University of Kentucky
UKnowledge

University of Kentucky Doctoral Dissertations

Graduate School

2005

ELIMINATING THE POSITION SENSOR IN A SWITCHED RELUCTANCE MOTOR DRIVE ACTUATOR APPLICATION

Jinhui Zhang
University of Kentucky

Right click to open a feedback form in a new tab to let us know how this document benefits you.

Recommended Citation

Zhang, Jinhui, "ELIMINATING THE POSITION SENSOR IN A SWITCHED RELUCTANCE MOTOR DRIVE ACTUATOR APPLICATION" (2005). *University of Kentucky Doctoral Dissertations*. 343.
https://uknowledge.uky.edu/gradschool_diss/343

This Dissertation is brought to you for free and open access by the Graduate School at UKnowledge. It has been accepted for inclusion in University of Kentucky Doctoral Dissertations by an authorized administrator of UKnowledge. For more information, please contact UKnowledge@lsv.uky.edu.

ABSTRACT OF DISSERTATION

Jinhui Zhang

The Graduate School

University of Kentucky

2005

ELIMINATING THE POSITION SENSOR IN A SWITCHED RELUCTANCE MOTOR DRIVE ACTUATOR APPLICATION

ABSTRACT OF DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By

Jinhui Zhang
Lexington, Kentucky

Director: Dr. Arthur V. Radun, Professor of Electrical engineering

Lexington, Kentucky

2005

Copyright © Jinhui Zhang 2005

ABSTRACT OF DISSERTATION

ELIMINATING THE POSITION SENSOR IN A SWITCHED RELUCTANCE MOTOR DRIVE ACTUATOR APPLICATION

The switched reluctance motor (SRM) is receiving attention because of its merits: high operating temperature capability, fault tolerance, inherent shoot-through preventing inverter topology, high power density, high speed operation, and small rotor inertia. Rotor position information plays a critical role in the control of the SRM. Conventionally, separate position sensors, are used to obtain this information. Position sensors add complexity and cost to the control system and reduce its reliability and flexibility.

In order to overcome the drawbacks of position sensors, this dissertation proposed and investigated a position sensorless control system that meets the needs of an electric actuator application. It is capable of working from zero to high speeds. In the control system, two different control strategies are proposed, one for low speeds and one for high speeds. Each strategy utilizes a state observer to estimate rotor position and speed and is capable of 4 quadrant operation.

In the low speed strategy a Luenberger observer, which has been named the inductance profile demodulator based observer, is used where a pulse voltage is applied to the SRM's idle phases generating triangle shaped phase currents. The amplitude of the phase current is modulated by the SRM’s inductance. The current is demodulated and combined with the output of a state observer to produce an error input to the observer so that the observer will track the actual SRM rotor position. The strategy can determine the SRM’s rotor position at standstill and low speeds with torques up to rated torque.

Another observer, named the simplified flux model based observer, is used for medium and high speeds. In this case, the flux is computed using the measured current and a simplified flux model. The difference between the computed flux and the measured flux generates an error that is input to the observer so that it will track the actual SRM rotor position. Since the speed ranges of the two control stragegies overlap, the final control system is capable of working from zero to high speed by switching between the two observers according to the estimated speed. The stability and performance of the observers are verified with simulation and experiments.

KEYWORDS: Switched Reluctance Motor, Sensorless Control, Flux model, Real Time Control, Actuator Application

ELIMINATING THE POSITION SENSOR IN A SWITCHED RELUCTANCE MOTOR DRIVE ACTUATOR APPLICATION

By

Jinhui Zhang

Director of Dissertation

Director of Graduate Studies

RULES FOR THE USE OF DISSERTATION

Unpublished dissertations submitted for the Doctor's degree and deposited in the University of Kentucky 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 of summaries of parts may be published only with the permission of the author, and the usual scholarly acknowledgements.

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

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

Name	Date

DISSERTATION

Jinhui Zhang

The Graduate School
University of Kentucky
2005

ELIMINATING THE POSITION SENSOR IN A SWITCHED RELUCTANCE MOTOR DRIVE ACTUATOR APPLICATION

DISSERTATION

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in the
College of Engineering
at the University of Kentucky

By
Jinhui Zhang

Lexington, Kentucky

Director: Dr. Arthur V. Radun, Professor of Electrical engineering

Lexington, Kentucky

2005

Copyright © Jinhui Zhang 2005

To my parents; Zhang, Senyou and Wang, Meifang, my wife, Li, Haoju, my sons, Leonald and Larry, my sisters, Xijing and Jin’ge.

Acknowledgements

I would like to sincerely thank Dr. Arthur Radun for his financial support and academic guidance since the fall semester, 2001. He exemplifies the high quality scholarship to which I aspire. Next, I wish to thank the complete Dissertation Committee, and the outside examiner, respectively, Dr. Jimmie J. Cathey, Dr. J. Robert Heath, Dr. L. S. Stephens, and Dr. Alan T. Male. Each individual provided great advice that guided me to successfully finish this work.

I thank Dr. YuMing Zhang for his support as the director of graduate studies, Qiang Li and Xiaohu Feng for their support in the experiments. I also would like to extend my thanks and gratitude to the staff working in the Department of Electrical and Computer Engineering at the University of Kentucky for providing me with office facilities all through the process of my doctoral studies.

Besides, I would like to thank my parents who have been supporting me from my early age to get good education. I also would like to express my appreciation to my wife for her sacrifice in taking care of my children and giving up the opportunity of Ph.D study. My children are the source of my encouragement.

In addition to the above listed people, I also would like to thank all my friends in the Electrical and Computer Engineering department, the Chinese Students and Scholars Association, and the Lexington Chinese Christian Church for their encouragement, support, and prayers.

Table of contents

Acknowledgements iii
Table of figures vii
Table of tables ix
Chapter 1 Introduction 1
Chapter 2 Background 4
2.1 SRM basics 4
2.1.1 SRM structure 4
2.1.2 SRM flux linkage 5
2.1.3 SRM torque 6
2.1.4 SRM inverter 9
2.1.5 SRM phase current 10
2.2 An analytical model of the SRM 11
2.2.1 The non-overlap case 12
2.2.2 The overlap case 14
2.3 A torque method to obtain the flux of SRM 16
2.3.1 Power losses 17
2.3.1.1 Eddy current loss and hysteresis loss 17
2.3.1.2 Avoiding the loss induced measurement errors 18
2.3.2 Obtaining flux from the static torque 19
2.3.3 Comparing with the classic method 21
2.3.3.1 The Classic method 21
2.3.3.2 The torque method 23
2.3.3.3 Comparison between the torque method and the classic method 24
2.3.4 Applying the torque method to FEA 25
2.3.4.1 The torque method applied to FEA 25
2.3.4.2 An analytical model 25
2.3.4.3 Comparison between the experimental results, FEA results and the analytical model 26
Chapter 3 Simulation model and hardware implementation of the SRM sensorless control 28
3.1 SRM sensorless control system simulation model in Matlab/Simulink 28
3.1.1 The EMI filter 28
3.1.2 Current Regulator 32
3.1.3 The software implementation block 36
3.2 SRM sensorless drive system hardware implementation 38
3.2.1 Power Inverter 39
3.2.2 The current regulator board 40
3.2.2.1 Signal conditioning 41
3.2.2.2 Low speed position demodulator 41
3.2.2.3 Voltage integrator (flux generator) 42
3.2.2.4 Voltage comparator 43
3.2.2.5 FPGA implementation of current regulator logic 44
3.2.2.6 The printed circuit board 47
3.2.3 DSP implementation 47
Chapter 4 Inductance profile demodulator based state observer sensorless control 51

4.1 Sensorless control review 51
4.1.1 The first category, injected currents are used 51
4.1.2 The second category, the torque producing current is used 54
4.1.2.1 Observer based sensorless control 54
4.1.2.2 Chopping current pattern based sensorless control 55
4.1.2.3 Other methods using the torque producing currents 56

4.2 Proposed control strategies 56

4.3 The state observer 57

4.4 Error function definition 59

4.5 System stability and performance of the observer 64

4.6 The speed limitation of the inductance profile demodulator based observer 69
4.6.1 The time delay of the demodulator 69
4.6.2 Sampling frequency limitation 72

4.7 Simulation results 72
4.7.1 Zero speed simulation 73
4.7.2 Medium speed operation 74

4.8 Experiment results 76
4.8.1 Inductance asymmetry of the motor 76
4.8.2 Starting process 79
4.8.3 Constant speed operation 80

4.9 Speed limitation 83

4.10 The torque drop 84

4.11 The rotor position resolution 87

4.12 Transient response 87

Chapter 5 A simplified flux model based state observer sensorless control 90

5.1 A simplified flux model 90
5.1.1 Introduction of flux models 90
5.1.2 Breaking the simplified flux model into two cases 91
5.1.2.1 The case with no stator and rotor pole overlap 91
5.1.2.2 The overlap case 92
5.1.3 Verifying the model with experiment measurement 96

5.2 The simplified flux model based observer 97

5.3 The error function with the simplified model 98

5.4 Stability and performance of the simplified flux model based observer 100

5.5 Simulation results 101

5.6 Experimental results 103

5.7 Speed limitation 106

5.8 Transient response 107

5.9 Combination of the two strategies 109

Contributation and future research 111

Appendices 114

Appendix I. VHDL code in the FPGA chip 114

Appendix II. C/C++ code in DSP ____ 130

Appendix III. The torque equations for the overlap case______________________ 151

Appendix IV. The parameters of the switched reluctance motor________________ 152

Table of figures

Figure 2.1 SRM structure and geometrical dimensions 4
Figure 2.2 the SRM flux linkage 6
Figure 2.3 the SRM flux versus current at different rotor positions at 1° intervals obtained by FE analysis 6
Figure 2.4 the SRM torque versus rotor position 7
Figure 2.5 the total torque output vs rotor position 8
Figure 2.6 the total torque output vs rotor position with a fault of phase A 8
Figure 2.7 the total torque output vs rotor position with a fault of phase A and B 9
Figure 2.8 the SRM inverter 10
Figure 2.9 the SRM inverter working modes 10
Figure 2.10 SRM phase current 11
Figure 2.11 the rotor position in the non-overlap case 13
Figure 2.12 the approximated rectangular geometry in the non-overlap case 13
Figure 2.13 the contours of the main flux and the fringing flux 15
Figure 2.14 the SRM phase circuit with losses 18
Figure 2.15 the static model of SRM 19
Figure 2.16 the one-phase inverter circuit used in the classic method 21
Figure 2.17 the experimental set-up for the classic method 22
Figure 2.18 the voltage and current waveforms obtained at the aligned rotor position 22
Figure 2.19 the flux loop and the adjusted flux curve 23
Figure 2.20 the set-up diagram for the torque method 24
Figure 2.21 the experimental set-up for the torque method 24
Figure 2.22 the measured flux obtained using the classic method (solid lines) and the torque method (dashed lines) 25
Figure 2.23 The predicted flux linked by an SRM phase obtained using FEA static torque (dashed lines) and using an analytical model (solid lines) 26
Figure 2.24 Torque versus the rotor position, experimental torque (dashed lines), FEA results (dotted lines) and the analytical model results (solid lines) at the phase current equal to 10A, 20A, 30A, and 40A 27
Figure 3.1 SRM simulation system in Matlab/Simulink 29
Figure 3.2 the EMI filter circuit 29
Figure 3.3 I_{POW} and I_L waveforms in the EMI filter 32
Figure 3.4 the voltage comparators in the current regulator 33
Figure 3.5 the logic circuit of the current regulator 35
Figure 3.6 the current regulator’s digital input and output signals relative to the scaled phase current 36
Figure 3.7 the ideal and actual region of torque-producing current 37
Figure 3.8 the two cases of the on and off angles 37
Figure 3.9 the software implementation block structure 38
Figure 3.10 the hardware implementation of the SRM sensorless control system 39
Figure 3.11 the bus bar assembly 40
Figure 3.12 the experimental bus bar assembly 40
Figure 3.13 the low pass filter and demodulator circuit 42
Figure 3.14 the voltage integrator circuit 43
Figure 3.15 the voltage comparator circuit 44
Figure 3.16 the hestersis area of the voltage comparator 44
Figure 3.17 the interface between DSP and FPGA 45
Figure 3.18 the clock divider's register transfer level schematic 47
Figure 3.19 the current regulator board 48
Figure 3.20 the signal flow of the printed circuit board 49
Figure 3.21 the flow chart of the program in DSP 50
Figure 4.1 an example of the injected current and torque producing current 52
Figure 4.2 the classification of sensorless control strategies 53
Figure 4.3 the block diagram of the inductance profile demodulator based observer 58
Figure 4.4 the injected current 60

Figure 4.5 error function value versus the rotor position 62
Figure 4.6 the error function value versus the rotor position with consideration of the torque producing current 63
Figure 4.7 the error function value versus the rotor position curves at different rotor position error, -5, -4, -3, -2, -1, 1, 2, 3, 4, and 5 mechanical degrees 63
Figure 4.8 the error function value average versus the rotor position error 68
Figure 4.9 the Bode plot of the low pass filter function 70
Figure 4.10 the estimated and actual rotor position in degree of the zero speed simulation 73
Figure 4.11 the estimated rotor speed of the zero speed simulation 73
Figure 4.12 the phase currents of all 4 phases of the zero speed simulation 74
Figure 4.13 the error function generated signal of the zero speed simulation 74
Figure 4.14 the estimated and actual rotor positions when the motor is running at 2000 rpm 75
Figure 4.15 the estimated rotor speed when the motor is turning at 2000 rpm 75
Figure 4.16 the current of the 4 phases when the motor is turning at 2000 rpm 76
Figure 4.17 the error function value versus time when the motor is running at 2000 rpm 76
Figure 4.18 the electrical torque of the SRM when the motor is running at 2000 rpm 77
Figure 4.19 the $g(\theta)$ asymmetry of the motor 77
Figure 4.20 the inductance profiles of the 4 phases based on the measured $g(\theta)$ profiles 78
Figure 4.21 the error function value versus the rotor position at the rotor position error, 2 degrees (error_2) and -2 degrees (error_n2) with the inductance asymmetry 78
Figure 4.22 the estimated and actual rotor positions of the starting process 79
Figure 4.23 the estimated rotor position error during the starting process 80
Figure 4.24 the estimated rotor speed during the starting process 80
Figure 4.25 the error function value during the starting process 81
Figure 4.26 the estimated and actual rotor positions at steady state 81
Figure 4.27 the estimated position error at steady state 82
Figure 4.28 the estimated rotor speed at steady state 82
Figure 4.29 the error function value 83
Figure 4.30 the current of a phase 83
Figure 4.31 the speed limitation of the observer at t_{sample}=600μs 85
Figure 4.32 the speed limitation of the observer at t_{sample}=60μs 85
Figure 4.33 the estimated rotor position error at t_{sample}=60μs in the speed limitation experiment 86
Figure 4.34 the step response of the estimated and actual rotor speeds 88
Figure 4.35 the estimated rotor position transient response 88
Figure 4.36 the estimated rotor speed transient response 89
Figure 5.1 the ideal magnetization curve and piece wise linear curves 93
Figure 5.2 the comparison of the simplified model to the experimental data 97
Figure 5.3 the sensorless control system using the simplified model 98
Figure 5.4 the error function based on the simplified flux model 99
Figure 5.5 the average value of the error function versus the rotor position error 100
Figure 5.6 the calculated flux linkage and the actual flux linkage in the simplified flux model based observer sensorless control simulation with the SRM running at 190 rpm 102
Figure 5.7 the rotor position error versus time in the simplified flux model based observer sensorless control simulation with the SRM running at 190 rpm 102
Figure 5.8 the experimental set-up of the simplified flux model based observer sensorless control system using the simplified model 103
Figure 5.9 the estimated flux linkage and the actual flux linkage for phase A in the sensorless control experiment with the SRM operating at 190 rpm 104
Figure 5.10 the rotor position error versus time in the simplified flux model based observer sensorless control experiment with the SRM operating at 190 rpm 105
Figure 5.11 the phase current at high speeds 105
Figure 5.12 the estimated rotor speed limitation of the simplified flux based observer 106
Figure 5.13 the estimated rotor position error of the simplified flux based observer 107
Figure 5.14 the transient response of the rotor speed with $H_3=2 \times 10^3$ and $H_4=2 \times 10^4$ 108
Figure 5.15 The transient response of the rotor speed with $H_3=2 \times 10^6$ and $H_4=2 \times 10^6$ 108
Figure 5.16 the estimated rotor speed with the combination of the two observers 110
Figure 5.17 the error signal with the combination of the two observers 110

Table of tables

Table 2.1 FEA predicted flux using the vector potential, using the FEA torque, and predicted by the analytical model. 27
Table 3.1 the conversion table of the current sensor HAW-20P........41
Table 3.2 the interface bit assignment........45
Table 4.1 The torque output with different rotor position errors........86
Table 4.2 the resolution of the estimated and actual rotor positions........87

Chapter 1 Introduction

The switched reluctance motor (SRM) drive is a relative newcomer to the motor drive industry. The SRM is an electric motor in which torque is produced by the tendency of its movable part to move to a position where the inductance of the excited winding is maximized [1]. The SRM is considered as an alternative to conventional motors in variable speed applications. High efficiency at rated load and low cost make SRMs suitable to drive pumps, compressors, and fans. It is a good choice to be customized for applications ranging from turbine starter/generators to electric cars to washing machines because of its high power density and high efficiency [1]. Its phase independence characteristic makes it fault tolerant for critical applications. It is being investigated for various industrial and military applications, including electronic power steering and anti-lock braking systems in conventional vehicles and the main propulsion unit for electric/hybrid vehicles, aircraft engine starters and fuel pumps [2]. SRMs can be developed to meet the requirements of systems from a few watts to hundreds of kilowatts. The existing commercial applications include laboratory centrifuges, variable speed drives, slide door operators, screw air compressors, washing machines, food processors, air conditioning, vacuum cleaning systems, and roll door systems.

The first recognizable SRM was built by Davidson as a traction drive for an electric locomotive in 1838. But due to its poor performance it was not widely applied. Being driven with modern power electronics and using electronic controls, SRMs can achieve remarkably better performance. The stepper motor, invented and patented in the 1920's by C. L. Walker, included many features of modern veriable reluctance (VR) stepper motors and therefore of the SRM. Belsord and Hoft in 1971 and 1972 described many of the essential features of the modern SRM, with electronic commutation positively synchronized with rotor position [3]. The first reference to the name "switched reluctance" was made by S.A.Nasar in a paper in the IEE Proceedings in 1969, but it was used to describe a rudimentary switched reluctance machine [4]. Dr. Lawrenson and his colleagues connected the term switched reluctance with the modern form of the SRM. The term became popular from the 1980s onwards, through the efforts of the first commercial exploiters of the technology, Switched Reluctance Drives Ltd., which is located in the north of

England and a part of Emerson Electric Co. The machines are alternatively known as variable reluctance motors (VRM), reflecting the origins of the technology being derived from VR stepper motors. It is also called an electronically commutated reluctance (ECR) motor [1] to emphasize the character of its commutator. Another name of the SRM is brushless reluctance motor that underlines the fact that SRM is brushless.

The SRM has several advantages over conventional motors.

1) Efficient, it maintains high efficiency over wide speed and load ranges.
2) Quick start, the fact that there is no winding, commutator or permanent magnets on its rotor, and there are no brushes on its stator, along with its salient rotor poles make the SRM's rotor inertia less than that of its conventional peers so that it can accelerate more quickly.
3) Low cost, simple construction allows low manufacturing cost. Its stator and rotor are built up from a stack of salient pole laminations. There is no winding mounting cost for the rotor.
4) Wide speed range, it does not have a brush commutator mechanical speed limit, no rotor winding, and no rotor magnets so it can run up to high speeds with no specific mechanical arrangement. It also can run at low speeds and zero speed providing full rated torque.
5) Four quadrant operation, it can run forward or backward in either motoring or generating mode.
6) Shape adaptable, it may be designed as a pancake, or long to match available space [5].
7) Fault tolerant, its unique inverter topology prevents its inverter from experiencing an inverter shoot-through failure. In each leg of the inverter, there is a phase winding in between the two switches preventing shoot-through.
8) Sensorless, SRM control is possible without position sensors. The rotor position information can be obtained from the electrical parameters of the phases because of the large inductance change and flux change during an electrical period of rotor rotation.

The SRM also has some disadvantages.

1) The SRM requires a small air gap to maximize its power density which makes it more difficult to manufacture. It is also a source of inductance

asymmetry.

2) The position sensor is a fragile part of the SRM control system. In some situations, position sensors are not allowed to be used. For instance, sensor wires are prohibited in hermetically sealed compressors. Sensorless control is expected to solve this problem.
3) Potential cost of the control electronics is high. But the cost is decreasing with the development of electronic technologies.
4) The torque output inherently changes with rotor position. Since the torque produced by each phase is pulse shaped, the sum of the torque generated by all phase is not generally smooth. It is possible to make the torque smoother with a more complex control.
5) Acoustic noise, induced by the time varying phase current which deforms the stator yoke with time, can be severe. Good mechanical construction can reduce this problem.
6) The design of the SRM's electromagnetic interference (EMI) filter is challenging because the inverter induces high ac harmonics in the DC input to the inverter.

The SRM can be rotary or linear, and the rotor can be interior or exterior. The windings may be excited separately or together depending on the phase number of the motor and the torque requirement.

Chapter 2 Background

2.1 SRM basics

2.1.1 SRM structure

The SRM consists of stator and rotor laminations, both with double salient poles. The SRM can be made with different number of phases, for example, 1-phase, 2-phase, 3-phase, 4-phase, and even more phases for different applications. Each phase is wound with alternating magnetic polarity on symmetrically located stator poles. The rotor has no winding or magnets. Due to the symmetry of the phases, there is often negligible mutual inductance between them. The excitation of a phase magnetizes both the stator and the rotor. This produces a torque, causing the rotor to align its poles with the poles of the excited stator. Thus, sequential phase excitation causes rotor motion, which synchronously aligns the rotor poles with the excited phases.

The section profile of a 4-phase SRM is shown in Figure 2.1. The four phases are named A, B, C and D respectively. In the figure, the rotor is at the aligned position with phase A. This 4-phase 8/6 (# of stator poles / # of rotor poles) SRM was used in this dissertation.

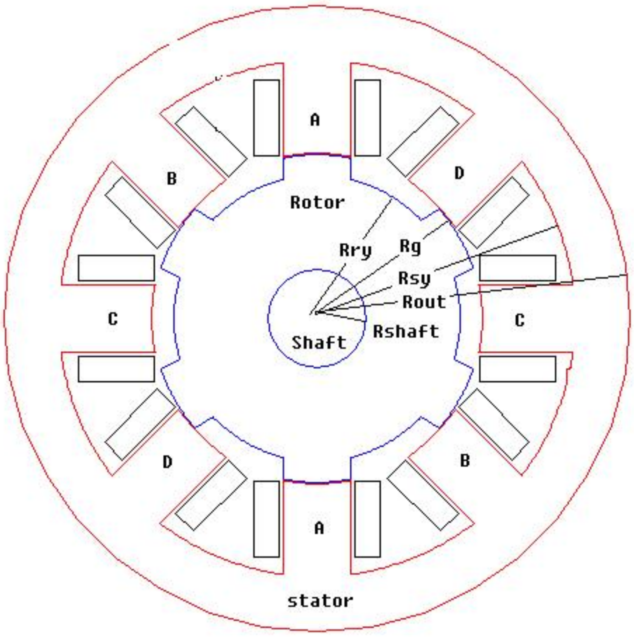


Figure 2.1 SRM structure and geometrical dimensions

Several dimensions labeled in Figure 2.1 will be used in the dissertation. Among them are:

R_{shaft}	the shaft radius;
R_{ry}	the rotor yoke radius;
R_g	the rotor pole radius, i.e. the distance from the rotor center to the air gap;
R_{sy}	the stator yoke radius;
R_{out}	the outside radius of the stator;

There are two important dimensions that are not shown in the figure. They are:

g	the air gap, i.e. the distance between the stator pole and the rotor pole when they are aligned;
l_{stk}	the length of the lamination stack, in the direction perpendicular to the page;
S_{tf}	the lamination stacking factor which is the fraction of the lamination stack length occupied by iron;

2.1.2 SRM flux linkage

The magnetic flux density tends to take the route that has lower magnetic reluctance. Thus the field produces a force that drags the rotor towards the aligned rotor position of an excited phase. Sequentially exciting the phases brakes or drives the rotor continuously. The flux linkage calculated by finite element analysis (FEA) when the rotor is at 10 mechanical degrees from the aligned position with phase A is shown in Figure 2.2. The current direction of phase A is also shown in the figure. The two poles of phase A are on the top and the bottom, as shown in Figure 2.1. The flux tends to drag the rotor towards the aligned position of phase A in this case.

The flux linkage versus phase current curve family is shown in Figure 2.3. It is obtained by finite element analysis. The curves are plotted for every mechanical degree from the unaligned rotor position (-30° for this 4-phase SRM) to the aligned rotor position (0°), respectively from the bottom to the top. The aligned position is defined as the rotor position where any pair of rotor poles is exactly aligned with the stator poles of interest, for instance, the rotor position related to phase A in Figure 2.1. The unaligned position is the rotor position where the inter-polar axis of the rotor is aligned with the stator poles of interest, for example, the rotor position related to phase C in Figure 2.1. At the aligned position, because of the small air gap, the iron saturates at high currents. At the unaligned position, due to its large air gap, the iron is not susceptible to magnetic saturation. The SRM is designed to make the iron saturate at high currents in order to maximize the energy conversion.

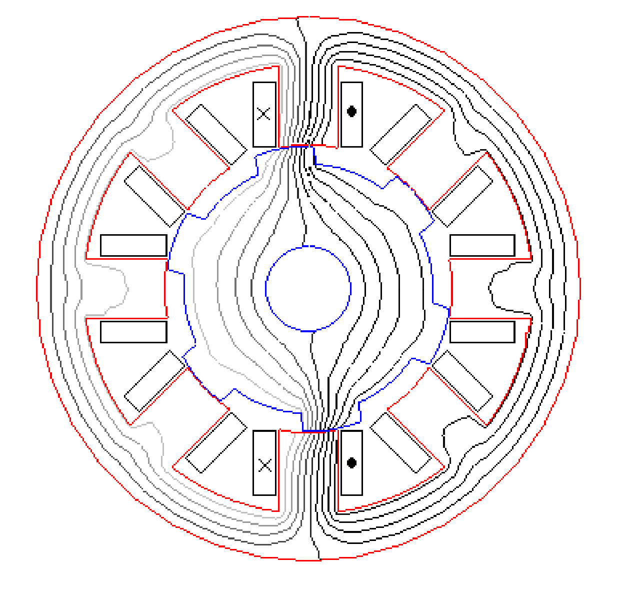


Figure 2.2 the SRM flux linkage

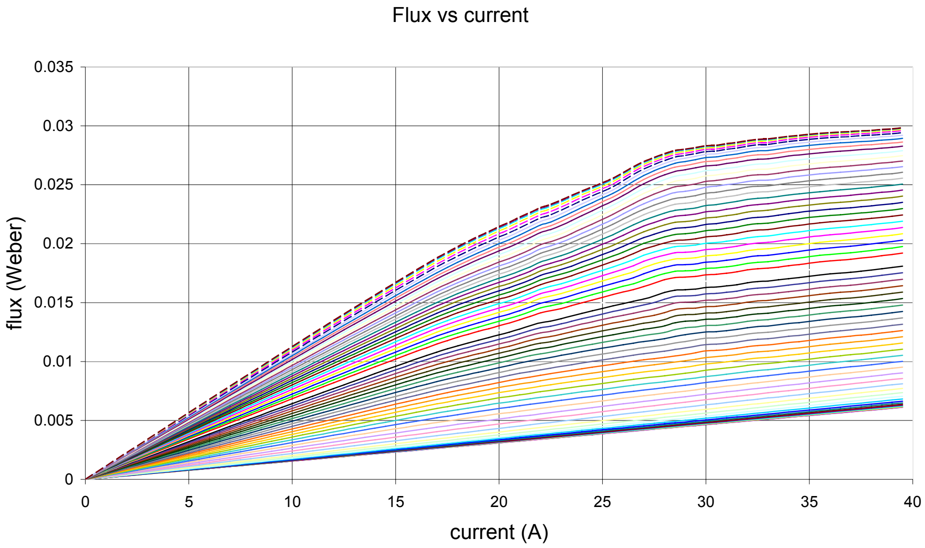


Figure 2.3 the SRM flux versus current at different rotor positions at 1° intervals obtained by FE analysis

2.1.3 SRM torque

The flux linkage tends to take the route that has the lowest reluctance, and this tendency produces a torque. The torque is a function of the phase current and the rotor position. There is no torque at the aligned position, but there is restoring toque that tends to return the rotor towards the aligned position at other rotor positions. The aligned rotor position is a stable equilibrium. At the unaligned position, the torque is also zero because it is at the minimum inductance. If the rotor is displaced to either side of the unaligned position, there is a toque that tends to displace it still further until it rotates to the closest aligned position. The unaligned position is an unstable equilibrium.

A family of static torque curves for different constant currents calculated by FEA for the 4-phase SRM used in this research is shown in Figure 2.4. In the figure, the torque curves for each of the 4 phases correspond to constant phase currents equal to 10, 20, 30, and 40 amperes. If the phases are energized at the rotor positions at which the torque is positive, the total torque output is positive and keeps driving or braking the rotor depending on the direction of rotor rotation. The total torque output in this case is shown in Figure 2.5.

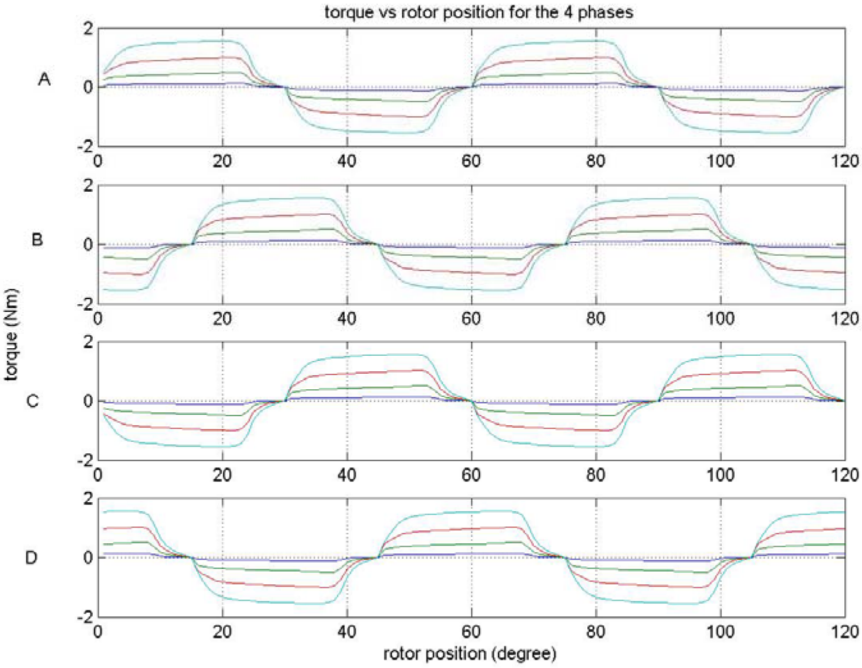


Figure 2.4 the SRM torque versus rotor position

The total torque output versus the rotor position with a fault of phase A is shown in Figure 2.6. It is zero when the rotor position is at the aligned position with regard to phase B, in this case, 15 degrees. The total torque output versus the rotor position with a fault of phase A and phase B is shown in Figure 2.7. The torque is zero at the rotor positions from the unaligned position of phase D to the aligned position of phase C.

This fact shows that the 4 phase SRM has inherent problem to work at zero speed for any position with rated torque and a fault of one phase or two phases.

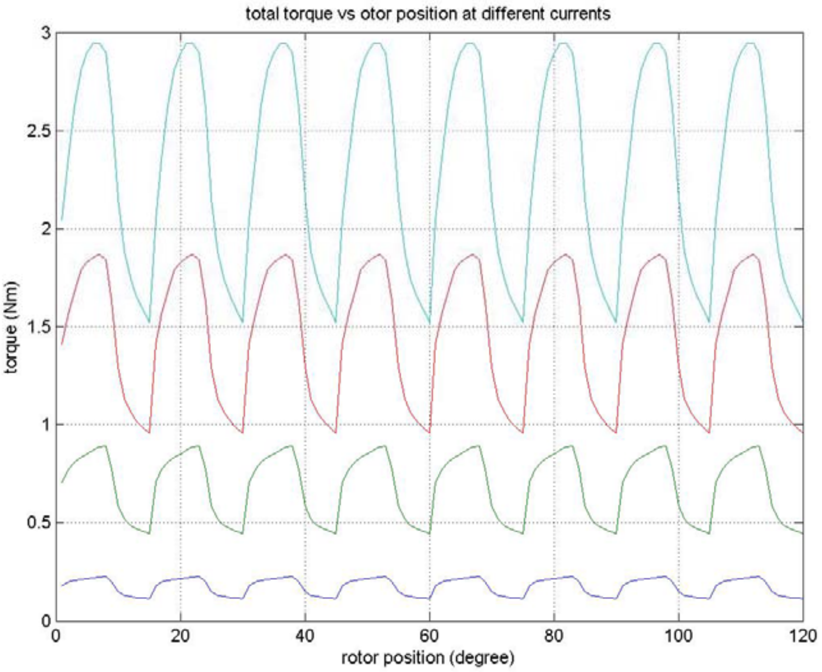


Figure 2.5 the total torque output vs rotor position

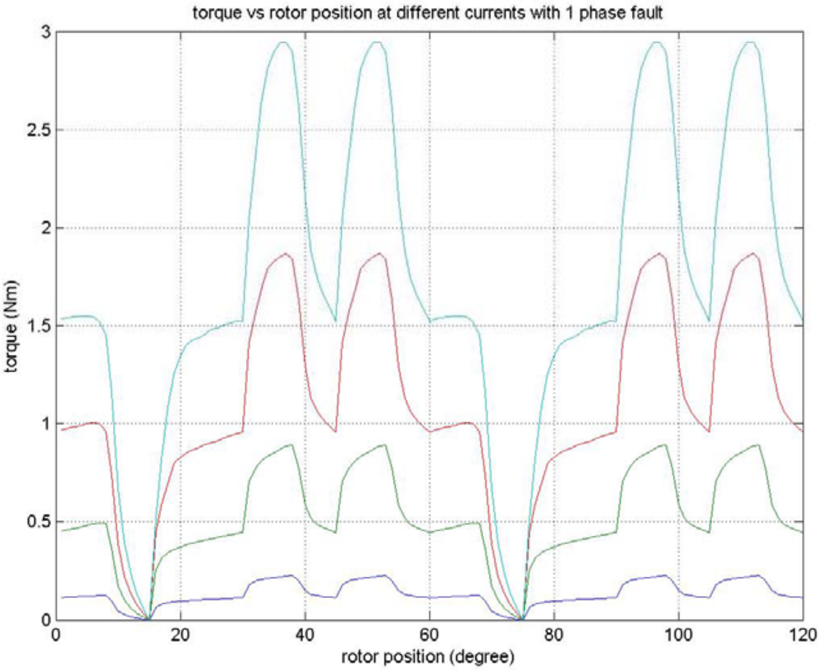


Figure 2.6 the total torque output vs rotor position with a fault of phase A

2.1.4 SRM inverter

The SRM inverter used in this research is shown in Figure 2.8. It has 4 legs for the 4-phase SRM. Each inverter leg has two power electronic switches and two diodes. The two switches and the two diodes of phase A are named Q1, Q2, D1, and D2 respectively, as shown in Figure 2.8.

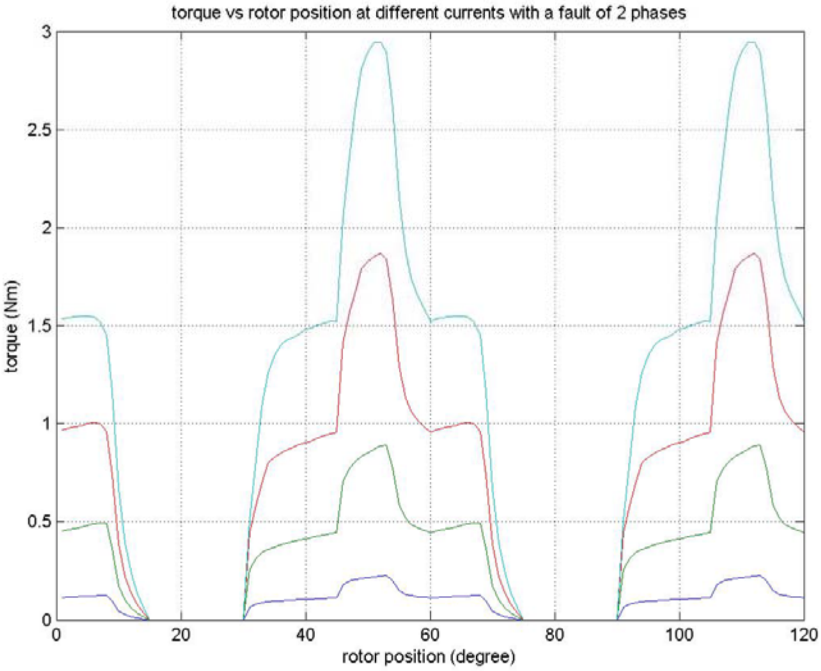


Figure 2.7 the total torque output vs rotor position with a fault of phase A and B

When both switches, Q1 and Q2 are on, the winding phase current increases and the winding is being charged. The equivalent schematic for this mode is shown in Figure 2.9a. When Q1 or Q2 is off, D1 or D2 respectively will conduct the remaining current This is called free-wheeling mode. In this mode one diode and one switch are on essentially shorting the winding. The flux keeps constant ideally in this mode and the current changes the slowest in this mode. At zero speed the current is ideally constant though the current actually decreases slowly due to the winding's resistance and the voltage drops across the inverter semiconductors. At higher speeds the SRM's back EMF will reduce the current when motoring and increase the current when generating. The equivalent schematic of the freewheeling mode is shown in Figure 2.9b. When both of the switches are turned off and there is current in the winding, both diodes will conduct current. This mode is called the discharging model. Now the voltage across the winding is the reverse of the power supply voltage. The power supply discharges the winding through the two diodes. The equivalent circuit schematic for the discharging mode is shown in Figure 2.9c. The phase current is controlled by sequencing the inverter through these three modes.

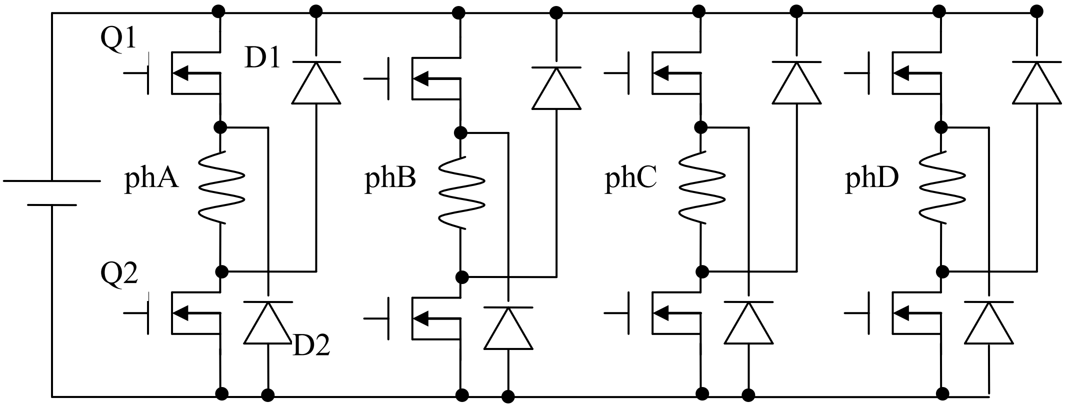
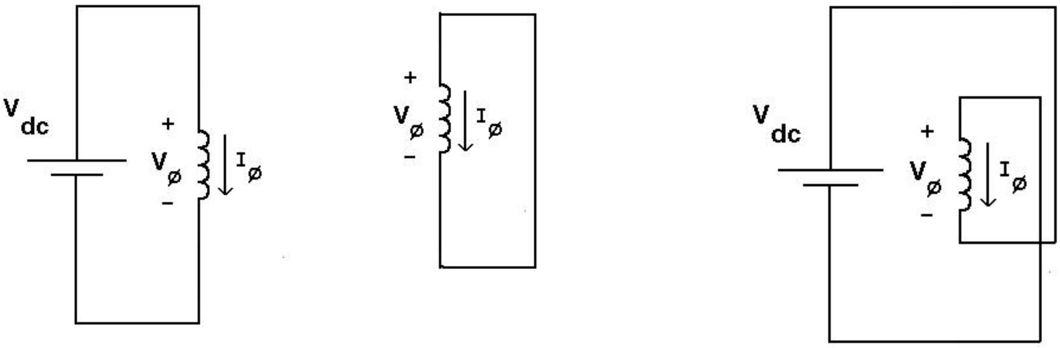


Figure 2.8 the SRM inverter



a b c

Figure 2.9 the SRM inverter working modes

2.1.5 SRM phase current

It is desirable to control the SRM's phase current to a constant value during the torque-producing period of time, as shown in Figure 2.10. First the phase current is increased to the desired constant current level called the commanded current before the rotor reaches the torque-producing region. This is accomplished by putting the inverter into the charging mode. Because of the SRM's low inductance before the rotor and stator poles overlap, the phase current rises up quickly in the charging mode. After the current reaches the commanded current determined by the commanded torque, the inverter goes to either the freewheeling mode (motoring) or the discharging mode to decrease the current. Once the current is less than the commanded current by a predetermined amount, the inverter goes back to the charging mode to increase the current again. This procedure is continued until the rotor is close to the end of the torque-producing region, after which the inverter is put into the discharging mode to reduce the phase current to zero rapidly to avoid producing the opposite torque.

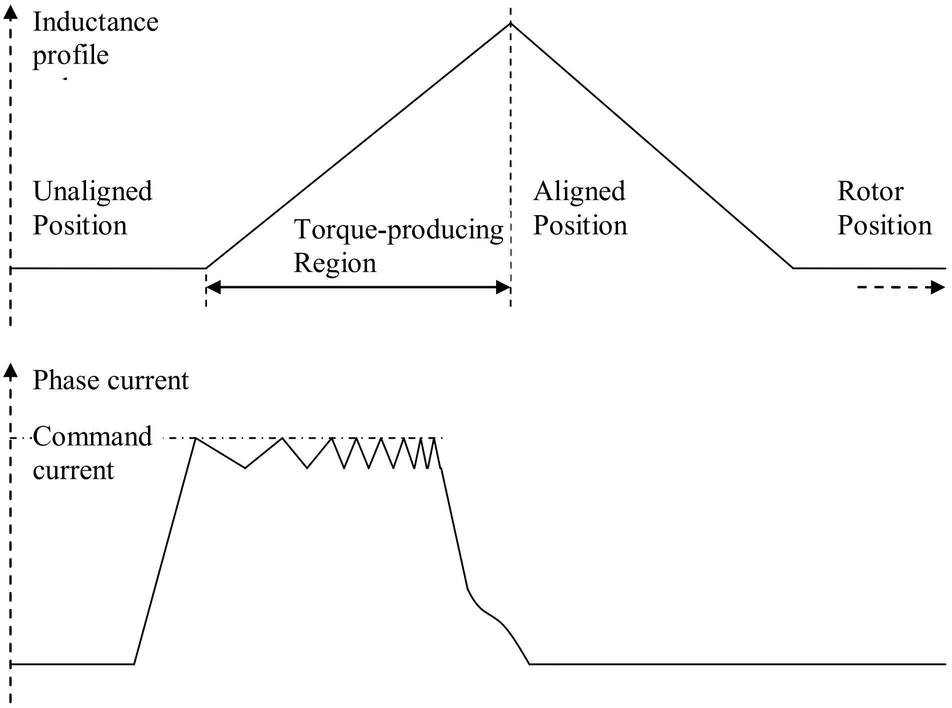


Figure 2.10 SRM phase current

2.2 An analytical model of the SRM

To model the SRM the flux linked by a phase must be determined from which other machine properties like torque, inductance and back EMF can be computed. Different methods can be used from FEA to curve fitting, from truncated Fourier series to exponential functions. In the simulation system that this dissertation uses, an analytical flux model is used. It is constructed by considering two cases according to the rotor position. The first case, termed the overlap case, consists of those rotor positions for which a rotor pole overlaps with the stator pole of interest. In the second case, namely the non-overlap case, the stator pole under consideration does not have any angular extent that overlaps with a rotor pole.

In the non-overlap case, it is assumed that the inductance is independent of current. There is only fringing flux and iron saturation is negligible. But in the overlap case, iron saturation needs to be considered and the total flux consists of both a main flux and a fringing flux.

The non-overlap case is explained in more detail in [6], while the overlap case is described in more detail in [7]. The basic results from these two references are introduced here because they are used as the SRM model of the simulation system in Matlab/Simulink.

2.2.1 The non-overlap case

The geometry of the SRM in this case is shown in Figure 2.11. The stator poles of phase A do not have any overlapped area with any rotor poles. The actual SRM geometry can be approximated with the unwrapped rectangular geometry in Figure 2.12. The dimensions lr_1 and lr_2 are equal if the rotor is at the unaligned position relative to the phase A stator pole and are unequal otherwise. The parameter lr is the total horizontal length of the rotor yoke between the two neighbor poles.

The flux linked by a phase in this case is divided into two parts. One part is contributed by the part of the magnetic field generated by the winding that goes from the stator pole to the rotor through the rotor slot between the two rotor poles. The other part is contributed by the part of the magnetic field generated by the winding that that returns through the stator slot. To obtain the contribution by the rotor, the vector potential A is introduced and the boundary conditions of the rectangular region between the rotor poles are defined. Then the basic magneto static theories are applied to solve the flux linked by the phase due to the magnetic field that goes to the rotor.

The flux linked by a phase (λ_{pr}), contributed by the rotor, in terms of the phase current (I_{Φ}) is

$$\lambda_{pr} = \frac{nser}{npar} 4 \mu_o N_p{}^2 l_{stk} \cdot l_r \sum_{n_{odd}} \frac{\dfrac{\sin\left(\dfrac{\pi n lr_1}{lr}\right)}{lr_1} + \dfrac{\sin\left(\dfrac{\pi n lr_2}{lr}\right)}{lr_2}}{(\pi n)^2 \tanh\left(\dfrac{\pi n hr}{lr}\right)} I_\phi \tag{2-1}$$

where nser is the number of pole windings in series and npar is the number of pole windings in parallel, N_p is the number of winding turns per pole, μ_o is the permeability of free space, n_{odd} is the odd integers from one to infinity.

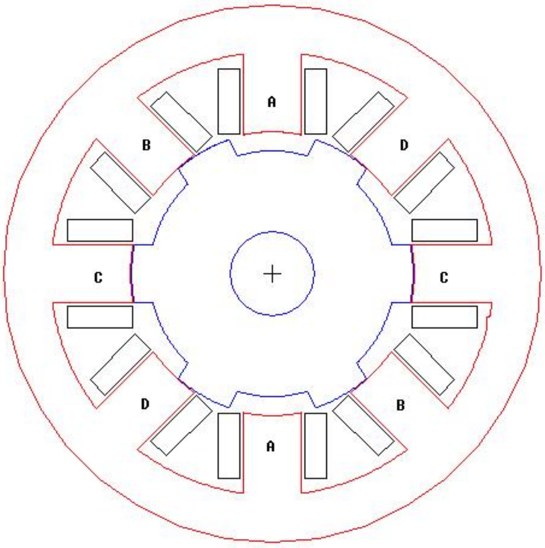


Figure 2.11 the rotor position in the non-overlap case

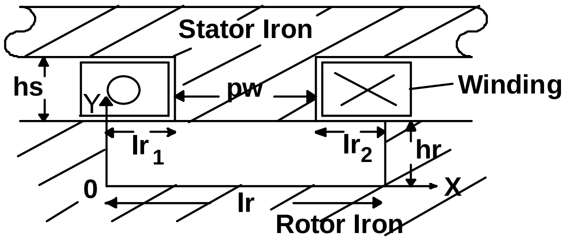


Figure 2.12 the approximated rectangular geometry in the non-overlap case

The starting point for finding the field in the stator slot is the approximate geometry in Figure 2.12. Like the rotor case, the stator slot also forms a rectangular box. The field actually has to be found in both stator slots on either side of the stator pole. Since the basic geometry is the same for both of these slots, the field solution only has to be obtained for one slot and then the same solution is applied to the other [6].

In [6], the stator contribution to the unaligned flux is:

$$\lambda_{ps} = \frac{nser}{npar}\frac{2N_p{}^2 l_{stk} I_\phi}{hs \cdot lw}\left\{\frac{2}{3}csy \cdot hs^2 - csx\frac{lw}{2} + \right.$$

$$+\sum_{n=1}^{\infty} ash_n\left[\cosh\left(\frac{\pi nhs}{ls}\right)-\frac{ls^2}{(\pi n)^2 hs\cdot lw}\sinh\left(\frac{\pi nhs}{ls}\right)\sin\left(\frac{\pi nlw}{ls}\right)\right]+$$

$$+\sum_{n=1}^{\infty} asp_n\left[\cos\left(\frac{\pi nlw}{ls}\right)-\frac{ls}{lw\pi n}\sin\left(\frac{\pi nlw}{ls}\right)\right]\Bigg\} \tag{2-2}$$

where l_s is the length of the stator slot in x direction, h_s is the height of the slot, l_w is the length of the winding in x direction, ash_n and asp_n are the Fourier coefficients of the solution given in [6].

Thus the total phase flux in the non-overlap case is

$$\lambda_{\phi,no}(\theta,i_\phi)=\lambda_{pr}(\theta,i_\phi)+\lambda_{ps}(\theta,i_\phi) \tag{2-3}$$

2.2.2 The overlap case

In the overlap case, the flux linked by a phase is broken into two parts, , one due to the main field called the main flux and the other due to the fringing field called the fringing flux. The major difference between these two fluxes is that their contour paths have different air gaps. The main flux is computed using a contour that passes through the small air gap that is between the rotor pole and the stator pole where they overlap. The fringing flux is computed using a contour that passes thorugh the greater air gap between the rotor yoke and the stator pole, as shown in Figure 2.13.

The main flux contribution, including the effect of iron saturation, to the total flux linked by a phase is [7]

$$\lambda_m(\theta,I_\phi)=\lambda_o\frac{p_w-R_g\theta}{\left(1+\frac{g}{l_{Fe,m}}\right)g}\left[\left(1+\frac{2g}{l_{Fe,m}}\right)\frac{I_\phi}{n_{par}}+\frac{l_{m1}B_{sat}}{\mu N_p}-\sqrt{\left(\frac{l_{m1}B_{sat}}{\mu N_p}\right)^2+\frac{2l_{m2}B_{sat}}{\mu N_p}\frac{I_\phi}{n_{par}}+\left(\frac{I_\phi}{n_{par}}\right)^2}\,\right] \tag{2-4}$$

where

$$\lambda_o=n_{ser}\cdot\mu_o\cdot\frac{{N_p}^2}{2}\cdot l_{stk}\cdot s_{tf}$$

Here $l_{Fe,m}$ is one half of the total length of the main flux contour in the iron and g is the air gap. The number of turns and the current in the winding around each of the two stator poles that make up the phase are N_p and I_Φ respectively. μ is the magnetic permeability of the iron, p_w is the stator pole width, and B_{sat} is the flux density when the iron is saturated. The iron is characterized by μ and B_{sat}.. The angle θ is taken to be zero at the aligned position.

