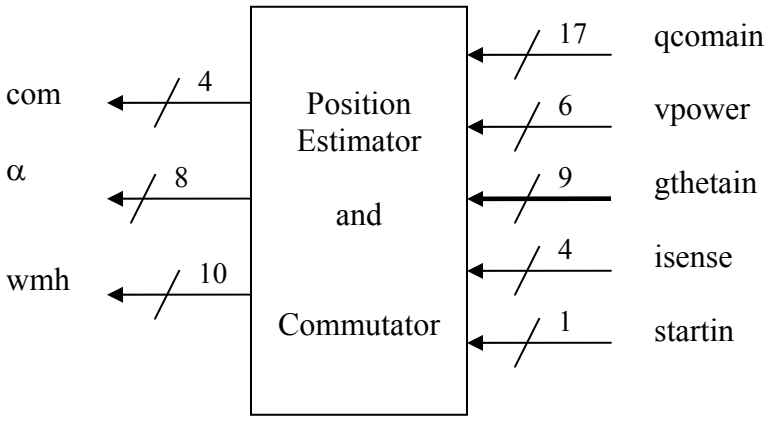

Figure 4.1: Block diagram of the Position Estimator and Commutator

The number of bits required to represent the inputs and outputs are selected to obtain the required accuracy. The width of the inputs and outputs must be determined in order to implement the position estimator and commutator. Each of the inputs and the outputs and their bit representation are discussed in this section.

The input 'qcomain' is a concatenation of the angles θ_{sm}, θ_{lrg} and the signal $\theta_{lrg}>\theta_{sm}$, which are given as a single input. As discussed earlier θ_{sm} and θ_{lrg} establish the range of rotor positions within which an SRM phase is to produce torque. Since there are six rotor poles, the angle between the rotor poles is 60º. Thus the values for θ_{sm} and θ_{lrg} can be anywhere between -30º to +30º. So, a 6-bit number can be used to represent these angles to within approximately 1º since $2^6 - 1 = 63$. However, the allowable error is 0.5º, so that it is necessary to choose a 7-bit number to represent the angle. One more bit is required to

represent the sign. Thus, an 8-bit word is used to represent each of θ_{sm} and θ_{lrg}. Since θlrg>θsm is a select signal as described in chapter 3, it is a 1-bit number. Thus, a 17-bit number is required to represent 'qcomain'.

The value of the DC input voltage 'vpower', which is the voltage applied to the motor, is typically 42 volts for the SRM drive system under consideration. The number of bits chosen to represent the value of the voltage is 6 bits since

$2^6 - 1 = 63$.

Thus assuming the input voltage is always less than 63V, the voltage can be represented to within 1V. The actual rotor position angle is unknown and what is measured is g(θ) (gtheta). This signal is measured from the demodulated (low pas filtered) current sense pulses. The analog version of this signal will be scaled to be between 0 and 12V. This analog signal will be converted to digital form before being input to the FPGA. The input 'gthetain' represents the four different g(θ) signals, one for each of the four phases. As seen in the Simulink block diagram in Fig. 4.1, 'gthetain is represented by a dark line indicating that it is a bus, consisting of input from all the four phases. The measured g(θ)s will be used to compute an error with the calculated g(α)s. To insure the accuracy of this error calculation which includes multiple multiplies and adds, the number of bits used to represent g(θ) in digital form is 9 bits, one bit more than used for the angles.

The input 'isense' is the select signal given as input to the estimator to choose between using g(θ) and g(α) in the error calculation depending on whether the given phase is producing torque. There is one 'isense' select line for each of the four phases. Thus 'isense' is a 4-bit number, with each bit coming from one of the four phases.

The input 'startin' is the select signal given to the commutator and is used to turn the commutator on or off. Thus, it is a 1-bit number.

The output 'wmh' is the estimated angular speed of the motor. For the SRM this position estimator is being designed for the machine inductance and the smallest current values that can be measured while still measuring the largest current values that are required limits the maximum speed the position estimator can operate at 2,500rpm which gives the maximum value of speed to be 262 radian/second. Therefore, the number of bits required

to represent wmh is 10 bits including the sign bit which is required because the speed is negative if the SRM is turning backwards.

Consider the number of bits to represent the estimated rotor position, α. The rotor position can vary from 0 to 360˚. Since there are 6 rotor poles, the angle between each of the poles is 60º. Now considering the position of the rotor pole with respect to the stator pole was defined to vary from -30º to +30º, the number of bits chosen for representing the rotor position should be able to represent 60 values including the sign. Choosing the allowable error for the estimated rotor position to be 0.5º, a 7-bit number is chosen. Since a sign bit is also required, a total of 8 bits is required to represent the estimated rotor position.

4.2 Angle wrapping

The fundamental principle of operation of a SRM is the variation in flux linkage with the change in the angular position of the rotor. When a rotor pole pair aligns with the stator pole pair, the flux linked by that stator phase is at a maximum. When the rotor pole pair moves away from the stator pole pair, it becomes unaligned with the stator phase, then the flux linked by the stator phase is at a minimum. Thus, the stator flux goes from maximum to its minimum as each of the six rotor poles pass through the stator poles.
The flux profile versus rotor position is as shown in figure 4.2. Since the SRM for which the position estimator is being implemented has four phases and its rotor has six poles, the flux will repeat six times in 360º so the angular period is 360º /6 = 60º.

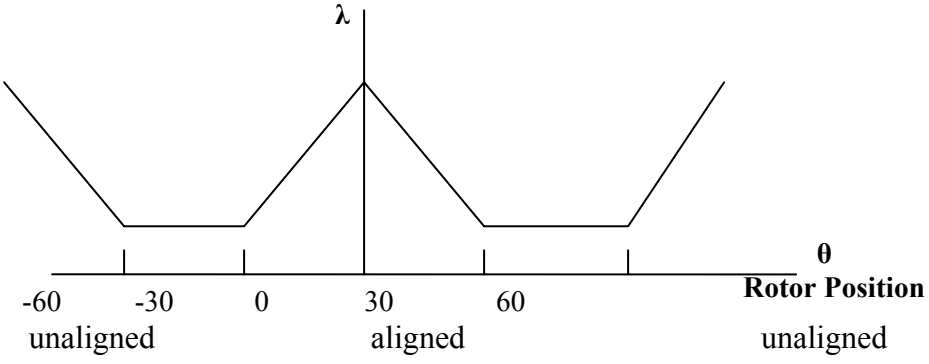

Figure 4.2: Flux linked by phase A as a function of the rotor position

As seen in the flux profile, the flux changes from unaligned to aligned position when the rotor position changes from -30º to 0º and then goes back to unaligned position from 0º to 30º. Since it repeats after every 60˚, it is sufficient to consider only the angles from -30º to +30˚ of rotor rotation.

Since there are four stator phases, angle between them is 360˚/4 = 90˚. Hence, the angle between each of the eight stator poles is 45º. Therefore, once the rotor position for one phase has been determined, by shifting the rotor position by multiples of 45˚, the rotor position for all the other phases is determined relative to their own stator poles.

The rotor position has the periodicity of the phase flux so that it will repeat every 60º just as the phase flux does. Thus, the rotor position only needs to be estimated for the interval from -30º to +30˚. Thus when the rotor position reaches 30˚ it goes back to -30˚. This is known as wrapping the angle at 30˚. Assume that θ represents the actual angle and α represents the wrapped angle confined between -30º and +30˚ Then if θ = 45º, then α= θ-60˚=-15˚.Similarly, if θ = 150º, then α= θ-120˚=30˚ and so on.

The type of the input and output signals and the number of bits assigned for each of them is summarized in the table 4.1.

I/O signal	Type	I/O, the signal represents	# of bits used to represent the signal
qcomain	input	Turn-on and turn-off angles input to all four phases of commutator	17
vpower	Input	Voltage to the motor	6
gthetain	Input	g(θ) input from all four phases	9*4=36
isense	Input	sense select signal to estimator	1*4=4
startin	Input	Start signal to all four phases of commutator	1
com	Output	Output from all four phases of commutator	1*4=4
α	Output	Estimated rotor position	8
wmh	Output	Rotor speed	10

Table 4. 1: The Input and Output signals of the circuit represented in the HDL code

4.3 Selection of the programmable device

The different programmable devices that can be used to implement the position estimator are

1. Microcontroller
2. Digital Signal Processor (DSP)
3. Field Programmable Gate Array(FPGA)

While microprocessors have been the dominant devices in use for general-purpose computing for the last decade, there is still a large gap between the computational efficiency of microprocessors and custom silicon. Reconfigurable devices, such as FPGAs, have come closer to closing that gap, offering a 10 times benefit in computational density over microprocessors, and often offering another potential 10 times improvement in yielded functional density on low granularity operations. On highly regular computations, reconfigurable architectures have a clear superiority to traditional processor architectures. On tasks with high functional diversity, microprocessors use silicon more efficiently than reconfigurable devices. Microprocessors are not specifically designed to do calculations in real time.

FPGAs have proven extremely efficient for certain processing tasks. The key to their cost/performance advantage is that conventional processors are often limited by instruction bandwidth and execution restrictions or by an insufficient number or type of functional units. FPGAs exploit more program parallelism. By dedicating significantly less instruction memory per active computing element, they achieve a 10 times improvement in functional density over microprocessors. At the same time this lower memory ratio allows reconfigurable devices to deploy active capacity at a finer grained level, allowing them to realize a higher yield of their raw capacity, sometimes as much as 10 times than conventional processors.

Based on all the factors, the FPGA is expected to be able to perform the position estimation function with a shorter sampling/update time. The position estimator and commutator functions described above have already been implemented by engineers at Mechatronic Systems using a TMS320C6701-150 DSP based commercial DSP board. This implementation resulted in a sampling and position estimate update time that just met the SRM drive-system requirements. Simulink simulations showed that an analog (zero sampling and up date time) implementation of the position estimator resulted in smaller errors compared to the DSP implementation. Thus, the system performance will be improved if the sample and position estimate update time is reduced from what was achieved with the DSP. Thus, the research presented here was undertaken to verify that a FPGA based implementation of the position estimator and commutator would result in a design with a reduced sample and update time compared to the DSP implementation and to quantify the improvement.

4.4 Field Programmable Gate Arrays

A Field Programmable Gate Array (FPGA) is a microchip made with millions of programmable logic gates. FPGAs are readily programmable and can be programmed and reprogrammed repeatedly. They must be programmed by users to connect the chip's resources in the appropriate manner to implement the desired functionality.

A FPGA contains a regular, extendable, flexible and programmable architecture of logic blocks surrounded by input/output blocks on the perimeter. These functional blocks are linked together by a hierarchy of highly versatile programmable interconnects.
The basic block diagram of an FPGA is as shown in the figure 4.3.

.

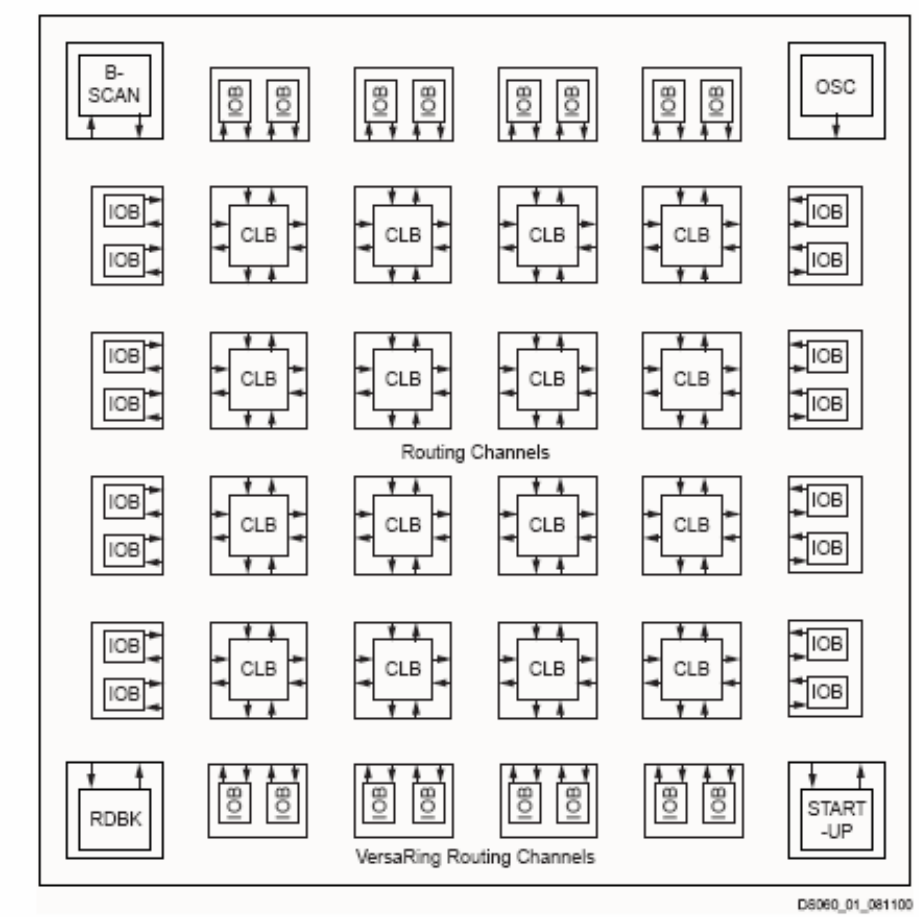

Figure 4.3 Block diagram of an FPGA

The basic components of an FPGA are:

1. CLBs (Configurable Logic Blocks)
2. IOBs (Input/Output Blocks)
3. Switch matrix (resources for interconnection)

An FPGA consists of thousands of CLBs. Each CLB consists of a small number of inputs and outputs, a look-up table (LUT), flip-flops and a few basic gates. Multiplexers are used to configure the interconnections between CLB components and the inputs and outputs of the CLB. LUTs are used to implement combinational logic by implementing the truth tables corresponding to the logic circuit. The flip-flops are used as sequential components and can be configured to operate on either edge of the clock or as latches. Thus, a CLB can be configured to implement the combinational and sequential

components that have been assigned to it. Loading these configuration bits for each CLB within the FPGA is referred to as the process of programming the FPGA.

The perimeter of configurable Input/Output Blocks (IOBs) provides a programmable interface between the internal logic array and the external device package pins. Each IOB contains a few logic gates and flip-flops. The input and output signals can directly pass to the pin or can be stored in a flip-flop as shown in figure 4.4.

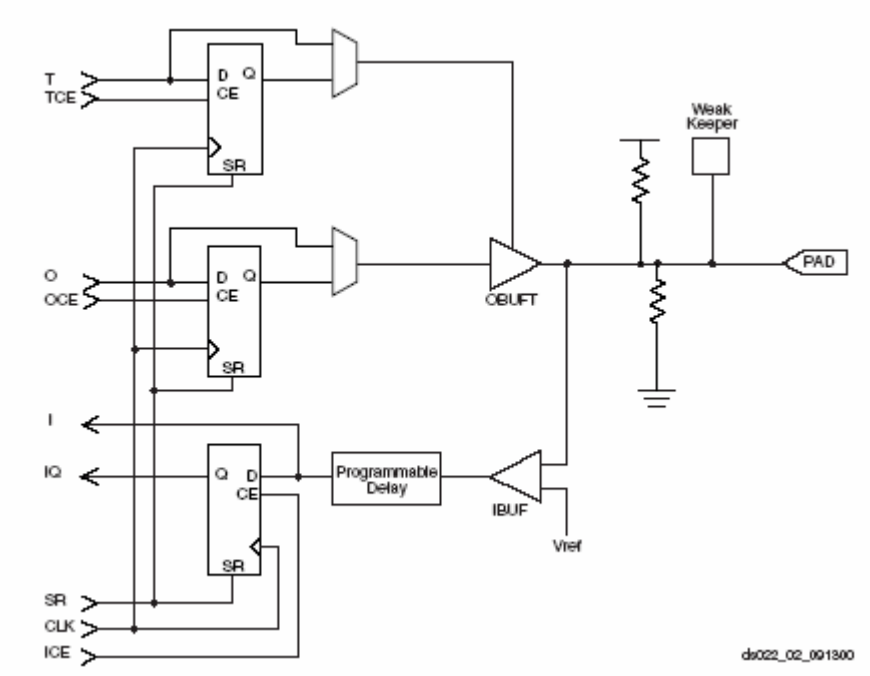

Figure 4.4 Block diagram of a Virtex IOB

Programmable-interconnection resources within the FPGA provide routing paths to connect inputs and outputs of the IOBs and CLBs into logic networks.

The interconnection wires in the FPGA are organized as horizontal and vertical routing channels between rows and columns of logic blocks. At the intersection of the horizontal and vertical wires are switches, which collectively form a switch matrix.

The FPGA user logic functions and interconnections are determined by the configuration program data stored in internal static memory cells.

The factors affecting the selection of the programming device are:

1. Number of available IOBs
2. Total number of system gates.

Number of IOBs

The Input/Output Block (IOB) provides a programmable, bidirectional interface between I/O pin and the FPGA's internal logic.

