Control and Design of Multi-Use Induction Machines:
Traction, Generation, and Power Conversion

by

Al-Thaddeus Avestruz

S.B. in Physics, Massachusetts Institute of Technology (1994)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2006

© Massachusetts Institute of Technology 2006. All rights reserved.

Author

Department of Electrical Engineering and Computer Science
25 May 2006

Certified by:

Steven B. Leeb
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by:

Arthur C. Smith
Chairman, Department Committee on Graduate Students
Control and Design of Multi-Use Induction Machines: Traction, Generation, and Power Conversion

by

Al-Thaddeus Avestruz

Submitted to the Department of Electrical Engineering and Computer Science on 25 May 2006, in partial fulfillment of the requirements for the degree of Master of Science

Abstract

An electrical machine can be made to convert electrical power while performing in its primary role of transforming electrical energy into mechanical energy. One way of doing this is to design the machine with multiple stator windings where one winding acts as a primary for drive and power, and the others as secondaries for electrical power. The challenge is to control the mechanical outputs of torque and speed while independently regulating the electrical outputs of voltage and current. This thesis analyzes and demonstrates an approach that takes advantage of topological symmetries in multiphase systems to overcome this challenge. This method is applied, but not relegated to induction machines.

Thesis Supervisor: Steven B. Leeb
Title: Professor of Electrical Engineering and Computer Science
Acknowledgments

I would like to thank my advisor Professor Steven Leeb for his efforts, patience, encouragement, and confidence in me, without which, all of this would not be possible.

To Meelee Kim, for her unwavering love and support.

To my parents, Suzanne and Fred, and my grandparents for their prayers and love: Lola Beth, Lola Jeannette, and Lolo Nes, who started my interest in science and engineering. In loving memory of my grandfather Alfonso Palencia.

To my brother Mark, without whose many discussions, I would have a lesser perspective of my work. To my sister Camille, whose future I look forward to with great confidence.

To Robert Cox, Chris Laughman, and Warit Witchakool for the many technical discussions that made work so much more interesting. To my office-mate Brandon Pierquet, for the discussions and entertainment and for the solutions to the little annoyances in LaTex.

And to everyone else in LEES, especially Professors Kirtley and Perreault, and to Vivian Mizuno.

I am sure to have forgotten to mention many who have helped me, but I am grateful despite the short lapse in memory.

This work was supported by the Center for Materials Science and Engineering, a Materials Research Science and Engineering Center funded by the National Science Foundation under award number DMR 02-13282, and by the Grainger Foundation.
# Contents

1 Introduction ........................................... 17  
1.1 Machines with Multi-Use Capability .................. 18  
1.2 Design Challenges and Innovations ................. 19  
1.3 Previous Work ........................................ 20

2 Power Conversion Control By Zero Sequence Harmonics 21  
2.1 Induction Machine Model ................................. 21  
2.2 A Transformer Model for a Stator with Multiple Windings ........... 24  
2.3 Zero Sequence Harmonic Control in Three Phase Systems .......... 26  
  2.3.1 Voltage Output Control by Harmonic Amplitude Variation ...... 29  
  2.3.2 Zero Sequence Circuit .................................. 32  
2.4 Toward a Machine Design ................................ 34  
  2.4.1 Fundamental Design Limitations ...................... 35  
  2.4.2 Zero Sequence Control of Three-Phase, Three-Legged Transformer .... 35  
  2.4.3 Scaling Laws .............................................. 35

3 Vector Drive Control ...................................... 39  
3.1 Constant Volts per Hertz Drive ................. 39  
3.2 Indirect Field Oriented Control .................. 40  
3.3 Cartesian Feedback in the Synchronous Current Regulator .......... 41  
3.4 Integrator Anti-Windup .................................. 42  
3.5 Simulation .................................................. 45  
3.6 Implementation ............................................ 47

4 Inverter Design ........................................... 49  
4.1 Power Module and Digital Signal Processor ............. 49  
4.2 Sine Wave Generation .................................... 49  
  4.2.1 Synchronous PWM ...................................... 49  
  4.2.2 Table-Based Implementation ...................... 51  
  4.2.3 Parabolic Approximations ............................. 51  
4.3 Three Phase Filters with Four Legs ................ 58  
4.4 Minimal Implementations for Phase Current Measurement ........ 63
Table of Contents

4.4.1 Balanced Three Phase, Wye Grounded and Ungrounded ........................................ 63
4.4.2 Multiple Stator ........................................................................................................... 65
4.4.3 Estimation from Inverter Current Out of the DC Bus .................................................. 66

5 Firmware Design .................................................................................................................. 69
  5.1 Time Slicing Algorithm ................................................................................................. 69
  5.2 Data Structures ............................................................................................................. 70
  5.3 Output Voltage Regulation Module ............................................................................. 71
  5.4 Synchronous PWM Module .......................................................................................... 71
  5.5 Synchronous Current Controller .................................................................................... 71

6 Conclusions and Future Work ............................................................................................... 75

A SPICE Deck for Multistator Transformer ............................................................................ 77
  A.1 PSPICE–Wye-Ungrounded ............................................................................................ 77
  A.2 PSPICE–Wye-Grounded ............................................................................................... 80
  A.3 PSPICE–Wye-Grounded with Zero Sequence Transformer ........................................... 83

B MATLAB Script for Parabolic Approximations of Sine Function ......................................... 87
  B.1 Script ............................................................................................................................. 87
  B.2 Functions ....................................................................................................................... 97
    B.2.1 h1fit() ..................................................................................................................... 97
    B.2.2 h2fit() ..................................................................................................................... 98
    B.2.3 hinffit() ............................................................................................................... 98
    B.2.4 mindistfit() ......................................................................................................... 98
    B.2.5 thdopt() ............................................................................................................... 98
    B.2.6 thd() ..................................................................................................................... 98
    B.2.7 infnorm() .............................................................................................................. 99

C Field Oriented Control Simulink Model .............................................................................. 101
  C.1 Block Models .............................................................................................................. 101
  C.2 Induction Motor S-Function ......................................................................................... 105

D Motor Control Embedded Firmware ...................................................................................... 113
  D.1 Main Motor Control Module .......................................................................................... 113
    D.1.1 mot_ctrl.c ........................................................................................................... 113
    D.1.2 umacros.h .......................................................................................................... 123
  D.2 Sine Inverter PWM Module ........................................................................................ 126
    D.2.1 sin_pwm.c ....................................................................................................... 126
    D.2.2 Header Files ...................................................................................................... 132
    D.2.3 sin_pwm.h ....................................................................................................... 133
  D.3 Volts per Hertz Module ................................................................................................ 135
    D.3.1 vfcontrol.c ........................................................................................................ 135

~ 8 ~
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>D.3.2 vfcontrol.h</td>
<td>142</td>
</tr>
<tr>
<td>D.4 Serial Communications Module</td>
<td>143</td>
</tr>
<tr>
<td>D.4.1 serialcomm.c</td>
<td>143</td>
</tr>
<tr>
<td>D.4.2 serialcomm.h</td>
<td>150</td>
</tr>
<tr>
<td>D.5 Peripheral Driver Module</td>
<td>151</td>
</tr>
<tr>
<td>D.5.1 periphs.c</td>
<td>151</td>
</tr>
<tr>
<td>D.5.2 periphs.h</td>
<td>153</td>
</tr>
</tbody>
</table>
### List of Figures

1.1 Photograph of Multi-Use Induction Motor Testbed ........................................ 18  
1.2 Multi-Stator Induction Machine ................................................................. 19  
2.1 Electrical Model of Three-Legged Transformer .............................................. 24  
2.2 Transformer Model In-Circuit for Identical Stators Wound In-Hand ................. 25  
2.3 Effect of Series Inductance in the 3rd Harmonic Control Function ................. 26  
2.4 Multiple Stator Connections Driving Three-Phase Rectifiers ......................... 27  
2.5 Drive Waveform with Third Harmonic ........................................................... 28  
2.6 3d Control Surface for Peak Amplitude Control .......................................... 29  
2.7 Third Harmonic Peak Amplitude Control Plotted Parametrically with Phase ....... 30  
2.8 PI Control of Dc Output Using 3rd Harmonic ................................................ 31  
2.9 T-Model for the Zero Sequence Circuit ........................................................ 32  
2.10 Zero sequence transformer ............................................................................. 33  
2.11 Direct zero sequence transformer .................................................................... 34  
2.12 Exogenously driven zero sequence transformer .............................................. 34  
2.13 Utilization and Scaling Using Triple Insulated Wire ....................................... 37  
3.1 DQ-Space Contour and Allowed Trajectory in Synchronous Current Regulator Saturation ............................................................................................................ 44  
3.2 Simulation of induction motor under field-oriented control with speed and torque load steps .............................................................................................................. 46  
3.3 Implementation of a Field-Oriented Speed Controller for a Multi-Use Induction Machine ......................................................................................................... 48  
4.1 International Rectifier’s Integrated Power Module ............................................ 50  
4.2 Sine Reference from 108-element DLT ............................................................... 52  
4.3 THD and Maximum Error versus Table Length for Direct Lookup Table ........... 52  
4.4 Inverter Voltage Waveforms ............................................................................. 53  
4.5 Inverter Current for Fundamental + 25% Third Harmonic .................................. 53  
4.6 Error from Half-Wave Parabolic Sine Approximations ..................................... 57  
4.7 Errors from Quarter-Wave Parabolic Approximations to a Sine ....................... 59  
4.8 Line-to-line Three Phase Filter ........................................................................... 61
List of Figures

4.9 Phase-to-Neutral Three Phase Filter .............................................. 62
4.10 Coupled Inductor Three Phase Filter .............................................. 64
4.11 Recovery of Rotor-Linked Current from Linear Combination of Stator Current Measurements .............................................. 66
4.12 FFT of Stator Currents in Rotor-Linked Current Recovery ...................... 67

5.1 Activity Diagram for Closed-Loop Control of Dc Output ......................... 72
5.2 Activity Diagram for Synchronous Current Controller ............................ 73

A.1 PSPICE Schematic–Wye-Ungrounded .............................................. 78
A.2 PSPICE Schematic–Wye-Grounded .............................................. 81
A.3 PSPICE Schematic–Wye-Grounded with Zero Sequence Transformer ............ 84

C.1 Top Level Model ................................................................. 102
C.2 Speed Controller ................................................................. 103
C.3 Field Oriented Controller .......................................................... 104
C.4 Induction Motor Model .............................................................. 105
List of Tables

3.1 Aardvark Machine Parameters ........................................ 45
4.1 THD and Maximum For a DLT Sine Reference Using Different Interpolating Functions ........................................ 51
4.2 Coefficients for Half-Wave Sine Approximations ................. 58
4.3 Coefficients for Quarter-Wave Sine Approximations ............. 60
5.1 Examples of Task Timing ............................................. 70
fficiency, economy and elegance are hallmarks of good design. The idea of multi-use machines is an attempt to mitigate the waste and superfluity that is needlessly epidemic in a contemporary world starved of energy.

Motor and generator drives have been and continue to be crucial to a wide range of industrial and commercial products and manufacturing processes; among these, variable speed drives (VSDs) that operate over a range of mechanical shaft speeds are invaluable. Since the beginning, folks have found ways to vary the shaft speed of a motor; many exist, but circuit design and components in power electronics have advanced to a point were it is more appealing, both in performance and economics, for motors to be combined with power electronics in commercial and industrial VSDs.[1]

Many products use VSDs, including modern air handling and ventilation systems that run fans at speeds and power that are optimal, ensuring occupant comfort while keeping energy consumption to a minimum. Other examples include computer-controlled machine tools such as mill machines and lathes that need continuously variable speeds for different materials and tasks; hybrid and electric vehicles (EVs) use variable speed drives for traction or propulsion. These systems typically use power electronics to control the flow of power to the motor in controlling the speed. In addition to needing power for their drives, these systems often include other power supplies that typically includes distribution through a network of power buses through different parts of the system.
Chapter 1: Introduction

Figure 1.1: Photograph of what we have fondly called the Aardvark machine.

1.1 Machines with Multi-Use Capability

An electrical machine can be made to convert electrical power while performing in its primary role of transforming electrical energy into mechanical energy. One way of doing this is to design the machine with multiple stator windings where one winding acts as a primary for drive and power, and the others as secondaries for electrical power. The challenge is to control the mechanical outputs of torque and speed while independently regulating the electrical outputs of voltage and current.

The salient feature in these multi-use machines is the integration of magnetics for traction, generation and power conversion. While it is not eminently clear whether there is a convincing scaling law advantage in size and weight with this type of integration, it is evident that there is an economy to using the same control and power electronics for multiple purposes. Beyond rudimentary calculations for simple integrated pole face geometries, detailed studies of scaling laws for a variety of structures and magnetic circuit configurations is largely outside the scope of this work and is an obvious topic for follow-on research in machine design.

This thesis report analyzes and demonstrates an approach that takes advantage of topological symmetries in multi-phase systems to overcome this challenge. This method has been applied, but not relegated to induction machines.
1.2 Design Challenges and Innovations

There are a number of design challenges that must be addressed before one can assert whether this technology is viable for commercial and industrial applications. Overcoming the first challenge of proof of concept in a previous thesis [2] opened an avenue for continued research.

The first of the questions asks how one provides a wide range and monotonic control of the dc secondary output. It is answered by the use of the 3\textsuperscript{rd} harmonic amplitude at a phase of $\pi$ radians for closed-loop control of the output; in general, the control-to-output function for dc output voltage is non-monotonic for other values of phase. Also, a look in [2] shows that the control of dc output using zero sequence phase, while effective, only provides a narrow range of variation.

The next addresses an issue that is endemic in most contraptions that we would like to use for multi-use machines: it is that of vanishingly small zero sequence reactance. A good solution, while not universal, is a variety of topologies that use a zero sequence transformer to improve the magnetizing inductance in the zero sequence circuit. Another solution, of course, is to redesign the machines, but that is the topic for another thesis.

Then, we must talk about the inverter. Not only must it be very good sinusoidal current source, but it must also be a very good voltage source of 3\textsuperscript{rd} or other zero sequence harmonics; it is the way you integrate the idea of zero sequence voltage with a field-oriented controller that is based on stator currents, which is the established means for variable speed drive.

This begs the question of generality. We talk for example about cars having two independent dc buses for power: the new one is 42 volts and the legacy is 12V. We can perhaps increase the number of phases in the machine so that there is more than one set of "independent" zero sequence harmonics. There are a number of ways to handle this: one can have a machine that physically has a greater number of inherent phases; the other, is to create these additional phases with linear combinations of the inherent phases. So, we start take a look at general
Chapter 1: Introduction

n-phase circuits that provide supernumerary zero sequence harmonic sets, and heterophasic transformer circuits.

How far can we go? As always, it depends on the compromises that one is willing to accept. It also matters how well we can design the machine, and this depends on at least an initial understanding of the design limitations and scaling laws. The complement to "how far" is "where else" and this is a question about machines other than induction ones. A glance at the ubiquity of the Lundell alternator adds not only to the intellectual, but also to the economic appeal.

This work provides some of the foundations and proof-of-concepts.

1.3 Previous Work

Initial work and construction of the induction machine testbed had been performed by Jack W. Holloway in a previous thesis project [2]. This work included the demonstration of dc output control of a wye-grounded winding by varying the phase of the added 3rd harmonic in the inverter and the independence to zero sequence harmonics of an identical winding with the wye ungrounded.
Chapter 2

Power Conversion Control By Zero Sequence Harmonics

Zero sequence current in a multi-phase system is the portion of current that runs in the same direction through all the connection phases at the same time. This means that this component of the current has the same amplitude and angular phase in every connection phase.\(^1\) It is the part that we can consider "common-mode" to all phase connections: the vector sum of the currents in these phases.\(^2\)

One can also speak of a zero sequence voltage. If the circuit has a balanced terminal impedance and a zero sequence path, then it is that portion of the voltage at each phase that creates a zero sequence current. Often, we describe a zero sequence voltage in a way that is similar to a zero sequence current: having the same amplitude and angular phase in every connection, but it is not necessarily the case that a zero sequence voltage results in zero sequence current, as it is also the case that a positive or negative sequence voltage can result in a zero sequence current.

2.1 Induction Machine Model

A three phase induction machine can be described by the magnetic flux linkages among three stationary windings (stator) and three moving windings (rotor). Equation 2.1a describes every permutation of how the flux through every winding is linked to a current in its own and every

\(^1\)To avoid ambiguity, we make a distinction in this paragraph between connection phase, which is the actual physical connection to the circuit, and angular phase, which is the constant parameter in the argument of a periodic function that for the moment we assume to be unmodulated.

\(^2\)Though a Fourier transform, periodic signals can be represented as a sum of sinusoids. These sinusoids in turn can be represented as phasors.
Chapter 2: Power Conversion Control By Zero Sequence Harmonics

others’ winding.[3]

\[
\begin{bmatrix}
\lambda_S \\
\lambda_R
\end{bmatrix} =
\begin{bmatrix}
L_S & L_{SR} \\
L_{SR}^T & L_R
\end{bmatrix}
\begin{bmatrix}
i_S \\
i_R
\end{bmatrix}
\] (2.1a)

where the stator flux in the stationary reference frame

\[
\lambda_S =
\begin{bmatrix}
\lambda_{as} \\
\lambda_{bs} \\
\lambda_{cs}
\end{bmatrix} =
\begin{bmatrix}
L_s & L_{ab} & L_{ab} \\
L_{ab} & L_s & L_{ab} \\
L_{ab} & L_{ab} & L_s
\end{bmatrix}
\begin{bmatrix}
i_{as} \\
i_{bs} \\
i_{cs}
\end{bmatrix}
\] (2.1b)

the rotor flux

\[
\lambda_R =
\begin{bmatrix}
\lambda_{ar} \\
\lambda_{br} \\
\lambda_{cr}
\end{bmatrix} =
\begin{bmatrix}
L_r & L_{ab} & L_{ab} \\
L_{ab} & L_r & L_{ab} \\
L_{ab} & L_{ab} & L_r
\end{bmatrix}
\begin{bmatrix}
i_{ar} \\
i_{br} \\
i_{cr}
\end{bmatrix}
\] (2.1c)

and the mutual inductance matrix between the stator and the rotor

\[
L_{SR} =
\begin{bmatrix}
L_c \cos \theta_r & L_c \cos(\theta_r + 2\pi/3) & L_c \cos(\theta_r - 2\pi/3) \\
L_c \cos(\theta_r - 2\pi/3) & L_c \cos \theta_r & L_c \cos(\theta_r + 2\pi/3) \\
L_c \cos(\theta_r + 2\pi/3) & L_c \cos(\theta_r - 2\pi/3) & L_c \cos \theta_r
\end{bmatrix}
\] (2.1d)

When one applies the well-known Park’s transform [4]

\[
T = \frac{2}{3}
\begin{bmatrix}
\cos \theta & \cos \left(\theta - \frac{2\pi}{3}\right) & \cos \left(\theta + \frac{2\pi}{3}\right) \\
-\sin \theta & -\sin \left(\theta - \frac{2\pi}{3}\right) & -\sin \left(\theta + \frac{2\pi}{3}\right) \\
\frac{1}{2} & \frac{1}{2} & \frac{1}{2}
\end{bmatrix}
\] (2.2a)

and for completeness delineating its inverse

\[
T^{-1} =
\begin{bmatrix}
\cos \theta & -\sin \theta & 1 \\
\cos \left(\theta - \frac{2\pi}{3}\right) & -\sin \left(\theta - \frac{2\pi}{3}\right) & 1 \\
\cos \left(\theta + \frac{2\pi}{3}\right) & -\sin \left(\theta + \frac{2\pi}{3}\right) & 1
\end{bmatrix}
\] (2.2b)

~ 22 ~
to 2.1a, the flux linkages become block matrices, each of which are diagonal and time-invariant,

\[
\begin{bmatrix}
\lambda_{dqs} \\
\lambda_{dqr}
\end{bmatrix} =
\begin{bmatrix}
L_s & M \\
M & L_R
\end{bmatrix}
\begin{bmatrix}
i_{dqs} \\
i_{dqr}
\end{bmatrix},
\]

(2.3a)

where

\[
L_s = \begin{bmatrix}
L_{as} & 0 \\
0 & L_{as}
\end{bmatrix},
\]

(2.3b)

\[
L_R = \begin{bmatrix}
L_{ar} & 0 \\
0 & L_{ar}
\end{bmatrix},
\]

(2.3c)

\[
M = \begin{bmatrix}
M & 0 \\
0 & M
\end{bmatrix},
\]

(2.3f)

\[
M = \frac{3}{2} L_c.
\]

(2.3g)

The state equations for the induction motor using flux are given by

\[
\begin{align*}
-\dot{\lambda}_{ds} &= r_s i_{ds} - \omega \lambda_{qs} - v_{ds} \\
-\dot{\lambda}_{qs} &= r_s i_{qs} + \omega \lambda_{ds} - v_{qs} \\
-\dot{\lambda}_{dr} &= r_r i_{dr} - \omega_s \lambda_{qr} \\
-\dot{\lambda}_{qr} &= r_r i_{qr} + \omega_s \lambda_{dr},
\end{align*}
\]

(2.4a)

(2.4b)

(2.4c)

(2.4d)

and torque of electrical origin by

\[
\tau_m = \frac{3}{2} p (\lambda_{qr} i_{dr} - \lambda_{dr} i_{qr}),
\]

(2.5)

and the equations of motion by

\[
\dot{\omega}_r = \frac{1}{J} (\tau_m - \tau_l),
\]

(2.6)

where \( p \) is the number of pole pairs, \( J \) is the moment of inertia, and \( \tau_l \) is the load torque.
2.2 A Transformer Model for a Stator with Multiple Windings

Each phase in the stator windings is shifted by 120 electrical degrees; e.g., the flux linked between phase a and phase b is given by

\[ L_{ab} = L_c \cos(120^\circ) = \frac{1}{2} L_c. \]  

(2.7)

Usually, the coupling is symmetric as well as equal among the phases, i.e., \( L_{ab} = L_{ac} = L_{ba} = \ldots \).

In a machine with multiple stators, additional flux linkages exist between the stator windings. Between two stators,

\[
\begin{bmatrix}
\lambda_{S_1} \\
\lambda_{S_2}
\end{bmatrix} =
\begin{bmatrix}
L_{S_1} & M_{12} \\
M_{21} & L_{S_2}
\end{bmatrix}
\begin{bmatrix}
\dot{i}_{S_1} \\
\dot{i}_{S_2}
\end{bmatrix}.
\]

(2.8)

In the next section, one will see how this bears more than just a casual similarity to a three-legged, three-phase transformer. In fact, the multiple stators of an induction by themselves behave the same way as the transformer illustrated in Figure 2.1, despite being wrapped around a circle.