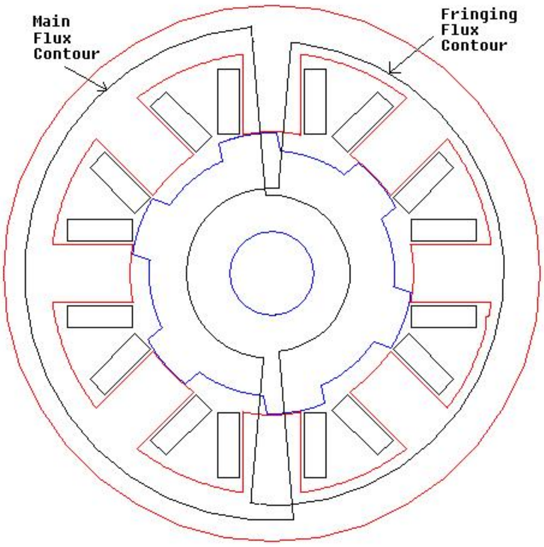


Figure 2.13 the contours of the main flux and the fringing flux

The fringing flux is found by simply substituting the effective fringing flux air gap, g_f, [7] for the main flux air gap g and substituting the area of the fringing flux path for the area of pole overlap. Thus the fringing flux is

$$\lambda_f(\theta, I_\phi) = \lambda_o \frac{R_g\theta}{g_f}\left[\left(1+\frac{2g_f}{l_{Fe,f}}\right)\frac{I_\phi}{n_{par}} + \frac{l_{f1}B_{sat}}{\mu N_p} - \sqrt{\left(\frac{l_{f1}B_{sat}}{\mu N_p}\right)^2 + \frac{2l_{f2}B_{sat}}{\mu N_p}\frac{I_\phi}{n_{par}} + \left(\frac{I_\phi}{n_{par}}\right)^2}\,\right] \tag{2-5}$$

In the equation,

$l_{f1} = l_{Fe,f} + (\mu_r + 1)g_f$ and $l_{f2} = l_{Fe,f} - (\mu_r - 1)g_f$

where $l_{Fe,f}$ is a half of the total length of the fringing flux contour in iron.

The total flux linked by an SRM phase in the overlap case is simply the sum of the main flux and the fringing flux linked by that phase.

$$\lambda_{\phi,ov}(\theta, I_\phi) = \lambda_m(\theta, I_\phi) + \lambda_f(\theta, I_\phi) \tag{2-6}$$

The main flux is given by (2-4) and the fringing flux is given by (2-5).

The flux over a full period of rotor position is

$$\lambda_\phi(\theta, I_\phi) = \begin{cases} \lambda_{\phi,ov}(\theta, I_\phi) & (|\theta| \le \dfrac{p_w}{R_g} \quad or \quad overlap) \\ \lambda_{\phi,no}(\theta, I_\phi) & (|\theta| > \dfrac{p_w}{R_g} \quad or \quad non-overlap) \end{cases} \tag{2-7}$$

2.3 A torque method to obtain the flux of SRM

Because of its salient poles and the fact that iron saturation plays a critical role in its operation, it is difficult to model the SRM precisely [8]. Ultimately it is necessary to measure the flux linked by a phase of the SRM to predict its performance and to verify model results. Typically a pulse voltage is applied to one phase of the SRM with its rotor locked at a fixed rotor position. As the current in the phase increases the phase current is measured and the phase voltage is integrated to obtain the flux as a function of current. This process is repeated at different rotor positions to obtain a family of flux curves for different rotor positions. If first a positive voltage pulse is applied to the phase to increase the current and then a negative voltage pulse is applied to return the current to zero, it is found that the increasing and decreasing flux curves are not equal and a loop occurs, because of both iron and copper losses. The flux loop resulting from the losses incurred during the measurement complicates determining the phase flux linkage curves, which should be lossless.

There is an alternative "torque method" to measure the flux linked by a phase that does not need a correction for losses. It applies a dc current to the SRM's phase instead of the pulse voltage normally used, to avoid the iron losses. The output torque is measured instead of the phase voltage so that the results do not depend on the copper losses. The static torque curves of the SRM are measured versus phase current at a fixed rotor position and the process is then repeated for different rotor positions. Then this data, a measurement of the unaligned inductance, and conservation of energy are used to compute the loss independent flux linked by the SRM's phase. The same approach can be used to compute the SRM's flux leakage from finite element analysis (FEA) computed static torque data, simplifying the computation of the nonlinear flux linkage curves.

2.3.1 Power losses

It is shown below how the power losses (including iron losses and copper loss) are avoided in the torque method for measuring the flux linked by a phase.

2.3.1.1 Eddy current loss and hysteresis loss

According to [9, 10], the eddy current loss per unit volume is proportional to square of the derivative of flux density.

$$P_{veddy} = K_{eddy}\left(\frac{dB}{dt}\right)^2 \qquad \text{2-8}$$

where K_{eddy} is a constant of proportionality, B is the magnetic field density in the winding, and P_{veddy} is the power loss per unit volume due to the eddy current. Because the phase voltage V_{ph} is proportional to the derivative of flux density,

$$V_{ph} \propto \frac{dB}{dt} \qquad \text{2-9}$$

the eddy current losses are proportional to the phase voltage squared

$$P_{veddy} \propto K_{eddy} \cdot V_{ph}{}^2 \qquad \text{2-10}$$

and thus the eddy current losses can be modeled as a resistor connected in parallel with the electromagnetic phase voltage V_{ph}.

Hysteresis loss is more complicated to compute analytically. It can be expressed as being proportional to the derivative of flux density [9]. To obtain a simple conceptual schematic of the SRM's phase circuit, the two iron losses mechanisms are modeled as a resistor R_{iron} in parallel with the winding as shown in Figure 2.14.

The copper loss is the power consumed by the phase resistance, which is represented by a resister R_{copper} in series with the phase winding as shown in Figure 2.14.

The classic way to measure the flux linked by an SRM phase is to lock its rotor, apply a pulse voltage V_{ph}, measure I_{copper} and integrate the phase voltage V_{ph} to get the flux. This process is repeated at various rotor positions to obtain a family of flux versus current curves. It is clear from Figure 2.14 that the measurable current I_{copper} is not equal to the electromagnetic winding current I_{ph} because of the iron losses and the measurable phase voltage V_{ph} is not equal to the electromagnetic winding voltage because of the copper resistance. The flux linked by an SRM phase using this method has inherent errors due to the iron and copper losses.

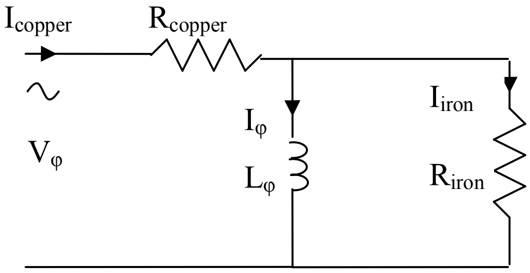


Figure 2.14 the SRM phase circuit with losses

2.3.1.2 Avoiding the loss induced measurement errors

The SRM's phase circuit shown in Figure 2.14 reduces to the circuit shown in Figure 2.15 when the input is a dc current and thus the nonlinear SRM phase inductance L_{φ} behaves as a short circuit, shorting the iron loss resistor. In this case all of the measured current I_{copper} flows through the shorted equivalent magnetic winding and none flow through the iron loss resistance so that the winding current I_{copper} is exactly equal to the current I_{φ}. Because the measured torque only depends on the change of the energy stored in the winding the voltage across the copper loss resistor R_{copper} never needs to be known and thus the value of R_{copper} does not need to be known.

Even though the model of the SRM phase in Figure 2.14 and Figure 2.15 is not perfect the concept presented is more general than the model. When the input voltage is dc and the rotor is not rotating, the eddy current loss and hysteresis loss are both zero since they both are due to the time rate of change of the magnetic field in the iron.

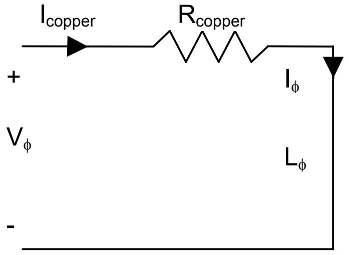


Figure 2.15 the static model of SRM

2.3.2 Obtaining flux from the static torque

The flux linked by the SRM's phase is found using conservation of energy. The starting point is the co-energy defined in the usual way.

$$dW'(\theta,i) = T_e(\theta,i)d\theta + \lambda(\theta,i)di \qquad \text{2-11}$$

Here θ is the rotor position, i is the phase current, $W'(\theta,i)$ is the co-energy, $T_e(\theta,i)$ is the static torque, $\lambda(\theta,i)$ is the flux linkage as a function of the phase current and the rotor position. The Co-energy is computed by integrating the torque at a fixed phase current (di = 0). Because co-energy is conserved the value of this integral does not depend on the choice of path in the i-θ space.

$$W'(i,\theta) = W'(\theta_{ini},i) + \int_{\theta i}^{\theta} T_e(\theta,i)d\theta \qquad \text{2-12}$$

where θ_{ini} is the initial rotor position, which is taken to be the unaligned rotor position. For the unaligned rotor position, the co-energy can be computed simply because there is no magnetic iron saturation at this position.

$$W'(\theta_u, i) = \frac{1}{2} L_u \cdot i^2 \qquad \text{2-13}$$

Here θ_u is the unaligned rotor position and L_u is the phase inductance at the unaligned rotor position. The unaligned inductance can be computed from a single FEA calculation or found from a single experimental measurement, at a sufficiently low current that the SRM losses have a very small effect. The flux linked by a phase is the derivative of co-energy relative to the phase current holding the rotor position constant, as shown below

.

$$\lambda(\theta, i) = \frac{\partial W'(\theta, i)}{\partial i} \qquad \text{2-14}$$

From experiments or finite element analysis, a $N_i \times N_\theta$ torque matrix T_e is created. Its rows represent different rotor positions from the unaligned rotor position to the aligned rotor position, and its columns represent different phase current samples from zero to a maximum value. Current samples are $i_1, i_2 \cdots i_{N_i}$, while rotor positions are $\theta_1, \theta_2 \cdots \theta_{N_\theta}$. Here i_1 is defined to be zero and θ_1 is defined to be the unaligned rotor position θ_u. With these definitions equations (2-12) and (2-14) are transformed into their discrete form in (2-15) and (2-16).

$$\begin{aligned} &W'(\theta_k, i_j) = W'(\theta_1, i_j) + \sum_{n=2}^{k} [T_e(\theta_n, i_j) \times (\theta_n - \theta_{n-1})] \\ &\quad k = 2, 3 \cdots N_\theta,\ j = 1, 2 \cdots N_i \end{aligned} \qquad \text{2-15}$$

$$\begin{aligned} &\lambda(\theta_k, i_j) = \frac{W'(\theta_k, i_j) - W'(\theta_k, i_{j-1})}{i_j - i_{j-1}} \\ &\quad k = 1, 2 \cdots N_\theta,\ j = 2, 3 \cdots N_i \end{aligned} \qquad \text{2-16}$$

The initial values of the co-energy and flux at the boundaries of the problem are

$$W'(\theta_1, i_j) = \frac{1}{2} L_u \, i_j^{\,2} \qquad j = 1, 2 \cdots N_i \qquad 2\text{-}17$$

$$\lambda(\theta_k, i_1) = 0 \qquad k = 1, 2 \cdots N_\theta \qquad 2\text{-}18$$

Here $i_1 = 0$ A.

2.3.3 Comparing with the classic method

In this section, the experimental results for the classic method and the torque method to measure the SRM's flux linage curves are compared.

2.3.3.1 The Classic method

As discussed above, the classic method of measuring the flux linked by a SRM phase is to apply a pulse voltage while the rotor is locked at a certain position. The current is measured and the voltage is integrated to obtain flux and thus the flux versus current curve is plotted. A dc power supply and a simple one-phase inverter consisting of two power MOSFETs and two diodes are used to generate the pulse voltage to the phase under test. A signal generator provides the input signals to the two power MOSFETs that conduct the increasing phase current. The two diodes conduct the decreasing current when the two MOSFETs are off. The circuit used in the classic method and its experimental set-up are shown in Figure 2.16 and Figure 2.17 respectively. The voltage and current waveforms obtained at the aligned rotor position are shown in Figure 2.18. The sudden increase of the current in the increasing period is due to the iron saturation.

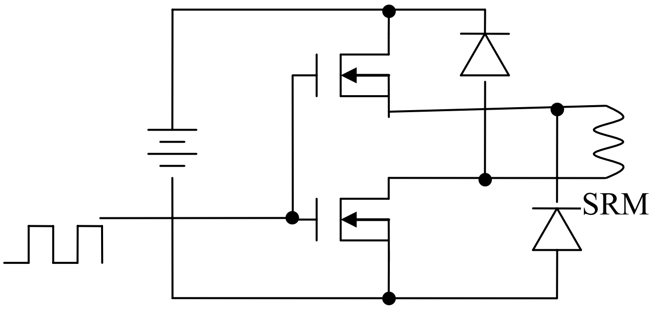


Figure 2.16 the one-phase inverter circuit used in the classic method

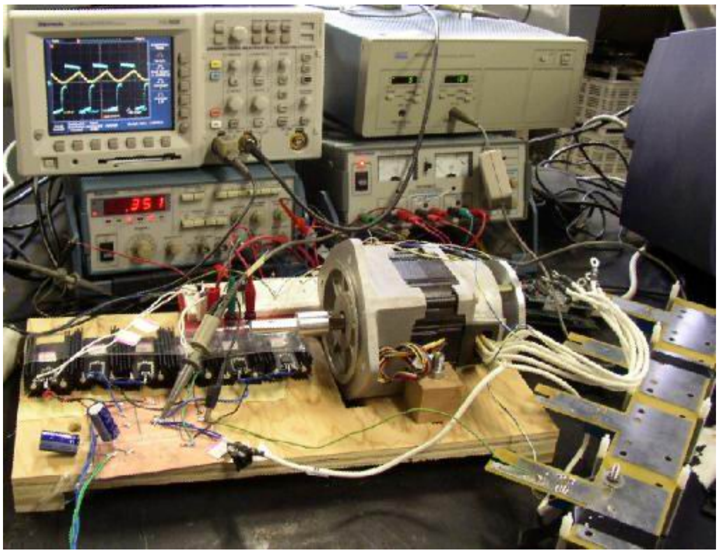


Figure 2.17 the experimental set-up for the classic method

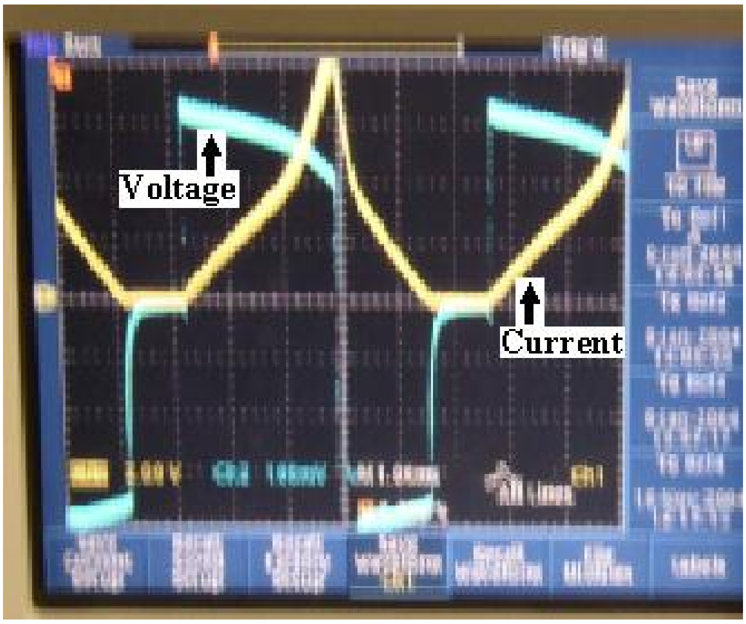


Figure 2.18 the voltage and current waveforms obtained at the aligned rotor position

Because the copper losses and the iron losses, a loop is formed in the flux-current space because the rising and the falling currents do not match. The flux loop and an adjusted flux curve obtained at the aligned rotor position are shown in Figure 2.19. In this figure, the phase current and the flux are both filtered digitally to eliminate the high frequency noise. The adjusted flux curve is obtained from the flux loop data by subtracting an I_ϕ R_ϕ voltage drop from the terminal voltage before it is integrated to obtain the flux. The value of R_ϕ is adjusted so there is no loop. Doing this assumes the R_ϕ value does not change as the current changes and that the iron losses can be modeled as an equivalent constant series resistance. An interesting result is that the adjusted curve is not in the middle of the loop as would be expected. This happens because the voltage across the electromagnetic winding is lower than the terminal voltage when the phase current is increasing and larger in value when the phase current is decreasing. Thus the flux increases less rapidly and to a lower value than the terminal voltage would indicate when the flux increases and decreases more rapidly than the terminal voltage would indicate when the flux decreases.

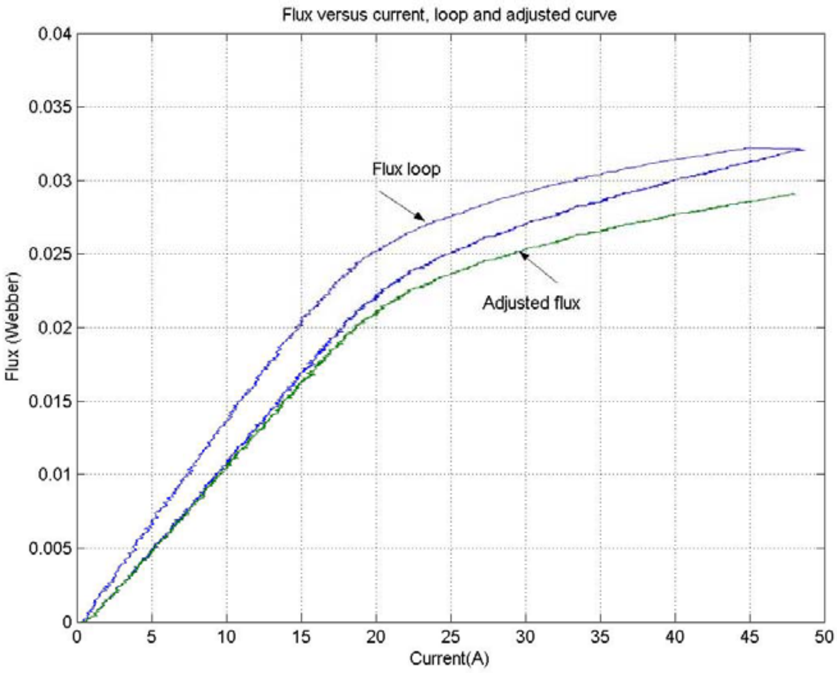


Figure 2.19 the flux loop and the adjusted flux curve

2.3.3.2 The torque method

The diagram and the experimental set-up for the torque method are shown in Figure 2.20 and Figure 2.21. A torque transducer is used to measure the static torque and a position sensor is used to obtain the rotor position information. Equations (2-16, 17, 18 and 19) are used to obtain the flux from the torque data. In the experiment Ni is 40 and N_θ is 30. It is best to obtain more data in the current direction rather than the θ direction since a derivative of the co-energy with respect to current is required and derivatives are more numerically noisy than integrals.

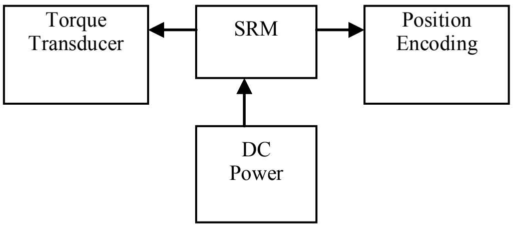


Figure 2.20 the set-up diagram for the torque method

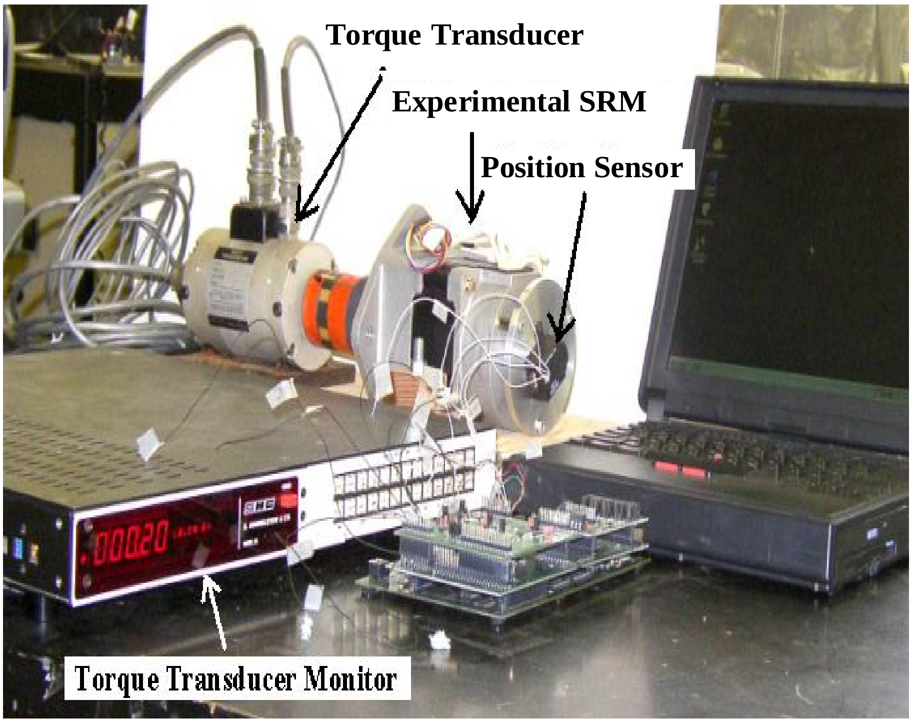


Figure 2.21 the experimental set-up for the torque method

2.3.3.3 Comparison between the torque method and the classic method

The experimental fluxes obtained by using the torque method and the classic method are shown in Figure 2.22. In this figure, the flux curves are presented at the aligned position (0°), 5°, 10°, 15°, 20°, 25° and the unaligned position (30°). The curves are smoothed by curve fitting with polynomial function of degree 5. Though the results are similar for both methods they are not identical, presumably because of the errors inherent in compensating the classic data for losses. Another error that must be dealt with in the classic method is differences in the time at which the voltage and current are sampled that result from the dynamic nature of the experiment. Because the torque method uses a static experiment this sampling error does not occur.

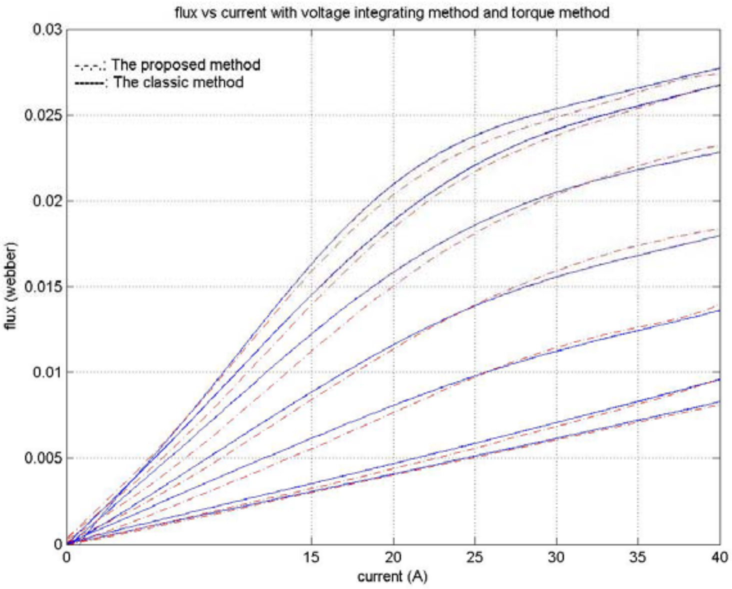


Figure 2.22 the measured flux obtained using the classic method (solid lines) and the torque method (dashed lines)

2.3.4 Applying the torque method to FEA

Since calculating torque in finite element analysis (FEA) when there is iron saturation is easier to automate than calculating the flux linked by a phase directly (from the vector potential), the torque method is expected to be a useful approach for obtaining the flux linked by a SRM phase from FEA. The FEA application of the torque method is verified with the measured data and results from an analytical model [6, 7].

2.3.4.1 The torque method applied to FEA

The finite element analysis model of the experimental motor is built with Ansoft Maxwell 2D software and the static torque is calculated. The FEA model drawing is shown in Figure 2.1. The calculated torque is used to obtain the flux linked by a phase. To use finite element analysis, the dimensions of the motor, turn number of the winding, iron material need to be known.

2.3.4.2 An analytical model

An analytical flux model that includes iron saturation in the motor is presented in [6, 7]. The dimensions, iron properties, and number of turns used in FEA are used in this analytical model.

2.3.4.3 Comparison between the experimental results, FEA results and the analytical model

The comparison between the FEA results and the analytical model is shown in Figure 2.23. In this figure, the flux curves are presented at the aligned position (0°), 5°, 10°, 15°, 20°, 25° and the unaligned position (30°). The curves are smoothed by curve fitting with polynomial function of degree 5. Presumably the FEA results are more accurate and the error between these results and the analytical model results are due to the difficulty of modeling the nonlinear behavior of the SRM analytically.

Both the analytical and FEA flux linkage curves predict higher flux levels than the measured data. Also, both the analytical and FEA flux linkage curves saturate more strongly than the measured curves do. This is most likely due to errors in modeling the iron B-H curve. The experimental SRM is a commercial machine and the manufacturer considers the iron characteristics and the motor geometry to be proprietary information. Thus both the analytical and FEA results assumed the iron was 3.25% SiFe with an initial relative permeability of 5000 and a saturation flux density equal to 1.8 T. The dimensions used in both the analytical and FEA calculations were obtained from measurements of the partially disassembled machine. This is another source of error. The torque obtained from the experimental measurement, FEA computation and the analytical model is shown in Figure 2.24. Polynomial curve fitting is applied to the measured torque and the FEA computed torque. The difference between the FEA torque and the analytical model torque is due to the difficulty of modeling the nonlinear behavior of the SRM analytically.

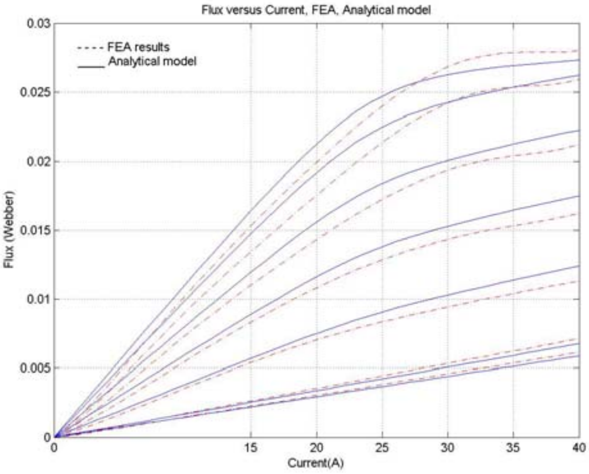


Figure 2.23 The predicted flux linked by an SRM phase obtained using FEA static torque (dashed lines) and using an analytical model (solid lines)

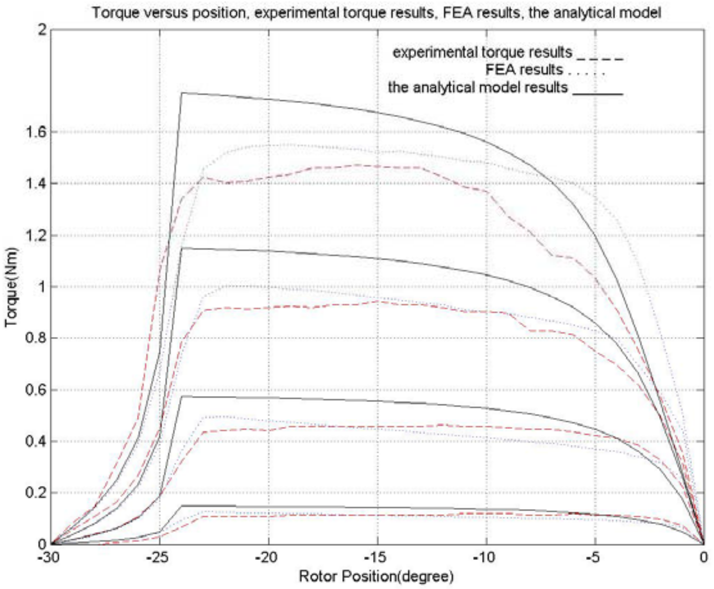


Figure 2.24 Torque versus the rotor position, experimental torque (dashed lines), FEA results (dotted lines) and the analytical model results (solid lines) at the phase current equal to 10A, 20A, 30A, and 40A

The flux linked by a phase was also computed using the vector potential at the aligned position for two different currents. These results are compared to those obtained using the FEA torque and the analytical model results in Table 2.1.

Table 2.1 FEA predicted flux using the vector potential, using the FEA torque, and predicted by the analytical model.

I_ϕ	Flux from vector potential (Weber)	Flux from torque (Weber)	Flux from the analytical model (Weber)
1 A	1.12×10^{-3}	1.12×10^{-3}	1.12×10^{-3}
40A	0.0289	0.0299	0.0273

Chapter 3 Simulation model and hardware implementation of the SRM sensorless control

In this chapter, the SRM sensorless control system simulation and implementation in hardware and software are presented. The SRM used here is manufactured by Rocky Mountain Technologies. It is a 42Vdc, 2 hp peak power, four phase or 8/6 SRM with a maximum speed of 15,000 rpm. The simulation is done using Matlab/Simulink. The system is implemented with a digital signal processor (DSP) of TMS320C6711 made by Taxes Insturments, an A/D converter board, a separate analog current regulator with a digital control logic circuit, and a standard 4 phase SRM inverter..

3.1 SRM sensorless control system simulation model in Matlab/Simulink

The control system was designed and simulated using Matlab/Simulink. The SRM drive system structure is shown in Figure 3.1. It consists of five components, an electromagnetic interference (EMI) filter, a power electronic inverter, an SRM, a current regulator and a software implementation block.

The EMI filter eliminates ac harmonics generated by the inverter in the DC input current. The current regulator keeps the phase currents equal to the commanded current when the SRM phases need to be energized. It also controls the inverter switches to create the sensing currents when the phases are idle and the sensing currents are needed. The software implementation block generates the commands to the current regulator including the commanded current and the logic signal to control whether the inverter should generate torque producing current or sensing current. The software implementation block estimates the rotor position from the measured phase current and/or flux input from the analog-to-digital board. Since the inverter and SRM model have been described earlier, the EMI filter, the current regulator, and the software implementation block will be described in this chapter.

3.1.1 The EMI filter

The EMI filter consists of an inductor and a capacitor, both of which have parasitic resistance, as shown in the dashed block in Figure 3.2.

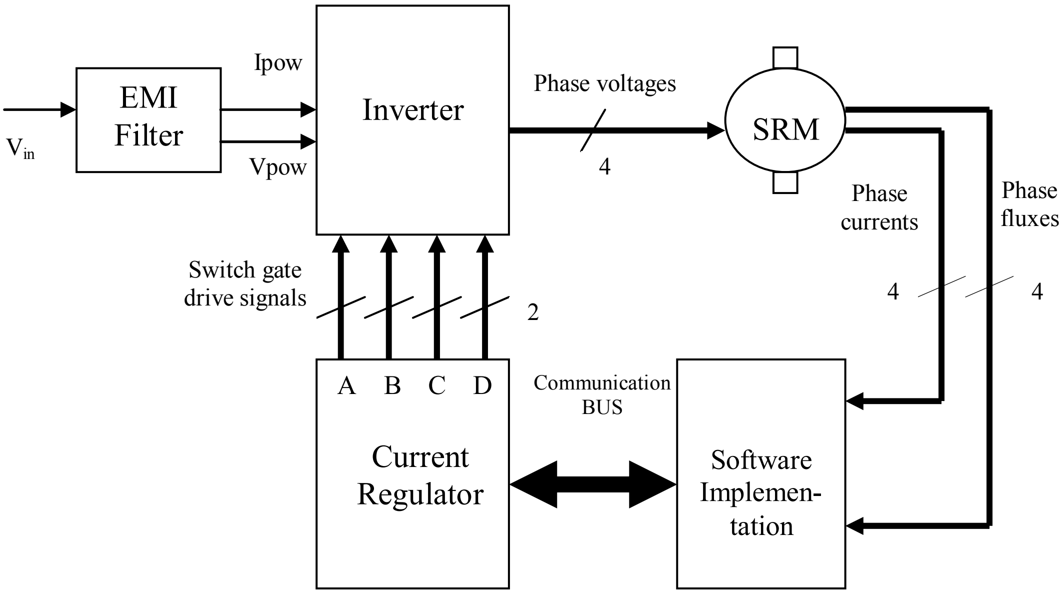


Figure 3.1 SRM simulation system in Matlab/Simulink

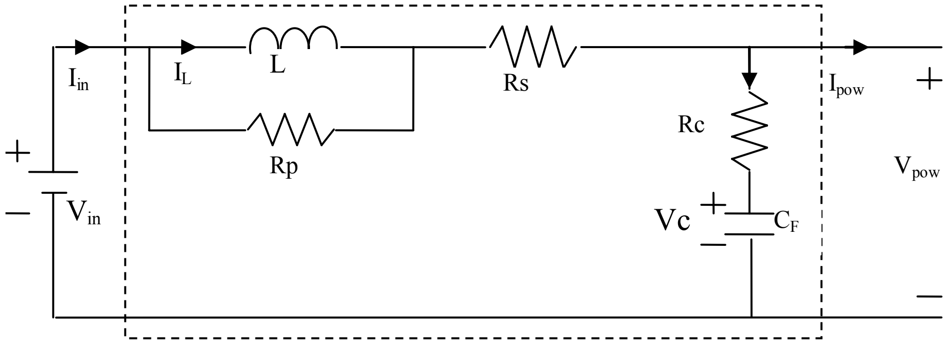


Figure 3.2 the EMI filter circuit

To model the EMI filter, its state equations need to be obtained so that the state space function block in Matlab/Simulink can be used. The Matlab/Simulink filter model is developed with the inputs V_{in} and I_{POW}, and the outputs, I_L and V_{POW}. Two states are defined in the state equations. One is the current in the inductor, I_L and the other is the voltage across the capacitor, V_C. According to KVL,

$$V_{in} = L\frac{dI_L}{dt} + R_S\left(I_L + \frac{L}{R_P}\frac{dI_L}{dt}\right) + V_C + R_C C_F\frac{dV_C}{dt} \qquad 3\text{-}1$$

According to KCL,

$$I_L + \frac{L}{R_P}\frac{dI_L}{dt} = C_F\frac{dV_C}{dt} + I_{POW} \qquad \text{3-2}$$

where the parameters, L, R_S, R_P, R_C, and C_F are the inductor, the parasitic resistor of the inductor in series with the inductor, the parasitic resistor of the inductor in parallel with the inductor, the parasitic resistor of the capacitor in series with the capacitor and the capacitor, as illustrated in Figure 3.2. The two states, I_L and V_C, are also shown in the figure.

After rearranging, (3-1) and (3-2) become

$$V_{in} = L(1+\frac{1}{R_P})\frac{dI_L}{dt} + R_C C_F\frac{dV_C}{dt} + R_S I_L + V_C \qquad \text{3-3}$$

$$I_{POW} = \frac{L}{R_P}\frac{dI_L}{dt} - C_F\frac{dV_C}{dt} + I_L \qquad \text{3-4}$$

In matrix format this is

$$\begin{bmatrix} L(1+\frac{1}{R_P}) & R_C C_F \\ \frac{L}{R_P} & -C_F \end{bmatrix}\begin{bmatrix} \frac{dI_L}{dt} \\ \frac{dV_C}{dt} \end{bmatrix} + \begin{bmatrix} R_S & 1 \\ 1 & 0 \end{bmatrix}\begin{bmatrix} I_L \\ V_C \end{bmatrix} = \begin{bmatrix} V_{in} \\ I_{POW} \end{bmatrix} \qquad \text{3-5}$$

In the standard state space format (3-5) can be arranged into the standard form

$$\dot{x} = Ax + Bu \qquad \text{3-6}$$

Where

$$x = \begin{bmatrix} I_L \\ V_C \end{bmatrix}, \qquad A = -\begin{bmatrix} L(1+\frac{1}{R_P}) & R_C C_F \\ \frac{L}{R_P} & -C_F \end{bmatrix}^{-1} \begin{bmatrix} R_S & 1 \\ 1 & 0 \end{bmatrix},$$

$$B = \begin{bmatrix} L(1+\frac{1}{R_P}) & R_C C_F \\ \frac{L}{R_P} & -C_F \end{bmatrix}^{-1}, \qquad u = \begin{bmatrix} V_{in} \\ I_{POW} \end{bmatrix}$$

The output V_{POW} is

$$V_{POW} = V_C + R_C C_F \frac{dV_C}{dt} \qquad 3\text{-}7$$

In matrix format, the outputs are

$$\begin{bmatrix} I_{in} \\ V_{POW} \end{bmatrix} = \begin{bmatrix} I_L \\ V_C \end{bmatrix} + \begin{bmatrix} L/R_p & 0 \\ 0 & R_C C_F \end{bmatrix} \begin{bmatrix} \frac{dI_L}{dt} \\ \frac{dV_C}{dt} \end{bmatrix} \qquad 3\text{-}8$$

In the standard state space format the output is expressed as

$$y = Cx + Du \qquad 3\text{-}9$$

Where

$$C = I + \begin{bmatrix} L/R_p & 0 \\ 0 & R_C C_F \end{bmatrix} \cdot A, \qquad D = \begin{bmatrix} L/R_p & 0 \\ 0 & R_C C_F \end{bmatrix} \cdot B$$

Since A, B, C and D matrices are all known, the standard Matlab/Simulink state space function block can be applied in the SRM motor drive model. The currents I_{POW} and I_L are shown in Figure 3.3. It shows that the current in the inductor, I_L, has much less high frequency harmonics than the inverter bus current I_{POW}. In the experimental system, L=0.63 mH, C_F=16mF, Rs=70mΩ , Rp=2Ω, Rc=0Ω.

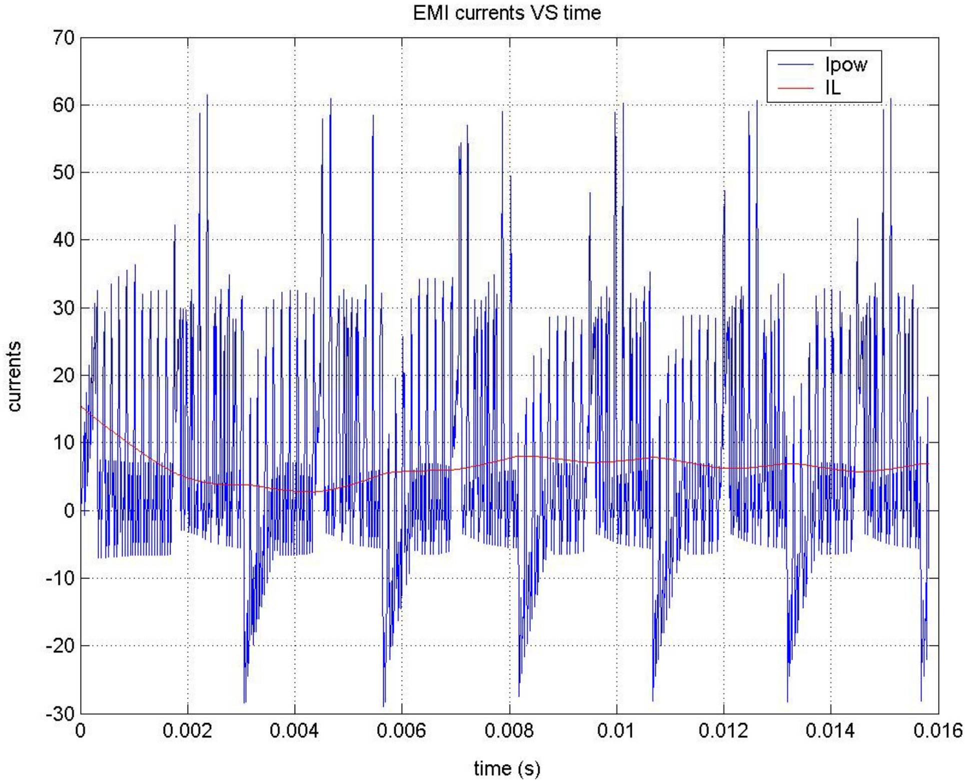


Figure 3.3 I_{POW} and I_L waveforms in the EMI filter

3.1.2 Current Regulator

The function of the current regulator is to regulate the current in the SRM phases. It consists of 4 sub-regulators, one for each phase. The 4 sub-regulators work independently. Each sub-regulator consists of an analog part and a digital part. In the analog part the analog phase current signal is input to three voltage comparators to generate three digital signals that regulate the SRM phase current, control the sense currents, and protect the inverter.

The three voltage comparators with their inputs and outputs are shown conceptually in Figure 3.4. The phase current is input to all three comparators. It is compared to the commanded current in the first comparator, to a low current reference in the second comparator and to an over current reference in the third comparator. The three comparators are named as comp1, comp2 and comp3 respectively. The outputs of the comparators are named as I_low, I_chop, and I_over.

I_low is ‘1’ when the phase current is higher than the low current reference. It is ‘0’ when the phase current is lower than the low current reference and determines when sense pulses can be applied to the SRM. It is needed because the sensing current can only be injected into a phase to obtain the rotor position information after the torque producing current has ideally gone to zero because all sense currents must start from zero current to only depend on the unsaturated phase inductance and not the initial value of the current. Setting the low current reference very low, when I_low is ‘0’, one can safely say that the torque producing current varnishes so that the sensing current can be injected. In the experimental system the low current reference is 5A while the peak torque producing current is 40A.

The control signal I_chop is ‘1’ when the phase current is higher than the commanded current and it is ‘0’ when the phase current is lower than the commanded current by a hysteresis amount. When I_chop is ‘0’ and the phase is in the torque producing region, the two inverter phase switches are turned on to charge the phase windings. When I_chop is ‘1’, one of the two inverter phase switches is turned off to decrease the phase current if the SRM is motoring. If the SRM is generating both switches are turned off to decrease the phase current.

The control signal I_over is an inverter circuit protection signal. It is ‘1’ when the phase current is higher than the over current reference. When this happens a latch is set and the current regulator is shut down immediately. The control signal I_over is ‘0’ when the phase current is lower than the over current reference and has no effect on the current regulator.

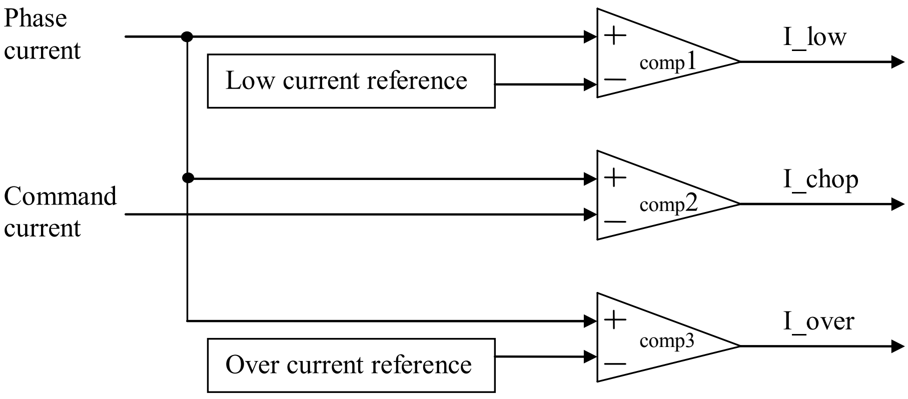


Figure 3.4 the voltage comparators in the current regulator

The digital part of the current regulator is a logic circuit. The register transfer level schematic of the digital part of one of the four sub-regulators is shown in Figure

3.5. Its inputs are:

modin: the injection current pulse signal, also called sensing pulse signal, a 20 KHz, 50% duty cycle signal;

comin: the torque producing command signal, a ‘1’ means input current to produce torque , ‘0’ means do not produce torque and enter the sensing mode;

I_low: output of comp1, ‘1’ means the phase current is higher than the low current reference and thus do not apply sensing pulses to the SRM phase, ‘0’ means the phase current is lower than the low current reference and sensing pulses can be applied to the SRM phase.

I_chop: output of comp2, ‘1’ means the phase current is higher than the commanded current. ‘0’ means the phase current is lower than the commanded current by a hysteresis amount.

I_over: output of comp3, ‘1’ means the phase current is higher than the over current reference, ‘0’ means the phase current is lower than the over current reference.

The outputs of the digital part of one of the four sub-regulators are:

sense: sensing current indication signal, ‘1’ means that the sensing current can be injected, ‘0’ means the opposite;

Q1, Q2: a ‘1’ means turn on the respective inverter switch and ‘0’ means turn off the respective inverter switch;

Shutdown: shutdown command, ‘1’ means the circuit needs to be shut down, ‘0’ means the opposite.

This logic circuit gives the input signals to the gate drives of the two switches. When the command signal, comin, is ‘0’, the pulse signal modin will be routed directly to Q1 and Q2 overriding any other control signals so that sensing current is injected into the phase. When comin is ‘1’, no sensing pulses can be applied to the SRM and the circuit will turn the two switches on and off to keep the phase current constant at the commanded current.

The top RS flip-flop in Figure 3.5 produces the output named sense and is used to make sure that the sensing current is only injected after the torque producing current vanishes. This is accomplished with the I_low signal and a two modulation cycle time delay produced by the two D flip-flops before the RS flip-flop. The bottom RS flip-flop is used to save the over current fault indication. When a '0' to '1' transition of I_over occurs, the negative output, Qn, of the RS flip-flop will be reset to '1'. It will not be set back to '0' until the enable has a '0' to '1' transition. The T flip-flop distributes the switching frequency evenly between the upper and lower switches when the inverter is regulating the phase current in the freewheeling mode where only one switch is turned off to decrease the current.

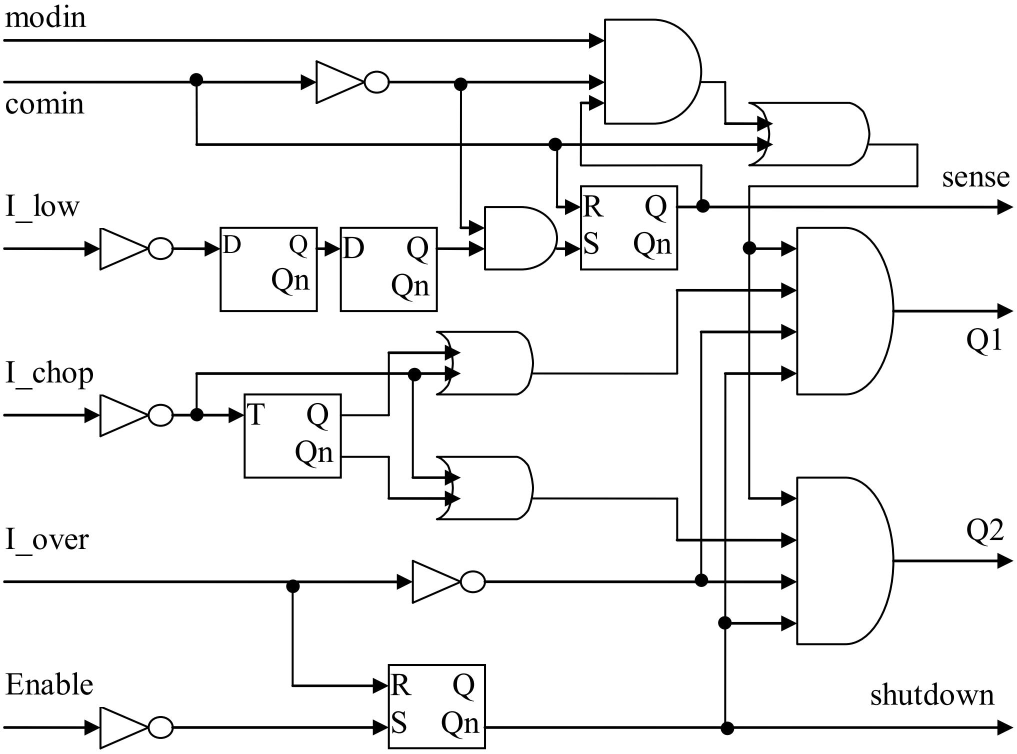


Figure 3.5 the logic circuit of the current regulator

The digital circuit is modeled with Matlab/Simulink and simulated with the rest of the drive system. The simulation results for the input and output signals are shown in Figure 3.6. In the figure, the horizontal axis is time in seconds. All the signals shown in the figure except the phase current are digital. The plotted phase current has been scaled down so its maximum value is 2. The I_over signal is never '1' in simulation because the commanded current is always lower than the over current reference so that I_chop changes to a '1' before I_over does so that the phase current decreases in freewheeling mode. The over current protection is still useful in reality because it will shut down the inverter immediately to protect the inverter should a control error or inverter switch failure occur.

3.1.3 The software implementation block

The software implementation block in the Matlab/Simulink model of the SRM drive system contains the commanded current computation, advance angles computation, a commutator, and two sensorless control strategies.

When the SRM is rotating its phases need to be energized before the rotor reaches the torque-producing region so that the phase current can build up to the commanded current at the beginning of the torque producing region. The phases need to be de-energized before the end of the torque-producing region because the phase current needs some time to decrease to zero and thus to limit the production of torque opposite to the desired torque. The ideal torque-producing region and phase current waveform are shown in Figure 3.7. In the ideal case, the turn-on angle, namely θ_{on_ideal}, is where the stator poles just start to overlap with a pair of rotor poles. The turn-off angle, namely θ_{off_ideal}, is the aligned position. If from θ_{on_ideal} to θ_{off_ideal} the linked flux increases, the phase current will produce positive torque. As described above, the actual turn-on and turn-off angles need to be moved forward to θ_{on} and θ_{off} respectively as shown in Figure 3.8. How much they need to be moved forward depends on the rotor speed, the power supply voltage V_{pow}, and the commanded current.

