This is to certify that the

thesis entitled

UTILIZING AN FPGA IN CONJUCTION WITH A DSP PROCESSOR
TO FACILITATE THE IMPLEMENTATION OF
INDUCTION MOTOR CONTROL

presented by

John William Kelly

has been accepted towards fulfillment
of the requirements for

M.S. degree in Electrical Eng

EG Strangas
Major professor

Date 5 Dec 97

UTILIZING AN FPGA IN CONJUNCTION WITH A DSP PROCESSOR TO FACILITATE THE IMPLEMENTATION OF INDUCTION MOTOR CONTROL

By

John William Kelly

A THESIS

Submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Electrical Engineering

1997

ABSTRACT

UTILIZING AN FPGA IN CONJUNCTION WITH A DSP PROCESSOR TO FACILITATE THE IMPLEMENTATION OF INDUCTION MOTOR CONTROL

By

John William Kelly

The digital control program of an induction motor can be separated into two distinct components, the actual control program and the pulse width modulation (PWM) program. The control program is feedback dependent and sequential in nature, while the PWM program does not depend on feedback and is memoryless. This thesis explores the feasibility of implementing the PWM program on dedicated hardware, functioning in parallel to the control processor. The intention is decreasing the overall computation time. However, it was determined that only a partial PWM dedicated program was needed to improve overall processing time. The other part of the PWM program was implemented on a DSP processor. The DSP processor was also responsible for an indirect rotor flux field orientation control program, which regulated the speed of a 1/12 horsepower induction motor. Experimental results show this type of hardware parallelism decreases computation time, resulting in greater inverter switching frequency.

To Katherine B. Torpey and Annie Kelly

ACKNOWLEDGEMENTS

I would like to thank professor Elias Strangas for his encouragement, support and guidance. I would also like to thank professor Khalil and professor Shanblatt for their time and effort in being on my committee.

Finally, I owe a special thanks to my parents, William and Patricia Kelly, and to my wife Amy for their support, encouragement and love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

A typical digital control program of an induction motor can be separated into two parts, the actual control program and the pulse width modulation (PWM) program. Sensing the stator currents and rotor speed, the control program determines a reference voltage space vector required to achieve the desired response. The implementation of this voltage is the task of the voltage-fed inverter, via a Pulse Width Modulation scheme. From the angle and magnitude of the desired control voltage space vector, the PWM program produces timing pulses for the inverter switches.

The digital implementation of the control and PWM programs is computationally time consuming. The long processing time is at the expense of the inverter voltage resolution. A lower resolution means a lower inverter switching frequency which results in slower response and greater harmonic effects. Therefore, the overall goal in designing a program structure is to attain the highest switching frequency possible. The purpose of this thesis is to explore the idea of utilizing a field programmable gate array (FPGA) in parallel with a digital signal processor (DSP) to decrease the processing time of the total control program of an induction motor.

Before expanding on the decided optimal roles of the FPGA and DSP in the overall program design, overviews on both devices will be discussed.

1.1 THE DSP OVERVIEW

The digital signal processor was chosen for its inherent speed advantage, which is due to its parallel processing architecture. The DSP processor used was the 50MHz, AT&T DSP32C. Two major components of the processor are the control arithmetic unit (CAU) and the data arithmetic unit (DAU). The CAU is responsible for arithmetic calculations required for logic and control functions. Functioning in parallel with the CAU, the DAU is responsible for all 32-bit floating-point operations.

The computational speed advantage of the DSP processer is realized in the DAU. The DAU consists of a dedicated floating-point multiplier and adder, which work in parallel to perform the DAU's basic operation, the inner product step:

$$a = b + c * d \qquad (1\text{-}1)$$

The DAU performs this operation in a four-stage pipelined instruction sequence, fetch-multiply-accumulate-write. First c and d are fetched from memory and multiplied together. Next the product of c and d is added to b. Finally, the sum is written to memory or an I/O port. The instruction cycle time is 80 ns. [1]

1.2 THE FPGA OVERVIEW

A Field Programmable Gate Array consists of three general components; logic blocks, routing resources and I/O blocks. Each one of the components is reprogrammable, which is why FPGAs are a popular prototype tool. The I/O block connects the external package pins to internal logic blocks. Routing resources connect the I/O blocks to Logic Blocks and Logic Blocks to each other. The programmable switching matrices allow Logic Blocks to be connected in an array type architecture on the chip, Figure 1.1. [2]

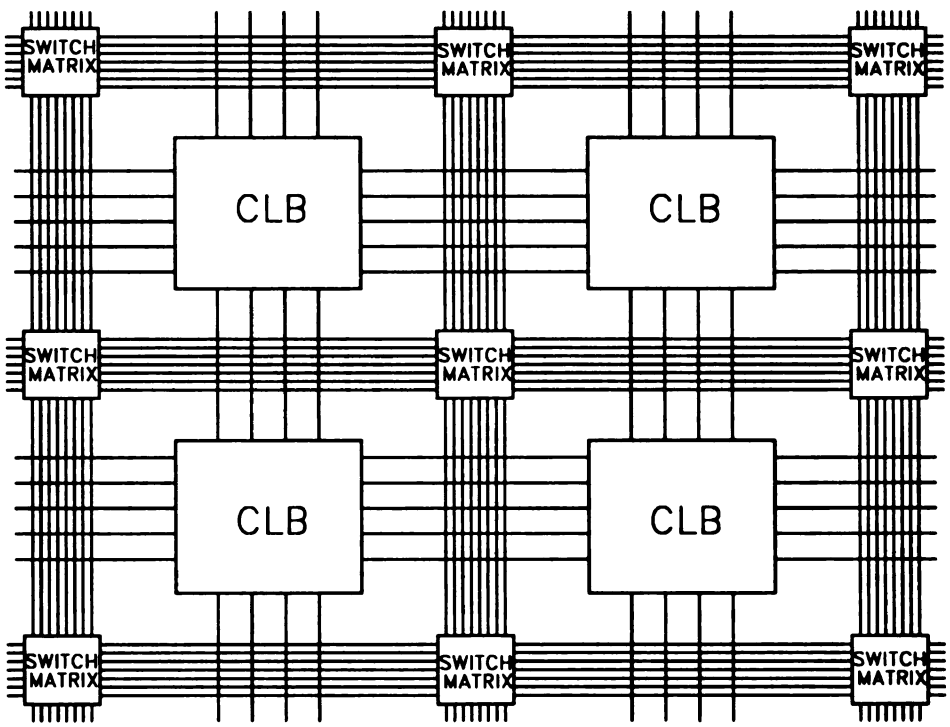

Figure 1.1 - Routing paths between CLBs

The logic block implements the desired function. There are two methods which a combinational function is realized in a FPGA. The function is either implemented by a multiplexer scheme or with a look-up table (LUT). The LUT is the predominant architecture used today. Often called a function generator, the LUT along with the I/O block and switching matrix is configured by static random-access memory (SRAM) cells on the chip. The program written by the user is downloaded from a PC or EPROM to the SRAM cells. Figure 1.2 shows the components of the Combinational Logic Block (CLB) of the Xilinx XC4000 FPGA.