![Figure 2.1: Electrical model for the primary of a Three-Legged Transformer. \( k_p \) is the phase-to-phase coupling coefficient.](image)

The coupling coefficient between windings of a transformer is the fraction of the flux coupled
Section 2.2: A Transformer Model for a Stator with Multiple Windings

between the primary and the secondary and is given by

\[ k = \frac{M}{\sqrt{L_1 L_2}}, \]  

(2.9a)

which generalizes to a matrix element

\[ k_{mn} = \frac{M_{mn}}{\sqrt{L_m L_n}} \]  

(2.9b)

that relates an arbitrary winding to another, where \( L_1 \) and \( L_2 \) are the primary and secondary side open-circuit inductances, respectively, and \( M \) is the mutual inductance between windings.

The inductance matrix given by Equation 2.8 has been implemented in PSPICE as illustrated in Figure 2.2.

![Diagram](image)

Figure 2.2: In-circuit transformer model of the flux linkage between two identical stator windings wound in-hand. A zero sequence transformer is described in §2.3.2.

This coupling between phases can be implemented in SPICE by coupled inductor statements; Appendix A contains the SPICE deck as well as the schematic file for PSPICE. Note that in Figure 2.2, the negative coupling coefficients and subsequent negative inductances are captured in the winding polarity of the coupled inductors, which helps to illustrate a better physical

~ 25 ~
sense of what the flux is doing. Figure 2.3 illustrates how the model can be used to determine parameter effects on the system design.

![Graph showing simulation result](image)

**Figure 2.3:** Simulation result of the effect of a change on the series inductance in the 3rd harmonic control function.

### 2.3 Zero Sequence Harmonic Control in Three Phase Systems

By introducing one or more triple-n harmonics into the drive voltage of a three-phase machine, the rectified output voltage from grounded-wye windings can be controlled and subsequently regulated. To first-order, these triple-n harmonic voltages produce triple-n harmonic currents, and introduce negligible net torque on the rotor, effectively decoupling voltage regulation from drive.

The rectifiers in Figure 1.1 are designed to operate in the discontinuous conduction mode. In this case, rectifiers in winding 2 behave as peak detectors of the line-to-neutral voltages, with
Section 2.3: Zero Sequence Harmonic Control in Three Phase Systems

Figure 2.4: Multiple Stator Connections Driving Three-Phase Rectifiers.
Chapter 2: Power Conversion Control By Zero Sequence Harmonics

the dc output voltage given by

\[ V_2 = V_{k1} \sin \theta_p + V_{k3} \sin(3\theta_p + \phi_3), \]  

(2.10)

where \( V_{k1} \) and \( V_{k3} \) are the amplitudes of the fundamental and third harmonic inverter voltages, respectively, \( \phi_3 \) is the phase angle of the third harmonic relative to the fundamental and \( \theta_p \) is the phase angle where the drive voltage is at a maximum. The angle \( \theta_p \) is given by the extremum relation for the drive voltage

\[ \frac{dV_2}{d\theta_p} = V_{k1} \cos \theta_p + V_{k3} \cos(3\theta_p + \phi_3) = 0, \]  

(2.11)

which unfortunately is transcendental.

Figures 2.6 and 2.7 illustrates how the dc output voltage will vary with third harmonic amplitude and phase for a rectifier operating in discontinuous mode. One notices that for a third harmonic phase \( \phi_3 = \pi \), the dc output voltage is not only monotonic, but also linear with third harmonic voltage \( V_{k3} \); the reason for this is immediately obvious from Figure 2.5, as the peaks of the fundamental and third harmonic occur coincidentally. While not proven, it can be plausibly argued that while \( V_2 \) is monotonic with \( V_{k3} \) over various intervals for different values of \( \phi_3 \), linearity as well as the widest range of \( V_{k3} \) for monotonicity occurs only for \( \phi_3 = \pi \).

Figure 2.5: Drive Waveform with Third Harmonic
2.3.1 Voltage Output Control by Harmonic Amplitude Variation

The dc output $V_2$ of a grounded wye-connected rectifier can be controlled by varying the third harmonic voltage applied to the primary drive winding in Figure 1.1. For $\phi_3 = 0$, it can be exactly calculated (for a system that can be modeled as having no ac-side line inductance) that $V_2$ is monotonic with $|V_{k3}|$ for $|V_{k3}| > |V_{k1}|/6$ (at $|V_{k3}| = |V_{k1}|/6$, $|V_2| = \sqrt{3}|V_{k1}|/2$). The dependence of $V_2$ on $V_{k3}$ is plotted in Figure 2.7(a) for values of $\phi_3$ between 0 and $\pi$. At $\phi_3 = \pi$, $V_2$ is affine for $V_{k3} > 0$. At other value $\phi_3$ closed-form solutions to $V_2(V_{k3})$ are likely to not exist, but numerical methods can be used to estimate where this function is monotonic. Figure 2.7(b) shows that experimental data does indeed agree with calculations.

The strategy used for $\pi$-phase harmonic control is founded on the fact that the peaks of the 3rd harmonic and the fundamental coincide. In this case, where we have assumed discontinuous current in the phase connections of the secondary windings and 1:1 turns ratio, the dc output voltage will be very nearly equal to the sums of the peak voltages, $V_{k1} + V_{k3}$. In the implementation, a closed-loop PI controller ensures as $V_{k1}$ drops, which for example is the case when the speed is lowered, that the 3rd harmonic $V_{k3}$ makes up the difference.
Chapter 2: Power Conversion Control By Zero Sequence Harmonics

2.5 - 3
2 - 1.5
1 - 0.5
0

(a) MATLAB Calculations

(b) Experimental Results[5]

Figure 2.7: Peak amplitude control using third harmonic voltage amplitude ($V_3$) over a spread of phase ($\phi_3$) with fundamental voltage ($V_1$) held constant.
Section 2.3: Zero Sequence Harmonic Control in Three Phase Systems

Figure 2.8: PI Control of Dc Output Using Third Harmonic. The top trace shows $V_2$, the dc output of the grounded-wye secondary winding.
2.3.2 Zero Sequence Circuit

The zero sequence circuit is the portion of a multi-phase system where current can run in the same direction at the same time; it is that the part that is connected to one might call "common" to all the phases—with nomenclatures that include wye or neutral. Not all polyphase systems have a zero sequence path, often systems that do are called wye-grounded.

Zero Sequence Reactance in a Three Phase Circuit

A key issue in driving a zero sequence current through two magnetically coupled windings with grounded wyes is the effective magnetizing inductance, which is the phase-to-phase leakage in an induction machine, hence is typically kept as small as possible for good machine performance \[6\]. Figure 2.9 shows a zero sequence circuit model for an induction machine with two identical stators. Small magnetizing inductance \(M_0\) results in high zero sequence reactive current and represents additional loss in the stator resistance \(R_s\), as well as additional switch stress in the power electronics.

One method to increase the zero sequence reactance is to decrease the phase-to-phase coupling. Looking for the moment at the flux from phase \(a\) of a three-legged transformer,

\[
\lambda_a = L_c i_a - k_p \frac{L_c}{2} i_b - k_p \frac{L_c}{2} i_c, \quad (2.12)
\]

which implies that for a balanced set of currents with \(k_p\) now less than unity, each leg must now support a higher volt-seconds, hence resulting in a larger core.

From this transformer picture of the inductance matrix, it becomes more obvious that the
amount of flux coupled to the secondary winding solely depends on the coupling between the 
positive inductance windings \((L_c)\) being large (i.e. nearly unity \(k_0^3\)), yet with little coupling 
\((k_p)\) between adjacent phases (e.g. \(a\) and \(b\)). For a stator geometry with little saliency, which 
is mostly the case with induction motors, the coupling between phases tends to be typically 
high. When viewed as the single-circuit T-model illustrated in Figure 2.9, this results in a zero 
sequence magnetizing inductance that is deplorably small; a zero sequence transformer in series 
with the neutral current path alleviates this problem with the caveat that it is now the zero 
sequence transformer that must transfer essentially all the zero sequence power to the secondary 
side.

**Zero Sequence Transformer Topologies**

One solution to the problem of small magnetizing inductance is shown in Figure 2.10, where 
a zero sequence transformer with an acceptable magnetizing inductance is used in the wye 
connection; a turns ratio other than unity offers an additional degree of freedom in optimizing 
machine design through the scaling of the zero sequence voltage and current. This zero se-
quency transformer can be integrated into the machine back-iron, although this is not currently 
implemented.

![Zero Sequence Transformer](image)

*Figure 2.10: Zero sequence transformer.*

Power can be derived directly from the wye point as illustrated in Figure 2.11. In this 
topology, fewer rectifiers are required and a spit-capacitor ground is not needed; power to the 
output is derived solely from the zero sequence harmonics, so rectifier currents do not contribute 
to torque ripples, even without special accommodations in the control. Because the rectifiers 
only draw zero sequence current, standard field-oriented control schemes with fewer current 
sensors are more easily implemented.

\[ k_0 = M/L_c \] for identical stator windings.
Although using a direct zero sequence transformer offers a number of advantages, the trade-off is that all the dc output power is converted solely through a single-phase circuit, whereas the topology shown in Figure 2.10 allows, in certain regimes of operation, a portion of the power to be transferred to the output through the three phase circuit.

Under instances where the stator is not driven by an inverter, such as in an alternator, zero sequence harmonics can be exogenously driven through the wye as illustrated in Figure 2.12.
issue. In transformers where the primary is greater than some voltage threshold\(^4\) and where the secondary must be safe, the requirements for isolation (creepage and clearance) and insulation breakdown are strict and regulated by legislation.

While this section does not pretend to be a comprehensive treatise on the design of multi-use machines, nor does it suggest a design example, it attempts to offer some perspectives on the design limitations and advantages, which provides a motivation and some foundations for future work.

### 2.4.1 Fundamental Design Limitations

In the discussion of a machine design, there is an advantage to keeping the coupling between phases high, in very much the same way that a three-legged three phase transformer is better than three separate single phase transformers: the flux in each leg of the core has to carry only 1/2 of the flux than for each separate single phase transformer, resulting in a weight and volume savings for a given amount of apparent power.

### 2.4.2 Zero Sequence Control of Three-Phase, Three-Legged Transformer

### 2.4.3 Scaling Laws

The rationale for a multiple output transformer (i.e. \(1 : N : M : \ldots\)) as opposed to multiple single transformers (i.e. \(1 : N, 1 : M, \ldots\)) appears to be two-fold. As power conversion is combined into a single piece of magnetics, both weight and volume is lower than the aggregate of the single transformers; per unit power handling capability along with better overall window utilization.

#### Area Product

The power handling capability of a transformer is proportional to the area product \(A_p\) which is the product of the the window area \(W_a\) and the core cross-sectional area \(A_c\),

\[
A_p = W_a A_c. \tag{2.13}
\]

Both the weight and the volume of the transformer increase sub-linearly with \(A_p\) and hence also with power capability \([7]\),

\[
V \propto A_p^{0.75}. \tag{2.14}
\]

\(^4\)CE standards are 60V
Window Utilization

Isolation between primary and secondaries requires the use of a substantial amount of insulating tape between windings or the use of a triple-insulated wire\textsuperscript{5} to meet safety standards (e.g. CE and UL). With a lower power winding, the insulation consists of a higher percentage of the overall cross-sectional area of the wire. This results in a lower window utilization $K_u$ for lower power windings. A poignant example of this is illustrated in Figure 2.13, where the insulation is a significant fraction of the wire cross-section. Despite being a good fraction of the cross-section, the insulation is still much lighter than the enclosed copper so that the unit weight of the wire still increases pretty much linearly with its power handling capability.\textsuperscript{6}

Induction Machine

In any case, when the voltage amplitude of the third harmonic is less than that of the fundamental, there will be a fundamental component in the stator current and hence will contribute a torque ripple. However, one expects that a large fundamental drive voltage occur at high speed, where cogging is not as significant of an issue. At zero or low speed, rectifier current will be zero sequence. These have consequences in terms of per phase winding and core utilization.

\footnotetext[5]{Furukawa Tex-E\textsuperscript{®}.}

\footnotetext[6]{The power handling capability of a wire is related to its temperature rise for a given amount of current, hence is inversely proportional to unit resistance. Because of that, the amount of power that a wire can handle is proportional to cross-sectional area, or to the square of its diameter.}
Figure 2.13: Better utilization of the wire cross-section by the conductor is unambiguous for larger diameters in triple insulated wire.
Chapter 3

Vector Drive Control

The general concern of a variable speed drive is accurate and fast control of speed. However facetious as this sounds, it is not so straightforward of an endeavor in an induction machine. Speed and torque have no easy relation to voltage and terminal currents as on a dc machine. One must expend more effort to implement the classic speed control loop that has a minor loop for torque.

3.1 Constant Volts per Hertz Drive

In an induction machine with constant slip, flux is inversely proportional to frequency. For operation with constant flux, the ratio of voltage to frequency is held constant. At a given flux, the maximum speed attained when the equivalent back-EMF equals the available inverter voltage. At higher speeds, one must operate the machine at a constant voltage, while the flux decreases with speed: this is know as field-weakening or constant-power operation. The maximum available torque falls roughly in proportion to the inverse square of the frequency. [4]

This type of drive is typically good for VSDs that generally operate in the steady and where the best transient performance is not required (e.g. HVACs operating under relatively constant load). In a closed-loop speed control system, response and its complement, disturbance rejection, is ultimately determine by the controllability of torque and hence flux. During a transient, neither constant slip, nor any value of slip is guaranteed with a constant V/Hz drive. In addition, at low speeds and high torques, the implied low voltage and high current means that voltage drops across the stator resistance become a serious limitation for a drive system whose control variable is terminal voltage.

While the advantage of constant V/Hz operation is that it is simple to implement and does not require current sensors except perhaps for fault detection, it is most likely not good enough.
Chapter 3: Vector Drive Control

for traction and generation applications like EVs where speed and torque is both widely and strongly varying. Attempts at enhancing this method of speed controller include [9], but because of advancements in digital signal processors and subsequent price competitiveness, field oriented control methods have become more popular and accessible for high performance applications.

3.2 Indirect Field Oriented Control

Field oriented control takes advantage of the relation given by Equation 2.5

$$\tau_m = \frac{3}{2} p (\lambda_{qr} i_{dr} - \lambda_{dr} i_{qr})$$

to control the machine torque by specifying a flux and a current. If we set $\lambda_{qr} = 0$ and hold $\lambda_{dr} = \Lambda_0$ constant, then the torque is proportional to $i_{qr}$, which resembles how the torque relates to terminal current in dc machine.

\[ \lambda_{qr} = 0 \]  
\[ \lambda_{dr} = \Lambda_0 \quad \text{Constant}, \]  

so the torque in Equation 2.5 can be written as

$$\tau = \frac{3}{2} p \Lambda_0 i_{qr}. \quad (3.2)$$

$\lambda_{qr} = 0$ implies that $\lambda_{qr} = 0$ which results on the constraint in slip frequency

$$\omega_s = -\frac{r_r i_{qr}}{\lambda_{dr}} = \frac{r_r M}{L_{ar} \lambda_{dr} i_{qs}}. \quad (3.3)$$

The torque in stator coordinates is then given by

$$\tau = \frac{3}{2} p \frac{M}{L_{ar}} \Lambda_0 i_{qs}. \quad (3.4)$$

An estimator for $\lambda_{dr}$ can be derived from the first order differential equation

$$\dot{\lambda}_{dr} + \frac{r_r}{L_{ar}} \lambda_{dr} = \frac{r_r M}{L_{ar} i_{ds}}, \quad (3.5)$$

where $T_r = L_{ar}/r_r$ is referred to as the rotor time constant. $\lambda_{dr}$ can then be programmed by
Section 3.3: Cartesian Feedback in the Synchronous Current Regulator

$i_{qr}$ through a first-order transfer function,

$$\lambda_{dr} = \frac{M}{sT_r + 1}. \quad (3.6)$$

Field-oriented control methods require the control of the current in the stator that is magnetically linked to the rotor. In a multiple stator machine, this current measurement is corrupted by the additional loads presented by the secondary stators. This can be resolved by subtracting those load components from the primary stator currents to get the drive currents, which is described in §3.3. The field-oriented system illustrated in Figure 3.3 includes an implementation of a synchronous frame regulator. [10] gives a good discussion on tuning, stability and robustness of field-oriented controllers over parameter variations.

3.3 Cartesian Feedback in the Synchronous Current Regulator

The advantage of regulating current in the synchronous frame of reference is that the stator currents are represented as dc. In RF parlance, this is the consequence of mixing down the sinusoidal ac currents in the stator terminals to a dc baseband. In this synchronous reference frame, zero steady state error can be achieved by placing a pole at the origin in the controller, i.e. integral control. In the stator reference frame, the currents are sinusoidal and hence cannot have zero steady error for a proportional-integral controller. \(^1\)

By applying the conditions for field-oriented control in §3.2, the following state equations can be derived for the direct and quadrature stator currents from the state equations for $d\lambda_{ds}/dt$ and $d\lambda_{ds}/dt$:

$$-L_{as} \frac{di_{ds}}{dt} = r_s i_{ds} - \omega \left( L_{as} - \frac{M^2}{L_{ar}} \right) i_{qs} - v_{ds} \quad (3.7a)$$

$$- \left( L_{as} - \frac{M^2}{L_{ar}} \right) \frac{di_{qs}}{dt} = r_s i_{qs} + \omega L_{as} i_{ds} - v_{qs}. \quad (3.7b)$$

It is apparent from Equation 3.7a that there is a strong coupling term between the direct and quadrature axis currents that is proportional to the synchronous frequency.

A number of assumptions simplify the design of a controller for the Aardvark induction machine. A key assumption in designing the synchronous current controller is that the electrical

\(^1\)See Roberge[11] for a discussion on the error series derivation, which is germane to the tracking error of a proportional-integral controller to a sinusoid.
time constants of the machine are much smaller than the mechanical time constants. This is important because it allows us to satisfy the condition that the bandwidth of a minor loop be higher than the outer loop crossover frequency.

3.4 Integrator Anti-Windup

There are two output limits in any real inverter: current limit and voltage saturation (i.e. compliance of the current source). The maximum current that ought to be allowed depends on the physical limits of the power devices and the load. Voltage saturation is the result of a finite dc bus voltage, hence limiting the time rate of changes in flux \((d\lambda/dt)\). In the short-time scale, this is due to the time rate of change in current \((di/dt)\), and on the longer time scale (or steady state) by winding resistances (stator and rotor) and to the time rate of change of mutual inductance, which is proportional to the speed \(\omega\). Abstractly, by the product rule

\[
v = \frac{d\lambda}{dt} = L\frac{di}{dt} + i\frac{dL}{dt}.
\]  

Voltage saturation level is actually subtle: when one wants an inverter output with as few harmonics as possible, saturation occurs when the peak amplitude of the sine wave fundamental equals the one-half of the dc bus voltage for the half-bridge inverter (the full dc bus voltage for a full-bridge inverter); however, if over-modulation is allowed, the fundamental amplitude can be as high as \(4/\pi\) times larger by applying 100% duty cycle for 50% of the time, i.e. a symmetric square wave.

In terms of \(dq\)-axis quantities the current limit

\[
is_{max}^2 \leq i_{ds}^2 + i_{qs}^2.
\]  

The voltage saturation limit in this \(dq\) space

\[
v_{s,\max}^2 \leq v_{ds}^2 + v_{qs}^2.
\]  

From a control perspective, either limit presents itself as a classic actuator saturation. In a controller that integrates the error between the command (or reference) and the output, this error accumulates causing a large overshoot even when the actuator comes out of saturation and the setpoint been reached. That which is not a classical about this situation is the limitation of the magnitude of a vector of control variables (a MIMO system); a further complication is that the integrator in the controller is designed with an integrator for each variable so that there
will be zero error in the steady state.

From Equation 3.4, $i_{ds}$ programs the flux in the machine and $i_{qs}$ programs the torque. Typically, the flux is programmed to some optimal value below the rated motor speed; above this speed, the flux is decreased in inverse proportion to the speed, resulting in a constant power operating regime. If we assume that the speed changes at a much lower rate than the torque, the dynamics to consider in the controller design are that of the torque while that of the speed over the relevant time scale is invariant.

The condition for voltage saturation in Equation 3.10 allows for one degree of freedom, which we choose to be $v_d$. In the polar coordinate frame, the contour is described by

$$\theta = \cos^{-1} \frac{v_{ds}}{v_{s,max}}.$$  \hspace{1cm} (3.11)

This allows $i_{ds}$ to still be programmed through $v_{ds}$ during saturation as illustrated in Figure 3.1.

In arranging the saturation conditions this way, we maintain the dc motor analogue, where the torque is constrained while the flux remains a free variable.\(^2\)

The speed is also controlled by a PI controller, but it has SISO (single input, single output) dynamics, with an LTI function of the error commanding a torque, which we assume to be proportional to stator quadrature current $i_{qs}$. The field, or flux is proportional to $i_{ds}$ which value is determined by a field-weakening function of speed that we presume to have no dynamics.\(^3\)

We would like to saturate the output of the controller for any of several reasons: a current limit given by Equation 3.9, a mechanical damage torque limit, and an electrical torque limit which depends on the flux.\(^4\) The signaling of either these limits results in a relatively straightforward anti-windup strategy, such as limiting the speed-control integrator.

The current limit is the result of a number of factors. As already mentioned, these include a hard current limit to prevent physical damage and current source compliance due to a finite inverter voltage. Because the speed controller is much slower than the current controller, its integrator winds up at a much lower rate. We would like to signal a saturation from the current controller to the speed-control integrator only for longer time scale saturation events due to such things as demanding more torque than what is within the limits of the setting for the flux. If the speed controller with its field-weakening algorithm and torque limits were ideal, these longer time scale voltage saturation events would not occur; however, time-varying parameters

\(^2\)In a speed controller with field weakening, flux is a non-linear function of speed.

\(^3\)Only perhaps presumptuous in that the flux dynamics have a time scale in the neighborhood of the rotor time constant ($T_r$ given in Equation 3.5), which we assume to be much smaller than the mechanical time constants.

\(^4\)The electrical torque limits can be precalculated from the flux.
Figure 3.1: Contour of the saturation limit for the synchronous current regulator and the allowed trajectory in dq-space. There is a maximum torque limit that can also be described on the dq axis, but ought to be considered as part of the speed controller and not included in this diagram.
such as stator and rotor resistances, as well as nonlinearities in the iron permeability may well cause the speed controller to ask for more than the current regulator can provide.

The question is how does one determine the cause of the voltage saturation. Recall Equation 3.8. One way to do that is to keep track of the magnitude of the time derivative of the current \( di/dt \) during the voltage saturation. If

\[
\left\| \frac{d}{dt} \begin{bmatrix} i_{ds} \\ i_{qs} \end{bmatrix} \right\|_2 \leq \epsilon
\]

where \( \epsilon \) is some threshold while the voltage is still saturated, then current controller signals a saturation event to the speed controller.