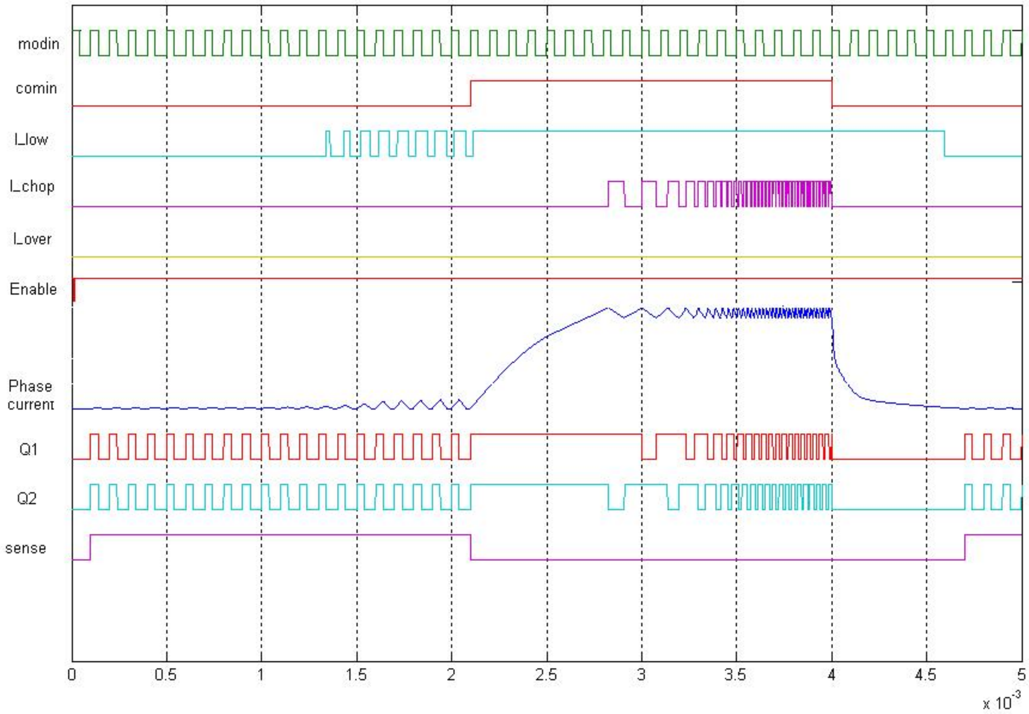


Figure 3.6 the current regulator’s digital input and output signals relative to the scaled phase current

The commutator produces commands to all of the phases based on the estimated rotor position in the sensorless control systems. Conceptually the output of the commutator for a given phase is '1' if the estimated angle input to the commutator is between θ_{on} and θ_{off}. Things are a little more complicated because the rotor position is wrapped into an electrical period, which is from $-\theta_u$ to θ_u. If the turn-on angle θ_{on} is greater than $-\theta_u$, as shown in Figure 3.8a, the commutator energizes the phase in the bold region, in which $\theta_{on} \le \theta \le \theta_{off}$. If θ_{on} is less than $-\theta_u$ and thus wrapped, as shown in Figure 3.8b, the commutator energizes the phase in two separate bold regions, in which $-\theta_u \le \theta \le \theta_{off}$ or $\theta_{on} \le \theta \le \theta_u$.

The commanded current is determined by the torque command. It is simply set as a linear function of the torque command, as in (3-10).

$$I_{comm} = k_{iT} \cdot T_{comm} + I_o \qquad 3\text{-}10$$

Here I_{comm} is the commanded current, k_{iT} is the linear coefficient, T_{comm} is the commanded torque, and I_o is the offset current.

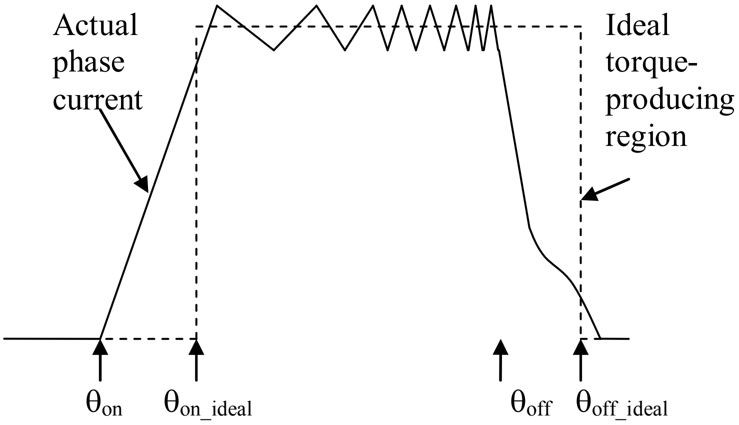


Figure 3.7 the ideal and actual region of torque-producing current

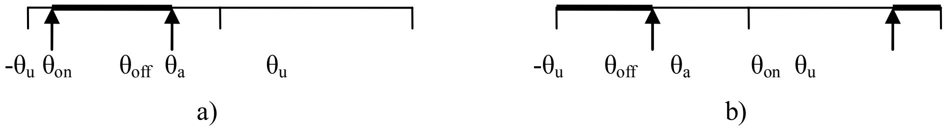


Figure 3.8 the two cases of the on and off angles

The structure of the Simulink model's software implementation block is shown in Figure 3.9. The sensorless control in Figure 3.9 acquires data and estimates the rotor position. This will be explained in more detail in the following chapters.

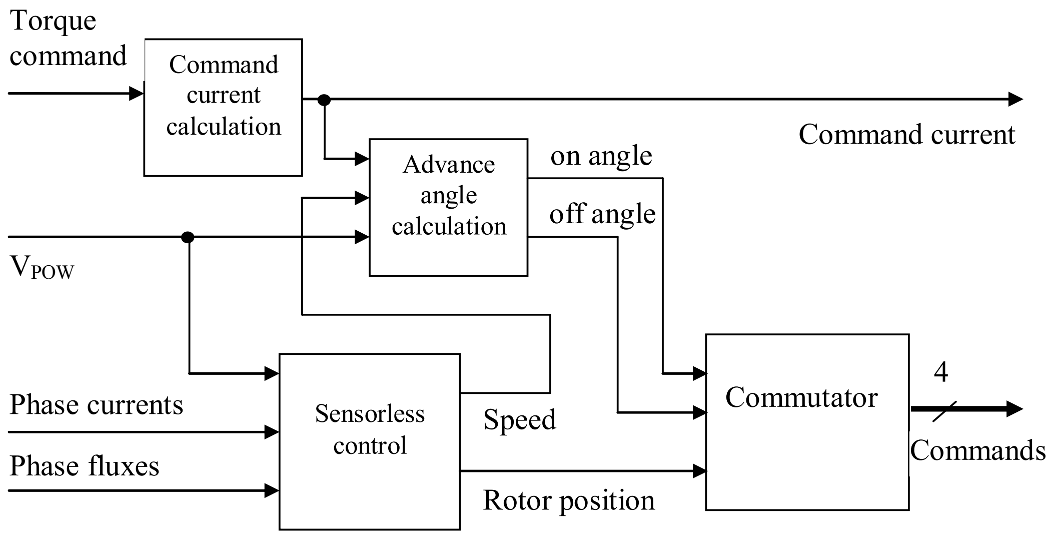


Figure 3.9 the software implementation block structure

3.2 SRM sensorless drive system hardware implementation

The SRM drive system is implemented with three main components, a power inverter, a printed circuit board (PCB) current regulator using a Field Programmable Gate Array (FPGA) chip, and a Digital Signal Processor (DSP) with an analog-to-digital converter (ADC) board, as shown in Figure 3.10. The inverter is implemented with a power printed circuit board bus bar assembly. The current regulator is implemented with a PCB board and an Actel ProASIC APA500K FPGA chip. The microprocessor function unit is implemented in a TI TMS320C6711 floating point DSP. The ADC board is a TI THS1206 evaluation board. It samples the phase currents and/or phase fluxes. The maximum sampling rate is 6 MSPS and the resolution is 12 bits. Since the ADC board can only sample 4 channels, an analog multiplexer is used when 8 channels need to be sampled for the simplified flux model based observer. An HEDS-55X optical encoder position sensor is used to verify the accuracy of the estimated rotor position. The EMI filter and the advance angle algorism are not in the hardware implementation. The detailed parameters of the experimental SRM are shown in appendix IV.

3.2.1 Power Inverter

The torque producing current is high, so the inverter needs to be implemented with PCB bus bars that have rather thick copper. The power electronic switches and diodes need to be mounted on heat sinks to limit their temperature rise due to their switching and conduction losses. To design the power bus bar assembly, four nodes of each inverter phase leg are defined. These nodes are called "power, ground, upper, and lower. Among them, the two nodes, power and ground, are shared by all of the inverter phases. The other two nodes, upper and lower, are independent for each phase and are denoted upperA, lowerA, upperB, lowerB, upperC, lowerC, upperD, and lowerD for phase A, B, C and D respectively, as shown in Figure 3.11a.

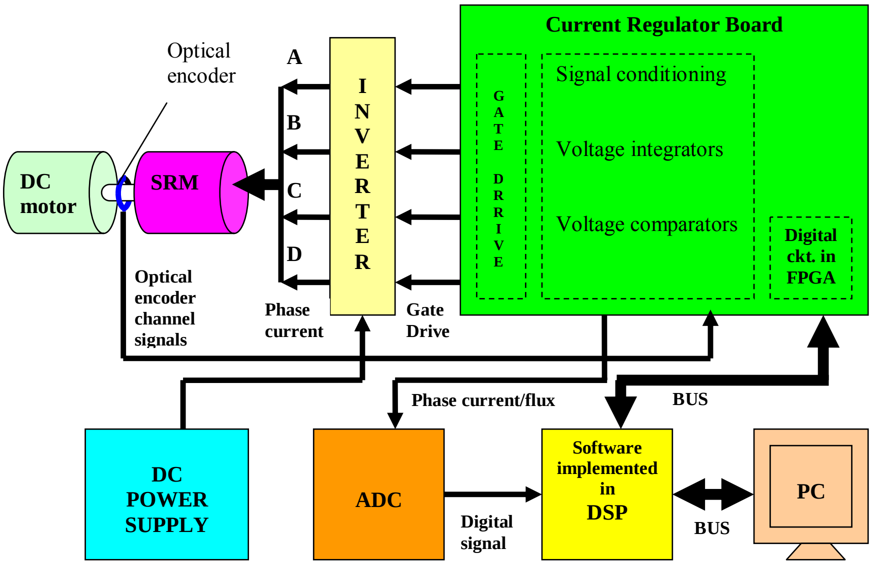
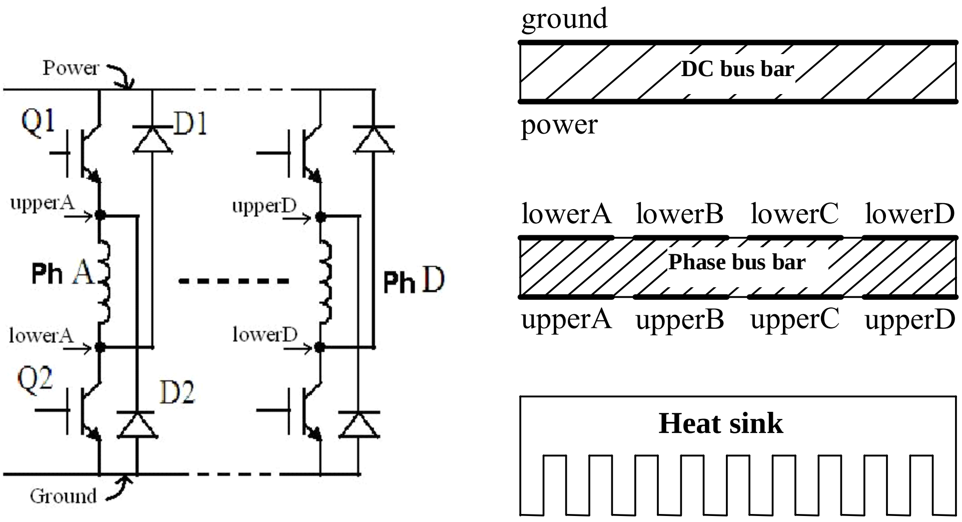


Figure 3.10 the hardware implementation of the SRM sensorless control system

The inverter consists of three layers separated by stand offs, the DC PCB bus bar, the PCB phase bus bar, and the heat sink, which are assembled together in vertical direction from the top to the bottom, as shown in Figure 3.11b. All of the inverter power electronic switches and diodes are mounted on the heat sink to conduct the heat from the device losses away from the devices and into the ambient air. The phase bus bar provides 8 nodes, upperA, lowerA, upperB, lowerB, upperC, lowerC, upperD, and lowerD. The DC bus bar provides two nodes, power and ground. These nodes are connected to the switches and diodes as shown in the schematic Figure 3.11a with Litz wire. The experimental power inverter is shown in Figure 3.12.



a) b)

Figure 3.11 the bus bar assembly

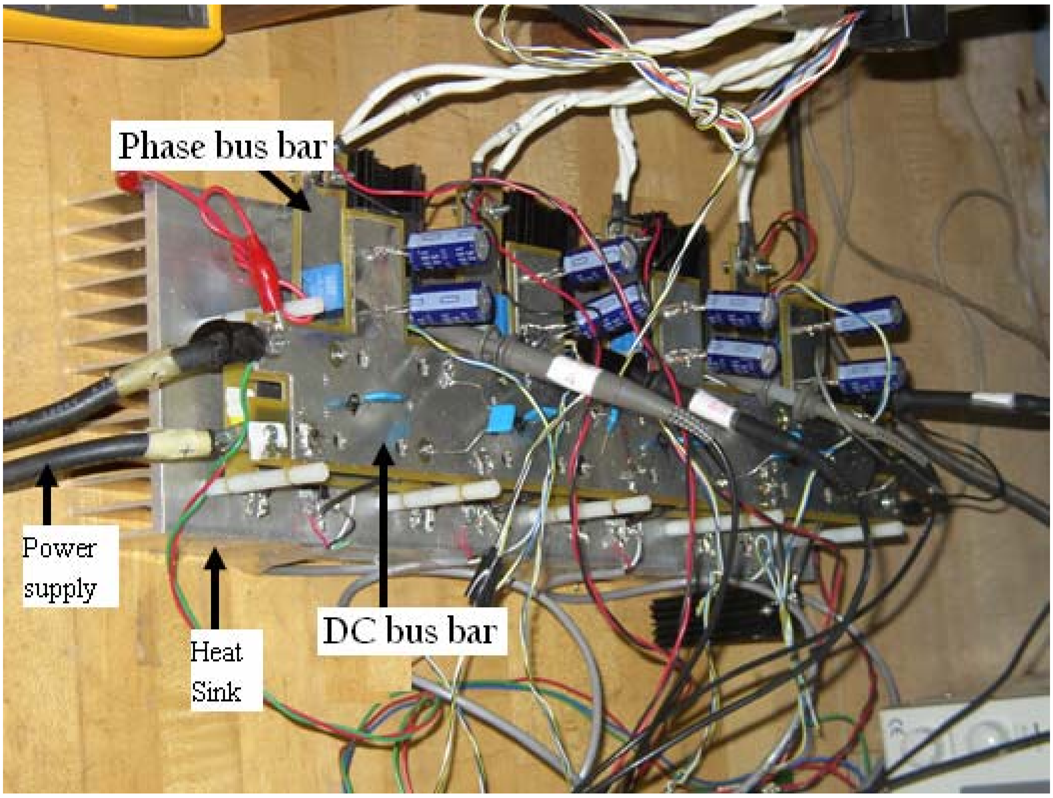


Figure 3.12 the experimental bus bar assembly

3.2.2 The current regulator board

The current regulator PCB board includes signal conditioning, voltage comparators, voltage integrators, low pass filters, analog switches and a FPGA.

3.2.2.1 Signal conditioning

Since the chosen ADC board can only sample voltage signals between 1.5V and 3.5V, every signal being sampled needs to be signal conditioned into this voltage range. The current sensor used is a LEM's HAW-20P. Its conversion table is shown in Table 3.1.

Table 3.1 the conversion table of the current sensor HAW-20P

Current (A, input)	0	10	20	30	40	50 (maximum)
Voltage (V, output)	0	2	4	6	8	10

Assuming the maximum SRM current is 40 A, the voltage range of the current sensor's output is 0 ~ 8 V. After multiplying its output by a gain of 0.2 and then adding a reference voltage of 1.5 V, the voltage range into the ADC board is 1.5 ~ 3.5 V.

Besides the phase current, the demodulated sensing current signal and the flux also need to be signal conditioned. The sensing current is demodulated with a low pass filter and the filter output is level shifted into the voltage 1.5 ~ 3.5 V range. The phase flux is obtained by integrating the phase voltage using an analog integrator. The output of this analog integrator is then level shifted into the 1.5 ~ 3.5 V range before it is sampled by the ADC board.

3.2.2.2 Low speed position demodulator

At low speeds, sensing voltage pulses are applied to the SRM phases that are not being used at that time to produce torque. The resulting phase current is amplitude modulated by the SRM's phase inductance. The amplitude modulated phase current is demodulated with a low pass filter to obtain the position information in the inductance variation. Only a low pass filter is required because both the SRM current and SRM inductance are always positive. The output signal of the filter is named $g(\theta)$. This signal is proportional to the inverse of the phase inductance. The demodulator is described in detail in the following chapter. The filter circuit schematic is shown in Figure 3.13.

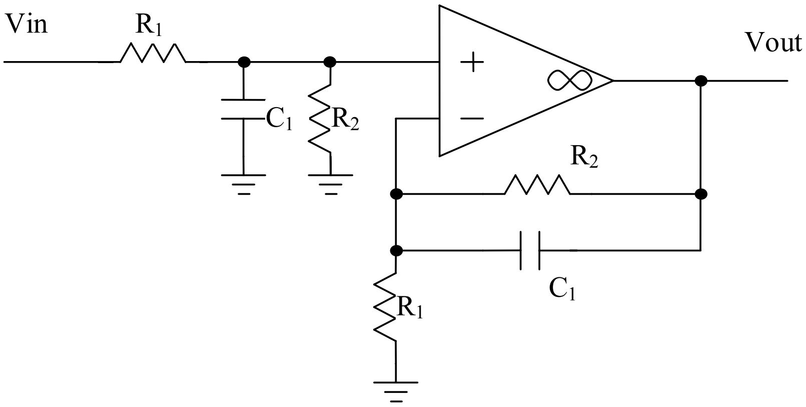


Figure 3.13 the low pass filter and demodulator circuit

The transfer function of the low pass filter is

$$f(s) = \frac{Vout(s)}{Vin(s)} = \frac{R_2/R_1}{1 + R_2 \cdot C_1 \cdot s} \qquad 3\text{-}11$$

3.2.2.3 Voltage integrator (flux generator)

At high speeds position sensing uses the measured phase flux while the SRM is producing torque. No sense pulses are used. The measured phase flux is obtained by integrating the phase voltage. The DC offset voltage of the operational amplifier used in the integrator circuit will create an error over a torque producing period of time if the phase voltage is not high enough or the period is too long. This means the flux generators can not work at zero speed. In addition, it must be insured that the output of the integrator is set to zero each time the current goes to zero since it is known that the flux is zero when the current is zero. The circuit schematic of the integrator is shown in Figure 3.14. The actual phase voltage drop across the winding inductance is the measured phase voltage minus the voltage drop across the internal resistance of the winding. Thus the phase flux is given by (3-12) when the phase is producing torque.

$$\lambda_\phi = \int_0^t (V_\phi - I_\phi \cdot R_\phi)\, dt \qquad 3\text{-}12$$

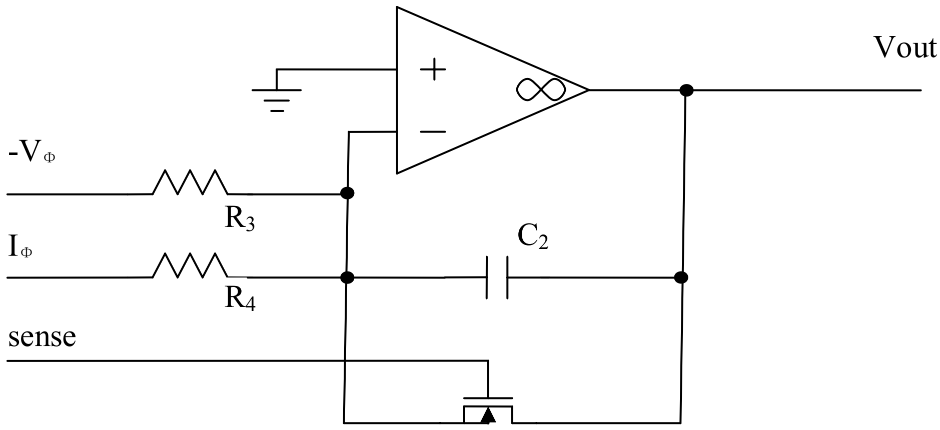


Figure 3.14 the voltage integrator circuit

The relationship between the input and the output of the voltage integrator is

$$Vout(t)=\begin{cases}\frac{1}{R_3\cdot C_2}\cdot\int_0^t\left(V_\phi(t)-\frac{R_3}{R_4}\cdot I_\phi(t)+V_{offset}\right)dt & if \quad sense=0\\ 0 & if \quad sense=1\end{cases} \qquad \text{3-13}$$

The ratio of R_3 to R_4 is determined by the internal series resistance of the phase winding. The output the integrator is the actual flux value times $\frac{1}{R_3\cdot C_2}$ plus the error due to the operational amplifiers offset voltage V_{offset}. The MOSFET in Figure 3.14 is turned on to set the measured flux to zero whenever the current is zero and thus it is know that the flux is zero.

3.2.2.4 Voltage comparator

The voltage comparators are used to generate the digital signals, I_low, I_chop, and I_over. A typical voltage comparator circuit is shown in Figure 3.15. The capacitors are used to eliminate high frequency AC harmonics or noise. The pull up voltage Vcc is chosen to be the digital circuit power supply so that the outputs of the comparators can be directly routed to the digital circuit. The resister R_6 provides hysteresis as shown in Figure 3.16.

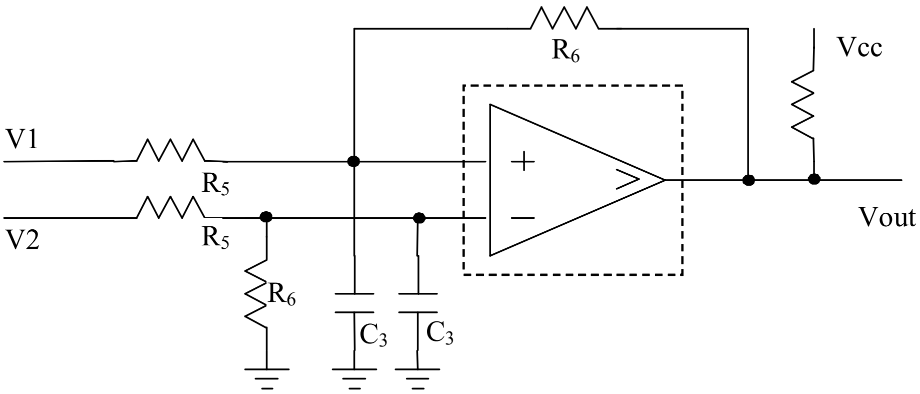


Figure 3.15 the voltage comparator circuit

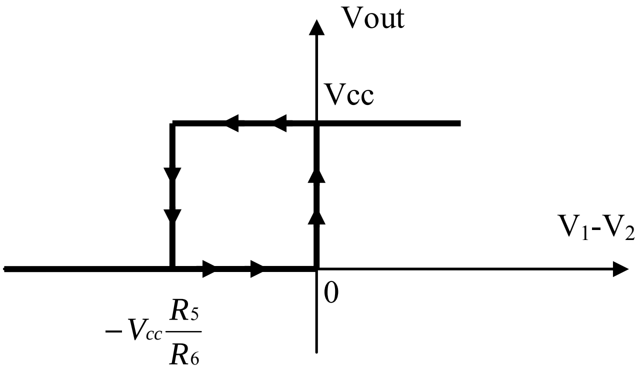


Figure 3.16 the hestersis area of the voltage comparator

3.2.2.5 FPGA implementation of current regulator logic

To realize the digital part of the current regulator, an FPGA chip is chosen and programmed with the VHDL language. The VHDL code is in appendix I.

Besides the 4 copies of the digital circuit shown in Figure 3.5 required for the 4-phase SRM, there are two components in the FPGA chip that have not been described so far. One of them is a data communication interface with the DSP. The other one is a frequency divider.

Through the data communication interface, the DSP obtains the digital sense signals in Figure 3.5 and the outputs of the optical encoder position sensor. The microprocessor sends the enable signal and MorG signal to the FPGA chip. The enable signal is used to enable the control system. When it is ‘1’, the control system is enabled. The MorG signal is used to define the operation mode of the SRM. When it is ‘1’, the SRM works in motoring mode. When it is ‘0’, the SRM works in generating mode. The data communication interface uses a 1 MHz 50% duty cycle clock signal. The interface consists of an 8x2 multiplexer and a 1x8 demultiplexer with storage, as shown in Figure 3.17. The bit assignment is shown in Table 3.2.

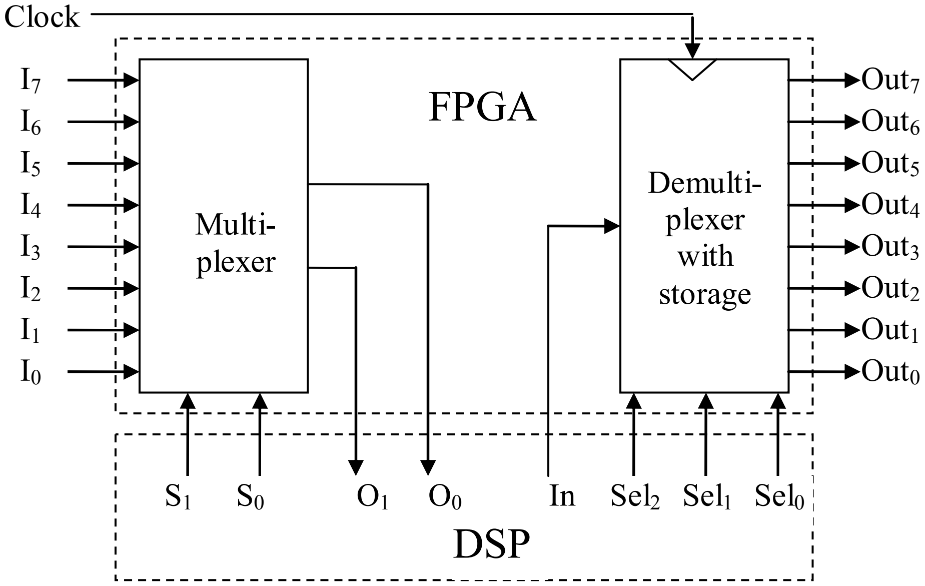


Figure 3.17 the interface between DSP and FPGA

Table 3.2 the interface bit assignment

Interface Signal	Connected to	Connected signal	Description
I_7	FPGA internal	Sense[3]	Sense signal for phase D, generated by the logic circuit
I_6	FPGA internal	Sense[2]	Sense signal for phase C, generated by the logic circuit
I_5	FPGA internal	Sense[1]	Sense signal for phase B, generated by the logic circuit
I_4	FPGA internal	Sense[0]	Sense signal for phase A, generated by the logic circuit
I_3	FPGA internal	shutdown	Over current fault signal, generated by the logic circuit
I_2	FPGA internal	I	Optical encoder channel I signal
I_1	FPGA internal	A	Optical encoder channel A signal
I_0	FPGA internal	B	Optical encoder channel B signal
S_1	DSP Timer 1	Timer1	Select line bit #1, generated by DSP timer 1
S_0	DSP McBSP0	DX	Select bit #0, generated by DSP McBSP0

Table 3.2 the interface bit assignment (continued)

O_1	DSP McBSP0	CLKS	Output line bit #1, read by DSP McBSP0
O_0	DSP McBSP0	DR	Output line bit #0, read by DSP McBSP0
In	DSP McBSP0	CLKR	Input signal, generated by DSP McBSP0
Sel_2	DSP McBSP0	FSR	Select line #2, generated by DSP McBSP0
Sel_1	DSP McBSP0	CLKX	Select line #1, generated by DSP McBSP0
Sel_0	DSP McBSP0	FSX	Select line #0, generated by DSP McBSP0
Out_7	FPGA internal	IorV[1]	Select line # 1 of an analog multiplexer
Out_6	FPGA internal	IorV[0]	Select line # 0 of an analog multiplexer
Out_5	FPGA internal	Enable	Logic circuit enable signal generated by the DSP
Out_4	FPGA internal	MorG	MorG signal in the logic circuit, generated by DSP
Out_3	FPGA internal	Comm[3]	Figure 3.5 Comin signal generated by the DSP for phase D
Out_2	FPGA internal	Comm[2]	Figure 3.5 Comin signal generated by the DSP for phase C
Out_1	FPGA internal	Comm[1]	Figure 3.5 Comin signal generated by the DSP for phase B
Out_0	FPGA internal	Comm[0]	Figure 3.5 Comin signal generated by the DSP for phase A
Clock	FPGA internal	clock	1 MHz 50% duty cycle to drive the logic circuit

The frequency divider generates a 20 KHz 40% duty cycle pulse signal from the 1 MHz 50% duty cycle clock signal. The 20 KHz pulse signal is used to control the power electronics switches in the inverter when the sensing current needs to be injected. The frequency is chosen as high as possible while insuring the sensing current is high enough to measure and low enough to not produce significant torque or iron saturation. A high sensing frequency allows a demodulator low pass filter with a higher break frequency which in turn reduces the filter’s delay error. The register transfer level schematic of the frequency divider is shown in Figure 3.18.

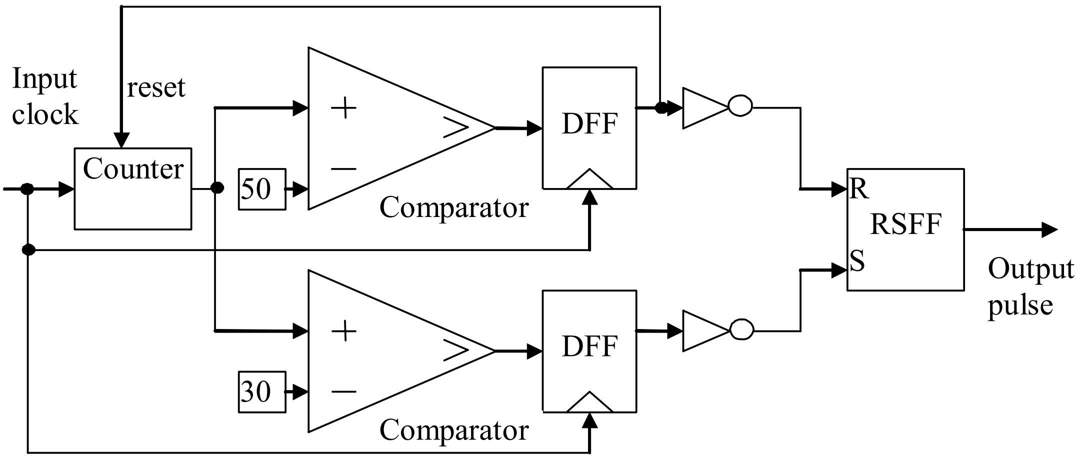


Figure 3.18 the clock divider's register transfer level schematic

3.2.2.6 The printed circuit board

The printed circuit board schematic circuit is drawn in Electronics Workbench's Multisim and it is then converted into an input file for the Ultiboard PCB layout software. The 3D view of the PCB board generated by the Ultiboard Software is shown in Figure 3.19. It consists of 4 copies of the circuit shown in Figure 3.20, each copy for a different SRM phase. It also consists of 4 gate drives circuits, the FPGA and interface connectors. The signal flow of one phase on the print circuit board is shown in Figure 3.20.

3.2.3 DSP implementation

The flow chart of the DSP C++ software is shown in Figure 3.21 and the main part of the code is in appendix II. At the beginning, the program initializes all the parameters, disables the current regulator, clears all storage matrices, chooses the low speed sensorless strategy, enters the start mode that is used to find the initial rotor position, and then starts the timer for a software interrupt that calls function 'call_microcontroller' periodically.

The software interrupt is activated by the timer every t_{sample} seconds. When the interrupt occurs, the function call_microcontroller is called. In the first t_{start} seconds, the motor works in the start mode to find out the initial rotor position. In the start mode, there is no command sent to any phase to produce torque so the SRM xremains at standstill. The time t_{start} needs to be long enough for the observer to converge to the initial position. After t_{start}, the low speed sensorless strategy is used to control the SRM. The sensorless control must work from zero speed and from zero torque to rated torque. With the low speed strategy, the sense signals in Figure 3.5 for each of the phases are used to determine which phases are idle. The sensing currents are demodulated, sampled, and input to the microprocessor where an error function generates an error signal to drive the observer. The rotor position and speed are then estimated.

When the low speed sensorless strategy is being used and the estimated speed exceeds 100 rad/s, the controller changes to the high speed sensorless strategy. If the high speed sensorless strategy is being used and the estimated speed drops to less than 50 rad/s, the controller changes to the low speed strategy. In between the two speeds, the present control strategy will be used. The two strategies will be described in more detail in the following chapters.

In the high speed sensorless strategy, the phase currents and phase fluxes are measured and sampled by the microprocessor. A simplified analytical flux model calculates the phase fluxes and the difference between the calculated fluxes and the measured fluxes is an error that drives the observer. The observer then estimates the rotor position and speed.

After the rotor position is estimated, it is input to the commutator. The commutator's outputs, are the commands to each phase to produce torque or not produce torque according the estimated rotor position.

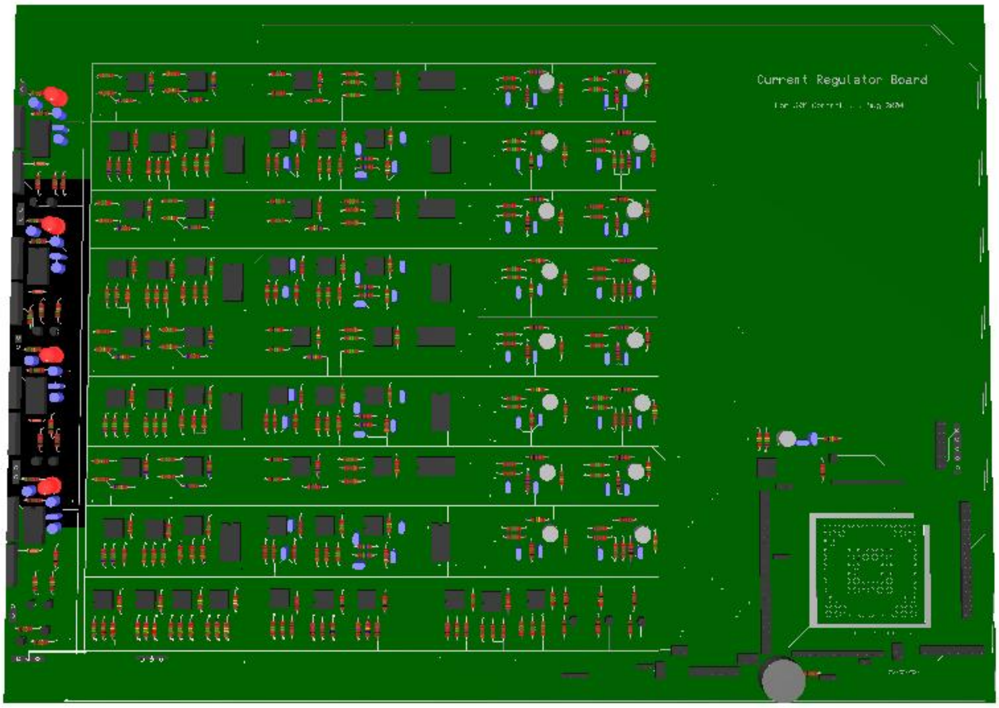


Figure 3.19 the current regulator board

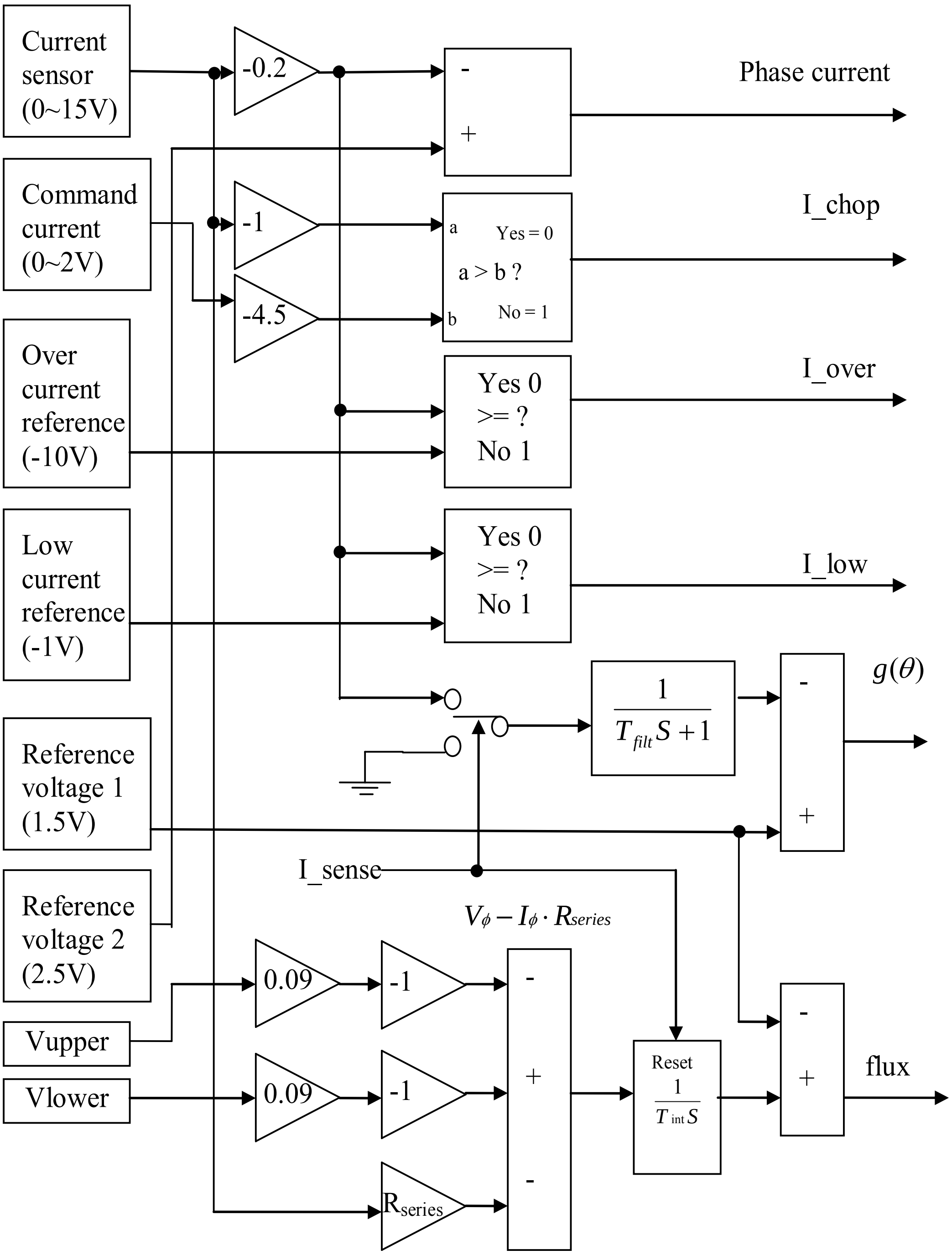


Figure 3.20 the signal flow of the printed circuit board

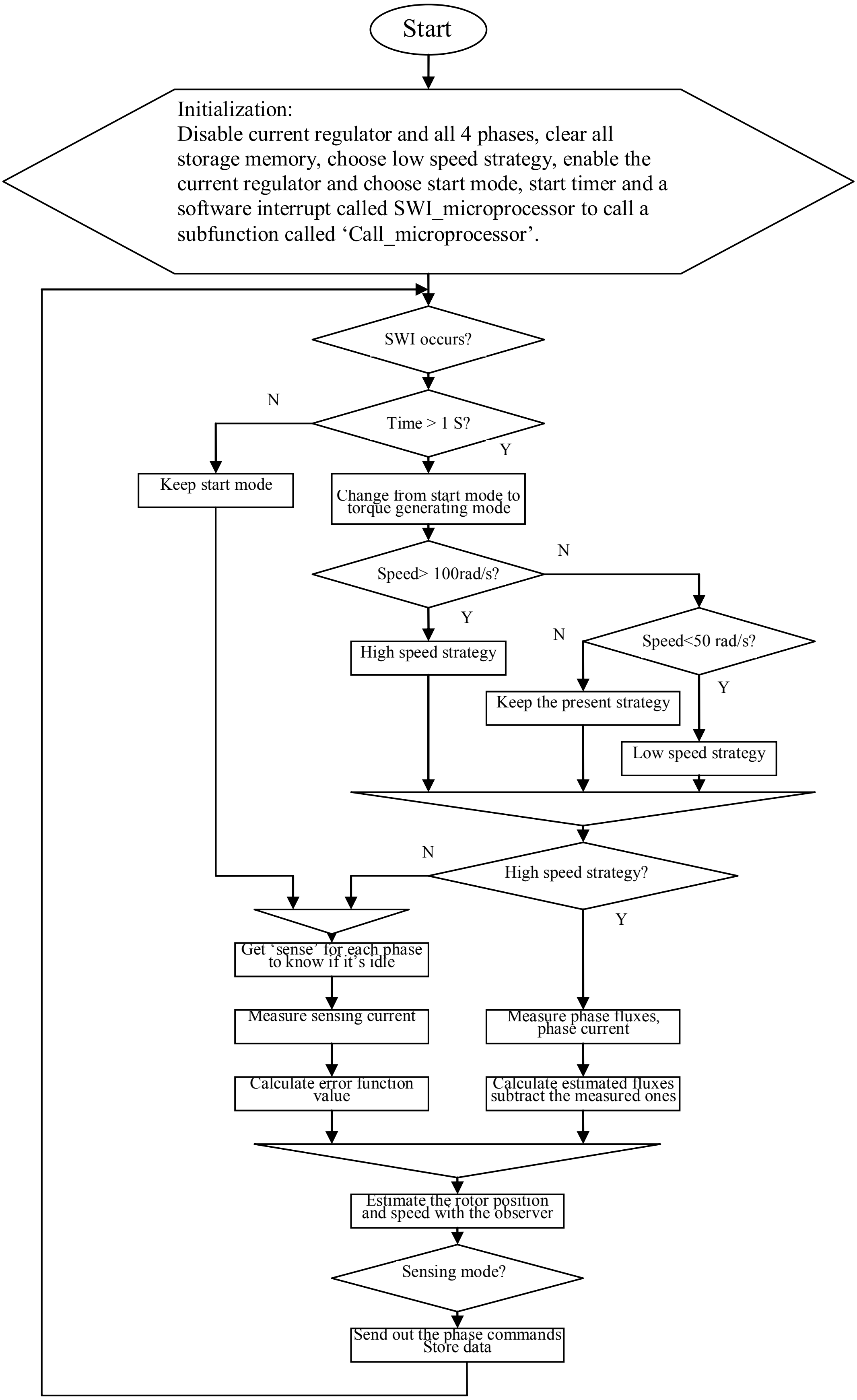

Start
Initialization:
Disable current regulator and all 4 phases, clear all storage memory, choose low speed strategy, enable the current regulator and choose start mode, start timer and a software interrupt called SWI_microprocessor to call a subfunction called 'Call_microprocessor'.
SWI occurs?
Time > 1 S?
N
Y
Keep start mode
Change from start mode to torque generating mode
Speed> 100rad/s?
N
Y
High speed strategy
Speed<50 rad/s?
N
Y
Keep the present strategy
Low speed strategy
High speed strategy?
N
Y
Get 'sense' for each phase to know if it's idle
Measure sensing current
Calculate error function value
Measure phase fluxes, phase current
Calculate estimated fluxes subtract the measured ones
Estimate the rotor position and speed with the observer
Sensing mode?
Send out the phase commands Store data

Figure 3.21 the flow chart of the program in DSP

Chapter 4 Inductance profile demodulator based state observer sensorless control

4.1 Sensorless control review

Rotor position information plays a critical role in the control of the SRM. Conventionally, a separate position sensor, either a resolver or an optical encoder, is used to get this information. A resolver is a rotating transformer where the coupling between the primary winding on the rotor and the two secondary windings on the stator depends on the shaft position. An optical encoder is mounted on the shaft and with the shaft turning the optical encoder generates a pulse output voltage each time the rotor rotates through a fixed angle on one or more channels. The position sensors add complexity and cost to the SRM drive system and reduce its reliability.

In order to overcome the drawbacks of the position sensors, a number of methods have been proposed to control the SRM without position sensors. These sensorless control strategies can be divided into three categories. In the first category, small currents are injected into the idle phases. An example of the injected currents is shown in Figure 4.1. The currents are so small that they do not produce noticeable torque and the iron does not saturate. In this case the relationship between the current and the corresponding inductance is independent of the current. The small currents are measured and used to estimate the rotor position since the currents contain the rotor position information. In the second category, the torque producing currents are used to estimate the rotor position. A typical torque producing current is shown in Figure 4.1. Since the torque producing current is relatively high the iron typically saturates due to the nature of SRM. This effect of iron saturation needs to be considered to obtain the correct rotor position from the current information. The third category of sensorless control methods has not been proposed so far. It is a mixed method that not only injected currents but also the torque producing currents are used to estimate the rotor position. It can be chosen when the application requires the SRM to work at zero speed, low speeds and high speeds. The classification of the sensorless strategies is shown in Figure 4.2.

4.1.1 The first category, injected currents are used

Several methods have been proposed to use small injected currents to obtain the

SRM's rotor position [12-15]. All of these methods are based on the fact that the phase inductance of the SRM is a function of its rotor position independent of the phase current if the current is small. This is true if the injected current is low enough that the iron does not saturate. This group of methods has advantages and disadvantages. Advantages: 1) They work at low speeds, zero speed and starting. 2) They do not need to consider the effect of iron saturation that makes the inductance profile nonlinear with the phase current. 3) They do not need to consider the complicated flux model so that the real time computations required to implement them on a microprocessor can be done rapidly. Disadvantages: 1) They have difficulty working at high speeds. This is because the frequency of the injected currents is limited by the SRM phase inductance and the injected currents may need to go through low pass filters that generate a time delay. For the approach developed here the delay time introduced by such a filter results in an ever increasing position error as the speed increases. Another reason these injected current methods can not work at high speeds is that the injected current time windows become small at high speeds so that the currents do not have enough information about the rotor position. 2) Some strategies need additional hardware to inject the sense currents. 3) Some strategies need memory to store look up tables that contain injected current amplitude versus rotor position data.

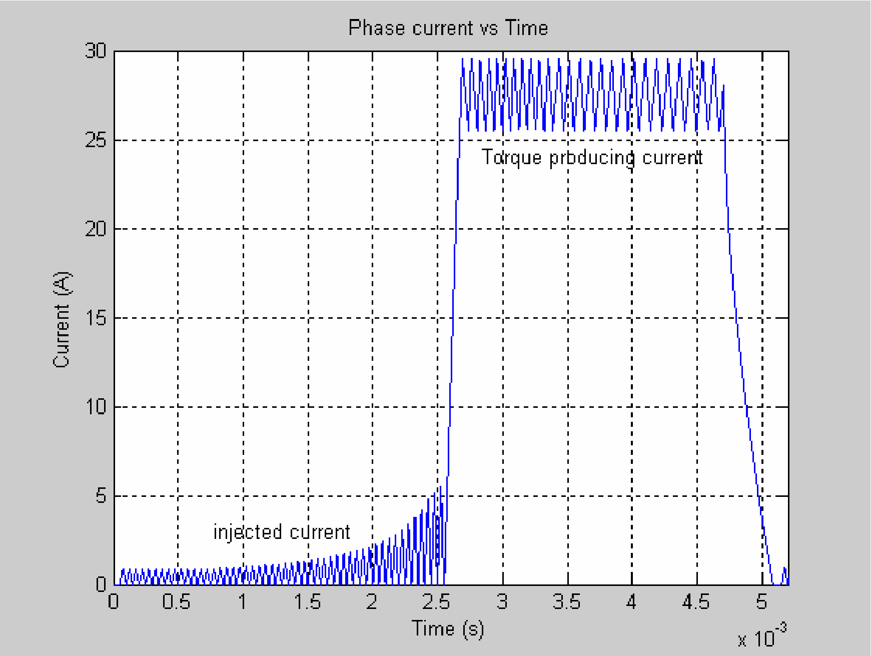


Figure 4.1 an example of the injected current and torque producing current

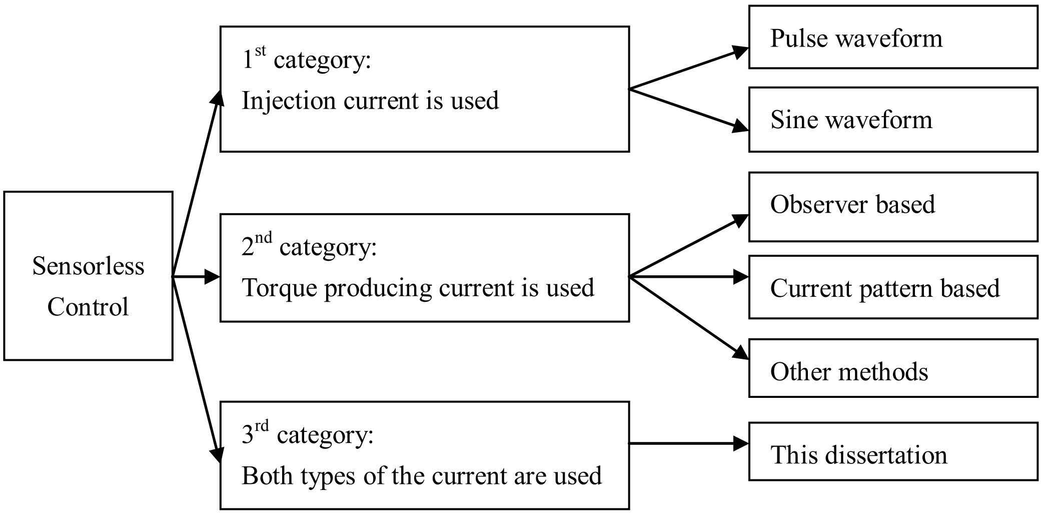


Figure 4.2 the classification of sensorless control strategies

A sinusoidal current was injected by Brosse A. et al. into the SRM through a separate converter [12]. The induced voltage signal depends on the rotor position. This voltage was measured and its power was evaluated. The value was then translated into the rotor position through a prior stored look up table that contained the signal power values at a number of rotor positions. An observer and PI controller were used to get the rotor position. This method gave continuous rotor position information, but it needs additional hardware to inject the sinusoidal current.

A pulse voltage was applied to the idle phases by Harris W. D. et al. [13], Suresh G. et al. [14], and Gao H. et al. [15]. The resulting current was measured and used to calculate the rotor position in [13]. An observer was also used to offer high accuracy position estimation, but it needed memory to store look up tables. The current was demodulated into the rotor position using an envelop detector which worked as a counter counting the successive current peaks [14]. This significantly increases the required time for the observer to converge to the correct angle and does not work at zero speed. Thus at zero speed each phase is excited and then the amplitudes of all phase currents are compared to know the rotor position roughly [15]. Obviously torque can not be produced continuously at zero speed with this method. This simple method for estimating the rotor position working at standstill is adequate to start the SRM but not for operating for significant periods of time at stand still.

4.1.2 The second category, the torque producing current is used

The methods in this category only use the torque producing current. State observers or phase current patterns are used to identify the rotor position. These methods also have advantages and disadvantages. Advantages: 1) They work over a large speed range including high speeds. 2) They do not need additional hardware for current injection. They only need motor terminal measurements. Disadvantages: 1) They have inherent problems working at low speeds, especially at zero speed. A small dc offset can cause voltage integrators to fail at zero to low speeds since these methods integrate phase voltage to measure phase flux. 2) The current pattern does not change quickly enough to determine a continuous rotor position for those methods that use current patterns. 3) In many of these methods intensive computation is required to complete flux calculations or they need memory to store a flux model. 4) Iron saturation needs to be considered in this case because the torque producing current is typically high enough to cause iron saturation.