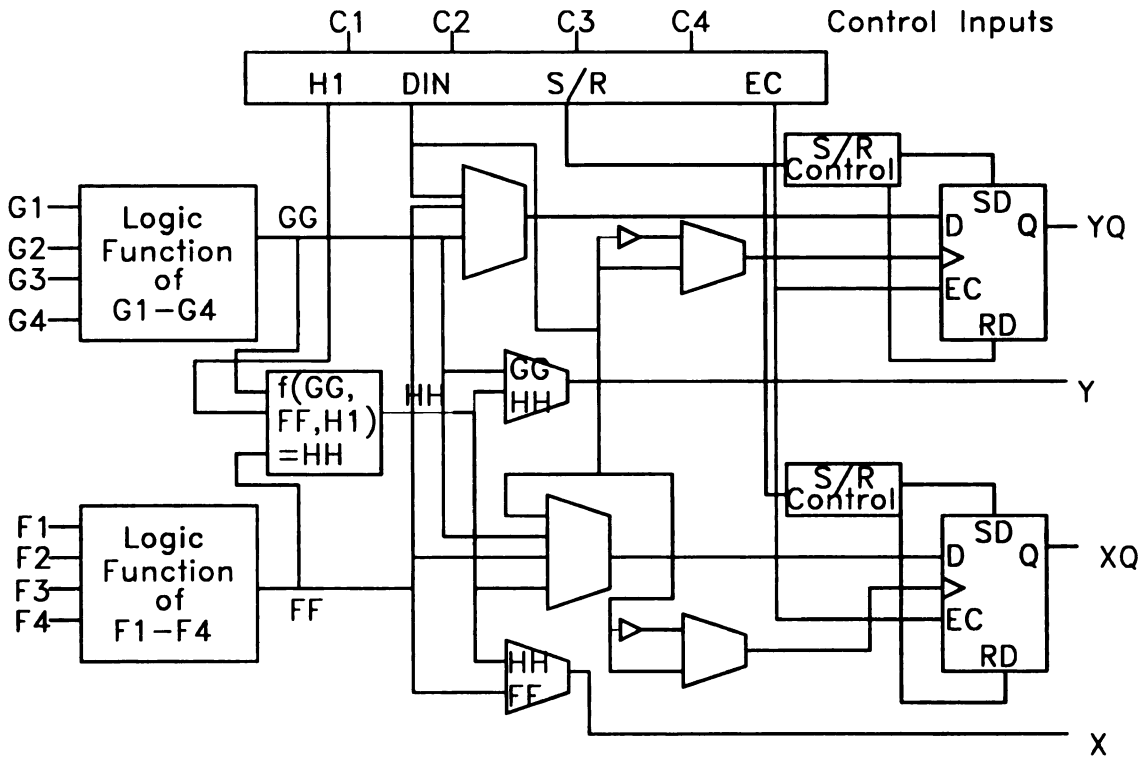

Figure 1.2 - Combinational Logic Block of the Xlinx XC4000 FPGA

From Figure 1.2 it can be seen that the Combinational Function generator has eight variable inputs, G1-G4 and F1-F4. Therefore this CLB can implement the combinational function f(G1-G4,F1-F4). The programmer may design this network either by using a CAD program or writing the Boolean equations. However, in the LUT architecture, the function is implemented by its truth table. The combinational function generator, or lookup table, of the CLB contains the truth table of the function. The output part of the truth table is stored in the SRAM cell and can be reconfigured for different functions. The process of going from a cad program or Boolean expression of a network to a truth table is called Combinational Logic Synthesis.

1.3 PROGRAM ALLOCATION BETWEEN THE FPGA AND DSP

The ideal scheme would utilize the FPGA to implement the total PWM program while the DSP handled the control program. Because the single-input, single-output PWM program is memoryless and feedback independent, it seemed practical to implement the PWM program as a combinational logic circuit on a FPGA. The DSP would determine the magnitude and angle of the voltage space vector and pass these values to the FPGA.

With this information, the first step of the FPGA program would be to perform several trigonometric operations, similar to:

$$A\sin(\alpha) \tag{1-1}$$

$$Asin(\alpha +- n\pi/3) \qquad (1\text{-}2)$$

In order to digitally evaluate these functions, a coordinate rotation digital computer (CORDIC) processor [3] was developed for FPGA implementation, Figure 1.3. However, the cordic processor was complex and CLB expensive. A finite-state-machine had to be designed to control the cordic algorithm. Due to a limited number of CLBs, the resolution of the processor was only 8 bits. Therefore resolution of the PWM signal would be at maximum 255 pulses per period of the fundmental signal. Implementing a cordic processor along with additional PWM arithmetic operations was determined too expensive in terms of CLBs.

Therefore, instead of the total PWM program being implemented on the FPGA, it was more practical to have the trigonometric calculations resolved on the DSP. With the inverse magnitude of the voltage space vector, the DSP determines the cosine and sine of the voltage space vector angle in two multiplication operations. This program topology makes it unnecessary to determine the actual angle on the DSP via a lengthy arctangent algorithm and unnecessary to implement a CLB, costly cordic processor on the FPGA. Due to hardware and time constraints a further simplification was make to the FPGA's PWM responsibility. Half of the PWM timing information is calculated by the DSP. The extra twenty six lines of code, approximately 2.1μs, could be straightforwardly implemented in a combinational logic fashion on a bigger FPGA, if the DSP required more time for the control program. However, the need for more DSP control processing

time was not necessary. Implementing an indirect rotor flux orientation control scheme, a inverter switching frequency of 12.4kHz was obtained.

The following chapter examines the PWM algorithm in terms of the space vector. This is followed by detailing the DSP and FPGA responsibilities in implementing the PWM algorithm. In order to test the PWM program, an indirect rotor flux orientation control scheme was developed for implementation, which is also described in detail. Using this control scheme and the PWM program , a 1/12 hp induction motor was run at different set speeds, 20 - 230 rad/sec. The motor was run with a load and without a load, in order to test the response of the motor. Plots of the speed are presented and discussed. Finally, an evaluation of utilization of a FPGA in conjunction with the DSP is made, along with future recommendations.

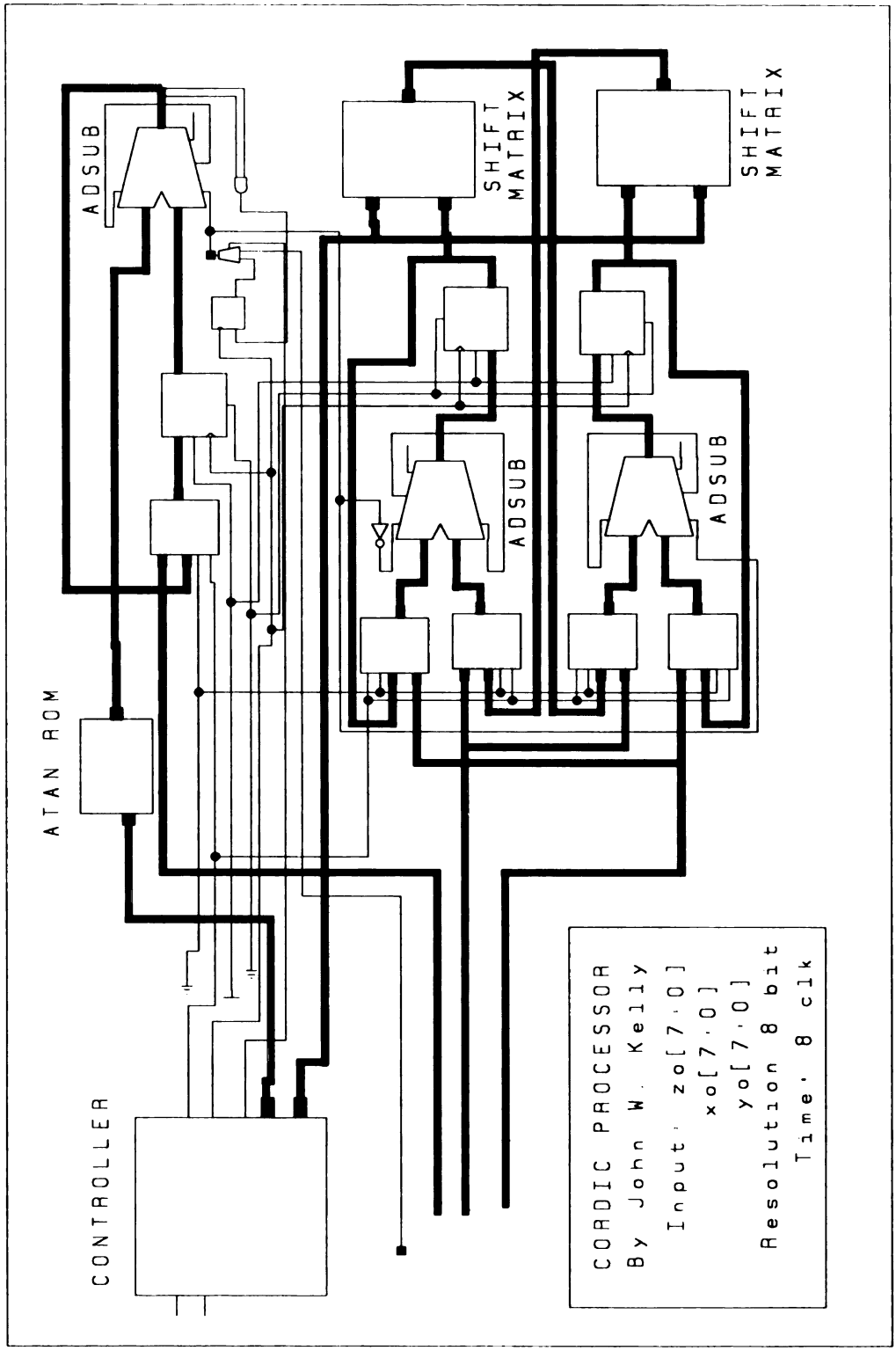

Figure 1.3 - CORDIC Processor

Chapter 2

SPACE VECTOR PULSE WIDTH MODULATION

The three-phase stator voltages of an induction motor can be represented by a space vector as

$$u_s = \frac{2}{3}(u_{sa} + u_{sb}e^{2\pi/3} + u_{sc}e^{4\pi/3}) \qquad (2\text{-}1)$$

At synchronous speed the voltage space vector rotates around the complex plane at synchronous speed and with a constant magnitude. The basic idea of the PWM scheme implemented, is that by controlling the magnitude and angle of the voltage space vector it is possible to control the motor. Space Vector Pulse Width Modulation (SPWM) is a technique for implementing a desired voltage space vector from a d.c. source.