Since the total number of bits required to represent all of the inputs and outputs is 81and each of the bits is assigned to a separate I/O port, the number of independent I/O ports required is 81. This method of transmitting data, where each bit of information is assigned to an individual port is called parallel data transmission.

Number of system gates

There are many intermediate signals generated, like the signal to represent the error calculated between g(θ) and g(α), the signals given as input and taken as output from the two integrators. There are many calculations including multiplications and divisions to be done, which require many logic gates.

4.5 Type of FPGA

Based on the availability of the required number of IOBs and the number of system gates and multipliers, either of the following devices can be used.

1. Virtex XCV800
2. Spartan XC3S1000

The datasheets for each of these devices are found in the Xilinx website, the link to which is given in the reference 5 and 6.
The number of available resources in each of the FPGA chips, as given in the data sheets is summarized in the table 4.2.

Device	**Virtex XCV800**	**Spartan XC3S1000**
System gates	888,439	1M
Total CLBs	4704	1920
Maximum user I/O	240	391
Dedicated multipliers	N/A	24

Table 4.2 Comparison of Resources available in Virtex and Spartan

The design is implemented on both these devices to determine which would be a better choice. The implementation results are as shown below.

Implementation results using Virtex XCV800

Release 6.1.03i Par G.26

Selected Device: v800hq240-4

Number of Slices:	7844	out of 9408	83%
Number of Slice Flip Flops:	10787	out of 18816	57%
Number of 4 input LUTs:	7178	out of 18816	38%
Number of bonded IOBs:	129	out of 170	75%

Number of GCLKs:	1	out of 4	25%

Implementation results using Spartan XC3S1000

Selected Device: 3s1000ft256-4

Number of Slices:	7705	out of 7680	100% (*)
Number of Slice Flip Flops:	10769	out of 15360	70%
Number of 4 input LUTs:	7103	out of 15360	46%
Number of bonded IOBs:	92	out of 173	53%
Number of MULT18X18s:	28	out of 24	116% (*)
Number of GCLKs:	3	out of 8	37%

WARNING:Xst:1336 - (*) More than 100% of Device resources are used

As one can observe the number of slices (logic blocks) used by the Spartan chip is more than what is available and hence the design wouldn't fit onto the Spartan XC3S1000. The number of multipliers available in the Spartan chip is also less than required. Though using the Core generator would reduce the number of multipliers required it would increase the number of logic gates required. Thus, the Spartan XC3S1000 is not a feasible choice. As seen in the implementation result of the Virtex chip, the design will fit onto it while not having excessive resources unused. Hence, the Xilinx Virtex XCV800 is chosen to implement the circuit and test it.

4.6 Xilinx Virtex XCV800

The general characteristics of a FPGA have been discussed in the previous section. The main characteristics of the chosen FPGA, the Xilinx Virtex XCV800 are discussed here.

The Virtex user-programmable FPGA, shown in Figure 4.5, comprises two major configurable elements: configurable logic blocks (CLBs) and input/output blocks (IOBs). CLBs interconnect through a general routing matrix (GRM). The GRM comprises an

array of routing switches located at the intersections of horizontal and vertical routing channels. Each CLB nests into a VersaBlock that also provides local routing resources to connect the CLB to the GRM. The VersaRing I/O interface provides additional routing resources around the periphery of the device. This routing improves I/O routability and facilitates pin locking.

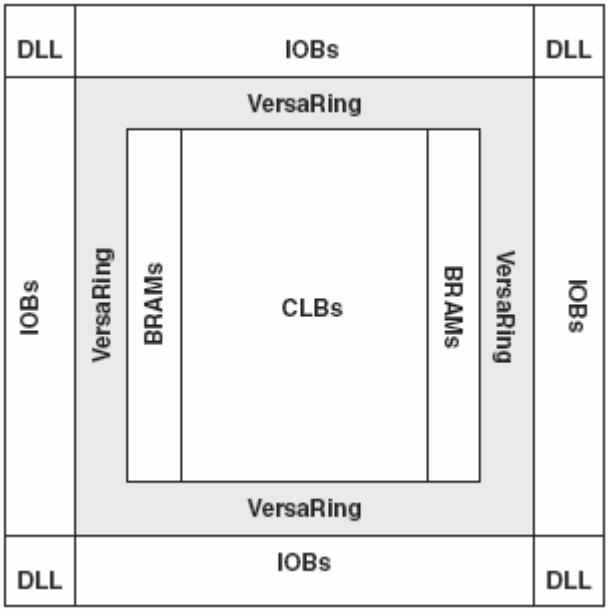

Figure 4.5Virtex architecture overview

The Virtex architecture also includes dedicated block memories, Clock DLLs for clock-distribution delay compensation and clock domain control and 3-State buffers (BUFTs) associated with each CLB that drive dedicated segmentable horizontal routing resources.

4.7 Digital Design Flow

The design flow is the sequence of events that begin with some abstract specification of a design and ends with a configured FPGA. This design procedure consists of five steps: Design entry, Simulation, Synthesis, Implementation, Device download and program file formatting. The design flow described here is in reference to the Xilinx ISE (Integrated Synthesis Environment) 6.1i CAD tool and is as illustrated in the figure. However, most of the activities will have a counterpart in any vendors' design flow.

MODERN CAD-TOOL BASED DIGITAL SYSTEM DESIGN FLOW

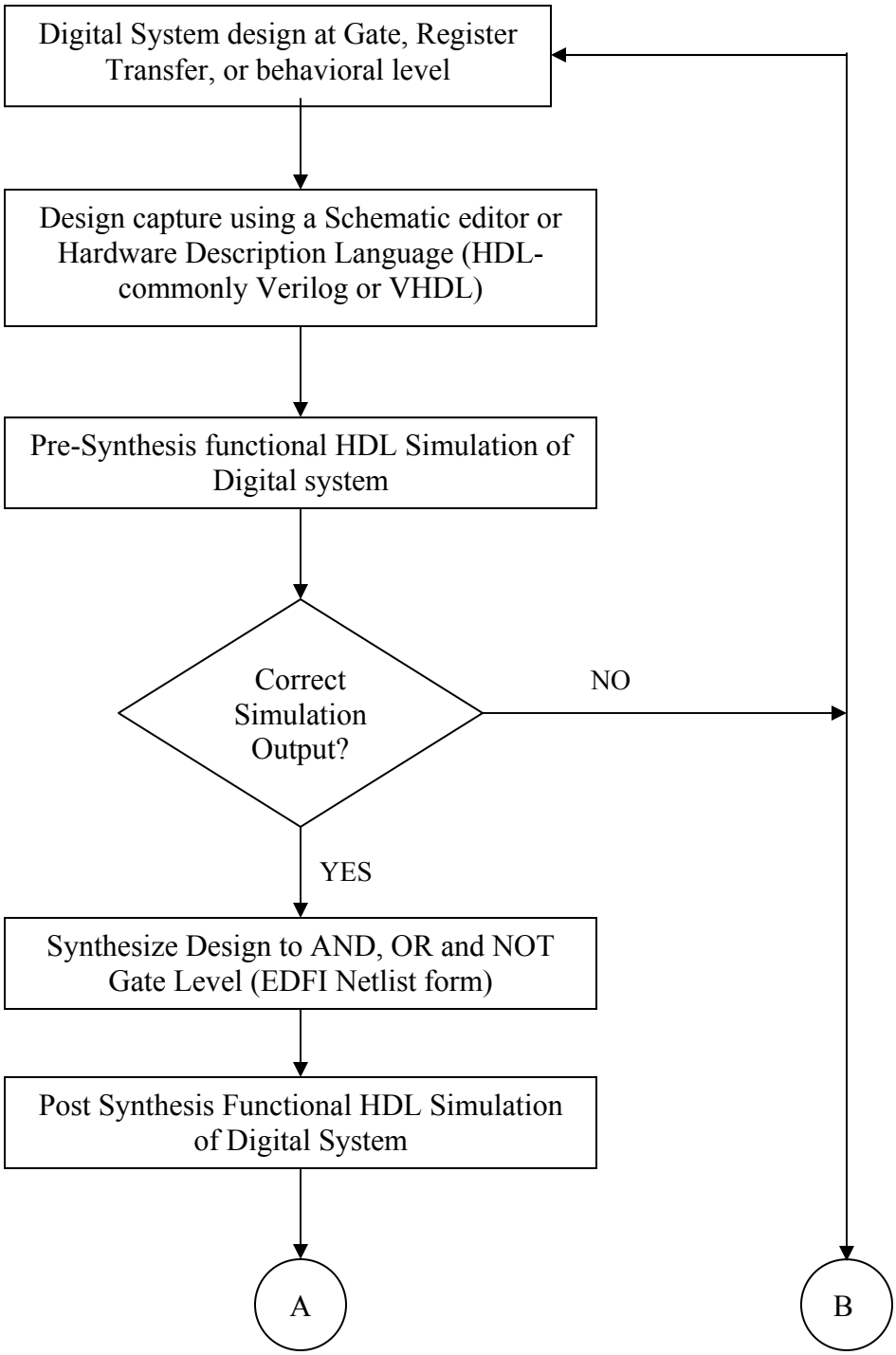

Figure 4.6 Digital Design Flow

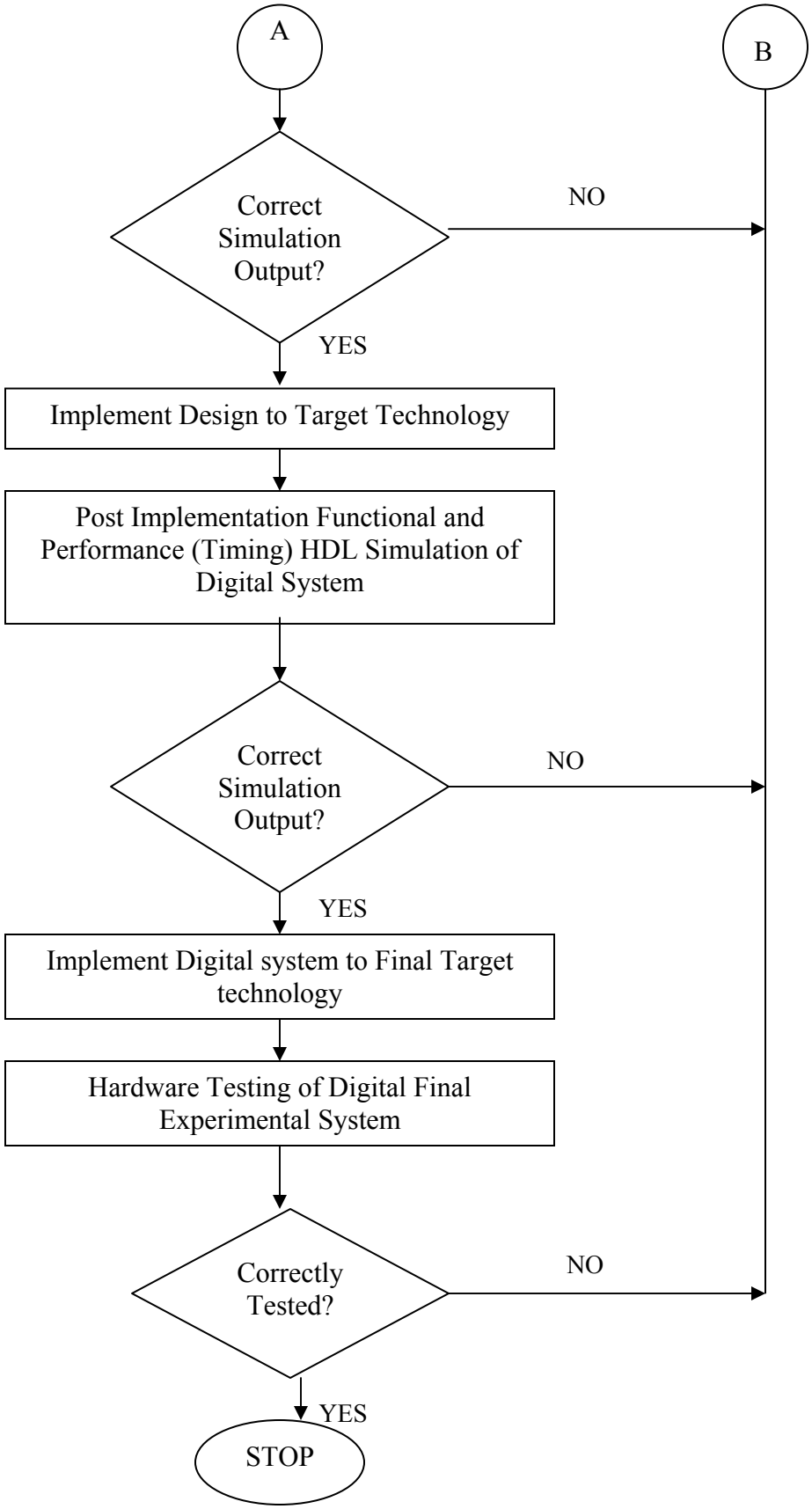

Figure 4.6 Digital Design Flow (cont.)

The initial description of any design may be in the form of state diagrams or Boolean expressions, but these are refined through various stages to into an FPGA implementation. These various stages are discussed below.

Design Entry: The design entry describes the functionality of the design. It could be done by schematic capture or a state transition diagram or by constructing an HDL based model using Verilog HDL or VHDL. An HDL model is constructed by writing HDL code using a text editor. Modern simulators and synthesis tools provide syntax-directed editors and facilities for insertion of language templates to facilitate easier coding. This step produces the HDL source for a model that is analyzed to an internal form while it is checked for conformance to the syntax and semantics of the HDL. [10]

Behavioral Simulation: The HDL model is simulated at the Register Transfer level (RTL) to establish functional correctness. This is the step that involves simulating the functionality of a device to determine that it is working as per the specification and that it will produce correct results. This type of simulation is very important to get as many bugs out of the HDL code as possible. After the design entry is done, a functional simulation is done. If there is an error the 'design entry' step is re-visited and necessary changes are made leading to a successful simulation.

Synthesis: It is the process where the RTL design is optimally translated to the gate level design which can be mapped to the logic blocks in the FPGA meeting the timing and area constraints as desired by the user. In this step, the HDL code is converted into a device netlist format.

Implementation: Design implementation is the process of mapping, placing, routing and generating a BIT file for the design.
Mapping: Once the gate-level netlist is designed, the next step is mapping the design onto an FPGA. The design is mapped to the primitives such as function generators, flip-flops or latches that are used in the target chip.

Place and Route: The mapped design is placed by assigning the primitives to configurable logic blocks (CLBs). After placing, the primitives are connected by routing the connections through the switch matrix. Once the design is placed and routed, accurate information about timing delays between parts of the circuit can be obtained. After place and route, the design is simulated for design verification since post place and route simulation is more accurate than the functional simulation.

Bit generation: A bitstream is generated from the physical place and route information.

Programming: The configuration bits or bitstream is loaded into the target FPGA. The chip has now been configured to implement the design.

In this thesis, the EDA tool used for the design is the Xilinx ISE (Integrated Synthesis Environment) 6.1i and the HDL selected was Verilog HDL.

The HDL code for the required design is written using the Verilog Hardware design language.
Each of the codes for the blocks galpha, error, and integrators instantiated into the main design circuit are discussed here separately in detail.

The functional block diagram of sensetheta is as shown in the figure 4.7. The codes of the blocks denoted by error, intH1, intH2, wrap are instantiated into the main code. Each of the blocks is discussed in detail here.