In general there are two groups of methods to realize the rotor position estimation using the torque producing currents. In the first group, state observers are used to estimate the rotor position [16-21]. In the second group, current patterns, the increasing and decreasing phase current slopes, are used to obtain the phase inductance and hence the rotor position [22-26]. There are several other practical methods that also use the torque producing current to realize sensorless control of the SRM [27-29].

4.1.2.1 Observer based sensorless control

Lumsdaine A. et al. used state observers to estimate the SRM's rotor position. In their observers, the states are the phase fluxes, the rotor position and the rotor speed [16, 17]. The phase currents were measured and estimated by a SRM flux model. The difference between the measured and the estimated fluxes drove the state observers. Several practical observers were given and stability was proven. The flux model used in the observers was Fourier series based. Husain I. et al. used sliding mode state observers to estimate the rotor position [18-20]. The rotor position and the rotor speed were used as states in the observers. The phase voltage was integrated to obtain the flux digitally and then the phase current was estimated with an analytical flux model. The difference between the measured current and the estimated current was computed and drove the stator observer. A geometry based simplified analytical flux model of

SRM was used in [20]. Its simplicity made it possible to run in a real-time controller. The phase voltage was integrated to get the estimated flux and the actual flux was obtained by a simple flux model, which is an exponential function of rotor position and current. Then the difference was used to drive a sliding mode observer to obtain the rotor position. In these papers, the flux was obtained by integrating the phase voltage digitally. Due to the high frequency of the phase voltage when the current chops, the sampling rate needs to be very high in this case. Yang I. -W. et al. also used a state observer in which phase currents and the rotor speed were states [21]. The difference between the estimated phase current and the measured phase current drove the observer. Two observers, a sliding mode observer and a binary mode observer were proposed and verified experimentally.

This group of methods gives continuous and smooth rotor position information and good stability with sophisticated control system gains.

4.1.2.2 Chopping current pattern based sensorless control

The increasing and decreasing slopes of the chopping current were used to estimate the rotor position in [22-26]. Suresh G. et al. proposed an equation in that the rotor position was unknown and the slopes of the phase current and other terminal measurements are known variables [22]. Fahimi B. et al. studied these methods at every speed range and gave a practical method to compute the rotor position [23]. Salmasi F. R. et al. built another equation to solve for the rotor position for low speed applications [24, 25]. Gao H. et al. proposed a method that worked at low speeds [26]. Back EMF was detected by the slopes of the phase currents, and then the current command was adjusted to assure that the currents were applied on either positive slopes or negative slopes of the inductance profile depending on generating or motoring mode. This group of methods provides simple control that is relatively easy to implement in a real-time controller. No additional hardware was required because only SRM terminal measurement of voltage and current are needed. But these methods suffer from problems with high frequency noise in the phase current. They are difficult to implement in high speed applications because they typically differentiate the phase current and thus amplify high frequency noise if the differentiating circuits have large bandwidth.

4.1.2.3 Other methods using the torque producing currents

There are also some other methods to realize sensorless control of SRM using the torque producing currents that do not fall into the above categories. Lyons J. P. et al. integrated the phase voltage to get the actual flux and compared it with a known flux value at a reference rotor position [27]. When the actual flux of a phase is equal to the known flux value, the rotor then is at the reference position with regard to the phase. Mondal S. K. et al. gave a current command to a phase according to the current patterns of other phases [28]. Mese E. et al. used an artificial neural network (ANN) to realize sensorless control [29]. The flux linkage and phase current were input to the neural networks, and the rotor position was the output of the networks. Training data were obtained from a SRM flux model or experiments.

4.2 Proposed control strategies

With the development of microprocessor and DSP technology, computation intensive and accurate control strategies are now feasible. Since none of the methods described above can work well over the whole SRM speed range, more than one control strategy is required in a large speed range application. Generally, for low speed application, current injection has inherent advantage for starting from standstill with rated torque. For high speed applications, the sensorless control strategies based on the torque producing currents are better choices. That is because in this case there is no limitation introduced by the choice of injection frequency and the torque producing currents have longer time windows at high speeds, and hence they provide more information to obtain the rotor position. In this dissertation, a sensorless control system that utilizes a strategy for zero and low speeds and a strategy for high speeds is proposed. The zero speed and low speed strategy uses injected currents, while the high speed strategy uses the torque producing current. Since this control system uses both the injected current and the torque producing current, it falls into the third sensorless control category.

The low speed strategy is described in this chapter and the high speed strategy will be described in the following chapter.

At zero to medium speeds, a pulse voltage signal is applied to the idle SRM phases to generate sensing current or injected current. The injected current is modulated by the SRM's phase inductance and contains the rotor position information. . If the actual rotor position is not equal to the estimated rotor position, then the injected current amplitude will be different from the computed current amplitude. The difference generates an error signal through a deliberately defined error function. The error signal then drives a Luenberger observer. This method including the demodulation of the SRM’s modulated phase current, the proposed error function, and observer has been named the inductance profile demodulator based observer. It works at zero speed because for any SRM rotor position there always are idle phases that the sensing current can be injected into. It has difficulty working at high speeds. The reason is in part because the injected current is demodulated using a low pass filter whose break frequency is determined by the frequency of the injected current. The demodulator low pass filters have an inherent time delay determined by their break frequency. At high speeds, the time delay generates a position error proportional to speed that makes the sensorless control fail. Another contributor to the inductance profile demodulator based observer sensorless control failure is that the time windows for injecting current becomes smaller at high speeds so that they do not contain enough information for error function and observer to figure out the rotor position.

4.3 The state observer

The electromechanical operation of a SRM can be modeled by

$$\frac{d\theta}{dt} = \omega \qquad 4\text{-}1$$

$$\frac{d\omega}{dt} = -\frac{B}{J}\omega + \frac{1}{J}[T_e - T_L] \qquad 4\text{-}2$$

where θ is the rotor position, ω is the rotor speed. B is the viscous damping, J is the inertia, T_e is the electrically generated torque, and T_L is the mechanical load torque.

For simplicity, viscous damping is lumped into T_L. It is also assumed that T_e is equal to T_L, which means the motor is at steady state and running at a constant speed. This assumption is reasonable because the electrical time constants are usually much less than the mechanical time constants. With these considerations equation (4-2) is simplified as

$$\frac{d\omega}{dt} = 0 \qquad 4\text{-}3$$

The corresponding observer model

$$\frac{d\hat{\theta}}{dt} = \hat{\omega} + H_1 \cdot f(\theta, \hat{\theta}) \qquad 4\text{-}4$$

$$\frac{d\hat{\omega}}{dt} = H_2 \cdot f(\theta, \hat{\theta}) \qquad 4\text{-}5$$

where H_1 is the proportional gain for the position, H_2 is the proportional gain for the speed, $\hat{\theta}$ is the estimated rotor position, and $\hat{\omega}$ is the estimated rotor speed. The function $f(\theta, \hat{\theta})$ is the error signal reflecting the difference between the estimated rotor position and the actual rotor position. A block diagram of the observer is shown in Figure 4.3.

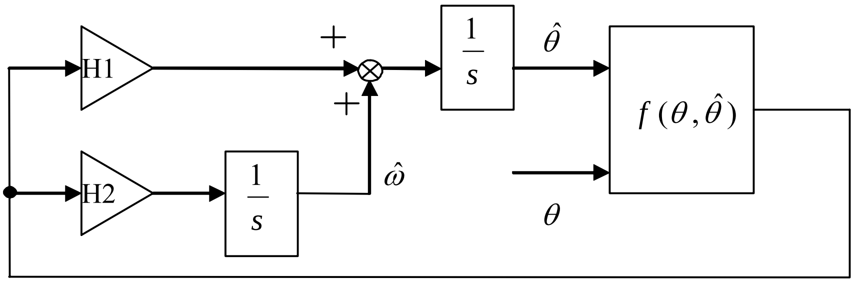


Figure 4.3 the block diagram of the inductance profile demodulator based observer

Subtracting (4-1) from (4-4) and (4-3) from (4-5) gives the observer's error dynamics.

$$\frac{de_\theta}{dt} = e_\omega + H_1 \cdot f(\theta, \hat{\theta}) \qquad 4\text{-}6$$

$$\frac{de_{\omega}}{dt} = H_2 \cdot f(\theta, \hat{\theta}) \tag{4-7}$$

where the rotor position error $e_{\theta} = \hat{\theta} - \theta$, and the rotor speed error $e_{\omega} = \hat{\omega} - \omega$. In matrix format, it becomes

$$\begin{bmatrix} \frac{de_{\theta}}{dt} \\ \frac{de_{\omega}}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e_{\theta} \\ e_{\omega} \end{bmatrix} + \begin{bmatrix} H_1 \\ H_2 \end{bmatrix} \cdot f(\theta, \hat{\theta}) \tag{4-8}$$

The challenge of the observer design is the error function. At zero and low speeds a sensing current is injected into the idle phases and this injected current contains the rotor position information that is used to generate the error fuction.

4.4 Error function definition

The phase inductance L(θ) is a function of rotor position when the iron is not saturated as shown in Figure 4.4. . It has its maximum value at the aligned position and its minimum value at the unaligned position. When a fixed duration pulse voltage is applied to the phase, a current like the one shown in Figure 4.4 is obtained. The cycle average value of the current is called g(θ), where θ is the rotor position, as shown in Figure 4.4.

To obtain the relationship between the inductance function L(θ) and g(θ), basic circuit theory is applied.

$$V(t) = L(\theta) \cdot \frac{dI(t)}{dt} \tag{4-9}$$

Here V(t) is the pulse voltage, I(t) is the injected current, and t is time. It is assumed that the rotor position does not change during a pulse period because the mechanical time constant is much greater than the pulse period. Solving I(t) in terms of V(t) gives (4-10).

$$I(t) = \frac{1}{L(\theta)\cdot}\int_0^t V(t)\,dt \qquad 4\text{-}10$$

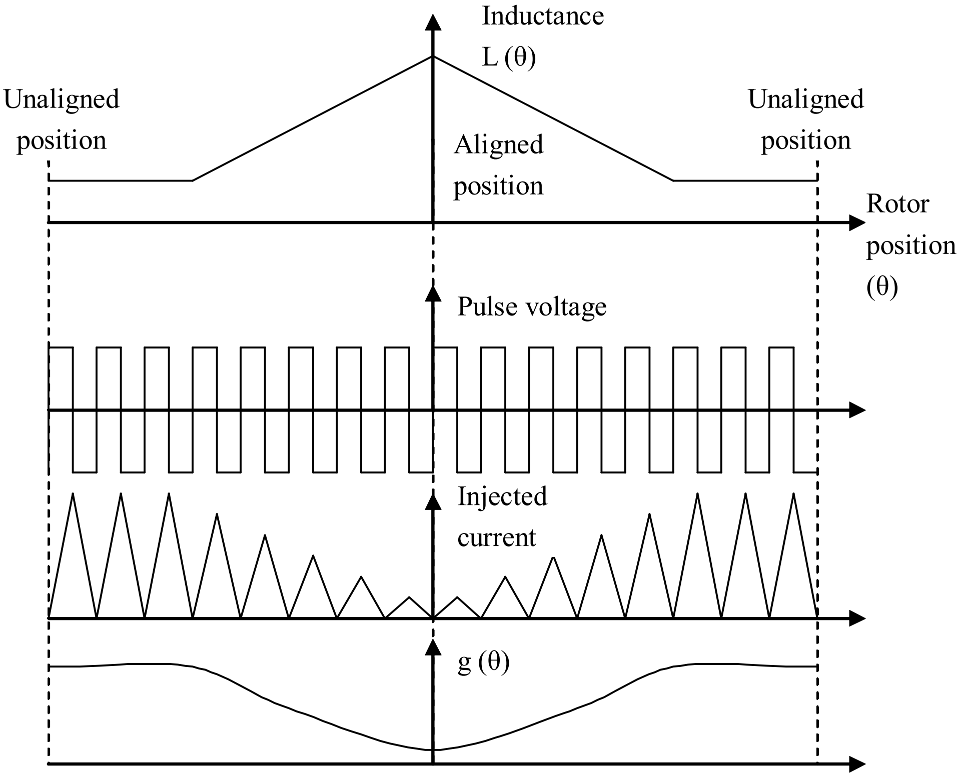


Figure 4.4 the injected current

The injected current is a repetitive triangle waveform signal. The peak of the triangle current I(θ) is

$$I(\theta) = \frac{Vs \cdot D \cdot T}{L(\theta)} \qquad 4\text{-}11$$

where Vs is the peak of the pulse voltage and approximately equal to the inverter's dc power supply voltage, T is the period, and D is the duty cycle. The result in (4-11) is only valid if the current starts from zero. To make sure the current goes to zero after each period, the duty cycle D should not be greater than 50%. The average current value of the triangle I_{ave} (θ) is (4-12).

$$I_{ave}(\theta) = \frac{Vs \cdot D^2 \cdot T}{L(\theta)} \qquad 4\text{-}12$$

The g(θ) function is defined to be equal to I_{ave}(θ).

$$g(\theta) = I_{ave}(\theta) \qquad 4\text{-}13$$

The g(θ) for the various phases are named g_1(θ), g_2(θ), g_3(θ), g_4 (θ), for phase A, B, C and D respectively. They are not only given by (4-12) but can also be measured by measuring the injected current in each of the 4 phases. If the rotor position is estimated as $\hat{\theta}$, the g (θ) value can be estimated through

$$g(\hat{\theta}) = \frac{Vs \cdot D^2 \cdot T}{L(\hat{\theta})} \qquad 4\text{-}14$$

The estimated $g(\hat{\theta})$ are named as $g_1(\hat{\theta})$, $g_2(\hat{\theta})$, $g_3(\hat{\theta})$, and $g_4(\hat{\theta})$ for phase A, B, C and D respectively. They are calculated using (4-12) with the estimated rotor position.

$$g_i(\hat{\theta}) = \frac{Vs \cdot D^2 \cdot T}{L_i(\hat{\theta})} \qquad i = 1,2,3,4 \qquad 4\text{-}15$$

Where $L_i(\hat{\theta})$ is the inductance function of the ith phase.

The error function $f(\theta,\hat{\theta})$ is defined as (4-16)

$$\begin{aligned} f(\theta,\hat{\theta}) = {} & g_1(\theta)\cdot g_2(\hat{\theta}) - g_2(\theta)\cdot g_1(\hat{\theta}) + \\ & g_2(\theta)\cdot g_3(\hat{\theta}) - g_3(\theta)\cdot g_2(\hat{\theta}) + \\ & g_3(\theta)\cdot g_4(\hat{\theta}) - g_4(\theta)\cdot g_3(\hat{\theta}) + \\ & g_4(\theta)\cdot g_1(\hat{\theta}) - g_1(\theta)\cdot g_4(\hat{\theta}) \end{aligned} \qquad 4\text{-}16$$

assuming none of the phases are producing torque. The error function value versus the rotor position θ is plotted in Figure 4.5 when the estimated rotor position $\hat{\theta}$ is 2 mechanical degrees greater than θ (i.e. $e_\theta = 2^\circ$).

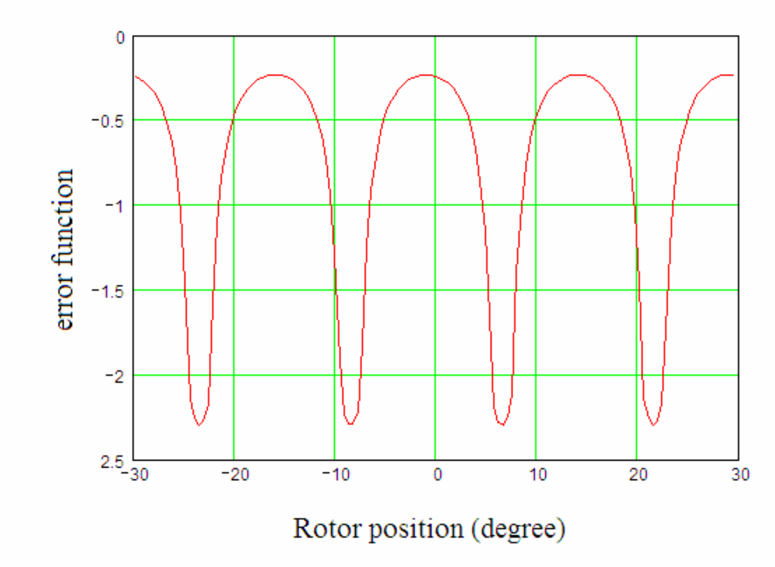


Figure 4.5 error function value versus the rotor position

It is seen that the value of the error function takes on different values depending on the rotor's position for the same error but that its value is negative in the whole electrical period. It is also verified that the error function value is always negative when the estimated rotor position is greater than the actual rotor position and that it is monotonic with the angle error. This assures that this error function can be used as a feedback signal for the observer.

This error function can only be used in the start mode when there is no torque production, and the sensing current is injected into all of the phases to detect the rotor position. When the motor needs to produce torque, no sensing current can be injected for those rotor positions which torque is produced. For these rotor positions the sense signal is one, i.e. sense =1, and g(θ) for that phase is set equal to $g(\hat{\theta})$.

$$g(\theta) = \begin{cases} I_{ave}(\theta) & \text{when sensing current is injected} \\ g(\hat{\theta}) & \text{when torque producing current is in phase} \end{cases} \qquad \text{4-17}$$

With this definition of the error function, the function's value is plotted in Figure

4.6 versus the rotor position when the estimated rotor position $\hat{\theta}$ is 2 mechanical degrees greater than the actual rotor position θ and positive torque is being generated. The error function values using (4-16) and (4-17) versus rotor position when the estimated rotor position error is -5, -4, -3, -2, -1, 1, 2, 3, 4, 5 mechanical degrees are shown in Figure 4.7.

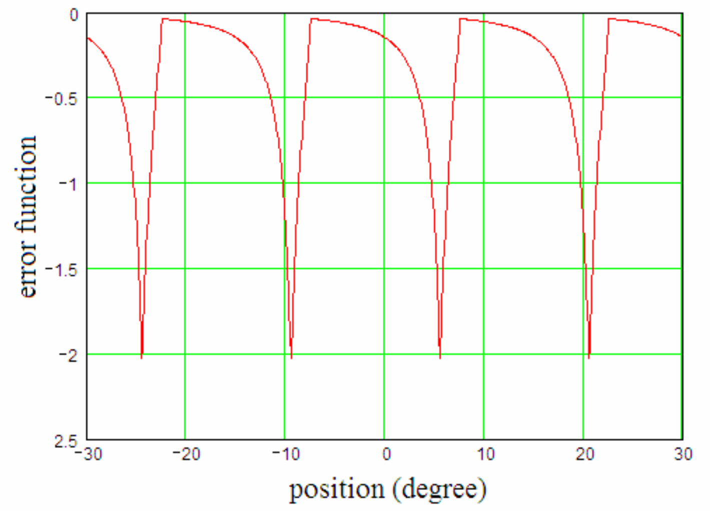


Figure 4.6 the error function value versus the rotor position with consideration of the torque producing current

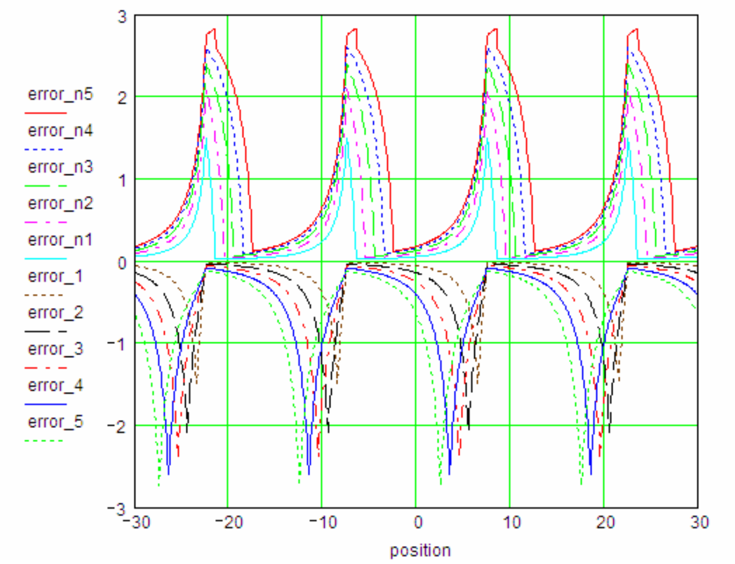


Figure 4.7 the error function value versus the rotor position curves at different rotor position error, -5, -4, -3, -2, -1, 1, 2, 3, 4, and 5 mechanical degrees

The error function defined by (4-17) is still monotonic and negative when the rotor position error is greater than zero. It is monotonic and positive when the rotor position error is less than zero. This assures that the error function with this definition can be used as a feedback signal for the observer.

4.5 System stability and performance of the observer

The error function in (4-16) can be rewritten as

$$f(\theta,\hat{\theta}) = -\, f_1(\hat{\theta},\, \theta_e) \qquad \text{4-18}$$

Applying (4-21) into (4-8), the observer becomes

$$\frac{d}{dt}\begin{bmatrix} e_\theta \\ e_\omega \end{bmatrix} = \begin{bmatrix} e_\omega + H_1 \cdot f_1(\hat{\theta}, e_\theta) \\ H_2 \cdot f_1(\hat{\theta}, e_\theta) \end{bmatrix} \qquad \text{4-19}$$

It can be rearranged as

$$\dot{X} = f_2(X, u) \qquad \text{4-20}$$

where

$$X = \begin{bmatrix} e_\theta \\ e_\omega \end{bmatrix},\ f_2(X,u) = \begin{bmatrix} e_\omega + H_1 \cdot f_1(u, e_\theta) \\ H_2 \cdot f_1(u, e_\theta) \end{bmatrix},\ \ u = \hat{\theta} \qquad \text{4-21}$$

It is obviously a nonlinear system. To analyze its stability, a scalar W is defined as

$$W = X^T Q X \qquad \text{4-22}$$

where Q is a positive definite 2 x 2 matrix. Note that X, W, u are all functions of time. By definition, $W \geq 0$ and W=0 when X=0. The temporal derivative of W is

$$\dot{W} = \dot{X}^T Q X + X^T Q \dot{X} \qquad 4\text{-}23$$

Applying (4-20) into (4-23), it becomes

$$\dot{W} = f_2(X,u)^T Q X + X^T Q f_2(X,u) \qquad 4\text{-}24$$

Since $\dot{W}$ and the two terms on the right ride are all scalars, transposing the first term or the second term gives

$$\dot{W} = f_2(X,u)^T (Q+Q^T) X = X^T (Q+Q^T) f_2(X,u) \qquad 4\text{-}25$$

To assure the system is stable, $\dot{W}$ needs to be negative so that every state will decay to zero. Thus the condition for the system to be stable is

$$f_2(X,u)^T (Q+Q^T) X < 0 \quad \textbf{or} \quad X^T (Q+Q^T) f_2(X,u) < 0 \qquad 4\text{-}26$$

Choosing Q=I, the identity matrix, applying (4-21) into (4-26), the condition becomes

$$e_\omega \cdot e_\theta + H_1 \cdot f_1(\hat{\theta}, e_\theta) \cdot e_\theta + H_2 \cdot f_1(\hat{\theta}, e_\theta) \cdot e_\omega < 0 \qquad 4\text{-}27$$

A sufficient condition for stability is

$$H_1 = -\frac{1}{2 f_1(\hat{\theta}, e_\theta)} (e_\omega + \frac{N}{e_\theta}) \qquad 4\text{-}28$$

$$H_2 = -\frac{1}{2 f_1(\hat{\theta}, e_\theta)}(e_\theta + \frac{N}{e_\omega}) \qquad \text{4-29}$$

where N>0.

Meeting the requirements in (4-28) and (4-29) assure the stability of the nonlinear system given in (4-8). Due to the complexity of $f_1(\hat{\theta}, e_\theta)$, a more practical condition needs to be developed. Assume

$$f(\theta, \hat{\theta}) = -k(\hat{\theta}) \cdot e_\theta \qquad \text{4-30}$$

where $k(\hat{\theta})$ is a nonlinear periodic function of the rotor position that can be estimated from the results in Figure 4.7. Its period is an electrical period divided by the phase number, in this case, 15 degrees. Its boundaries are $k_{f\min}$ and $k_{f\max}$.

$$k_{\mathrm{f}\min} \leq k(\hat{\theta}) \leq k_{f\max} \qquad \text{4-31}$$

Inserting (4-30) into (4-8), gives

$$\begin{bmatrix} \frac{de_\theta}{dt} \\ \frac{de_\omega}{dt} \end{bmatrix} = \begin{bmatrix} -H_1 \cdot k(\hat{\theta}) & 1 \\ -H_2 \cdot k(\hat{\theta}) & 0 \end{bmatrix} \begin{bmatrix} e_\theta \\ e_\omega \end{bmatrix} \qquad \text{4-32}$$

The Eigen values of the characteristic matrix for fixed $\hat{\theta}$ are

$$Eigen_{1,2} = -\frac{1}{2} H_1 \cdot k(\hat{\theta}) \pm \frac{1}{2} \sqrt{H_1{}^2 \cdot k(\hat{\theta})^2 - 4H_2 \cdot k(\hat{\theta})}$$

According to classic control theory, the Eigen values need to be negative real numbers or have negative real parts for the system to be stable.

$real[Eigen_{1,2}]<0$ 4-33

Since $k(\hat{\theta})>0$, to satisfy the stability requirement imposed on H_1 and H_2 is

$H_1>0$, $H_2>0$ 4-34

The settling time for the rotor speed and the rotor position is a function of the rotor position. If $H_1{}^2\cdot k(\hat{\theta})^2<4H_2\cdot k(\hat{\theta})$, the two Eigen values are conjugate complex numbers with a common real part of $H_1\cdot k(\hat{\theta})$. If $H_1{}^2\cdot k(\hat{\theta})^2=4H_2\cdot k(\hat{\theta})$, the two Eigen values are identical and they are $H_1\cdot k(\hat{\theta})$. If $H_1{}^2\cdot k(\hat{\theta})^2>4H_2\cdot k(\hat{\theta})$, the two Eigen values are unequal real numbers. In this case, the settling time will be determined by the Eigen value that has smaller absolute real part, which is $-H_1\cdot k(\hat{\theta})+\sqrt{H_1{}^2\cdot k(\hat{\theta})^2-4H_2\cdot k(\hat{\theta})}$. The settling time is approximately 5 time constants or 5 divided by the real part of the Eigen value that has a smaller real part.

$$T_{sattle}=\begin{cases}\dfrac{2.5}{H_1\cdot k(\hat{\theta})} & if \quad H_1{}^2\cdot k(\hat{\theta})^2\le 4H_2\cdot k(\hat{\theta})\\ \dfrac{2.5}{H_1\cdot k(\hat{\theta})-\sqrt{H_1{}^2\cdot k(\hat{\theta})^2-4H_2\cdot k(\hat{\theta})}} & if \quad H_1{}^2\cdot k(\hat{\theta})^2>4H_2\cdot k(\hat{\theta})\end{cases}$$

4-35

When the settling time is greater than the time for one electrical period of the rotor position, the average value of the error function over an electrical period, called the error average function $fave(\theta_e)$, can be used to determine the system performance. If the SRM is not rotating this approximation can not be used and (4-35) must be used. The error average function $fave(\theta_e)$ is defined as

$$fave(\theta_e)=\frac{1}{2\theta_u}\cdot\sum_{\theta=-\theta_u}^{\theta_u} f(\theta,\theta+\theta_e)$$

4-36

The error average function value versus the rotor position error is plotted in

Figure 4.8. In the plot, the horizontal axis is the rotor position error and the vertical axis is the error average function value.

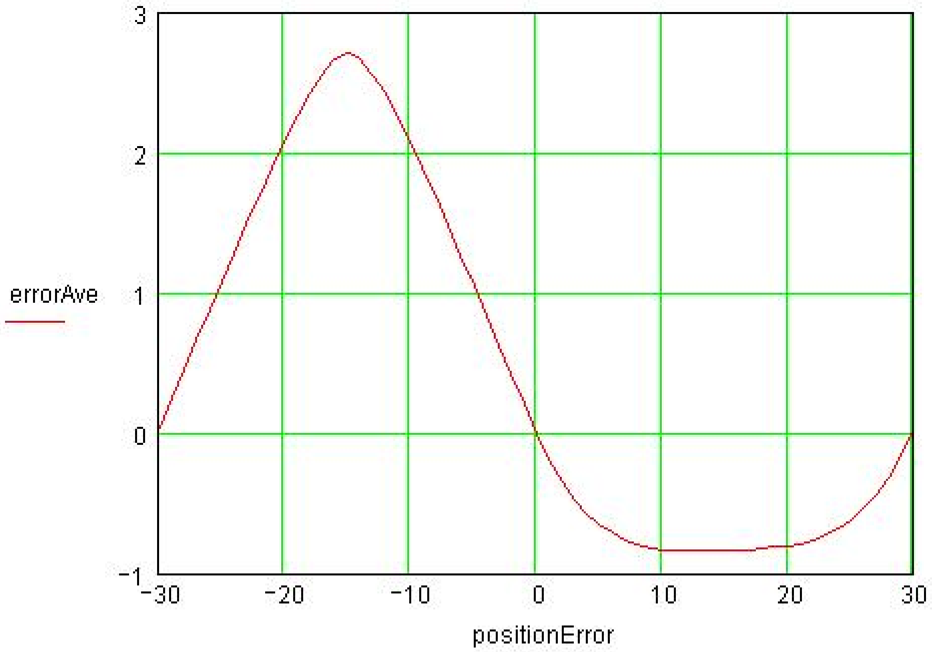


Figure 4.8 the error function value average versus the rotor position error

When the rotor position error is positive and small, the error average function value is monotonic and negative and when the rotor position error is negative and small, the error average function value is monotonic and positive. To linearize the error average function, it is assumed that the operating point is at zero position error, i.e. $\theta_e=0$.

$$fave(\theta_e) = -k_{ave}\,\theta_e \qquad \text{4-37}$$

where k_{ave} is the absolute value of the slope of the curve in Figure 4.8 at the origin.

Substituting $f(\theta,\hat{\theta})$ with $fave(\theta_e)$ in (4-8) and applying (4-37) into it gives

$$\begin{bmatrix} \frac{de_\theta}{dt} \\ \frac{de_\omega}{dt} \end{bmatrix} = \begin{bmatrix} -H_1\cdot k_{ave} & 1 \\ -H_2\cdot k_{ave} & 0 \end{bmatrix} \begin{bmatrix} e_\theta \\ e_\omega \end{bmatrix} \qquad \text{4-38}$$

Since k_{ave} is a positive number, the system is still stable if (4-33) is satisfied. The settling time of the control system has the same format of (4-35) except that k(θ) is substituted by k_{ave}.

$$T_{sattle} = \begin{cases} \dfrac{2.5}{H_1 \cdot k_{ave}} & if \quad H_1{}^2 \cdot k_{ave}{}^2 < 4H_2 \cdot k_{ave} \\ \dfrac{2.5}{H_1 \cdot k_{ave} - \sqrt{H_1{}^2 \cdot k_{ave}{}^2 - 4H_2 \cdot k_{ave}}} & if \quad H_1{}^2 \cdot k_{ave}{}^2 < 4H_2 \cdot k_{ave} \end{cases} \qquad \text{4-39}$$

4.6 The speed limitation of the inductance profile demodulator based observer

The inductance profile demodulator based observer has two inherent speed limits. One of them is caused by the frequency of the modulating signal and the resulting time delay of the demodulator, which is a low pass filter, the other one is caused by the sampling frequency of the microprocessor implementation of the observer and error function.

4.6.1 The time delay of the demodulator

The demodulator used is the low pass filter circuit shown in Figure 3.12. Its transfer function is given by (3-11). The Bode plot of the transfer function is shown in Figure 4.9.

The transfer function in polar coordinates is

$$f(\omega) = \frac{R_2 / R_1}{\sqrt{1 + (R_2 \cdot C_1 \cdot \omega)^2}} e^{j \tan^{-1}(-R_2 \cdot C_1 \cdot \omega)} \qquad \text{4-40}$$

Assume the input current to the demodulator is sinusoidal (the first harmonic of the current)

$$V_{in}(t) = V_{in} \cdot \cos[\omega(t + t_0)] \qquad \text{4-41}$$

It can be transformed into an Euler format

$$V_{in}(t) = V_{in} \cdot \mathrm{Re}\{e^{j\omega(t+t_0)}\} \qquad 4\text{-}42$$

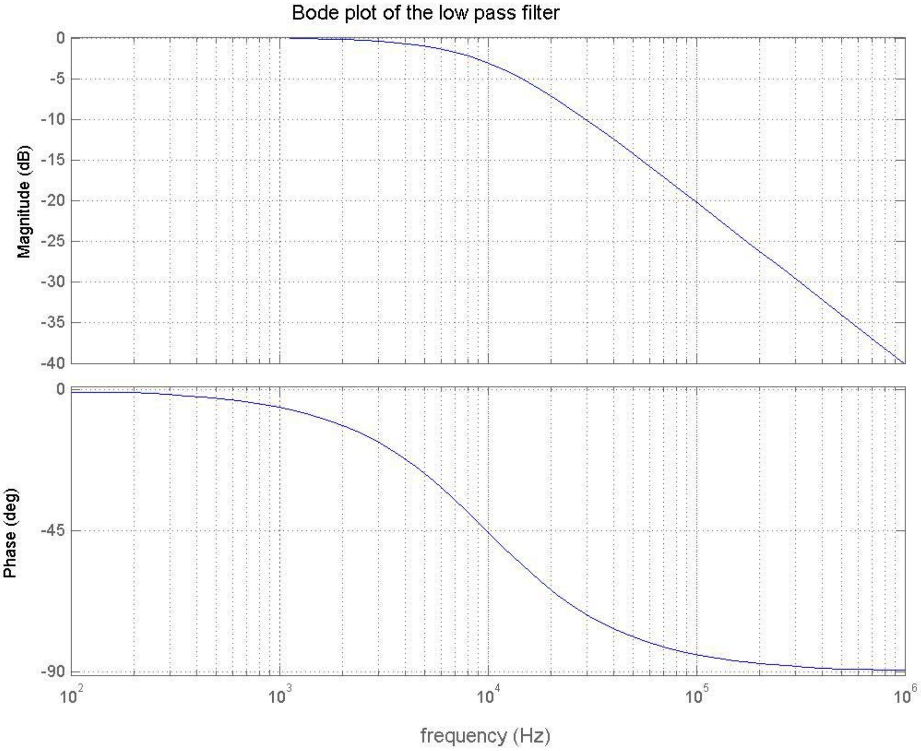


Figure 4.9 the Bode plot of the low pass filter function

where V_{in} is the amplitude, ω is the frequency, ωt_0 is the phase. The output of the demodulator is just the transfer function times the input voltage

$$V_{out}(t) = V_{in} \cdot \frac{R_2 / R_1}{\sqrt{1 + (R_2 \cdot C_1 \cdot \omega)^2}} \cdot \mathrm{Re}\left\{ e^{j\omega\left[t + t_0 - \frac{\tan^{-1}(R_2 \cdot C_1 \cdot \omega)}{\omega}\right]} \right\} \qquad 4\text{-}43$$

Taking the real part the output voltage is

$$V_{out}(t) = V_{in} \cdot \frac{R_2 / R_1}{\sqrt{1 + (R_2 \cdot C_1 \cdot \omega)^2}} \cdot \cos\left\{ \omega \left[t + t_0 - \frac{\tan^{-1}(R_2 \cdot C_1 \cdot \omega)}{\omega} \right] \right\} \qquad 4\text{-}44$$

The time delay t_d of the transfer function is the time difference between the input voltage and the output voltage.

$$t_d = \frac{\tan^{-1}(R_2 \cdot C_1 \cdot \omega)}{\omega} \qquad 4\text{-}45$$

For the demodulator to work correctly $R_2 \cdot C_1 \cdot \omega$ must be small,

$$\tan^{-1}(R_2 \cdot C_1 \cdot \omega) \approx R_2 \cdot C_1 \cdot \omega \qquad 4\text{-}46$$

Thus the time delay through the demodulator is simply a constant.

$$t_d = R_2 \cdot C_1 \qquad 4\text{-}47$$

The time delay corresponds to an error in the estimated rotor position which depends on speed.

$$\theta_{error} = t_d \cdot \omega_m = R_2 \cdot C_1 \cdot \omega_m \qquad 4\text{-}48$$

If the maximum allowed rotor position error is $\theta_{errorMax}$, then the maximum rotor speed that this observer can work at is

$$\omega_{m\,Max1} = \frac{\theta_{errorMax}}{R_2 \cdot C_1} \qquad 4\text{-}49$$

4.6.2 Sampling frequency limitation

According to the Nyquist sampling theorem, the sampling frequency needs to be at least twice that of the maximum frequency of the original signal so that the full information will be preserved.

The inductance profile occurs 6 times in one rotor revolution, and it is symmetrical on the aligned rotor position axis. To preserve the nth harmonic, assuming the first harmonic is the inductance profile waveform itself, the sample frequency needs to be at least 24n times of the rotor speed.

$$F_{sample} \geq 24n \frac{\omega_m}{2\pi} \tag{4-50}$$

where F_{sample} is the sampling frequency, as shown below

$$F_{sample} = \frac{1}{T_{sample}} \tag{4-51}$$

where T_{sample} is the sampling time. Thus the sample time limited maximum speed the position esimator can work at is

$$\omega_{mMax2} = \frac{0.2617}{n \cdot T_{sample}} \tag{4-52}$$

The actual speed limit is the minimum of the above two limitations, ω_{mMax1} and ω_{mMax2} .

$$\omega_{mMax} = \min\left\{\frac{\theta_{errorMax}}{R_2 \cdot C_1}, \frac{0.2617}{n \cdot T_{sample}}\right\} \tag{4-53}$$

The speeds mentioned above in the dissertation are specificed in section 4.9.

4.7 Simulation results

The sensorless control system is simulated using the Matlab/Simulink model.

The observer gains are chosen as H_1=200, and H_2=10000. The inductance profile demodulator based observer is simulated and shown to work from zero speed to medium speeds (5,000 rpm).

4.7.1 Zero speed simulation

At zero speed, the rotor is locked at a certain position and one or two corresponding phases produce torque. The simulation results are shown in Figure 4.10 and 4.11below.

The estimated position reaches steady state at 0.006s. The steady state error is 1.4 mechanical degrees.

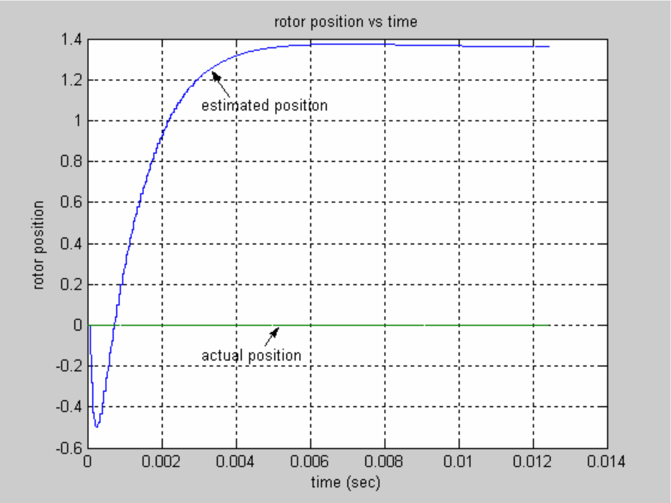


Figure 4.10 the estimated and actual rotor position in degree of the zero speed simulation

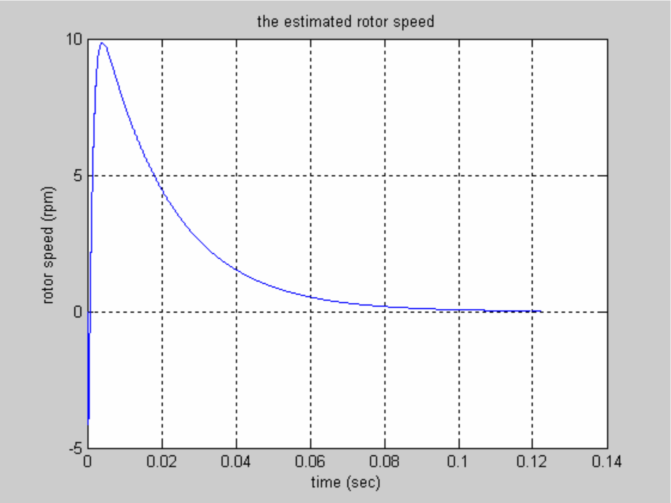


Figure 4.11 the estimated rotor speed of the zero speed simulation

The phase currents are shown in Figure 4.12. During the transient time, phase A was energized with torque producing current for a moment, then as the observer figured out the rotor position the correct Phase B was instead energized. The sensing current was injected into the idle phases

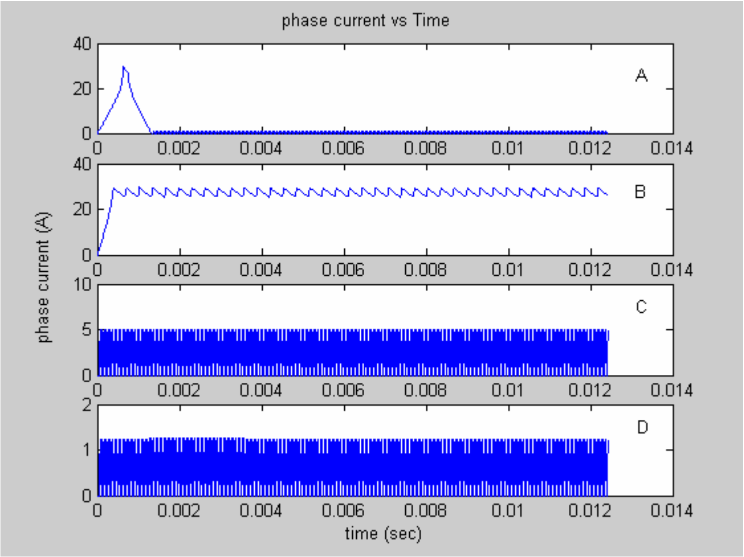


Figure 4.12 the phase currents of all 4 phases of the zero speed simulation

The error function signal $f(\theta,\hat{\theta})$ generated for the zero speed simulation is shown in Figure 4.13. At steady state, the error function output becomes close zero.

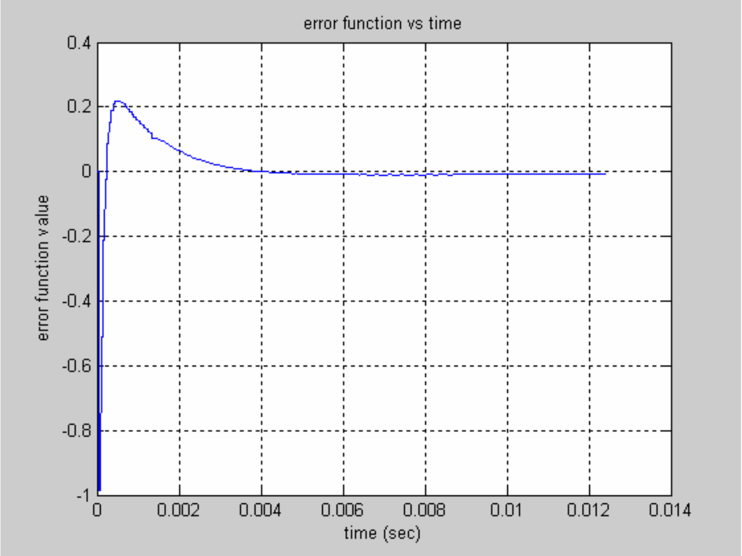


Figure 4.13 the error function generated signal of the zero speed simulation

4.7.2 Medium speed operation

The inductance profile demodulator based observer is simulated with the motor running at 2000 rpm. The estimated rotor position and the actual rotor position are shown in Figure 4.14. The estimated rotor position follows the actual rotor position very well. The estimated rotor speed is shown in Figure 4.15. It oscillates around the correct value of 2000 rpm because the observer system is nonlinear and differences between the measured g(θ) and the calculated g(θ). The estimated speed transient is over in about 10ms. The current in each of the 4 phases are shown in Figure 4.16. The error function value is shown in Figure 4.17, and the electrical torque output of the motor is shown in Figure 4.18.

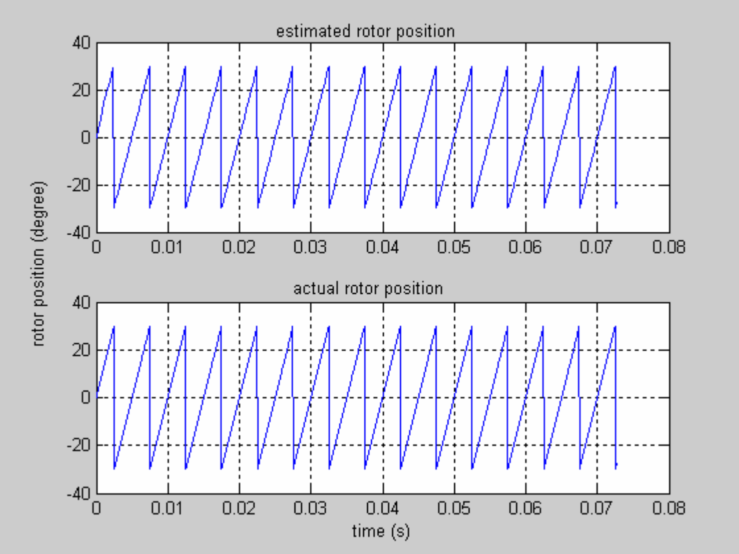


Figure 4.14 the estimated and actual rotor positions when the motor is running at 2000 rpm

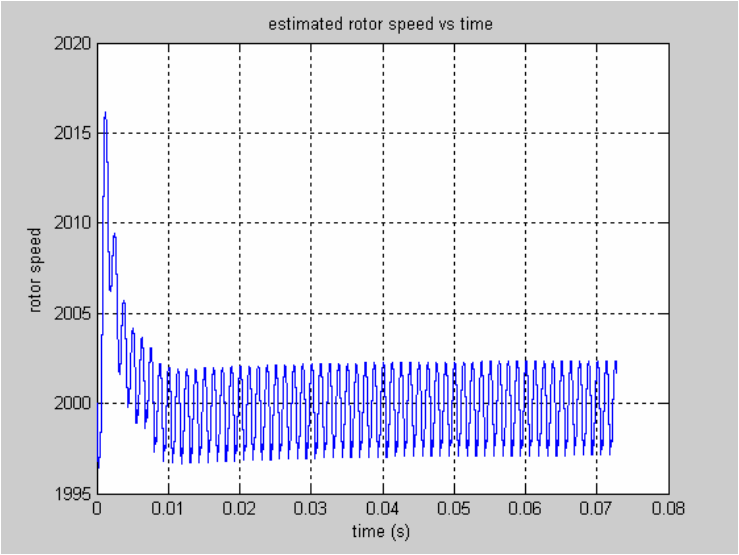


Figure 4.15 the estimated rotor speed when the motor is turning at 2000 rpm

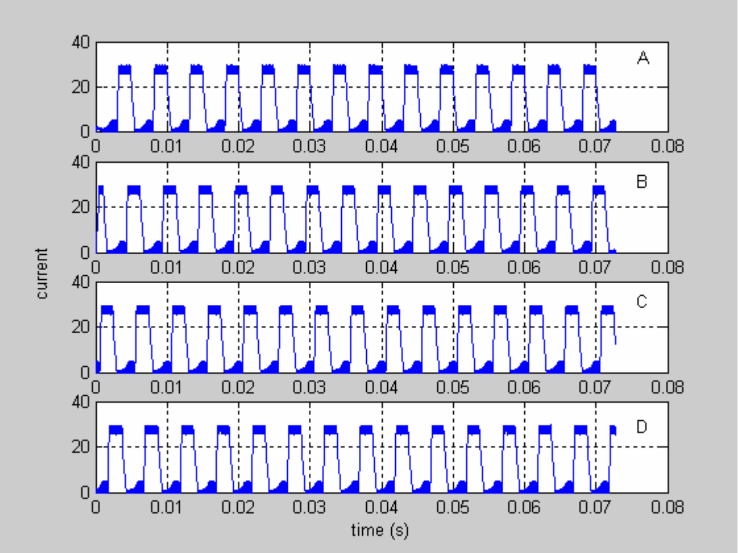


Figure 4.16 the current of the 4 phases when the motor is turning at 2000 rpm

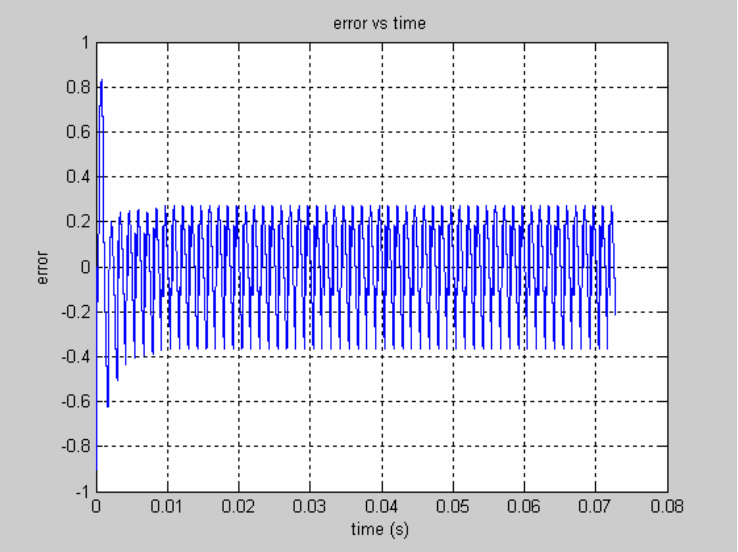


Figure 4.17 the error function value versus time when the motor is running at 2000 rpm

4.8 Experiment results

4.8.1 Inductance asymmetry of the motor

Due to manufacturing tolerances, the inductance profiles among the 4 phases of the experimental SRM are not identical. This is caused in part by the different length of the stator poles and the rotor poles. Since the inductance profiles are not identical, the g(θ) profiles are not identical either, as shown in Figure 4.19. The ripples on the profiles are due to the high frequency modulation sensing current.