2.1 THE INVERTER

The SPWM technique produces a switching sequence, pulses of varying duration, for turning on and off the IGBTs of an inverter. Therefore, understanding SPWM requires a basic understanding a voltage-fed inverter.

Figure 2.1 is the schematic for a six step inverter. The combination of the "on/off" states of the three pairs of IGBTs results in eight possible switching states. The terminal voltage of each leg of the inverter is U_{t1}, U_{t2} and U_{t3}.

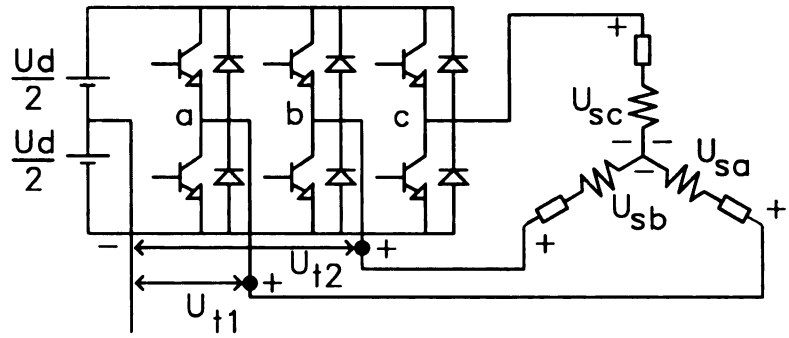

Figure 2.1 -Terminal and phase voltages of an inverter

For a purely resistive load, the combination of terminal voltages produces the "stair-step" like phase voltages: U_{sa}, U_{sb} and U_{sc} , Figure 2.2. [4]

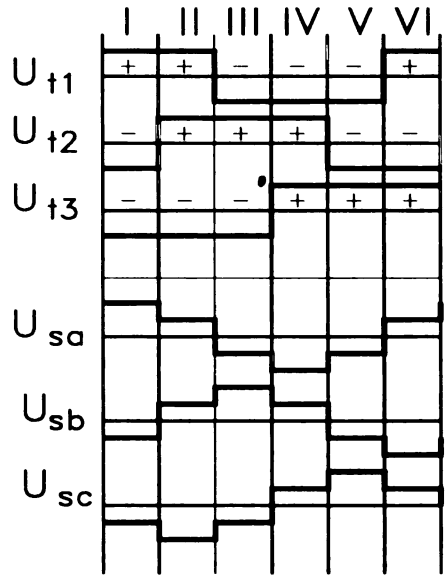

Figure 2.2 -Three-phase inverter with resistive load

The Roman numerals of Figure 2.2 are the switching states. The "+" indicates that the top IGBT is closed or on.

Figure 2.3 shows the space vector representation of the switching states. States ***VII*** and ***VIII*** are zero voltage output states.

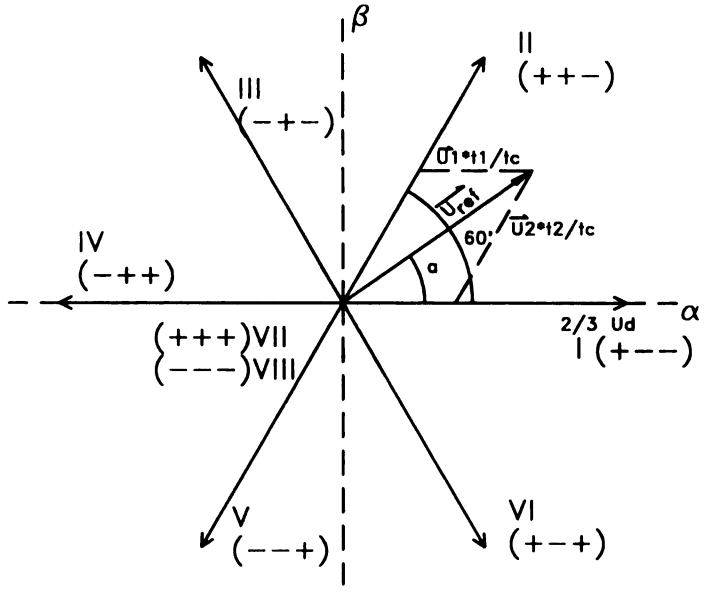

Figure 2.3 -Inverter switching states in the complex plane

Thus, each switching state represents a different voltage space vector value. Additional voltage values can be achieved if the space vector is averaged between two adjacent fundamental states and a zero state. The resulting voltage space voltage u_{ref} in Figure 2.3 is the average of the voltage of the three adjacent inverter switching states. Over a t_c period, the switching state is (+ - -) for a duration of t_1, for a time of t_2 the

switching state is (+ + -), for a time of $t_o/2$ the switching state is (+ + +) and for $t_o/2$ the switching state is (- - -).

$$t_c = \frac{t_o}{2} + t_1 + t_2 + \frac{t_o}{2} \tag{2-2}$$

The time t_c is half the switching period. In order to reduce inverter current harmonics, t_c should be chosen to be as small as possible. The constraints on the relative length of t_c are the speed of the control and pulse width modulation programs and the physical switching ability of the inverter. The DSP SPWM program has the ability to change t_c, the switching frequency, for each pulse. The time t_o is determined from t_1, t_2 and t_c from equation (2-2).

From Figure 2.3, the equation of the voltage space vector u_{ref} is

$$| u_{ref} | t_c = u1 \ t_1 + u2 \ t_2 \tag{2-3}$$

In terms of rectangular coordinates

$$t_1 \ \frac{2}{3} \ u_d \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \ t_2 \ \frac{2}{3} \ u_d \begin{bmatrix} \cos 60 \\ \sin 60 \end{bmatrix} = t_c \ \frac{2}{3} \ u_d \ m \begin{bmatrix} \cos \alpha \\ \sin \alpha \end{bmatrix} \tag{2-4}$$

The modulation index m is defined as:

$$m = \frac{|u_{ref}|}{u_d} \tag{2-5}$$

Therefore, for a voltage space vector between states I and II, the times are:

$$t_1 = t_c * m * \frac{\sin(60-\alpha)}{\sin 60} \tag{2-6}$$

$$t_2 = t_c * m * \frac{\sin(\alpha)}{\sin 60} \tag{2-7}$$

Figure 2-4 shows the inverter input pulses. Every pulse begins with the switching state VII, (- - -), and ends with the switching state VII, (+ + +).

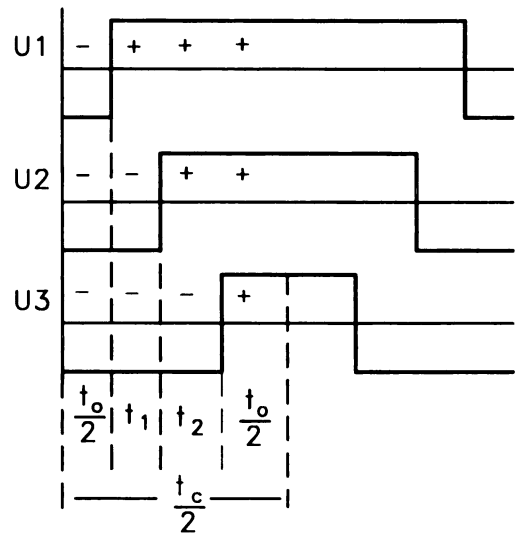

Figure 2.4 -SPWM pulse

The procedure is the same for space vectors in the other five sectors, with angles adjusted accordingly.[5]

Chapter 3

DSP IMPLEMENTATION OF SPWM

The control program determines the required voltage space vector in the stator coordinates. This space vector is represented in its stator complex quaduature components u_α and u_β. The quaduture voltage components are the inputs to the DSP part of the SPWM program. With these two components, the DSP computes the sine and cosine of the angle and the magnitude of the voltage space vector.

3.1 SPACE VECTOR CALCULATIONS

From the stator voltages , the complex quadature voltages u_α and u_β are defined as

$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} u_{sa} \\ u_{sb} \\ u_{sc} \end{bmatrix} \tag{3-1}$$

The magnitude of the voltage space vector is found by

$$| u_{ref} | = \sqrt{(u_\alpha * u_\alpha + u_\beta * u_\beta)} \tag{3-2}$$

However, first the inverse of $| u_{ref} |$ is calculated. The algorithm used is a Chebychev approximation. The execution time of this computation is approximately 2.93 μs with 21 significant bit accuracy. [6]

The angle of the space vector is not found directly, rather the sine and cosine of the angle are determined.

$$\cos(\alpha) = \frac{u_\alpha}{|u_{ref}|} \tag{3-3}$$

$$\sin(\alpha) = \frac{u_\beta}{|u_{ref}|} \tag{3-4}$$

By comparing the values of sine and cosine, the switching sector which the voltage space vector resides can be determined. This avoids the need to calculate the arctangent, which is computationally expensive. The algorithm for determining the switching state requires approximately 1.1 μs.