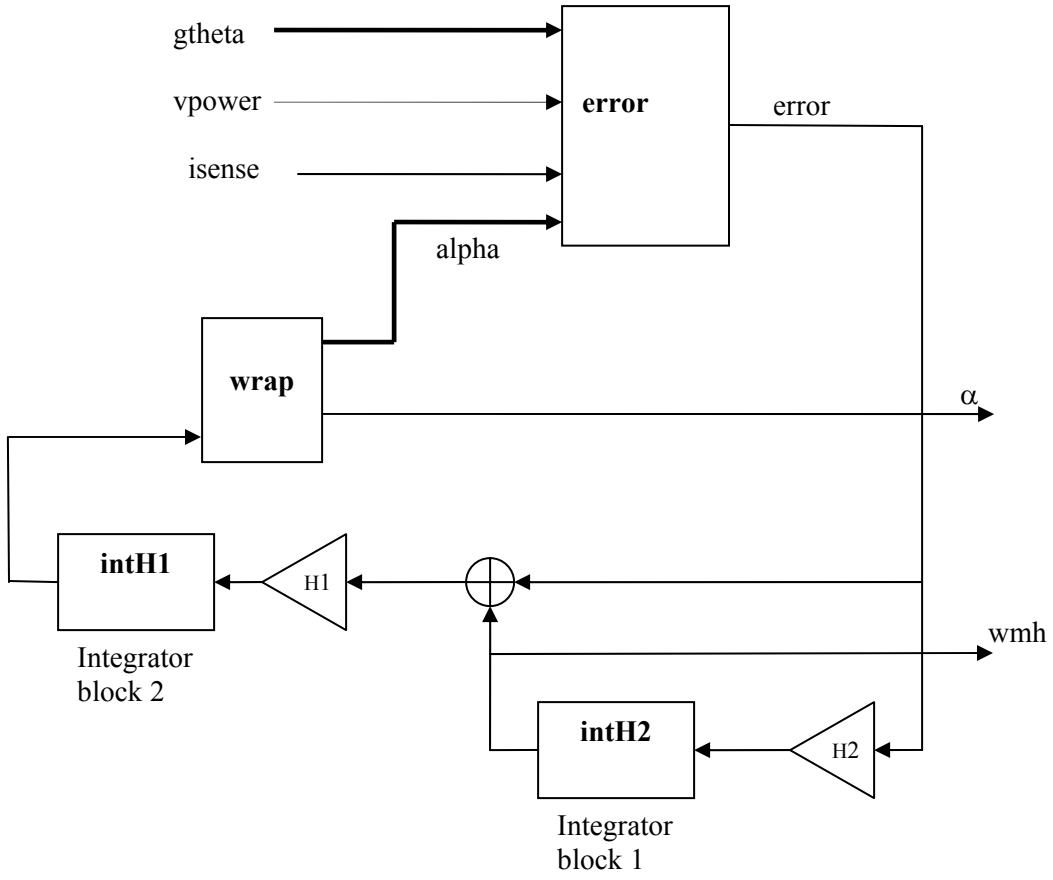

Figure 4.7 Block diagram of the block sensetheta

The block diagram to calculate the value of g(α), when a value of α is given as input is shown in the figure 4.8.

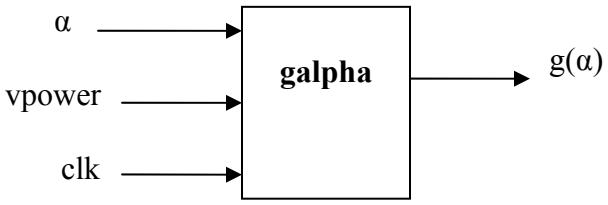

Figure 4.8 Block diagram of the block galpha

The function g(α) is calculated by using the equations 4.1 and 4.2.

$$g(\alpha) = \frac{vpower * (D_{mod})^2}{100^2 * F_{mod} * (Laideal - mideal * alpha)} \qquad \text{if alpha<thetaTm} \qquad (4.1)$$

$$g(\alpha) = \frac{\text{vpower} * (\text{D mod})^2}{100^2 * \text{F mod} * (\text{Lpideal} - \text{mpideal} * (\text{alpha} - \text{thetaTm})} \quad \text{if alpha>thetaTm} \quad (4.2)$$

The function output g(α),and its input α are the variables in the equation. The remaining parameters vpower, D_{mod}, F_{mod}, Laideal, mideal, Lpideal, mpideal seen in the equation are constants. In order to reduce the logic required to represent these constants, they are represented by binary numbers instead of integers that use more logic blocks. Some of these constants are real numbers, so multiplication factors are used, so that they are represented in binary form. The table 4.4 shows the actual values of the constants, the number of bits used to represent them in binary form and their binary representation.

Constant denoted by	**Actual value**	**# of bits required**	**Binary representation**
Vpower	42	6	6'b101010
D_{mod}	40	6	6'b101000
F_{mod}	1000	10	10'b0111111000
Laideal	0.0084*16384= 137	8	8'b10001001
mideal	0.0183*16384=300	9	9'b100101100
Lpideal	0.0075*16384=12	4	4'b1100
mpideal	0.0021*16384=34	6	6'b100010

thetaTm	0.4189*180/3.14=24	5	6'b011000

Table 4.4 Constants used in the block galpha of the HDL code

As seen in table 4.4, the value of Laideal is a real number (0.0084), which cannot be represented in binary form easily. Thus it is multiplied by the factor 16384 which gives the new number 137., It is the decimal number that is then represented in binary form as shown in table 4.4. The same is done for the constants denoting mideal, Lpideal and mpideal.

As seen in the equations 4.1 and 4.2, there are several arithmetic operations like addition, subtraction, multiplication division and conditional operations that must be handled by the Verilog HDL and the EDA tool. Performing all of the operations except division is easily handled by Verilog HDL and the EDA tool. The Xilinx tool can handle division by two and higher powers of two like 2^1, 2^2, 2^3, etc., but cannot perform division by numbers other than multiples of two. A special algorithm is required to perform that operation. Various methods are available to overcome this problem.

1. Algorithmic method: Several algorithms are available which can perform division by iterative subtraction techniques. But this method is too tedious and complex.
2. Using Core Generator: The core generator available in the Xilinx tool can be used to generate a core to perform division. The same core can be used to generate different instances as long as the parameters are of the same width. This is a very effective method when many divisions have to be performed, though it takes several clock cycles to complete one division. A pipelined division core can be used to improve efficiency.
3. Using multiplication factors: In this method, the divisor is multiplied by a factor so that it is converted into powers of 2. Dividing by this power of 2 can be performed easily. This is approach is illustrated in the following example.

In the block integrator representing intH1, the input has to be multiplied by 3.5μs, but since it cannot be represented in binary form, it is multiplied by a factor such that it can be represented in binary format and also equals that value. Thus it is multiplied by a factor equal to 30/1024*1024*8 which is equal to 3.5μs and also the divisor part which is 1024*1024*8 are all powers of 2 and thus the division can be carried on without any trouble. However this method cannot be used where the input changes periodically since different factors will have to be used for different inputs.

In this thesis, both the method using a core generator and the method using multiplication factors are used.

A pipelined divider core has been generated for the block galpha where g(α) is calculated from α. Here the dividend and divisor are of fixed width for different values of input and thus the core divider is used to obtain the outputs. Another instance of the same pipelined divider core has also been used for other functions besides the g(θ) function with different sets of inputs and outputs but with the same bit width, thus reducing the number of logic blocks used. The issues relating to the Core Generator are discussed in the next section.

4.8 Core Generator

The Xilinx CORE Generator System offers an optimized, predefined set of building blocks for common functions. It provides a catalog of user-customizable functions ranging in complexity from simple arithmetic operators (adders, accumulators, and multipliers), memories and FIFOs, to networking interfaces and system-level building blocks such as filters and transforms. It simplifies the design steps and brings the design to completion faster while still achieving high performance.

The cores delivered through the CORE Generator can be tailored to the design requirements through their user-friendly core customization GUIs. Simply by specifying the parameters, an optimized core can be generated for the target FPGA device. The core generation process fabricates the logic for the core, partitions it into configurable logic

blocks (CLBs), and then places the CLBs relative to each other. The relative placement of CLBs making up a core is maintained as the core is integrated into the overall design and placed anywhere in the FPGA. The Core generator is used to generate a multiplier core and a divider core in this thesis.

CORE MULTIPLIER

The multiplier core is a high-speed parallel implementation that multiplies an N-bit wide variable times an M-bit wide variable and produces an N+M bit result. It accepts the parameters and accordingly creates a design using a parameterized Verilog HDL recipe. Verilog HDL instantiation code and a schematic symbol are created along with the netlist for the design. An area-efficient, high-speed algorithm is used to give an efficient, tightly packed design. Each stage is pipelined for maximum performance. In addition to this area-efficient design, the CORE Generator contains a performance optimized design that yields a 10% to 20% increase in speed, but uses more CLB resources.

The multiplier generator core here is used to generate a parallel multiplier.
The parallel multiplier takes 2 input buses, A and B each of N bits width where N can be 1 to 64, and calculates the multiplication of the values on these buses in parallel giving out an output Q of 2N bits wide. A schematic of inputs and outputs is shown in the figure 4.9. The inputs and output can be either of type signed or unsigned.

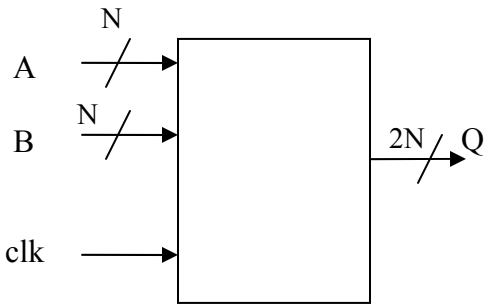

Figure 4.9 Schematic diagram of the multiplier core

The multiplier core is generated using look-up tables (LUTs) available in the FPGA device. It can be generated by using dedicated multiplier blocks too, but at the cost of latency.

This multiplier core is used in the design to carry out many multiplications, which otherwise would have required more built in multiplier blocks than are available in the FPGA.

This is illustrated by implementing the code, which used core-generated multipliers, and the code, which used built-in multiplier blocks separately. This is implemented on a Spartan XC3S1000. The implementation results are given below.

The implementation result for a design in which only built in multiplier blocks were used to carry out all the required multiplications are shown below.

Selected Device : 3s1000fg320-4

Number of Slices:	6844	out of 7680	89%
Number of Slice Flip Flops:	10322	out of 15360	67%
Number of 4 input LUTs:	5733	out of 15360	37%
Number of bonded IOBs:	200	out of 221	90%
Number of MULT18X18s:	28	out of 24	116% (*)
Number of GCLKs:	1	out of 8	12%

WARNING:Xst:1336 - (*) More than 100% of Device resources are used

As seen above, the number of internal multiplier blocks required is 28 while the number of available multiplier blocks is 24. Thus the design does not fit into the chosen FPGA. The implementation result for the design in which a pipelined multiplier core was used to create different instances to carry out all of the multiplications are shown below.

Selected Device: 3s1000fg320-4

Number of Slices:	7451 out of 7680 97%
Number of Slice Flip Flops:	10690 out of 15360 69%
Number of 4 input LUTs:	6703 out of 15360 43%
Number of bonded IOBs:	200 out of 221 90%
Number of MULT18X18s:	4 out of 24 16%
Number of GCLKs:	1 out of 8 12%

Now the number of internal multiplier blocks being used is reduced to 4 since all the required multiplications are carried on using slices and LUTs. As can be seen from both the results, the number of slices and LUTs being used in the second method is only slightly greater than in the first one. Thus, it can be concluded that using the core generator multiplier in the design is area-efficient.

CORE DIVIDER

The Xilinx LogiCORE Pipelined Divider divides an M-bit-wide variable dividend by an N-bit-wide variable divisor. The result of the division is an Mbit-wide quotient with an N-bit-wide integer remainder.

The input data can be unsigned or signed. Dividend values can range from 1 to 24 bits, divisor values can range from 3 to 24 bits, and fractional remainder values may range from 3 to 24 bits. It is an efficient, high-speed, parallel implementation.

The Schematic of the core divider is shown in the figure 4.10.

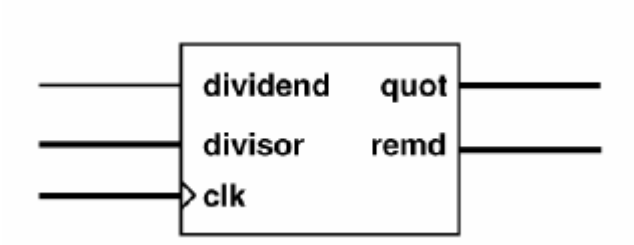

Figure 4.10 Schematic diagram of the core divider

The design is highly pipelined. The amount of pipelining can be reduced to decrease the area of the design at the expense of throughput. In the fully pipelined mode, the design supports one division per clock cycle after an initial latency. The design also supports the options of 2, 4, and 8 clock cycles per division after an initial latency.

The total latency (number of clocks required to get the first output) is a function of the bit width of the dividend.

In this thesis, a pipelined divider core is generated. The bit-width of the dividend and divisor is 20 bits. Since the dividend is 20bits and the remainder is chosen to be an integer, the latency is 24, i.e., it takes 24 cycles to get the first output, since the number of clock cycles per division is chosen to be 1.

Two instances of the core are created to carry out different division operations. As mentioned earlier, this core is used in the block galpha to carry out the division as given in function equations. Since the divisor is not a multiple of 2, the Xilinx tool cannot carryout the division and since the inputs of the division change at every calculation iteration, it is better to generate a core.

Now, consider the block diagram of the error block as shown in the figure 4.11. The values of g(θ) and α of all the four phases are given as inputs. The block galpha is instantiated into the error block gives out g(α). The number of bits required to represent α, g(α) and g(θ) is already discussed. The same number of bits is used to represent the values for each of the four phases.

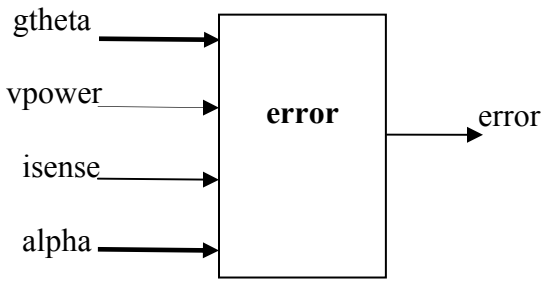

Figure 4.11 Block diagram of the block errorlow

The input isense consists of four one-bit signals from each of the phases. It determines whether the measured g(θ) from a phase is available for computing the error or if the computed g(α) must be used in its place because torque is being generated by that phase at the particular time instance. Depending on value of the 4 different isense inputs, logic is written to calculate the error using the equation repeated below.

$$error_{tot}(\theta,\alpha) = \left(g_2(\theta)g_1(\alpha) - g_1(\theta)g_2(\alpha)\right) + \left(g_3(\theta)g_2(\alpha) - g_2(\theta)g_3(\alpha)\right) + (g4(\theta)g3(\alpha) - g3(\theta)g4(\alpha)) + \left(g_1(\theta)g_4(\alpha) - g3(\theta)g_1(\alpha)\right)$$

Note that in this equation one or more of the g(θ)s might actually its corresponding g(α) if they are not available due to that phase generating torque. As seen in the equation, there are several multiplications involved, which require several multiplier blocks. The number of multiplier blocks required is out of range of the Virtex XCV800 FPGA. Thus a multiplier core is generated and the required numbers of instances of it are created to perform the different multiplication operations.

The block diagram of the integrator blocks is as shown in the figure 4.12.

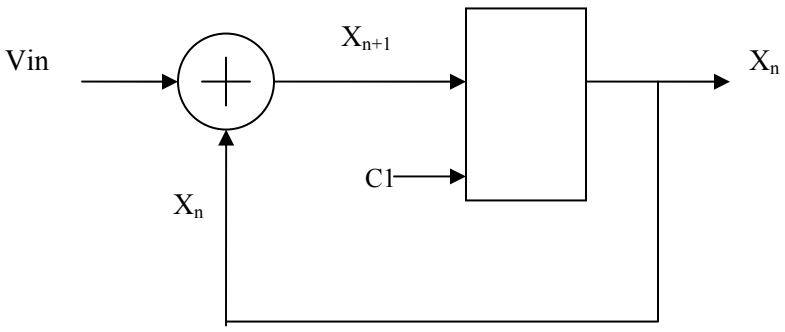

Figure 4.12 Block diagram of the Integrator circuit

The equation that describes the integrator blocks is equation 4.1.

$$X_{n+1} = Vin^{*} T + X_n \qquad (4.1)$$

where Vin the input to the integrator

X_{n+1} is the next state

X_n is the present state which is given as feedback and added to the input to get the next state.

C1 is the clock to the integrator whose period is T.

For a given input, the equation performs the integration and gives out the output.

The block diagram of the block wrap is as shown in figure 4.13.

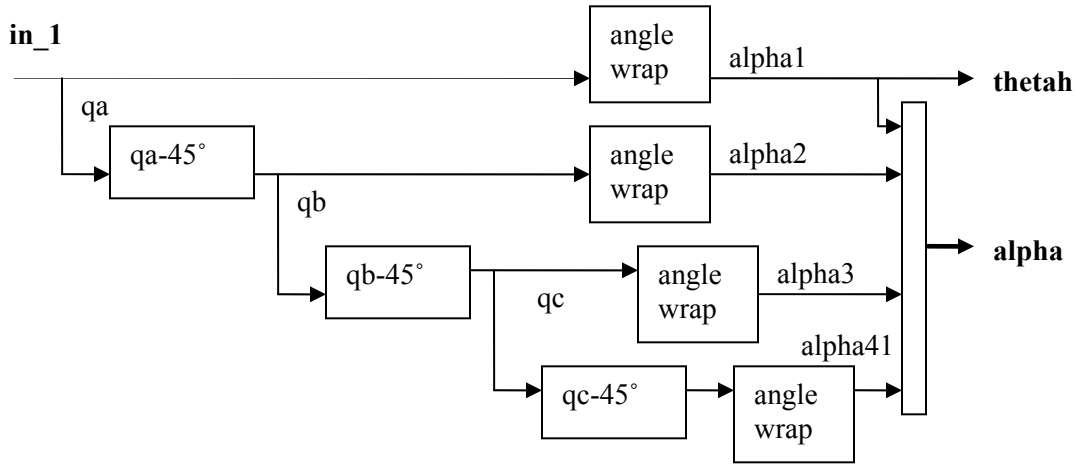

Figure 4.13 Block diagram for shifting and wrapping the angles

The output of integrator block 2 is the estimated rotor position α is the input to the block wrap. Since there are 8 stator poles, each is 45˚ apart. Therefore, the difference between the aligned position of one phase and its nearest neighbor is 45˚, i.e. the rotor must rotate 45˚ to come into alignment with the next phase. A new angle is generated for each of the phases, which is 45˚ away from the previous phase's angle. Then each of the new angles corresponding to each phases are wrapped such that they fall within the range -30˚ to +30˚. This is done so that the same function g(α) can be used for each phase while still getting the appropriately shifted outputs.