### 3.5 Simulation

<table>
<thead>
<tr>
<th>Stator Resistance</th>
<th>( R_a )</th>
<th>2.0Ω</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rotor Resistance</td>
<td>( R_2 )</td>
<td>1.5Ω</td>
</tr>
<tr>
<td>Stator Reactance(^\dagger)</td>
<td>( X_1 )</td>
<td>2.8Ω</td>
</tr>
<tr>
<td>Rotor Reactance(^\dagger)</td>
<td>( X_2 )</td>
<td>2.8Ω</td>
</tr>
<tr>
<td>Mutual Reactance(^\dagger)</td>
<td>( X_M )</td>
<td>42.09Ω</td>
</tr>
<tr>
<td>Free Moment of Inertia</td>
<td>( J_0 )</td>
<td>0.0168 kg \cdot m^2</td>
</tr>
</tbody>
</table>

The electrical parameters for the Aardvark induction machine are listed in Table 3.1. These parameters were derived by J. Holloway in [2] from a non-linear least square fit to the start up transient of the induction machine using IEEE blocked-rotor and no-load tests as well as impedance measurements for values for the initial guess.

The parameters in Table 3.1 along with an indirect field oriented controller and the strategies for anti-windup form the basis for a Simulink\(^\circledR\) model and simulation. Figure 3.2 predicts good transient performance under step speed reversals and steps in torque load. A diagram of the simulation can be found in Appendix C.

\(^\dagger\)Reactances are customarily referenced to 60 Hz.
Figure 3.2: Simulation of induction motor under field-oriented control with speed and torque load steps.
3.6 Implementation

The controller illustrated in Figure 3.3 is currently being implemented with minimal current sensing as described in §4.4.2. There were a number of issues that precluded the inclusion of results in this thesis. These included a number of hardware issues that included slow and erratic behavior in the opto-coupler circuit for the speed sensor and incorrect gains in the current sensing circuits. In the firmware, timing miscues in the field oriented control routine caused incorrect updates to the PWM routine.

A new opto-coupler circuit, as well as current sensing circuitry have been designed and tested, but not yet integrated into the motor controller. Field-oriented control firmware is currently being debugged and results are forthcoming.
Figure 3.3: Implementation of a field-Oriented controller for a multi-use induction machine. The secondary dc output is regulated by varying the 3rd or other zero sequence harmonic in the PWM inverter. In this diagram, the turns ratios between the stator windings and the ratios in the zero sequence transformer are assumed to be 1:1. Other turns ratios are possible by proper scaling when subtracting the transformed stator winding currents.
Chapter 4

Inverter Design

4.1 Power Module and Digital Signal Processor

A half-bridge inverter was designed around International Rectifier’s PIIPM15P12D007 programmable isolated integrated power module. The power module contains the power electronics (e.g. IGBTs, gate drives, etc.), ac mains rectifiers, as well as a TI TMS320LF2406A DSP for digital control of the motor drive and for any zero sequence control of dc output voltages.

The combined power electronic and control platform used in this thesis is shown in Figure 4.1. The PIIPM15P12D007 from International Rectifier (IR) combines all the necessary power electronics (i.e. IGBTs, gate drives, and protection) with a TMS320LF2406A control-optimized DSP from Texas Instruments, along with associated sensors, peripherals, auxiliary supplies, and communications (e.g. RS485, JTAG, CAN). Although this platform has been discontinued by IR, it is close to an ideal model for the development of digital motor control systems.

4.2 Sine Wave Generation

4.2.1 Synchronous PWM

In variable speed drive, the inverter fundamental frequency varies with speed; in a field-oriented controller, it also varies with torque. When using a constant PWM frequency, non-integer ratios between the PWM frequency and the fundamental result in subharmonic content as a result of the “beating”. Synchronous PWM was achieved by varying the PWM switching frequency about a nominal (e.g. 10 kHz) so that the switching frequency is always a multiple \( n \) of the generated sine wave; an algorithm for hysteresis about the transition points of \( n \) was included.
Chapter 4: Inverter Design

(a) PIIPM Integrated Power Module

(b) PIIPM Schematic

(c) PIIPM Block Diagram

Figure 4.1: International Rectifier's Integrated Power Module[12]
Section 4.2: Sine Wave Generation

to eliminate switching frequency jitter and oscillation. A good discussion of synchronous PWM can be found in [13].

4.2.2 Table-Based Implementation

A sine wave drive with low spurious harmonic content is important for rectifier output voltage control. The three-phase inverter is based on a 108-element sine reference table that drives a symmetric PWM whose output is illustrated by the MATLAB plot in Figure 4.2. The size of the table was chosen to be both a multiple of 3 (aligned three-phase system) and 4 (quarter-wave symmetry). This results in a THD (total harmonic distortion) of 1.71% and a maximum error of 2.90%.

Table 4.1: THD and Maximum For a DLT Sine Reference Using Different Interpolating Functions

<table>
<thead>
<tr>
<th>Interpolating Function</th>
<th>THD (%)</th>
<th>Maximum Error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ceiling()</td>
<td>3.37</td>
<td>5.81</td>
</tr>
<tr>
<td>floor()</td>
<td>3.37</td>
<td>5.81</td>
</tr>
<tr>
<td>round()</td>
<td>1.71</td>
<td>2.90</td>
</tr>
</tbody>
</table>

Although not implemented, an equivalent 27-element quarter-wave table could be used. The generation of the third harmonic is achieved by accessing every third table entry during each PWM update. A key to the generation of a sine wave with low harmonic content is the alignment of the PWM switching instances with the table element entries, which ensures synchronous PWM. A discussion of table-based implementations are presented in [14]. Figure 4.4 shows no harmonic content to at least 1.25 kHz with a 60 Hz fundamental and a third harmonic amplitude that is 50% of the fundamental. The algorithm is simple computationally because it performs only a direct table lookup and does not require interpolation. With this algorithm, the resolution for third harmonic phase modulation is determined by the size of the table. If a better resolution is required without the penalty of a large table size, interpolation for only the third harmonic lookup is required.

4.2.3 Parabolic Approximations

Real-time, on-the-fly second order approximations are a good alternative to look-up table based implementations. Angular resolution is limited only by the working precision of the desired number type. A second-order approximation requires at most three multiplications and three
Chapter 4: Inverter Design

Figure 4.2: Sine reference created from a 108-element direct-lookup table.

Figure 4.3: THD and maximum error versus table length N for a sine reference using a direct lookup table.
Section 4.2: Sine Wave Generation

Figure 4.4: Inverter output voltage.

Figure 4.5: Inverter Current for Fundamental + 25% Third Harmonic
additions; a half-wave approximation requires one compare while a quarter-wave approximation requires at most three compares, if one assumes the argument to the approximating function is already limited to principal values, i.e. \([0, 2\pi]\). Over the approximating interval (e.g. \([0, \pi]\) for a half wave),

\[ S(\theta) = x_2(\theta - x_1)^2 + x_0 \]  

(4.1)

A number of authors have used parabolic approximations for DFSS (direct-digital frequency synthesis) in communications, but have used either a three-point fit (zero crossing and peak with no errors) [15], least-squares fitting [16], or Taylor series approximations [16, 17]. In addition, 4th order approximations based on doubly iterated parabolic approximations have been proposed [15]. In the proceeding sections, we will see that choosing the appropriate metric for the fitting optimization gives a more appropriate result for an intended application. Only a second order approximation is shown, but a higher order iterated parabolic approximation with the appropriate metric can easily be implemented.

The coefficients of the approximation are chosen in some optimal way:

- Maximizing the fundamental.
- Minimizing harmonic distortion.
- Minimizing percentage error.
- Zero error at the endpoints.
- Zero error at the peak.
- Zero error at the zero crossings.
- \(L^1\) optimal.
- \(L^2\) optimal, which minimizes the mean square error.
- \(L^\infty\) optimal, which minimizes peak error.
- And so forth ...

In general, these conditions do not result in the same coefficients and are sometimes conflicting. As a second-order approximation, only three degrees of freedom are available.

As a sine reference for an inverter, values of the approximating function must match at the endpoints of the sub-intervals; this does not necessarily mean that there must be zero error at
Section 4.2: Sine Wave Generation

these points. However, enforcing zero error at the zero crossings minimizes crossover distortion and prevents an ambiguity that could lead to a systematic dc offset.

In the case of a half-wave approximation, enforcing zero error at the zero crossings leaves only one degree of freedom for any other optimization, whereas in the quarter-wave approximation, two degrees of freedom are still available, which in reviewing Tables 4.2 and 4.3, result in better optimization figures of merit.

In the calculation of the Park’s transform, it seems reasonable that the sine approximation be $L^\infty$ optimal, which results in the smallest peak error for the calculation of d-axis and q-axis quantities, while the sine reference for the inverter may use coefficients that reduce total harmonic distortion. Ultimately, it is a multi-parameter design optimization in the design of the inverter where proper weighting of such things as torque ripple and efficiency in the machine, controller stability, among many others, must be taken into consideration.

Minimizing total harmonic distortion is not equivalent to maximizing the fundamental, which is equivalent to minimizing the objective function

$$\mathcal{G}(x_0, x_1, x_2) = \left| \int_0^{2\pi/n} \sin^2 \theta' \, d\theta' - \int_0^{2\pi/n} [x_2(\theta' - x_1) + x_0] \sin \theta' \, d\theta' \right|$$

(4.2)

$$= \left| \int_0^{2\pi/n} \mathcal{S}(\theta') \sin(\theta') \, d\theta' \right|$$

(4.3)

where $n$ is the number of sub-intervals and the error

$$\mathcal{S}(\theta) = S(\theta) - \sin \theta.$$  

(4.4)

Total harmonic distortion (THD) when the dc term is zero, is defined as the ratio of rms value of the harmonics in the waveform to the rms value of the fundamental,

$$\text{THD} = \sqrt{2 \left( \frac{1}{2\pi/n} \int_0^{2\pi/n} \mathcal{S}^2(\theta') \, d\theta' \right) - 1}$$

(4.5)

where $a_1$ is the fundamental Fourier coefficient,

$$a_1 = \frac{n}{\pi} \int_0^{2\pi/n} S(\theta') \sin \theta' \, d\theta'.$$

~ 55 ~
Chapter 4: Inverter Design

Optimizing for total harmonic distortion is equivalent to minimizing

\[ \mathcal{G}(x_0, x_1, x_2) = \frac{\int_0^{2\pi/n} S^2(\theta') \, d\theta'}{a_1^2}, \]  

(4.6)

which is proportional to the reciprocal of the square of the distortion factor. The distortion factor is the ratio of the rms of the fundamental to the rms of the waveform.

The \( L^\infty \) norm to minimize becomes

\[ \mathcal{G}(x_0, x_1, x_2) = \| \cdot \|_\infty = \sup \{ |\mathcal{E}(\theta)| : \theta \in [0, 2\pi/n] \}, \]  

(4.7)

and the \( L^2 \) norm, or least-squares objective

\[ \mathcal{G}(x_0, x_1, x_2) = \| \cdot \|_2^2 = \int_0^{2\pi/n} \sigma^2(\theta) \, d\theta'. \]  

(4.8)

Half-Wave Approximation

A half-wave approximation is an optimal second-order fit to each of two sub-intervals: \([0, \pi]\) and \([\pi, 2\pi]\). The error over the approximating interval are illustrated in Figure 4.6 for a fit to a unit amplitude sine wave. Table 4.2 was calculated in MATLAB using a uniform grid of 10,000 points using both constrained and unconstrained non-linear optimizations. The optimization for THD appeared sensitive to the initial conditions, which indicates that the minimum might be relatively flat, or that multiple local minima exist. For these calculations, the initial conditions for all the optimizations are the case for zero error at the peak and at the zero crossings.

Quarter-Wave Approximation

The quarter-wave approximation is an optimal second order fit to a sine wave over the interval \([0, \pi/2]\). Table 4.3 and Figure 4.7 shows that this approximation results in better values in comparison to the half-wave case for both THD and error, respectively. These calculations were performed with 100,000 points over the quarter-wave interval. In the unconstrained case, it seems that a lower THD was achieved by fitting using least squares than for fitting by directly optimizing THD, possibly similar reasons that the THD fit was sensitive to initial conditions in the half-wave case.

The quarter-wave approximation with an \( L^\infty \)-optimal fit results in slightly better THD and much better maximum error figures (1.67% and 1.65%, respectively) than in the 108-element
Section 4.2: Sine Wave Generation

Figure 4.6: Error from parabolic half-wave approximations to a sine function with unit amplitude. $L^1(-)$; $L^2(\cdots)$; $L^\infty (-\cdot)$; Maximal Fundamental (---); Minimum THD (---); Zero Error for Peak and Zero Crossing (---).

\[\text{\copyright 57 \copyright}\]


Chapter 4: Inverter Design

Table 4.2: Coefficients for Half-Wave Sine Approximations (10,000 point discretization for all calculations).

<table>
<thead>
<tr>
<th>Goal</th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$\phi$</th>
<th>THD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L^1$</td>
<td>0.9851</td>
<td>1.5708</td>
<td>-0.4270</td>
<td>143</td>
<td>2.84</td>
</tr>
<tr>
<td>$L^2$</td>
<td>0.9802</td>
<td>1.5708</td>
<td>-0.4177</td>
<td>2.9829</td>
<td>2.64</td>
</tr>
<tr>
<td>$L^\infty$</td>
<td>0.9719</td>
<td>1.5708</td>
<td>-0.4051</td>
<td>0.0281</td>
<td>2.95</td>
</tr>
<tr>
<td>Max. Fundamental</td>
<td>0.9831</td>
<td>1.6286</td>
<td>-0.4199</td>
<td>8.26 x 10^-5</td>
<td>6.77</td>
</tr>
<tr>
<td>Min. Distortion</td>
<td>0.9924</td>
<td>1.5708</td>
<td>-0.4229</td>
<td>0.0238</td>
<td>2.64</td>
</tr>
<tr>
<td>Zero Crossing Error</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Zero Error Peak</td>
<td>1.  \pi/2</td>
<td>-4/\pi^2</td>
<td></td>
<td></td>
<td>3.93</td>
</tr>
<tr>
<td>$L^1$</td>
<td>0.9721</td>
<td>1.5708</td>
<td>-0.3940</td>
<td>240.30</td>
<td>3.93</td>
</tr>
<tr>
<td>$L^2$</td>
<td>0.9675</td>
<td>1.5708</td>
<td>-0.3921</td>
<td>7.2249</td>
<td>3.93</td>
</tr>
<tr>
<td>$L^\infty$</td>
<td>0.9618</td>
<td>1.5708</td>
<td>-0.3898</td>
<td>0.0382</td>
<td>3.93</td>
</tr>
<tr>
<td>Max. Fundamental</td>
<td>0.9689</td>
<td>1.5708</td>
<td>-0.3927</td>
<td>0.0131</td>
<td>3.93</td>
</tr>
<tr>
<td>Min. Distortion</td>
<td>1.0000</td>
<td>1.5708</td>
<td>-0.4053</td>
<td>0.0641</td>
<td>3.93</td>
</tr>
</tbody>
</table>

direct table lookup case (1.71% and 2.91%), in the case where there is no error at the zero crossings.

In addition to being useful as a sine reference for the inverter, a quarter wave approximation forms the basis for the cos$^{-1}$ function that is necessary to calculate the phase output\(^1\) of the synchronous current regulator.

\subsection*{4.3 Three Phase Filters with Four Legs}

A number of issues arise from currents and voltages at the PWM frequency (10-30 kHz). Among these include capacitive currents which can cause wear in bearings and cause inadvertent ground loops that cause additional inverter loading and create additional difficulties in sensor measurement. The PWM waveform is rich with harmonics, which makes electromagnetic compatibility another issue even with relatively PWM switching frequencies. A number of authors have have tried to analyze\([18]\) and mitigate\([19]\) these issues.

Three-phase filters are used to reduce the PWM frequency content from the inverter in driving the stator. Although these filters represent additional cost and component count, it reduces losses in both the stator windings and the core in addition to eliminating resonances.

\(^1\)Equation 3.11.

~ 58 ~
Figure 4.7: Error from parabolic quarter-wave approximations to a sine function with unit amplitude. $L^1(-)$; $L^2(\cdots)$; $L^\infty (-)$; Maximal Fundamental (---); Minimum THD(---); Zero Error for Peak and Zero Crossing (----).
Table 4.3: Coefficients for Quarter-Wave Sine Approximations. (10,000 point uniform discretization for all calculations).

<table>
<thead>
<tr>
<th>Goal</th>
<th>$x_0$</th>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$f$</th>
<th>THD (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>$L^1$</td>
<td>1.0341</td>
<td>1.7596</td>
<td>-0.3445</td>
<td>34.63</td>
<td>1.27</td>
</tr>
<tr>
<td>$L^2$</td>
<td>1.0325</td>
<td>1.7676</td>
<td>-0.3382</td>
<td>0.3518</td>
<td>1.19</td>
</tr>
<tr>
<td>$L^\infty$</td>
<td>1.0273</td>
<td>1.7723</td>
<td>-0.3315</td>
<td>0.0139</td>
<td>1.32</td>
</tr>
<tr>
<td>Max. Fundamental</td>
<td>0.9948</td>
<td>1.7676</td>
<td>-0.4133</td>
<td>8.76 $\times 10^{-5}$</td>
<td>1.32</td>
</tr>
<tr>
<td>Min. Distortion</td>
<td>1.0453</td>
<td>1.5708</td>
<td>-0.3424</td>
<td>0.0492</td>
<td>1.32</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Zero Crossing Error</th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Zero Error Peak</td>
<td>1</td>
<td>$\pi/2$</td>
<td>$-4/\pi^2$</td>
<td></td>
<td>3.8</td>
</tr>
<tr>
<td>$L^1$</td>
<td>1.0612</td>
<td>1.8730</td>
<td>-0.3025</td>
<td>50.95</td>
<td>1.66</td>
</tr>
<tr>
<td>$L^2$</td>
<td>1.0524</td>
<td>1.8563</td>
<td>-0.3054</td>
<td>0.6804</td>
<td>1.65</td>
</tr>
<tr>
<td>$L^\infty$</td>
<td>1.0378</td>
<td>1.8261</td>
<td>-0.3112</td>
<td>0.0175</td>
<td>1.67</td>
</tr>
<tr>
<td>Max. Fundamental</td>
<td>0.9699</td>
<td>1.5744</td>
<td>-0.3913</td>
<td>4.28 $\times 10^{-5}$</td>
<td>3.81</td>
</tr>
<tr>
<td>Min. Distortion</td>
<td>1.0455</td>
<td>1.8564</td>
<td>-0.3034</td>
<td>0.0272</td>
<td>1.65</td>
</tr>
</tbody>
</table>

due to parasitics and inductance nonlinearities from high frequency effects, which can couple changes in zero sequence current to $dq$-axis flux.

There are a number of ways to do three phase filters; among the most convenient is to use series inductors and shunt capacitors between the inverter output and the motor. Figure 4.8 shows such a filter with the capacitors connected across the line in a \( \Delta \) configuration which similar to that used in [20, 21]. This configuration filters only line-to-line currents and is useful in circuits where the wye is ungrounded, or where one wants the neutral to have a low impedance. It is apparent from the filter response that the attenuation on the line with frequency is 2\textsuperscript{nd} order, but only a 1\textsuperscript{st} order $LR$ response between the resistive load and series inductance for the the zero sequence path. The single-phase, off-line analogy is using differential inductors with $X$-capacitors (line-to-line).

In a multi-use application, both line and neutral currents require filtering. The filter in Figure 4.9 attenuates high frequencies from both phase and zero sequence components identically. This topology suffers if there are mismatches in the line-to-neutral capacitors by unbalancing the response, hence converting some of the differential current to common mode, which is reminiscent of what happens with $Y$-capacitors in single-phase, off-line applications.

The improved filter illustrated in Figure 4.10 is more complicated because it uses a coupled inductor along with series inductors and shunt capacitors. This circuit is a novel three-phase, four-wire derivation of the coupled inductor filters in [22, 23, 24] In this topology, the zero
Figure 4.8: Line-to-line filter with delta-connected capacitors do not filter zero sequence components, but provide additional inductance in the zero sequence path. Filter response with Y-connected 10Ω resistive load. (—) Line Response; (—__) Zero Sequence Response.(LTSpice)
Figure 4.9: Typical three phase filter for both phase-to-phase and zero sequence components. Zero sequence may be introduced by mismatches in filter components. Filter response with Y-connected 10Ω resistive load. (--) Line Response; (---) Zero Sequence Response. (LTSpice)
sequence components are filtered separately from the positive and negative sequence components. Because the coupled inductor is connected similarly to a common-mode choke, the net flux in the core is only that of the zero sequence current if the leakages between each leg is small enough. This topology is necessarily better than using just a common-mode choke because power is a significant amount of power is expected to be carried on the neutral (i.e. zero sequence), which benefits by having a filter with a 2nd order response; this is quite different from the single-phase, off-line case where one tries to minimize the current in the ground wire.

There are a number of considerations in the design of these three-phase filters from the perspective of doing current control (§3.3): ignoring the filter by placing the filter breakpoints well above the current-loop crossover frequency, or including the filter in the plant dynamics. In either case, the fact that filter damping depends on the resistance seen at the stator drive terminal (i.e. rotor effective resistance and secondary stator rectifier load) becomes an issue at light loads. Typically, the series resistance of the inductor and the stator provide a bound on the damping. If additional damping is required, small series resistances can be added to the capacitor, and if the degradation in the filter response by doing this is not acceptable, explicit dampening legs can be placed in parallel to the capacitors.

4.4 Minimal Implementations for Phase Current Measurement

For field-oriented drive control, only the positive and negative sequence (or equivalently, the d-axis and q-axis) currents need to be measured because zero sequence current does not contribute to torque. Zero sequence current measurement, however, may be useful for detecting overcurrent conditions in the power conversion circuit.

4.4.1 Balanced Three Phase, Wye Grounded and Ungrounded

In a balanced three-phase line,

\[ i_a + i_b + i_c = 0 \]  \hspace{1cm} (4.9)

In this case, there are only two independent variables (e.g. \( i_a, i_b \)) to be measured. The third can be calculated, \( i_c = -(i_a + i_b) \).

When the the wye is ungrounded, there can be no zero sequence current, which is the
Figure 4.10: Three phase filter with additional coupled inductor stage for zero sequence components. Filter response with Y-connected 10Ω resistive load. (--) Line Response; (-----) Zero Sequence Response. (LTSpice)
simplest balanced three-phase case for current measurement.

\[ i_a = I \cos(\omega t) \]
\[ i_b = I \cos(\omega t - 2\pi/3) \]
\[ i_c = I \cos(\omega t + 2\pi/3) \]

When the wye is grounded, but the currents are still balanced even though there is a zero sequence current, i.e.

\[ i'_a = \frac{i_0}{3} + i_a = \frac{i_0}{3} + I \cos(\omega t) \]
\[ i'_b = \frac{i_0}{3} + i_b = \frac{i_0}{3} + I \cos(\omega t + 2\pi/3) \]
\[ i'_c = \frac{i_0}{3} + i_c = \frac{i_0}{3} + I \cos(\omega t - \pi/3) \]

Equation 4.9 still holds, and only two current sensors are needed to recover the sequence currents. This can be accomplished by subtracting the \(1/3\) of the zero sequence current from each of the phases on each of the sensors.