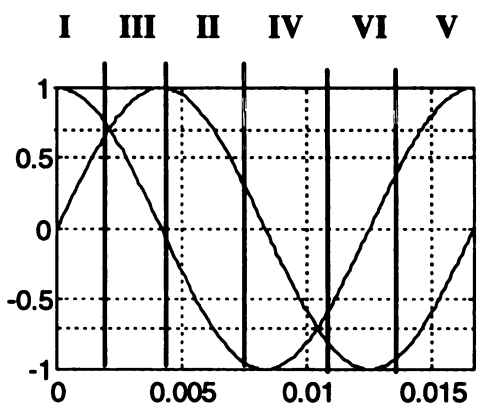

Figure 3.1 -Determining switching state from sine and cosine

3.2 SPWM TIMING CALCULATIONS

Once the voltage space vector switching sector is determined, the SPWM pulse timing information can be resolved. Using the trigonometric identity

$$\sin(\phi - \xi) = \sin(\phi)\cos(\xi) - \cos(\phi) * \sin(\xi) \tag{3-5}$$

there is no need to use a lengthy sine algorithm to determine timing information. For the DSP, determining the timing routine requires a series of floating-point multiplication and addition operations. For example, the timing information of sector I is:

$$t1 = tc * m * \frac{(\sin(60) * \cos(\alpha) - \cos(60) * \sin(\alpha))}{\sin(60)} \tag{3-6}$$

$$t2 = tc * m * \frac{\sin(\alpha)}{\sin(60)} \tag{3-7}$$

The value of the inverse of sin(60) is a constant and is stored as a fixed variable in the DSP SPWM program.

From t_o, t_1 and t_2, the DSP program determines the off times of the SPWM pulses according to which switching sector the voltage space vector resides.

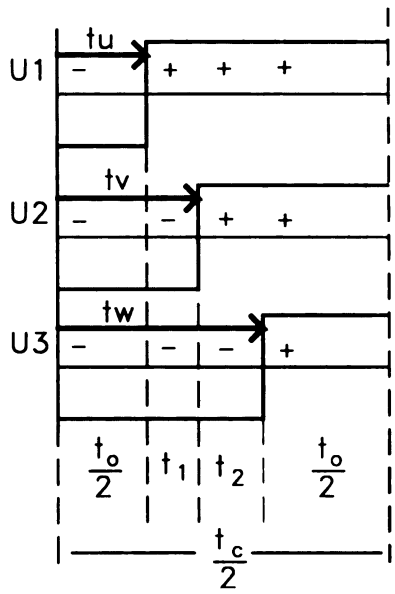

Figure 3.2 -SPWM pulse

From Figure 3.2, *tu*, *tv* and *tw* are the times until the next switching state for the three phases, respectively.

3.3 EXECUTION TIME

Starting with the acquistion of the stator quaduture voltage components, u_α and u_β, until the transfer of the SPWM timing information to the FPGA, the execution time of the DSP SPWM program is approximately 9.7μs. Part of this time is used to determine $2t_c$, the SPWM period. Therefore, not only the magnitude of the space vector can be varied but the its frequency can be varied as well, without sacrificing resolution.

3.4 TOTAL PROGRAM TIMING

After the SPEW timing information is determined, it is serially transferred as a 32-bit word at a rate of 6.25 MHz to the FPGA. Once this word is latched into the serial output buffer of the DSP, the control loop begins again. The actual data transfer to the FPGA requires 5.3 μs. However, the data transfer and control program occur concurrently

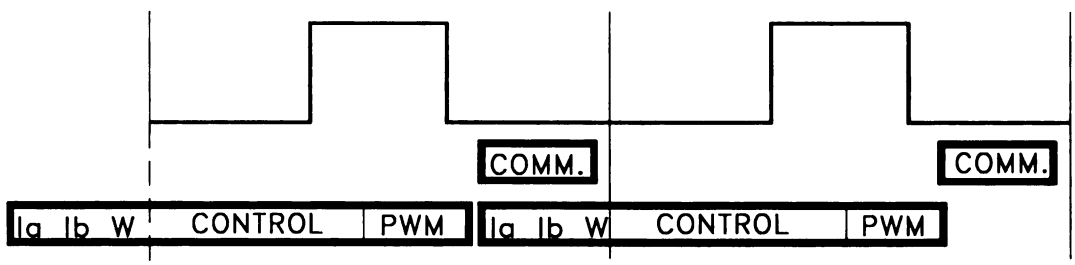

Figure 3.3 -DSP Timing

Chapter 4

FPGA IMPLEMENTATION OF SPWM

Half of the timing information, t_u, t_v, t_w and t_c is passed from the DSP to the FPGA. From these values, the FPGA calculates the other half of the timing information and produces the PWM pulse. The design of the FPGA PWM program consists of several key components, each of which will be described in detail.

An additional task assigned to the FPGA PWM program is to apply a time constraint on the PWM pulse's "on" and "off" times. These constraints are meant to meet inverter switching specifications.

4.1 FPGA - DSP COMMUNICATION

At the end of the control program the DSP initiates the communication with the FPGA by sending a load signal. Along with the load signal (**OLD**), four other signals are connected to the FPGA: the DSP communication clock (**OCK**) the serial output line (**DO**), the end load signal (**OSE**) and the serial input signal (**DI**). In Figure 4.1 the schematic of the FPGA program , the signals are seen as input pads.

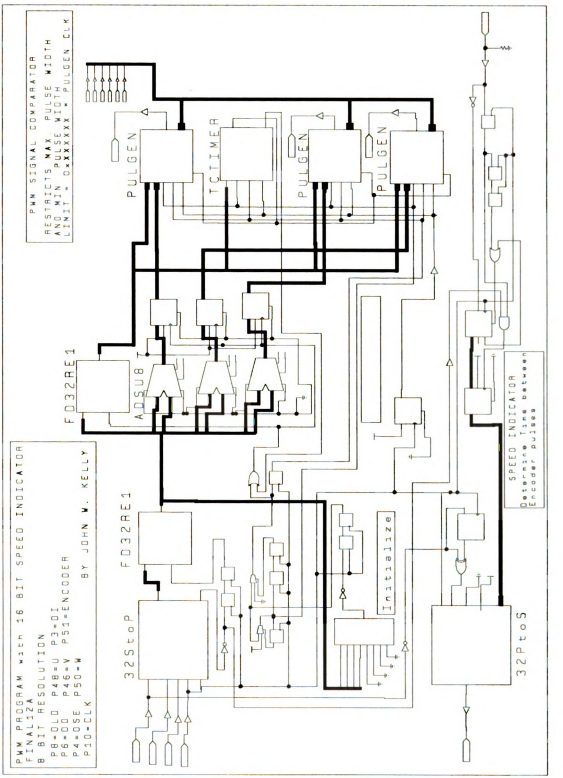

Figure 4.1 -The schematic of the FPGA program

4.2 FPGA PROGRAM OVERVIEW

From half of the SPWM pulse timing information t_c, t_u, t_v, and t_w, the FPGA calculates the other half of the timming information. First, $t_{u,v,w}$ is substracted from t_c. Next, t_c and (t_c-$t_{u,v,w}$) are doubled, simultaneously with the execution of the pulse. Due to the symmetry of the SPWM pulse, $2(t_c-t_{u,v,w})$ is the total "on" time of the pulse and $2t_c$ is the period of the pulse. After the time $t_{u,v,w}$ the SPWM pulse goes high and stays high for a time of $2(t_c-t_{u,v,w})$. After a time of $2t_c$ new timing information is loaded.

4.3 FPGA PROGRAM COMPONENTS

The *32STOP* component of the FPGA program is a passive thirty-two bit serial to parallel register. After the completion of the serial download, the thirty-two bit timing word is latched into a storage register, *FD32RE*. When the timing information is latched, the DSP post-processing of the timing data begins, allowing the DSP to restart the control program. First, t_u, t_v and t_w are subtracted from t_c by an eight bit high speed ripple-carry subtracter. This difference along with the original timing information is again latched and stored until needed by the *PULGEN* and *TCTIMER* components.

The *TCTIMER* component multiplies t_c by two, resulting in the SPWM pulse period. At the end of the pulse period, *TCTIMER* resets the counters of the *PULGEN* components. During the next clock pulse *TCTIMER* receives the new t_c and initiates the loading of the timing information to the three *PULGEN* components. Figure 4.2 shows

the schematic of the *TCTIMER*. The signal **CLR/LD** is an initialization signal. The signal **LD2** resets the counters of the *TCTIMER* and the *PULGENs* and triggers timing data latches. The *TCTIMER* has the capability to producing a pulse frequency of 100 kHz.