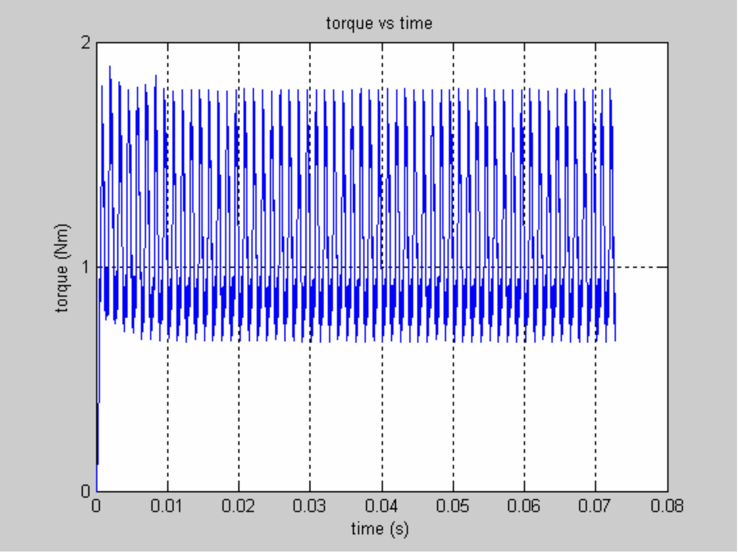


Figure 4.18 the electrical torque of the SRM when the motor is running at 2000 rpm

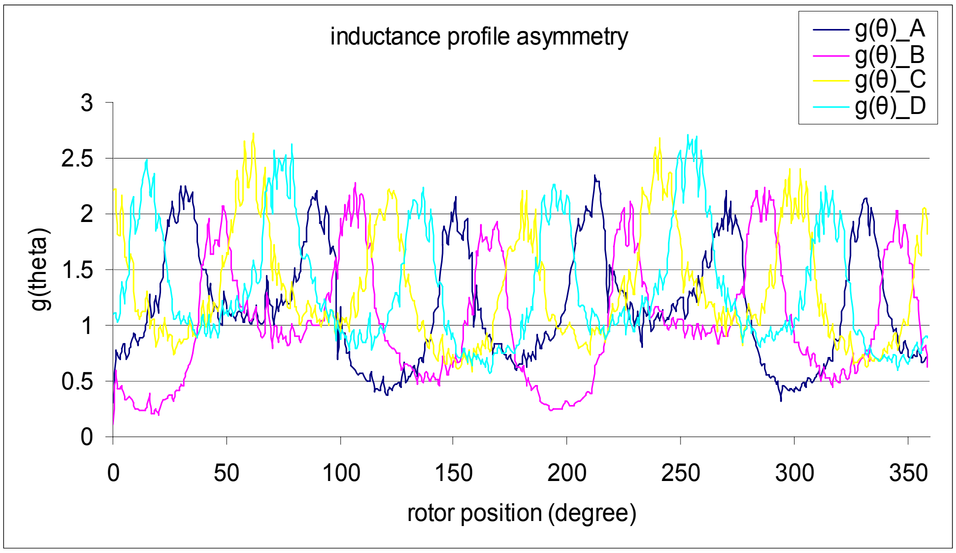


Figure 4.19 the g(θ) asymmetry of the motor

The aligned inductance and the unaligned inductance among the 4 phases are different. In each phase, the aligned inductance and the unaligned inductance with different pairs of the rotor poles are different. Since the g(θ) profiles are repetitive with a period equal to 180 degrees, the rotor position period is changed from 60 degrees to 180 degrees. Based on the measured g(θ) profiles, the inductance profiles for the experimental SRM were computed and are shown in Figure 4.20.

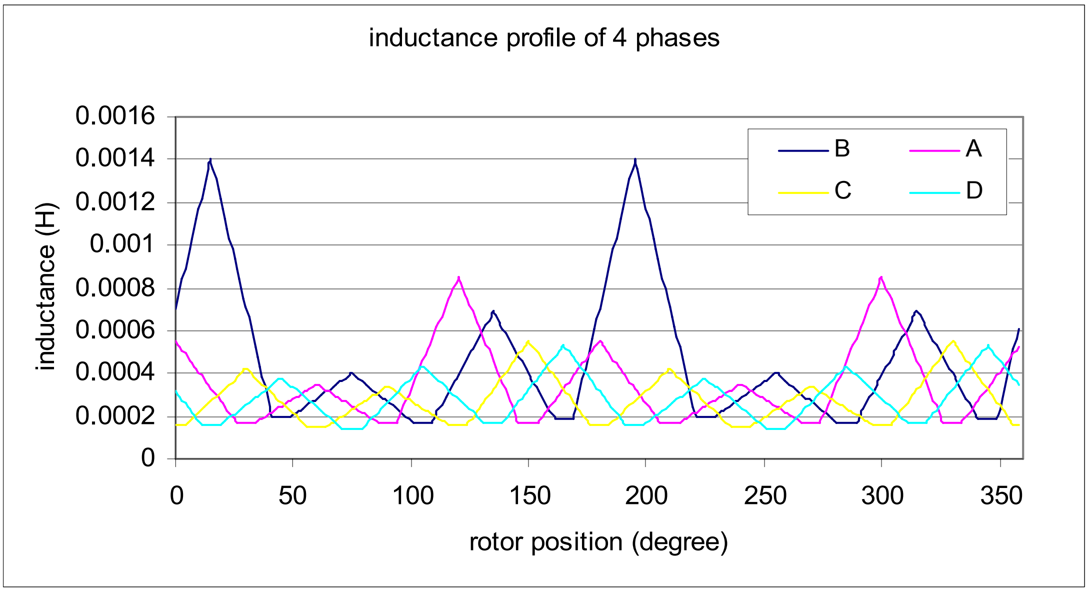


Figure 4.20 the inductance profiles of the 4 phases based on the measured g(θ) profiles

To test if the error function in (4-16) and (4-17) still work with the inductance asymmetry, the error function value versus the rotor position curves with a rotor position error of 2 degrees and -2 degrees are plotted in Figure 4.21 for the experimental SRM.

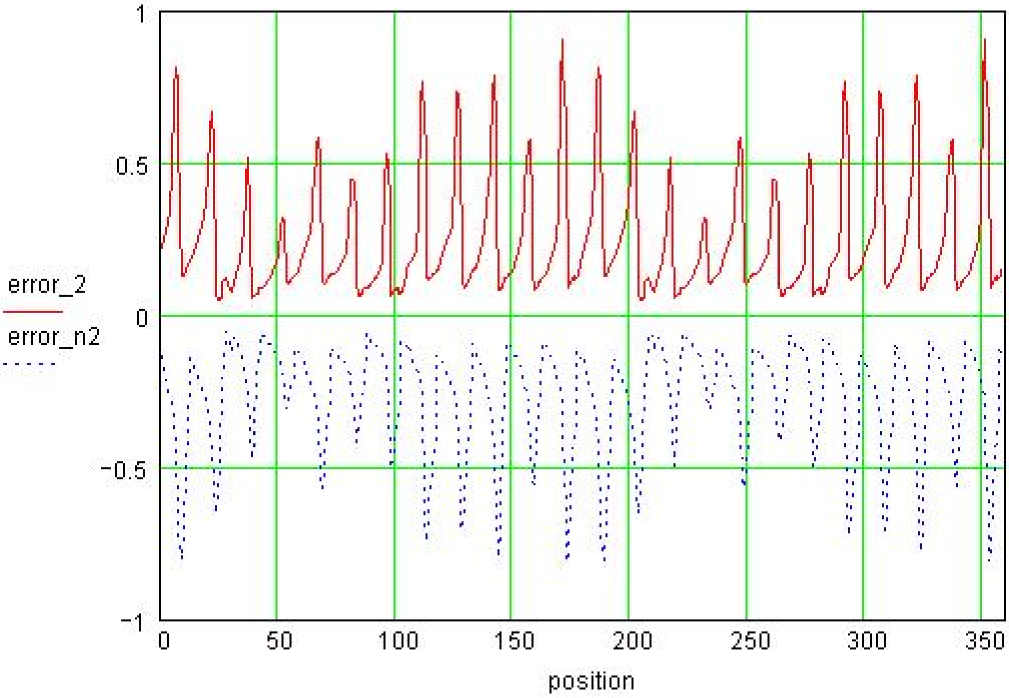


Figure 4.21 the error function value versus the rotor position at the rotor position error, 2 degrees (error_2) and -2 degrees (error_n2) with the inductance asymmetry

The error functions in the plots have different amplitudes because of the inductance asymmetry. The error function value is still monotonic and positive when the rotor position error is positive over a whole electrical period. It is monotonic and negative when the rotor position error is negative over a whole electrical period. This insures that the error function will still work with the inductance asymmetry.

4.8.2 Starting process

The inductance profile demodulator based observer is implemented in the experimental system. A DC motor is used to load the SRM. For the initial data the torque command and resulting current command are set low so that the shaft friction of the DC motor which is the only SRM load is adequate to prevent the SRM from accelerating too fast to record the data. A start process is recorded in the figures below. The estimated and actual rotor positions are shown in Figure 4.22. Note that the estimated rotor position is shifted up 180 degrees for easier viewing. The estimated rotor position error is shown in Figure 4.23. It is within ±5 degrees worst case including the noise with an rms value equal to less than 2 degrees. The noise is due to the high frequency modulation current. The large spikes in the error that go above 20 degrees on the curve are due to the fact that the rotor positions are wrapped into an electrical period. When one of the rotor positions is wrapped from 180 to 0 degree, the difference between these two rotor positions is momentarily close to 180 or -180 degrees and results in the spikes. The estimated rotor speed is shown in Figure 4.24. The error function value is shown in Figure 4.25.

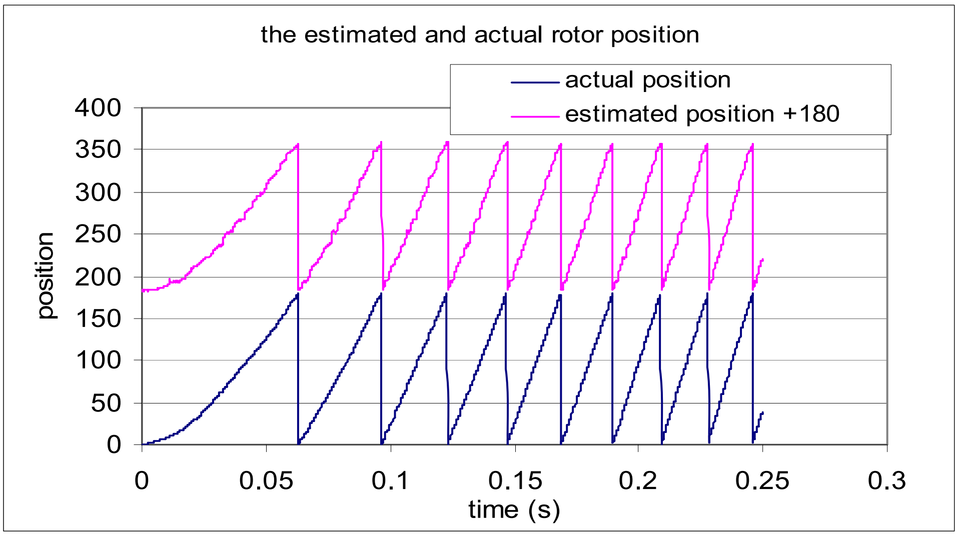


Figure 4.22 the estimated and actual rotor positions of the starting process

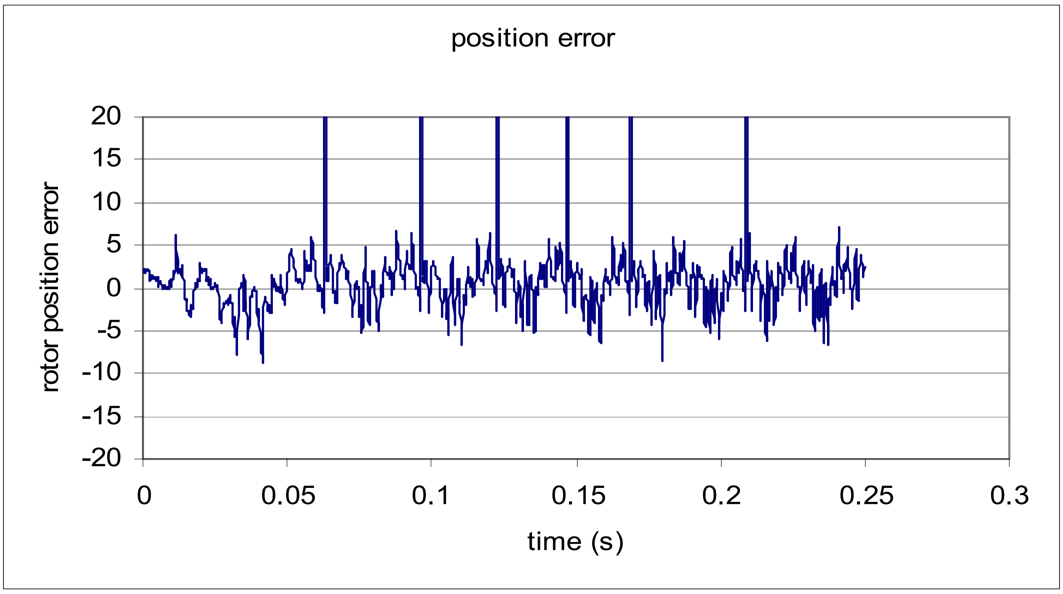


Figure 4.23 the estimated rotor position error during the starting process

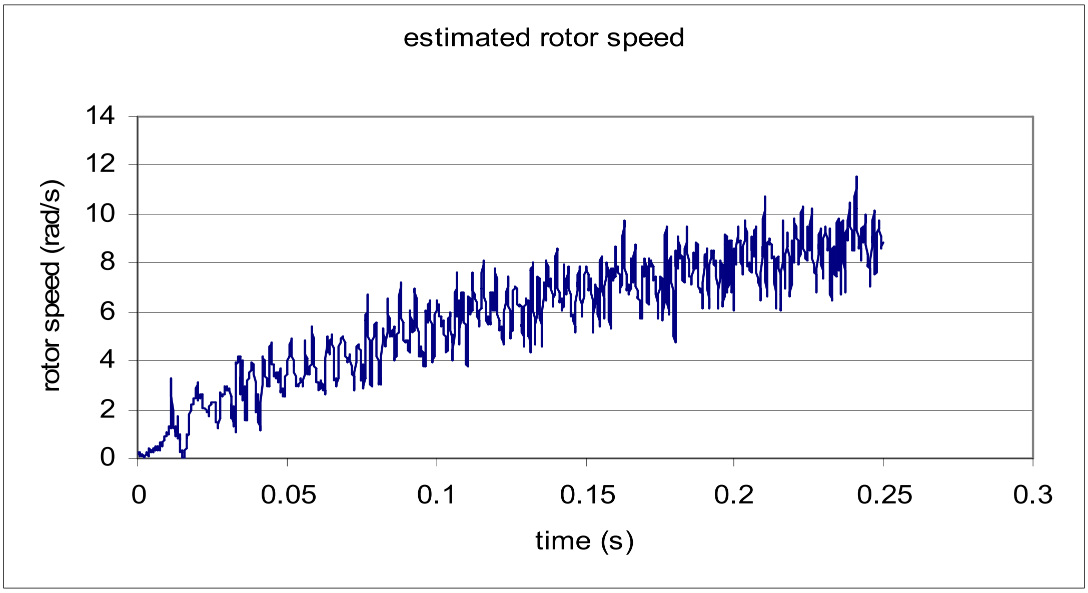


Figure 4.24 the estimated rotor speed during the starting process

4.8.3 Constant speed operation

Data has also been taken when the SRM is turning at a constant speed of 15.0 rad/s. The estimated and actual rotor positions are shown in Figure 4.26. The estimated rotor position error is shown in Figure 4.27. The estimated rotor speed is shown in Figure 4.28. The error function value is shown in Figure 4.29. The noise on these curves is also due to the modulation current. The SRM phase current is shown in Figure 4.30. The waveform shows the alternating low amplitude modulation current and the high amplitude torque producing current.

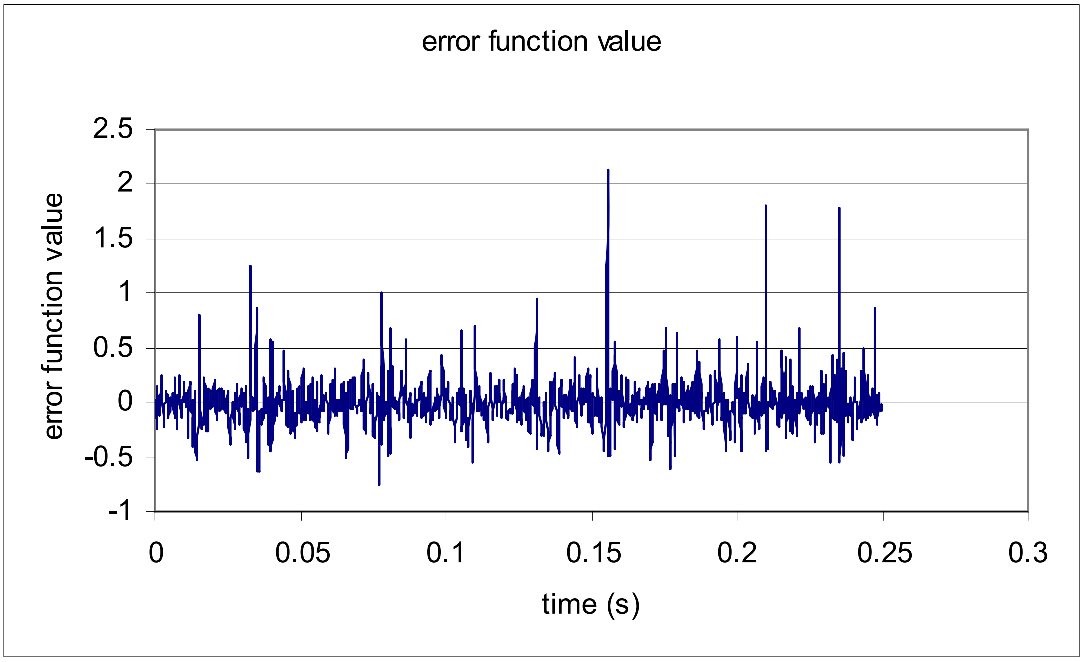


Figure 4.25 the error function value during the starting process

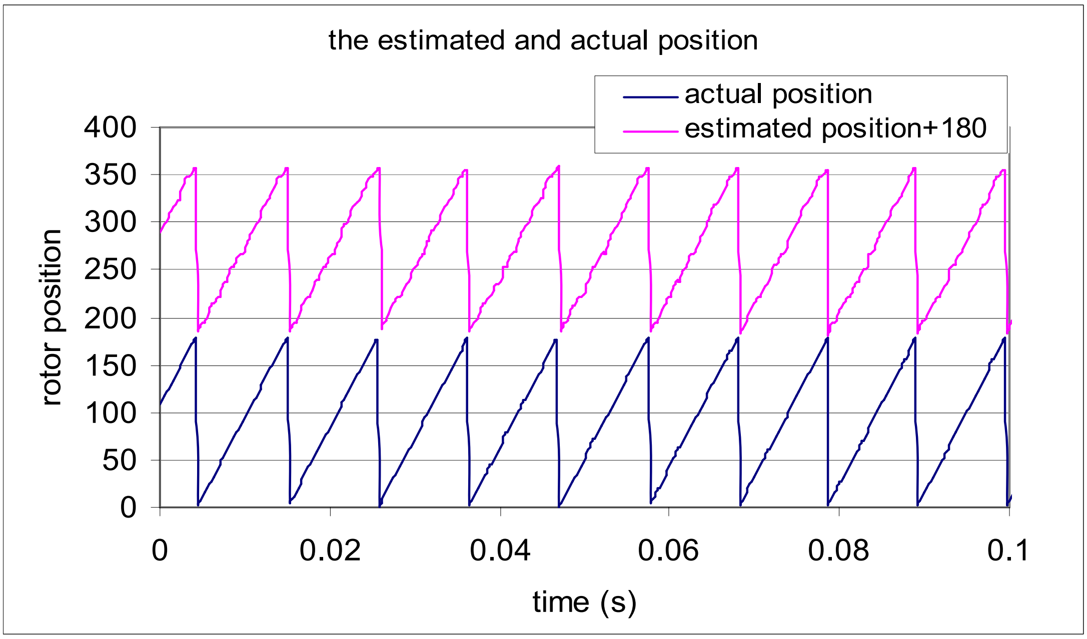


Figure 4.26 the estimated and actual rotor positions at steady state

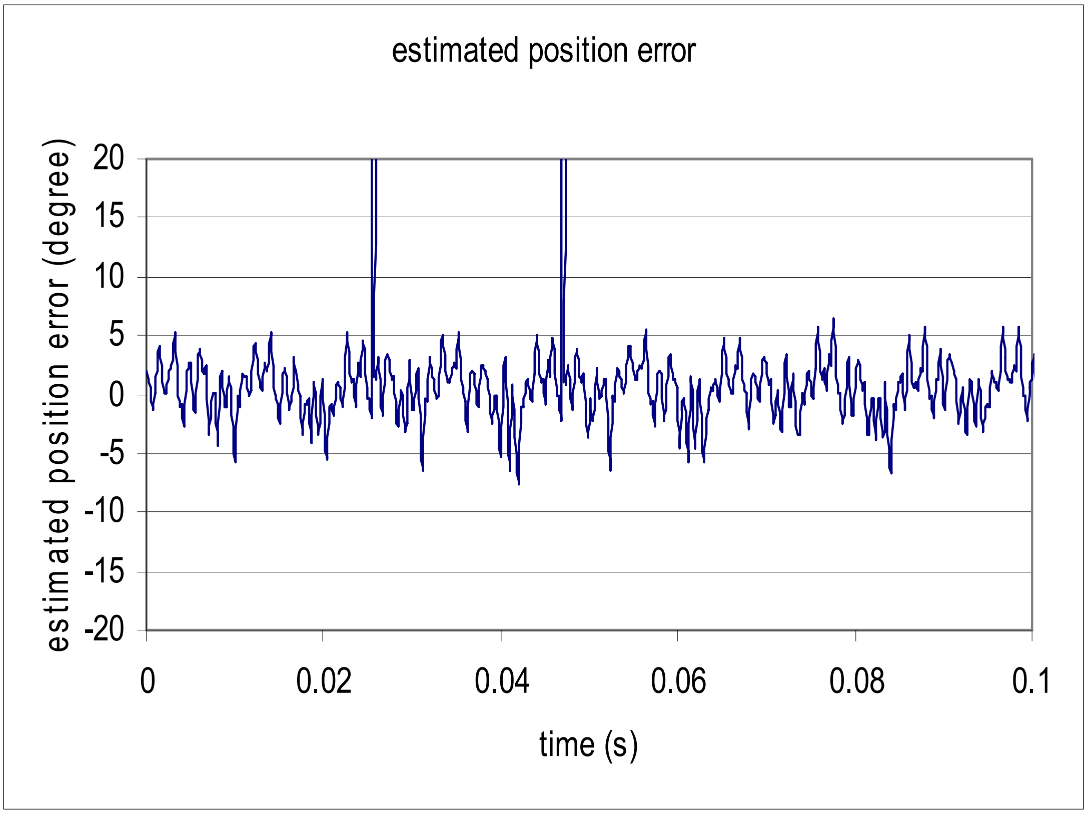


Figure 4.27 the estimated position error at steady state

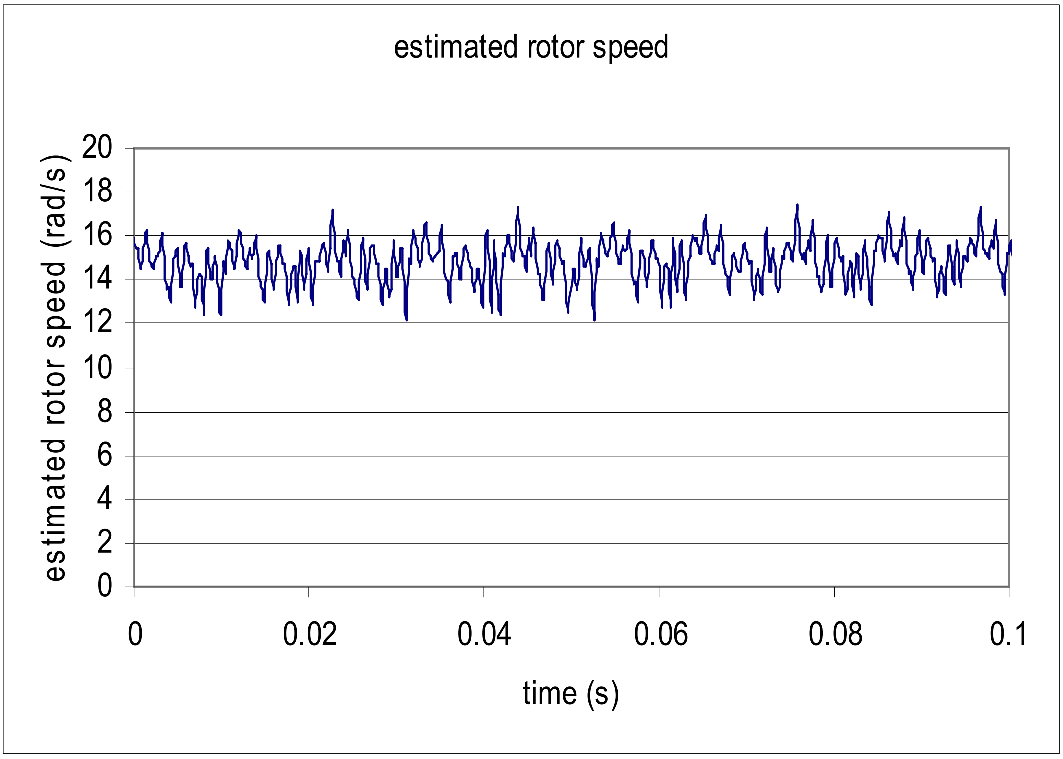


Figure 4.28 the estimated rotor speed at steady state

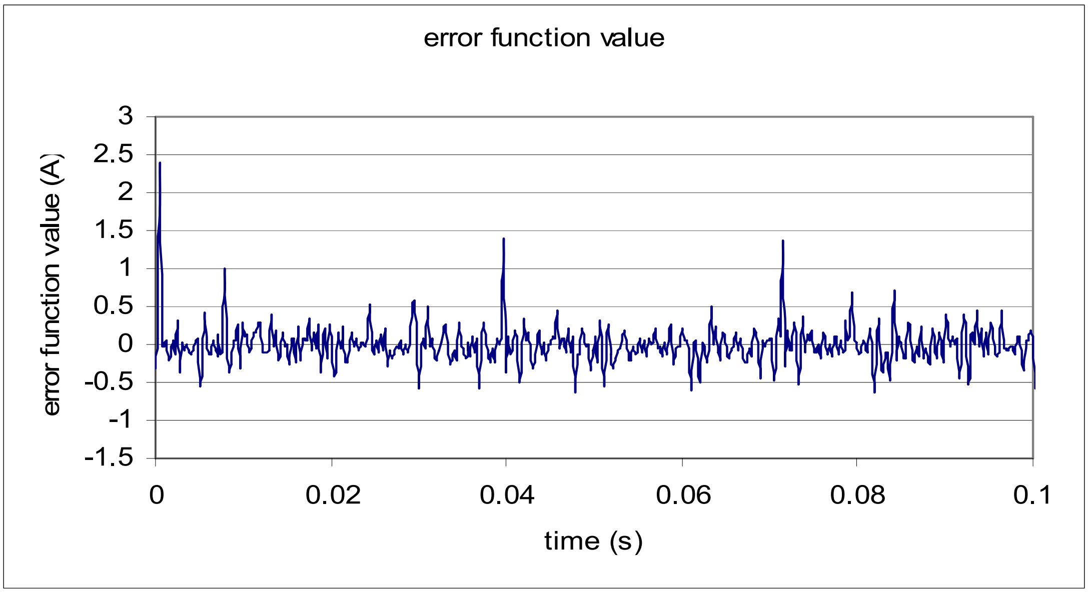


Figure 4.29 the error function value

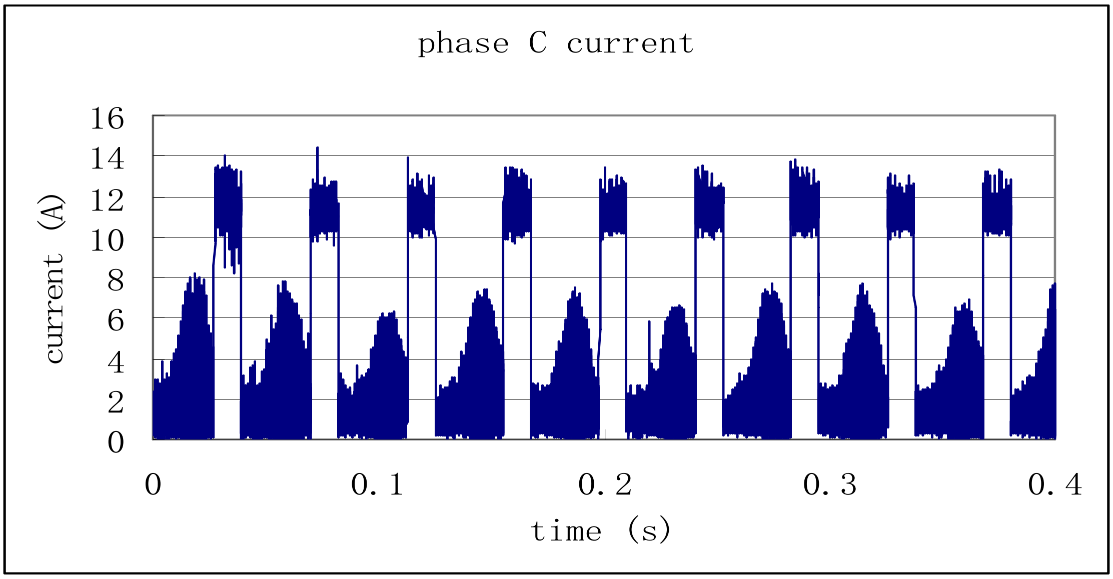


Figure 4.30 the current of a phase

4.9 Speed limitation

In the hardware implementation of the inductance profile demodulator based observer, the components chosen for the low pass filter demodulator are R_2=18.2 KΩ, C_1=560 pF. The predicted maximum operating speed limited by the demodulator is ω_{mMax1} =860 rad/s using (4-40). When the sampling time is T_{sample}=600μs, the sampling time speed limitation is $\omega_{mMax2} = 218$ rad/s according to (4-52) if n is chosen to be 2. The sampling time includes the analog to digital conversion time, the computation time of the control algorithm, and the data communication time. A higher speed DSP, a DSP with general purpose input/output (GPIO), a more efficient program or a higher speed ADC board can help reduce the sampling time. Furthermore, if a portion or the whole part of the program can be implemented into the FPGA chip as a special purpose microprocessor, it can run much faster. The actual observed speed limitation of the inductance profile demodulator based observer is 218 rad/s. When the SRM accelerates and reaches the speed limitation, the estimated angles will be incorrect causing the torque producing currents to be produced at the wrong rotor positions and the electrical torque decreases. Figure 4.31 shows that when the speed reaches 180 rad/s, the inductance profile demodulator based observer starts to fail. Using this measured maximum speed in (4-43), n is computed to be 2.4. This means that for the sensorless control to work properly, the first, the second, and a part of the third harmonic of the inductance profile need to be preserved in the sampling.

In another experiment, the sampling time is set to T_{sample}=60μs using a more efficient program. Now $\omega_{mMax2} = 1131$ rad/s according to (4-52) if n is chosen to be 4. The speed limitation of the observer should now be determined by the demodulator's time delay and be equal to ω_{mMax1} , which is 860 rad/s. The estimated rotor speed for this experiment is shown in Figure 4.32. It is seen that the maximum experimental speed limit is 500 rad/s, lower than the theoretical speed limit. This is most likely due to the asymmetry of the experimental SRM causes larger rotor position error, as shown in Figure 4.33. The average position error is 10 mechanical degrees, which is 1/3 of the torque producing region. This position error causes a decrease in torque and hence a decrease in the speed.

4.10 The torque drop

The torque drops with a position error. The torque output is shown in Table 4.1 with different rotor position error. The rotor position error is the estimated position subtracted by the actual position in degree. In this case, the torque producing region is from -25 degree to 0 degree. It is found that the torque is close to zero when the rotor position error is -10 degrees. This is the reason why the rotor speed can not increase any longer when it reaches its maximum in Figure 4.33.

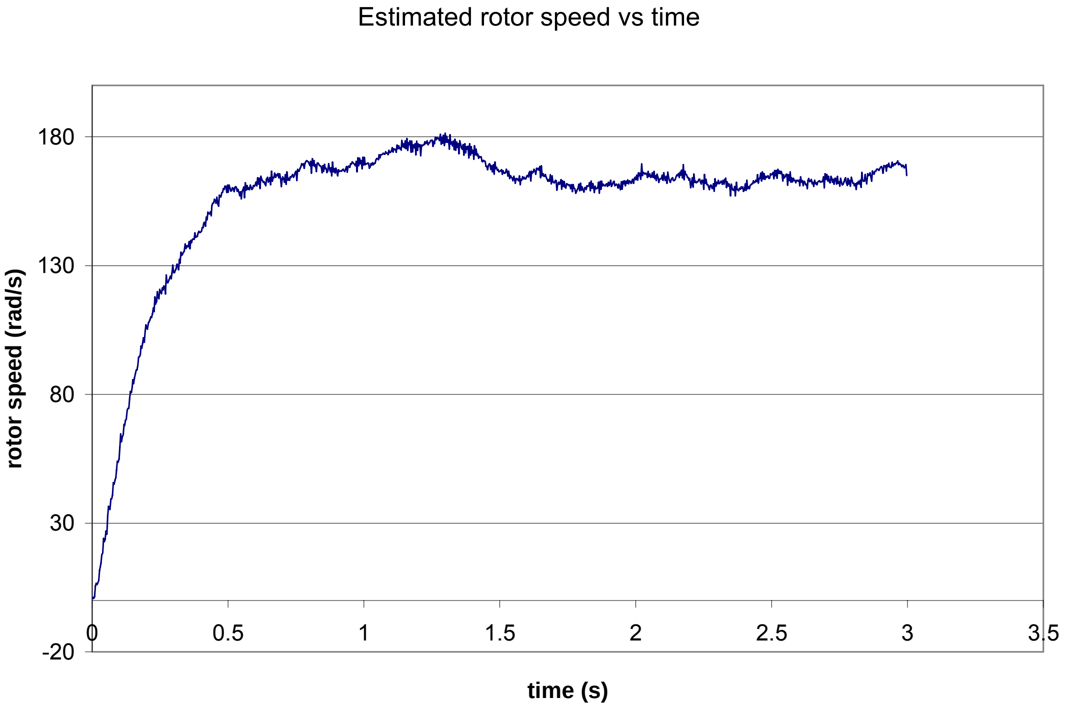


Figure 4.31 the speed limitation of the observer at t_{sample}=600μs

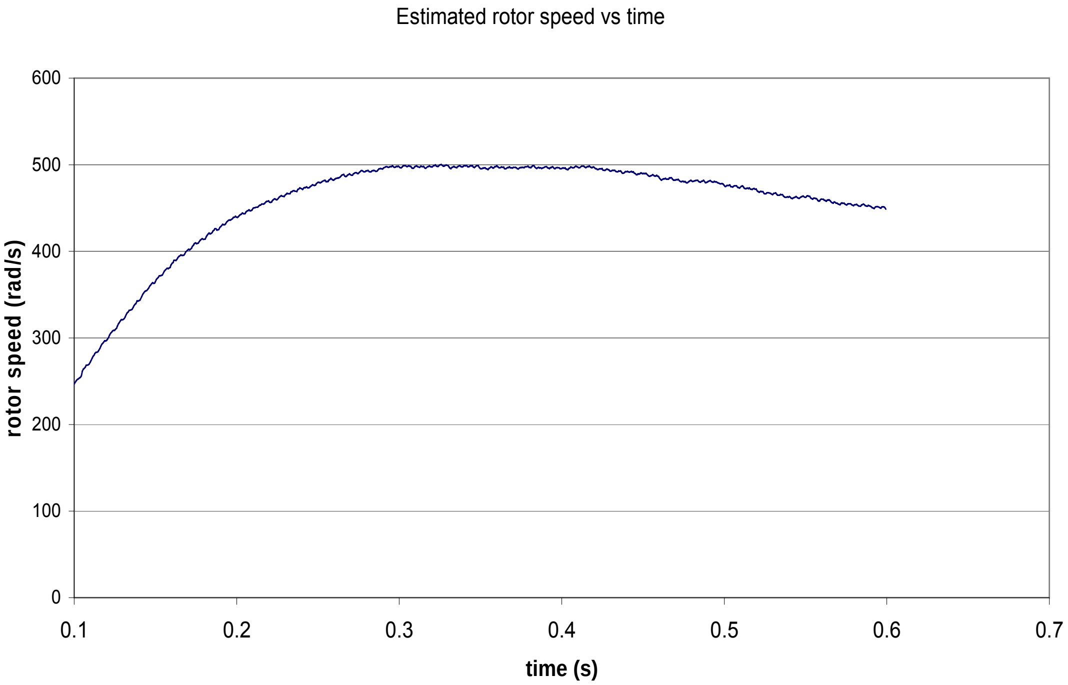


Figure 4.32 the speed limitation of the observer at t_{sample}=60μs

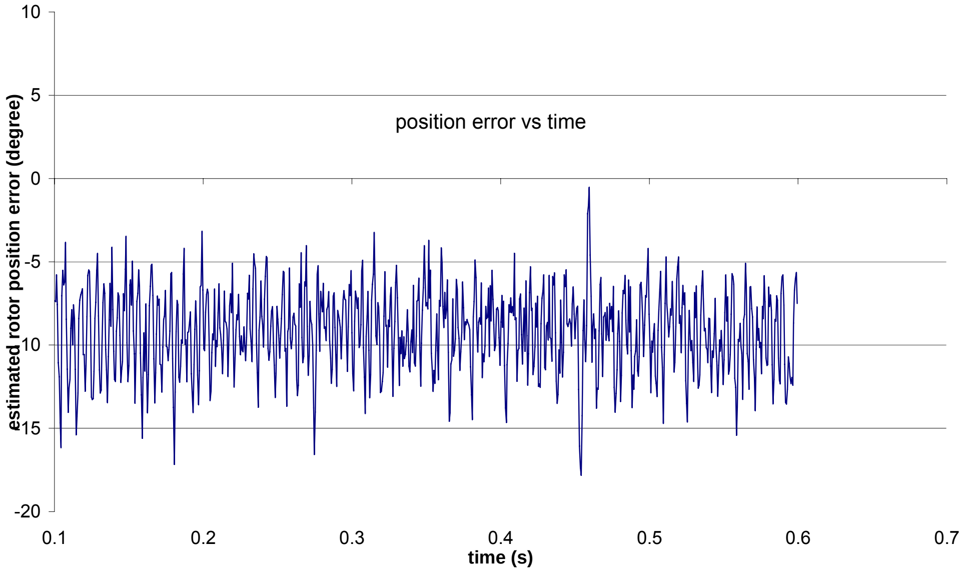


Figure 4.33 the estimated rotor position error at t_{sample}=60μs in the speed limitation experiment

Table 4.1 The torque output with different rotor position errors

Position error (degree)	Torque (Nm)	Percent of the max torque		Position error (degree)	Torque (Nm)	Percent of the max torque
0	2.2824	100.00%				
1	2.2644	99.21%		-1	2.2568	98.88%
2	2.2192	97.23%		-2	2.2076	96.72%
3	2.1512	94.25%		-3	2.1212	92.94%
4	2.0676	90.59%		-4	2.0084	88.00%
5	1.9436	85.16%		-5	1.8812	82.42%
6	1.8504	81.07%		-6	1.7336	75.96%
7	1.7552	76.90%		-7	1.5528	68.03%
8	1.6592	72.70%		-8	1.3528	59.27%
9	1.562	68.44%		-9	1.1484	50.32%
10	1.4632	64.11%		-10	0.9424	41.29%
11	1.364	59.76%		-11	0.7368	32.28%
12	1.2636	55.36%		-12	0.532	23.31%
13	1.1624	50.93%		-13	0.3272	14.34%
14	1.0608	46.48%		-14	0.1236	5.42%
15	0.9596	42.04%		-15	-0.0792	-3.47%

4.11 The rotor position resolution

The estimated rotor position resolution is basically the rotor speed times the sampling time. The actual rotor position is obtained by an optical encoder, which produces 360 pulses every mechanical cycle. The resolution of the estimated and actual rotor position is listed in Table 4.2 at different rotor speeds and sampling time.

Table 4.2 the resolution of the estimated and actual rotor positions

Rotor speed (rpm)	Estimated position resolution when $t_{sampling}$=600us (degree)	Estimated position resolution when $t_{sampling}$=60us (degree)	Actual position resolution (degree)
0	0	0	1
500	1.8	0.18	1
1000	3.6	0.36	1
5000	18	1.8	1
10000	36	3.6	1
15000	54	5.4	1

4.12 Transient response

The theoretical settling time is determined using (4-30) is 0.024s with k_{ave}=11.46, H_1=200, and H_2=10000. To obtain the settling time experimentally, two experiments have been done. In the first experiment, the rotor shaft is grabbed suddenly when it is turning at a constant speed. The rotor speed changes from the constant speed to zero immediately, as shown in Figure 4.34. It is seen that the estimated rotor speed responses goes to zero 0.03s later than the actual rotor speed does. In the second experiment, the SRM is controlled with the optical encoder position sensor. With the rotor turning at a constant speed, a sudden change is made to the estimated rotor position. The transient response of the estimated rotor position and the rotor speed are shown in Figure 4.35 and Figure 4.36. The experimental settling time for the rotor position and the rotor speed are 0.02s and 0.03s respectively.

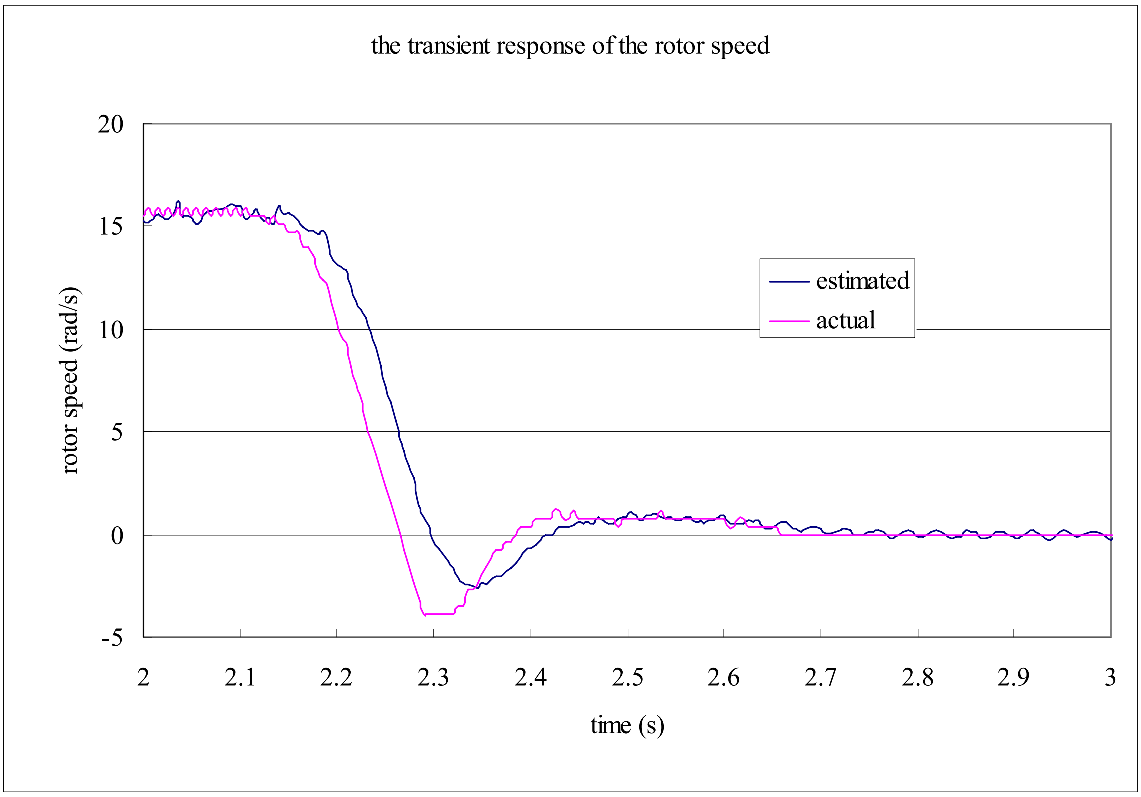


Figure 4.34 the step response of the estimated and actual rotor speeds

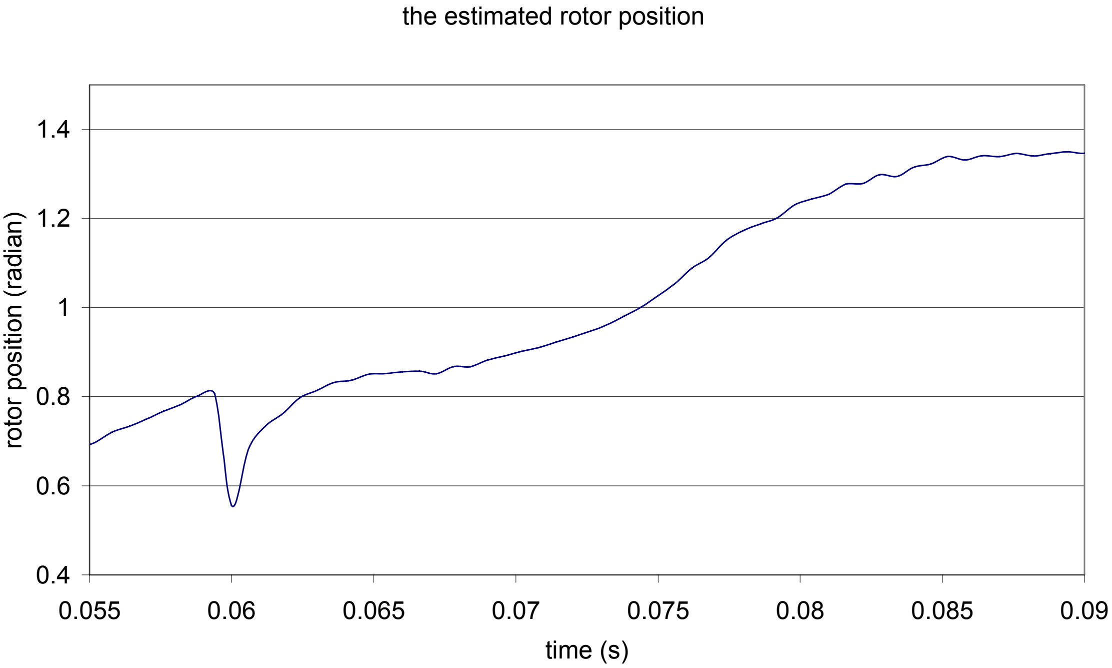


Figure 4.35 the estimated rotor position transient response

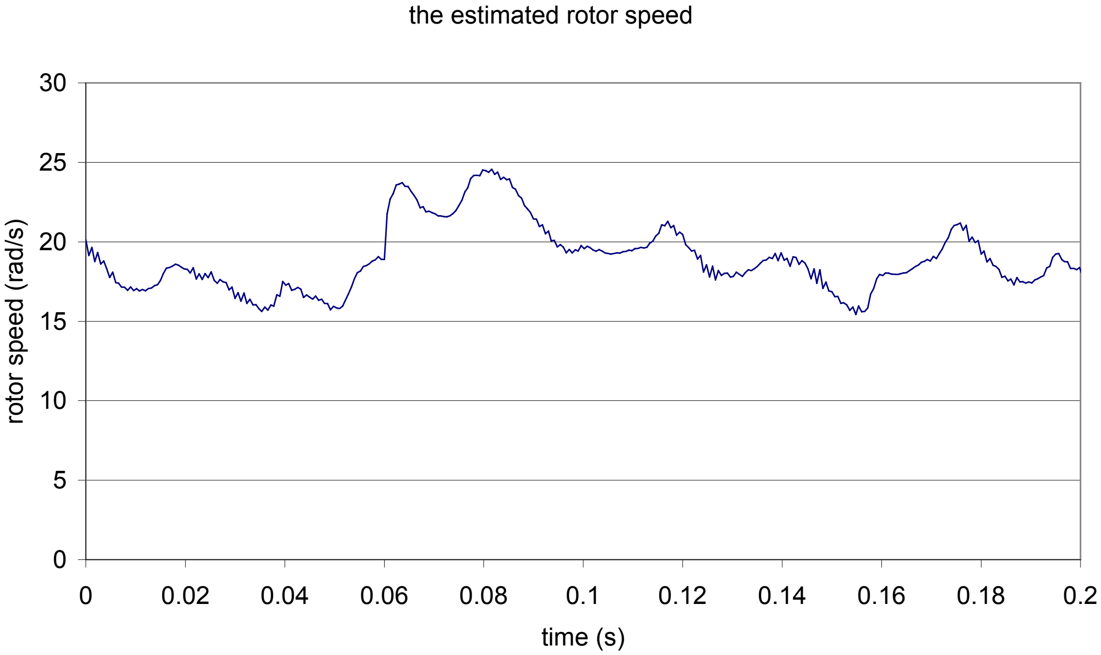


Figure 4.36 the estimated rotor speed transient response

Chapter 5 A simplified flux model based state observer sensorless control

In this chapter, a simplified analytical flux model of the SRM is developed. It is then implemented into an observer of a sensorless SRM control system for medium and high speed applications.

5.1 A simplified flux model

To compute the flux in real time, a simple SRM flux model is developed. The model accuracy has been traded for speed of computation so that the model run in a real time controller. A detailed SRM model has been proposed in [6, 7]. It is simplified by modeling the magnetization curve for the Fe with two piecewise linear curves. The piecewise analytical formula for flux linkage and instantaneous torque are obtained using basic electromagnetic theory. Because of the mathematical simplicity, the model provides rapid computation for a real time controller or state observer. This simplified model does not need any experimental data from the motor. It only needs the geometrical dimensions and magnetic parameters of the iron.

5.1.1 Introduction of flux models

The flux linked by a SRM phase is a function of its current and rotor position assuming the phases are independent. Computation of the flux linked by an individual phase of the SRM is a significant challenge because of its salient poles and the fact that iron saturation plays a critical role in the SRM's operation. Several papers have been published that present flux models for the SRM. Because of the complexity of these models for the flux linked by an SRM phase, they are not applicable for use in a rotor position state observer, which must run at high speed in a microprocessor or DSP. It is expected that an analytical SRM model will be a good choice for a state observer. The detailed analytical model presented previously is too unwieldy for a state observer [6, 7]. Curve fitting to obtain an analytical flux model for the SRM is another alternative. The disadvantage of curve fitting is that it requires significant data that must be obtained from measurements or from finite element analysis [31, 32]. Another approach combines the flux function versus phase current at the aligned rotor position, the flux function versus phase current at the unaligned rotor position, and a suitable angular function in between these rotor positions to obtain an analytical flux model [33]. The suitable angular function for use in between the aligned and unaligned rotor positions is complicated and nonlinear so that this model is a not a good choice to implement a state observer in a real time control system. Truncated Fourier series functions have also been used to express the inductance of a SRM phase, but this approach is also complicated and not a good choice to implement a state observer for sensorless control [34, 35].

5.1.2 Breaking the simplified flux model into two cases

The simplified analytical SRM model is constructed by considering two cases, the case where the stator poles of a given phase overlap with the rotor poles and the case where the stator poles of a given phase do not overlap with the rotor poles, according to the rotor position. The model is based on the basic magnetic field laws so that it does not need experimental data from the machine or any finite element analysis results. The model only needs the geometrical dimensions, number of turns, winding connections and the magnetic characteristics of the iron, all of which can be obtained from the manufacturer of the motor. The model runs rapidly in a microprocessor because it does not have any series, square root, sine or cosine functions to be computed, all of which take a long time to compute in a microprocessor.