### 4.4.2 Multiple Stator

In a machine with multiple stators, a field oriented controller requires access to the component of the phase currents in the stator that links flux to the rotor. While a detailed model of the leakage inductances and coupling coefficients is required for the exact currents, these parameters are generally time-varying and nonlinear because of the effects of temperature and magnetic saturation.

Several approximations are appropriate in the Aardvark induction machine. Because the stator primary and secondaries are wound in-hand, one can assume that these windings are well-coupled. This means that the current in the magnetizing inductances is small relative to the overall phase current. The remaining phase current then consists of the currents from the rotor and the secondaries reflected back to the primary.

In a machine with unity turns ratio between the primary and secondaries and with good coupling in the stator-stator windings, as it is in the Aardvark machine, the phase currents reflect back to the primary. In this way, the correct linear combination of current measurements from the different stator windings recovers the rotor-linked flux current as Figures 4.11 and 4.12 illustrates. The recovery of the linked current in these figures is calculated based on the assumption that there is good coupling of the zero sequence, which is the case since a zero
Figure 4.11: Illustration of how the phase current that links flux to the rotor \( (I_{rec}) \) can be recovered from a linear combination of current measurements from different stator windings. \( I_{a1} \) is the current phase \( a \) of the primary, \( I_{n1} \) is from the neutral of the primary, and \( I_{a2} \) is that of phase \( a \) of the secondary.

sequence transformer was part of the circuit\(^2\). The recovered linked current is

\[
I_{rec} = I_{a1} - I_{a2} - \frac{2}{3}I_{n1}.
\]

4.4.3 Estimation from Inverter Current Out of the DC Bus

A common objection to the idea of using an induction motor as a multi-use machine is that such a large number of current sensors are required. In a "normal" machine, the phase currents are balanced and there is no zero sequence current, hence only two current sensors are required for field-oriented control. With a multi-use machine, one expects that at best two current sensors are required for each addition stator winding.

The idea of using a single current sensor on the dc link along with the already available knowledge of the switching states has been demonstrated in [25, 26]. By extending this idea from the inverter dc link to the output dc buses, only one additional current sensor is required

\(^2\)See Figure 2.10.
Figure 4.12: The FFT of the currents used in the recovery of current that links flux to the rotor. The dc components were subtracted and a Hanning window was used.

for each secondary stator winding, which will probably be necessary anyway for short-circuit and overload sensing and protection.
Chapter 5

Firmware Design

The DSP programme is multi-tasking and real-time. The typical approach is to use a real-time operating system (RTOS). In this design, I have used a program architecture that does not use an RTOS, but is multi-tasking and real-time.

The architecture uses a round-robin approach within a superloop with preemptive tasks handled by interrupts. Although events are handled within the super loop, tasks do not run to completion, but rather, are time-sliced. In addition, a number of persistent data structures are shared publicly among the tasks and are not explicitly protected by an operating system. Semaphores are included as part of these data structures as means for mutual exclusion, but each task is responsible for checking and setting these semaphores.

This architecture is advantageous for small programs where both the intellectual and programmatic overhead of an RTOS is undesired. A certain coding discipline is required as is understanding the timing requirements of particular routines. In any case, multi-tasking, real-time firmware is not for the unwary.

5.1 *Time Slicing Algorithm*

The task functions in the main loop are time triggered from a hardware counter. As each function is called in the main loop, it checks the counter to determine whether it should be in an active state or a wait state. When the function, or more precisely the task, becomes active, one iteration is performed. Actually, the function may actually iterate several times, it depends on just how the task is defined. This task iteration runs to completion and only can be preempted by an interrupt. For this scheme to work, the task iteration must complete within the timing limits set by the timing budget, which must be decided at design-time. In addition, interrupts add to the iteration time, hence a timing margin for each task is needed.
for the worst-case interrupt scenario, which includes context saving and loading along with the actual service routine. It goes without saying that there can only be few interrupts and that the interrupt service routines (ISR) must be short. There are a few ways to ensure this, for example the ISR can either complete the task quickly or set a flag to wake a normal task. In this architecture, it is important to note that tasks are created at design-time and cannot be spawned.

Table 5.1 shows the execution times and approximate periods for each of the tasks. `update_PWM()` and `controlV3()` are normal task functions while `periodic_ISR()` is an ISR that completes its task quickly.

One may notice that the periodicity for the example functions in Table 5.1 to be rather slow. Let’s examine the slowest, `control.V3()`. Because the output voltage is controlled on the peaks of a three-phase waveform, in which 60 Hz is the highest frequency, the fastest that one can actuate a control is three times 60 Hz or 180 Hz, which also true for π-phase 3rd harmonic control. The most periodic is `periodic_ISR()`, which updates the PWM duty cycle from a 108-element sine reference table. At the fastest which is 60 Hz, the update rate is 1/108th of a 60 Hz period which is 154 μs.

<table>
<thead>
<tr>
<th>Task</th>
<th>Execution</th>
<th>Period</th>
</tr>
</thead>
<tbody>
<tr>
<td>update_PWM()</td>
<td>111 μs</td>
<td>500 μs</td>
</tr>
<tr>
<td>periodic_ISR</td>
<td>8 μs</td>
<td>154 μs</td>
</tr>
<tr>
<td>controlV3()</td>
<td>117 μs</td>
<td>1 ms</td>
</tr>
</tbody>
</table>

### 5.2 Data Structures

As currently implemented, several persistent data structures are shared among different tasks. For example `sin_pwm` contains information about the switching frequency, fundamental and harmonic frequencies and amplitudes, among others; `VFRamping` is used for volts per hertz ramping and contains information about the voltage and frequency steps of the drive, as well as the ramp rate.

While it may be considered memory intensive to have too many persistent data structures, in the currently limited use case for this application, all the tasks remain active, so it seems appropriate in many cases to use these persistent data structures rather than maintain a `mailbox`, or some other means of data passing.
Section 5.3: Output Voltage Regulation Module

Hardware registers and other peripherals are generally abstracted from the general tasks by a data structure such as *sin_pwm*. Each of these data structures maintained by a separate task such as *update_PWM()* that updates the peripherals and lower level drivers and ensures that there are no collisions during these updates.

5.3 Output Voltage Regulation Module

The algorithm for the regulation of dc rectifier output using the π-phase 3rd harmonic is illustrated in the UML\(^1\) activity diagram in Figure 5.1. This diagram describes what happens during a single time slice.

The maximum amplitude for the 3rd harmonic voltage is determined by the available voltage headroom, i.e. 100% PWM duty cycle at the peak of the inverter output, which is a combination of 1st and 3rd harmonics.

5.4 Synchronous PWM Module

Updates to the PWM module occur all at once. During an update all tasks are mutually excluded from the PWM data structure. *update_PWM()* has the job of brokering the transfer of PWM parameters from the higher level tasks to the interrupt service routine as well as making all the low-level calculations. Every task gets a chance to update *sin_pwm* because of the end position in the round-robin queue of *update_PWM*. *update_PWM()* checks a semaphore to see if the *periodic_interrupt_isr()* is being handled, and if not, excludes the this interrupt service routine and updates the parameters.

Synchronous PWM is implemented so that the switching frequency is an integral multiple of the highest zero sequence harmonic and hence also the fundamental. The actual multiple used to determine the switching frequency changes as upper and lower boundaries. To prevent oscillations near the boundaries, hysteresis is included in the algorithm.

5.5 Synchronous Current Controller

The main function for the synchronous current controller is illustrated in the activity diagram in Figure 5.2. The synchronous current controller keeps track of the voltage and current limits and calls appropriate handlers during voltage saturation and soft overcurrents.

\(^1\)UML—Universal Modeling Language, although it isn’t claimed that these diagrams are compliant to the most recent standards.
Chapter 5: Firmware Design

get $V_1$ get $V_{dcr\_ref}$

\{ $V_3\_max < V_{max} - V_1$ \}

calculate $V_{dcr\_error}$

$V_{dcr} = \text{dc rectifier output from secondary winding.}$

$\text{Accum} = \text{Accum} + V_{dcr\_error}$

$V_3 = K_p \cdot V_{dcr\_error} + K_i \cdot \text{Accum}$

$V_3\_\text{out} = V_3$

$V_3\_\text{min} \text{ can be zero or } -18\% \text{ of } V_{max}...$

Figure 5.1: Activity diagram for closed-loop control of dc output using a proportion-integral controller with accumulator anti-windup along with overflow and underflow control.

\sim 72 \sim
Section 5.5: Synchronous Current Controller

Speed Controller ensures that $i^*$ and $i_q^*$ are within torque and current limits.

- $i_q^* \rightarrow$ Torque
- $i_d^* \rightarrow$ Flux

get $i_d^*$, $i_q^*$

get Stator_Currents

calculate $i_d$, $i_q$, $i_{mag}$

- $[i_{mag}^2 > i_{max}^2]$ => call soft_overcurrent_handler()
- $[i_{mag}^2 < i_{min}^2]$ => Accum_iq = Accum_iq_min, Accum_iq = Accum_iq_max

calculate id_error, iq_error

- Accum_id = Accum_id + id_error
- Accum_iq = Accum_iq + iq_error

- $V_d = Kd*p*i_d\_error + Kd*i_d$ Accum_iq

- $V_q = Kq*p*i_q\_error + Kq*i_q$ Accum_iq

- $V_{mag}^2 = V_{d}^2 + V_{q}^2$

- $V_{mag}^2 > V_{max}^2$ => call saturation_handler()

- $V_{mag}^2 < V_{min}^2$ => $V_{mag} = V_{max}$

$\psi = \arccos(V_{d}/V_{mag})$, $V_{mag} = V_{mag}$

Figure 5.2: Activity diagram for control of stator current in the synchronous frame using a proportional-integral controller with accumulator anti-windup along with overflow and underflow control.
Chapter 6

Conclusions and Future Work

A number of challenges were surmounted which resulted in a number of innovations. An enabling concept has been the use of zero sequence harmonics where the peak of the fundamental coincides with a peak of the harmonic to control the dc rectifier output of a secondary on the stator. This concept has been demonstrated in a 1.5 hp induction machine using a closed-loop, proportional-integral controller to control a π-phase third harmonic so that a dc output can be regulated. Another enabling concept has been the use of a zero sequence transformer, which allows the efficient transfer of zero sequence harmonics to the secondary.

To realize the system, an inverter and drive control had to be developed. The design of the inverter was crucial in that it had produce an accurate sine wave and superposing it with any other harmonic without overwhelming the digital signal processor. Two implementations were carefully analyzed: a table-based sine reference and a parabolic approximation to pieces of the sine. Ultimately, the parabolic approximation appears superior to a stored table that can easily fit into the DSP. In the future, it may be possible to implement a 4th-order approximation using a doubly-iterated parabolic approximation, which in the literature appears to have excellent performance.

Two drive control methods have been developed. The volts per hertz drive has been successfully implemented while the indirect field-oriented controller has been shown to work in simulation, but with results of the implementation forthcoming. A number of practical and theoretical issues arise with the field-oriented controller. These include anti-windup, cross-coupling of the dq-axis variables in the plant and the implementation of current regulator. The synchronous current regulator for a multi-use machine is different than what is commonly used in field-oriented controllers in that it must supply a drive current that links flux to the rotor, but must also provide a voltage with which to regulate a dc rectifier output. In a sense, it must be both a current and a voltage source. Such a thing is possible using feedback so that
Chapter 6: Conclusions and Future Work

the fundamental is controlled as a current loop and the zero sequence harmonics controlled by a voltage loop. To achieve this, the inputs are conceptually the frequencies and voltage amplitudes of the fundamental and harmonics, along with a phase, which are essentially polar coordinates, as opposed to the Cartesian $d$- and $q$ axis variables.

Included is a preliminary analysis of what might be possible with the idea of multi-use machine, by quoting some rough scaling laws.

To achieve this, a number of analysis and simulation tools have been developed, in MATLAB, Simulink and SPICE. Some concepts, some of which are novel, have been reframed into appropriate contexts.

Future results include implementation results from the field-oriented controller along with a careful analysis of an example design for a multi-use machine, so that better comparisons with competing technologies can be made.
Appendix A

SPICE Deck for Multistator Transformer

A.1 PSPICE—Wye-Ungrounded

* Schematics Netlist *

L_L4 $N_0001$N_0002 {Lc/2}
L_L6 $N_0003$N_0004 {Lc/2}
L_L5 vz0 $N_0002$ {Lc/2}
L_L8 $N_0003$ vz0 {Lc/2}
L_L7 $N_0005$N_0006 {Lc/2}
L_L14 $N_0007$N_0008 {Lc/2}
L_L17 $N_0009$N_0010 {Lc/2}
L_L15 $N_0007$ vz1 {Lc/2}
L_L13 $N_0011$N_0012 {Lc/2}
L_L18 vz1 $N_0012$ {Lc/2}
L_L16 vz1 $N_0010$ {Lc/2}
L_L1 $N_0013$N_0001 {Llk}
L_L2 $N_0014$N_0004 {Llk}
L_L3 $N_0015$N_0005 {Llk}
L_L10 $N_0011$N_0016 {Llk}
L_L11 $N_0008$N_0017 {Llk}
L_L12 $N_0009$N_0018 {Llk}
V_Vb $N_0019$N_0020
+sin 0 {Vki} {f1} 0 0 240
V_Vc $N_0021$N_0020
+sin 0 {Vki} {f1} 0 0 120
V_Vn $N_0020$ 0
+sin 0 {Vki} {f3} 0 0 0
D_DB $N_0022$ v01 Dbreak
Figure A.1: Wye-Ungrounded
Section A.1 : PSPICE-Wye-Ungrounded

D_D9 $N_0023 v01 Dbreak
D_D10 $N_0024 v01 Dbreak
D_D11 v02 $N_0022 Dbreak
D_D12 v02 $N_0023 Dbreak
D_D13 v02 $N_0024 Dbreak
Kn_K1 L_L4 L_L6
+ L_L13 L_L14 {kp}
Kn_K4 L_L5 L_L7
+ L_L17 L_L18 {kp}
Kn_K7 L_L8 L_L9
+ L_L15 L_L16 {kp}
R_R1 $N_0025 $N_0013 1
R_R2 $N_0019 $N_0014 1
R_R3 $N_0021 $N_0015 1
R_R4 $N_0016 $N_0022 1
R_R5 $N_0017 $N_0023 1
R_R6 $N_0018 $N_0024 1
V_Va $N_0025 $N_0020
+SIN 0 {Vk1} {f1} 0 0 0
C_C1 v00 v01 1000uF
C_C2 v02 v00 1000uF
R_R7 v00 v01 200
R_R8 v02 v00 200
R_R11 v00 vz1 100MEG
R_R9 0 vz1 1
L_L9 vz0 $N_0006 {Lc/2}

.PARAM f1=60 f3=180
.PARAM kp=1 k0=1
.PARAM Lc=100mH Llk=100uH
.PARAM Vk1=300 Vk3=0.001

** Analysis setup **
.tran Ons 2
.OPTIONS ABSTOL=10pA
.OPTIONS RELTOL=0.005
.OPTIONS VNTOL=10uV
.OP

* From [PSPICE NETLIST] section of pspiceev.ini:
.lib "nom.lib"

~ 79 ~
Appendix A: SPICE Deck for Multistator Transformer

.INC "three-legged transformer4.net"
.INC "three-legged transformer4.als"

.probe

.END

A.2 PSPICE—Wye-Grounded

* Schematics Netlist *

L_L4 $N_0001 $N_0002 \{Lc/2\}
L_L6 $N_0003 $N_0004 \{Lc/2\}
L_L5 0 $N_0002 \{Lc/2\}
L_L8 $N_0003 0 \{Lc/2\}
L_L7 $N_0005 $N_0006 \{Lc/2\}
L_L9 0 $N_0006 \{Lc/2\}
L_L14 $N_0007 $N_0008 \{Lc/2\}
L_L17 $N_0009 $N_0010 \{Lc/2\}
L_L15 $N_0007 vzi \{Lc/2\}
L_L13 $N_0011 $N_0012 \{Lc/2\}
L_L18 vzi $N_0012 \{Lc/2\}
L_L16 vzi $N_0010 \{Lc/2\}
L_L1 $N_0013 $N_0001 \{Llk\}
L_L2 $N_0014 $N_0004 \{Llk\}
L_L3 $N_0015 $N_0005 \{Llk\}
L_L10 $N_0011 $N_0016 \{Llk\}
L_L11 $N_0008 $N_0017 \{Llk\}
L_L12 $N_0009 $N_0018 \{Llk\}
C_C1 v00 v01 1000uF
C_C2 v02 v00 1000uF
R_R8 v02 v00 200
V_Vb $N_0019 $N_0020
+\text{SIN} 0 \{V_{k1}\} \{f1\} 0 0 240
V_Vc $N_0021 $N_0020
+\text{SIN} 0 \{V_{k1}\} \{f1\} 0 0 120
V_Vn $N_0020 0
+\text{SIN} 0 \{V_{k3}\} \{f3\} 0 0 0
D_D8 $N_0022 v01 Dbreak
D_D9 $N_0023 v01 Dbreak
D_D10 $N_0024 v01 Dbreak

~ 80 ~
Figure A.2: Wye-Grounded
Appendix A: SPICE Deck for Multistator Transformer

D_D11    v02 $N_0022 Dbreak
D_D12    v02 $N_0023 Dbreak
D_D13    v02 $N_0024 Dbreak
Kn_K1    L_L4 L_L6
+ L_L13 L_L14  {kp}
Kn_K4    L_L5 L_L7
+ L_L17 L_L18  {kp}
Kn_K7    L_L8 L_L9
+ L_L15 L_L16  {kp}
R_R1    $N_0025 $N_0013 1
R_R2    $N_0019 $N_0014 1
R_R3    $N_0021 $N_0015 1
R_R4    $N_0016 $N_0022 1
R_R5    $N_0017 $N_0023 1
R_R6    $N_0018 $N_0024 1
V_Va    $N_0025 $N_0020
+SIN 0 {Vk1} {f1} 0 0 0
R_R7    v00 v01 200
R_R9    0 vz1 1
R_R10   vz1 v00 0.001

.PARAM Lc=100mH Llk=100uH
.PARAM f1=60 f3=180
.PARAM kp=1 k0=1
.PARAM Vk1=300 Vk3=0.001

** Analysis setup **
.tran Ons 2
.OP

* From [PSPICE NETLIST] section of pspiceev.ini:
.lib "nom.lib"

.INC "three-legged transformer2.net"
.INC "three-legged transformer2.als"

.probe

.END
Section A.3 : PSPICE–Wye-Grounded with Zero Sequence Transformer

### A.3 PSPICE–Wye-Grounded with Zero Sequence Transformer

* Schematics Netlist *

```plaintext
L_L4   vpa $N_0001  \{Lc/2\}
L_L6   $N_0002 vpb  \{Lc/2\}
L_L5   vz0  $N_0001  \{Lc/2\}
L_L8   $N_0002 vz0  \{Lc/2\}
L_L14  $N_0003 vsb  \{Lc/2\}
L_L15  $N_0003 vz1  \{Lc/2\}
L_L13  vsa $N_0004  \{Lc/2\}
L_L1   $N_0005 vpa  \{Llk\}
L_L2   $N_0006 vpb  \{Llk\}
L_L3   $N_0007 vpc  \{Llk\}
L_L10  vsa $N_0008  \{Llk\}
L_L11  vsb $N_0009  \{Llk\}
L_L12  vsc $N_0010  \{Llk\}
C_C1   v00 v01 1000uF
C_C2   v02 v00 1000uF
R_R8   v02 v00 200
D_D11  v02 $N_0011 Dbreak
D_D12  v02 $N_0012 Dbreak
D_D13  v02 $N_0013 Dbreak
Kn_K1  L_L4 L_L6
+ L_L13 L_L14 \{kp\}
Kn_K4   L_L5 L_L7
+ L_L17 L_L18 \{kp\}
Kn_K7   L_L8 L_L9
+ L_L15 L_L16 \{kp\}
L_L17  vsc $N_0014  \{Lc/2\}
L_L7   vpc $N_0015  \{Lc/2\}
L_L18  vz1 $N_0004  \{Lc/2\}
R_R7   v00 v01 200
D_D8   $N_0011 v01 Dbreak
D_D10  $N_0013 v01 Dbreak
D_D9   $N_0012 v01 Dbreak
R_R1   $N_0016 $N_0005 1
R_R2   $N_0017 $N_0006 1
R_R3   $N_0018 $N_0007 1
R_R6   $N_0010 $N_0013 1
R_R5   $N_0009 $N_0012 1
```

~ 83 ~
Figure A.3: Wye-Grounded with Zero Sequence Transformer
Section A.3 : PSPICE-Wye-Grounded with Zero Sequence Transformer

R_R4 $N_0008 \ $N_0011 \ 1
R_R9 0 v00 1
L_L16 vzi $N_0014 \ {Lc}/2
L_L9 vzo $N_0015 \ {Lc}/2
K_TX2 L1_TX2 L2_TX2 1
L1_TX2 $N_0019 \ 0 \ 100mH
L2_TX2 $N_0020 \ v00 \ 100mH
R_R10 $N_0019 \ vz0 \ 1
V_Va $N_0016 \ $N_0021
+\text{SIN} \ 0 \ \{Vk\} \ 60 \ 0 \ 0 \ 0
V_Vb $N_0017 \ $N_0021
+\text{SIN} \ 0 \ \{Vk\} \ 60 \ 0 \ 0 \ -120
V_Vc $N_0018 \ $N_0021
+\text{SIN} \ 0 \ \{Vk\} \ 60 \ 0 \ 0 \ +120
V_Vn $N_0021 \ 0
+\text{SIN} \ 0 \ \{Vk\} \ 180 \ 0 \ 0 \ 180
L_L20 vzi $N_0020 \ 1mH

.PARAM \ f1=60 \ f3=180
.PARAM \ kp=1 \ kO=1
.PARAM \ Lc=100mH \ Llk=1mH
.PARAM \ Vk1=0.1 \ Vk3=299.9

** Analysis setup **
 tran Ons 2
 OP

* From [PSPICE NETLIST] section of pspiceev.ini:
  .lib "nom.lib"

  .INC "three-legged transformer3.net"
  .INC "three-legged transformer3.als"

.probe

.END
Appendix B

MATLAB Script for Parabolic Approximations of Sine Function

B.1 Script

```matlab
%%M-file to generate plots and coefficients for various 2nd order sine approximations

clear;
npts = 10000;
tmesh = 0:pi/npts:pi;
x0 = [-4/pi^2 pi/2 1]';
str = cell(11,11);

%L1 fit
k=1;
[y, f] = fminsearch(@(x) hlfit(x,tmesh), x0)
[r, a1, df] = thd(y,tmesh,pi);
str(k,1) = {'L1-Fit'}
for i = 2:4
    str(k,i) = {y(i-1)};
end
str(k,5) = {'b-'};
str(k,6) = {f};
str(k,7) = {4*mindistfit(y,tmesh)/npts}; %half wave = 4/npts; qtr=8/npts
str(k,8) = {r};
str(k,9) = {a1};
str(k,10) = {df};

%L2 fit
k=k+1;
str(k,1) = {'L2-Fit'}
[y, f] = fminsearch(@(x) hlfit(x,tmesh), x0)
[r, a1, df] = thd(y,tmesh);
for i = 2:4
```