The *PULGEN* components produce the actual SPWM pulse. At the end of a pulse, the *PULGEN* components loads new timing information, $t_{u,v,w}$ and $(t_c\text{-}t_{u,v,w})$. A sixteen bit counter immediate begins counting to $t_{u,v,w}$. During the count of $t_{u,v,w}$, the signal $(t_c\text{-}t_{u,v,w})$ is doubled, resulting in the total "on" time of each SPWM pulse. When the count reaches $t_{u,v,w}$ the output of a sixteen bit magnitude comparator goes high. This signal enables a second counter that begins counting to $2(t_c\text{-}t_{u,v,w})$. When the count reaches $2(t_c\text{-}t_{u,v,w})$ the output of a second sixteen bit magnitude compartor goes high. Therefore, the SPWM pulse is an 'exclusive or' of the two magnitude comparators.

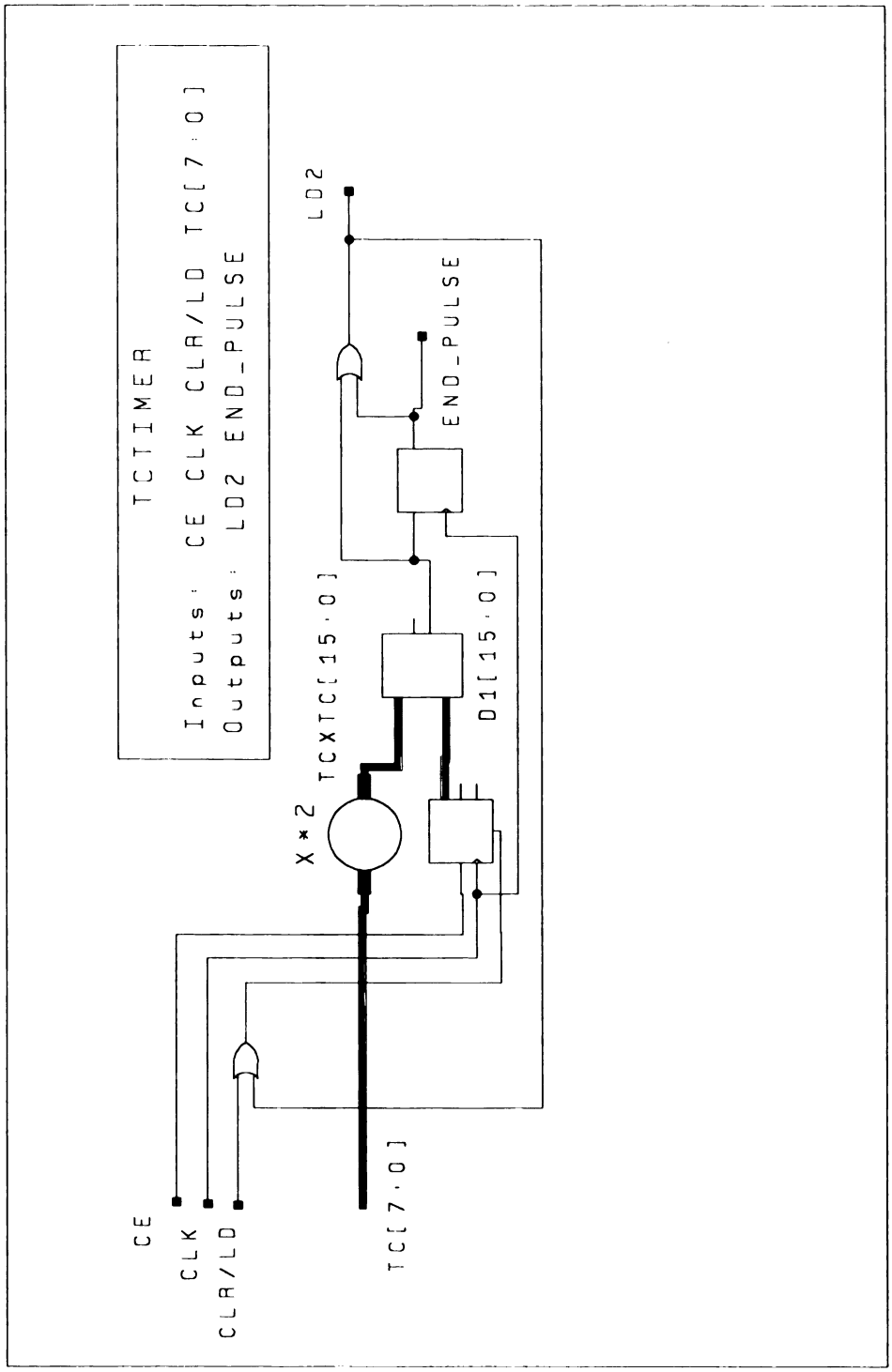

Figure 4.2 -*TCTIMER* component

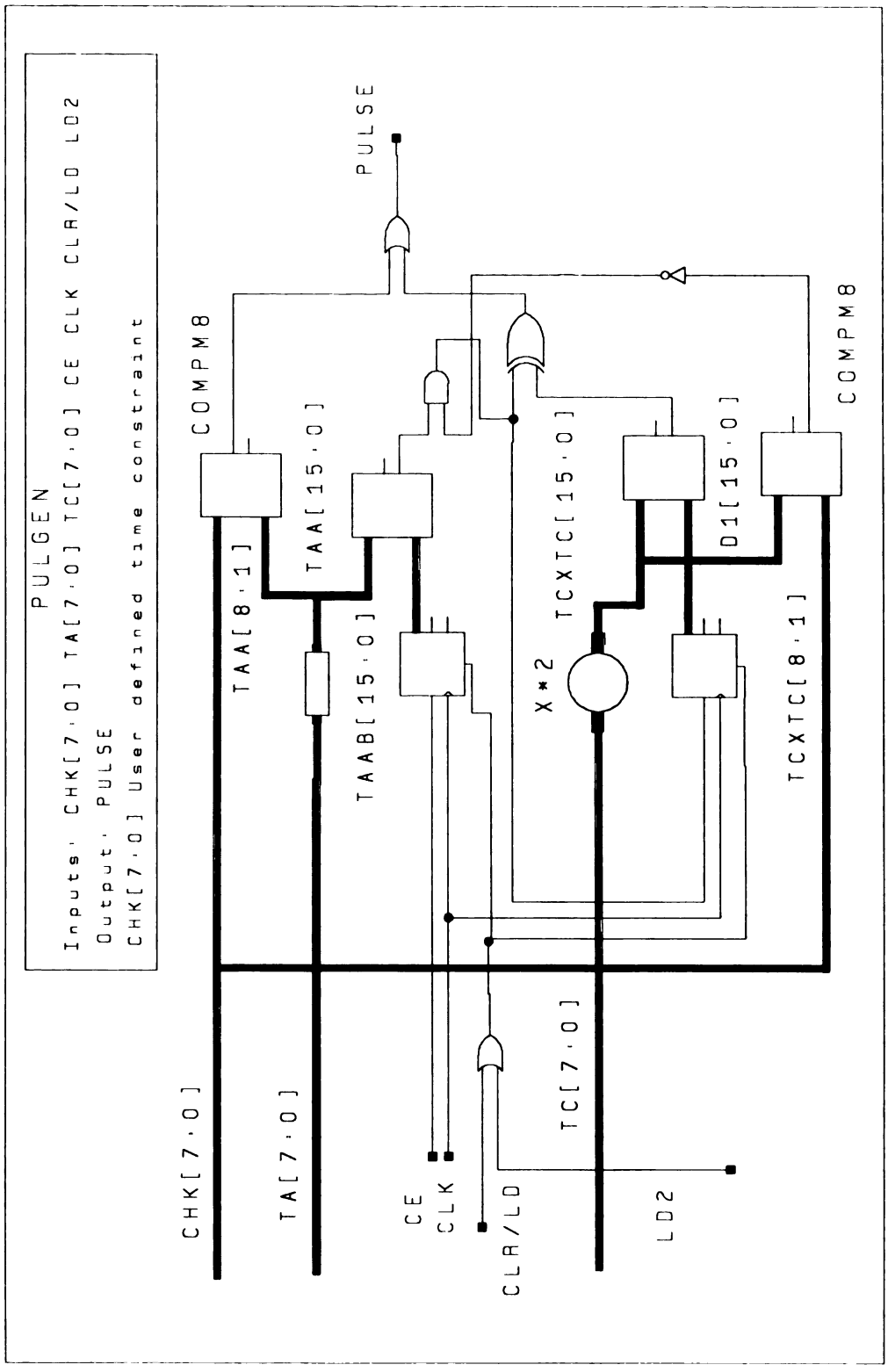

Figure 4.3 -*PULGEN* component

4.4 PULSE "ON/OFF" CONSTRAINTS

Figure 4.3 is the schematic of the *PULGEN* component. The load signal, **LD2,** generated by *TCTIMER* resets the counters. The signal **PULSE** is the SPWM pulse signal sent to the output pads of the FPGA. The *PULGEN* is capable of producing a pulse with "on" or "off" time of 2μs. However, the inverter may not be able to switch at this speed. The IGBTs of the actual inverter used were rated at a minimum of 5μs. Switching faster than the inverter specified rate may damage the IGBTs. Therefore, a constraint is placed on the minimal time of the "on" and "off" of the SPWM pulse, Figure 4.4. This constraint is implemented in *PULGEN* by evaluating the timing information. In Figure 4-3 there are two eight bit magnitude comparators that compares the times $2(t_c-t_{u,v,w})$ and $t_{u,v,w}$ to user defined times. If the "on" time is smaller than the user defined time, the SPWM pulse will remain low for the entire period. Likewise, if the "off" time of the SPWM pulse is smaller than the user defined time, the pulse will remain high for the entire period, meaning the inverter never switches states.