The block diagram of the commutator circuit for one phase is shown in figure 4.14.

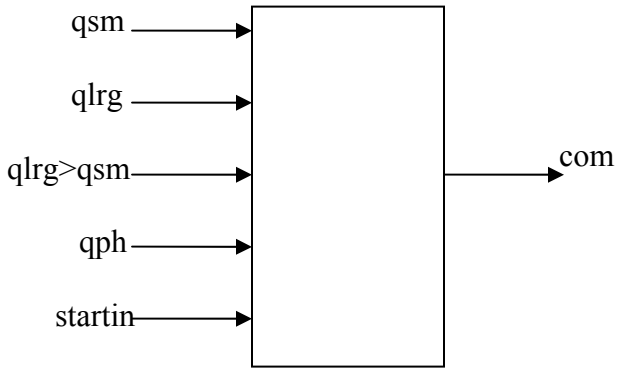

Figure 4.14 Block diagram of commutator circuit for one phase

The code for this circuit is written and instantiated for the four phases. It generates a four bit output.

The Verilog code for the block Sensetheta was written and simulated first. Then the module for the Commutator was added to it and simulated. The Verilog code for the whole circuit is given in the appendix.

The code is implemented on a Xilinx Virtex XCV800 FPGA using the Xilinx EDA tool. Before proceeding to the simulation results, let us look at the results of implementing the complete circuit on the Virtex XCV800 FPGA.

Implementation results

Using Xilinx tools, the coded design is synthesized. After synthesis, the design is translated, mapped, placed and routed onto the selected chip. The results of the implementation are shown below.

Release 6.1.03i Par G.26

Selected Device : v800hq240-4

Number of Slices:	7844	out of 9408	83%
Number of Slice Flip Flops:	10787	out of 18816	57%
Number of 4 input LUTs:	7178	out of 18816	38%
Number of bonded IOBs:	129	out of 170	75%
Number of GCLKs:	1	out of 4	25%

The device utilization summary shows that the available resources are properly utilized.

4.9 Simulation Results

After the implementation of the circuit on the Virtex XCV800 using Xilinx tools, it is simulated using the ModelSim simulator. These simulated results are presented in this section. These simulated results will be compared with the results of the Simulink model in the next chapter.

The result obtained by simulating each of the blocks is presented first and then the final simulated results are discussed. First, the simulation result obtained for the block galpha is discussed.

Figure 4.15 shows the ModelSim simulation results for the block galpha for one value of alpha.

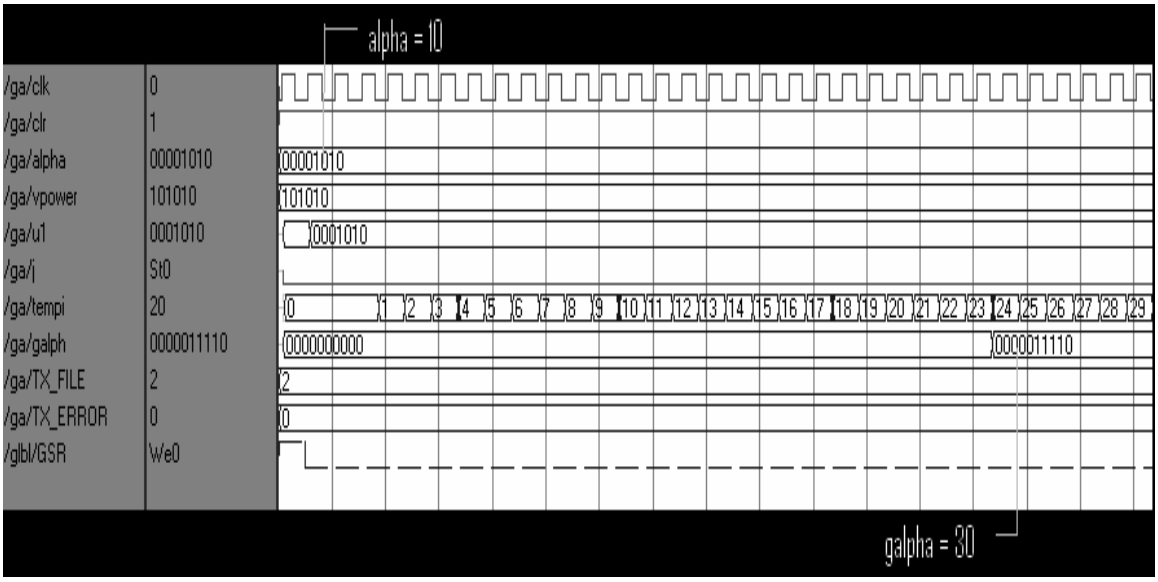

Figure 4.15 ModelSim simulation result for calculating g(α) for a given α.

As seen in the figure, to calculate g(α) for a given α, it takes 24 clock cycles. This is because of the pipelined divider core instantiated into the block galpha to carryout the division operation. Actually the input α has to be in given in radians, but for convenience it is given in degrees. The value of α is given to be 13 degrees (0.226 rad). Therefore, the result is also a scaled value. The actual value would be the result divided by 256.

Next, the error block is simulated and the results are shown in figure 4.15.

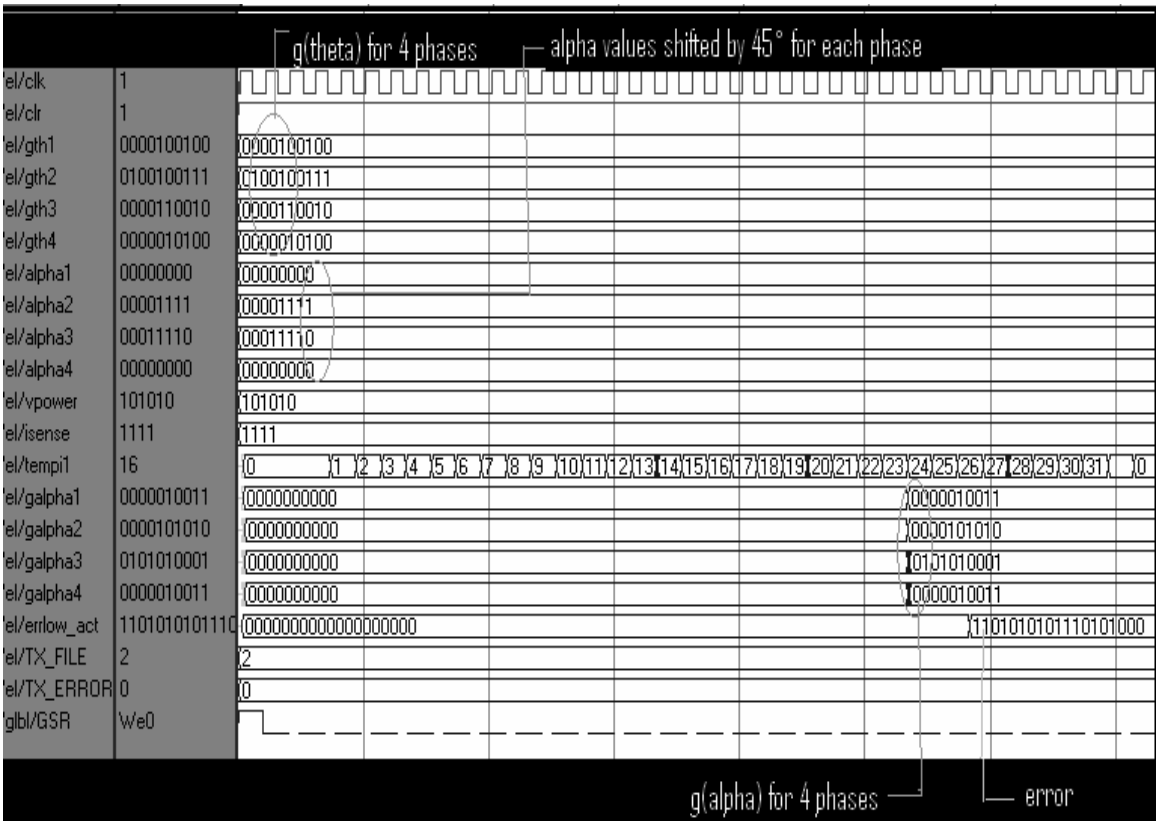

Figure 4.16 ModelSim simulation result for calculating error.

The figure 4.17 shows the simulated result of the block errorlow. The values of g1(θ), g2(θ), g3(θ), g4(θ) are 36, 295, 50,20 respectively. These g(θ) values are given as inputs as shown in the figure. The value of α1=0° is given. α2, α3, α4 are shifted by 45° and wrapped to lie in between -30° and +30°. These values are calculated to be 15, 30 and 0 respectively. As seen in the figure, g1(α), g2(α), g3(α), g4(α) have been calculated to be 19, 42, 337, 19 respectively. Using all these values, the error has been calculated as shown in the figure. The result is a scaled value since the inputs have been multiplied by a scale factor.

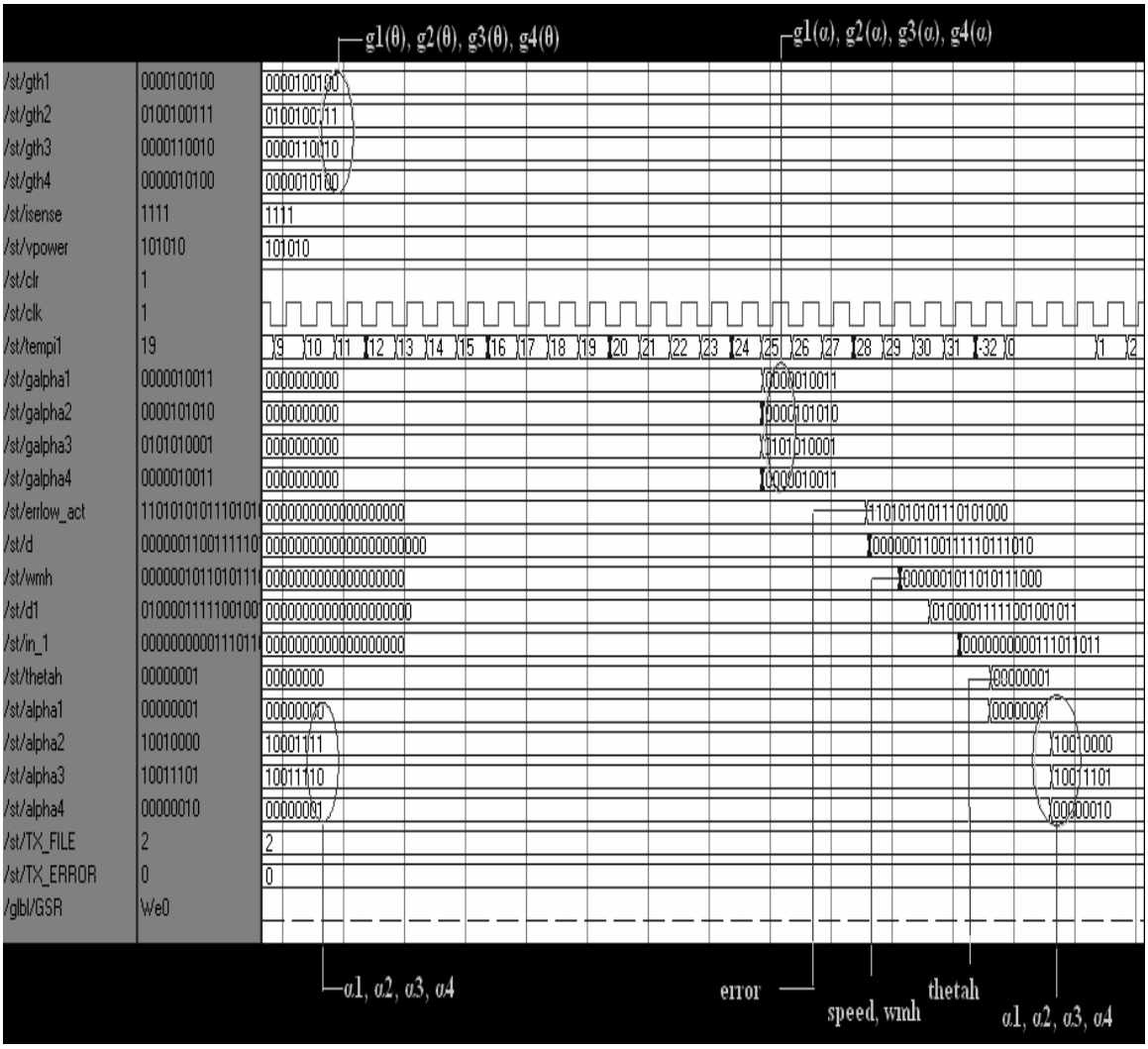

Figure 4.17 Simulation result of the rotor position estimator

Now, consider the simulation result of the block sensetheta. Consider the actual rotor position θ1=13°. The values of θ2, θ3, θ4 are taken as explained earlier. The values of g(θ) for each of the phases are calculated and given as input. The initial value of the estimated rotor position will be taken as α1=0°. Then the shifted and wrapped values of α2, α3, α4 will be 15°, 30°, 0°. For these values of α, g(α) will be calculated for all four phases. Using all these values, the error will be calculated. The calculated error would be given as input to the integrator1, which would give out motor speed as output. This is shown in figure 4.17. The calculated error is added to the motor speed and given as input to integrator2. Integrator2's output is the estimated rotor position, α. This value will be

given as feedback to the circuit as α1. And again, the same process will be repeated until the estimated rotor position converges to the actual rotor position. In this case, until α reaches the value of 13°. Since g(θ) is not changing with time θ is not changing with time and thus the rotor's speed is zero. In turn this means that the final estimated value α of the rotor position will not be changing with time and the final estimated value of the rotor's velocity is zero. This behavior is shown in figure 4.18.

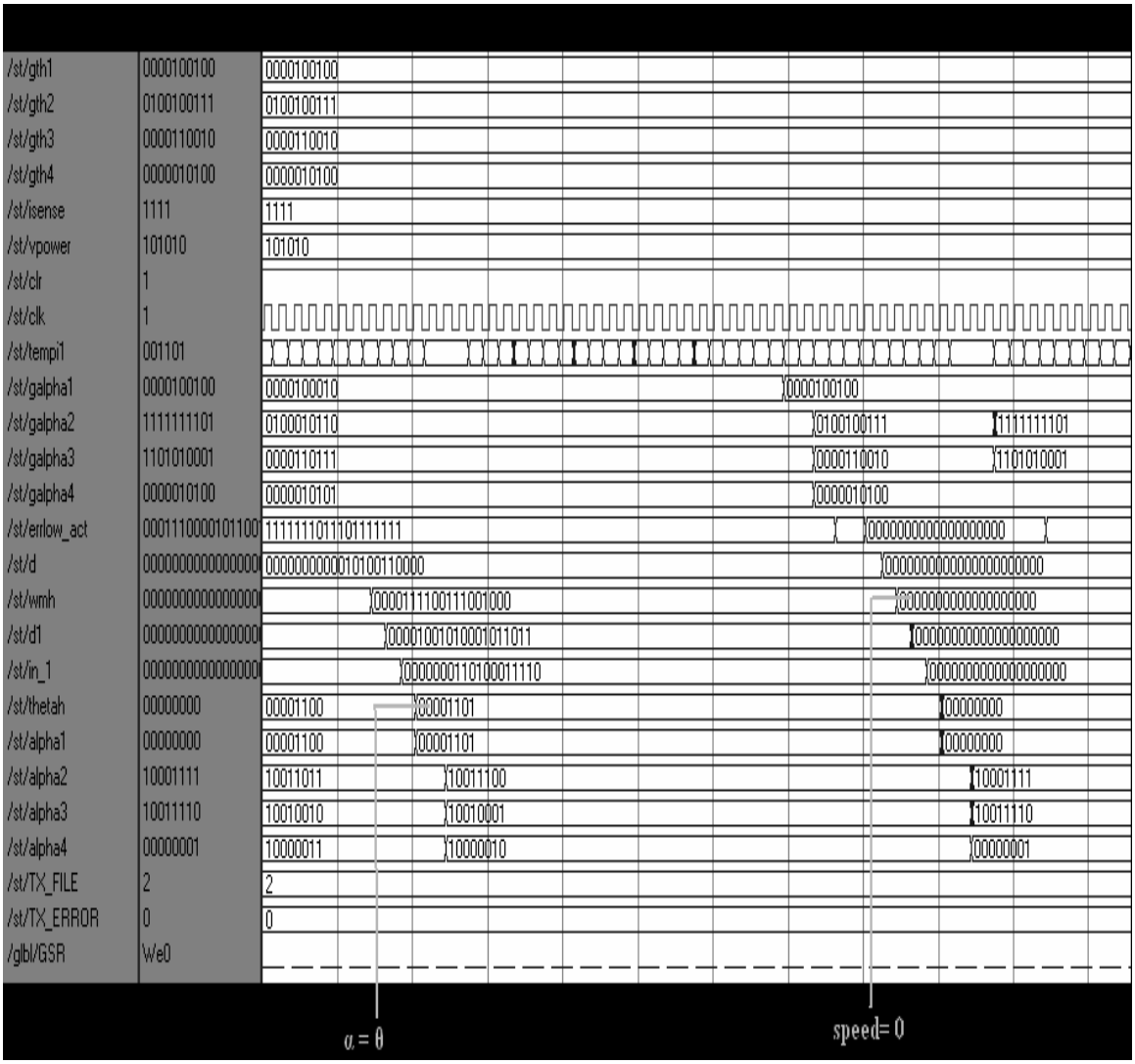

Figure 4.18 Simulation result showing the final output when α becomes equal to θ and speed becomes zero.