~ 87 ~
Appendix B: MATLAB Script for Parabolic Approximations of Sine Function

```matlab
29  str(k,i) = {y(i-1)};
end
31  str(k,5) = {'g:'};
    str(k,6) = {f};
33  str(k,7) = {4*mindistfit(y,tmesh)/npts}; \% half wave = \(4/npts\); \(qtr=8/npts\)
    str(k,8) = {r};
35  str(k,9) = {al};
    str(k,10) = {df};
37  \%
39  Linf fit
41  k = k + 1;
42  str(k,1) = {'Linf-Fit'}
    [y,f] = fminsearch(@(x) hinffit(x,tmesh),x0)
43  [r,al,df] = thd(y,tmesh);
    for i = 2:4
44      str(k,i) = {y(i-1)};
end
47  str(k,5) = {'r-'};
    str(k,6) = {f};
49  str(k,7) = {4*mindistfit(y,tmesh)/npts}; \% half wave = \(4/npts\); \(qtr=8/npts\)
    str(k,8) = {r};
51  str(k,9) = {al};
    str(k,10) = {df};
53  \%
55  Min Distortion Fit
57  k = k + 1;
58  str(k,1) = {'Min-Distortion-Fit'}
    [y,f] = fminsearch(@(x) mindistfit(x,tmesh),x0)
59  [r,al,df] = thd(y,tmesh);
    for i = 2:4
61      str(k,i) = {f(i-1)};
end
63  str(k,5) = {'c-'};
    str(k,6) = {f};
65  str(k,7) = {4*mindistfit(y,tmesh)/npts}; \% half wave = \(4/npts\); \(qtr=8/npts\)
    str(k,8) = {r};
67  str(k,9) = {al};
    str(k,10) = {df};
69  \%
71  Min Distortion Fit2
73  k = k + 1;
    str(k,1) = {'Min-Dist-Fit2'}
75  x00 = [-0.4177 1.5708 0.9802]; \% Use L2 guess for initial fit
    [y,f] = fminsearch(@(x) thdopt(x,tmesh),x00)
77  [r,al,df] = thd(y,tmesh);
    for i = 2:4
```

~ 88 ~
77\text{\textbackslash{}texttt{str(k, i) = \{} y(i-1) \};}}$

\textbf{end}\texttt{str(k, 5) = \{} 'k-\';}}$

\texttt{str(k, 6) = \{} f \};}}$

\texttt{str(k, 7) = \{}4*\text{mindistfit}(y, \text{tmesh})/npts \}; \% half wave = 4/npts; qtr=8/npts}}$

\texttt{str(k, 8) = \{} r \};}}$

\texttt{str(k, 9) = \{} a1 \};}}$

\texttt{str(k, 10) = \{} df \};}}$

\texttt{str(k, 11)=\{} 'k+' \};}}$

\%Zero pk and crossing error

\texttt{k=k+1;}}$

\texttt{str(k, 1) = \{} 'Pk\_and\_Zero\_Cross' \}}$

\texttt{y = x0;}}$

\texttt{[r, a1, df] = \text{thd}(y, \text{tmesh});}}$

\texttt{for i = 2:4}}$

\texttt{str(k, i) = \{} y(i-1) \};}}$

\texttt{end}}$

\texttt{str(k, 5) = \{} 'b-' \};}}$

\texttt{str(k, 6) = \{} f \};}}$

\texttt{str(k, 7) = \{}4*\text{mindistfit}(y, \text{tmesh})/npts \}; \% half wave = 4/npts; qtr=8/npts}}$

\texttt{str(k, 8) = \{} r \};}}$

\texttt{str(k, 9) = \{} a1 \};}}$

\texttt{str(k, 10) = \{} df \};}}$

\texttt{str(k, 11) = \{} 'bo' \};}}$

\%L1 fit, zero crossing error

\texttt{k=k+1;}}$

\texttt{str(k, 1) = \{} 'L1\_Fit\_Zero\_Cross' \}}$

\texttt{options = \text{optimset}('\text{LargeScale}', 'off');}}$

\texttt{[y, f] = \text{fmincon}(\@s(x) \text{hfit(x,tmesh)})x0,[],[],[],[],[],[],[]... \@confuneq(options)}}$

\texttt{[c, ceq] = \text{confuneq}(y)}}$

\texttt{[r, a1, df] = \text{thd}(y, \text{tmesh});}}$

\texttt{for i = 2:4}}$

\texttt{str(k, i) = \{} y(i-1) \};}}$

\texttt{end}}$

\texttt{str(k, 5) = \{} 'g-' \};}}$

\texttt{str(k, 6) = \{} f \};}}$

\texttt{str(k, 7) = \{}4*\text{mindistfit}(y, \text{tmesh})/npts \}; \% half wave = 4/npts; qtr=8/npts}}$

\texttt{str(k, 8) = \{} r \};}}$

\texttt{str(k, 9) = \{} a1 \};}}$

\texttt{str(k, 10) = \{} df \};}}$

\%L2 fit, zero crossing error

\texttt{k=k+1;}}$

\texttt{str(k, 1) = \{} 'L2\_Fit\_Zero\_Cross' \}}$

\texttt{options = \text{optimset}('\text{LargeScale}', 'off');}}
Appendix B: MATLAB Script for Parabolic Approximations of Sine Function

```matlab
[y, f] = fmincon(@x h2fit(x, tmesh), x0, [], [], [], [], [], [], ... @confuneq, options)
[c, ceq] = confuneq(y)
[r, al, df] = thd(y, tmesh);
for i = 2:4
    str(k,i) = {y(i-1)};
end
str(k,5) = {'r:'};
str(k,6) = {f};
str(k,7) = {4*mindistfit(y, tmesh)/npts};  % half wave = 4/npts; qtr=8/npts
str(k,8) = {r};
str(k,9) = {al};
str(k,10) = {df};

% Linf fit, zero crossing error
k=k+1;
str(k,1) = {'Linf Fit Zero Cross'}
options = optimset('LargeScale', 'off');
[y, f] = fmincon(@x hinffit(x, tmesh), x0, [], [], [], [], [], [], ... @confuneq, options)
[c, ceq] = confuneq(y)
[r, al, df] = thd(y, tmesh);
for i = 2:4
    str(k,i) = {y(i-1)};
end
str(k,5) = {'c-'};
str(k,6) = {f};
str(k,7) = {4*mindistfit(y, tmesh)/npts};  % half wave = 4/npts; qtr=8/npts
str(k,8) = {r};
str(k,9) = {al};
str(k,10) = {df};

% Max fundamental, zero crossing error
k=k+1;
str(k,1) = {'Min Distort Z Cross'}
options = optimset('LargeScale', 'off');
[y, f] = fmincon(@x mindistfit(x, tmesh), x0, [], [], [], [], [], [], ... @confuneq, options)
[c, ceq] = confuneq(y)
[r, al, df] = thd(y, tmesh);
for i = 2:4
    str(k,i) = {y(i-1)};
end
str(k,5) = {'m-'};
str(k,6) = {f};
str(k,7) = {4*mindistfit(y, tmesh)/npts};  % half wave = 4/npts; qtr=8/npts
str(k,8) = {r};
str(k,9) = {al};
```

~ 90 ~
\begin{verbatim}
173 str(k,10) = {df};
175 \%Min distortion fit 2, zero crossing error
176 k=k+1;
177 str(k,1) = {'Min_Distort2_ZCross'};
178 x00 = [-0.3921 1.5708 0.9675];
179 options = optimset('LargeScale','off');
180 \quad [y,f] = fmincon(@(x) thdopt(x,tmesh),x00,[],[],[],[],[],[],...
181 \quad \quad confuneq,options)
182 \quad [c,ceq] = confuneq(y)
183 \quad [r,al,df] = thd(y,tmesh);
184 \quad for i = 2:4
185 \quad \quad str(k,i) = {y(i-1)};
186 end
187 str(k,5) = {'k-'};
188 str(k,6) = {f};
189 str(k,7) = {4*mindistfit(y,tmesh)/npts}; \%half wave = 4/npts; qtr=8/npts
190 str(k,8) = {r};
191 str(k,9) = {al};
192 str(k,10) = {df};
193 str(k,11) = {'k+'};
195 \% Sa = y(1)*((tmesh - y(2)*ones(size(tmesh))).^2 + y(3)*ones(size(tmesh));
196 \% error = Sa - sin(tmesh);
198 \% figure(1)
199 \quad plot(tmesh/pi,error);
200 \% figure(2)
201 \quad plot(tmesh/pi,sin(tmesh),tmesh/pi,Sa);
202 clear i,y; y = zeros(1,3);
203 figure(1);
204 for i = 1:6
205 \quad for j = 1:3;
206 \quad \quad y(j,1) = str{i,j+1}
207 \quad end
208 \quad Sa = y(1)*((tmesh - y(2)*ones(size(tmesh))).^2 + ...
209 \quad \quad y(3)*ones(size(tmesh));
210 \quad error = Sa - sin(tmesh);
211 plot(tmesh,error,str{1,5});
212 hold on;
213 \quad if isa(str{i,11},'char')
214 \quad \quad plot(downsampel(tmesh,(npts/10)),...
215 \quad \quad \quad downsample(error,(npts/10)),str{i,11});
216 \quad end
217 end
219 \%set(gca,'XTick',[0 pi/4 pi/2 3*pi/4 pi]);
220 \%set(gca,'XTickLabel','tgx0|tgxpi4|tgxpi2|tgx3pi4|tgxpi');
221 \%set(gca,'YTick',[-0.15 -0.1 0.05 0 0.05 0.1]);
223 \%set(gca,'YTickLabel',...
Appendix B: MATLAB Script for Parabolic Approximations of Sine Function

```matlab
221    % 'tgym0p15|tgym0p1|tgym0p05|tgym0p05|tgym0p1'
222    xlabel('tgxxtheta')
223    ylabel('tgyyerror')
224    hold off;
225    figure(2);
226    for i = 6:11
227        for j = 1:3;
228            y(j,1) = str{i,j+1}
229        end
230        Sa = y(1)*(tmesh - y(2)*ones(size(tmesh))).^2 + ...
231            y(3)*ones(size(tmesh));
232        error = Sa - sin(tmesh);
233        plot(tmesh, error, str{i,5});
234        hold on;
235        if isa(str{i,11}, 'char')
236            plot(downsample(tmesh,(npts/10)),...
237                downsample(error,(npts/10)),str{i,11});
238        end
239    end
240    % set(gca, 'XTick',[0 pi/4 pi/2 3*pi/4 pi]);
241    % set(gca, 'XTickLabel','0\pi/4 | \pi/2|3\pi/4 | \pi');
242    % xlabel('\theta (radians)')
243    % set(gca, 'YTick',[-0.15 -0.1 0.05 0 0.05 0.1]);
244    % set(gca, 'YTickLabel','tgym0p15|tgym0p1|tgym0p05|tgym0p05|tgym0p1');
245    xlabel('tgxxtheta')
246    ylabel('tgyyerror')
247    hold off;
248
249    %Quarter Wave Approximation
250    tmesh = 0:pi/2/npts:pi/2;
251    x0 = [-4/pi^2 pi/2 1]';
252    str2 = cell(11,10);
253    %L1 fit
254    k=1;
255    [y,f] = fminsearch(@(x) h1fit(x,tmesh),x0)
256        [r,al,df] = thd(y,tmesh);
257    str2(k,1) = {'L1_Fit'}
258        for i = 2:4
259            str2(k,i) = {y(i-1)};
260        end
261    str2(k,5) = {'b-'};
262    str2(k,6) = {f};
```

~ 92 ~
str2(k,7) = \{8*\text{mindistfit}(y,tmesh)/npts\}; \%half wave = \frac{4}{npts}; qtr=8/npts
str2(k,8) = \{r\};
str2(k,9) = \{a1\};
str2(k,10) = \{df\};

\%L2 fit
k=k+1;
[y,f] = \text{fminsearch}(@(x) \text{h2fit}(x,tmesh),x0)
[r,a1,df] = \text{thd}(y,tmesh);
str2(k,1) = \{'L2 Fit\'}
\textbf{for } i = 2:4
\hspace{1em}str2(k,i) = \{y(i-1)\};
\textbf{end}
str2(k,5) = \{'g:\'};
str2(k,6) = \{f\};
str2(k,7) = \{8*\text{mindistfit}(y,tmesh)/npts\}; \%half wave = \frac{4}{npts}; qtr=8/npts
str2(k,8) = \{r\};
str2(k,9) = \{a1\};
str2(k,10) = \{df\};

\%LinF fit
k=k+1;
[y,f] = \text{fminsearch}(@(x) \text{hinfit}(x,tmesh),x0)
[r,a1,df] = \text{thd}(y,tmesh);
str2(k,1) = \{'LinF Fit\'}
\textbf{for } i = 2:4
\hspace{1em}str2(k,i) = \{y(i-1)\};
\textbf{end}
str2(k,5) = \{'r--\'};
str2(k,6) = \{f\};
str2(k,7) = \{8*\text{mindistfit}(y,tmesh)/npts\}; \%half wave = \frac{4}{npts}; qtr=8/npts
str2(k,8) = \{r\};
str2(k,9) = \{a1\};
str2(k,10) = \{df\};

\%Max Fundamental Fit
k=k+1;
[y,f] = \text{fminsearch}(@(x) \text{mindistfit}(x,tmesh),x0)
str2(k,1) = \{'Min-Dist Fit\'}
\textbf{for } i = 2:4
\hspace{1em}str2(k,i) = \{y(i-1)\};
\textbf{end}
str2(k,5) = \{'c--\'};
str2(k,6) = \{f\};
str2(k,7) = \{8*\text{mindistfit}(y,tmesh)/npts\}; \%half wave = \frac{4}{npts}; qtr=8/npts
str2(k,8) = \{r\};
str2(k,9) = \{a1\};
str2(k,10) = \{df\};
%Min Distortion Fit 2
k=k+1;
x00 = [-0.3882 1.7676 1.0325];
y, f = fminsearch(@(x) thdopt(x, tmesh), x00)
str2(k,1) = {'Min_Dist_Fit2'}
for i = 2:4
    str2(k,i) = {y(i-1)};
end
str2(k,5) = {'k-'};
str2(k,6) = {f};
str2(k,7) = {8*mindistfit(y,tmesh)/npts}; %half wave = 4/npts; qtr=8/npts
str2(k,8) = {r};
str2(k,9) = {a1};
str2(k,10) = {df};
str2(k,11) = {'k+'};

%Zero pk and crossing error
k=k+1;
y = x0;
[r, a1, df] = thd(y, tmesh);
str2(k,1) = {'Zero_pk_and_cross'}
for i = 2:4
    str2(k,i) = {y(i-1)};
end
str2(k,5) = {'g-'};
str2(k,6) = {f};
str2(k,7) = {8*mindistfit(y,tmesh)/npts}; %half wave = 4/npts; qtr=8/npts
str2(k,8) = {r};
str2(k,9) = {a1};
str2(k,10) = {df};
str2(k,11) = {'bo'};

%L1 fit, zero crossing error
k=k+1;
options = optimset('LargeScale','off');
y, f = fmincon(@(x) h1fit(x, tmesh), x00, [], [], [], [], [], [], [], @confuneq2, options)
[c, ceq] = confuneq2(y)
[r, a1, df] = thd(y, tmesh);
str2(k,1) = {'L1 Fit Zero.Cross '}
for i = 2:4
    str2(k,i) = {y(i-1)};
end
str2(k,5) = {'g-'};
str2(k,6) = {f};
str2(k,7) = {8*mindistfit(y,tmesh)/npts}; %half wave = 4/npts; qtr=8/npts
str2(k,8) = {r};
Section B.1: Script

\[ \text{str2(k,9)} = \{a1\}; \]
\[ \text{str2(k,10)} = \{df\}; \]

\%L2 fit, zero crossing error
\n\text{k=k+1;}
\text{options = optimset('LargeScale','off');}
\text{[y,f] = fmincon(@(x) h2fit(x,tmesh),x0,[],[],[],[],[],[],[],[],...}
\text{@confuneq2,options)}
\text{[c,ceq] = confuneq2(y)}
\text{[r,al,df] = thd(y,tmesh);}
\text{str2(k,1) = 'L2-Fit-Z-Cross'}
\text{for i = 2:4}
\text{str2(k,i) = \{y(i-1)\};}
\text{end}
\text{str2(k,5) = '\text{r}:';}\]
\text{str2(k,6) = \{f\};}
\text{str2(k,7) = \{8*mindistfit(y,tmesh)/npts\}; \%half wave = 4/npts; qtr=8/npts}
\text{str2(k,8) = \{r\};}
\text{str2(k,9) = \{a1\};}
\text{str2(k,10) = \{df\};}

\%Linf fit, zero crossing error
\n\text{k=k+1;}
\text{options = optimset('LargeScale','off');}
\text{[y,f] = fmincon(@(x) hinffit(x,tmesh),x0,[],[],[],[],[],[],[],...}
\text{@confuneq2,options)}
\text{[c,ceq] = confuneq2(y)}
\text{[r,al,df] = thd(y,tmesh);}
\text{str2(k,1) = 'Linf-Fit-Z-Cross'}
\text{for i = 2:4}
\text{str2(k,i) = \{y(i-1)\};}
\text{end}
\text{str2(k,5) = \{'c-.';}\]
\text{str2(k,6) = \{f\};}
\text{str2(k,7) = \{8*mindistfit(y,tmesh)/npts\}; \%half wave = 4/npts; qtr=8/npts}
\text{str2(k,8) = \{r\};}
\text{str2(k,9) = \{a1\};}
\text{str2(k,10) = \{df\};}

\%Max fundamental, zero crossing error
\n\text{k=k+1;}
\text{options = optimset('LargeScale','off');}
\text{[y,f] = fmincon(@(x) mindistfit(x,tmesh),x0,[],[],[],[],[],[],[],...}
\text{@confuneq2,options)}
\text{[c,ceq] = confuneq2(y)}
\text{[r,al,df] = thd(y,tmesh);}
\text{str2(k,1) = 'Min-Dist-Z-Cross'}
\text{for i = 2:4}
\text{str2(k,i) = \{y(i-1)\};}
\text{end}
Appendix B: MATLAB Script for Parabolic Approximations of Sine Function

str2(k,i) = {y(i-1)};
end
str2(k,5) = {'m--'};
str2(k,6) = {f};
str2(k,7) = {8*mindistfit(y,tmesh)/npts}; % half wave = 4/npts; qtr=8/npts
str2(k,8) = {r};
str2(k,9) = {a1};
str2(k,10) = {df};

% Min distortion fit 2, zero crossing error
k=k+1;
x00 = [-0.3054 1.8563 1.0524];
options = optimset('LargeScale','off');
[y,f] = fmincon(@(x) thdopt(x,tmesh),x0,[],[],[],[],[],[],[],options);
[c,ceq] = confuneq2(y);
[r,a1,df] = thd(y,tmesh);
str2(k,1) = {'Min_Dist2_ZCross'}
for i = 2:4
    str2(k,i) = {y(i-1)};
end
str2(k,5) = {'k--'};
str2(k,6) = {f};
str2(k,7) = {8*mindistfit(y,tmesh)/npts}; % half wave = 4/npts; qtr=8/npts
str2(k,8) = {r};
str2(k,9) = {a1};
str2(k,10) = {df};
str2(k,11) = {'k+'};

figure(3);
for i = 1:6
    for j = 1:3;
        y(j,1) = str2{i,j+1}
    end
    Sa = y(1)*(tmesh - y(2)*ones(size(tmesh))).^2 + ...
        y(3)*ones(size(tmesh));
    error = Sa - sin(tmesh);
    plot(tmesh,error,str2{i,5});
    hold on;
    if isa(str2{i,11},'char')
        plot(downsample(tmesh,(npts/10)),...
             downsample(error,(npts/10)),str2{i,11});
    end
end
% set(gca,'XTick',[0 pi/4 pi/2 3*pi/4 pi]);
% set(gca,'XTickLabel','0| pi/4| pi/2| 3pi/4| pi ');
% xlabel('theta (radians)')
% ylabel('Error (Unit Amplitude)')
Section B.2: Functions

B.2.1 h1fit()

function f = h1fit(x,tmesh)

% L1 fitting sine fitting function
y1 = x(3)*tmesh.^2 + x(2)*tmesh + x(1)*ones(size(tmesh));
y2 = abs(y1 - sin(tmesh));
f = sum(y2);
Appendix B: MATLAB Script for Parabolic Approximations of Sine Function

B.2.2 h2fit()

function f = h2fit(x, tmesh)
% L2 fitting sine fitting function
% y1 = x(3)*tmesh.^2 + x(2)*ones(size(tmesh));
y1 = x(1)*(tmesh - x(2)*ones(size(tmesh))).^2 + x(3)*ones(size(tmesh));
y2 = (y1 - sin(tmesh)).^2;
f = sum(y2);

B.2.3 hinfit()

function f = hinfit(x, tmesh)
% Linf fitting sine fitting function
% y1 = x(3)*tmesh.^2 + x(2)*ones(size(tmesh));
y1 = x(1)*(tmesh - x(2)*ones(size(tmesh))).^2 + x(3)*ones(size(tmesh));
y2 = abs(y1 - sin(tmesh));
f = max(y2);

B.2.4 mindistfit()

function f = mindistfit(x, tmesh)
% Min Distortion fitting sine fitting function
% y1 = x(3)*tmesh.^2 + x(2)*ones(size(tmesh));
y1 = x(1)*(tmesh - x(2)*ones(size(tmesh))).^2 + x(3)*ones(size(tmesh));
y2 = sum(y1.*sin(tmesh));
y3 = sum(sin(tmesh).^2);
f = abs(y2-y3);

B.2.5 thdopt()

function f = thdopt(x, tmesh)
% Minimum THD fitting for sine wave approximation
% y1 = x(3)*tmesh.^2 + x(2)*ones(size(tmesh));
y1 = x(1)*(tmesh - x(2)*ones(size(tmesh))).^2 + x(3)*ones(size(tmesh));
y2 = sum(y1.*sin(tmesh));
a1 = (2/length(tmesh))*sum(y1.*sin(tmesh));

7 df = (a1/sqrt(2))/sqrt(y2); % Distortion Factor
8 %f = sqrt((1/df)^2 - 1);
9 %f = (1/df)^2 - 1;
11 %f = -(df)^2;
f = (1/df)^2;

B.2.6 thd()

function [f, a1, df] = thd(x, tmesh, width)
% THD calculation for sine wave approximation
y1 = x(1)*(tmesh - x(2)*ones(size(tmesh))).^2 + x(3)*ones(size(tmesh));
Section B.2 : Functions

4 \( y_2 = \frac{\text{sum}(y_1 \cdot 2)}{\text{length}(\text{tmesh})}; \)

6 \( \% a_1 = \frac{2}{\text{width} \cdot \text{length}(\text{tmesh})}) \cdot \text{sum}(y_1 \cdot \sin(\text{tmesh])); \)
\( \% a_1 = \frac{2}{\text{width} \cdot \text{length}(\text{tmesh})}} \cdot \text{sum}(\sin(\text{tmesh})); \cdot 2; \)

8 \( a_1 = \frac{2}{\text{length}(\text{tmesh})} \cdot \text{sum}(y_1 \cdot \sin(\text{tmesh})); \)
\( \% y_3 = \sqrt{2} \cdot \text{sqrt}(y_2)/\text{length}(\text{tmesh})/a_1; \% 1/\text{Distortion Factor} \)

10 \( \text{df} = (a_1/\text{sqrt}(2))/\text{sqrt}(y_2); \% \text{Distortion Factor} \)

12 \( f = \sqrt{(1/\text{df})^2 - 1}; \)

B.2.7 infnorm()

1 \textbf{function} f = infnorm(tmesh, yvals)
\( \% \text{Lin}_f \text{ fitting sine fitting function} \)
3 \( \% y_1 = x(3) \cdot \text{tmesh} \cdot 2 + x(2) \cdot \text{tmesh} + x(1) \cdot \text{ones(size(tmesh))}; \)
\( y_1 = yvals; \)
5 \( y_2 = \text{abs}(y_1 - \sin(\text{tmesh})); \)
\( f = \text{max}(y_2); \)
Appendix C

Field Oriented Control Simulink Model

C.1 Block Models
Appendix C: Field Oriented Control Simulink Model

Figure C.1: Top Level Model
Hysteresis has to be properly set to prevent transition oscillation, or lockup.