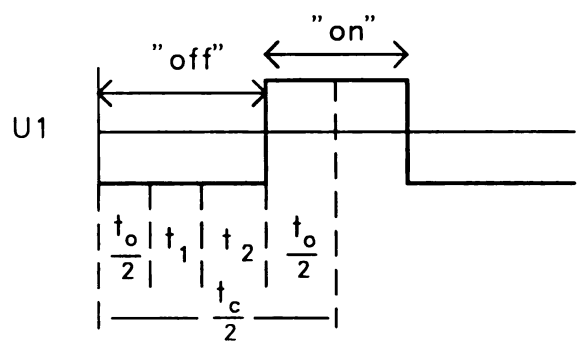

Figure 4.4 -SPWM pulse

Figures 4.5 and 4.6 show the actual PWM signal with pulse time constraints. In each of the plots the top signal (which is the same PWM phase signal) has an imposed time contraint, while the bottom signal is free of a time constraint.

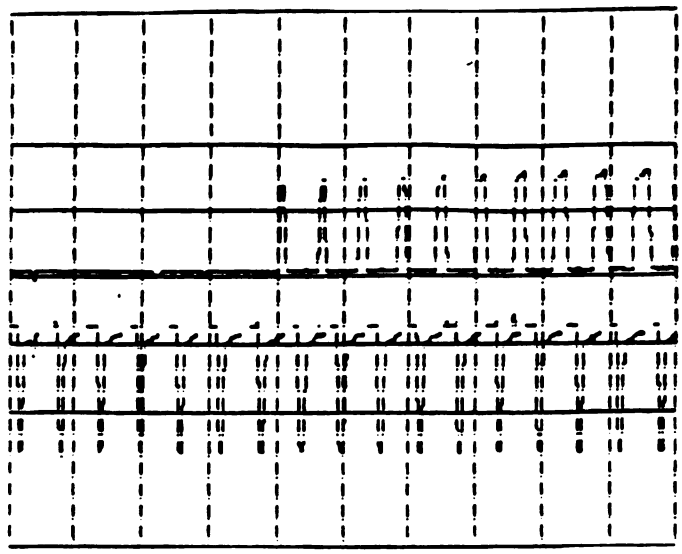

Figure 4.5 - The elimination of pulses with small "on" times

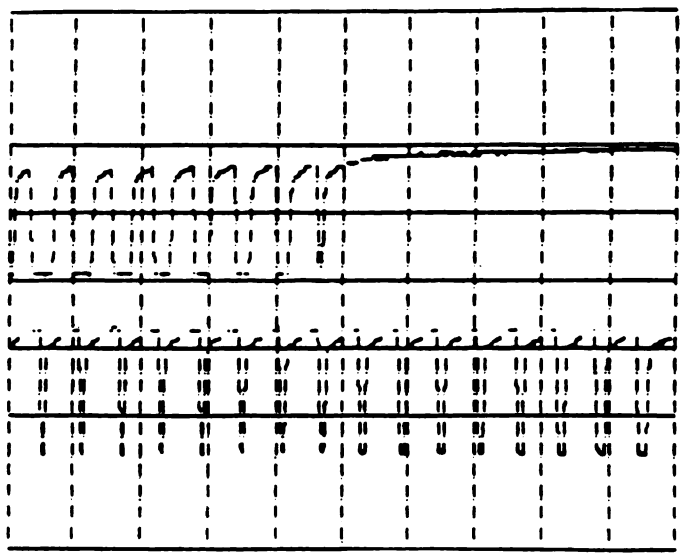

Figure 4.6 - The elimination of pulses with small "off" times

In order to determine the effects of placing "on" and "off" timing constraints on the SPWM pulse, the motor was run with no load at rated voltage and current. The switching frequency of the SPWM signal was 24.8 kHz. Figures 4.7,8,9 demonstrate the effects of placing time constraints on the pulse.

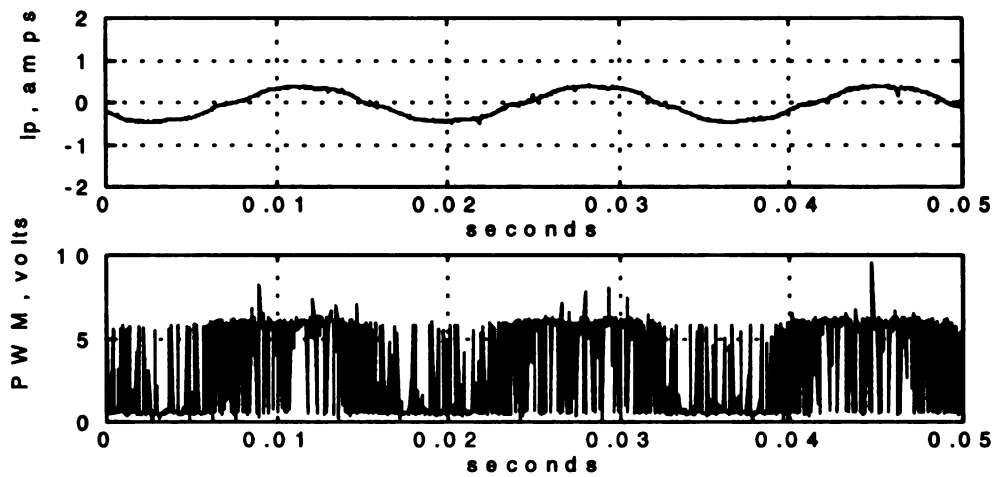

Figure 4.7 -Motor Phase Current and PWM Signal with 8 μs time Pulse Constraint

In Figure 4.7 the timing constraint was set at 8μs. However, the times t_u and $2(t_c-t_u)$ of the PWM signal were not less than 8μs. The phase current appears relatively sinusoidal with an amplitude of 0.7 amps.

Next, the pulse time constraint was increased to 9 µs. As seen in Figure 4.8, the pulse "on" and "off" constraints had an effect on the PWM signal. The modified PWM signal

introduces harmonics on the phase current. As the result of the harmonics, the motor draws more current.