CHAPTER 5
TESTING OF THE ESTIMATOR ON AN FPGA

5.1 Comparison of Simulink and Verilog Design Results

The results obtained by simulating the Simulink model and those obtained by simulating the design using ModelSim Simulator are presented here for comparison. The estimated rotor position information is plotted using Matlab. For different values of the input $g(\theta) = [g_1(\theta)\ g_2(\theta)\ g_3(\theta)\ g_4(\theta)]$, the Simulink model is simulated. The design created in Verilog is also simulated for the same values of $g(\theta)$ and the predicted estimate of the rotor position is plotted.

Consider the rotor position to be fixed at some angle. Now this position of the rotor has to be estimated since the rotor position cannot be measured directly. Knowing the inverse inductance value $g(\theta)$ for each of the phases at the actual rotor position, the state estimator can be used to estimate the position of the rotor. The simulation starts with the estimated rotor position α equal to zero since the rotor position is not known. The Simulink model is simulated and its results are compared to the post synthesis simulated Verilog design. The value of the estimated rotor position, when the error defined in previous chapters becomes zero, must be equal to the actual rotor position.
Figure 5.1 shows the simulated result obtained by simulating the Simulink model for $\theta=18°$. Figure 5.2 shows the simulated result obtained from the post synthesis simulation of the Verilog design for $\theta=18°$ using the Modelsim Simulator.

For $\theta=18°$, the values of $g1(\theta)$, $g2(\theta)$, $g3(\theta)$, $g4(\theta)$ are calculated and given as input to the Simulink model and the Verilog design model. While simulating the post synthesis Verilog design model using Modelsim Simulator, it is observed that the time taken to calculate the error for the actual and estimated values of the inverse inductance and to obtain the estimated rotor position value from it is 7μs, which is the sample time. The

Simulink model is simulated for the same sample time and then the results obtained by both the Simulink model and the Verilog design model are compared.

This estimated position value, α is given as feedback and the same process repeats until the error decays to zero. The estimated position value when the error equals zero is the actual rotor position. As seen in the figure, the process repeats after every 7μs and the total time taken for the estimated value to become equal to the actual value is about 3.5ms.

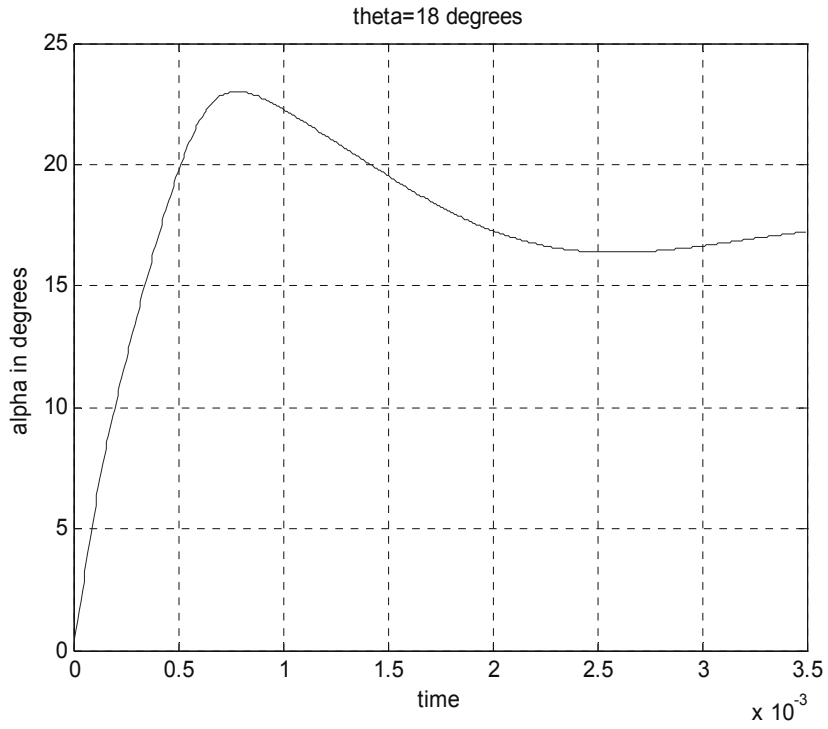

Figure 5.1 Simulated estimated rotor position transient obtained from the Simulink model for θ=18°.

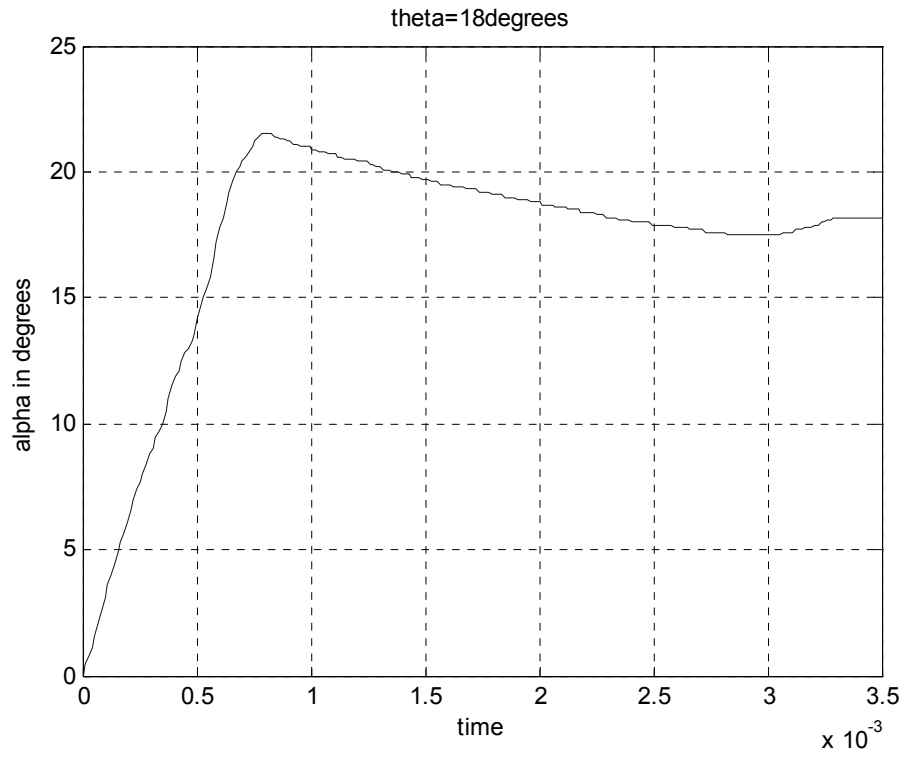

Figure 5.2 Simulated estimated rotor position transient obtained from the post synthesis Verilog model for θ=18°.

As seen in the figures 5.1 and 5.2, the post synthesis simulation result almost complies with the Simulink simulation result. Since the values of g(θ) are fractional numbers, multiplication factors are used so that they can be represented in digital form as Verilog deals with only binary integer numbers. Similarly, the calculated error is also in the range of -0.9 to +0.6, hence multiplication factors are used to represent the error in binary integer form. Many intermediate signals like the inputs and outputs of the integrators are also approximated to the nearest integer. Because of these approximations, there is a slight difference in the post synthesis Verilog simulation result and the Simulink simulated result, though they follow similar trajectories and the time taken to obtain the estimated rotor position is the same.

The figures 5.3 and 5.4 show the calculated error plotted in Matlab for the Simulink model and the experimental result obtained by simulating the Verilog design respectively. As the estimated rotor position value α reaches the actual rotor position value θ, the error has to decay to zero. It is observed that the time at which error becomes zero as seen in figure 5.4 is the time when α becomes equal to θ as seen in figure 5.3. As mentioned earlier, since the process repeats every 7μs, the error is calculated every 7μs. Thus, the error is plotted versus time for every 7μs.

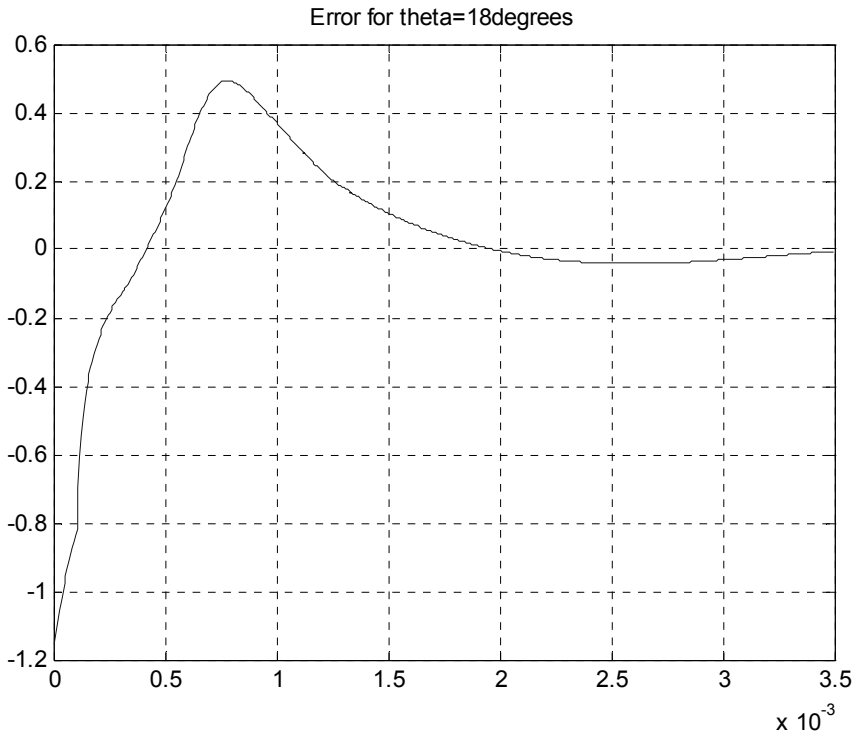

Figure 5.3 Simulated result of the calculated error for the Simulink model for θ=18°.

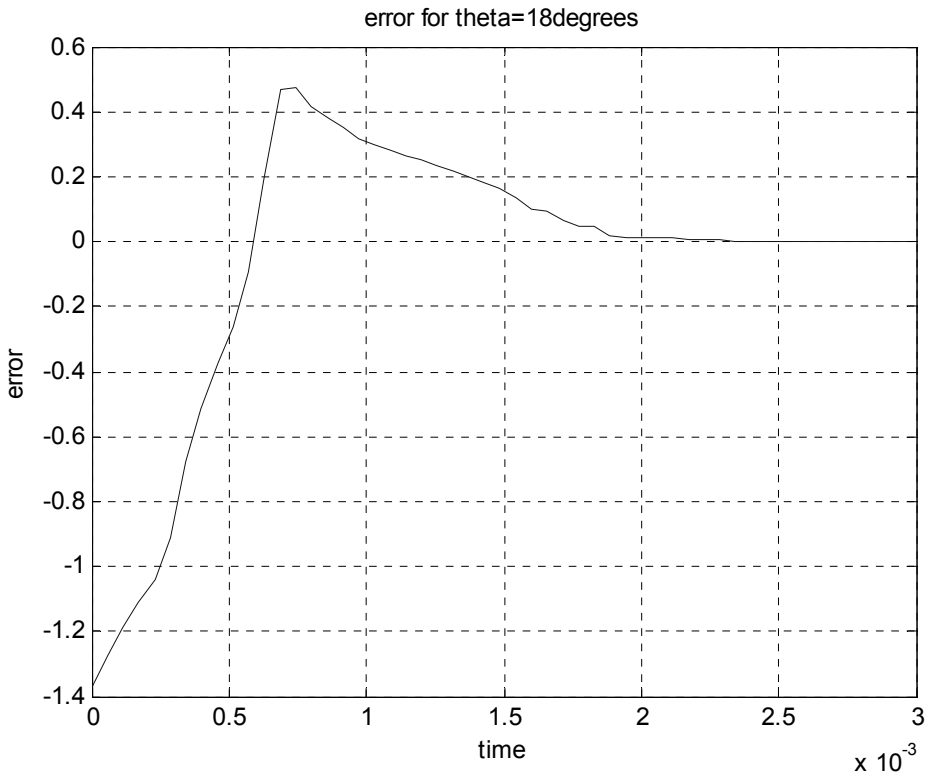

Figure 5.4 Simulated error transient obtained from the post synthesis Verilog model for θ=18°.

Another set of results for θ=13° is presented in figures 5.5 and 5.6. The input values of g1(θ), g2(θ), g3(θ), g4(θ) are given as input to both the Simulink model and the post synthesis Verilog model and the circuit is simulated. The values obtained for the estimated rotor position and error is plotted using Matlab for both sets of results. Figure 5.5 shows the simulated result obtained from the Simulink model.

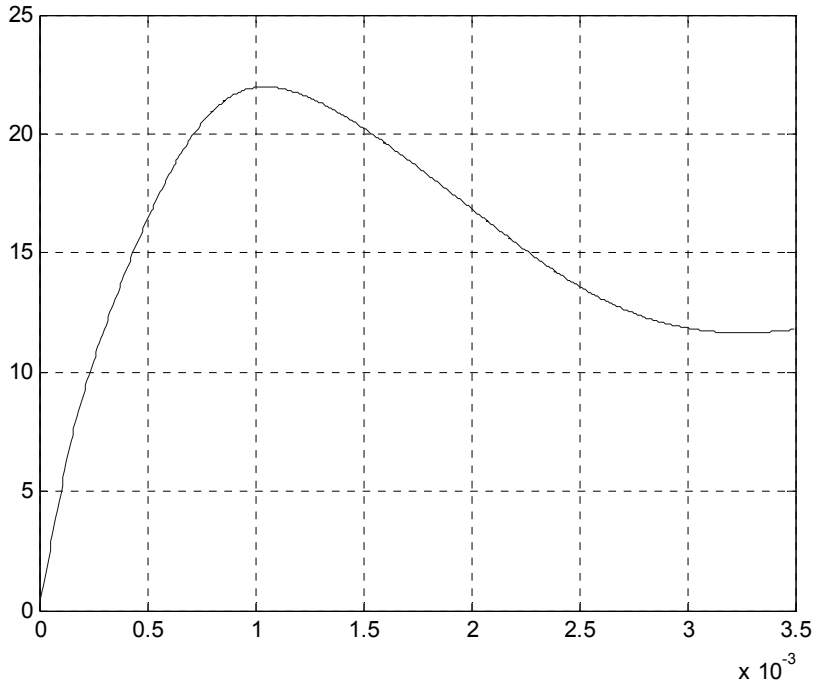

Figure 5.5 Simulated estimated rotor position transient obtained from the Simulink model for θ=13°.

Figure 5.6 shows the simulated result obtained by simulating the post synthesis Verilog design with an actual fixed rotor position equal to θ=13°. The Verilog result matches the general shape and duration of the result obtained from the Simulink model.

As the circuit is simulated, the estimated rotor position starts from its in error initial value of zero and varies until it becomes equal to the actual rotor position, at which time the error becomes zero.