Figure C.2: Speed Controller
Anti-windup Switch

Figure C.3: Field Oriented Controller
Section C.2: Induction Motor S-Function

C.2 Induction Motor S-Function

![Diagram of Induction Motor S-Function]

Figure C.4: Induction Motor Model

```matlab
function [sys, x0, str, ts] = ind_motor(t, x, u, flag, P, X0)

%SFUNIMPL General M-file S-function template

% With M-file S-functions, you can define your own ordinary differential
% equations (ODEs), discrete system equations, and/or just about
% any type of algorithm to be used within a Simulink block diagram.

% The general form of an M-File S-function syntax is:
% [SYS, X0, STR, TS] = SFUNC(T, X, U, FLAG, P1, ..., Pn)

% What is returned by SFUNC at a given point in time, T, depends on the
% value of the FLAG, the current state vector, X, and the current
% input vector, U.

% FLAG RESULT DESCRIPTION
% 0 [SIZES, X0, STR, TS] Initialization, return system sizes in SYS,
% initial state in X0, state ordering strings in STR, and sample times in TS.
% 1 DX Return continuous state derivatives in SYS.
```

~ 105 ~
Appendix C : Field Oriented Control Simulink Model

20 % 2 DS Update discrete states SYS = X(n+1)
21 % 3 Y Return outputs in SYS.
22 % 4 TNEXT Return next time hit for variable step sample time in SYS.
23 % 5 {} Reserved for future (root finding).
24 % 9 [ ] Termination, perform any cleanup SYS=[].
25 %
26 % The state vectors, X and X0 consists of continuous states followed by discrete states.
27 %
28 % Optional parameters, P1,...,Pn can be provided to the S-function and used during any FLAG operation.
29 %
30 % When SFUNC is called with FLAG = 0, the following information should be returned:
31 %
32 % SYS(1) = Number of continuous states.
33 % SYS(2) = Number of discrete states.
34 % SYS(3) = Number of outputs.
35 % SYS(4) = Number of inputs.
36 % Any of the first four elements in SYS can be specified as -1 indicating that they are dynamically sized. The actual length for all other flags will be equal to the length of the input, U.
37 % SYS(5) = Reserved for root finding. Must be zero.
38 % SYS(6) = Direct feedthrough flag (1=yes, 0=no). The s-function has direct feedthrough if U is used during the FLAG=3 call. Setting this to 0 is akin to making a promise that U will not be used during FLAG=3. If you break the promise then unpredictable results will occur.
39 % SYS(7) = Number of sample times. This is the number of rows in TS.
40 %
41 % X0 = Initial state conditions or [ ] if no states.
42 % STR = State ordering strings which is generally specified as [ ].
43 % TS = An m-by-2 matrix containing the sample time (period, offset) information. Where m = number of sample times. The ordering of the sample times must be:
44 %
45 % TS = [0 0, : Continuous sample time.
46 % 0 1, : Continuous, but fixed in minor step sample time.
47 % PERIOD OFFSET, : Discrete sample time where
48 % PERIOD > 0 & OFFSET < PERIOD.
49 % -2 0]; : Variable step discrete sample time

~ 106 ~
Section C.2: Induction Motor S-Function

where FLAG=4 is used to get time of next hit.

There can be more than one sample time providing they are ordered such that they are monotonically increasing. Only the needed sample times should be specified in TS. When specifying more than one sample time, you must check for sample hits explicitly by seeing if

$$\text{abs(round}((T-\text{OFFSET})/\text{PERIOD}) - (T-\text{OFFSET})/\text{PERIOD})$$

is within a specified tolerance, generally $1e-8$. This tolerance is dependent upon your model's sampling times and simulation time.

You can also specify that the sample time of the S-function is inherited from the driving block. For functions which change during minor steps, this is done by specifying \(SYS(7) = 1\) and \(TS = [-1 0]\). For functions which are held during minor steps, this is done by specifying \(SYS(7) = 1\) and \(TS = [-1 1]\).

Copyright 1990–2002 The MathWorks, Inc.

$Revision: 1.18$

The following outlines the general structure of an S-function:

```matlab
switch flag,
% Initialization
 case 0,
    [sys,x0,str,ts]=mdlInitializeSizes(X0);
% Derivatives
 case 1,
    sys=mdlDerivatives(t,x,u,P);
% Update
 case 2,
    sys=mdlUpdate(t,x,u);
```

\(\sim 107 \sim\)
Appendix C: Field Oriented Control Simulink Model

% Outputs %

case 3,

```
sys=mdlOutputs(t,x,u,P);
```

% GetTimeOfNextVarHit %

case 4,

```
sys=mdlGetTimeOfNextVarHit(t,x,u);
```

% Terminate %

```
case 9,
sys=mdlTerminate(t,x,u);
```

% Unexpected flags %

```
otherwise

error(['Unhandled flag =',num2str(flag)]);
```

end

function [sys,x0,str,ts]=mdlInitializeSizes(X0)

```
sizes = simsizes;
sizes.NumContStates = 7;
sizes.NumDiscStates = 0;
```

~ 108 ~
Section C.2 : Induction Motor S-Function

sizes.NumOutputs = 11;
sizes.NumInputs = 13;
sizes.DirFeedthrough = 1;
sizes.NumSampleTimes = 1;  \% at least one sample time is needed

sys = simsizes(sizes);

\% initialize the initial conditions
\%
x0 = X0;

\% str is always an empty matrix
\% str = [];

\% initialize the array of sample times
\% ts = [0 0];

\% end mdlInitializeSizes

\% mdlDerivatives
\% Return the derivatives for the continuous states.
\%
function sys=mdlDerivatives(t,x,u,P)
lds = x(1);
lqs = x(2);
10s = x(3);
ldr = x(4);
lqr = x(5);
10r = x(6);
wr = x(7);
w = u(1);
vds = u(2);
vqs = u(3);
v0s = u(4);
rs = u(5);
rr = u(6);
Las = u(7);
Appendix C: Field Oriented Control Simulink Model

212 Las0 = u(8);
    Lar = u(9);
214 Lar0 = u(10);
    M = u(11);
216 J = u(12);
    TL = u(13); %Load torque

218    ids = (Lar*lds - M*ldr)/(Las*Lar - M^2);
220    iqs = (Lar*lqs - M*lqr)/(Las*Lar - M^2);
    i0s = 10s/Las0;
222    idr = (-M*lds + Las*ldr)/(Las*Lar - M^2);
    iqr = (-M*lqs + Las*lqr)/(Las*Lar - M^2);
224    iOr = 10r/Lar0;
226    v0r = 0;

228    TM = (3/2)*P*(lqr*idr - ldr*iqr); %Motor torque

230    dlds = -rs*ids + w*lqs + vds;
232    dlqs = -rs*iqs - w*lds + vqs;
234    d10s = v0s - rs*i0s;
    dldr = -rr*idd + (w-wr)*lqr;
236    dlqr = -rr*iqr - (w-wr)*ldr;
    d10r = v0r - rr*i0r;
238    dwr = (TM-TL)/J;

240    sys = [dlds dlqs d10s dldr dlqr di0r dwr];

242    % end mdlDerivatives

244    % mdlUpdate
    % Handle discrete state updates, sample time hits, and major time step requirements.
    %
248    %
    function sys=mdlUpdate(t,x,u)
250    sys = [];

252    % end mdlUpdate

254    %
256    % mdlOutputs
258    % Return the block outputs.
    %

~ 110 ~
Section C.2: Induction Motor S-Function

function sys=mdlOutputs(t,x,u,P)

lds = x(1);
lqs = x(2);
10s = x(3);
ldr = x(4);
lqr = x(5);
10r = x(6);
wr = x(7);

w = u(1);

vds = u(2);

vqs = u(3);

v0s = u(4);

rs = u(5);

rr = u(6);
Las = u(7);
Las0 = u(8);
Lar = u(9);
Lar0 = u(10);
M = u(11);
J = u(12);

TL = u(13); %Load torque

ids = (Lar*lds - M*ldr)/(Las*Lar - M^2);
lqs = (Lar*lqs - M*lqr)/(Las*Lar - M^2);
i0s = 10s/Las0;
ldr = (-M*lds + Las*ldr)/(Las*Lar - M^2);
lqr = (-M*lqs + Las*lqr)/(Las*Lar - M^2);
10r = 10r/Lar0;

TM = (3/2)*P*(lqr*idr - ldr*iqr); %Motor torque

sys = [ids iqs i0s wr TM lds lqs l0s ldr lqr l0r];

% end mdlOutputs

% mdlGetTimeOfNextVarHit
% Return the time of the next hit for this block. Note that the result is
% absolute time. Note that this function is only used when you specify a
% variable discrete-time sample time [-2 0] in the sample time array in
% mdlInitializeSizes.

function sys=mdlGetTimeOfNextVarHit(t,x,u)
Appendix C: Field Oriented Control Simulink Model

308    %sampleTime = 1;    % Example, set the next hit to be one second later.
310    %sys = t + sampleTime;
312    sys = [];

% end mdlGetTimeOfNextVarHit

% mdlTerminate

% Perform any end of simulation tasks.

% function sys=mdlTerminate(t,x,u)
322    sys = [];
324    % end mdlTerminate
Appendix D

Motor Control Embedded Firmware

D.1 Main Motor Control Module

D.1.1 mot_cntl.c

//TMS320LF2406A Main Module Module

//Peripheral TMS320LF2406A

//Author: Al-Thaddeus Avestruz

//Created: 1 Dec 2004

//Copyright 2004 Al-Thaddeus Avestruz

//

// mot_cntl.c

/* */

#include <stdlib.h>

#include <string.h>

#include "regs240x.h"

#include "pwm/include/2407pwm.h"

#include "pwm/include/svgen.h"

#include "sysvecs.h"

#include "sine_pwm_init.h"

#include "umacros.h"

#include "serialcomm.h"

#include "periph.h"

#include "vfcontrol.h"

#define WAIT_STATES 0x40;

#define SET_LO(x,b)  ((x)&=~(1<<b))

#define SET_HI(x,b) ((x)|(1<<b))

~ 113 ~
```c
#define CLKOUT 40000000

// LIMITS

#define RAMP_END 256000L
#define RAMP_END 12800L // need the L postfix
#define V1_MAX 32767
#define F1_MAX (60*256)
#define RAMP_dV 100L
#define RAMP_dF 100L
#define RAMP_VINTVL ((long)(RAMP_dV*RAMP_END)/(long)V1_MAX)
#define RAMP_FINTVL ((long)(RAMP_dF*RAMP_END)/(long)F1_MAX)

void interrupt periodic_isr (void);
void interrupt phantom(void);
void trap(void);
void setup_PLL(void);

// void RampVF(void);

/* pwm stuff */

volatile unsigned long isr_count = 0;
volatile unsigned long evCounterA = 0;
volatile unsigned long evCounterB = 0;
volatile int tmpregister1 = 0;
volatile int tmpregister2 = 0;
extern volatile unsigned long Ramp_Count;
/*
typedef struct
{
    char * array;
    unsigned int index;
    unsigned int length;
    int lock;
    int full;
    int empty;
    int reading;
} typecBuffer;
*/

typecBuffer SerTxBuffer = {0, 0, 64, 0, 0, 0, 0};
```

Appendix D : Motor Control Embedded Firmware
type Ramp  VFRamping = {1,0,1,1,0,0,0};
    type Vout  VControl = {0, 0,0,0,0,0,0,0};

main ()
{
    idiv_t idiv_r;
    unsigned long ltmpl = 0;
    signed int itmp = 0;
    signed int itoggle = 1;
    disable_ints();
    WDXR = 0x68;
    MCRA = MCRA&(~0x4000); // Makes port IO BP6
    setup_PLL();
    setup_PWM(&sin_pwm);
    setupEVB();
    setupTimer3();
    SetupSerial(); sendChar('E');
    // setup_ADC();
    setupV3ADC();
    SCSR1 = 0x0001;  // Clear ILLADR bit
    IFR = 0xffff;  /* Clear all interrupts. */
    IMR = 0x0002 + 0x0001;  // Enable INT2. and INT1 interrupt mask register
    // strcpy(strtmp,"v.1.0");
    // sendString(strtmp);
    // EVAIFRA = 0xffff; /* Clear all EV1 group A EV interrupt flags. */
    // EVAIMRA = 0x0080; /* Enable Timer 1 period interrupt interrupts */
    ENABT3();
    enable_ints();
    sendChar('F');
    // RampVF();  // Volts/Hz Ramp

~ 115 ~
Appendix D: Motor Control Embedded Firmware

```c
126  sin_pwm.V1 = 0;
127  sin_pwm.V3 = 0;
128  sin_pwm.F1.f = 1;
129  sin_pwm.F1.n = 8;
130  update_PWM(&sin_pwm);
132  VFRamping.direction = 1;
133  VFRamping.period = 25600L;
134  VFRamping.dF = 60*256;
135  VFRamping.dV = 32767;
136  VFRamping.fstep = 100L;
137  VFRamping.vstep = 100L;
138  RampVF(&sin_pwm, &VFRamping);
140  /*
142  //12/23/04
144  // VFRamping.direction = -1;
146  // VFRamping.period = 6000L;
148  // VFRamping.dF = 20*256;
150  // VFRamping.dV = 1;
152  // VFRamping.fstep = 350L;
154  // VFRamping.vstep = 250L;
156  RampVF(&sin_pwm, &VFRamping);
158  */
160  VFRamping.direction = -1;
162  VFRamping.period = 12800L;
164  VFRamping.dF = 1;
166  // VFRamping.dV = 6554;
168  // VFRamping.dV = 15000;
170  // VFRamping.dV = 12000;
172  VFRamping.fstep = 0;
174  VFRamping.vstep = 100L;
176  RampVF(&sin_pwm, &VFRamping);
178  */
180  */
```

~ 116 ~
Section D.1: Main Motor Control Module

VFRamping.direction = -1;
VFRamping.period = 2400L;
VFRamping.dF = 1;
VFRamping.dV = 6554;
VFRamping.fstep = 0;
VFRamping.vstep = 1000L;
RampVF(&sin_pwm, &VFRamping);

VFRamping.direction = -1; //12-23-04
VFRamping.period = 8000L;
VFRamping.dF = 1;
VFRamping.dV = 5000;
VFRamping.fstep = 0;
VFRamping.vstep = 100L;
RampVF(&sin_pwm, &VFRamping);

// sin_pwm.V3 = -6554;

VControl.igain = 5;
VControl.pgain = 5;
VControl.igain = 60000;
VControl.pgain = 60000;
VControl.igain = 50;
VControl.pgain = 50;
VControl.igain = 25;
VControl.pgain = 25;
VControl.vcommand = 389;
VControl.vcommand = 76;
VControl.vcommand = 420;
VControl.vcommand = 480;
VControl.vcommand = 275; //old value 225

sin_pwm.V3 = -10000; update_PWM(&sin_pwm);
// sin_pwm.V3 = -10000;

sin_pwm.V3 = 1000; update_PWM(&sin_pwm);
sin_pwm.V3 = -10923; update_PWM(&sin_pwm);

createBuffer(&SerTxBuffer);
SerTxBuffer.empty = 1;
while(1)
{

if (isr_count >= 2000L)
{

/*

~ 117 ~
*/
Appendix D: Motor Control Embedded Firmware

//VControl.vcommand = 389 + itoggle*30;
VControl.vcommand = VControl.vcommand + itoggle*30;
itoggle = -1*itoggle;
isr_count = 0;
}

/*
if (0 == VFRamping.ramping)
{
    VFRamping.direction = -1;
    VFRamping.period = 12000L;
    VFRamping.dF = 20*256;
    VFRamping.dV = 10000;
    VFRamping.fstep = 100L;
    VFRamping.vstep = 100L;
}
RampVFControl(&sin_pwm, &VFRamping, &evCounterB, 4000L, 1);

if (0 == VFRamping.ramping)
{
    VFRamping.direction = 1;
    VFRamping.period = 12000L;
    VFRamping.dF = 20*256;
    VFRamping.dV = 10000;
    VFRamping.fstep = 100L;
    VFRamping.vstep = 100L;
}
RampVFControl(&sin_pwm, &VFRamping, &evCounterB, 4000L, 2);
*/

if (0 == VFRamping.ramping) //12-20-04
{
    VFRamping.direction = -1;
    VFRamping.period = 6000L;
    VFRamping.dF = 20*256;
    VFRamping.dV = 10000;
    VFRamping.fstep = 250L;
    VFRamping.vstep = 250L;
}
RampVFControl(&sin_pwm, &VFRamping, &evCounterB, 2000L, 1);

if (0 == VFRamping.ramping)
{
    VFRamping.direction = 1;
    VFRamping.period = 6000L;
    VFRamping.dF = 20*256;
    VFRamping.dV = 10000;
    VFRamping.fstep = 250L;
    VFRamping.vstep = 250L;

~ 118 ~
RampVFControl(&sin_pwm, &VFRamping, &evCounterB, 2000L, 2);

sin_pwm.V3 = -(32767 - sin_pwm.V1);
// Hack to demonstrate third harmonic voltage regulation
sin_pwm.V3 = 0;
// controlV3(&VControl, &sin_pwm);

update_PWM(&sin_pwm);
// Single update for everybody to ensure synchronous update

writeSerBuffer(&SerTxBuffer, VControl.insum, &evCounterA, 500, 1);
writeSerBuffer(&SerTxBuffer, VControl.verror, &evCounterA, 500, 2);
writeSerBuffer(&SerTxBuffer, VControl.vcommand, &evCounterA, 500, 3);
writeSerBuffer(&SerTxBuffer, VControl.accum, &evCounterA, 500, 4);
writeSerBuffer(&SerTxBuffer, sin_pwm.V3, &evCounterA, 500, 5);
writeSerBuffer(&SerTxBuffer, VControl.vbus, &evCounterA, 500, 6);
terminate_wrtSerBuffer(&SerTxBuffer, &evCounterA, 500);

// sin_pwm.V1 = 0x7fff;
// sin_pwm.V1 = 26213;
// sin_pwm.V3 = 0x7fff;
// sin_pwm.V3 = 0;
// sin_pwm.F1.f = 60*256;
// sin_pwm.F1.n = 8;

if (isr_count >= 5000L)
if (isr_count >= 5000L)
{
    sin_pwm.V3 = -sin_pwm.V3;
    sin_pwm.V3 = sin_pwm.V3 + itoggle*10000;
    update_PWM(&sin_pwm);
    itoggle = -itoggle;
    isr_count = 0;
}

sendChar('V');
// printADC(8);
// printADC(5);
// sendChar('B');
// printADC(3);
Appendix D: Motor Control Embedded Firmware

```c
// sendChar('H'); //this works

int main() {
    /* end main() */
}

interrupt void high_interrupt_isr() {
    if (1 == BITGET(EVAFRA,0)) { //Power Drive Interrupt
        sin_pwm.fault.flag = 1;
        sin_pwm.fault.count++;
        sin_pwm.V3 = 0;
        sin_pwm.V1 = 0;
        update_PWM(&sin_pwm);
        sendChar('f');
    }
}

interrupt void periodic_interrupt_isr() {
    disable_ints();
    PBDATDIR |= 0x0040; //Set IOBP6 High
    /* print_reg('A', IFR); */
    /* print_reg('B', IMR); */
    /* print_reg('C',EVAFRA); */
    /* print_reg('D',EVAIMRA); */
    /* sendChar('
'); */
    if (1 == BITGET(EVAFRA,7)) { //TiPINT Flag
        PBDATDIR |= 0x0040; //Set IOBP6 High
        handle_PWM_interrupt(&sin_pwm);
        EVAIMRA = 0x080; /* Enable Timer 1 period interrupt interrupts */
        EVAFRA = 0xffff; /* Clear all EV1 group A EV interrupt flags */
    }
}
```
sendChar ('P');
print_reg ('A', IFR);
print_reg ('B', IMR);
print_reg ('C', EVAIFRA);
print_reg ('D', EVAIMRA);
sendChar ('\r'); sendChar ('\n');
PBDATDIR &= ~0x0040; //Sets IOBP6 Low

if (1 == BITGET(EVBIFRA, 7)) //T3PINT Flag
{
    isr_count++;
    Ramp_Count++;
    evCounterA++;  
    evCounterB++;
    update_Vout(&VControl);
    EVBIFRA = 0xffff;
}

/* Done with the ISR */

IFR = 0xffff; /* Clear all interrupts.
*/ //To do: clear only the flag that was handled;
this should automatically clear w/o intervention
IMR = 0x0002; /* Enable INT2. */

EVAIFRA = 0xffff; /* Enable Timer 1 period interrupt interrupts */
EVAIMRA = 0x0080; /* Enable Timer 1 period interrupt interrupts */
EVBIFRA = 0xffff; /* Clear all EV1 group A EV interrupt flags. */
PBDATDIR &= ~0x0040; //Sets IOBP6 Low
cenable_ints();

void trap()
{
    //Square wave on the contactor pin (#13) on the serial connector.
    MCRA = MCRA & (~0x4000);

    //make that pin an output pin
    PBDATDIR = PBDATDIR | 0x4000;