5.1.2.1 The case with no stator and rotor pole overlap

When the stator and rotor poles do not overlap it is assumed that the phases are independent, there is no iron saturation and that the SRM phase inductance varies quadratically with the rotor position away from the unaligned position.

$$L_n(\theta) = L_u + (L_{po} - L_u)\frac{(\theta_u - \theta)^2}{(\theta_u - \theta_{pf})^2} \qquad 5\text{-}1$$

Here $L_n(\theta)$ is the inductance function applicable to the non-overlap case, L_u is the inductance at the unaligned rotor position, θ_u is the unaligned rotor position angle, and θ_{pf} is the effective rotor position boundary at which the rotor and stator poles just start to overlap. Thus the flux linked by a single SRM phase when there is no rotor and stator pole overlap is

$$\lambda(I_\phi,\theta) = L_n(\theta)\cdot I_\phi \qquad \text{5-2}$$

Here I_ϕ is the phase current. All of the quantities in (5-1) and (5-2) are known except θ and I_ϕ so that in this case the inductance parameters can be pre-computed before (5-2) is used in a state observer.

The torque can be expressed as in (5-3) by using conservation of energy

$$T(I_\phi,\theta) = -(L_{po} - L_u)\frac{(\theta_u - \theta)}{(\theta_u - \theta_{pf})^2} I_\phi{}^2 \qquad \text{5-3}$$

5.1.2.2 The overlap case

To model the flux with rotor and stator pole overlap, iron saturation needs to be considered. To simplify the flux model, the magnetization curve is simplified as two linear curves. One represents the unsaturated iron and the other represents the saturated iron. Since there is in general only a partial overlapped area between the stator poles of interest and the rotor poles, the total phase flux is broken into the main flux and fringing flux. The main flux passes from the stator to the rotor where the stator and rotor poles overlap. The fringing flux passes from the stator to the rotor where the stator and rotor poles do not overlap. The main flux and the fringing flux are computed separately but their equations have the same form.

5.1.2.2.1 The piece wise linear magnetization curves

With the stator and rotor poles overlapping, iron saturation in the SRM is important. The iron's magnetization curve is modeled as a piece wise linear curve.

$$\begin{aligned} B(H) &= \mu H & (H \le H_{sat}) \\ &= \mu H_{sat} + \mu_1 (H - H_{sat}) & (H > H_{sat}) \end{aligned} \qquad \text{5-4}$$

Here $B(H)$ is the flux density, which is a function of the magnetic intensity H. The parameter B_{sat} is the saturation flux density of the iron, μ_0 is the permeability of free space, μ_1 is the approximate saturated iron permeability, μ is the unsaturated permeability of the iron, and $H_{sat} = B_{sat} / \mu$ is the value of magnetic intensity at which saturation begins. The ideal magnetization curve and the approximate piece wise linear curve are shown in Figure 5.1 for SiFe.

5.1.2.2.2 Breaking the flux into main and fringing fluxes

With pole overlap the flux linked by a phase is broken into two parts, the main flux and the fringing flux. The contours of the two fluxes are shown in Figure 2.10. The main flux is due to the field that passes from the stator to the rotor where the stator and rotor poles overlap and thus the air gap is small. The fringing flux is due to the field that passes from the stator to the rotor where the stator and rotor poles do not overlap and thus where the air gap is larger.

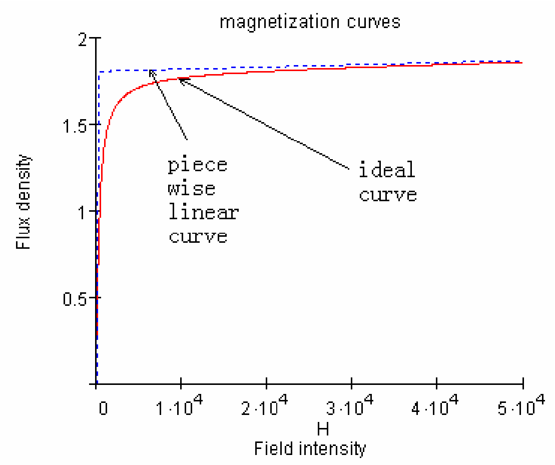


Figure 5.1 the ideal magnetization curve and piece wise linear curves

5.1.2.2.3 The main flux

According to Ampere's law,

$$H_{Fe,m} \cdot l_{Fe,m} + H_{g,m} \cdot g = N_p \cdot \frac{I_\phi}{npar} \qquad 5\text{-}5$$

Here $H_{Fe,m}$ is the H field in the iron part of the main flux contour, $H_{g,m}$ is the H field in the air gap part of the main flux contour, N_p is the number of turns per stator pole, $l_{Fe,m}$ is equal to half of the length of the iron part of the main flux path, g is the air gap on one side of the rotor between the rotor and stator poles where they overlap, and $npar$ is the number of pole windings in parallel for a phase.

The relationship between the B field, the H field in the iron and the H field in the air gap when the iron is not saturated is

$$B_m = \mu \cdot H_{Fe,m} = \mu_o \cdot H_{g,m} \qquad \text{5-6}$$

Iron saturation occurs when the B field in the iron reaches the value B_{sat}. At this point, the H field can be expressed as

$$B_{Fe,m} = B_{sat} = \mu H_{sat} \qquad \text{5-7}$$

The current at which iron saturation occurs for the main flux $I_{m,sat}$ can be obtained by combining (5-5), (5-6) and (5-7). It is a constant and given by (5-8).

$$I_{m,sat} = \frac{npar \cdot B_{sat}}{N_p \mu}\left(l_{Fe,m} + \frac{\mu}{\mu_o} g \right) \qquad \text{5-8}$$

The relationship between the H field in the iron and the B field when the iron is saturated can be expressed as

$$H_{Fe,m} = (B_m - B_{sat}) / \mu_1 + B_{sat} / \mu \qquad \text{5-9}$$

Combining (5-5), (5-6), and (5-9), the main flux B field with and without iron saturation is given by (5-10).

$$\begin{aligned} B_m(I_\phi) &= \frac{I_\phi}{npar} \cdot \frac{\mu N_p^{\,2}}{l_{fe,m} + \dfrac{\mu}{\mu_o} g} && (I_\phi \le i_{m,sat}) \\ &= N_p \mu_o \frac{\dfrac{N_p I_\phi}{npar} + l_{fe,m} B_{sat}\left(\dfrac{1}{\mu_1} - \dfrac{1}{\mu} \right)}{l_{fe,m} \dfrac{\mu_o}{\mu_1} + g} && (I_\phi > i_{m,sat}) \end{aligned} \qquad \text{5-10}$$

The main flux, denoted λ_m, is the main flux density times the overlapped area between the stator and rotor poles. It is expressed as

$$\lambda_m(I_\phi,\theta) = nser \cdot N_p \cdot B_m(I_\phi) \cdot R_g \cdot (\theta_{pf} - \theta) \cdot l_{stk} \cdot STF \qquad \text{5-11}$$

Here $nser$ is the number of windings in series for a phase, θ_{pf} is the effective stator pole width [7], STF is the stacking factor, l_{stk} is the length of the stack, and the expression $R_g \cdot (\theta_{pf} - \theta) \cdot l_{stk} \cdot STF$ is the overlapped area of the stator pole with a rotor pole. Note that the rotor position θ is zero when a pair of rotor poles is aligned with the phase's stator poles.

5.1.2.2.4 The fringing flux

Using the same process used for the main flux, the fringing field and fringing flux can be computed for the region in Figure 2.10 where the rotor and stator poles do not overlap. The results for the main field can be used with the air gap g replaced with the larger fringing air gap to obtain the fringing field. The fringing air gap is

$$g_f(\theta) = g + g_o \cdot \theta / \theta_{pf} \qquad \text{5-12}$$

The fringing air gap depends on rotor position where g_o is the air gap required to obtain the correct unsaturated inductance value at the rotor position where the rotor poles and the stator poles just start to overlap. With the above consideration the saturation current for the fringing flux $I_{f,sat}(\theta)$, which is a function of rotor position, can be expressed as

$$I_{f,sat}(\theta) = \frac{npar \cdot B_{sat}}{N_p \mu}\left(l_{Fe,f} + \frac{\mu}{\mu_o} g_f(\theta) \right) \qquad \text{5-13}$$

Here $l_{Fe,f}$ is equal to a half of the length of the iron part of the fringing flux contour. Similarly, the fringing flux density is

$$
\begin{aligned}
B_f(I_\phi,\theta) &= \frac{I_\phi}{npar}\cdot\frac{\mu N_p^{\,2}}{l_{fe,m}+\frac{\mu}{\mu_o}gf(\theta)} && (I_\phi \le I_{f,sat}(\theta)) \\
&= N_p\mu_o\frac{\frac{N_p I_\phi}{npar}+l_{fe,m}B_{sat}\left(\frac{1}{\mu_1}-\frac{1}{\mu}\right)}{l_{fe,m}\frac{\mu_o}{\mu_1}+gf(\theta)} && (I_\phi > I_{f,sat}(\theta))
\end{aligned}
\qquad \text{5-14}
$$

The fringing flux, denoted λ_f , is the fringing flux density times the non-overlapped area of the stator pole.

$$\lambda_f(I_\phi,\theta) = nser\cdot N_p\cdot B_f(I_\phi,\theta)\cdot R_g\cdot\theta\cdot l_{stk}\cdot STF \qquad \text{5-15}$$

5.1.2.2.5 The total flux when the rotor and the stator overlap

The total flux is the sum of the main flux and the fringing flux

$$\lambda_o(I_\phi,\theta) = \lambda_m(I_\phi,\theta)+\lambda_f(I_\phi,\theta) \qquad \text{5-16}$$

Here the function $\lambda_o(I_\phi,\theta)$ denotes the total flux when the rotor and the stator overlap.

The instantaneous torque can be obtained using conservation of energy and the flux shown in (5-16). The complete equations to compute the instantaneous torque for the overlap case are shown in appendix III.

5.1.3 Verifying the model with experiment measurement

The flux linked by a phase computed with the simplified model is compared with the measured flux from a 2 Hp peak power 4 phase 8/6 SRM with a maximum speed of 15,000 rpm. Figure 5.2 shows a comparison of the predicted flux linked by a phase computed with the simplified model with the measured flux from the commercial SRM. They match well. In the simplified model, the degree to which the iron saturates is determined by the value of μ_1. The values of the iron parameters chosen by the simplified model are μ=1000 μ_o, μ_1= 50 μ_o, B_{sat}=1.6T. (The results are presumably different for different phases)

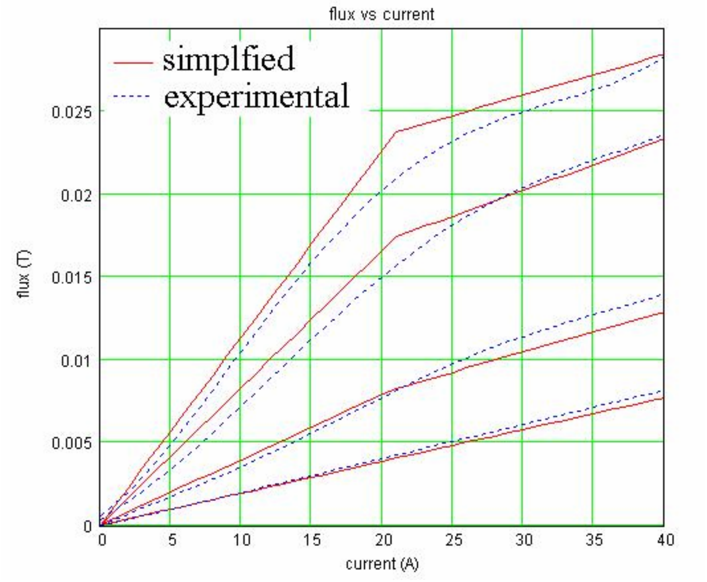


Figure 5.2 the comparison of the simplified model to the experimental data

5.2 The simplified flux model based observer

To estimate the rotor position of the SRM at higher speeds using the torque producing SRM current, another Luenberger observer is proposed. In the state observer, the rotor position and the rotor speed are the two states as in the low speed case. A new error function is defined to drive the observer using the simplified flux model. The error function is defined as

$$f(\theta,\hat{\theta}) = \sum_{j=1}^{4} sign(\hat{\theta}_j)\left(\hat{\lambda}_j(i_j,\hat{\theta}_j) - \lambda_j(i_j,\theta_j)\right) \qquad 5\text{-}17$$

where $\hat{\theta}_j$ is the estimated rotor position for the jth phase (it is zero when the jth phase is at the aligned position), i_j is the jth phase's measured current, $\hat{\lambda}_j$ is the jth phase's calculated flux linkage, which is predicted by the simplified model using the estimated rotor position and the measured phase current, λj is the jth phase's actual flux linkage. The actual flux linkage is a function of the actual rotor position θ_j and the measured phase current, but it is obtained by measuring the integration of the phase voltage. The function $sign(\hat{\theta}_j)$ is the sign of the estimated rotor position for the jth phase. It is 1 when $\hat{\theta}_j$ is greater than zero, -1 when $\hat{\theta}_j$ is less than zero, and 0 when $\hat{\theta}_j$ is equal to zero. Note that the estimated rotor position $\hat{\theta}_j$ is wrapped into the electrical period (from $-\theta_u$ to θ_u).

With the error definition, the error dynamic of the state observer becomes

$$\begin{bmatrix} \frac{de_\theta}{dt} \\ \frac{de_\omega}{dt} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} e_\theta \\ e_\omega \end{bmatrix} + \begin{bmatrix} H_3 \\ H_4 \end{bmatrix} \cdot \sum_{j=1}^{4} sign(\hat{\theta}_j)\left(\hat{\lambda}_j(i_j, \hat{\theta}_j) - \lambda_j(i_j, \theta_j)\right) \qquad 5\text{-}18$$

where H3 and H4 are the state observer gains. Because the integral of the measured voltage is used for the measured flux this state observer only works at speeds high enough that offset voltages in the integrator circuit do not create too large of an error over the integrating interval. In addition, it must be insured that the output of the integrator is set to zero each time the current goes to zero. This is realized by having the sense signals turn on MOSFETs to short the integrating capacitors. The integrator schematic in the experimental system is shown in Figure 3.13. The block diagram of the state observer is shown in Figure 5.3.

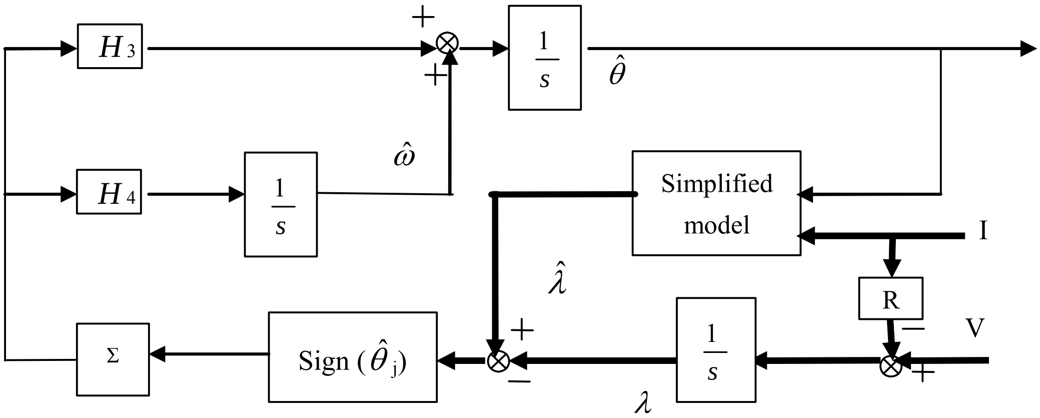


Figure 5.3 the sensorless control system using the simplified model

5.3 The error function with the simplified model

To verify that the error function defined in (5-17) has the desired properties to drive the observer, the error function value versus the rotor position curves when the rotor position error is 1, 2, 3, 4, 5, -1, -2, -3, -4, and -5 mechanical degrees are plotted in Figure 5.4. The error function is monotonic and negative when the rotor position error is positive. It is monotonic and positive when the rotor position error is negative. This insures that this error function can be used to drive the observer.

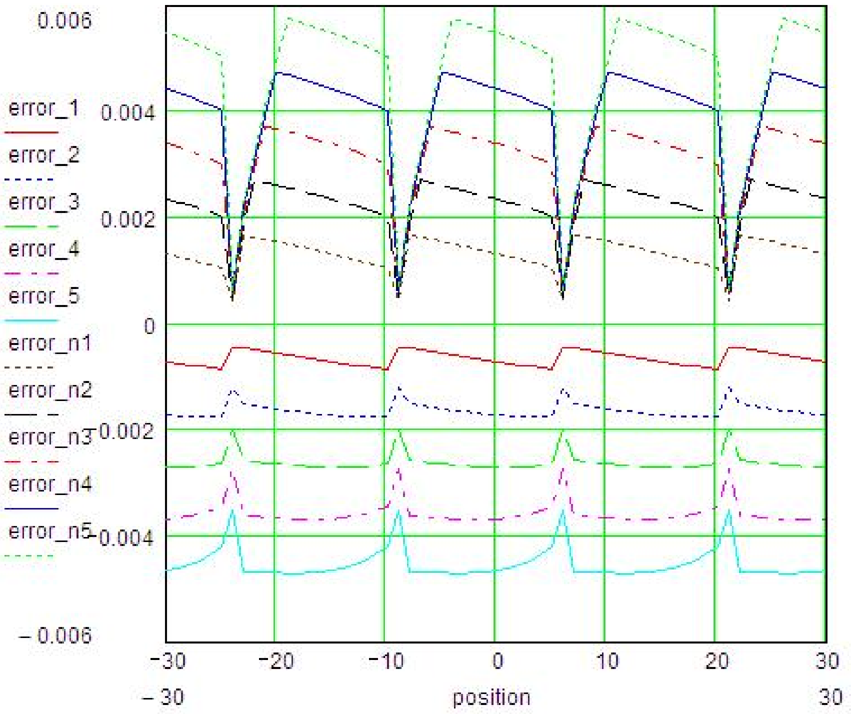


Figure 5.4 the error function based on the simplified flux model

The average value of the error function over an electrical period versus the position error is shown in Figure 5.5. The sign of the average value of the error function is opposite to the sign of the rotor position error except when the rotor position error is from -30 to -26 mechanical degrees. This can cause the sensorless control to fail when the estimated rotor position is 26 to 30 degrees less than the actual one. Note that the curve does not go exactly through the origin. This is because in the simulation file, the actual flux is obtained with the detailed flux model in [6, 7], while the calculated flux is predicted through the simplified flux model and there are differences between the flux computed with the two models at the same current and rotor position. This models the differences that exist between the actual SRM flux and the flux computed using the simplified SRM model for the same current and rotor position in the actual system.

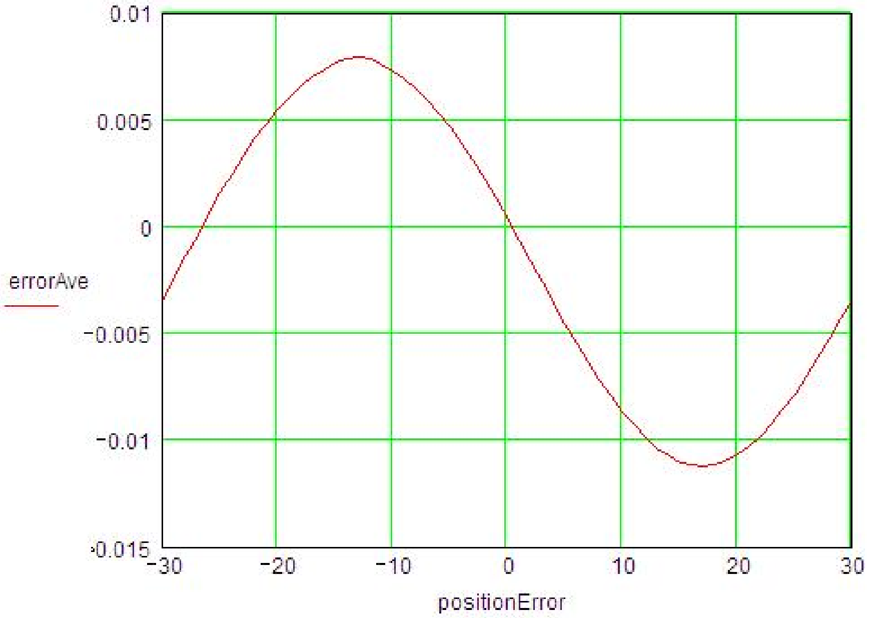


Figure 5.5 the average value of the error function versus the rotor position error

5.4 Stability and performance of the simplified flux model based observer

Since the control system time constant is generally greater than the period of time of the rotor position, the error average function can be used to characterize the error function when the rotor is running at sufficiently high speeds. To linearize the error average function, it is assumed that a zero function value occurs with zero angle error.

$$f(\theta_e) = -k_{ave2} \cdot \theta_e \qquad 5\text{-}19$$

In (5-19) k_{ave2} is the derivative of the error average function in Figure 5.5 with respect to the position error where the position error is zero.

Applying (5-19) into (5-18), the simplified flux model based observer becomes

$$\frac{d}{dt}\begin{bmatrix} e_\theta \\ e_\omega \end{bmatrix} = \begin{bmatrix} -H_3 \cdot k_{ave2} & 1 \\ -H_4 \cdot k_{ave2} & 0 \end{bmatrix} \begin{bmatrix} e_\theta \\ e_\omega \end{bmatrix} \qquad 5\text{-}20$$

The Eigen values of the characteristic matrix in (5-20) are

$$Eigen_{3,4} = -\frac{1}{2}H_3 \cdot k_{ave2} \pm \frac{1}{2}\sqrt{H_3{}^2 \cdot k_{ave2}{}^2 - 4H_4 \cdot k_{ave2}} \qquad 5\text{-}21$$

According to control theory, the Eigen values need to be negative real numbers or have negative real parts so that the system is exponentially stable.

$$real[Eigen_{3,4}] < 0 \qquad 5\text{-}22$$

Since $k_{ave2} > 0$, to satisfy (5-22), H_1 and H_2 are set as

$$H_3 > 0,\ H_4 > 0 \qquad 5\text{-}23$$

If $H_3{}^2 \cdot k_{ave2}{}^2 < 4H_4 \cdot k_{ave2}$, the two Eigen values are conjugate complex numbers with a common real part. If $H_3{}^2 \cdot k_{ave2}{}^2 = 4H_4 \cdot k_{ave2}$, the real part of the two Eigen values are identical and they are $H_3 \cdot k_{ave2}$. If $H_3{}^2 \cdot k_{ave2}{}^2 > 4H_4 \cdot k_{ave2}$, the two Eigen values are unequal real numbers. In this case, the settling time will be determined by the greater Eigen value, which is $-\frac{1}{2}H_3 \cdot k_{ave2} + \frac{1}{2}\sqrt{H_3{}^2 \cdot k_{ave2}{}^2 - 4H_4 \cdot k_{ave2}}$. The settling time is approximately five time constants and thus 5 over the real part of the Eigen value with the smaller real part.

$$T_{sattle} = \begin{cases} \dfrac{2.5}{H_3 \cdot k_{ave2}} & if \quad H_3{}^2 \cdot k_{ave2}{}^2 < 4H_4 \cdot k_{ave2} \\ \dfrac{2.5}{H_3 \cdot k_{ave2} - \sqrt{H_3{}^2 \cdot k_{ave2}{}^2 - 4H_4 \cdot k_{ave2}}} & if \quad H_3{}^2 \cdot k_{ave2}{}^2 > 4H_4 \cdot k_{ave2} \end{cases} \qquad 5\text{-}24$$

5.5 Simulation results

The simplified model is used in the state observer to predict the calculated flux while the detailed model is used to simulate the actual SRM. Simulation results for the estimated flux and the actual flux when the motor runs at 190 rpm are shown in Figure 5.6. The rotor position error between the estimated rotor position and the actual rotor position is shown in Figure 5.7. The spikes in the rotor position errors in Figure 5.7 are due to the fact that the estimated rotor position and the actual rotor position values are wrapped to stay within one electrical period and when this wrapping occurs the error momentarily is equal to the angle of one electrical cycle.

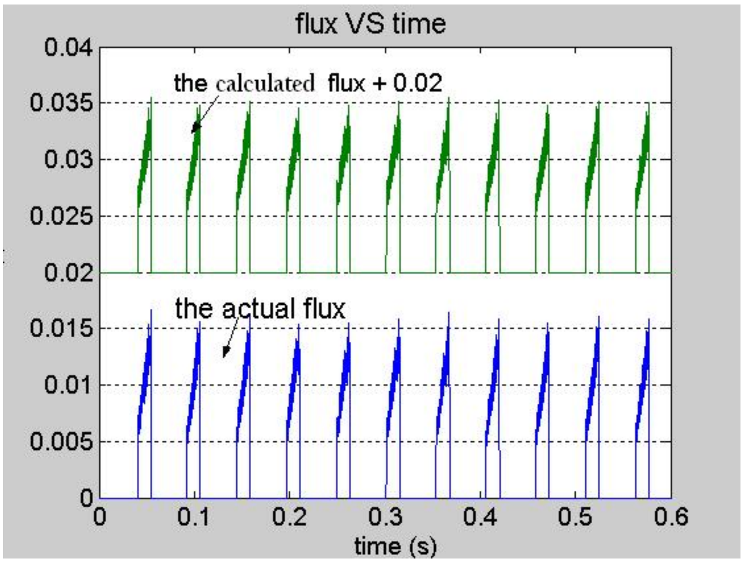


Figure 5.6 the calculated flux linkage and the actual flux linkage in the simplified flux model based observer sensorless control simulation with the SRM running at 190 rpm

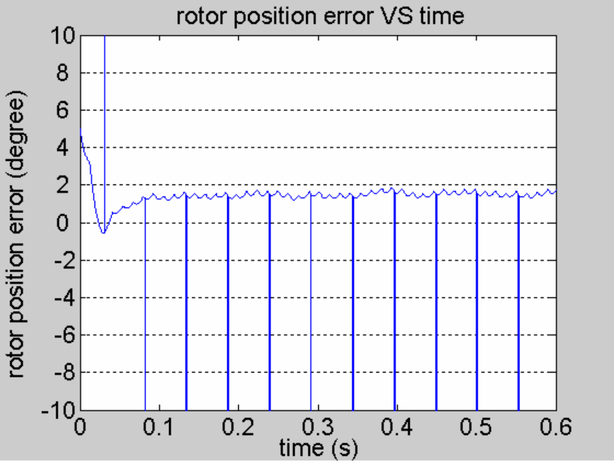


Figure 5.7 the rotor position error versus time in the simplified flux model based observer sensorless control simulation with the SRM running at 190 rpm

5.6 Experimental results

The experimental SRM drive system is shown in Figure 5.8.

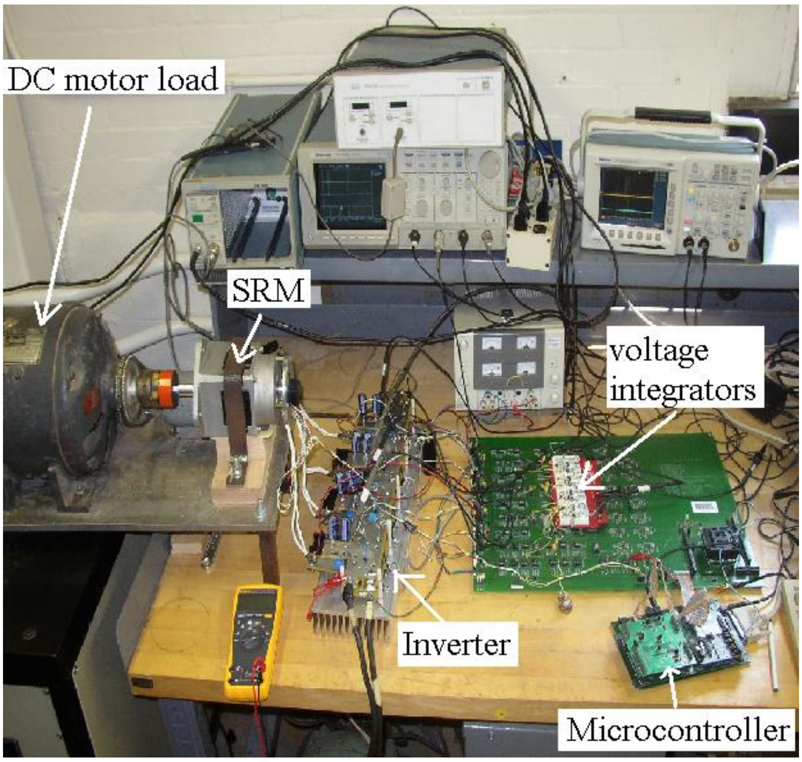


Figure 5.8 the experimental set-up of the simplified flux model based observer sensorless control system using the simplified model

The calculated and actual fluxes from the experimental system operating at 190 rpm are shown in Figure 5.9 and the rotor position error from the experimental system is shown in Figure 5.10. The spikes in the rotor position errors in and Figure 5.10 are due to the fact that the estimated rotor position and the actual rotor position values are wrapped to stay within an electrical period and when this wrapping occurs the error momentarily is equal to the angle of one electrical cycle. Because of the inductance asymmetry and the signal conditioning circuit asymmetry, the measured fluxes need to be adjusted by adding dc offsets and multiplying factors. After these adjustments, the electrical period is 60 degrees, but the period of 180 degrees is still used to be consistent with the inductance profile demodulator based observer. The experimental data is taken at steady state; while the simulation result shows the state observer's settling time with an initial position error of 5 degrees. Note that the estimated fluxes in Figure 5.6 and Figure 5.9 are shifted up 0.02 Weber to separate the flux plots for easier viewing. The angle error ripple in the actual system is larger and has a lower frequency (six flux pulses per cycle) than predicted by simulation. This is due in part to the fact that the phase inductance of the experimental machine depended on which rotor poles were near the stator pole of that phase. There were also differences in the phase inductance from phase to phase. None of these machine asymmetries were modeled in the simulation. Also note that the modeling errors apparent in Figure 5.2 do not create a noticeable error in the estimated flux and only a small error in the estimated rotor position. The phase current at high speeds is shown in Figure 5.11. The current does not chop as it does when the rotor runs at low speeds. Note that there are still sense pulses at high speed when there is no torque producing current even though they are not used. These sense pulses can be eliminated with an improved control design.

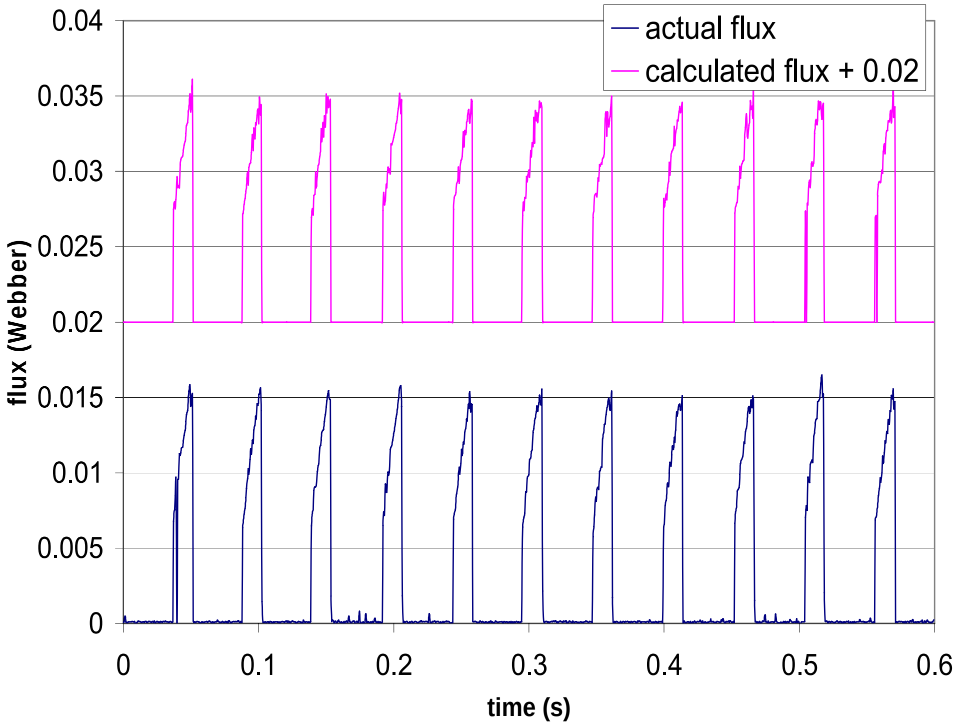


Figure 5.9 the estimated flux linkage and the actual flux linkage for phase A in the sensorless control experiment with the SRM operating at 190 rpm

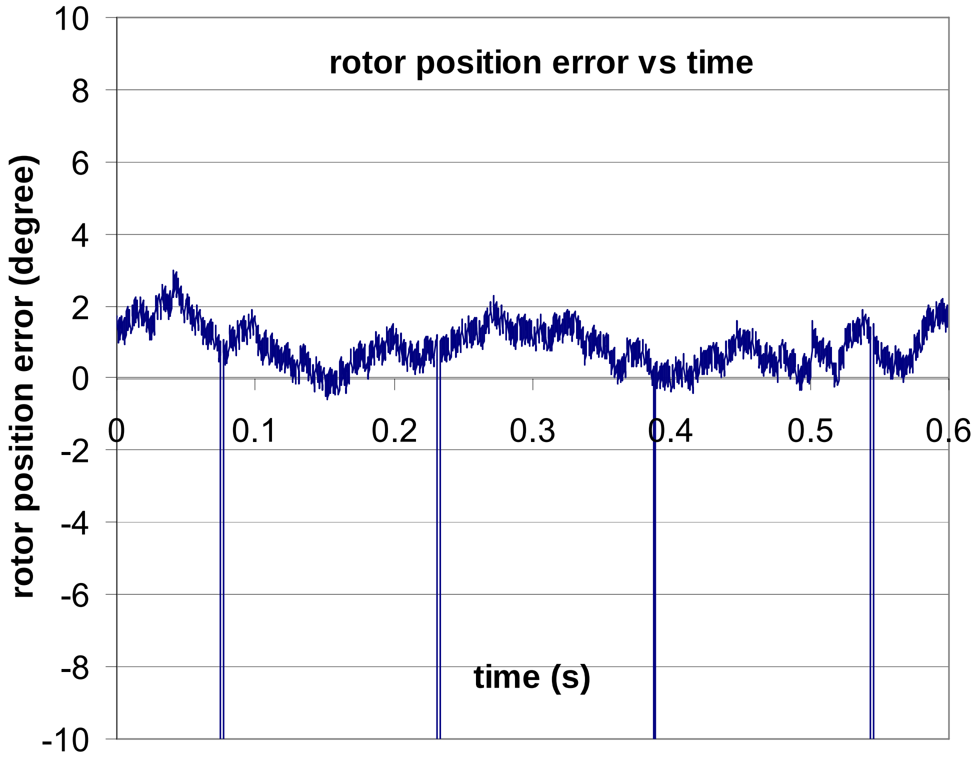


Figure 5.10 the rotor position error versus time in the simplified flux model based observer sensorless control experiment with the SRM operating at 190 rpm

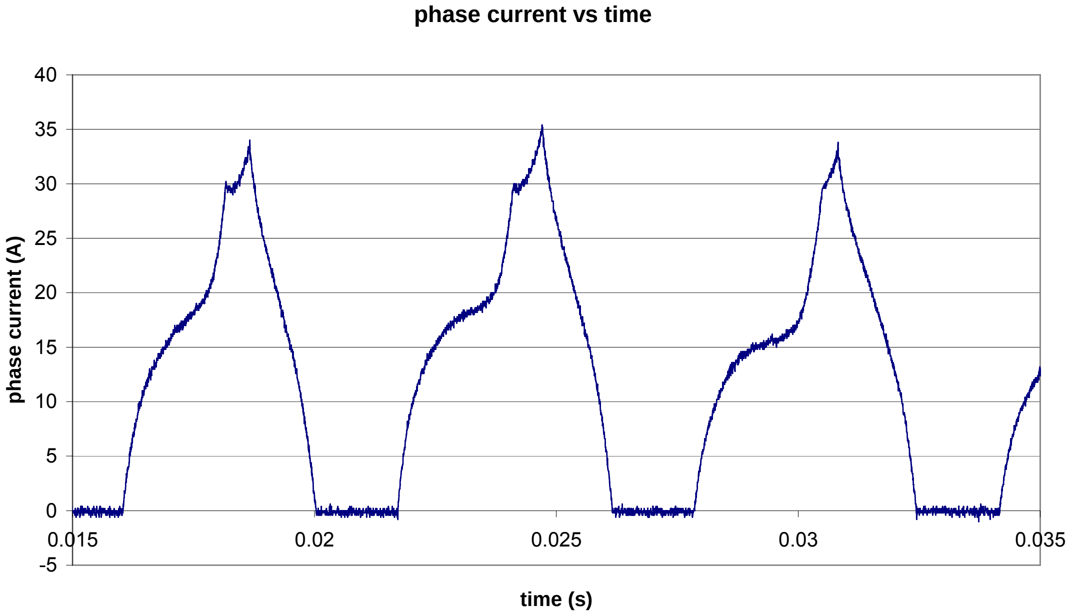


Figure 5.11 the phase current at high speeds

5.7 Speed limitation

The speed limitation of the observer depends on the digital sampling rate. Assume k samples need to be taken in a torque producing period for the rotor position to be determined. It is assumed that the torque producing region is 3/4 of electrical period, i.e. 45 degrees at high speeds. In this case the speed limit for the simplified flux model based observer is

$$\omega_{m\,Max3} = \frac{1}{8 \cdot k \cdot T_{sample}} \tag{5-25}$$

Because this observer needs to sample 8 channels, both currents and fluxes for 4 phases, the sampling time is noticeably greater than for the inductance profile demodulator based observer. In this observer, T_{sample}=100µs, so $\omega_{m\,Max3}$ =1250 when k is chosen to be 1. The actual speed response without load is shown in Figure 5.12. The estimated rotor position error is shown in Figure 5.13. The position error starts to be rather big when the rotor speed reaches 10,000 rpm so that the output torque decreases.

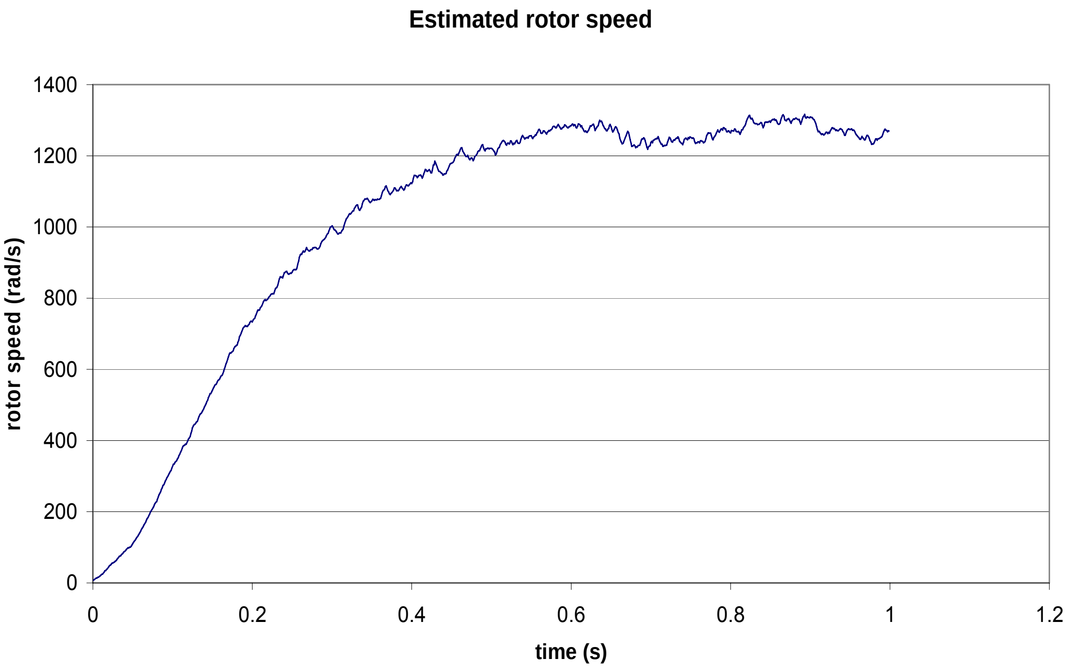


Figure 5.12 the estimated rotor speed limitation of the simplified flux based observer

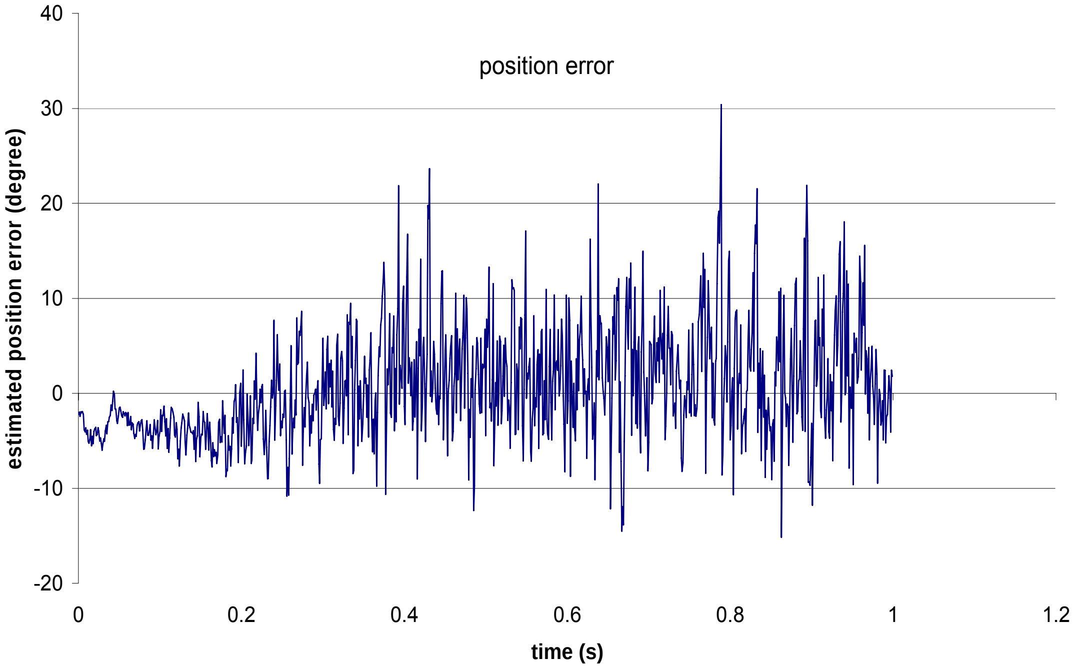


Figure 5.13 the estimated rotor position error of the simplified flux based observer

5.8 Transient response

Using Figure 5.5, the coefficient k_{ave2} is approximately 0.048. Then with H_3=2 x 10^3, H_4=1 x 10^4 and using (5-24), the settling time of the system is 0.25s. To obtain the transient response experimentally, the commanded current is increased suddenly when the rotor is running at a constant speed. The experimental transient response of the rotor position error and the rotor speed are shown in Figure 5.14 and Figure 5.15 respectively. In the experiment, the settling time for the rotor position and the rotor speed is 0.28s and 0.2s. When the gains H3 and H4 are changed to 2 x 10^4 and 2 x 10^6, the analytical settling time is 0.011s according to (5-24). The experimental transient response of the rotor speed is shown in Figure 5.16. In the experiment, the estimated rotor position was given a step change when the rotor was running at 200 rad/s. The estimated rotor speed settled down in 0.012s, which is close to the analytical value.

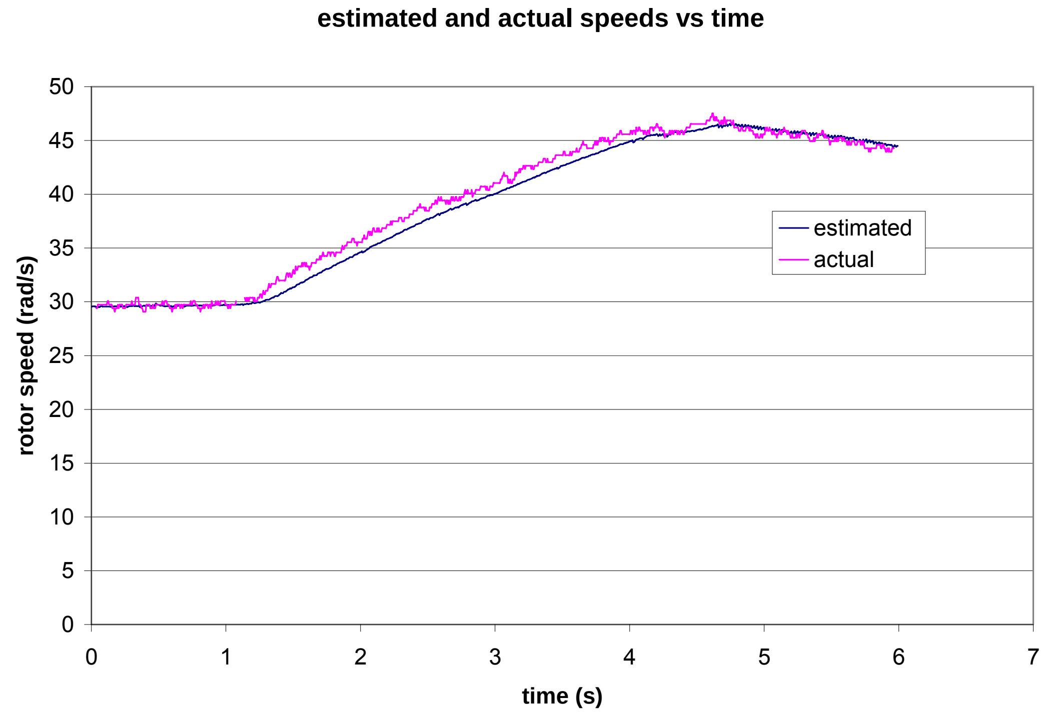


Figure 5.14 the transient response of the rotor speed with $H_3=2 \times 10^3$ and $H_4=2 \times 10^4$

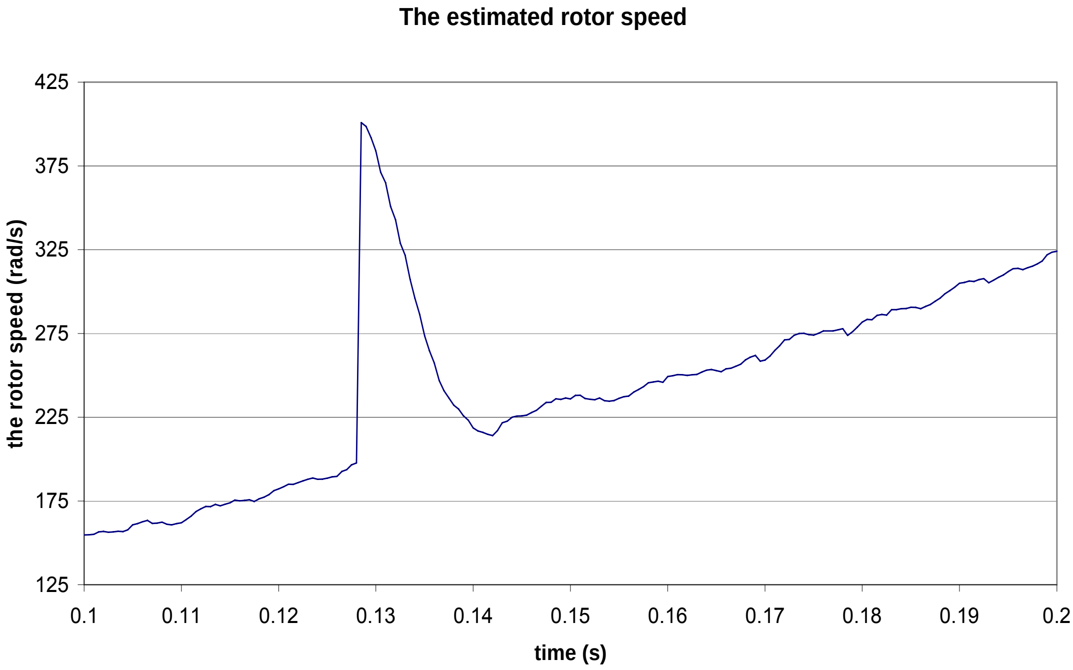


Figure 5.15 The transient response of the rotor speed with $H_3=2 \times 10^6$ and $H_4=2 \times 10^6$

5.9 Combination of the two strategies

The two control strategies are combined together so that full speed range operation with rated torque can be realized. At start and low speeds, the inductance profile demodulator based observer is applied. At medium and high speeds, the simplified flux model based observer is applied.

To determine the speed at which to switch between the two observers, namely the switching speed, it is assumed that the input voltage offset of the voltage integrator's operational amplifier is V_{offset}, and that the average integrator output voltage is V_{out}. To make sure the offset voltage does not create too large of an integrator error, it is required that the offset voltage integration error over a torque producing period is less than or equal to 1/10 of the output voltage. Here a torque producing period is assumed to θ_{period}. So the switch speed is

$$\omega_{m\,Switch} \geq \frac{\theta_{period} V_{offset}}{0.1 \cdot RC \cdot V_{out}} \qquad \text{5-26}$$

In the experiment, V_{out} is 1V, V_{offset}=0.02V, so the switch speed is 10 rad/s. Actually 100 rad/s is chosen to insure there is no integrator saturation due to unpredictable events and because the inductance profile demodulator based observer can operate to about 500 rad/s. The estimated rotor speed versus time curve in the experimental is shown in Figure 5.17. The spike at 100 rad/s is due to the switching from the inductance profile based observer to the simplified flux model based observer. The change in the torque (slope of the speed) due to the change in algorithm at 100rad/s is due to the change in control angles between the two algorithms. The error signal versus time curve is shown in Figure 5.18. The two observers are driven by two different error signals whose magnitudes are 100 times different in value, as shown in the figure. This is consistent with the order of magnitude difference in the settling times of the two observers with inductance profile demodulator based observer (low speed) being the faster one.