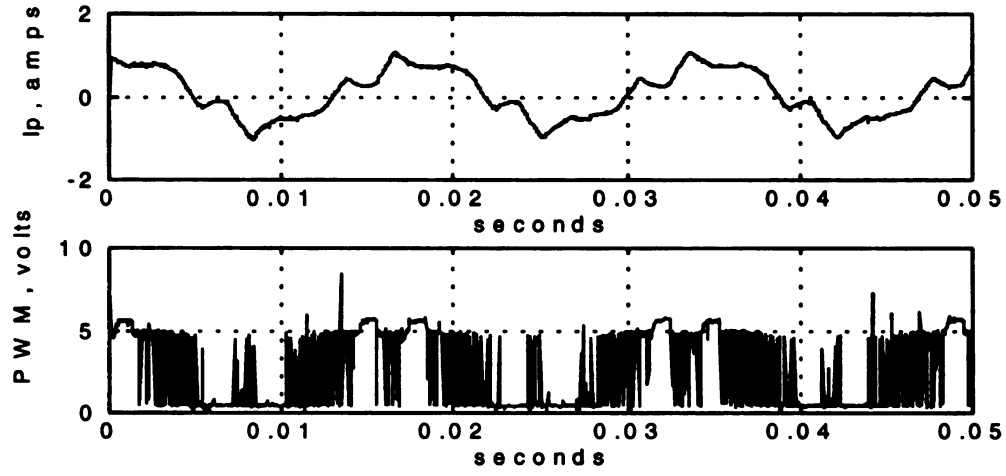

Figure 4.8 -Motor Phase Current and PWM Signal with 9μs Time Pulse Constraint

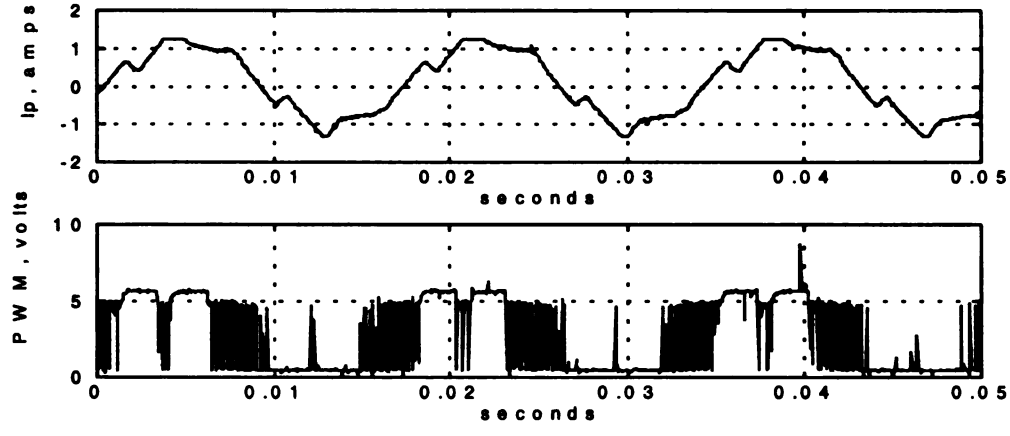

Figure 4.9 -Motor Phase Current and PWM Signal with 10μs Time Pulse Constraint

For implementing the control program, the speed of the motor was needed. From Figure 4.1, a counter is used to count the number of clock pulses between encoder pulses. This sixteen bit time is serially uploaded to the DSP simultaneous as timing information is downloaded.

The clock used to count out timing pulses is supplied by the DSP. The FPGA clock is fraction of the serial output clock, 3.25 MHz.

The total FPGA program requires 379 combinational logic block function generators, 15 I/O blocks and 297 combinational logic block flip-flops . The program was able to fit onto the Xilinx 4005 chip, which contains 5000 gates.

Chapter 5

INDIRECT ROTOR FLUX FIELD ORIENTATION

Due to the nonlinearity and mathematical complexity of an induction motor, the d.c. motor has been favored over the induction motor in terms of control response. However, it has been shown that by transforming the mathematical equations from the stator coordinates to rotor flux coordinates the equation of the induction motor become uncoupled. This simplification allows for an input-output relationship similar to the d.c. motor.

5.1 MATHEMATICAL DESCRIPTION OF INDUCTION MOTOR

The nonlinear differential equations which describe the behavior of an induction motor are:

$$R i_r + L_r \frac{di_r}{dt} + L_m \frac{d}{dt}\left(i_s \exp(-jP\varepsilon)\right) = 0 \tag{5-1}$$

$$R i_s + L_s \frac{di_s}{dt} + L_m \frac{d}{dt}\left(i_r \exp(jP\varepsilon)\right) = u_s \tag{5-2}$$

$$J \frac{d\omega_r}{dt} = m_{eL} - m_L = \frac{2}{3} P L_m \operatorname{Im}\left[i_s \left(i_r \exp(j\varepsilon) \right)^{*} \right] - m_L \tag{5-3}$$

$$\frac{d\varepsilon}{dt} = \omega_r \tag{5-4}$$

Where:

ε is the rotor position
ω_r is the rotor speed

ω_r is the rotor speed
m_{eL} driving torque
m_L load torque
J total inertia of drive
L_m mutual inductance
L_s stator inductance per phase
L_r rotor inductance per phase
R_s stator winding resistance per phase
R_r rotor winding resistance per phase
P pole pair
$\mathbf{i_s}$ space vector of stator current

$$i_s(t) = i_{s1}(t) + i_{s2}(t)e^{j2\pi/3} + i_{s3}(t)e^{j4\pi/3} \tag{5-5}$$

$\mathbf{i_r}$ space vector of rotor current

$$i_r(t) = i_{r1}(t) + i_{re}(t)e^{j2\pi/3} + i_{r3}(t)e^{j4\pi/3} \tag{5-6}$$

$\mathbf{u_s}$ space vector of stator voltage

$$u_s(t) = u_{s1}(t) + u_{s2}(t)e^{j2\pi/3} + u_{s3}(t)e^{j4\pi/3} \tag{5-7}$$

As seen in equations (5-1)-(5-3), there exists a complex relationship between the voltage input $\boldsymbol{u_s}$ and m_{eL} and ψ_r, electrically developed torque and rotor flux respectively. Because of this coupled and complex relationship between inputs and outputs of an induction motor, d.c. motors have been favored over induction motor in terms of output response. The relationship between input and output, I_a and m_{eL}, of a separately excited d.c. motor is

$$m_{eL} = k\, i_f\, i_a \tag{5-8}$$

5.2 SIMPLIFICATION OF TORQUE EQUATION

In 1971 Blaschke [7] showed that by a coordinate transformation to rotor field orientation, the input-output relationship of a induction can be made to emulate the input-output relationship of a separately excited d.c. motor. The rotor flux field is characterized by the space vector ψ_r, a sinusoidal flux wave rotating in the airgap. The rotor flux vector, ψ_r, can be represented by $i_{mr}(t)$, the magnetizing current.

$$i_{mr}(t) = \frac{\psi_r}{L_r} \tag{5-9}$$

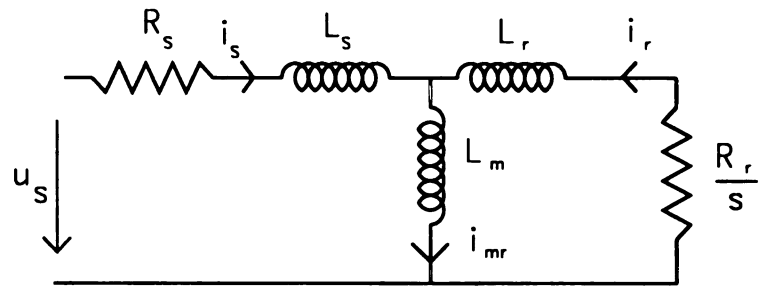

Figure 5.1 -Equivalent Circuit of induction motor

The $i_{mr}(t)$ vector rotates at angular speed of ω_{mr} relative to the stator reference frame.

$$\frac{d\rho}{dt} = \omega_{mr}(t) \tag{5-10}$$

In the rotor flux field coordinates, both the stator current and voltage space vectors, $\boldsymbol{i_s}(t)$ and $\boldsymbol{u_s}(t)$, can be decomposed into two orthongal parts: i_{sd}, i_{sq} and u_{sd}, u_{sq}.

$$i_s(t)e^{-j\rho} = i_{sd} + ji_{sq} \tag{5-11}$$

$$u_s(t)e^{-j\rho} = u_{sd} + ju_{sq} \tag{5-12}$$

As shown in Figure 5.2, the current i_{sd} is proportional to i_{mr}. In the rotor flux field orientation the developed torque is

$$m_{eL} = k\, i_{mr}\, i_{sq} \qquad k = \frac{2L_m}{3(1+\sigma_r)} \tag{5-13}$$

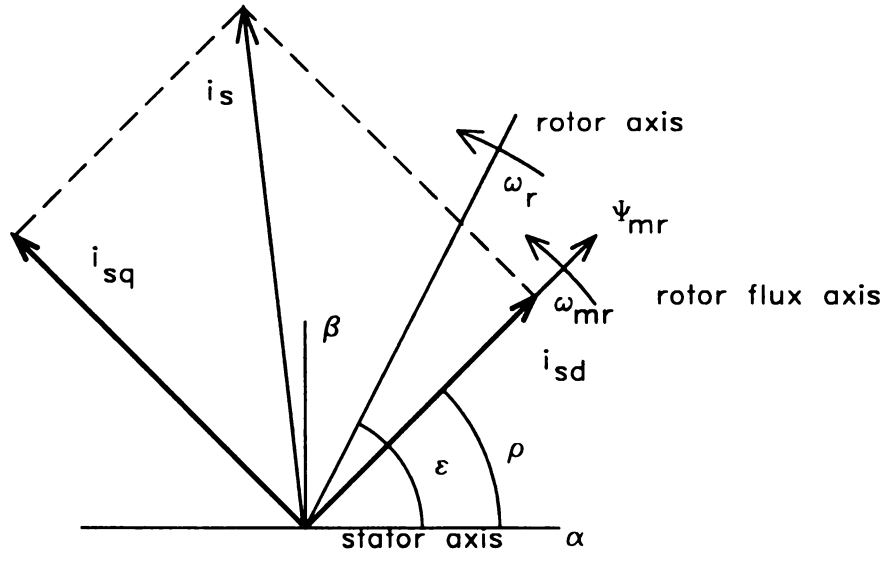

Fig 5.2 -Stator, Rotor and Rotor Flux Coordinates

The uncoupled dynamic equations of the induction motor in the rotor field orientation are:

$$\sigma_s \frac{L_s}{R_s}\frac{d(i_{sd})}{dt} + i_{sd} = \frac{u_{sd}}{R_s} + \omega_r \sigma \frac{L_s}{R_s} i_{sq} - (1-\sigma)\frac{L_s}{R_s}\frac{d(i_{mr})}{dt} \tag{5-14}$$

$$\sigma_s \frac{L_s}{R_s}\frac{d(i_{sq})}{dt} + i_{sq} = \frac{u_{sq}}{R_s} - \omega_r \sigma \frac{L_s}{R_s} i_{sd} - (1-\sigma)\frac{L_s}{R_s} i_{mr} \tag{5-15}$$

$$\frac{L_r}{R_r}\frac{d(i_{mr})}{dt} + i_{mr} = i_{sd} \tag{5-16}$$

$$\frac{d(\rho)}{dt} + \omega = \frac{R_r i_{sq}}{L_r i_{mr}} + \omega_r \tag{5-17}$$

$$J\frac{d(\omega_r)}{dt} = m_{eL} - m_l = k i_{mr} i_{sq} - m_l \qquad k = \frac{2L_m}{3(1+\sigma_r)} \tag{5-18}$$

From equation (5-13), the expression for the developed toque, appears similar to the torque expression of a separately excited d.c. motor, equation (5-8). Analogous to the d.c motor field current, i_{mr} is maintained at maximum level. Thus, the torque is controlled by i_{sq}.