~ 121 ~
while(1) {
    if (GPTCON & (1<<13))
      PBDATDIR |= 0x0040;
    else
      PBDATDIR &= ~0x0040;
}

void setupPLL() {
  /* setup the PLL module */
  asm("SPLK #0041h,PLLCNT ");
  asm("SPLK #00B1h,PLLCNT2 ;CLKIN(XTAL)=10MHz, PLL*2.0=20MHz; ");
  asm("SPLK #0081h,PLLCNT1 ;CLKMD=PLL Enable, fSYSCLK=f.CPUCLK/2");
  asm("SPLK #0080h,PLLCNT1 ;CLKMD=PLL Enable, fSYSCLK=f.CPUCLK/4");
  asm("SPLK #40C0h,SYSCR ;CLKOUT=CPUCLK");
  SCSR1 = 0x0000; //0x0000;
}

void interrupt phantom() {
  // IFR = 0xffff; /* Clear all interrupts. */
}

/* RampVF(void) */
int ramping = 0;
unsigned long ltmp = 0;
// ldiv_t ldiv_r;
// unsigned long lcount = 0;
unsigned long prevFCount = 0;
unsigned long prevVCount = 0;
sin_pwm.V1 = 0;
sin_pwm.V3 = 0;
sin_pwm.F1.f = 1*256;
sin_pwm.F1.n = 8;
update_PWM(&sin_pwm);
disable_ints();
isr_count = 0;
Section D.1: Main Motor Control Module

```c
enable_ints();
ramping = 1;

ltmp = (long)RAMP.VINTVL;

while (1 == ramping)
{
    tmpregister1 = T3PR;
    tmpregister2 = T3CNR;
    if (sin_pwm.F1.f < F1_MAX)
    {
        if ((isr_count - prevFCount) > (long)RAMP.FINTVL)
        {
            sin_pwm.F1.f = sin_pwm.F1.f + RAMP.dF;
            update_PWM(&sin_pwm);
            prevFCount = isr_count;
        }
    }
    if (sin_pwm.V1<(unsigned int)V1_MAX)
    {
        if ((isr_count - prevVCount) > (long)RAMP.VINTVL)
        {
            sin_pwm.V1 = sin_pwm.V1 + RAMP.dV;
            //PBDATDIR |= 0x0040; //Set IOBP6 High
            update_PWM(&sin_pwm);
            //PBDATDIR &= ~0x0040; //Sets IOBP6 Low
            prevVCount = isr_count;
        }
    }
    if(isr_count >= RAMP.END)
        ramping = 0;
}
```

D.1.2 umacros.h

// Utility Macros

// Utility Macros for TMS320LF2406A
// Author: Al-Thaddeus Avestruz
// Created: 7 November 2004
// Copyright 2004 Al-Thaddeus Avestruz
// umacros.h
//REV 1.0

#ifndef __UMACROS_
#define __UMACROS_

#define BITSETL(x,b) ((x)&=~(1<<(b)))
#define BITSETH(x,b) ((x)l=(1<<(b)))
#define BITGET(x,b) (((x)>>b)&0x0001)
#define SETLO(x,b) ((x)&=~(1<<(b)))
#define SETHI(x,b) ((x)=(1<<(b)))

#define PI 3.1415927
#define TWOPI 6.2831853
#define TWOPIBYTHREE 2.0943951
#define SEVENPI 21.991485751286
#define ONEBYTWOPI 0.159154943091895

#define disableints() asm("-----setc-----intm-----")
#define enableints() asm("-----clrc-----intm-----");
asm("_NOP");asm("_NOP");asm("_NOP") //Bug SDSsq29090

#define sgn(x) (-((x)<0) + ((x)>0))

typedef struct
{
  unsigned int f;
  int n;
} typeuQint;

typedef volatile unsigned int typeFlag;

typedef volatile struct
{
  unsigned int locked;
  unsigned int id;
  unsigned int life;
} typeSemaphore;

typedef volatile struct
{
  typeSemaphore R; //Read
  typeSemaphore W; //Write
  typeFlag U;     //Update
} typeRWSemaphore;

typedef volatile struct
{
union
{
    int * iarray;
    unsigned int * uiarray;
    char * txtarray;
    float * farray;
    long * larray;
} addr;
unsigned int length;
unsigned int rindex;
unsigned int windex;
typeFlag overflow;
typeRWSemaphore sem;
}
typeCircBuffer;

typedef struct
{
    char * array;
    volatile unsigned int index;
    unsigned int length;
    volatile int lock;
    volatile int full;
    volatile int empty;
    volatile int reading;
} typeCircBuffer;

union TwoBytes
{
    unsigned int t;
    unsigned char bt[2];
    //To do: Check array alignments chars are 16 bits
};

struct bitfield
{
    unsigned int b7:1;
    unsigned int b6:1;
    unsigned int b5:1;
    unsigned int b4:1;
    unsigned int b3:1;
    unsigned int b2:1;
    unsigned int b1:1;
    unsigned int b0:1;
};
typedef union Byte
{
    struct bitfield bit;

    ~ 125 ~
Appendix D: Motor Control Embedded Firmware

    unsigned char byte;

} byte;

#include // _UMACROS_

D.2 Sine Inverter PWM Module

D.2.1 sin_pwm.c

//Sine PWM Module
//
//SCI Peripheral TMS320LF2406A
//Author: Al-Thaddeus Avestruz
//Created: 11 November 2004
//Copyright 2004 Al-Thaddeus Avestruz

//
//sine_pwm.c
//REV 1.0
//12/1/04  Nearly constant switching frequency

#include "regs240x.h"
#include <stdlib.h>
#include "pwm/include/F2407pwm.h"
#include "pwm/include/svgen.h"
#include "umacros.h"
#include "serialcomm.h"
#include "sine_pwm.h"

/**
 * typedef struct
 {          
      signed int V1;
      unsigned int F1;
      signed int V3;
      unsigned int F3;
      signed int Ph3;  //not yet implemented
      unsigned int fs;
      unsigned int N1;
      unsigned int N3;
      unsigned int tpd;
      int mflag;
 } typeSIN_PWM;
 */

//Data Tables
#define TBLLEN 108  //Should always be a multiple of 3
#define TBLLEN3 TBLLEN/3
#define TBLLEN23 2*TBLLEN/3

~ 126 ~


Section D.2: Sine Inverter PWM Module

```c
#define TBLLEN_2 TBLLEN/2
#pragma DATA SECTION(SinTab, ".tables")
// #pragma CODE SECTION(update_PWM, "fast")

signed int SinTab[] =
{0,1905,3804,5690,7557,9398,11207,12978,14706,16383,
 18006,19567,21062,22486,23834,25101,26283,27376,28377,
 29282,30087,30791,31390,31884,32269,32545,32712,32767,
 32712,32545,32269,31884,31390,30791,30087,29282,28377,
 27376,26283,25101,23834,22486,21062,19567,18006,16384,
 14706,12978,11207,9398,7557,5690,3804,1905,
 0,-1905,-3804,-5690,-7557,-9398,-11207,-12978,-14706,-16383,
-18006,-19567,-21062,-22486,-23834,-25101,-26283,-27376,-28377,
-29282,-30087,-30791,-31390,-31884,-32269,-32545,-32712,-32767,
-32712,-32545,-32269,-31884,-31390,-30791,-30087,-29282,-28377,
-27376,-26283,-25101,-23834,-22486,-21062,-19567,-18006,-16384,
-14706,-12978,-11207,-9398,-7557,-5690,-3804,-1905};

//Functions
//To do: run this interrupt handler out of RAM
void handle_PWM_interrupt(typeSIN_PWM * pwm)
{
  div_t idiv_r;
  ldiv_t ldiv_r;

  signed int fna = 0;
  signed int fnb = 0;
  signed int fnc = 0;
  signed int fn3 = 0;
  signed int itmp = 0;

  //Private variables
  static unsigned int ncnt = 0;
  static unsigned int N1 = 0;
  static unsigned int n3 = 0;
  static unsigned int na = 0;
  static unsigned int nb = 0;
  static unsigned int nc = 0;
  static unsigned int tpd_2 = TMR_PERIOD/2;
  static signed int VI = 0;
  static signed int V3 = 0;
  static signed int V3tmp = 0;
  static signed int tpd = TMHPERIOD;
  //must be signed for the compiler to do the //multiply properly

  //To do: fix tpd so it can be unsigned and still multiply properly
```
if (pwm->mflag)     //Check mutex flag for update
{
    V1 = pwm->V1;
    V3tmp = pwm->V3;
    tpd = pwm->tpd;
    tpd_2 = tpd>>1;
    N1 = pwm->N1;
    T1PR = pwm->tpd;    /* Timer Period Register*/
}

if ((0==n3)&&(TBLLEN_2==n3)) V3=V3tmp;
// update only on zero crossings of phi3

ncnt++;  
if (ncnt >= N1)  
{
//Fundamental
    //To do: error here maybe should be na++<TBL_LEN
    if (na<TBL_LEN) na++; else na=0;
    if (na>TBLLEN2.3) nb = na - TBLLEN2.3;
        else nb = na + TBLLEN2.3;    //if (ka + K/3)>K
    if (na>TBLLEN.3) nc = na - TBLLEN.3;
        else nc = na + TBLLEN2.3;    //if (ka + 2*K/3)>K

    //To do: MSK V1 also.  e.g. 
    //if ((0==na)||( TBL_LEN_2==na))  V1=V1new;

    itmp = ((long)V1 * (long)SinTab[na])>>16;
        //Voltage Scale (14 bits max)
    itmp = 2*itmp;
    fna = ((long)itmp * (long) tpd)>>16;//really a mult by tpd/2
    if (fna>0) fna-=1;
    if (fna<0) fna+=1;
    itmp = ((long)V1 * (long)SinTab[nb])>>16;
    itmp = 2*itmp;
    fnb = ((long)itmp * (long) tpd)>>16;
    if (fnb>0) fnb-=1;
    if (fnb<0) fnb+=1;
    itmp = ((long)V1 * (long)SinTab[nc])>>16;
    itmp = 2*itmp;
    fnc = ((long)itmp * (long) tpd)>>16;
    if (fnc>0) fnc-=1;
    if (fnc<0) fnc+=1;

    //3rd Harmonic
    if ((n3+3)<TBL_LEN) n3=n3+3; else n3=0;
Section D.2: Sine Inverter PWM Module

```c
    itmp = ((long)V3 * (long)SinTab[n3])>>16;
    itmp = 2*itmp;
    fn3 = ((long)itmp * (long)tpd)>>16;
    if (fn3>0) fn3 -= 1;
    if (fn3<0) fn3 += 1;

    /* Phase A duty cycle */
    CMPR1 = ((unsigned int)((fna + fn3 + tpd_2));
    // Compare threshold
    itmp = CMPR1;
    /* Phase B duty cycle */
    CMPR2 = ((unsigned int)((fnb + fn3 + tpd_2));
    /* Phase C duty cycle */
    CMPR3 = ((unsigned int)((fnc + fn3 + tpd_2));

    ncnt = 0;
};
```

```c
    pwm->mflag = 0;  // release mutex
    if (27==na)
    {
        printreg ('M',CMPR1);
        printreg ('T', tpd);
    }
    if (81==na)
    {
        printreg ('N',CMPR1);
        printreg ('U', tpd);
    }
```

```c
    void update_PWM(typeSIN_PWM * pwm)
    {
        div_t idiv_r;
        ldiv_t ldiv_r;
        unsigned long ltmp = 0;
        static unsigned long fs0thresh = FS0;

        while (pwm->mflag);  // Wait for flag to clear
```

```c
    pwm->F3.f = 3*pwm->F1.f;
    pwm->F3.n = pwm->F1.n;
    ltmp = ((unsigned long)TBLLEN * (unsigned long)pwm->F1.f);
    ltmp = ltmp>>pwm->F1.n;
    ldiv_r = ldiv((unsigned long) fs0thresh , ltmp);
```

```c

// 129 //
```
Appendix D: Motor Control Embedded Firmware

```c
pwm->N1 = (unsigned int)ldiv_r.quot;
if ((2*ldiv_r.rem) > ltmp) pwm->N1+=1; // rounding
// To do: add rounding hysteresis so won't oscillate between fs's

ltmp = pwm->fs;
pwm->fs = ((unsigned long)pwm->N1 /
    *(unsigned long)TBL_LEN * /
    (unsigned long)(pwm->F1.f)>>pwm->F1.n);
// align switching period with the table

if (ltmp > pwm->fs) fs0thresh = FS0 - FS0HYST;
else if (ltmp < pwm->fs) fs0thresh = FS0 + FS0HYST;
ldiv_r = ldiv((unsigned long) CLKOUT, /
    2*PRESCALE*(unsigned long) pwm->fs);
pwm->tpd = (unsigned int) ldiv_r.quot; // calc new timer period
// To do: is this less than 32767
if (pwm->tpd > 32767) pwm->tpd=32767;
// added 2/20/05 because handle_PWM_interrupt() won't do >32767
if (pwm->tpd< (unsigned int)MIN_TMR1PD) /
    pwm->tpd = (unsigned int)MIN_TMR1PD;
// limit the switching freq

// Update parameters — setting the flag causes isr to
// update all parameters
disable_ints();
pwm->mflag = 1;
enable_ints();
}

void setup_PWM(typeSIN_PWM * pwm)
{
    // To do: disable interrupts
    /* setup the modules */

    MCRA = MCRA&(~0x4000);
PBDATDIR = PBDATDIR&0x4000;

    /* Set T3PWM low */
    SET_LO(MCRC, 10); // Select IOPF2
    SET_HI(PFDATDIR, 10); // Output
    SET_LO(PFDATDIR, 2); // Pin low

    /* reset any faults */
    reset_PWM_fault();
```

~ 130 ~
PIACKR1 = 0x0001; /* Ack PDP Interrupt — Si Errata
EVAIFRA = 0xFFFF; // Clear EVA interrupt flags

/* Set up PWM the correct way: */
SCSR1 |= 0x0004; // EVA Clock Enable

pwm->V1 = 0;
pwm->V3 = 0;
pwm->F1.f = 1*256;
pwm->F1.n = 8;
update_PWM(pwm);
T1PR = pwm->tpd;
/* Set Timer Period explicitly so that we can go into the interrupt*/

DBTCONA = DBTCON.INIT.STATE;
/* Setup the bridging IGBT gate driver polarities */
ACTRA = COMPARE1.AL +
       COMPARE2.AH +
       COMPARE3.AL +
       COMPARE4.AH +
       COMPARE5.AL +
       COMPARE6.AH;

CMPI1 = 0; /* Phase A duty cycle */
CMPI2 = 0; /* Phase B duty cycle */
CMPI3 = 0; /* Phase C duty cycle */

COMCONA=0xa200; // 10100010 CENABLE CLD0 FCOMPOE
T1CON = PWM.INIT.STATE;
MCRA |= 0x0fc0;

EVAIFRA = 0xffff; /* Clear all EV1 group A EV interrupt flags. */
EVAIMRA = 0x0080 + 0x0001;
/* Enable Timer 1 period interrupts and PDP*/

/* Setup done */
/* reset any faults */
reset_PWM_fault();

// To do: enable interrupts

}  

void reset_PWM_fault(void)
Appendix D : Motor Control Embedded Firmware

```c
int i;
/* Clear faults on drivers */
SET_LO(MCRC,3);    // Select IOPE3
SET_HI(PEDATDIR,11); // Output
SET_HI(PEDATDIR,3); // Pin high
SET_LO(PEDATDIR,3); // Pin low

/* Assert latch reset */
SET_LO(MCRB,8);    // Select IOPD0
SET_HI(PDDATDIR,8); // Output
SET_LO(PD_DATDIR,0); // Pin low
for(i=0;i<100;i++)
    asm("nop");    // Delay to allow the latch to reset
SET_HI(PD_DATDIR,0); // Pin high
}

D.2.2 Header Files

sin_pwm_init.h

//Sine PWM Initializations Header File
//SCI Peripheral TMS320LF2406A
//Author: Al-Thaddeus Avestruz
//Created: 11 November 2004
//Copyright 2004 Al-Thaddeus Avestruz
//
#include "sine_pwm.h"
//REV 1.0

//Data Structures
/*
typedef volatile struct
{
    signed int V1;
typeuQint F1;
signed int V3;
typeuQint F3;
signed int Ph3;    //not yet implemented
unsigned long fs;
unsigned int N1;
unsigned int N3;
unsigned int tpd;
unsigned int prescale;
typeFault fault;
unsigned int vbus;
```

~ 132 ~
Section D.2: Sine Inverter PWM Module

```c
29   volatile int mflag;
    } typeSIN_PWM;
31 */

33 typeSIN_PWM sin_pwm =
    {
35   0,   //V1
36   {1*256,8},   //F1
37   0,   //V3
38   {3*256,8},   //F3
39   0,   //Ph3
40   FS0,   //fs
41   1,   //N1
42   1,   //N3
43   TMR_PERIOD,  //tpd
44   1,   //prescale
45   {0,0},  //fault.flag, fault.count
46   0,   //mflag 0=exclude
    };

D.2.3 sin_pwm.h

1 //Sine PWM Module Header File
2 //
3 //SCI Peripheral TMS320LF2406A
4 //Author: Al-Thaddeus Avestruz
5 //Created: 11 November 2004
6 //Copyright 2004 Al-Thaddeus Avestruz
7 //
8 //sine_pwm.h
9 //REV 1.0
11 #include <stdio.h>
12 #include  "umacros.h"
13 //
14 //define FS0 10800
15 //Nominal Switching Frequency needs to be a multiple of TBL.LEN
16 //define FS0HYST 108
17 #define FS0 30800
18 //Nominal Switching Frequency needs to be a multiple of TBL.LEN
19 #define FS0HYST 308
21 #define PRESCALE 1
23 #define TMR_PERIOD (5000/PRESCALE)
   //Nominal Timer Period— Must be less than 32767
25   //for handle_PWM_interrupt() to work properly
```
Appendix D : Motor Control Embedded Firmware

#define MINTMR1PD (200/PRESCALE)
//To do: Make an ifndef here
#define CLKOUT 40000000

//Data Structures
/*
typedef struct
{
  signed int V1;
  unsigned int F1;
  signed int V3;
  unsigned int F3;
  signed int Ph3;  //not yet implemented
  unsigned int fs;
  unsigned int N1;
  unsigned int N3;
  unsigned int tpd;
  volatile int mflag;
} typeSIN_PWM;
*/
typedef volatile struct
{
  unsigned int flag;
  unsigned int count;
} typeFault;

typedef volatile struct
{
  signed int V1;
  typeuQint F1;
  signed int V3;
  typeuQint F3;
  signed int Ph3;  //not yet implemented
  unsigned long fs;
  unsigned int N1;
  unsigned int N3;
  unsigned int tpd;
  unsigned int prescale;
  typeFault fault;
  unsigned int vbus;
  volatile int mflag;
} typeSIN_PWM;

//Function Prototypes

void handle_PWM_interrupt(typeSIN_PWM *);
D.3 Volts per Hertz Module

D.3.1 vfcontrol.c

```c
#include <stdlib.h>
#include "regs240x.h"
#include "periphs.h"
#include "umacros.h"
#include "sine_pwm.h"
#include "vfcontrol.h"

define NEGTHRESH 100

typedef struct
{
    int direction; // (plus or minus 1)
    unsigned long period;
    unsigned int dV;
    unsigned int dF;
    unsigned int fstep;
    unsigned int vstep;
} typeRamp;

extern volatile unsigned long RampCount = 0;

void RampVF(typeSINPWM * pwm, typeRamp * ramp)
{
    int ramping = 0;
    unsigned long prevFCount = 0;
    unsigned long prevVCount = 0;
    unsigned long fintvl = 0;
```
Appendix D: Motor Control Embedded Firmware

```c
unsigned long vintvl = 0;
unsigned int currentdF = 0;
unsigned int currentdV = 0;
signed int dir = 1;
signed int currentF = 0;
signed int currentV = 0;

ldiv_t ldiv_r;

ldiv_r = ldiv((long)ramp->vstep * ramp->period, (long)ramp->dV);
vintvl = ldiv_r.quot;

ldiv_r = ldiv((long)ramp->fstep * ramp->period, (long)ramp->dF);
fintvl = ldiv_r.quot;

disable_ints();
Ramp.Count = 0;
enable_ints();

while (Ramp.Count <= ramp->period)
{
    //
    // tmpregister1 = T3PR;
    // tmpregister2 = T3CNT;
    if (currentdF < ramp->dF)
    {
        if ((Ramp.Count - prevFCount) > fintvl)
        {
            currentdF += ramp->fstep;
            currentF = pwm->F1.f;
            currentF = currentF + (ramp->direction * ramp->fstep);
            if (currentF >0) pwm->F1.f = (unsigned int) currentF;
            else pwm->F1.f = 1;
            update_PWM(pwm);
            prevFCount = Ramp.Count;
        }
    }

    if (currentdV < ramp->dV)
    {
        if ((Ramp.Count - prevVCount) > vintvl)
        {
            currentdV += ramp->vstep;
            currentV = (unsigned int)_pwm->V1;
            currentV = currentV + (ramp->direction * ramp->vstep);
            if (currentV >=0) pwm->V1 = currentV;
            else pwm->V1 = -currentV;
            // PBDATDIR |= 0x0040; // Set IOBP6 High
            update_PWM(pwm);
        }
    }
}
```

~ 136 ~
Section D.3 : Volts per Hertz Module

89 //
90 // PBDATDIR = 0x0040; //Sets IOBP6 Low
91 prevVCount = Ramp_Count;
92 }
93 //while
94 }
95
96 void RampVFControl(typeSIN_PWM *pwm, typeRamp *ramp,
97 volatile unsigned long *ptimer, / unsigned long trigger, unsigned int myinstance)
99 {
100 static unsigned long prevFCount = 0;
101 static unsigned long prevVCount = 0;
102 static unsigned long fintvl = 0;
103 static unsigned long vintvl = 0;
104 static unsigned int currentdF = 0;
105 static unsigned int currentdV = 0;
106 static unsigned int triggered = 0;
107 signed int currentF = 0; //temp variables
108 signed int currentV = 0;
109 ldiv_t ldiv_r;
110
111 if (0 == ramp->ramping)
112 {
113     ramp->ramping = myinstance;
114     ldiv_r = ldiv((long)ramp->vstep * ramp->period, (long)ramp->dV);
115     vintvl = ldiv_r.quot;
116     ldiv_r = ldiv((long)ramp->fstep * ramp->period, (long)ramp->dF);
117     fintvl = ldiv_r.quot;
118     prevFCount = 0;
119     prevVCount = 0;
120     currentdF = 0;
121     currentdV = 0;
122     triggered = 0;
123     disable_ints();
124     ptimer = 0;
125     enable_ints();
126 }
127 if ((ptimer->trigger) && (myinstance == ramp->ramping) && !triggered)
130 ~ 137 ~
{  
  *ptimer = 0;
  triggered = 1;
}

if (triggered && (myinstance == ramp->ramping))
{
  if (*ptimer <= ramp->period)
  {
    // tmpregister1 = T3PR;
    // tmpregister2 = T3CNT;
    if (currentdF < ramp->dF)
    {
      if (((*ptimer - prevFCount) > fintvl))
      {
        currentdF += ramp->fstep;
        currentF = pwm->F1.f;
        currentF = currentF + (ramp->direction * ramp->fstep);
        if (currentF > 0) pwm->F1.f = (unsigned int) currentF;
        else pwm->F1.f = 1;
        //
        update_PWM(pwm);
        prevFCount = *ptimer;
      }
    }
    if (currentdV < ramp->dV)
    {
      if (((*ptimer - prevVCount) > vintvl))
      {
        currentdV += ramp->vstep;
        currentV = (unsigned int) pwm->V1;
        currentV = currentV + (ramp->direction * ramp->vstep);
        if (currentV >= 0) pwm->V1 = currentV;
        else pwm->V1 = -currentV;
        //
        PBDATDIR |= 0x0040;  //Set IOBP6 High
        update_PWM(pwm);
        //
        PBDATDIR &= ~0x0040;  //Sets IOBP6 Low
        prevVCount = *ptimer;
      }
    }
    else
    {
      ramp->ramping = 0;
      triggered = 0;
      disable_ints();
      *ptimer = 0;
      enable_ints();
    }  
}  