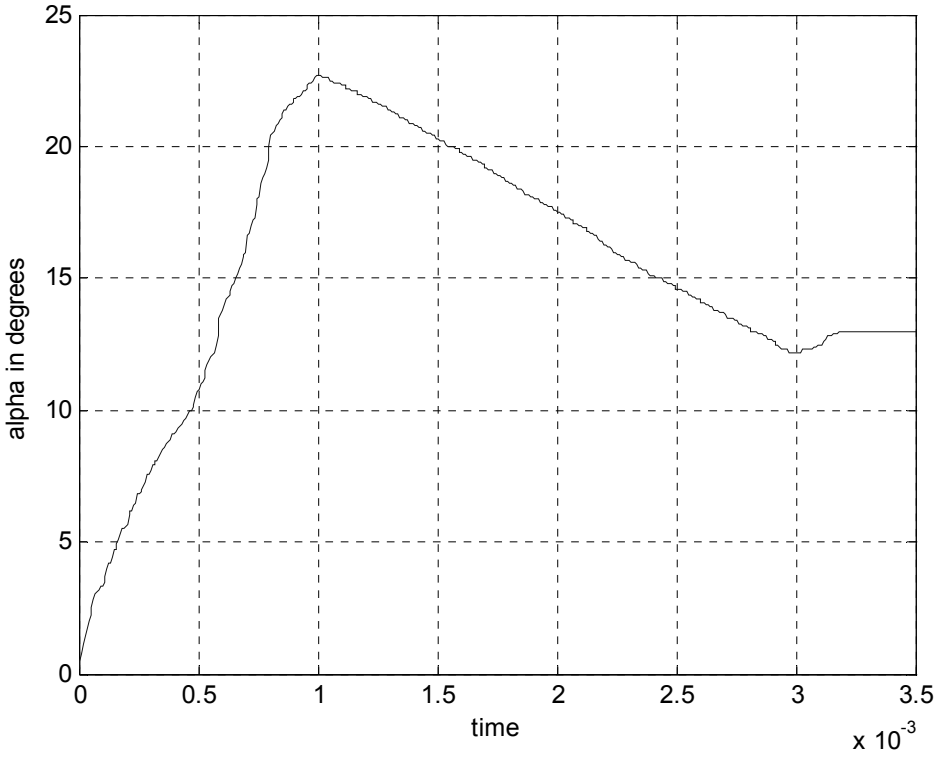

Figure 5.6 Simulated estimated rotor position transient obtained from the post synthesis Verilog model for θ=13°.

Figure 5.7 shows the simulated error obtained from the Simulink model when θ=13°. From figures 5.5 and 5.7, it is observed that the error decays to zero as the estimated rotor position reaches the actual value. Figure 5.11 shows the simulated estimated rotor position transient obtained from the post synthesis Verilog model for θ=13°.

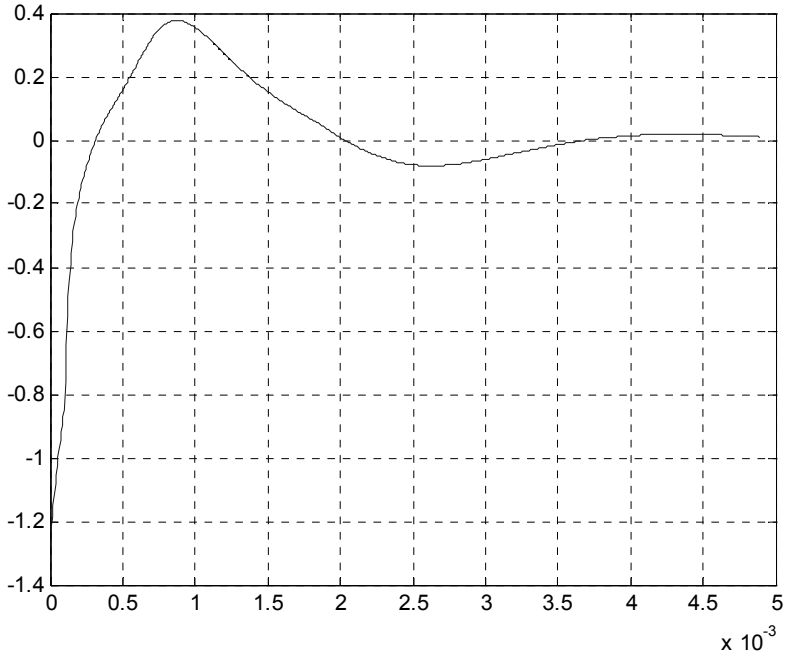

Figure 5.7 Simulated error transient obtained from the Simulink model for θ=13°.

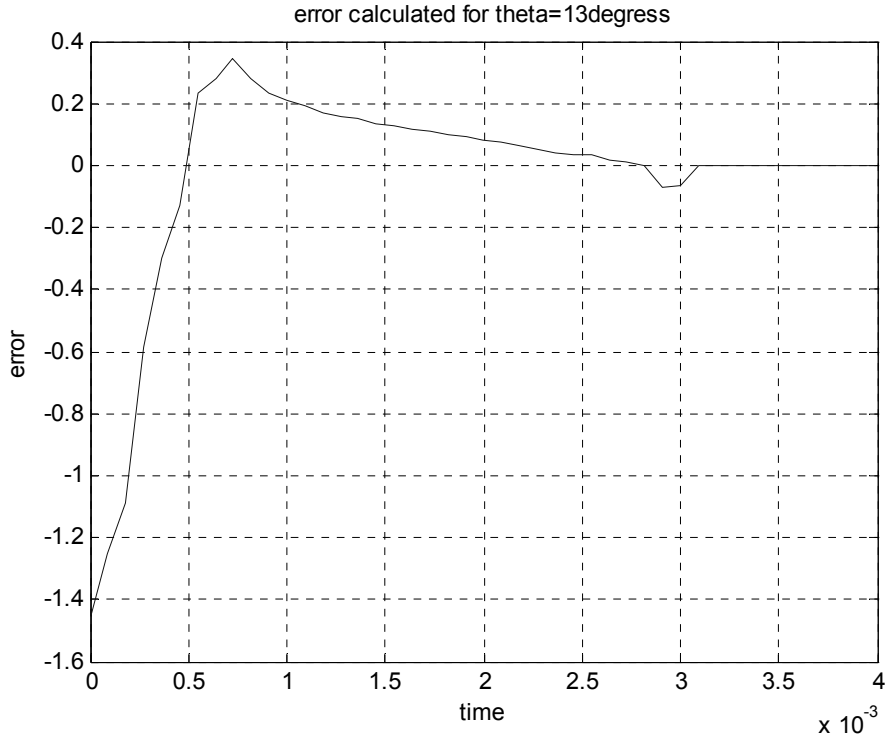

Figure 5.8 Simulated error transient obtained from the post synthesis Verilog model for θ=13°.

5.2 Testing the FPGA Circuit

The code written in Verilog must be converted to a bit stream before it can be downloading into the FPGA. After downloading the bits, the system can be tested by giving the inputs as required. Here, the system is tested using a Xilinx XSV board, which contains the Virtex XCV800 FPGA.

XSV Board

The XSV Board brings the power of the XILINX Virtex FPGA embedded in a framework for processing video and audio signals. The XSV Board can take a single Virtex FPGA from 50K to 800K gates in size.[8]

The XSV Board has a variety of interfaces for communicating with the outside world: parallel and serial ports, Xchecker cable, a USB port, PS/2 mouse and keyboard port and 10/100 Ethernet PHY layer interface. There are also two independent expansion ports, each with 38 general-purpose I/O pins connected directly to the Virtex FPGA.

XSV Board Features

The XSV Board includes many resources, but the ones used in this thesis to implement the position estimator circuit are presented here.

- Programmable logic chips:

  XILINX Virtex FPGA: Virtex FPGAs from 57 Kgates (XCV50) up to 888 Kgates (XCV800) in a 240-pin PQFP or HQFP package is compatible with the XSV Board. The Virtex FPGA is the main repository of programmable logic on the XSV Board. XCV800 has been used in this thesis.

XILINX XC95108 CPLD: The CPLD is used to manage the configuration of the Virtex FPGA via the parallel port, serial port, or Flash RAM. The CPLD also controls the configuration of the Ethernet PHY chip.

- Programmable oscillator that provides a clock signal to the FPGA and CPLD derived form a 100 MHz base frequency.

- Two expansion headers interface the FPGA to external circuitry through 76 general- purpose I/Os.

- Four pushbuttons and one eight-position DIP switch provide general-purpose inputs to the FPGA and CPLD.

- Two LED digits and one LED bargraph let the FPGA and CPLD display status information.

- Parallel/serial port interfaces let the CPLD send and receive data in a parallel or serial format similar to a PC.

- ATX power connector or 9 VDC power jack lets the XSV Board receive power from a standard ATX power supply or a 9 VDC power supply.

The locations of these resources are indicated in the simplified view of the XSV Board shown in appendix. Each of these resources will be described in the following section.

Setting the XSV Board Clock Oscillator Frequency

The XSV Board has a 100 MHz programmable oscillator. The 100 MHz master frequency can be divided by factors of 1, 2, ... up to 2052 to get clock frequencies of 100 MHz, 50 MHz, ... down to 48.7 KHz, respectively. The divided frequency is sent to the rest of the XSV Board circuitry as a clock signal. The divisor is stored in non-volatile

memory in the oscillator chip so it will resume operation at its programmed frequency whenever power is applied to the XSV Board.

Programming the Interface

The Virtex FPGA is the main repository of programmable logic on the XSV Board. The CPLD manages the configuration of the FPGA via the parallel port or from the Flash memory. Therefore, the CPLD must be configured so that it implements the necessary interface. The CPLD stores its configuration in its internal non-volatile memory so the interface is restored each time power is applied to the XSV Board.

Downloading Virtex configuration bits

Once the CPLD is programmed with the downloading interface circuit, you can download bit streams into the Virtex FPGA.

Assigning Inputs and Outputs

Each of the input and output bits of the design are assigned to the pins of the expansion headers. The inputs can also be assigned to the Pushbuttons and Dip switch. The output bits can be assigned to the LEDs or the LED bargraph. This is done using the Xilinx tools where the input/ output bits can be assigned to any of the desired pins. After downloading the program into the FPGA, the assigned pins can be checked to see if the result on it is a high or a low.

5.3 Testing the system

The control system for the SRM drive is shown in figure 5.9. To test the estimator and commutator design without the rest of the system, the inputs and outputs must be generated. This is done as summarized below.

1. The input to the estimator, $g_i(\theta)$ which is the digitized output of the low pass filter, is generated in the program itself. The system is tested for different inputs of $g_i(\theta)$.
2. The inputs to the commutator, on/off angles, which has to be given from the microprocessor of the motor control system, are generated in the program itself.
3. The output of the commutator has to be given to the current regulator.
4. The estimated rotor position which does not come out of the FPGA in figure 5.9 was brought out and measured with a logic analyzer.

Figure 5.10 shows the experimental setup used for testing the circuit.

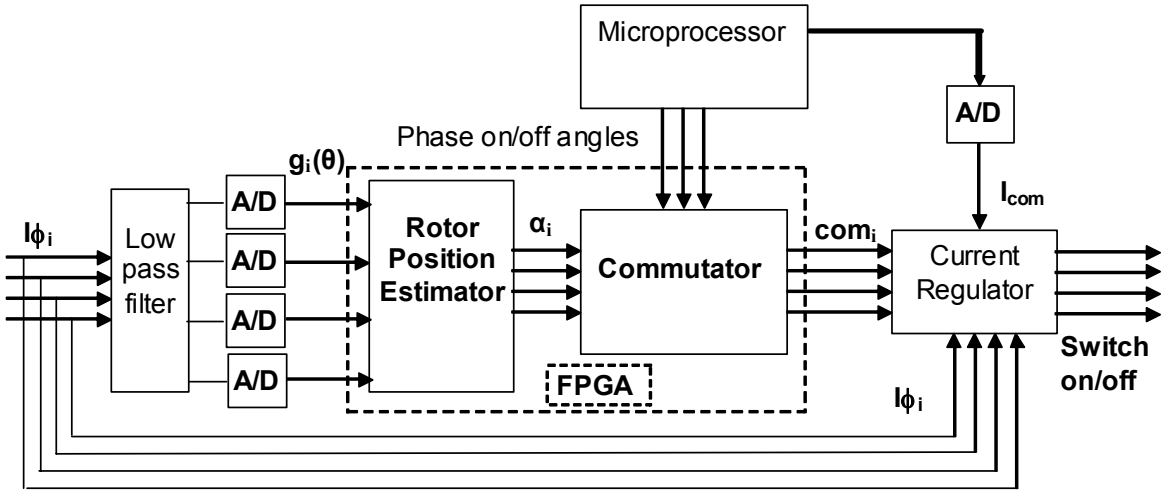

Figure 5.9 Block diagram of the SRM Control system

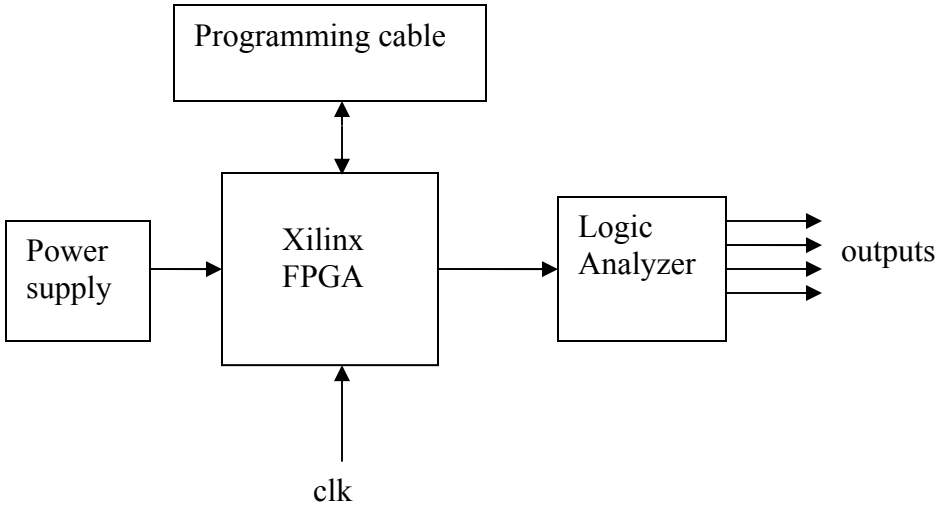

Figure 5.10 Experimental setup for testing the design

The steps involved in programming the FPGA and testing it are given below.

1. The power is given to the FPGA using the 9 VDC power supply.
2. The frequency of the clock to the FPGA is set by setting the oscillator frequency on the XSV board. Here the frequency is set to 5MHz.
3. After setting the frequency, the CPLD is configured so that it acts as an interface between the parallel port and the FPGA. Then the FPGA is ready to be programmed.
4. Before generating the bit stream, the inputs and outputs should be assigned to the pins on the FPGA. This is done by generating a ucf file using the Xilinx tools. Actually this assigns some pins to the inputs and outputs. But the pin assignments can be changed by editing the constraints file and assigning the pins as needed.
5. After the pins are assigned, the bit stream is generated from the Verilog code using the Xilinx tools. These bits are downloaded into the FPGA through the parallel port. Since the FPGA is volatile it has to be programmed each time the power is turned on. Once the code is downloaded onto an FPGA, the testing can be done.

6. The results are captured using a Logic Analyzer. The logic analyzer channels are connected to the pins of the FPGA. The results on these pins are captured in the logic analyzer and can be viewed by connecting the logic analyzer to a PC.

5.4 FPGA implementation results

A test program was written to evaluate various sub-blocks of the Verilog program within the FPGA. The bit file for the block galpha where the estimated rotor position α is given as input and the output is g(α) is programmed into the FPGA. For different values of α, the output is checked. The results captured by the logic analyzer are shown in the figure 5.11 for α=15°. As shown in the figure 5.11, the FPGA output g(α)=42 after 24 clock cycles. The values are scaled values and not the exact values obtained from the Simulink model. Converting the scaled values showed that they were equal to the output from the Simulink model.