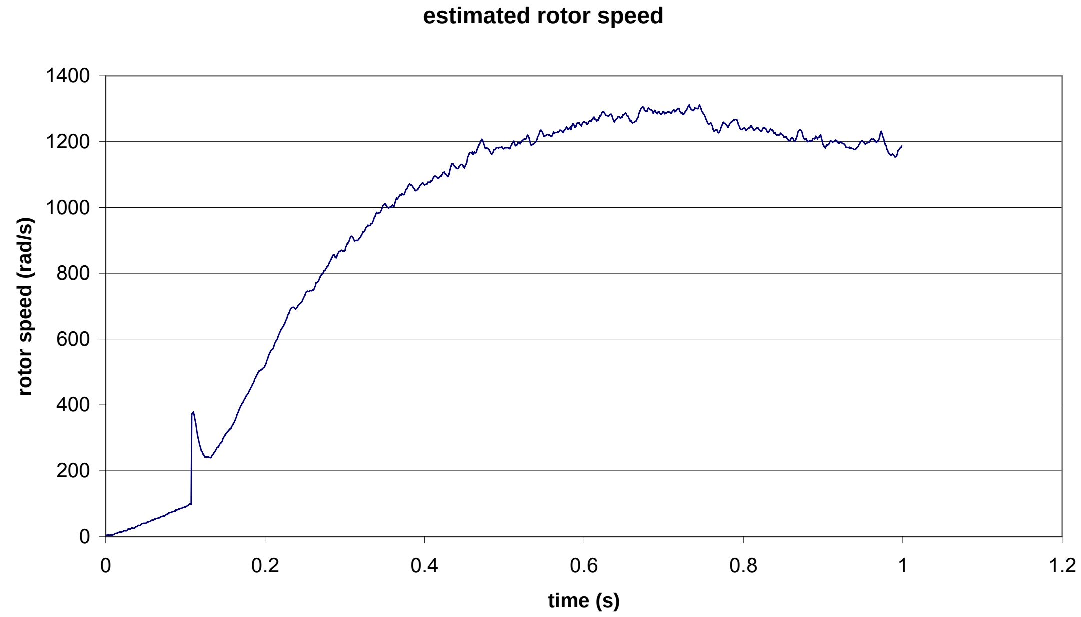


Figure 5.16 the estimated rotor speed with the combination of the two observers

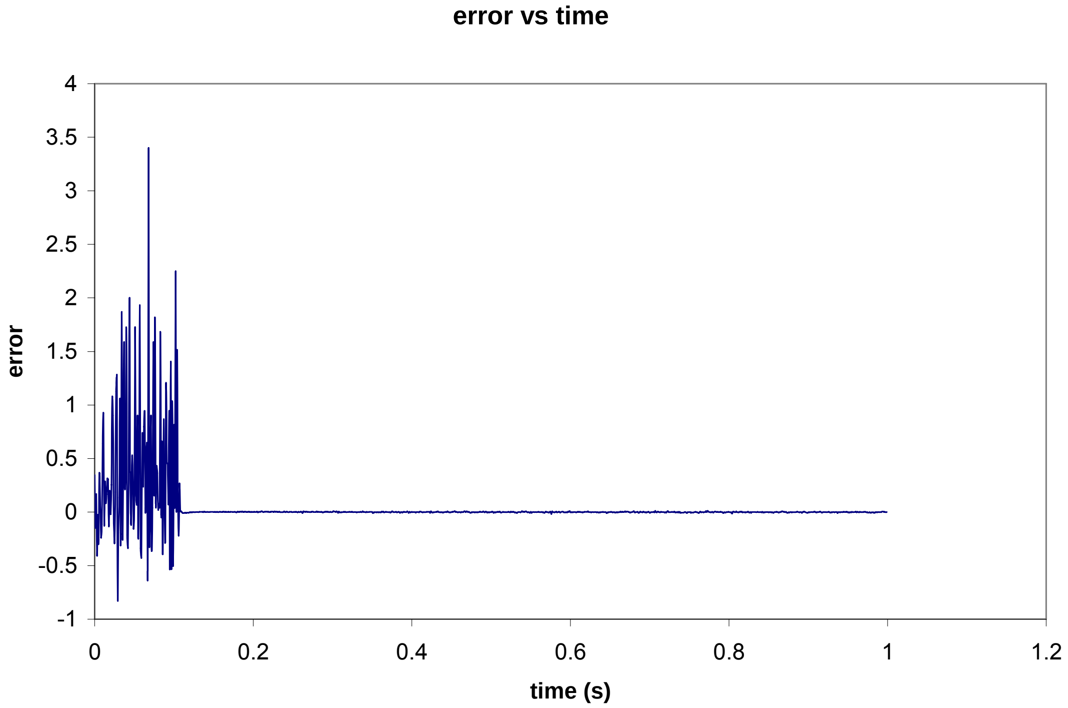


Figure 5.17 the error signal with the combination of the two observers

Contributation and future research

To eliminate the position sensor in the SRM drive system for applications that must operate from zero speed to high speeds with any torque output, a control strategy that combines two new position estimation methods has been proposed, designed and evaluated in this dissertation. In each method, a state observer is applied to estimate the rotor position and speed.

For low speeds, an inductance profile demodulator based observer is utilized. A relatively high frequency pulse voltage is applied to the idle phases producing a triangle shaped current that is modulated by the SRM's phase inductance. The current is then demodulated and used to produce an error between the actual and estimated rotor position. This error is used in an observer to estimate the rotor position. It is based on the fact that the inductance is a function of the rotor position and independent of current when the current is small and the iron does not saturate. The method works at zero speed to medium speeds from zero to rated torque. It is capable of 4 quadrant operation. It can find the rotor's position at startup without rotating the rotor. The factors that determine the maximum rotor speeds that the inductance profile demodulator based observer can work at have been given. The demodulator, basically a low pass filter, has an inherent time delay that results a large position error at high speeds. The larger position error can cause the sensorless control to fail. Another factor is that the sampling rate needs to be high enough to preserve the inductance profile information from the demodulated signal. It was verified in the dissertation using two different sampling rates. In the experiment that uses the lower sampling rate, the sampling rate limits the rotor speed, while in the higher sampling rate experiment, the time delay limits the rotor speed. The transient response experiment was conducted. The experimental settling time is 0.024s, while the settling time of the rotor speed and rotor position is 0.02 and 0.03 respectively. The system stability was investigated and researched with simulation and experiments. The system is robust because that it can work with inductance asymmetry.

For medium and high speeds, another observer, namely simplified flux model based observer, is used to estimate the rotor position and rotor speed. In this observer, the flux is calculated using the measured current and a simplified flux model. The simplified flux model is based on a published detailed analytical flux model. Because of its simplicity, it can be run in real time rapidly in a microprocessor. The calculated flux is compared with the measured flux to produce an error that drives the observer. It is also capable of 4 quadrant operation. The factor that determines the maximum speed this observer can work is basically the sampling rate. The stability and performance of the observer has been verified with simulation and experiments.

Since the two speed ranges overlap, the control system is capable of working from zero to high speed by switching between the two observers according to the estimated speed.

The inductance profile demodulator based observer is unique. It can figure out the rotor position at zero speed with the rated torque for any position without rotor rotation. It doesn't need additional inverter to inject the modulation current. Since there are always idle phases into which the modulation current can be injected, the rotor position can be figured out without rotor rotation for any position from the modulation current. Then the inverter can apply torque producing current to the phase(s) according to the estimated rotor position to produce torque. This is one of the requirements of the actuator application. The sensorless control can still be accomplished with the inductance asymmetry. This demonstrates the robustness of the system.

The simplified flux model based observer can work at higher speeds than its peers because it integrates the phase voltage to obtain the flux from hardware, instead of digitally. This significantly lowers the required sampling rate for the discrete control system. The simplified flux model is very simple for computation. There is no series, exponential functions, floating point division or square root function which need much more computation time than addition, sbustraction and multiplication. This remarkably lowers the required computation time. These two facts make the observer run at very high speeds.

The estimated rotor position from the two observers is very accurate from zero to medium speeds. The torque produced by the SRM falls off when the position estimators are in error and this loss of torque was used to determine the maximum speeds the position estimators are capable of operating at.

Several things can be done to extend the application and improve the performance of the control system. A feedback loop can be added to control the rotor speed by regulating the commanded current. The commanded current can be set by the microcontroller and converted into an analog signal using a digital-to-analog converter.

The fault tolerance of the SRM drive system at zero and low speeds needs to be investigated. There is an inherent problem for a 4-phase SRM to work at zero speed for any rotor position with one faulted phase. This is because the torque producing time windows of the remaining 3 phases don’t cover the full 360º of rotor rotation. . The position estimator developed has the potential of working at low speeds with one faulted phase and possibly two faulted phases. This potential needs to be investigated.

The on and off angles should be set as functions of the rotor speed, power supply voltage, commanded currents in simulation, but they are held constant in the experiment. This should be done in the future. The angles can also be optimized to reduce the torque ripple.

The program in the DSP, including the commutator, can be implemented in the FPGA to reduce the computation time. If the total DSP program can be implemented into the FPGA, it would work as a special purpose microprocessor, which can run much faster than the DSP.

The EMI filter was not implemented into the experimental system. This should be done in the future.

The inductance asymmetry can be added to the simulation model to describe the actual system better. Also the generating mode of SRM needs to be evaluated experimentally. The position estimator was designed to operate in all four quadrants. However no generating experiments were conducted. In this dissertation, two SRM drive system states, the rotor position and the rotor speed are used in the position estimator’s observer. The phase currents or phase fluxes can be used as additional states in the observer to potentially estimate the rotor position more accurately.

Appendices

Appendix I. VHDL code in the FPGA chip

The entities tree:

- i_reg_4ph_dsp
  - divider
    - counter
    - comparator_50
    - comparator_30
  - i_reg_4ph_comp
    - ph_ckt_new
  - interfact_fpga_dsp_3

Module i_reg_4ph_dsp

```
--
-- Updated on Sep 2 2004
-- Q1gate, Q1source, Q2gate, Q2source → Q1 and Q2
-- IorV(1 downto 0) added
-- Modin is used instead of IO_clock
--
-- Updated on Dec. 17 2004
-- Inverted Q1 and Q2 in ph_ckt_new module for level shifting
-- Inverted A, B and I signals from the op encoder for level shifting
--
-- Updated on Jan. 26 2005
-- Changed the polarity of Over_I_Probe to active low to drive
-- shutdown signal of the Dual Gate Drive chips.
--
-- Updated on Feb. 4 2005
-- Use IO_clock to generate modin signal (1M, 0.5D → 20KHz, 0.6D)
--
-- Updated on Sep. 12 2005
-- MorG was used to get rid of the modulation current when flux method is used
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity i_reg_4ph_dsp is
    Port ( modin : out std_logic; -- generated by IO_clock, test it with an output
                IO_clock : in std_logic;
                sense_low : in std_logic_vector(3 downto 0);
                I_chop : in std_logic_vector(3 downto 0);
                over_I : in std_logic_vector(3 downto 0);
                didtcomp : in std_logic_vector(3 downto 0);
                do_sense_high : out std_logic_vector(3 downto 0);
                didtout : out std_logic_vector(3 downto 0);
                Q1 : out std_logic_vector(3 downto 0);
                Q2 : out std_logic_vector(3 downto 0);
                I_sense : out std_logic_vector(3 downto 0);
                IorV : out std_logic_vector(1 downto 0);
                A : in std_logic;
                B : in std_logic;
                I : in std_logic;
                Timer1 : in std_logic;
```

```
                  DX : in std_logic;
                  DR : out std_logic;
                  CLKS : out std_logic;
                  CLKR : in std_logic;
                  FSR : in std_logic;
                  CLKX : in std_logic;
                  FSX : in std_logic;
                     enable_probe: out std_logic;
                  MorG_probe: out std_logic;
                     comin_probe: out std_logic_vector(3 downto 0);
                     over_I_probe: out std_logic
                  );
end entity i_reg_4ph_dsp;
architecture Behavioral of i_reg_4ph_dsp is
signal enable, MorG, over_I_out: std_logic;
signal comin, I_sense_temp: std_logic_vector(3 downto 0);
signal modin_tmp: std_logic;
signal IO_clock_c : std_logic;
signal modin_c : std_logic;
signal comin_c, sense_low_c, I_chop_c, over_I_c : std_logic_vector(3 downto 0);
signal Q1_c, Q2_c: std_logic_vector(3 downto 0);
signal IorV_c: std_logic_vector(1 downto 0);
    ■  signal do_sense_out_c: std_logic_vector(3 downto 0);
signal didtcomp_c, do_sense_high_c, didtout_c: std_logic_vector(3 downto 0);
signal A_c, B_c, I_c: std_logic;
signal Timer1_c, DX_c, DR_c: std_logic;
signal CLKS_c, CLKR_c, FSR_c, CLKX_c, FSX_c: std_logic;
    ■  Probe signals
signal enable_probe_c, MorG_probe_c, over_I_probe_c: std_logic;
signal comin_probe_c, I_sense_c: std_logic_vector(3 downto 0);
signal A_inv, B_inv, I_inv: std_logic;
component GL33
    port(
     GL          :out  std_logic;
     PAD              :in    std_logic);
end component;
component IB33
    port(PAD : in std_logic := ‘U’; Y : out std_logic);
end component;
component OB33PH
    port(PAD : out std_logic; A : in std_logic := ‘U’);
end component;
component i_reg_4ph_comp is
    Port ( modin : in std_logic;
                  comin : in std_logic_vector(3 downto 0);
                  sense_low : in std_logic_vector(3 downto 0);
                  I_chop : in std_logic_vector(3 downto 0);
                  over_I : in std_logic_vector(3 downto 0);
                  didtcomp : in std_logic_vector(3 downto 0);
                  enable : in std_logic;
                  MorG : in std_logic;
                  Q1: out std_logic_vector(3 downto 0);
                  Q2: out std_logic_vector(3 downto 0);
                  over_I_out : out std_logic;
                  do_sense_out : out std_logic_vector(3 downto 0);
                  do_sense_high : out std_logic_vector(3 downto 0);
                  didtout : out std_logic_vector(3 downto 0));
end component i_reg_4ph_comp;
component interface_fpga_dsp_3 is
    Port ( A : in std_logic;
                  B : in std_logic;
                  I : in std_logic;
                  Enable : out std_logic;
                   MorG: out std_logic;
                     omm. : out std_logic_vector(3 downto 0);
                  IorV : out std_logic_vector(1 downto 0);
                  over_I : in std_logic;
                  I_sense : in std_logic_vector(3 downto 0);
                   clock: in std_logic;      -- to dffs for holding output signals
                  Timer1 : in std_logic; -- select line 1 of mux
                  DX : in std_logic;         -- select line 0 of mux
                  DR : out std_logic;        -- output line 1 of mux
                  CLKS : out std_logic;    -- output line 0 of mux
                  CLKR : in std_logic;--    input line 1 of decoder
                  FSR: in std_logic;    --    input line 0 of decoder
                  CLKX : in std_logic;-- select line 1 of decoder
                  FSX : in std_logic);-- select line 0 of decoder
end component interface_fpga_dsp_3;
component divider is
    port (clockin, enable: in std_logic;
             clockout: out std_logic);
end component divider;
begin
    ■   test probes
enable_probe_c <= enable;
MorG_probe_c      <=    MorG;
over_I_probe_c <= not over_I_out;
comin_probe_c    <= comin;
I_sense_c           <= I_sense_temp;
A_inv                  <= not A_c;
B_inv                  <= not B_c;
I_inv                 <= not I_c;
-- Define input and output pads
-- common inputs
    IO_clock_pad : GL33
       port map(PAD => IO_clock, GL => IO_clock_c);
    A_pad : IB33
       port map(PAD => A, Y => A_c);
    B_pad : IB33
       port map(PAD => B, Y => B_c);
    I_pad : IB33
       port map(PAD => I, Y => I_c);
    Timer1_pad : IB33
       port map(PAD => Timer1, Y => Timer1_c);
    DX_pad : IB33
       port map(PAD => DX, Y => DX_c);
    CLKR_pad : IB33
       port map(PAD => CLKR, Y => CLKR_c);
    FSR_pad : IB33
       port map(PAD => FSR, Y => FSR_c);
    CLKX_pad : IB33
       port map(PAD => CLKX, Y => CLKX_c);
    FSX_pad : IB33
       port map(PAD => FSX, Y => FSX_c);
    ■   common output(s)
    modin_pad : OB33PH      -- Added for testing modin generated by IO_clock
       port map(PAD => modin, A => modin_c);
    DR_pad : OB33PH
       port map(PAD => DR, A => DR_c);
    CLKS_pad : OB33PH
```

```
   port map(PAD => CLKS, A => CLKS_c);
IorV_0_pad : OB33PH
   port map(PAD => IorV(0), A => IorV_c(0));
IorV_1_pad : OB33PH
   port map(PAD => IorV(1), A => IorV_c(1));
MorG_probe_pad : OB33PH
   port map(PAD => MorG_probe, A => MorG_probe_c);
enable_probe_pad : OB33PH
   port map(PAD => enable_probe, A => enable_probe_c);
comin_probe_0_pad : OB33PH
   port map(PAD => comin_probe(0), A => comin_probe_c(0));
comin_probe_1_pad : OB33PH
   port map(PAD => comin_probe(1), A => comin_probe_c(1));
comin_probe_2_pad : OB33PH
   port map(PAD => comin_probe(2), A => comin_probe_c(2));
comin_probe_3_pad : OB33PH
   port map(PAD => comin_probe(3), A => comin_probe_c(3));
over_I_probe_pad : OB33PH
   port map(PAD => over_I_probe, A => over_I_probe_c);
■  phase #0 inputs and outputs
sense_low_0_pad : IB33
   port map(PAD => sense_low(0), Y => sense_low_c(0));
I_chop_0_pad : IB33
   port map(PAD => I_chop(0), Y => I_chop_c(0));
over_I_0_pad : IB33
   port map(PAD => over_I(0), Y => over_I_c(0));
didtcomp_0_pad : IB33
   port map(PAD => didtcomp(0), Y => didtcomp_c(0));
Q1_0_pad : OB33PH
   port map(PAD => Q1(0), A => Q1_c(0));
Q2_0_pad : OB33PH
   port map(PAD => Q2(0), A => Q2_c(0));
I_sense_0_pad : OB33PH
   port map(PAD => I_sense(0), A => I_sense_c(0));
do_sense_high_0_pad : OB33PH
   port map(PAD => do_sense_high(0), A => do_sense_high_c(0));
didt_out_0_pad : OB33PH
   port map(PAD => didtout(0), A => didtout_c(0));

■  phase #1 inputs and outputs
sense_low_1_pad : IB33
   port map(PAD => sense_low(1), Y => sense_low_c(1));
I_chop_1_pad : IB33
   port map(PAD => I_chop(1), Y => I_chop_c(1));
over_I_1_pad : IB33
   port map(PAD => over_I(1), Y => over_I_c(1));
didtcomp_1_pad : IB33
   port map(PAD => didtcomp(1), Y => didtcomp_c(1));
Q1_1_pad : OB33PH
   port map(PAD => Q1(1), A => Q1_c(1));
Q2_1_pad : OB33PH
   port map(PAD => Q2(1), A => Q2_c(1));
I_sense_1_pad : OB33PH
   port map(PAD => I_sense(1), A => I_sense_c(1));
do_sense_high_1_pad : OB33PH
   port map(PAD => do_sense_high(1), A => do_sense_high_c(1));
didt_out_1_pad : OB33PH
   port map(PAD => didtout(1), A => didtout_c(1));
■  phase #2 inputs and outputs
sense_low_2_pad : IB33
```

```
        port map(PAD => sense_low(2), Y => sense_low_c(2));
    I_chop_2_pad : IB33
        port map(PAD => I_chop(2), Y => I_chop_c(2));
    over_I_2_pad : IB33
        port map(PAD => over_I(2), Y => over_I_c(2));
    didtcomp_2_pad : IB33
        port map(PAD => didtcomp(2), Y => didtcomp_c(2));
    Q1_2_pad : OB33PH
        port map(PAD => Q1(2), A => Q1_c(2));
    Q2_2_pad : OB33PH
        port map(PAD => Q2(2), A => Q2_c(2));
    I_sense_2_pad : OB33PH
        port map(PAD => I_sense(2), A => I_sense_c(2));
    do_sense_high_2_pad : OB33PH
        port map(PAD => do_sense_high(2), A => do_sense_high_c(2));
    didt_out_2_pad : OB33PH
        port map(PAD => didtout(2), A => didtout_c(2));
    ■  phase #3 inputs and outputs
    sense_low_3_pad : IB33
        port map(PAD => sense_low(3), Y => sense_low_c(3));
    I_chop_3_pad : IB33
        port map(PAD => I_chop(3), Y => I_chop_c(3));
    over_I_3_pad : IB33
        port map(PAD => over_I(3), Y => over_I_c(3));
    didtcomp_3_pad : IB33
        port map(PAD => didtcomp(3), Y => didtcomp_c(3));
    Q1_3_pad : OB33PH
        port map(PAD => Q1(3), A => Q1_c(3));
    Q2_3_pad : OB33PH
        port map(PAD => Q2(3), A => Q2_c(3));
    I_sense_3_pad : OB33PH
        port map(PAD => I_sense(3), A => I_sense_c(3));
    do_sense_high_3_pad : OB33PH
        port map(PAD => do_sense_high(3), A => do_sense_high_c(3));
    didt_out_3_pad : OB33PH
        port map(PAD => didtout(3), A => didtout_c(3));
Modindff: process (IO_clock_c) is
begin
if (rising_edge(IO_clock_c)) then
modin_c <= not modin_tmp;
end if;
end process;
U0: divider
port map (clockin => IO_clock_c, enable => enable, clockout=> modin_tmp);
U1: i_reg_4ph_comp
port map (modin => modin_c, comin=>comin, sense_low=>sense_low_c,
          I_chop=>I_chop_c,          over_I=>over_I_c,          didtcomp=>didtcomp_c,
enable=>enable,MorG=>MorG,Q2source=>Q2source_c,Q1=>Q1_c, Q2=>Q2_c,
over_I_out=>over_I_out,do_sense_out=>I_sense_temp,do_sense_high=>do_sense_high_c,
didtout=>didtout_c);
U2: interface_fpga_dsp_3
port map (A=>A_inv, B=>B_inv, I=>I_inv, Enable=>enable,
             MorG=>MorG,    omm.=>comin, IorV=>IorV_c, over_I=>over_I_out,
             I_sense=>I_sense_temp, clock=>IO_clock_c, Timer1=>Timer1_c,
              DX=>DX_c,   DR=>DR_c,   CLKS=>CLKS_c,   CLKR=>CLKR_c,   FSR=>FSR_c,
CLKX=>CLKX_c, FSX=>FSX_c);
end architecture Behavioral;

Entity Divider
-- divider.vhd
-- To divide the IO_clock signal by 50 to get Modin signal
--                          1MHz           →                    20KHz
--          ----------------------------------------
--          |                                      |
--         \/                                      |
--         reset                                   |
-- IO_clock->counter -> comparator_50 ->DFF ->INV ----\
--                       |                                 RSFF → modin
--                       -> comparator_30 ->DFF ->INV ----/
-- duty cycle could be random, here 0.6 is chosen
-- created on 02/03/04
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity divider is
      port (clockin, enable: in std_logic;
               clockout: out std_logic);
end entity divider;
architecture RTL of divider is
signal comp1,comp2,d1,d2,s,r,srout: std_logic;
signal pre_state,next_state: std_logic;
signal sr: std_logic_vector(2 downto 0);
signal Q: std_logic_vector(5 downto 0);
signal temp_aclr: std_logic;
component counter is
      port(Enable, Aclr, Clock : in std_logic; Q : out
            std_logic_vector(5 downto 0)) ;
end component counter;
component comparator_50 is
      port( DataA : in std_logic_vector(5 downto 0); AEB : out
            std_logic) ;
end component comparator_50;
component comparator_30 is
      port( DataA : in std_logic_vector(5 downto 0); AEB : out
            std_logic) ;
end component comparator_30;
begin
sr<=s & r & pre_state;
temp_aclr<= d1 and enable;
counter_0: counter
      port map(Enable=>enable, Aclr=>temp_aclr, Clock=>clockin, Q=>Q);
comp_50_0: comparator_50
      port map(DataA=>Q, AEB=>comp1);
comp_30_0: comparator_30
      port map(DataA=>Q, AEB=>comp2);
s<=not d1;
r<=not d2;
clockout<=srout;
dff1: process(clockin)
begin
if(rising_edge(clockin)) then
d1<=comp1;
end if;
end process dff1;
dff2: process(clockin)
begin
if(rising_edge(clockin)) then
d2<=comp2;
end if;
end process dff2;
------------- describe S R Flip Flops-------------
srff_comb: process (sr) is
begin
case sr is
when”000” => srout<=’0’;    -- next_state is deleted because
when”001” => srout<=’1’;    -- it is the same as srout
when”010” => srout<=’0’;
when”011” => srout<=’0’;
when”100” => srout<=’1’;
when”101” => srout<=’1’;
when”110” => srout<=’0’;
when”111” => srout<=’0’;
when others=> srout<=’0’;
end case;
end process srff_comb;
srff_dff: process(clockin) is
begin
if (rising_edge(clockin)) then
pre_state<=srout;
end if;
end process srff_dff;
end architecture RTL;
```

Entity counter

- Version: 6.0 Production 6.0.0.133

```
library ieee;
use ieee.std_logic_1164.all;
library a500K;
entity counter is
    port(Enable, Aclr, Clock : in std_logic; Q : out
        std_logic_vector(5 downto 0)) ;
end counter;
architecture DEF_ARCH of    counter is
    component AND3
        port(A, B, C : in std_logic := ‘U’; Y : out std_logic) ;
    end component;
    component AND2
        port(A, B : in std_logic := ‘U’; Y : out std_logic) ;
    end component;
    component XOR2
        port(A, B : in std_logic := ‘U’; Y : out std_logic) ;
    end component;
    component DFFC
        port(CLK, D, CLR : in std_logic := ‘U’; Q : out std_logic
        ) ;
    end component;
   component INV
        port(A : in std_logic := ‘U’; Y : out std_logic) ;
    end component;
    signal Q_0_net, Q_1_net, Q_2_net, Q_3_net, Q_4_net, Q_5_net,
        Sum_1_net, Sum_2_net, Sum_3_net, Sum_4_net, Sum_5_net,
        Sum_6_net, Sum_0_net, AND2_1_Y, AND3_0_Y, AND2_2_Y,
        AND3_1_Y, AND3_2_Y, AND2_0_Y, INV_0_Y : std_logic ;
    begin
    Q(0) <= Q_0_net;
    Q(1) <= Q_1_net;
    Q(2) <= Q_2_net;
    Q(3) <= Q_3_net;
    Q(4) <= Q_4_net;
    Q(5) <= Q_5_net;
    AND3_2 : AND3
      port map(A => Q_2_net, B => Q_3_net, C => Q_4_net, Y =>
        AND3_2_Y);
    AND2_0 : AND2
      port map(A => AND3_0_Y, B => AND3_2_Y, Y => AND2_0_Y);
    XOR2_Sum_6_inst : XOR2
      port map(A => AND2_0_Y, B => Q_5_net, Y => Sum_6_net);
    DFFC_Q_3_inst : DFFC
      port map(CLK => Clock, D => Sum_4_net, CLR => INV_0_Y, Q =>
        Q_3_net);
    INV_0 : INV
      port map(A => Aclr, Y => INV_0_Y);
    AND2_1 : AND2
      port map(A => Enable, B => Q_0_net, Y => AND2_1_Y);
    AND3_0 : AND3
      port map(A => Enable, B => Q_0_net, C => Q_1_net, Y =>
        AND3_0_Y);
    XOR2_Sum_1_inst : XOR2
      port map(A => Enable, B => Q_0_net, Y => Sum_1_net);
    AND3_1 : AND3
      port map(A => AND3_0_Y, B => Q_2_net, C => Q_3_net, Y =>
        AND3_1_Y);
    DFFC_Q_5_inst : DFFC
      port map(CLK => Clock, D => Sum_6_net, CLR => INV_0_Y, Q =>
        Q_5_net);
    XOR2_Sum_2_inst : XOR2
      port map(A => AND2_1_Y, B => Q_1_net, Y => Sum_2_net);
    DFFC_Q_1_inst : DFFC
      port map(CLK => Clock, D => Sum_2_net, CLR => INV_0_Y, Q =>
        Q_1_net);
    DFFC_Q_2_inst : DFFC
      port map(CLK => Clock, D => Sum_3_net, CLR => INV_0_Y, Q =>
        Q_2_net);
    XOR2_Sum_3_inst : XOR2
      port map(A => AND3_0_Y, B => Q_2_net, Y => Sum_3_net);
    XOR2_Sum_4_inst : XOR2
      port map(A => AND2_2_Y, B => Q_3_net, Y => Sum_4_net);
    AND2_2 : AND2
      port map(A => AND3_0_Y, B => Q_2_net, Y => AND2_2_Y);
    XOR2_Sum_5_inst : XOR2
      port map(A => AND3_1_Y, B => Q_4_net, Y => Sum_5_net);
    DFFC_Q_4_inst : DFFC
      port map(CLK => Clock, D => Sum_5_net, CLR => INV_0_Y, Q =>
        Q_4_net);
    DFFC_Q_0_inst : DFFC
      port map(CLK => Clock, D => Sum_1_net, CLR => INV_0_Y, Q =>
        Q_0_net);
--   software bug, not in use
--     INV_Sum_0_inst : INV
--         port map(A => Enable, Y => Sum_0_net);
end DEF_ARCH;
```

Entity comparator_50

- ■ Version: 6.0 Production 6.0.0.133

```
library ieee;
use ieee.std_logic_1164.all;
library a500K;
entity comparator_50 is
    port( DataA : in std_logic_vector(5 downto 0); AEB : out
         std_logic) ;
end comparator_50;
architecture DEF_ARCH of    comparator_50 is
    component NAND3
         port(A, B, C : in std_logic := ‘U’; Y : out std_logic) ;
    end component;
    component AND3FTT
         port(A, B, C : in std_logic := ‘U’; Y : out std_logic) ;
    end component;
    component AND3FFT
         port(A, B, C : in std_logic := ‘U’; Y : out std_logic) ;
    end component;
--    software bug: ANDTree_Data_2_net is not used
--       signal Temp_0_net, Temp_1_net, ANDTree_Data_2_net : std_logic ;
    signal Temp_0_net, Temp_1_net : std_logic ;
    begin
    NAND3_AEB : NAND3
       port map(A => Temp_0_net, B => Temp_1_net,
       C => ‘1’, Y => AEB);
    AND3FTT_Temp_1_inst : AND3FTT
       port map(A => DataA(3), B => DataA(4), C => DataA(5),
       Y => Temp_1_net);
    AND3FFT_Temp_0_inst : AND3FFT
       port map(A => DataA(0), B => DataA(2), C => DataA(1),
       Y => Temp_0_net);
end DEF_ARCH;
```

Entity comparator_30

- Version: 6.0 Production 6.0.0.133

```
library ieee;
use ieee.std_logic_1164.all;
library a500K;
entity comparator_30 is
       port( DataA : in std_logic_vector(5 downto 0); AEB : out
            std_logic) ;
end comparator_30;
architecture DEF_ARCH of    comparator_30 is
       component AND3FTT
            port(A, B, C : in std_logic := ‘U’; Y : out std_logic) ;
       end component;
       component NAND3
            port(A, B, C : in std_logic := ‘U’; Y : out std_logic) ;
       end component;
--    software bug: ANDTree_Data_2_net is not used
--       signal Temp_0_net, Temp_1_net, ANDTree_Data_2_net : std_logic ;
       signal Temp_0_net, Temp_1_net : std_logic ;
       begin
       AND3FTT_Temp_0_inst : AND3FTT
          port map(A => DataA(0), B => DataA(1), C => DataA(2), Y =>
            Temp_0_net);
       NAND3_AEB : NAND3
          port map(A => Temp_0_net, B => Temp_1_net, C => ‘1’, Y => AEB);
       AND3FTT_Temp_1_inst : AND3FTT
          port map(A => DataA(5), B => DataA(4), C => DataA(3), Y =>
            Temp_1_net);
end DEF_ARCH;
```

Entity i_reg_4ph_comp

```
-- This module works well before August
---- Updated on Sep 2 2004
```

```
-- Q1gate, Q1source, Q2gate, Q2source → Q1 and Q2
-- Only modin is used, no IO_clock any more
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity i_reg_4ph_comp is
    Port ( modin : in std_logic;
            comin : in std_logic_vector(3 downto 0);
            sense_low : in std_logic_vector(3 downto 0);
            I_chop : in std_logic_vector(3 downto 0);
            over_I : in std_logic_vector(3 downto 0);
            didtcomp : in std_logic_vector(3 downto 0);
            enable : in std_logic;
            MorG : in std_logic;
            Q1: out std_logic_vector(3 downto 0);
            Q2: out std_logic_vector(3 downto 0);
            over_I_out : out std_logic;
            do_sense_out : out std_logic_vector(3 downto 0);
            do_sense_high : out std_logic_vector(3 downto 0);
            didtout : out std_logic_vector(3 downto 0));
end i_reg_4ph_comp;
architecture Behavioral of i_reg_4ph_comp is
component ph_ckt_new is
    Port ( modin :        in std_logic;    --10KHz    omm.
             comin :        in std_logic;    --Gengerating Torque
            sense_low :    in std_logic; --Current is zero
            I_chop :           in std_logic;
            over_I :           in std_logic;
            I_off :               in std_logic;
            enable :           in std_logic;
            MorG :             in std_logic;
            didtcomp:        in std_logic;    -- new input
            Q1gate,Q2gate: out std_logic;
            over_I_out :        out std_logic;
            do_sense_low : out std_logic;    -- hanged from do_sense_out
            do_sense_high, didtout: out std_logic    -- new outputs
            );
end component ph_ckt_new;
signal over_I_out_tmp: std_logic_vector(3 downto 0);
signal I_off: std_logic;
begin
I_off<= over_I_out_tmp(0) or over_I_out_tmp(1) or over_I_out_tmp(2) or over_I_out_tmp(3);
over_I_out<= I_off;
U0: ph_ckt_new
port map (modin=>modin, comin=>comin(0), sense_low=>sense_low(0),
           I_chop=>I_chop(0), over_I=>over_I(0), I_off=>I_off, enable=>enable, MorG=>MorG,
               didtcomp=>didtcomp(0),Q1gate=>Q1(0), Q2gate=>Q2(0),
               over_I_out=>over_I_out_tmp(0), do_sense_low=>do_sense_out(0),
               do_sense_high=>do_sense_high(0), didtout=>didtout(0));
U1: ph_ckt_new
port map (modin=>modin, comin=>comin(1), sense_low=>sense_low(1),
           I_chop=>I_chop(1), over_I=>over_I(1), I_off=>I_off, enable=>enable, MorG=>MorG,
               didtcomp=>didtcomp(1), Q1gate=>Q1(1), Q2gate=>Q2(1),
               over_I_out=>over_I_out_tmp(1), do_sense_low=>do_sense_out(1),
               do_sense_high=>do_sense_high(1), didtout=>didtout(1));
U2: ph_ckt_new
port map (modin=>modin, comin=>comin(2), sense_low=>sense_low(2),
           I_chop=>I_chop(2), over_I=>over_I(2), I_off=>I_off, enable=>enable, MorG=>MorG,
                    didtcomp=>didtcomp(2), Q1gate=>Q1(2), Q2gate=>Q2(2),
                    over_I_out=>over_I_out_tmp(2), do_sense_low=>do_sense_out(2),
                    do_sense_high=>do_sense_high(2), didtout=>didtout(2));
U3: ph_ckt_new
port map (modin=>modin, comin=>comin(3), sense_low=>sense_low(3),
                I_chop=>I_chop(3), over_I=>over_I(3), I_off=>I_off, enable=>enable, MorG=>MorG,
                    didtcomp=>didtcomp(3), Q1gate=>Q1(3), Q2gate=>Q2(3),
                    over_I_out=>over_I_out_tmp(3), do_sense_low=>do_sense_out(3),
                    do_sense_high=>do_sense_high(3), didtout=>didtout(3));
end Behavioral;

Entity ph_ckt_new
-- Built on Feb. 16, 2004
-- included didt circuit in the logic subsystem in the MATLAB model
-- signal MorG and Enable are set as regular Ios
--
-- Updated on Sep 2 2004
-- Q1gate, Q1source, Q2gate, Q2source → Q1 and Q2
-- only modin is used, no IO_clock any more
--
-- Updated on Dec. 17 2004
-- Inverted Q1 and Q2 in this module for level shifting
--
-- Updated on Jan 31 2005
-- Save fault SRFF is deleted to avoid the noice from over_I input
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity ph_ckt_new is
    Port ( modin :         in std_logic;    --10KHz    omm.
                comin :         in std_logic;    --Gengerating Torque
               sense_low :    in std_logic; --Current is zero
               I_chop :              in std_logic;
               over_I :              in std_logic;
               I_off :                   in std_logic;
               enable :              in std_logic;
               MorG :                in std_logic;    -- used to avoid modulation at high speeds
               didtcomp:         in std_logic;    -- new input
               Q1gate,Q2gate:          out std_logic;
               over_I_out :          out std_logic;
               do_sense_low : out std_logic;    -- hanged from do_sense_out
               do_sense_high, didtout: out std_logic    -- new outputs
               );
end entity ph_ckt_new;
architecture Behavioral of ph_ckt_new is
signal and_1_output, and_2_output, and_3_output, and_4_output: std_logic;
signal and_5_output, and_6_output: std_logic;       -- for the do_sense_high signal
signal or_1_output,    or_2_output,    or_3_output: std_logic;
signal comin_inv, sense_low_inv, I_chop_inv, I_off_inv, enable_inv: std_logic;
signal sense_low_inv_hold: std_logic;
signal Q1_temp,Q1,Q1_inv:std_logic;
signal QT_temp, Q_I_chop, Q_I_chop_inv: std_logic;
signal Q3_temp,Q3: std_logic;    -- for do_sense_high signal
signal D1_temp,D1_inv_temp,D1_temp_inv,D2_temp,D2_inv_temp:std_logic;
signal Q1on, Q2on, IO_clock_inv: std_logic;
signal sr1: std_logic_vector(2 downto 0);
signal sr2: std_logic_vector(2 downto 0);
signal sr3: std_logic_vector(2 downto 0); -- for do_sense_high signal
signal pre_state,next_state: std_logic;
signal pre_state_3,next_state_3: std_logic;    -- for do_sense_high RSFF

begin
sr1<=and_2_output & comin & pre_state; --combinational logic operation.
Sr3<=comin_inv & and_5_output & pre_state_3;      -- for do_sense_high
------------- do inverse----------------------
comin_inv<=not comin;
sense_low_inv<=not sense_low;
I_chop_inv<=not I_chop;
I_off_inv<=not I_off;
enable_inv<=not enable;
------------- describe “and” gates-------------
and_1_output<=modin and comin_inv and Q1 and MorG; -- added MorG to avoid modulation at high speeds
and_2_output<=sense_low_inv_hold and comin_inv;
and_3_output<=Q_I_chop; -- MorG and Q_I_chop;
and_4_output<=Q_I_chop_inv; --MorG and Q_I_chop_inv;
and_5_output<=I_chop and comin;      --- for do_sense_high signal
and_6_output<=Q3 and comin;      --- for do_sense_high signal
do_sense_high<= and_6_output;      --- for do_sense_high signal
Q1on<=or_1_output and or_2_output and enable and I_off_inv;
Q2on<=or_1_output and or_3_output and enable and I_off_inv;
------------- describe “or” gates--------------
or_1_output<=and_1_output or comin;
or_2_output<=and_3_output or I_chop_inv;
or_3_output<=and_4_output or I_chop_inv;
-------additional part------------
Q1gate<= not Q1on;
Q2gate<= not Q2on;
```

- Added on Jan. 31 2005 to avoid the noise of the over_I input

```
over_I_out<= over_I;
------------- describe Flip Flops--------------
srff1_comb: process (sr1) is
begin
case sr1 is
when”000” => Q1_temp<=’0’; next_state<=’0’;
when”001” => Q1_temp<=’1’; next_state<=’1’;
when”010” => Q1_temp<=’0’; next_state<=’0’;
when”011” => Q1_temp<=’0’; next_state<=’0’;
when”100” => Q1_temp<=’1’; next_state<=’1’;
when”101” => Q1_temp<=’1’; next_state<=’1’;
when”110” => Q1_temp<=’0’; next_state<=’0’;
when”111” => Q1_temp<=’0’; next_state<=’0’;
when others=> Q1_temp<=’1’; next_state<=’0’;
end case;
Q1<=Q1_temp;
do_sense_low<=Q1;
end process srff1_comb;
srff1_dff: process(modin) is
begin
if (rising_edge(modin)) then
pre_state<=next_state;
end if;
end process srff1_dff
--------------- T Flip Flop---------------------
TFF: process (I_chop_inv, enable) is
begin
    if (enable = ‘0’) then
        QT_temp <=’0’;
    else
    if (rising_edge(I_chop_inv)) then
        QT_temp <= not QT_temp;
          end if;
      end if;
        Q_I_chop<=QT_temp;
        Q_I_chop_inv<= not QT_temp;
end process TFF;
-------------- save fault S R Flip Flop--------
-- srff2_comb: process (sr2) is
-- begin
-- case sr2 is
-- when”000” => Q2_temp<=’0’; next_state_2<=’0’;
-- when”001” => Q2_temp<=’1’; next_state_2<=’1’;
-- when”010” => Q2_temp<=’0’; next_state_2<=’0’;
-- when”011” => Q2_temp<=’0’; next_state_2<=’0’;
-- when”100” => Q2_temp<=’1’; next_state_2<=’1’;
-- when”101” => Q2_temp<=’1’; next_state_2<=’1’;
-- when”110” => Q2_temp<=’0’; next_state_2<=’0’;
-- when”111” => Q2_temp<=’0’; next_state_2<=’0’;
-- when others=> Q2_temp<=’0’; next_state_2<=’0’;
-- end case;
-- Q2<=Q2_temp;
-- over_I_out<=Q2;
-- end process srff2_comb;
-- srff2_dff: process(modin) is
-- begin
-- if (rising_edge(modin)) then
-- pre_state_2<=next_state_2;
-- end if;
-- end process srff2_dff;
--------------- zero order holder and delay realized with 2 D flip flops------
Holder: process (modin) is
begin
if (rising_edge(modin)) then
D1_temp<=sense_low_inv;
end if;
end process Holder;
Delay: process (modin) is
begin
if (rising_edge(modin)) then
D2_temp<=D1_temp;
end if;
sense_low_inv_hold<= D2_temp;
end process Delay;
----------------------RSFF for do_sense_high signal-------
srff3_comb: process (sr3) is
begin
case sr3 is
when”000” => Q3_temp<=’0’; next_state_3<=’0’;
when”001” => Q3_temp<=’1’; next_state_3<=’1’;
when”010” => Q3_temp<=’0’; next_state_3<=’0’;
when”011” => Q3_temp<=’0’; next_state_3<=’0’;
when”100” => Q3_temp<=’1’; next_state_3<=’1’;
when”101” => Q3_temp<=’1’; next_state_3<=’1’;
when”110” => Q3_temp<=’0’; next_state_3<=’0’;
when”111” => Q3_temp<=’0’; next_state_3<=’0’;
when others=> Q3_temp<=’0’; next_state_3<=’0’;
end case;
Q3<=Q3_temp;
```

```
end process srff3_comb;
srff3_dff: process(modin) is
begin
if (rising_edge(modin)) then
pre_state_3<=next_state_3;
end if;
end process srff3_dff;
-------------process for didtout signal--------------
didtoutprocess: process (comin, didtcomp)
begin
if(comin=’1’) then
     didtout<=didtcomp;
else
     didtout<=’0’;
end if;
end process didtoutprocess;
end architecture Behavioral;

Entity interfact_fpga_dsp_3
-- The McBSP1 are used as GPIO, Timer1 is used as GPIO too.
-- A potential problem is that the clock signal frequency should be higher than the
-- frequency of the output signals
-- It works well according to simulation
-- It worked will before August 2004
--
-- Updated on Sep 2 2004
-- IorV(1 downto 0) added
--
-- Updated on Feb 4 2005
-- for the decoder, 3 select lines are set as FSR, CLKX, FSX
-- only one input is set CLKR so that only one output is assigned for one time
--
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
entity interface_fpga_dsp_3 is
       Port ( A : in std_logic;
                    B : in std_logic;
                    I : in std_logic;    -- because I appear once per resolution, not useful for position
                    Enable : out std_logic;
                     MorG: out std_logic;
                       omm. : out std_logic_vector(3 downto 0);
                    IorV : out std_logic_vector(1 downto 0);
                    over_I : in std_logic;
                    I_sense : in std_logic_vector(3 downto 0);
                     clock: in std_logic;      -- to dffs for holding output signals
                    Timer1 : in std_logic; -- select line 1 of mux
                    DX : in std_logic;          -- select line 0 of mux
                    DR : out std_logic;        -- output line 1 of mux
                    CLKS : out std_logic;    -- output line 0 of mux
                    CLKR : in std_logic;--    input line 1 of decoder
                    FSR: in std_logic;    --    input line 0 of decoder
                    CLKX : in std_logic;-- select line 1 of decoder
                    FSX : in std_logic);-- select line 0 of decoder
end interface_fpga_dsp_3;
architecture Behavioral of interface_fpga_dsp_3 is
signal A_tmp, B_tmp,I_tmp, reset_tmp, over_I_tmp, clock_tmp: std_logic;
signal I_sense_tmp: std_logic_vector(3 downto 0);
signal Timer1_tmp, DX_tmp, CLKR_tmp, FSR_tmp, CLKX_tmp, FSX_tmp: std_logic;
signal Enable_tmp, MorG_tmp,DR_tmp, CLKS_tmp: std_logic;
signal    omm._tmp: std_logic_vector(3 downto 0);
signal IorV_tmp: std_logic_vector(1 downto 0);
-- signal dir: std_logic; -- output of the counter, direction of the rotation
-- signal counter_out: std_logic_vector(1 downto 0); -- outputs of the counter
signal muxsel: std_logic_vector(1 downto 0);
signal decodersel: std_logic_vector(2 downto 0);
begin
-- PAD instantiation // will be realized in Actel Libero software
-- input PADs
  Timer1_tmp         <=   Timer1;
  DX_tmp             <=   DX;
  A_tmp              <=   A;
  B_tmp              <=   B;
  I_tmp              <=   I;
  -- reset_tmp       <=   reset;
  over_I_tmp         <=   over_I;
  clock_tmp          <= clock;
  I_sense_tmp(0)     <=   I_sense(0);
  I_sense_tmp(1)     <=   I_sense(1);
  I_sense_tmp(2)     <=   I_sense(2);
  I_sense_tmp(3)     <=   I_sense(3);
  CLKR_tmp    <= CLKR;
  FSR_tmp     <= FSR;
  CLKX_tmp    <= CLKX;
  FSX_tmp     <= FSX;
      ■  output PADs
Enable <= Enable_tmp;
  MorG <= MorG_tmp;
  DR <= DR_tmp;
  CLKS <= CLKS_tmp;
     omm. <=    omm._tmp;
  IorV <= IorV_tmp;
      ■  select lines for the mux and decoder
muxsel<= Timer1_tmp & DX_tmp;
decodersel<= FSR_tmp & CLKX_tmp & FSX_tmp;
mux8x2: process (muxsel, I_sense_tmp, over_I_tmp,A_tmp,B_tmp, I_tmp) is
begin
case muxsel is
when “11” => CLKS_tmp <= I_sense_tmp(3); DR_tmp <= I_sense_tmp(2);
when “10” => CLKS_tmp <= I_sense_tmp(1); DR_tmp <= I_sense_tmp(0);
when “01” => CLKS_tmp <= over_I_tmp;          DR_tmp <= I_tmp;
when “00” => CLKS_tmp <= A_tmp; DR_tmp <= B_tmp;
when others => CLKS_tmp <= A_tmp; DR_tmp <= B_tmp;
end case;
end process;
decoder: process (clock_tmp) is
begin
if (rising_edge(clock_tmp)) then
case decodersel is
when “111” => IorV_tmp(1) <= CLKR_tmp;
when “110” => IorV_tmp(0) <= CLKR_tmp;
when “101” => Enable_tmp    <= CLKR_tmp;
when “100” => MorG_tmp      <= CLKR_tmp;
when “011” =>    omm._tmp(3) <= CLKR_tmp;
when “010” =>    omm._tmp(2) <= CLKR_tmp;
when “001” =>    omm._tmp(1) <= CLKR_tmp;
when “000” =>    omm._tmp(0) <= CLKR_tmp;
when others => MorG_tmp<= CLKR_tmp;
end case;
end if;
end process;

end Behavioral;
```

Appendix II. C/C++ code in DSP

Only the C files programmed by the author are presented here. The header files and the files generated by the code composer studio are not presented here.