~138~
Section D.3: Volts per Hertz Module

```c
typedef struct {
    unsigned int vbus;
    unsigned int vcommand;
    unsigned int vmeas[8];
    unsigned int insum;
    signed int verror;
    signed int accum;
    unsigned int pgain;
    unsigned int igain;
    unsigned int mflag;
} typeVout;

void controlV3(typeVout *vout, typeSIN_PWM *pwm) /* PI Controller */ {
    int i = 0;
    signed int v3max = 0;
    signed long accum_max = 0;
    signed int poutput = 0;
    static signed long accumout = 0;
    signed int tmpoutput = 0;
    div_t idivr;

    if (1 == vout->mflag)
    {
        PBDATDIR |= 0x0040; // Set IOBP6 High
        vout->insum = 0;
        for (i=0; i<8; i++)
        {
            vout->insum = vout->insum + (vout->vmeas[i]>>5);
            // Sum eight 10 bit left justified unsigned inputs.
            // Full count 14 bits right justified.
        }

        v3max = 32767 - pwm->V1;
        accum_max = ((signed long) v3max)<<16; // 32 bit accumulator
        vout->vbus = vout->vbus>>4;
        // Bus voltage measurement 10 bits. Full count 14 bits
        vout->verror = (signed int) /
            (vout->vcommand<<4) - (signed int) vout->insum;
    }
```
Appendix D: Motor Control Embedded Firmware

```c
accumout += 4L * /
((signed long) vout->igain * (signed long) vout->verror);

// Anti-Windup: Saturate accumulator for underflow and overflow
if (accumout < 0) accumout = 0;
if (accumout > accum_max) accumout = accum_max;
vout->accum = (signed int) (accumout >> 16);

// Calculate proportional output
poutput = (10L * /
(signed long)vout->pgain * (signed long)vout->verror)>>16;
// poutput = 5*poutput;

// Check for possible overflow and then add gain outputs
if ((sgn(poutput) == sgn(vout->accum)) && (0 != sgn(poutput)))
{
  if (abs(poutput) > (32767 - abs(vout->accum)))
  {
    if (1 == sgn(poutput)) tmpoutput = 32767; // overflow
    else tmpoutput = -32767; // underflow
  }
  else tmpoutput = poutput + vout->accum;
}
else tmpoutput = poutput + vout->accum;

// Saturate V3 for underflow and overflow
// negative V3 values for phi = pi control
if (tmpoutput < 0) pwm->V3 = 0;
if (tmpoutput > v3max) pwm->V3 = -v3max;
else pwm->V3 = -abs(tmpoutput);

// PBDATDIR |= 0x0040; // Set IOBP6 High
// update_PWM(pwm);
// PBDATDIR &= ~0x0040; // Sets IOBP6 Low
disable_ints();
vout->mflag = 0;
enable_ints();
```

~ 140 ~
void update_Vout(typeVout * vout)
{
    static int i = 0;
    if ((0 == vout->mflag) && (0 == (ADCTRL2 & SEQ1.BSY)))
    {
        if (i<8)
        {
            MAXCONV = 0x0001; //Two conversions
            CHSELSEQ1 = 0x0008 + (0x0003<<4);
            ADCTRL2 |= RESET_SEQ1;
            ADCTRL2 |= SOC_SEQ1; //start conversion
                asm(".NOP");
                asm(".NOP");
                asm(".NOP");
                asm(".NOP");
            while (ADCTRL2 & SEQ1.BSY);
            vout->vmeas[i] = RESULT0;
            vout->vbus = RESULT1;
            i++;
        }
        else
        {
            i = 0;
            vout->mflag = 1;
        }
    }
}
void setupV3ADC(void)
{
    //Dedicated ADC Pins
    //Setup ADC Timer
    SCSR1 |= 0x0080;
    //Setup ADC Control Register
    ADCTRL1 = RESET_ADC + ADC_SOFT + ACQ_PRESCALE_X2 + CPS_CLK_2 + START_STOP;
    ADCTRL2 = INT_DIS_SEQ1 + INT_DIS_SEQ2;
    //Maximum Conversions per autoconvert
    MAXCONV = 0x0001; //2 conversions
    //ADC input channel sequencing
    CHSELSEQ1 = 8; //ADC in from DC Reg Output
Appendix D: Motor Control Embedded Firmware

329  CHSELSEQ1 += 0x0003<<4; //Inverter DC Bus Voltage
331  //Reset ADC
333  resetADC();

D.3.2 vfcontrol.h

1 //Motor Controller Module Header File
//
3 //SCI Peripheral TMS320LF2406A
//Author: Al-Thaddeus Avestruz
5 //Created: 13 December 2004
//Copyright 2004 Al-Thaddeus Avestruz
7 //
9 //vfcontrol.h
11 //REV 1.0

13 #include "umacros.h"

15 ifndef _VFCONTROL_
16 #define _VFCONTROL_
18
typedef struct
19 {     int direction; // (plus or minus 1)
21  unsigned long period;
23  unsigned int dV;
25  unsigned int dF;
27  unsigned int fstep;
29  unsigned int vstep;
31  int ramping;
33 } typeRamp;

37 typedef struct
39 {     unsigned int vcommand;
41  unsigned int vmeas[8];
43  unsigned int vbus;
45  unsigned int insum;
47  signed int verror;
49  signed int accum;
51  unsigned int pgain;
53  unsigned int igain;
55  unsigned int mflag;
57 } typeVout;

~ 142 ~
Section D.4: Serial Communications Module

D.4 Serial Communications Module

D.4.1 serialcomm.c

// Serial Communications Module
//
// SCI Peripheral TMS320LF2406A
// Author: Al-Thaddeus Avestruz
// Created: 2 November 2004
// Copyright 2004 Al-Thaddeus Avestruz
//
// serialcomm.c
// REV 1.0

#define ODD 0
#define EVEN 1

#include "regs240x.h"
#include "umacros.h"
#include "serialcomm.h"
#include <stdlib.h>

// SetupSerial tested 11/17/04

unsigned char SetupSerial(void)
{
  
// Prototypes

void RampVF(typeSIN_PWM * pwm, typeRamp * ramp);
void RampVFControl(typeSIN_PWM * pwm, typeRamp * ramp,
                    volatile unsigned long * ptimer,
                    unsigned long trigger,
                    unsigned int myinstance);
void controlV3(typeVout * vout, typeSIN_PWM * pwm);
void update_Vout(typeVout * vout);
void setupV3ADC(void);

//Globals

//typeRamp VFRamping = {1,0,1,1,0,0};
//volatile unsigned long Ramp_Count = 0;

#endif
Appendix D: Motor Control Embedded Firmware

23 // union TwoBytes brr; // Bit Rate Register
24 // byte bSCICCR;
25
26 // bSCICCR.byte = 0;
27
28 // Configure SCITx and SCIRx pins
29 MCR &= 0x0003;
30
31 // Enable SCI CLK
32 SCSR1 |= 0x0040; // Bit 6 SCI CLKEN
33
34 // Enable RS485 driver — needed for PIIPM
35 MCR &= ~0x0004; // Makes port IOPA2
36 PADATDIR |= 0x0004; // Set IOPA2 as output
37 PADATDIR &= ~0x0004; // Sets IOPA2 Low
38
39 // Register Setup SCICCR — SCI Communication Control Register
40
41 // bSCICCR.byte = DATA-1; // Sets lower 3 bits
42 // bSCICCR.bit.b7 = STOP-1;
43 // bSCICCR.bit.b6 = PARITY;
44 // bSCICCR.bit.b5 = PARITYEN;
45 // bSCICCR.bit.b4 = 0; // Disable Loopback
46 // bSCICCR.bit.b3 = 0; // 0 — Select Idle Line Mode; 1 — Address Bit Mode
47
48 // SCICCR = bSCICCR.byte;
49 SCICCR = 0x07;
50 BITSET.L(SCICCR,7);
51 BITSET.L(SCICCR,6);
52 BITSET.L(SCICCR,5);
53 BITSET.L(SCICCR,4);
54 BITSET.L(SCICCR,3);
55
56 // SCI Control Register
57 SCICTL1 = ((RXERRINT<<6)+(SWRESET<<5)+(TXWAKE<<3)+/
58 (SLEEP<<2)+(TXENA<<1)+RXENA);
59 SCICTL2 = ((RXBKIT<<1) + TXINT);
60
61 // Baud Rate Register Setup
62
63
64
65
66
67
68
69
Section D.4: Serial Communications Module

```c
// brr_it = ((int) CLKOUT)/((int) BAUD*8) - 1;
// Save the 16-bit value in local

// SCIHBaud = brr.bt[1];
// Write low byte to Baud Select Register High byte
SCIHBaud = brr.bt[0];
// Write high byte to Baud Select Register Low byte

SCIHBaud = 0x00; // 38400 baud with a 40 MHz CLKOUT
SCILBAUD = 0x81;

// Enable Receive Buffer Interrupt
// BITSET_H(SCIPRI,5); // Low priority
// BITSET_H(SCICTL2,1); // Enable
// BITSET_L(SCICTL2,1); // Disable

SCIPRI = ((TXPRIORITY<<6)+(RXPRIORITY<<5)+(SCISOFTFREE<<3));
setSCI();

// Enable SCI
// BITSET_H(SCICTL1,1); // Transmitter enable
// BITSET_L(SCICTL1,0); // Receiver disable

SCICTL1 |=0x0020;
}

// sendChar() tested 11/17/04
unsigned char sendChar(unsigned char cinput)
{
    while (0==BITGET(SCICTL2,7)); // Wait until buffer is empty
    SCITXBUF=cinput;
    return(1);
}

unsigned char sendc(unsigned char cinput)
{
    if (1==BITGET(SCICTL2,7)) // Check if buffer empty
    {
        SCITXBUF=cinput;
        return(1);
    }
    else return(0);
}
```
Appendix D: Motor Control Embedded Firmware

```c
void print_reg(char label, int reg)
{
    int i = 0;
    sendChar(label);
    sendChar('\\n');
    for (i=16; i>0; i--)
    {
        if ((12==i)) sendChar('\');
        if (1 == ((reg>>(i-1))& 0x0001)) sendChar('1');
        else sendChar('0');
    }
    sendChar('\\r'); sendChar('\\n');
}
inline unsigned char in_sendChar(unsigned char cinput)
{
    if (1==BITGET(SCICTL2,7))
    {
        SCITXBUF=cinput;
        return(1);
    }
    return(0);
}
unsigned char sendStrLit(const char *inputstr)
//To do; doesn't work.
//need to do extra stuff because can't get a pointer to program memory
{
    while ('\0' != *inputstr)
    {
        if (1==sendChar(*inputstr)) inputstr++;
    }
}
unsigned char sendString(char *inputstr)
{
    while ('\0' != *inputstr)
    {
        sendChar(*inputstr);
        inputstr++;
    }
}
unsigned char sendStringTask(char *inputstr, char *strqueue)
{
}
```

~ 146 ~
inline void resetSCI(void) {
    asm("_NOP");
    BITSETL(SCICTL1, 5); // Active low reset
    asm("_NOP");
}

inline void setSCI(void) {
    asm("_NOP");
    BITSETH(SCICTL1, 5); // Enable
    asm("_NOP");
}

void iprintd(char *string, unsigned int n) {
    unsigned int i = 0;
    const long a[5] = {10000, 1000, 100, 10, 1};
    ldiv_t ldiv_r;
    for (i = 0; i < 5; i++) {
        ldiv_r = ldiv((long)n, a[i]);
        *string = 0x0030 + (unsigned int) ldiv_r.quot;
        n = (unsigned int) ldiv_r.rem;
        String++;
    }
    *string = '\0'; // Terminate string
}

void createBuffer(typeBuffer *buffer) {
    buffer->array = (char *) calloc(buffer->length, sizeof(char));
}

void writeSerBuffer(typeBuffer *buffer, int invar, volatile unsigned long *ptimer, unsigned long trigger, 
    unsigned int myinstance) {
    static int i = 0;
    // To do: put these static in the struct
    // so each instance doesn't have to store its own
    static int j = 0;
}
Appendix D : Motor Control Embedded Firmware

```c
static int n = 0;
ldiv_t ldiv_r;
const long a[5] = {10000,1000,100,10,1};

if (0 == buffer->lock)
{
    n = invar; //To do: this is redundant
    buffer->lock = myinstance;
}

if (buffer->empty && (myinstance == buffer->lock) && !buffer->full && (*ptimer>trigger))
{
    if (i<(buffer->length - 2))
    {
        if (0==j)
        {
            if (n<0) *(buffer->array + buffer->index) = '-';
        else  *(buffer->array + buffer->index) = ' \';
        j++;
    }
    else
    {
        if (j<6)
        {
            ldiv_r = ldiv((long)n,a[j-1]);
            *(buffer->array + buffer->index) = 0x0030 + \\
                (unsigned int) labs(ldivr.quot);
            n = (unsigned int)ldivr.rem;
            j++;
        }
        else
        {
            *(buffer->array + buffer->index) = ' \';
            j = 0;
            buffer->lock = 0;
        }
    }
    buffer->index++;
}
else
{
    *(buffer->array + buffer->index) = 'V';
    //Signal overflow
    *(buffer->array + buffer->index) = '\0';
    //Terminate string
}
```

~ 148 ~
Section D.4: Serial Communications Module

```c
263         disable_ints();
264         *ptimer = 0;
265         buffer->lock = 0;
266         buffer->full = 1;
267         enable_ints();
268     }
269 }

273 void terminate_wrtSerBuffer(typecBuffer * buffer,
                                  volatile unsigned long *ptimer, unsigned long trigger)
274 {
275     //this guy closes up the buffer and sends it off to the interrupt
276     //driven serial out routine
277     if (buffer->empty && !buffer->lock && !buffer->full && (*ptimer>trigger))
278     {
279         *(buffer->array + buffer->index) = ';';
280         buffer->index++;
281         *(buffer->array + buffer->index) = '\0';
282         disable_ints();
283         *ptimer = 0;
284         buffer->full = 1;
285         enable_ints();
286     }
287     if (buffer->full && !buffer->reading) SCITXBUF = '\0';
288     //wakes up the interrupt
289 }

293 void sendBuffer(typecBuffer * buffer)
294 {
295     static int i = 0;
296     if (buffer->full)
297     {
298         if (i<buffer->index)
299             { SCITXBUF = *(buffer->array + i);
300                 buffer->reading = 1;
301                 i++;
302             }
303         else
304             { i=0;
305                 buffer->index = 0;
306                 buffer->full = 0;
307                 buffer->empty = 1;
308                 buffer->reading = 0;
309             }
310     }
311 }
```
Appendix D : Motor Control Embedded Firmware

D.4.2 serialcomm.h

// Serial Communications Module
// SCI Peripheral TMS320LF2406A
// Author: Al-Thaddeus Avestruz
// Created: 2 November 2004
// Copyright 2004 Al-Thaddeus Avestruz

//serialcomm.h
//REV 1.0

#ifndef _SERIALCOMM_
#define _SERIALCOMM_

#define BAUD 38400
#define CLKOUT 40000000

// Setup Parameters
#define DATA 8
#define PARITY_EN 0
#define PARITY ODD
#define STOP 1 // One or two stop bits

// SCICTL1
#define RXERRINT 0 // 0: disable rx error interrupts
#define SWRESET 0 // 0: Reset
#define TXWAKE 0 // 0: no wakefunc now
#define SLEEP 0 // 0: sleep disabled
#define TXENA 1 // 1: tx enable
#define RXENA 0 // 1: rx enable

// SCICTL2
#define RXBKINT 0 // 0: disable RX and break interr
#define TXINT 1 // 0: disable TXRDY-interr

// SCIPRI
#define TXPRIORITY 0 // 0: TXD-int on high priority (INT1)
#define RXPRIORITY 0 // 0: RXD-int on high priority (INT1)
#define SCISOFTFREE 2 // on emulator suspend complete SCI

// Function Prototypes

unsigned char SetupSerial();

unsigned char sendChar(unsigned char);
D.5 Peripheral Driver Module

D.5.1 periphs.c

#include "regs240x.h"
#include "umacros.h"
#include "periphs.h"
#include "serialcomm.h"

void setupEVB(void)
Appendix D : Motor Control Embedded Firmware

{ 
    EVBIMRA = 0x0000; //Mask all EVB interrupts
    // BITSET.H(SCSR1,3); //EVB Clock enable
    SCSR1 |= 0x0008;
    //Register may be Read-Modify-Write; BITSET may not work.
    EVBIFRA = 0xffff; //Reset all EVB flags
}

void setupTimer3(void)
{
    // BITSET.H(EVBIMRA,7); //enable Tmr3 period interrupts
    EVBIMRA = 0x0040;
    T3CON = T3SETUP;
    GPTCONB = 0x0000;
    T3PR = T3PERIOD; //Set period register
    // T3CNT = 0; //Clear T3 Counter
    EVBIFRA = 0xffff; //Reset Tmr3 interrupt flags
}

void setupADC(void)
{
    //Dedicated ADC Pins
    //Setup ADC Timer
    SCSR1 |= 0x0080;
    //Setup ADC Control Register
    ADCTRL1 = RESET_DEC + ADC_SOFT + ACQ_PRESCALE.X2 + \CPS_CLK.2 + START_STOP;
    ADCTRL2 = INT_DIS_SEQ1 + INT_DIS_SEQ2;
    //Maximum Conversions per autoconvert
    MAXCONV = 0x0000; //1 conversion
    //ADC input channel sequencing
    CHSELEQ1 = 0x0008; //ADC in from DC Reg Output
    CHSELEQ1 += 0x0003>>4; //Inverter DC Bus Voltage
    //Reset ADC
    resetADC();
}

unsigned int readADC(unsigned int channel)
{
    MAXCONV = 0x0000; //Single conversion
    CHSELEQ1 = channel;
    ADCTRL2 |= RESET_SEQ1;
    ADCTRL2 |= SOC_SEQ1; //start conversion

    ~ 152 ~
asm( "_NOP" );
asm("_NOP");
asm("_NOP");
asm("_NOP");

while (ADCTRL2 & SEQ1.BSY);
return(RESULT0);
}

void printADC(unsigned int channel)
{
    unsigned int result = 0;
    char strtmp[10] = "";
    result = readADC(channel);
    iprintd(strtmp,result);
    sendString(strtmp); sendChar('\n');
}

void resetADC(void)
{
    ADCTRL1 |= 0x4000; //Reset ADC
    asm("_NOP");
    ADCTRL1 &= ~0x4000; //Unreset ADC
}

D.5.2 periphs.h

//TMS320LF2406A Peripheral Driver Header Module
//Peripheral TMS320LF2406A
//Author: Al-Thaddeus Avestruz
//Created: 11 November 2004
//Copyright 2004 Al-Thaddeus Avestruz

#include "./pwm/include/F2407BMSK.h"
#include "umacros.h"

//Timer Parameters
#define T3PERIOD 500
#define T3PERIOD 20000

//T3CON
Appendix D: Motor Control Embedded Firmware

```c
#define T3SETUP (SOFT.STOP_FLAG + TIMER.CONT.UP + 
   TIMER.CLK.PRESCALE.X.1 + 
   TIMER.ENABLE.BY.OWN + TIMER.DISABLE + 
   TIMER.CLOCK.SRC.INTERNAL)
   //Timer compare disabled; Own period register

// For a 40 Mhz clock w/ prescale 1, \
\   tick = 500us for T3Period 20000

//GPTCONB

//ADC Bit Masks
//ADCTRL1
#define RESET_ADC 0x4000
#define ADC_SOFT 0x2000
#define ADC_NOSOFT 0x0000
#define ADC_FREE 0x1000
#define ADC_NOFREE 0x0000
#define ACQ_PRESCALE_X1 0x0000
#define ACQ_PRESCALE_X2 0x0100
#define ACQ_PRESCALE_X3 0x0200
#define ACQ_PRESCALE_X4 0x0300
#define CPS_CLK_1 0x0000
#define CPS_CLK_2 0x0080
#define CONT_RUN 0x0040
#define START_STOP 0x0000
#define INT_H_PRIORITY 0x0000
#define INT_L_PRIORITY 0x0020
#define DUAL_SEQUENCE 0x0000
#define CASCADE 0x0010
#define CAL_ENABLE 0x0008
#define BRIDGE_EN 0x0004
#define REFLO 0x0000
#define REFHI 0x0001
//ADCTRL2
#define RESET_SEQ1 0x4000
#define START_CAL 0x4000
#define SOC_SEQ1 0x2000
#define SEQ1_BSY 0x1000
#define INT_DIS_SEQ1 0x0000
#define INT_ENA1_SEQ1 0x0400
#define INT_ENA2_SEQ1 0x0800
#define INT_FLAG_SEQ1 0x0200
#define EVA_SOC_SEQ1 0x0100
#define RESET_SEQ2 0x0040
#define SOC_SEQ2 0x0020
#define SEQ2_BSY 0x0010
```

~ 154 ~
Section D.5: Peripheral Driver Module

```c
#define INT_DIS_SEQ2 0x0000
#define INT_ENA1_SEQ2 0x0004
#define INT_ENA2_SEQ2 0x0008
#define INT_FLAG_SEQ2 0x0002
#define EVBSOC_SEQ2 0x0001

//Prototypes
void setupEVB(void);
void setupTimer3(void);
void setupADC(void);
unsigned int readADC(unsigned int _channel);
void resetADC(void);
void printADC(unsigned int _channel);
void setupCapture(unsigned int _timer);
void handleCapture(void);

//Macros
#define ENABT3() (T3CON |= TIMER_ENABLE)
#define DISBT3() (T3CON &~ TIMER_ENABLE)
```
Bibliography


Bibliography