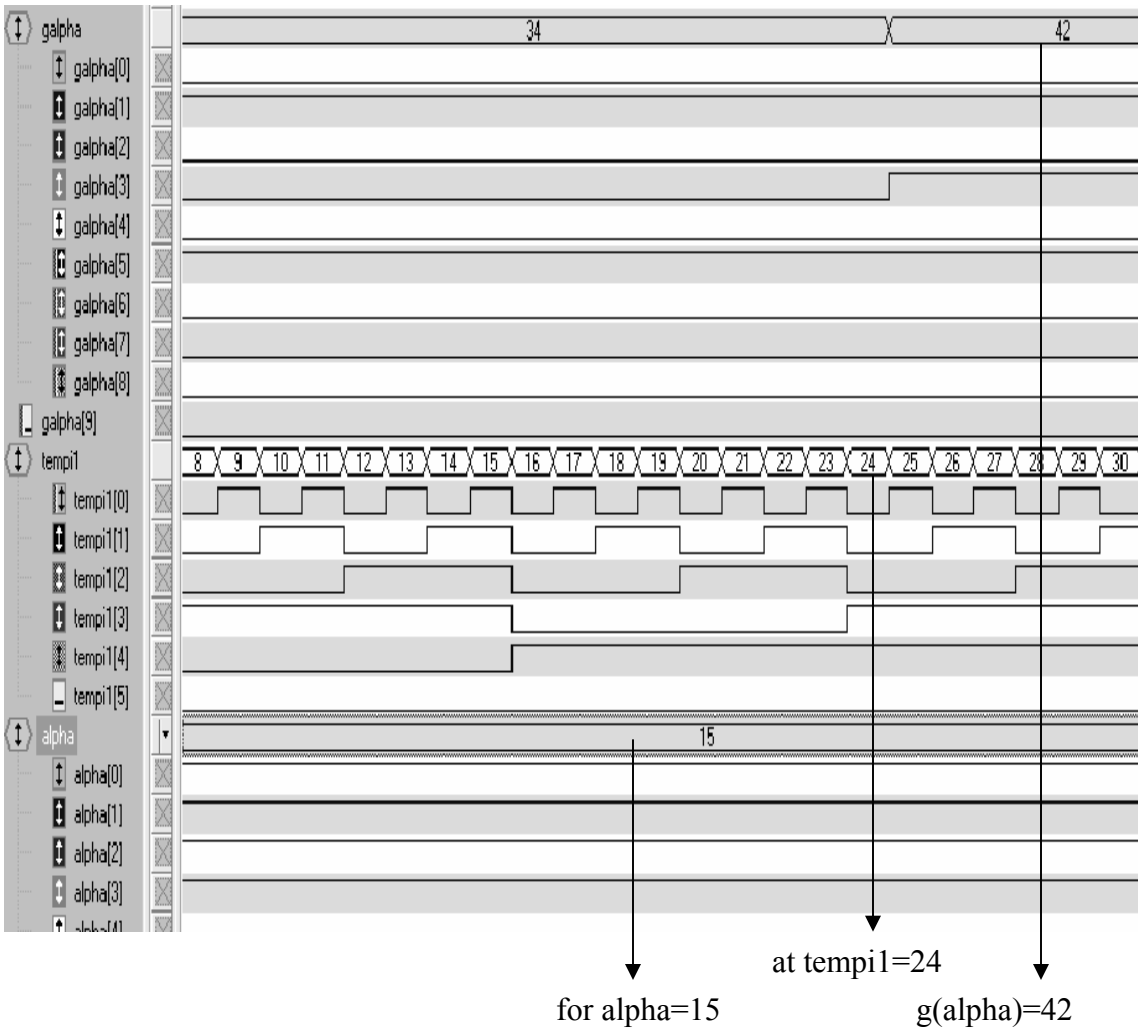

Figure 5.11 Experimental FPGA output for the block galpha

The error block was tested with the following inputs:

$g1(\theta)=36$; $g2(\theta)=295$; $g3(\theta)=50$; $g4(\theta)=20$;

$\alpha1=10$; $\alpha2=-35$; $\alpha3=-80$; $\alpha4=-135$

The error is calculated using equation 2.5. The error block output is shown in figure 5.12 and is again found to be the correct scaled value.

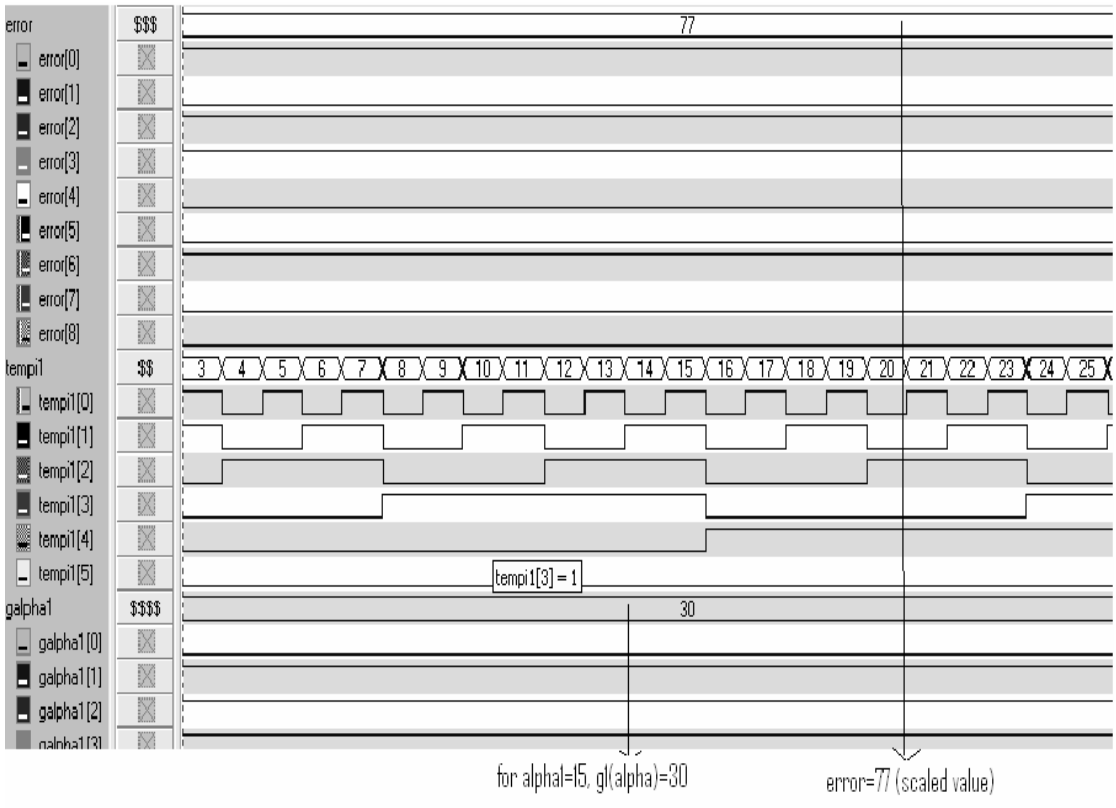

Figure 5.12 Experimental FPGA output for the block errorlow

The Commutator block was also tested separately on the FPGA. The following inputs have been given within the program while the input startin was assigned to the dipswitch. The output com, which is a 4-bit output, is assigned to the pins of the LED bargraph. Then the bit file is downloaded into the FPGA.

qcomin =17'b10001010000000010

qphin=8'b00001010

θph1=13

As seen from the inputs θsm=2, θlrg=20, θlrg>θsm=1. For the θphi values of the four phases, the output has to be generated as shown in table 5.1

θphi	θph> θsm	θph<θlrg	θlrg>θsm	com
13	1	1	1	1
-28	0	0	1	0
-17	0	1	1	0
2	1	0	1	0

Table 5.1 Commutator block output for the four phases

The bargraph shows the correct output 1000.

The position estimator block was also tested separately in the FPGA. For different input values of gi(θ) the circuit was tested.

In the first case the inputs are g1(θ)=36; g2(θ)=295; g3(θ)=50; g4(θ)=20;

The bit file generated for the block sensetheta is downloaded into the FPGA. The output values were captured using a logic analyzer and plotted using Matlab. Figure 5.13 shows the plotted values of the rotor position for the given input. As in the simulations the initial estimated rotor position output has the incorrect value of 0º and the estimated rotor ultimately converges to the correct value.

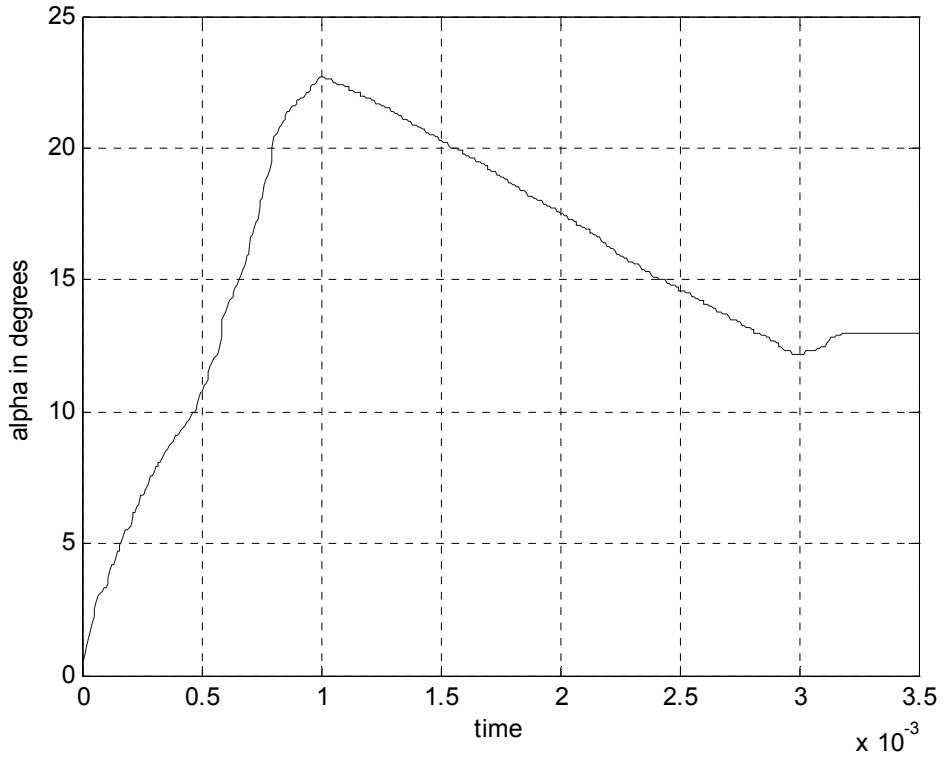

Figure 5.13 Experimentally measured FPGA output for an actual rotor position equal to θ=13º

The result in figure 5.13 matches well with the result in figure 5.5, which shows the simulation result of the Simulink model for θ=13º and still better with the post synthesis results in figure 5.6. The error output given by the FPGA is plotted and shown in figure 5.14.

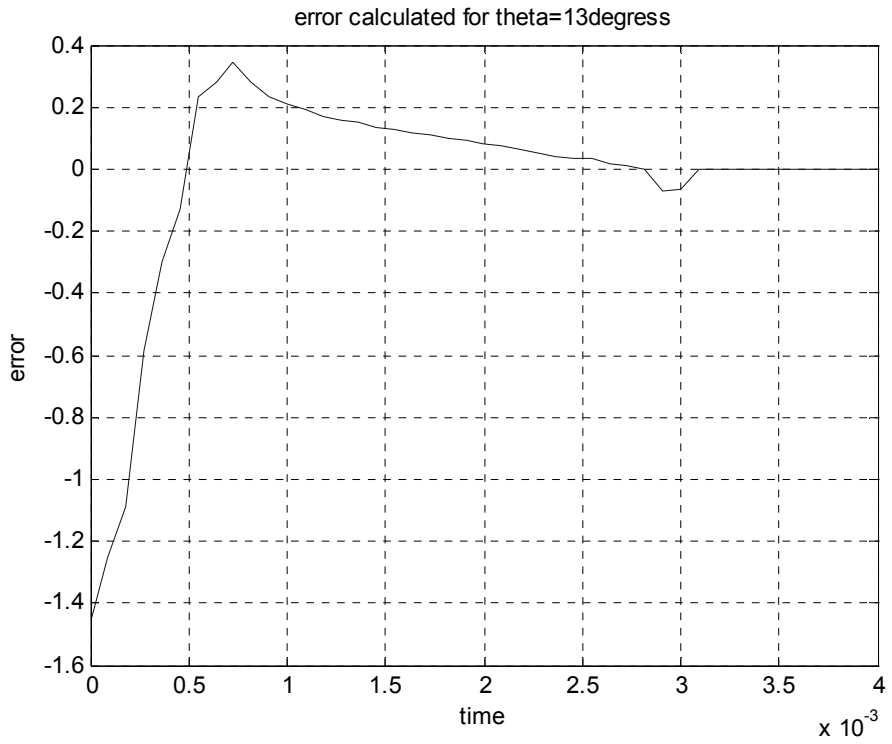

Figure 5.14 Experimentally measured FPGA result for the error for θ=13°.

Figure 5.15 shows the output of the FPGA captured by the Logic analyzer. The waveforms show the various outputs during the part of the transient where α becomes equal to 13º and thus equal to θ. The results in figure 5.15 also show that the error becomes zero at this point.

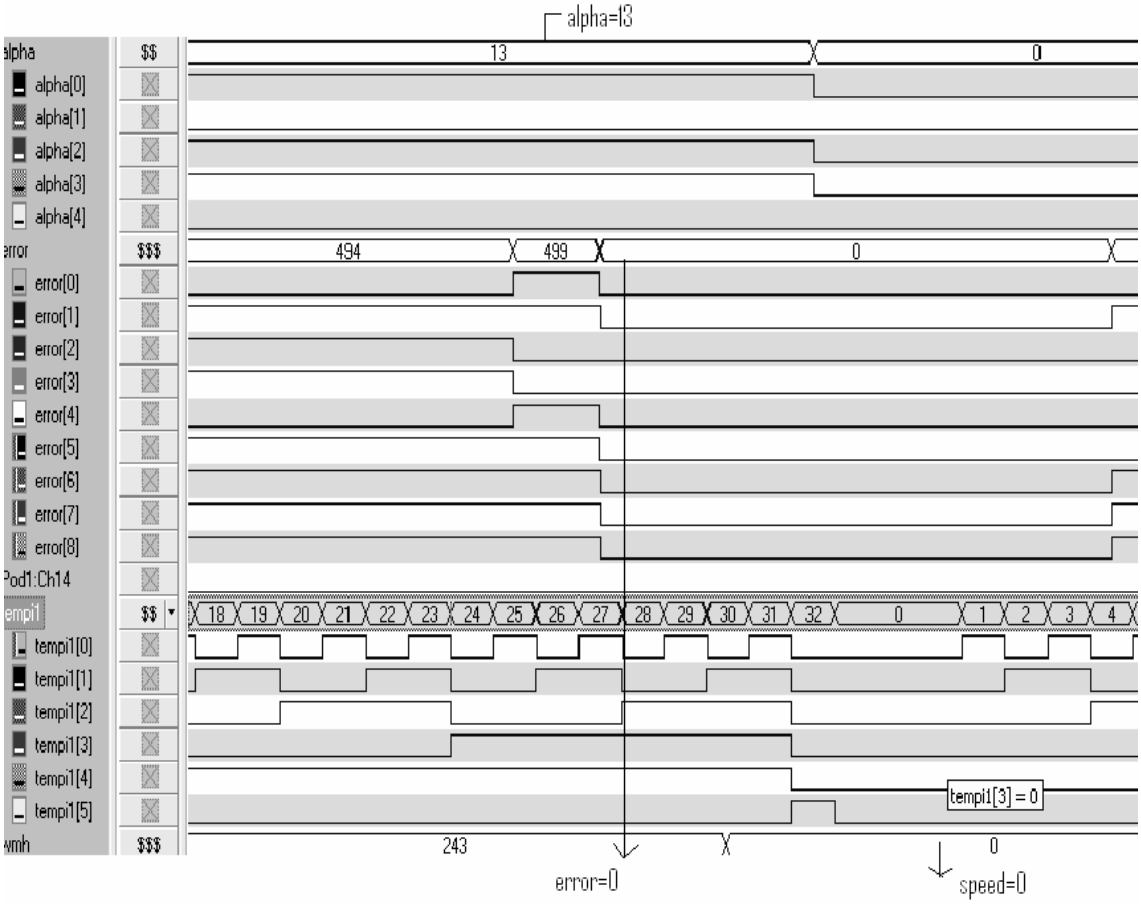

Figure 5.15 Waveform showing the FPGA output for the block sensetheta

Results similar to those in figures 5.11 through 5.15 were obtained for different inputs and found to match with the results given by the Simulink model in all cases. The output of the Position Estimator is given to the Commutator circuit and also tested on the FPGA. This was also found to be working correctly.

5.5 Conclusion

A Rotor Position Estimator for a Switched Reluctance Motor that had been developed previously is successfully implemented using Verilog and programmed into an FPGA. The rotor position estimator developed previously was available as a Simulink model and a FPGA design is created from it. The Verilog HDL design is validated by implementing the code on an FPGA and experimentally testing it. The experimental results obtained are compared with the Simulink model results and are found to match.

The results of the position estimator implementation on an FPGA can be compared with the position estimator implementation on a Signal Processor. It is found that the FPGA rotor position estimator with a 5MHz clock frequency can update its rotor position estimate every 7.0μs compared to an update time of 50μs for a TMS320C6701-150 DSP implementation using a commercial DSP board. This is approximately a 7 to one reduction in the update time.