Main.c

```
/*****************************************************************/
/* final program for the THS1206 EVM connected to an C6711 DSK.          */
/*                                                                       */
/* The following jumper setting should be used:                          */
/* Daughter Card Style THS1206 EVM:                                      */
/*      J1   1-2      /  J2   1-2    /  J3   2-5   /  J4   open          */
/*      J5   open     /  J6   open   /  J7   1-2   /  J10 closed         */
/*      J11 open      /  J12 2-3     /  J13 1-2                          */
/* Supply voltage from DSP, CLK from Timer 0, Input AINP                 */
/* AD converter address: 0xA0020000                                      */
/*                                                                       */
/* The following jumper setting should be used:                          */
/* Modular THS1206 EVM:                                                  */
/*      W1, W2, W3, W9, W10 – Closed                                     */
/*      W11 – Open                                                       */
/*      W5, W6 – 1-2                                                            */
/* Supply voltage from DSP, CLK from Timer 0, Input AINP                 */
/* AD converter address: 0xA0024000                                      */
/* DSP/BIOS II and CSL used                                              */
/*                                                                       */
/* This program runs sensorless control for SRM                          */
/* Copy right 2005 Jinhui Zhang, Arthur Radun,                           */
/* Power Electronics Lab, University of Kentucky                         */
/*****************************************************************/

/* include files for data converter support */
#include “dc_conf.h”
#include “t1206_fn.h”
#include “mcbsp.h”
#include “Rockymotin.h”
#include “lookuptable.h”
#include “variables.h”
/* include files for DSP/BIOS                    */
#include <std.h>
#include <swi.h>
#include <log.h>

/* include files for chip support library      */
#include <csl.h>
#include <csl_legacy.h>
#include <csl_irq.h>
#include <csl_timer.h>

#define phase 4 /* size of data buffer */
#define dataSaveSize 1000    // size of data saved
#define savestep 10 /* data save step based on the sampling time */

/* function prototypes */
TIMER_HANDLE init_timer0(unsigned int period);
void init_dsk(void);
void wait(void);
void GetSignals_McBSP1(void);
```

```
float errorlow(int);
float errorhigh(void);
void SenseTheta(void);
void Commutator(int);
void MicroController(void);
float flux(float theta, float phi);
void motor_init(void);
void storeData(void);
void sendOutComm(void);
void DoCalculationFunc(void);
void calculatePhaseIV(void);

/* DSP/BIOS objects, created with the Config Tool                    */
extern LOG_Obj trace;
extern far SWI_Obj SwiStartConversion;
// extern far SWI_Obj SwiDoCalculation;

int posR=0;
short gtheta_short[phase];
float gtheta[phase];
short phaseI_short[phase], phaseLamda_short[phase];
float phaseI[phase], phaseLamda[phase], phaseLamdah[phase];
int I_sense[phase];
int    omm.[phase],   omm._act[phase];
int    omm.[cDataSize],commHighA[cDataSize];
int    omm[cDataSize],commHighB[cDataSize];
int commC[cDataSize],commHighC[cDataSize];
int commD[cDataSize],commHighD[cDataSize];
float galpha[phase];
float thetah,omegah,error;
int thetahInt;
int observer, startRecord, sensorless, highSpeed;
int Enable, MorG, IorV0, IorV1, Enable_act,MorG_act,IorV0_act,IorV1_act;
float time;
int matrixCnt, savei;
float gthetaMatrix0[dataSaveSize];
float gthetaMatrix1[dataSaveSize];
float gthetaMatrix2[dataSaveSize];
float gthetaMatrix3[dataSaveSize];

float galphaMatrix0[dataSaveSize];
float galphaMatrix1[dataSaveSize];
float galphaMatrix2[dataSaveSize];
float galphaMatrix3[dataSaveSize];

int IsenseMatrix0[dataSaveSize];
int IsenseMatrix1[dataSaveSize];
int IsenseMatrix2[dataSaveSize];
int IsenseMatrix3[dataSaveSize];

int commMatrix0[dataSaveSize];
int commMatrix1[dataSaveSize];
int commMatrix2[dataSaveSize];
int commMatrix3[dataSaveSize];

float phaseLamdaMatrix0[dataSaveSize];
float phaseLamdaMatrix1[dataSaveSize];
float phaseLamdaMatrix2[dataSaveSize];
float phaseLamdaMatrix3[dataSaveSize];
float phaseLamdahMatrix0[dataSaveSize];
float phaseLamdahMatrix1[dataSaveSize];
float phaseLamdahMatrix2[dataSaveSize];
float phaseLamdahMatrix3[dataSaveSize];

float phaseIMatrix0[dataSaveSize];
float phaseIMatrix1[dataSaveSize];
float phaseIMatrix2[dataSaveSize];
float phaseIMatrix3[dataSaveSize];

int posRMatrix[dataSaveSize];
float posEMatrix[dataSaveSize]; // angle error
float thetahMatrix[dataSaveSize];
float omegahMatrix[dataSaveSize];
float errorMatrix[dataSaveSize];
float timeMatrix[dataSaveSize];
float runtimeMatrix[dataSaveSize];

LgUns time1,time2,time3,time4;
float runtime,runtime2,runtime3;

void main(void)
{
    TIMER_HANDLE hTimer;

    /* CSL_Init – required for the CSL functions of the driver    */
    CSL_Init();

    /* initialize the DSK and timer 0           */
    init_dsk();
    hTimer = init_timer0(ADC1_TIM_PERIOD);
    init_McBSP1();
    init_timer1();

    /* configure the data converter             */
    dc_configure(&Ths1206_1);

    motor_init();

    /* start the timer                          */
    TIMER_Start(hTimer);

    /* Let’s go... DSP/BIOS takes control and will generate         */
    /* a “PeriodFunc” software interrupt every second.              */
}

/*****************************************************************/
/* BlockReady                                                    */
/* This function will be called when the dc_rblock routine is    */
/* finished. It posts a DoCalculation software interrupt.        */
/*****************************************************************/
void BlockReady1206(void *pDC)
{
//      LOG_printf(&trace, “1206 Interrupt”);
//      SWI_post(&SwiDoCalculation);
DoCalculationFunc();
}
void BlockReady1206_V(void *pDC)
{
    calculatePhaseIV();
```

```
}
void BlockReady1206_I(void *pDC)
{
      if(IorV1_act!=0)
             {set_IorV1(0); wait(); IorV1_act=0;}
       if(IorV0_act!=1)
             {set_IorV0(1); wait(); IorV0_act=1;}
      dc_rblock(&Ths1206_1, phaseLamda_short, phase, BlockReady1206_V);
}

/****************************************************************/
/* SwiStartConversionFunc                                          */
/* This software    omm.   oni starts a new conversion using the          */
/* dc_rblock function.                                             */
/****************************************************************/
void StartConversionFunc()
{
       time1=CLK_gethtime();
//       dc_rblock(&Ths1206_1, gtheta_short, phase, &BlockReady1206);
       // Get current singals
       if (observer==0)    // inductance profile based observer is chosen
       {
       /****************************************************************/
       /* Inductance Profile Based Observer                               */
       /* Only gtheta is measured                                         */
       /****************************************************************/
       if(IorV1_act!=0)
              {set_IorV1(0); wait(); IorV1_act=0;}
       if(IorV0_act!=0)
              {set_IorV0(0); wait(); IorV0_act=0;}
      dc_rblock(&Ths1206_1, gtheta_short, phase, &BlockReady1206);
       }
      else if (observer==1)    // torque producing current based observer is chosen
       {
       /****************************************************************/
       /* Torque producing current Based Observer                         */
       /* Now phase current is measured and then phase voltage            */
       /****************************************************************/
       if(IorV1_act!=1)
              {set_IorV1(1); wait(); IorV1_act=1;}
       if(IorV0_act!=0)
              {set_IorV0(0); wait(); IorV0_act=0;}
      dc_rblock(&Ths1206_1, phaseI_short, phase,BlockReady1206_I);
       }
}

void DoCalculationFunc()
{
       int i,value[phase];
      for (i=0; i<phase; i++)
       {
             value[i] = gtheta_short[i] & 0x0FFF;
             gtheta[i] = 2.778-(6.78E-4)*value[i];
             // gtheta[i] = (Vref_plus-Vref_minus-(Vref_plus-Vref_minus)*value/4096)*5.0/3.6;
       }             // the signal conditioning board changed the analog signals' polarity
             // at the reference (Vref_plus+Vref_minus)/2
             // The offset is 1.5 V for the phase currents
             // times 5.0 to convert it into ampere
             // DC gain of the low pass filter is 3.6
       GetSignals_McBSP1();
```

```
        MicroController();
        sendOutComm();
        if (startRecord==1) storeData();
//      time2=CLK_gethtime();
//      runtime=(time2-time1)*4/150;
}

void calculatePhaseIV()
{
        int valueI[phase];
        int valueLamda[phase];
        valueI[0] = phaseI_short[0] & 0x0FFF;
        phaseI[0] = (2.0-2.0*valueI[0]*2.441E-4)*25.2;
        valueLamda[0] = phaseLamda_short[0] & 0x0FFF;
        phaseLamda[0] = (2-2*valueLamda[0]*2.441E-4-0.05)*0.01884*0.72;

        valueI[1] = phaseI_short[1] & 0x0FFF;
        phaseI[1] = (2.0-2.0*valueI[1]*2.441E-4)*25.2;
        valueLamda[1] = phaseLamda_short[1] & 0x0FFF;
        phaseLamda[1] = (2-2*valueLamda[1]*2.441E-4-0.018)*0.01884*0.8;

        valueI[2] = phaseI_short[2] & 0x0FFF;
        phaseI[2] = (2.0-2.0*valueI[2]*2.441E-4)*25.2;
        valueLamda[2] = phaseLamda_short[2] & 0x0FFF;
        phaseLamda[2] = (2-2*valueLamda[2]*2.441E-4-0.05)*0.01884*1.0;

        valueI[3] = phaseI_short[3] & 0x0FFF;
        phaseI[3] = (2.0-2.0*valueI[3]*2.441E-4)*25.2;
        valueLamda[3] = phaseLamda_short[3] & 0x0FFF;
        phaseLamda[3] = (2-2*valueLamda[3]*2.441E-4-0.045)*0.01884*0.8;

        GetSignals_McBSP1();
        MicroController();
        sendOutComm();
        if (startRecord==1) storeData();
        time2=CLK_gethtime();
       runtime=(time2-time1)*4/150;
}

/*************************************************************/
/* PeriodFunc                                                */
/* The function will be called every second by DSP/BIOS and  */
/* posts a StartConversion SWI to start a new conversion.    */
/*************************************************************/
void PeriodFunc()
{
        time+=tsamplem;
//      if (time>100) time=0;
       SWI_post(&SwiStartConversion);
}


void wait()
{
int i;
  for (i=0;i<5;i++) ;
}

/*****************************************************************************/
/* ChannelA()                                                                */
/* The function will to called every time with a pulse signal form A channel                    */
/*                                                                                               */
/*******************************************************************************/
void ChannelA(void)
{
// int pcr;
if (posR==5) startRecord=1;

/*
pcr=get_McBSP1_CLKS_DR();
if ((pcr & 0x00000010)==0x00000010) posR +=1;
        else if ((pcr & 0x00000010)==0x00000000) posR -=1;
                else LOG_printf(&trace, "read McBSP1 error");
*/
posR+=1;
if (posR==180)          posR =0;
//      else if (posR==-1) posR=179;

//////////////// To test the motor's symmetry //// 03-24-05

// THS1206Conversion();
// MicroController(Vpower,over_I,gtheta,I_sense,didtd,   omm.,thetah,&omegah,
//                              &Icomm, &Enable, &MorG,&error);
// storeData();

}

/*******************************************************************************/
/* ChannelI()                                                                    */
/* The function will to called every time with a pulse signal form I channel      */
/* To    omm.   onize the real rotor position
        */
/*******************************************************************************/
void ChannelI(void)
{

posR = 122;

}

/**************************************************************/
/* GetSignals_McBSP1                                          */
/* The function will be called by SwiGetSignals               */
/* It gets signals through McBSP1                             */
/**************************************************************/
void GetSignals_McBSP1(void)
{
        int pcr;
// Get circuit signals
        put_muxsel(1, 1);    // get I_sense[3] and I_sense[2]
        wait();
        pcr=get_McBSP1_CLKS_DR();
        if ((pcr & 0x00000040)==0x00000040) I_sense[3]=1;
                else if ((pcr & 0x00000040)==0x00000000) I_sense[3]=0;
                        else LOG_printf(&trace, "read McBSP1 error");
        if ((pcr & 0x00000010)==0x00000010) I_sense[2]=1;
                else if ((pcr & 0x00000010)==0x00000000) I_sense[2]=0;
                        else LOG_printf(&trace, "read McBSP1 error");

        put_muxsel(1, 0);    // get I_sense[1] and I_sense[0]
        wait();
        pcr=get_McBSP1_CLKS_DR();
        if ((pcr & 0x00000040)==0x00000040) I_sense[1]=1;
                else if ((pcr & 0x00000040)==0x00000000) I_sense[1]=0;
                        else LOG_printf(&trace, “read McBSP1 error”);
        if ((pcr & 0x00000010)==0x00000010) I_sense[0]=1;
                else if ((pcr & 0x00000010)==0x00000000) I_sense[0]=0;
                        else LOG_printf(&trace, “read McBSP1 error”);
}

void sendOutComm(void)
{

if ((Enable==1)    && (time>=0.2))
// if (Enable==1)
  {

        if (   omm.[3]!=   omm._act[3])
                {set_comm3(   omm.[3]); wait();    omm._act[3]=   omm.[3];}
        if (   omm.[2]!=   omm._act[2])
                {set_comm2(   omm.[2]); wait();    omm._act[2]=   omm.[2];}
        if (   omm.[1]!=   omm._act[1])
                {set_comm1(   omm.[1]); wait();    omm._act[1]=   omm.[1];}
        if (   omm.[0]!=   omm._act[0])
                {set_comm0(   omm.[0]); wait();    omm._act[0]=   omm.[0];}
  }

}

/*    MicroController Program        */

void MicroController(void)
{
SenseTheta();
Commutator(thetahInt);
/*
if (observer==0)
Commutator(thetahInt);       // using posR to do sensored control, thetahInt do sensorless
else if (sensorless==1)
Commutator(thetahInt);
else
Commutator(posR);
*/
}

void SenseTheta(void)
{
if (observer==0)
{
        omegah += error*H2*tsamplem;
        thetah += (omegah+error*H1)*tsamplem;
        while(thetah<0)
                thetah+=pi;
        while(thetah>pi)
                thetah-=pi;
        thetahInt=(int)(thetah*57.325);// 180/pi
        error=errorlow(thetahInt);       // calculate error after thetah to make sure it’s in rang(0 pi)

}
else
```

```
{
	omegah += error*H4*tsamplem;
	thetah += (omegah+error*H3)*tsamplem;
	while(thetah<0)
		thetah+=pi;
	while(thetah>pi)
		thetah-=pi;
	thetahInt=(int)(thetah*57.325);// 180/pi
	error=errorhigh();	// flux method

//	if(omegah>10) sensorless=1;			// high speed using sensorless control
//	else if (omegah<10) sensorless=0;	// low speed using sensored control

}

// added on for testing the response time
/*
if(time>5.0)
{
	startRecord=1;
	if (matrixCnt==100)
	thetah-=15*3.14/180;
}
*/

if(omegah>100) observer=1;
else if (omegah<50) observer=0;
if(omegah>100) highSpeed=1;
else if (omegah<80) highSpeed=0;
// if (omegah>10) startRecord=1;
}


float errorlow(int alphaInt)
{
float error;
  galpha[0]=gmatrix0[alphaInt];
  galpha[1]=gmatrix1[alphaInt];
  galpha[2]=gmatrix2[alphaInt];
  galpha[3]=gmatrix3[alphaInt];

// galpha[0]=gmatrix0[posR];
// galpha[1]=gmatrix1[posR];
// galpha[2]=gmatrix2[posR];
// galpha[3]=gmatrix3[posR];

if (I_sense[0]==0)		gtheta[0]=galpha[0];
if (I_sense[1]==0)		gtheta[1]=galpha[1];
if (I_sense[2]==0)		gtheta[2]=galpha[2];
if (I_sense[3]==0)		gtheta[3]=galpha[3];

error=galpha[0]*gtheta[1]-galpha[1]*gtheta[0]+
		galpha[1]*gtheta[2]-galpha[2]*gtheta[1]+
		galpha[2]*gtheta[3]-galpha[3]*gtheta[2]+
		galpha[3]*gtheta[0]-galpha[0]*gtheta[3];
// Filter out the high amplitude noise
// if (error>0.8) error=0.8;
// else if(error<-0.8) error=-0.8;
return (error);
}
```

```
float errorhigh(void)
{
int i;
float errorh;
float thetatemp[phase];
float temp=0;
// thetatemp[0]=posR*0.017444;      // actual angle
thetatemp[0]=thetah;                        // estimated angle
thetatemp[1]=thetatemp[0]+0.7854; // 45 degree phase shift
thetatemp[2]=thetatemp[1]+0.7854;
thetatemp[3]=thetatemp[2]+0.7854;
// thetatemp=thetah;                  // estimated angle
for (i=0;i<phase;i++)
{
       if((I_sense[i]==0)&&(   omm.[i]==1))
        {
              // wrap the angle into -30 ~ 30 period
              while(thetatemp[i] > 0.5236)
                     thetatemp[i] -= 1.0472;
              while(thetatemp[i] < -0.5236)
                     thetatemp[i] += 1.0472;
              phaseLamdah[i] = flux(thetatemp[i],phaseI[i]);
              if(thetatemp[i]>=0)
              temp +=   phaseLamdah[i]-phaseLamda[i];
              else
              temp += phaseLamda[i]-phaseLamdah[i];
       }
       else
       phaseLamdah[i] = 0.0;
}
errorh=temp;
return(errorh);
}

void Commutator(int alphaInt)
{
if (highSpeed==0)
{
  omm.[0]=   omm.[alphaInt];
  omm.[1]=   omm[alphaInt];
  omm.[2]=commC[alphaInt];
  omm.[3]=commD[alphaInt];
}
else
{
  omm.[0]=commHighA[alphaInt];
  omm.[1]=commHighB[alphaInt];
  omm.[2]=commHighC[alphaInt];
  omm.[3]=commHighD[alphaInt];
}
}

/***************************************************************************************/
/* motor_init()                                                                        */
/* The function will to called in the main function to initialize the motor           */
/* rotor position and initialize other variables                                       */
/***************************************************************************************/
void motor_init(void)
{
```

```
int i,j;                  // j is used in the look up table generation program
time=0;                       // the real time
omegah=0;                     // the estimated rotor speed
error=0;                  // the error for the estimator
thetah=qod*pi/180;  // The rotor position for Phase A in radian
observer=0;                   // 0→inductance profile based observer, 1→ flux control
startRecord=0;            // not start until the motor starts moving (channelA is called)
sensorless=1;             // 1=sensorless control, 0=sensored control
highSpeed=0;
matrixCnt=0;
savei=1;
// Initialize command signals
Enable=1;
MorG=1;
IorV1=0;
IorV0=0;
  omm.[3]=0;
  omm.[2]=0;
  omm.[1]=0;
  omm.[0]=0;

// Disable the current regulator

set_Enable(0);
wait();
Enable_act      =     0;

set_MorG(MorG);
wait();
MorG_act=     MorG;

set_IorV1(IorV1);
wait();
IorV1_act=     IorV1;

set_IorV0(IorV0);
wait();
IorV0_act=     IorV0;

set_comm3(   omm.[3]);
wait();
  omm._act[3] =        omm.[3];

set_comm2(   omm.[2]);
wait();
  omm._act[2] =        omm.[2];

set_comm1(   omm.[1]);
wait();
  omm._act[1] =        omm.[1];

set_comm0(   omm.[0]);
wait();
  omm._act[0] =        omm.[0];

// Enable the current regulator
set_Enable(Enable);
wait();
Enable_act      =     Enable;
```

```
// Give a current command, which is 0.619*4.5*5=14 Ampere
/*      DSS_spWrite(0x0000);    // cancelled to adjust the current command externally 04-04-05

        Enable=1;
        set_Enable(Enable);
        wait();
        Enable_act      =       1;
*/
// Initialize the rotor position to the aligned position with Phase A
/*
        set_comm0(1);
        wait();
           omm._act[0] =        1;

        waitlong();

        set_comm0(0);
        wait();
           omm._act[0] =        0;
*/
       posR=0;                                  // the actual rotor position

// generate a look up table for the commutator
for(i=0;i<cDataSize;i++)
{
j=i;
while(j<-30)
        j +=60;
while(j>30)
        j -=60;
if((j>=qon)&&(j<=qoff))
           omm.[i]=1;
else
           omm.[i]=0;
if((j>=qon2)||(j<=qoff2))
        commHighA[i]=1;
else
        commHighA[i]=0;

j=i+45;
while(j<-30)
        j +=60;
while(j>30)
        j -=60;
if((j>=qon)&&(j<=qoff))
           omm[i]=1;
else
           omm[i]=0;
if((j>=qon2)||(j<=qoff2))
        commHighB[i]=1;
else
        commHighB[i]=0;

j=i+90;
while(j<-30)
        j +=60;
while(j>30)
        j -=60;
if((j>=qon)&&(j<=qoff))
        commC[i]=1;
```

```
else
        commC[i]=0;
if((j>=qon2)||(j<=qoff2))
        commHighC[i]=1;
else
        commHighC[i]=0;

j=i+135;
while(j<-30)
        j +=60;
while(j>30)
        j -=60;
if((j>=qon)&&(j<=qoff))
        commD[i]=1;
else
        commD[i]=0;
if((j>=qon2)||(j<=qoff2))
        commHighD[i]=1;
else
        commHighD[i]=0;
}

for(i=0;i<dataSaveSize;i++)     // initialize data save matries
        {
        gthetaMatrix0[i]        =       0.0;
                gthetaMatrix1[i]        =       0.0;
                gthetaMatrix2[i]        =       0.0;
                gthetaMatrix3[i]        =       0.0;

        galphaMatrix0[i]        =       0.0;
                galphaMatrix1[i]        =       0.0;
                galphaMatrix2[i]        =       0.0;
                galphaMatrix3[i]        =       0.0;

                thetahMatrix[i]         =       0.0;
                omegahMatrix[i]                 =       0.0;
                errorMatrix[i]          =       0.0;
                posRMatrix[i]           =       0;
                posEMatrix[i]           =       0;

                commMatrix0[i]                  =       0.0;
                commMatrix1[i]                  =       0.0;
                commMatrix2[i]                  =       0.0;
                commMatrix3[i]                  =       0.0;

                IsenseMatrix0[i]        =       0;
                IsenseMatrix1[i]        =       0;
                IsenseMatrix2[i]        =       0;
                IsenseMatrix3[i]        =       0;

                phaseLamdaMatrix0[i]=   0.0;
                phaseLamdaMatrix1[i]=   0.0;
                phaseLamdaMatrix2[i]=   0.0;
                phaseLamdaMatrix3[i]=   0.0;

                phaseLamdahMatrix0[i]=  0.0;
                phaseLamdahMatrix1[i]=  0.0;
                phaseLamdahMatrix2[i]=  0.0;
                phaseLamdahMatrix3[i]=  0.0;
                phaseIMatrix0[i]      =     0;
                phaseIMatrix1[i]      =     0;
                phaseIMatrix2[i]      =     0;
                phaseIMatrix3[i]      =     0;

                runtimeMatrix[i]      =     0;
                timeMatrix[i]         =     0;
        }
// End of initialization of the rotor
}


/*****************************************************************************/
/* storeData()                                                              */
/* The function will store data every savestep*tsemplem seconds              */
/*****************************************************************************/

void storeData(void)
{
if (matrixCnt<dataSaveSize)
        if(savei>=savestep)
        {

                gthetaMatrix0[matrixCnt]=     gtheta[0];
                gthetaMatrix1[matrixCnt]=     gtheta[1];
                gthetaMatrix2[matrixCnt]=     gtheta[2];
                gthetaMatrix3[matrixCnt]=     gtheta[3];

                galphaMatrix0[matrixCnt]      =     galpha[0];
                galphaMatrix1[matrixCnt]      =     galpha[1];
                galphaMatrix2[matrixCnt]      =     galpha[2];
                galphaMatrix3[matrixCnt]      =     galpha[3];

                thetahMatrix[matrixCnt]       =     thetah;
                omegahMatrix[matrixCnt]             =     omegah;
                errorMatrix[matrixCnt]        =     error;
                posRMatrix[matrixCnt]         =     posR;
                posEMatrix[matrixCnt]         =     thetah*57.325-posR;

                commMatrix0[matrixCnt]        =       omm.[0];
                commMatrix1[matrixCnt]        =       omm.[1];
                commMatrix2[matrixCnt]        =       omm.[2];
                commMatrix3[matrixCnt]        =       omm.[3];

                IsenseMatrix0[matrixCnt]=     I_sense[0];
                IsenseMatrix1[matrixCnt]=     I_sense[1];
                IsenseMatrix2[matrixCnt]=     I_sense[2];
                IsenseMatrix3[matrixCnt]=     I_sense[3];

                phaseLamdaMatrix0[matrixCnt]=   phaseLamda[0];
                phaseLamdaMatrix1[matrixCnt]=   phaseLamda[1];
                phaseLamdaMatrix2[matrixCnt]=   phaseLamda[2];
                phaseLamdaMatrix3[matrixCnt]=   phaseLamda[3];

                phaseLamdahMatrix0[matrixCnt]=  phaseLamdah[0];
                phaseLamdahMatrix1[matrixCnt]=  phaseLamdah[1];
                phaseLamdahMatrix2[matrixCnt]=  phaseLamdah[2];
                phaseLamdahMatrix3[matrixCnt]=  phaseLamdah[3];

                phaseIMatrix0[matrixCnt]      =     phaseI[0];
                phaseIMatrix1[matrixCnt]        =       phaseI[1];
                phaseIMatrix2[matrixCnt]        =       phaseI[2];
                phaseIMatrix3[matrixCnt]        =       phaseI[3];

                runtimeMatrix[matrixCnt]        =       runtime;
                timeMatrix[matrixCnt]           =       time1;

                matrixCnt += 1;
                savei= 1;
        }
        else
                savei += 1;
// single sequence data
/*
else
        {
                matrixCnt=0;
        }
*/
}

float flux(float theta, float phi)
{
float theta_abs, phi_abs,thetatemp;
float Lno, gf, Isatf, lamdam, lamdaf;

theta_abs=fabs(theta);
thetatemp=thetapf-theta_abs;
phi_abs=fabs(phi);

        if (theta_abs>thetapf)
        {
                Lno=LuL+(thetau-theta_abs)*4.1583E-4;
//              Lno=LuL+((Lpo-LuL)/(thetau-thetapf))*(thetau-theta_abs);
                return (Lno*phi_abs);
        }
        else
        {
//              gf= geff+go*(1-Rg*(thetapf-theta_abs)/pwf);
                gf=0.0012-0.0023*thetatemp;
//              Isatf= Bsat*(lfe+2*gf*u/uo)/(u*N) ;
                Isatf= 2.3945+gf*9.5493E4 ;
                if(phi_abs<Isatm)
//                      lamdam=lstk*STF*Rg*(thetapf-theta_abs)*u*N*N*phi_abs/(lfe+2*g*u/uo) ;
                lamdam=0.0026*thetatemp*phi_abs;
        else
//
lamdam=N*lstk*STF*Rg*(thetapf-theta_abs)*(uo*N*phi_abs+lfe*Bsat+uo*lfe*Hsat)/(lfe);
                        lamdam=thetatemp*(3.7699E-5*phi_abs+0.4514)*0.1406;
                if(phi_abs<Isatf)
//                      lamdaf=lstk*STF*Rg*theta_abs*(u*N*N*phi_abs/(lfe+2*gf*u/uo)) ;
                        lamdaf=0.0012*theta_abs*(5.6549*phi_abs/(lfe+10000*gf)) ;
        else
//                      lamdaf=N*lstk*STF*Rg*theta_abs*(uo*N*phi_abs+lfe*Bsat+uo*lfe*Hsat)/(lfe+2*gf)
;
                        lamdaf=0.0353*theta_abs*(3.7699E-5*phi_abs+0.4514)/(lfe+2*gf);
        return (lamdaf+lamdam);
        }
}
```

Mcbsp_functions.c

```
  #include <c6x.h>
  #include “c6x11dsk.h”
  #include <csl.h>
  #include <csl_legacy.h>
  #include <csl_timer.h>
  #include “mcbsp.h”

#define SP1_SRGR_V                 0x00000000
#define SP1_SPCR_V                 0x00000000
#define SP1_PCR_V                  0x00003f0c   // DX,CLKR,FSR,CLKX,FSX are outputs
                                                 // CLKS, DR are inputs
                                                 // use with FPGA chip on the current
                                                 // regulator board.
                                                 // initialize enable =0

#define SP1_PCR_V                  0x00003000    // use to test optical encoder signals
                                                 // A->FSR, B->CLKX, I->CLKR
                                                 // A_fpga-> CLKS,    B_fpga->DR;


#define SP1_PCR_V                  0x00003f0b   // use with FPGA chip, <CLKS DR>=<c1, c0>,
                                               //                    <Enable, MorG>=<clkx,fsx>

// PCR description
//    31   30   29   28   27   26   25   24   23   22   21   20   19   18   17   16   Reserved
//    _______________|_______________|_______________|_______________|
//    |15 | 14 |  13 |  12 |  11 |  10 |    9 |    8 |    7 | 6 |    5 |   4 |    3 |    2 |    1 |    0 |
//    |0   |0    | 1    | 1    | 1    | 1    |1 |    1 |   0 |      0 |   0 |   0 |    0 | 0 |    0 |    0 |
//    reserved    RIOEN     FSRM      CLKRM    CLKSSTAT DRSTAT     FSRP     CLKRP
//                XIOEN     FSXM      CLKXM       res       DXSTAT      FSXP       CLKXP
// when McBSP is used as GPIO, XIOEN and RIOEN should be ‘1’ both,
// and XRST and RRST in SPCR register @ the 16th bit and the 0th bit
// should be ‘0’ both.
*/
/*************************************************************/
/* init_GPIO                                                  */
/* This initializes the McBSP                                 */
/*************************************************************/

void init_McBSP1(void)
{
      * (UINT32 *) McBSP1_SRGR = (UINT32) SP1_SRGR_V;
      * (UINT32 *) McBSP1_SPCR = (UINT32) SP1_SPCR_V;
      * (UINT32 *) McBSP1_PCR    = (UINT32) SP1_PCR_V;
      return;
}

void init_timer1(void)
{
      TIMER_setDatOut(_TIMER_hDev1, 0);
      return;
}
int get_McBSP0(void)
{
      int pcr;
      pcr = (int) ( * (UINT32 *) McBSP0_PCR );
      return (pcr);
}
```

```
// McBSP 0 functions
int get_McBSP0_CLKX(void)
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP0_PCR );
        pcr = pcr >> 1;
        pcr = pcr & 0x00000001;
        return (pcr);
}
int get_McBSP0_FSX(void)
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP0_PCR );
        pcr = pcr >> 3;
        pcr = pcr & 0x00000001;
        return (pcr);
}
int get_McBSP0_CLKR(void)
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP0_PCR );
        pcr = pcr >> 0;
        pcr = pcr & 0x00000001;
        return (pcr);
}
int get_McBSP0_FSR(void)
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP0_PCR );
        pcr = pcr >> 2;
        pcr = pcr & 0x00000001;
        return (pcr);
}
int get_McBSP0_DR(void)
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP0_PCR );
        pcr = pcr >> 4;
        pcr = pcr & 0x00000001;
        return (pcr);
}
int get_McBSP0_CLKS(void)
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP0_PCR );
        pcr = pcr >> 6;
        pcr = pcr & 0x00000001;
        return (pcr);
}
void put_McBSP0_CLKX(int clkxp)
{
        if(clkxp==0)
                        * (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) & 0xfffffffd;
        else
                        •    (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) | 0x00000002;
}
void put_McBSP0_FSX(int fsxp)
{
        if(fsxp==0)
```

```
            * (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) & 0xfffffff7;
    else
            •   (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) | 0x00000008;
}
void put_McBSP0_DX(int dx_stat)
{
    if(dx_stat==0)
            * (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) & 0xffffffdf;
    else
            •   (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) | 0x00000020;
}
void put_McBSP0_CLKR(int clkrp)
{
    if(clkrp==0)
            * (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) & 0xfffffffe;
    else
            •   (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) | 0x00000001;
}
void put_McBSP0_FSR(int fsrp)
{
    if(fsrp==0)
            * (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) & 0xfffffffb;
    else
            •   (UINT32 *) McBSP0_PCR    = (* (UINT32 *) McBSP0_PCR) | 0x00000004;
}

// McBSP 1 functions
int get_McBSP1_CLKX(void)
{
    int pcr;
    pcr = (int) ( * (UINT32 *) McBSP1_PCR );
    pcr = pcr >> 1;
    pcr = pcr & 0x00000001;
    return (pcr);
}
int get_McBSP1_FSX(void)
{
    int pcr;
    pcr = (int) ( * (UINT32 *) McBSP1_PCR );
    pcr = pcr >> 3;
    pcr = pcr & 0x00000001;
    return (pcr);
}
int get_McBSP1_CLKR(void)
{
    int pcr;
    pcr = (int) ( * (UINT32 *) McBSP1_PCR );
    pcr = pcr >> 0;
    pcr = pcr & 0x00000001;
    return (pcr);
}
int get_McBSP1_FSR(void)
{
    int pcr;
    pcr = (int) ( * (UINT32 *) McBSP1_PCR );
    pcr = pcr >> 2;
    pcr = pcr & 0x00000001;
    return (pcr);
}
int get_McBSP1_DR(void)
```

```
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP1_PCR );
        pcr = pcr >> 4;
        pcr = pcr & 0x00000001;
        return (pcr);
}
int get_McBSP1_CLKS(void)
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP1_PCR );
        pcr = pcr >> 6;
        pcr = pcr & 0x00000001;
        return (pcr);
}
void put_McBSP1_CLKX(int clkxp)
{
        if(clkxp==0)
                        * (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) & 0xfffffffd;
        else
                        •    (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) | 0x00000002;
}
void put_McBSP1_FSX(int fsxp)
{
        if(fsxp==0)
                        * (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) & 0xfffffff7;
        else
                        •    (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) | 0x00000008;
}
void put_McBSP1_DX(int dx_stat)
{
        if(dx_stat==0)
                        * (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) & 0xffffffdf;
        else
                        •    (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) | 0x00000020;
}
void put_McBSP1_CLKR(int clkrp)
{
        if(clkrp==0)
                        * (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) & 0xfffffffe;
        else
                        •    (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) | 0x00000001;
}
void put_McBSP1_FSR(int fsrp)
{
        if(fsrp==0)
                        * (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) & 0xfffffffb;
        else
                        •    (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) | 0x00000004;
}

int get_McBSP1_CLKS_DR(void)
{
        int pcr;
        pcr = (int) ( * (UINT32 *) McBSP1_PCR );
        pcr = pcr & 0x00000050;
        return (pcr);
}
int get_McBSP1_FSR_CLKX_CLKR(void)
{
    int pcr;
    pcr = (int) ( * (UINT32 *) McBSP1_PCR );
    pcr = pcr & 0x00000007;
    return (pcr);
}
void put_McBSP1_CLKR_FSR(int clkr, int fsr)
{
    if(clkr==1)
        if(fsr==1)
            * (UINT32 *) McBSP1_PCR   = (* (UINT32 *) McBSP1_PCR) | 0x00000005;
        else
            •   (UINT32 *) McBSP1_PCR   = ((* (UINT32 *) McBSP1_PCR) | 0x00000001) &
                0xfffffffb ;
    else
        if(fsr==1)
            * (UINT32 *) McBSP1_PCR   = ((* (UINT32 *) McBSP1_PCR) | 0x00000004) &
0xfffffffe ;
        else
            •   (UINT32 *) McBSP1_PCR   = (* (UINT32 *) McBSP1_PCR) & 0xfffffffa ;
}
void put_McBSP1_CLKX_FSX(int clkx, int fsx)
{
    if(clkx==1)
        if(fsx==1)
            * (UINT32 *) McBSP1_PCR   = (* (UINT32 *) McBSP1_PCR) | 0x0000000a;
        else
            •   (UINT32 *) McBSP1_PCR   = ((* (UINT32 *) McBSP1_PCR) | 0x00000002) &
                0xfffffff7 ;
    else
        if(fsx==1)
            * (UINT32 *) McBSP1_PCR   = ((* (UINT32 *) McBSP1_PCR) | 0x00000008) &
0xfffffffd ;
        else
            •   (UINT32 *) McBSP1_PCR   = (* (UINT32 *) McBSP1_PCR) & 0xfffffff5 ;
}

void put_muxsel(int timer1, int dx)
{
    TIMER_setDatOut(_TIMER_hDev1, timer1);
    put_McBSP1_DX(dx);
}

/* The last 4 digits of McBSP1_PCR are the values for
    FSX        FSR        CLKX            CLKR
The input of the encoder is CLKR, The select lines of the encoder are the other 3
        ---------
        |    D    | -> IorV1 (111)
        |    E    | -> IorV0    (110)
        |    C    | -> Enable (101)
        |    O    | -> MorG      (100)
CLKR -> |    D    | -> Comm[3] -→ phase 1 on the board (011)
        |    E    | -> Comm[2] -→ phase 1 on the board (010)
        |    R    | -> Comm[1] -→ phase 1 on the board (001)
        |         | -> Comm[0] -→ phase 1 on the board (000)
        ---------
        /\ /\ /\
        FSR     FSX
            CLKX
*/
void set_IorV1(int iorv1)
```

```
{
      if (iorv1==1)
            * (UINT32 *) McBSP1_PCR    = (* (UINT32 *) McBSP1_PCR) | 0x0000000f;
      else if (iorv1==0)
            * T32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x0000000e) & 0xfffffffe;
            •     return;
}
void set_IorV0(int iorv0)
{
      if (iorv0==1)
            * (UINT32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x00000007) &
0xfffffff7;
      else if (iorv0==0)
            * T32 *) McBSP1_PCR   = ((* (UINT32 *) McBSP1_PCR) | 0x00000006) & 0xfffffff6;

            •     return;
}
void set_Enable(int Enable)
{
      if (Enable==1)
            * (UINT32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x0000000d) &
0xfffffffd;
      else if (Enable==0)
            * T32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x0000000c) & 0xfffffffc;
            •     return;
}
void set_MorG(int morg)
{
      if (morg==1)
            * (UINT32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x00000005) &
0xfffffff5;
      else if (morg==0)
            * T32 *) McBSP1_PCR   = ((* (UINT32 *) McBSP1_PCR) | 0x00000004) & 0xfffffff4;

            •     return;
}
void set_comm3(int comm   omm.   if (comm   omm.
            * (UINT32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x0000000b) &
0xfffffffb;
      else if (comm   omm.
            * T32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x0000000a) & 0xfffffffa;
            •     return;
}
void set_comm2(int comm   omm.   if (comm   omm.
            * (UINT32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x00000003) &
0xfffffff3;
      else if (comm   omm.
            * (UINT32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x00000002) &
0xfffffff2;
      return;
}
void set_comm1(int comm)
{
      if (comm==1)
            * (UINT32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x00000009) &
0xfffffff9;
      else if (comm==0)
            * (UINT32 *) McBSP1_PCR    = ((* (UINT32 *) McBSP1_PCR) | 0x00000008) &
0xfffffff8;
      return;
```

```
}
void set_comm0(int comm)
{
	if (comm==1)
		* (UINT32 *) McBSP1_PCR	= ((* (UINT32 *) McBSP1_PCR) | 0x00000001) &
0xfffffff1;
	else if (comm==0)
		* (UINT32 *) McBSP1_PCR	= (* (UINT32 *) McBSP1_PCR) & 0xfffffff0;
	return;
}
```

Appendix III. The torque equations for the overlap case

$$T_o(I_\phi,\theta) = T_{om}(I_\phi,\theta) + T_{of}(I_\phi,\theta)$$

$$T_{om}(I_\phi,\theta) = -\frac{nser}{npar}\frac{\mu\, Np^2\, Rg\, lstk\, STF}{L_{Fe,m} + \dfrac{\mu}{\mu_o} g} I_\phi^{\ 2} \qquad (I_\phi \le I_{m,sat})$$

$$= -\frac{nser}{npar}\frac{\mu\, Np^2\, Rg\, lstk\, STF}{L_{Fe,m} + \dfrac{\mu}{\mu_o} g} I_{m,sat}^2 + T_{1m}(I_\phi,\theta) \qquad (I_\phi > I_{m,sat})$$

$$T_{1m}(I_\phi,\theta) = nser\,\mu_o\, Np\, Rg\, lstk\, STF \frac{\dfrac{Np(I_\phi^{\ 2} - I_{m,sat}^{\ \ 2})}{2npar} + B_{sat}\, L_{Fe,m}\left(\dfrac{1}{\mu_1} - \dfrac{1}{\mu}\right)(I_\phi - I_{m,sat})}{g + \dfrac{\mu_1}{\mu} L_{Fe,m}}$$

$$T_{of}(I_\phi,\theta) = \frac{nser}{2npar}\mu\, Np^2\, lstk\, STF\, I_\phi^{\ 2}\, R_g \frac{L_{fe,f} + \mu\, g_f(\theta) - \theta\cdot\mu\cdot\dfrac{g_o R_g}{pwf}}{L_{fe,f} + \mu\, g_f(\theta)} \qquad (I_\phi \le I_{f,sat}(\theta))$$

$$= T_{of1}(I_\phi,\theta) + T_{of2}(I_\phi,\theta) \qquad (I_\phi > I_{f,sat}(\theta))$$

$$T_{of1}(I_\phi,\theta) = \frac{nser}{2npar}\mu\, Np^2\, lstk\, STF\, I_{f,sat}(\theta)^2\, R_g \frac{L_{fe,f} + \mu\, g_f(\theta) - \theta\cdot\mu\cdot\dfrac{g_o R_g}{pwf}}{L_{fe,f} + \mu\, g_f(\theta)}$$

$$T_{of2}(I_\phi,\theta) = nser\,\mu_o\, Np\, lstk\, STF\, R_g\left[\frac{Np(I_\phi^{\ 2} - I_{f,sat}(\theta)^2)}{2npar} + \right.$$

$$\left. B_{sat}\, L_{fe,f}\left(\frac{1}{\mu_1} - \frac{1}{\mu}\right)(I_\phi - I_{f,sat}(\theta)\right]$$

Appendix IV. The parameters of the switched reluctance motor

parameters	value	description
R_{shaft}	0.306 inches	The shaft radius
R_{ry}	0.846 inches	The rotor yoke radius
R_g	1.031 inches	The distance from the center to the air gap
R_{sy}	1.623 inches	The stator yoke radius
R_{out}	1.968 inches	The outside radius
g	0.009 inches	The thickness of the air gap
l_{stk}	1.983 inches	The stack lamination length
S_{tf}	0.9	The stacking factor
θ_p	23.82^o	Rotor pole width in degree
B_{sat}	1.6 tesla	The saturation flux density
μ	$1000\ \mu_o$	The permeability of the unsaturated iron
μ_1	$50\ \mu_o$	The permeability of the saturated iron
P	1.2 hp	The rated power output
P_{peak}	2 hp	The peak power output
ω_{max}	15,000 rpm	The maximum rotor speed
La	1.27 mH	The inductance at the aligned position
Lu	0.19 mH	The inductance at the unaligned position

Note:

1. Since the manufacturer considers the parameters of the motor proprietary information, so the values shown here are measured or estimated and might not be accurate.
2. Due to the inductance asymmetry, the inductance at the aligned and unaligned position is obtaind for phase A with a certain pair of rotor poles.

REFERENCE

[1] Salmasi, F.R.; Fahimi, B.; Gao, H.; Ehsani, M.; Sensorless control of switched reluctance motor drive based on BEMF calculation, Applied Power Electronics Conference and Exposition, 2002. APEC 2002. Seventeenth Annual IEEE , Volume: 1 , 2002 Page(s): 293 -298 vol.1

[2] Husain I.; Radun A.; Nairus J.; Fault analysis and excitation requirements for switched reluctance generators, IEEE transactions on energy conversion, Volume: 17, No.1 2002 Page(s): 67 -72

[3] T.J.E.Miller, Switched reluctance motors and their control, Magna physics publications, 1993

[4] S. A. Nasar, DC switched reluctance motor, IEE proceedings, Vol 116, No.6, 1048-1049, 1969

[5] Cameron D. E. and J. H. Lang, ; Variable-Reluctance Generators in Electric Power System; IEEE Transactions on Industry Applications, Vol. 29, No. 6, 1993

[6] A. V. Radun, "Analytical calculation of the switched reluctance motor's unaligned inductance," IEEE Transactions on Magnetics, Vol: 35, No. 6, pp 4473-4481, November 1999

[7] A. V. Radun, Analytically computing the flux linked by a switched reluctance motor phase when the stator and the rotor poles overlap, IEEE transactions on magnetics, Vol. 36, Issue 4, July 2000, Page(s) 1996-2003

[8] Miller,T.J.E.;McGilp,M, "Nonlinear theory of the switched reluctance motor for rapid computer-aided design", Electric Power Applications, IEE Proceedings B [see also IEE Proceedings-Electric Power Applications] ,Volume: 137 , Issue: 6, Nov. 1990 Pages:337 – 347

[9] Raulin, V., Husain , I., and Radun, A. "Modeling of Losses in Switched Reluctance Machines," IEEE Trans. On Industry App., p 227-234, Nov./Dec. 2004

[10] P. Materu and R. Krishnan, "Estimation of Switched Reluctance Motor Losses", IEEE Industry Applications Society Annual Conf. Rec., Pittsburgh, pp. 79-90, 1988

[11] Edrington, C.S.; Fahimi, B., "An auto-calibrating model for an 8/6 switched reluctance motor drive: application to design and control", Power Electronics Specialist, 2003. PESC '03. IEEE 34th Annual Conference on ,Volume: 1 , 15-19 June 2003 Pages:409 - 415 vol.1

[12] Brosse, A.; Henneberger, G.; Schniedermeyer, M.; Lorenz, R.D.; Nagel, N.; "Sensorless control of a SRM at low speeds and standstill based on signal power evaluation", Industrial Electronics Society, 1998. IECON '98. Proceedings of the 24th Annual Conference of the IEEE , Volume: 3 , 31 Aug.-4 Sept. 1998 Pages:1538 - 1543 vol.3

[13] Harris, W.D.; Lang, J.H.; "A simple motion estimator for variable-reluctance motors", Industry Applications, IEEE Transactions on ,Volume: 26 , Issue: 2 , March-April 1990 Pages:237 – 243

[14] Gao, H.; Salmasi, F.R.; Ehsani, M.; "Sensorless control of SRM at standstill", Applied Power Electronics Conference and Exposition, 2001. APEC 2001. Sixteenth Annual IEEE , Volume: 2 , 4-8 March 2001 Pages:850 - 856 vol.2

[15] Suresh, G.; Fahimi, B.; Ehsani, M.; "Improvement of the accuracy and speed range in sensorless control of switched reluctance motors", Applied Power Electronics Conference and Exposition, 1998. APEC '98. Conference

Proceedings 1998., Thirteenth Annual ,Volume: 2 , 15-19 Feb. 1998 Pages:771 - 777 vol.2

[16] Lumsdaine, A.; Lang, J.H.; Balas, M.J.; "A State Observer for Variable Reluctance Motors: Analysis and Experiments", Circuits, Systems and Computers, 1985. Nineteeth Asilomar Conference on , November 6-8, 1985 Pages:660 – 664

[17] Lumsdaine, A.; Lang, J.H.; "State observers for variable-reluctance motors" Industrial Electronics, IEEE Transactions on ,Volume: 37 , Issue: 2 , April 1990 Pages:133 – 142

[18] Husain, I.; Islam, M.S.; "Observers for position and speed estimations in switched reluctance motors", Decision and Control, 2001. Proceedings of the 40th IEEE Conference on , Volume: 3 , 4-7 Dec. 2001 Pages:2217 - 2222 vol.3

[19] Hossain, S.A.; Husain, I.; Klode, H.; Lequesne, B.; Omekanda, A.M.; Gopalakrishnan, S.; "Four-quadrant and zero-speed sensorless control of a switched reluctance motor", Industry Applications, IEEE Transactions on , Volume: 39 , Issue: 5 , Sept.-Oct. 2003 Pages:1343 – 1349

[20] Hossain, S.A.; Husain, I.; "A geometry based simplified analytical model of switched reluctance machines for real-time controller implementation", Power Electronics, IEEE Transactions on , Volume: 18 , Issue: 6 , Nov. 2003, Pages:1384 – 1389

[21] Yang, I.-W.; Kim, Y.-S.; "Rotor speed and position sensorless control of a switched reluctance motor using the binary observer", Electric Power Applications, IEE Proceedings- , Volume: 147 , Issue: 3 , May 2000 Pages:220 – 226

[22] Suresh, G.; Fahimi, B.; Rahman, K.M.; Ehsani, M.; "Inductance based position encoding for sensorless SRM drives", Power Electronics Specialists Conference, 1999. PESC 99. 30th Annual IEEE ,Volume: 2 , 27 June-1 July 1999 Pages:832 - 837 vol.2

[23] Fahimi, B.; Emadi, A.; Sepe, B., Jr.; "Position sensorless control", Industry Applications Magazine, IEEE ,Volume: 10 , Issue: 1 , Jan-Feb 2004 Pages:40 – 47

[24] Salmasi, F.R.; Fahimi, B.; Gao, H.; Ehsani, M.; "Robust sensorless rotor position detection in switched reluctance motors for low speed applications", Power Electronics Specialists Conference, 2001. PESC. 2001 IEEE 32nd Annual , Volume: 2 , 17-21 June 2001 Pages:839 - 843 vol.2

[25] Salmasi, F.R.; Fahimi, B.; Gao, H.; Ehsani, M.; "Sensorless control of switched reluctance motor drive based on BEMF calculation"", Applied Power Electronics Conference and Exposition, 2002. APEC 2002. Seventeenth Annual IEEE , Volume: 1 , 10-14 Pages:293 - 298 vol.1 March 2002

[26] Gao, H.; Fahimi, B.; Salmasi, F.R.; Ehsani, M.; "Sensorless control of the switched reluctance motor drive based on the stiff system control concept and signature detection", Industry Applications Conference, 2001. Thirty-Sixth IAS Annual Meeting. Conference Record of the 2001 IEEE , Volume: 1 , 30 Sept.-4 Oct. 2001 Pages:490 - 495 vol.1

[27] Lyons, J.P.; MacMinn, S.R.; Preston, M.A.; "Flux-current methods for SRM rotor position estimation", Industry Applications Society Annual Meeting, 1991., Conference Record of the 1991 IEEE , 28 Sept.-4 Oct. 1991 Pages:482 - 487 vol.1

[28] Mondal, S.K.; Saxena, S.N.; Bhadra, S.N.; Muni, B.P.; "Evaluation of a novel analog based closed-loop sensorless controller for switched reluctance motor

drive", Industry Applications Conference, 2001. Thirty-Sixth IAS Annual Meeting. Conference Record of the 2001 IEEE , Volume: 3 , 30 Sept.-4 Oct. 2001 Pages:2073 - 2080 vol.3

[29] Mese, E.; Torrey, D.A.; "An approach for sensorless position estimation for switched reluctance motors using artifical neural networks", Power Electronics, IEEE Transactions on , Volume: 17 , Issue: 1 , Jan. 2002 Pages:66 – 75

[30] Fahimi, B.; Suresh, G.; Ehsani, M.; ""Review of sensorless control methods in switched reluctance motor drives", Industry Applications Conference, 2000. Conference Record of the 2000 IEEE ,Volume: 3 , 8-12 Oct. 2000 Pages:1850 - 1857 vol.3

[31] D. A. Torrey and J. H. Lang, "Modeling a nonlinear variable reluctance motor drive," IEE Proceedings, Vol. 137, pt. B, pp. 314-326, 1990

[32] T. J. E. Miller and McGilp, "Nonlinear theory of the switched reluctance motor for rapid computer-aided design," IEE Proceedings, Vol. 137, pt. B, pp. 337-347, 1990

[33] M. Stiebler and K. Liu, "An Analytical Model of Switched Reluctance machines," IEEE Transactions on Energy Conversion, Vol. 14, No. 4, pp. 1100-1105, Dec. 1999

[34] P. G. Barrass and B. C. Mecrow, “Flux and torque control of switched reluctance machines”, Electric Power Applications, IEE Proceedings- , Volume: 145 Issue: 6, Nov. 1998, Page(s): 519 –527

[35] Suresh, G.; Fahimi, B.; Rahman, K.M.; Ehsani, M.;’Inductance based position encoding for sensorless SRM drives’, Power Electronics Specialists Conference, 1999. PESC99. 30th Annual IEEE, Volume: 2 , 27 June-1 July 1999 Pages:832 - 837 vol.2

[36] Radun A. V., "Design considerations for the switched reluctance motor," IEEE Transactions on Industry Applications, Vol. 31, No. 5, pp. 1079 - 1087, September/October, 1995

[37] Radun A. V., and Y. Q. Xiang., "Switched reluctance starter/generator system modeling results," Trans. SAE, J. Aerosp., vol. 104, sec. 1, pp. 257-266, 1995

[38] Hussain, I., Radun, A. V., and Nairus, J., "Unbalanced Force Calculation in Switched Reluctance Machines," IEEE Transactions on Magnetics, Vol. 36, n. 1., pp. 330-338 January 2000

[39] Brosse, A.; Henneberger, G.; Schniedermeyer, M.; Lorenz, R.D.; Nagel, N.; "Sensorless control of a SRM at low speeds and standstill based on signal power evaluation", Industrial Electronics Society, 1998. IECON '98. Proceedings of the 24th Annual Conference of the IEEE , Volume: 3 , 31 Aug.-4 Sept. 1998 Pages:1538 - 1543 vol.3

[40] Harris, W.D.; Lang, J.H.; "A simple motion estimator for variable-reluctance motors", Industry Applications, IEEE Transactions on, Volume: 26, Issue: 2, March-April 1990 Pages: 237 – 243

[41] Zhang, Jinhui; Radun, A.V.; "A new method to measure the switched reluctance motor’s flux"; Applied Power Electronics Conference and Exposition, 2005, APEC 2005, Twentieth annual IEEE, Volume 3, 6-10 March, 2005, pages 1994-1999

[42] Zhang, Jinhui; Radun, A.V.; "A simplified analytical flux model of the switched reluctance motor "; International Electric Motors and Drives Conference, 2005, IEMDC 2005, Twentieth annual IEEE, Volume 3, 6-10 March, 2005, pages 1994-1999

VITA

Name: Jinhui Zhang
Date of Birth: September 3, 1976
Place of Birth: Zhengzhou, China
Eductaion:

- University of Kentucky, Expected in October 2005 **Ph. D.** of Electrical Engineering
- Tianjin University, China, March 2001 **M. S.** of Electrical Engineering
- Zhengzhou University, China, June 1997 **B. S.** of Electrical Engineering

Experience:

- Research Assistant, August 2001 to present, University of Kentucky
- Research Assistant, August 1998 to March 2001, Tianjin University

Publications:

- **Jinhui Zhang**, Arthur Radun, A simplified switched reluctance motor's flux model. IEEE's International Electrical Motor Drive Conference (IEMDC), May 2005
- **Jinhui Zhang**, Arthur Radun, A new method to measure switched reluctance motor's flux model. IEEE's Applied Power Electronics Conference (APEC), March 2005
- Ping Wang, **Jinhui Zhang**, A single phase active filter design and implementation. Power Supply Magazine, April 2000

Honors:

- Ranked #1 in the 1994 industrial automation class
- IEEE student member
- SAE student member