Appendix A : Verilog Code for Position Estimator and Commutator

```
//Verilog Code for Position Estimator

module ST_Comm(clk,thetah,errlow_act,tempi1,com);

reg [9:0]gth1,gth2,gth3,gth4;
reg [3:0]isense;
reg [5:0]vpower;
input clk;
reg clr;
output [7:0]thetah;
reg [7:0]thetah,qphin;
wire [7:0]u1,u2,u3,u4;
wire [18:0]wmh;
output [9:0]errlow_act;
reg [9:0]errlow_act;
wire [18:0]errlow;
reg [18:0]error;

reg [2:0]loop;
reg [9:0]H1;
reg [24:0]H2;
reg Gimod,sign,s;
wire j1,j2,j3,j4,y;
output [5:0]tempi1;
wire [18:0]in_1;
reg [18:0]tin_1,ta;
reg [24:0]d;
reg [24:0]d1;
reg [7:0]qa;
reg [7:0]qb,qc,qd;

reg [7:0]alpha1,alpha2,alpha3,alpha4;
wire [7:0]talpha2,talpha3,talpha4;
reg [15:0]x;

wire [9:0]gtheta1,gtheta2,gtheta3,gtheta4;

reg [16:0]qcomin;
reg startin;
output [3:0]com;

EL_NEW  errlw(clk,clr,gth1,gth2,gth3,gth4,alpha1,alpha2,alpha3,alpha4,vpower,isense,
```

```
tempi1,galpha1,galpha2,galpha3,galpha4,errlow);
intH2_NEW nit(wmh,d,clk,tempi1);

intH1_NEW nittt(in_1,d1,clk,tempi1,sign);

wrap phb(clk,qb,talpha2);
wrap phc(clk,qc,talpha3);
wrap phd(clk,qd,talpha4);

comm cc(com,qcomin,qphin,startin,clk);

always @(posedge clk)
begin
 x=x+1;
          gth1=36;
          gth2=295;
          gth3=50;
          gth4=20;
          isense=15;
          vpower=42;
          H2=16000000;
          H1=8000;
          Gimod=1;
          clr=1'b1;
          s=1'b0;

if(x<12230)
 begin
          errlow_act=error/1024;
          if (s==1'b1) errlow_act[9]=1'b1;
          else errlow_act[9]=1'b0;

          if(errlow[18]==1'b0)
            error=errlow;
          else
          begin
           error=~(errlow)+1;
            s=1'b1;
          end


          if (tempi1==28)
                    d=(error*62500)/1024;
          if(tempi1==30)
            if(x<10000)
                    d1=(error*8000)/256;
```

```
            if(x>10000)
                  d1=(error*8000)/128;
            if(x>4000)
              sign=1'b1;
                  else if(x<4000) sign=1'b0;


if(x<4655)
begin
 if(in_1<1300)
 begin
        tin_1=(in_1*10)/128;
        ta=in_1/128;
 end
 else if(in_1>1300)
 begin
   tin_1=(in_1*10)/64;
   ta=in_1/64;
 end
end
else if(x>4655)
begin
 tin_1=(in_1*10)/64;
 ta=in_1/64;
 end

        qa=ta;
        qb=qa-45;
        qc=qb-45;
        qd=qc-45;

        if(qb>128)
        begin
                qb[6:0]=~(qb)+1;
                qb[7]=1'b1;
        end
        if(qc>128)
        begin
                qc[6:0]=~(qc)+1;
                qc[7]=1'b1;
        end
        if(qd>128)
        begin
                qd[6:0]=~(qd)+1;
                qd[7]=1'b1;
        end
```

```
          alpha1=qa;
          alpha2=talpha2;
          alpha3=talpha3;
         alpha4=talpha4;

   thetah=tin_1;

     if(x==12220)
   begin
   qphin=13;
   qcomin=17'b10001010000000100;
          startin=1'b0;
   end
    if(x==12230)
        begin
          x=0;
          loop=loop+1;
          alpha1=0;
          alpha2=0;
          alpha3=0;
          alpha4=0;
        end
end
end

endmodule

//Verilog module for the error

module
EL_NEW(clk,clr,gth1,gth2,gth3,gth4,alpha1,alpha2,alpha3,alpha4,vpower,isense,tempi1,
galpha1,galpha2,galpha3,galpha4,errlow_act);

input clk,clr;

output [18:0]errlow_act;
reg [18:0]errlow_act;
output [9:0]galpha1,galpha2,galpha3,galpha4;
output [5:0]tempi1;
input [9:0]gth1,gth2,gth3,gth4; //factor=*256
wire [9:0]tgalpha1;
reg [9:0]ttgalpha1;
input [5:0]vpower;
input [3:0]isense;
input [7:0]alpha1,alpha2,alpha3,alpha4;
reg [18:0]temp_errlow;
```

```
reg [18:0]errlow;
wire [7:0]u1,u2,u3,u4;


reg [9:0]gtheta1,gtheta2,gtheta3,gtheta4;
wire j2,j3,j4;
reg [9:0]x;

wire [5:0]tempi2,tempi3,tempi4;
wire [19:0]mout1,mout2,mout3,mout4,mout5,mout6,mout7,mout8;


GA_NEW g1(clk,clr,alpha1,vpower,u1,j1,tempi1,galpha1);
GA_NEW g2(clk,clr,alpha2,vpower,u2,j2,tempi2,galpha2);
GA_NEW g3(clk,clr,alpha3,vpower,u3,j3,tempi3,galpha3);
GA_NEW g4(clk,clr,alpha4,vpower,u4,j4,tempi4,galpha4);

mult1 m1(clk,gtheta2,galpha1,mout1);
mult1 m2(clk,gtheta1,galpha2,mout2);
mult1 m3(clk,gtheta3,galpha2,mout3);
mult1 m4(clk,gtheta2,galpha3,mout4);
mult1 m5(clk,gtheta4,galpha3,mout5);
mult1 m6(clk,gtheta3,galpha4,mout6);
mult1 m7(clk,gtheta1,galpha4,mout7);
mult1 m8(clk,gtheta4,galpha1,mout8);

always @(posedge(clk))
begin

if (isense[0]==1'b1)  gtheta1=gth1;
        else          gtheta1=galpha1;
if (isense[1]==1'b1)  gtheta2=gth2;
        else          gtheta2=galpha2;
if (isense[2]==1'b1)  gtheta3=gth3;
        else          gtheta3=galpha3;
if (isense[3]==1'b1)  gtheta4=gth4;
        else          gtheta4=galpha4;

temp_errlow=((mout1-mout2)+(mout3-mout4)+(mout5-mout6)+(mout7-mout8));


errlow=temp_errlow;
errlow_act=errlow;
end
```

```
endmodule

// verilog module for the block galpha

module GA_NEW(clk,clr,alpha,vpower,u1,j,tempi,galph);
input [7:0]alpha;
output j;
reg j;
input [5:0]vpower;
input clk,clr;

output [9:0]galph;
reg [9:0]galph;
reg [18:0]temp_galph;

reg [18:0]quotF1,quotF2;
wire [18:0]tquotF1,tquotF2;
reg [18:0]Fcn11;
reg [18:0]Fcn12,Fcn22;
output [6:0]u1;
reg [6:0]u1;
reg [7:0]u1_5;
reg [18:0]prod;

reg [5:0]Dmod;
reg [13:0]Fmod;
reg [5:0]Lpideal,mpideal,thetaTm;
reg [7:0]Laideal;
reg [8:0]mideal;

wire [18:0]remF1,remF2;
reg [18:0]tremF1,tremF2;
reg sw1;

reg [5:0]i;
output [5:0]tempi;
reg [5:0]tempi;
reg k;
reg l;

divtemp ddd(Fcn11,Fcn12,tquotF1,remF1,clk);
divtemp dddd(Fcn11,Fcn22,tquotF2,remF2,clk);
always @(posedge(clk))
begin
```

```
 u1=alpha;


Dmod=40;
 Fmod=10000;

 Laideal=8'b10001001;     // 0.0084*16384  (137)
 mideal=9'b100101100;          // 0.0183*16384 (300)

 Lpideal=6'b001100;   // 0.00075*16384 (12)
 mpideal=6'b100010;  // 0.0021*16384 (34)
 thetaTm=6'b011000;  // 0.4189*180/3.14 (24deg)


 if(j==1'b1)
          l=1'b1;
else
          l=1'b0;
 end
  always @(negedge(clk))
begin

          if(clr==0)
          begin
                    i=0;
          end
          else if(clr==1 && l==1'b1)
          begin
                    i=0;
          end
          else
                     i=i+1;
          if(i==3)
          begin
           tempi=i-3;
           tempi=tempi+1;
          end
          if(i<3) tempi=0;
          if(i>3) tempi=tempi+1;

          Fcn11=67200;  //(( Dmod * Dmod )* vpower);


           u1_5=5*u1;
           if(Laideal>(u1_5))
```

```
        Fcn12=(Laideal-(u1_5));                  // (mideal*pi/180)=5 (deg)
                                                 //factor=(100*100*Fmod/16384)
 else Fcn12=~(Laideal-(u1_5))+1;


prod=6*(u1-thetaTm);                             // mpideal*10*pi/180=6
if(120>prod)   Fcn22=(120-prod);                 //factor=(10*100*100*Fmod/16384)
else Fcn22=~(120-prod)+1 ;                       //10*Lpideal=120

if(u1<=thetaTm) sw1=1'b1;
else sw1=1'b0;

tremF1=remF1;
tremF2=remF2;

if((tremF1*2)< Fcn12) tremF1=1'b0;
        else tremF1=1'b1;
if((tremF2*2)< Fcn22) tremF2=1'b0;
        else tremF2=1'b1;

 if(tempi==25)  k=1'b1;
        if(k==1'b1)
         begin
                quotF1=tquotF1;
                quotF2=tquotF2;

          if(sw1==1'b1)
                begin
                temp_galph=quotF1+tremF1;
                galph=(temp_galph*5)/128

                end
          else
                begin
                temp_galph=quotF2+tremF2;
                galph=(temp_galph*54)/128
                end
end

if (tempi==32)
                j=1'b1;
        else
                j=1'b0;
```

```
end

endmodule

// Verilog Module for Block integrator


module intH1_NEW(vout,vin,clk,tempi1,sign);
output [18:0]vout;
input [24:0]vin;
reg [18:0]vout;
input clk,sign;
input [5:0]tempi1;
reg [18:0]xn,xn1;

initial
begin
  vout=0;
end


always@(posedge clk)
begin

if(sign==1'b0)
  xn1=((vin*60)/(512*1024*8))+ xn;
  else if(sign==1'b1)
   xn1=xn-((vin*60)/(512*1024*8));

if (vin==0) xn1=0;
 vout=xn1;

if(tempi1==30) xn=xn1;

end

endmodule


// Verilog module for the block wrap

module wrap(clk,angle,wa);

input clk;
input [7:0]angle;
output [7:0]wa;
```

```
reg [7:0]wa,ta;

always @(posedge clk)
begin
  if((angle >= 0) & (angle <= 30))
  wa= angle;
  if((angle > 30) & (angle <= 90))
  wa= angle -60;
  if((angle > 90) & (angle <= 127))
  wa= angle -120;

 if((angle > 128) & (angle <= 158))
 begin
 wa[6:0]= angle- 128;
 wa[7]=1'b1;
 end

 if((angle > 158) & (angle <= 218))
 begin
 if (angle <188) wa[6:0]=~( angle- 188)+1;
 else    wa[6:0]=( angle- 188);
 wa[7]=1'b1;
 end

if((angle > 218) & (angle <= 255))
 begin
 if (angle <248) wa[6:0]=~( angle- 248)+1;
 else    wa[6:0]=( angle- 248);
 wa[7]=1'b1;
 end

end
endmodule


// Verilog Module for the block commutator


module comm(com,qcomin,qphin,startin,clk);
input startin,clk;

input [16:0]qcomin;
input [7:0]qphin;
output [3:0]com;
wire [7:0]qwb,qwc,qwd;
reg [7:0]qpha,qphb,qphc,qphd,qa,qb,qc,qd;
```

```
comm1 cm1(qcomin[7:0],qcomin[15:8],qcomin[16],qpha,startin,com[0]);
comm1 cm2(qcomin[7:0],qcomin[15:8],qcomin[16],qphb,startin,com[1]);
comm1 cm3(qcomin[7:0],qcomin[15:8],qcomin[16],qphc,startin,com[2]);
comm1 cm4(qcomin[7:0],qcomin[15:8],qcomin[16],qphd,startin,com[3]);

wrap phb(clk,qb,qwb);
wrap phc(clk,qc,qwc);
wrap phd(clk,qd,qwd);

always @(posedge(clk))
begin

    qa=qphin;
        qb=qa-45;
        qc=qb-45;
        qd=qc-45;

        if(qb>128)
        begin
                qb[6:0]=~(qb)+1;
                qb[7]=1'b1;
    end
        if(qc>128)
        begin
                qc[6:0]=~(qc)+1;
                qc[7]=1'b1;
    end
        if(qd>128)
        begin
                qd[6:0]=~(qd)+1;
                qd[7]=1'b1;
    end
    qpha=qa;
        qphb=qwb;
        qphc=qwc;
        qphd=qwd;

end

endmodule

// verilog module for commutator of one phase

 module comm1(qsm,qlrg,qlrg_qsm,qph,startin,comon);
input [7:0]qsm,qlrg,qph;
```

```
input qlrg_qsm,startin;
output comon;
reg comon;
reg [2:0]mux1;
reg mout;

always @(qph,qsm,qlrg,qlrg_qsm,startin)
begin

if(qph[7]==qlrg[7])
begin
        if(qph<=qlrg) mux1[2]=1'b1;else mux1[2]=1'b0;
end
else if((qph[7]==1'b1) & (qlrg[7]==1'b0)) mux1[2]=1'b1;
    else if((qph[7]==1'b0) & (qlrg[7]==1'b1)) mux1[2]=1'b0;


if(qph[7]==qsm[7])
begin
        if(qph>=qsm) mux1[1]=1'b1;else mux1[1]=1'b0;
end
else if((qph[7]==1'b1) & (qsm[7]==1'b0)) mux1[1]=1'b0;
    else if((qph[7]==1'b0) & (qsm[7]==1'b1)) mux1[1]=1'b1;

if(qlrg_qsm==1'b1) mux1[0]=1'b1;else mux1[0]=1'b0;

        case (mux1)
                3'b000: mout=1'b0;
        3'b001: mout=1'b1;
        3'b010: mout=1'b1;
        3'b011: mout=1'b0;
        3'b100: mout=1'b0;
        3'b101: mout=1'b0;
        3'b110: mout=1'b0;
        3'b111: mout=1'b1;
        endcase

  comon= (~(startin)& mout);

end

endmodule
```

The cores generated for the pipelined divider and multiplier are also instantiated in to the Verilog design.

References

1. Miller, TJE, “Switched Reluctance motors and their control”, Magna Physics Publishing and Clarendon Press, Oxford,1993
2. Krishnan, R., “ Switched Reluctance Motor drives: Modeling, Simulation, Analysis, Design and Applications”, CRC Press, 2001
3. Radun, A. V. "Analytically Calculating the SRM's Unaligned Inductance," IEEE Transactions on Magnetics, Vol. 35, n. 6., pp. 4473-4481 November/December 1999
4. Anwar, M. N., Hussain, I., and Radun, A. V., “A Comprehensive Design Methodology for Switched Reluctance Machines,” IEEE Transactions On Industry Applications, vol. 37, n. 6, pp. 1684-1692, November/December 2001
5. Xilinx, Virtex XCV800, http://direct.xilinx.com/bvdocs/publications/ds003.pdf.
6. Xilinx, Spartan XC3S1000, http://direct.xilinx.com/bvdocs/publications/ds099.pdf.
7. Agilent Logic Wave Analyzer, http://cp.literature.agilent.com/litweb/pdf/5968-5560E.pdf.
8. XSV board, www.xess.com-manuals-xsv-manual-v1_1.mdi.
9. Brown, S and Vranesic, Z.,” Fundamentals of Digital Logic with Verilog Design “, Mcgraw- Hill Science/Engineering/Math, 2002
10. Yalamanchili, Sudhakar, “ Introductory VHDL from Simulation to Synthesis”, Prentice Hall, 2001
11. Ciletti, Michael D., “Modeling, Synthesis and Rapid Prototyping with the Verilog HDL”, Prentice Hall, 1999
12. Hennessy, J and Patterson, D, “ Computer Architecture: A Quantitative Approach”, Morgan Kaufmann, 2003
13. Ciletti, Michael D., “ Starter’s Guide to Verilog 2001”, Prentice Hall, 2003

14. Gallegos-Lopez G, Kjaer PC and Miller TJE [1997] A New Sensorless Method for Switched Reluctance Motor Drives, IEEE-IAS 97, New Orleans, pp.564-570.

15. Perl, T.; Husain, I.; Elbuluk, M.; Design trends and trade-offs for sensorless operation of switched reluctance motor drives. Industry Applications Conference, IEEE, Volume: 1; Pages: 278 - 285 vol.1.

16. Ray, W.F.; Al-Bahadly, I.H.; Sensorless methods for determining the rotor position of switched reluctance motors. Power Electronics and Applications, 1993; Pages: 7 - 13 vol.6.

Vita

Srilaxmi Pampana was born in Hyderabad, India on May 2, 1980. She received her Bachelor of Technology degree from VNR Vignana Jyothi Institute of Engineering and Technology, JNTU, India in 2001. She worked as an intern at Hindustan Cables Limited from January 2001 to June 2001. She did her research work under Dr. Arthur Radun from August 2003 to October 2004.