Chapter 6

CONTROL SCHEME

Emulating the torque control scheme of separately excited d.c. motor, the rotor flux of the induction motor is maintained at maximum level, while i_{sq} is varied accordingly. Therefore, two feedback loops are maintained, the i_{sd} loop, which tracks a constant reference flux and the i_{sq} loop, which tracks torque and speed inputs. The output of the control loops are u_{sd} and u_{sq}. However, the first step of the control scheme is the transformation to rotor flux field coordinates.

6.1 TRANSFORMATION INTO ROTOR FIELD COORDINATES

As Shown in Figure 6.1, the motor stator currents are transformed from the three phase reference frame to the complex quadature reference frame.

$$i_{sc} = -i_{sa} - i_{sb} \tag{6-1}$$

$$\begin{bmatrix} i_\alpha \\ i_\beta \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \frac{1}{\sqrt{3}} & \frac{-1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} i_{sa} \\ i_{sb} \\ i_{sc} \end{bmatrix} \tag{6-2}$$

The complex quadature currents, i_α and i_β, along with the rotor speed ω_r are the inputs into a second order rotor flux observer.

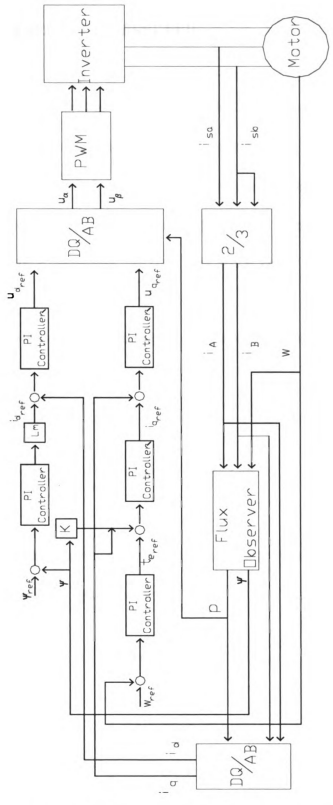

Figure 6.1 -Schematic of the Control Scheme [8]

6.2 SECOND ORDER FLUX OBSERVER

The key to the rotor field orientation control is knowledge of the position of the space vector ψ_r. With direct rotor field orientation control, Hall Effect probes measure the actual flux in the airgap of an induction motor. However, for indirect rotor field orientation control, the position of the rotor flux space vector is indirectly measured. One way to determine, indirectly, the rotor flux space vector is by implementing a rotor flux observer in the stator coordinates.

The rotor flux observer is described in the stator coordinates as

$$\frac{d(\psi'_a)}{dt} = -\frac{R_r}{L_r}\psi'_a - \rho\omega_r\psi'_b + \frac{R_r}{L_r}L_m i_a \tag{6-3}$$

$$\frac{d(\psi'_b)}{dt} = -\frac{R_r}{L_r}\psi'_b + \rho\omega_r\psi'_a + \frac{R_r}{L_r}L_m i_b \tag{6-4}$$

where ψ'_a and ψ'_b are estimates of rotor flux in the stator reference frame. The magnitude and angle of the rotor flux space vector are

$$\psi'_r = \sqrt{(\psi'_a * \psi'_a + \psi'_b * \psi'_b)} \tag{6-5}$$

$$\rho' = \arctan(\psi'_b/\psi'_a) \tag{6-6}$$

Defining the estimate error as $e_s = \dot{\psi}_r - \psi_r$, the error estimation is exponentially stable for a constant rotor resistance and has an upper ultimate bound for a nominal resistance [9].

With the estimated position of ψ_r, the complex quaduature stator currents can be transformed into i_{sd} and i_{sq}.

$$\begin{bmatrix} i_{sd} \\ i_{sq} \end{bmatrix} = \begin{bmatrix} \cos(\rho) & \sin(\rho) \\ -\sin(\rho) & \cos(\rho) \end{bmatrix} \begin{bmatrix} i_{sa} \\ i_{sb} \end{bmatrix} \tag{6-3}$$

The decoupled equation (5-13) and (5-14) are rewritten with the outputs in terms of u_{sd} and u_{sq}

$$u_{sd} = R_s i_{sd} + \sigma_s L_s \frac{d(i_{sd})}{dt} - \omega_r \sigma L_s i_{sq} + (1-\sigma) L_s \frac{d(i_{mr})}{dt} \tag{6-4}$$

$$u_{sq} = R_s i_{sq} + \sigma_s L_s \frac{d(i_{sq})}{dt} + \omega_r \sigma L_s i_{sd} + (1-\sigma) L_s i_{mr} \tag{6-5}$$

From equation (6-4), u_{sd} is denominated by i_{sd} Therefore, with a i_{sd} error as the input, a proportional and integral (PI) controller produces a reference u_{sq}.

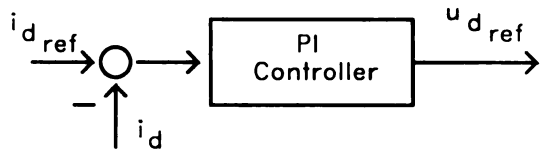

Figure 6.2 $-u_{sd}$, i_{sd} relationship

Similarly, the output u_{sq} is dominated by i_{sq} term.

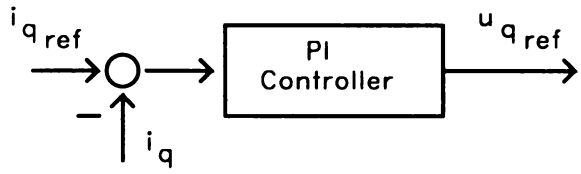

Figure 6.3 $-u_{sq}$, i_{sq} relationship

6.3 Transformation into Stator Coordinates

The voltage components u_{sq} and u_{sd} are transformed back into the stator reference frame by the inverse coordinate transformation.

$$\begin{bmatrix} u_\alpha \\ u_\beta \end{bmatrix} = \begin{bmatrix} \cos(\rho) & -\sin(\rho) \\ \sin(\rho) & \cos(\rho) \end{bmatrix} \begin{bmatrix} u_{sd} \\ u_{sq} \end{bmatrix} \tag{6-6}$$

Finally the complex quadauture stator voltage are transformed into three phase form.

$$\begin{bmatrix} u_{sa} \\ u_{sb} \\ u_{sc} \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ \frac{-1}{2} & \frac{\sqrt{3}}{2} \\ \frac{-1}{2} & \frac{-\sqrt{3}}{2} \end{bmatrix} \begin{bmatrix} u_{\alpha} \\ u_{\beta} \end{bmatrix} \tag{6-7}$$

Chapter 7

RESULTS

The purposed control scheme was simulated in Matlab in order to determine gain values for the PI controllers. Experimentally, a 1/12 hp induction motor was used to test the PWM and control program. A dc motor supplied the load to the induction motor to test speed control response.

7.1 INDUCTION MOTOR PARAMETERS

A small 1/12 horsepower induction motor was chosen in order to protect the inverter's IGBTs. If there was a problem with the program that caused a short across an IGBT leg, the lower current would be less likely to damage the IGBTs. The motor parameters were determined by measuring the power during a no-load test and a blocked rotor test.

Table 7.1 -Motor Parameters

3-∅ Induction Motor	ratings
horsepower	1/12
rated current	1.05 amp
rated voltage	115 V_{LL}
rated speed	1725 rpm
Rs	8 Ω
Rr	4 Ω
Ls	0.012885 h
Lr	0.012885 h
Lm	0.01885 h

7.2 SIMULATION OF CONTROL SCHEME

Simulations were done using Matlab. The motor, the control scheme and the flux observer were modeled in a continuous fashion. Therefore the model had a total of thirteen continuous states, see Appendix B for simulation program.

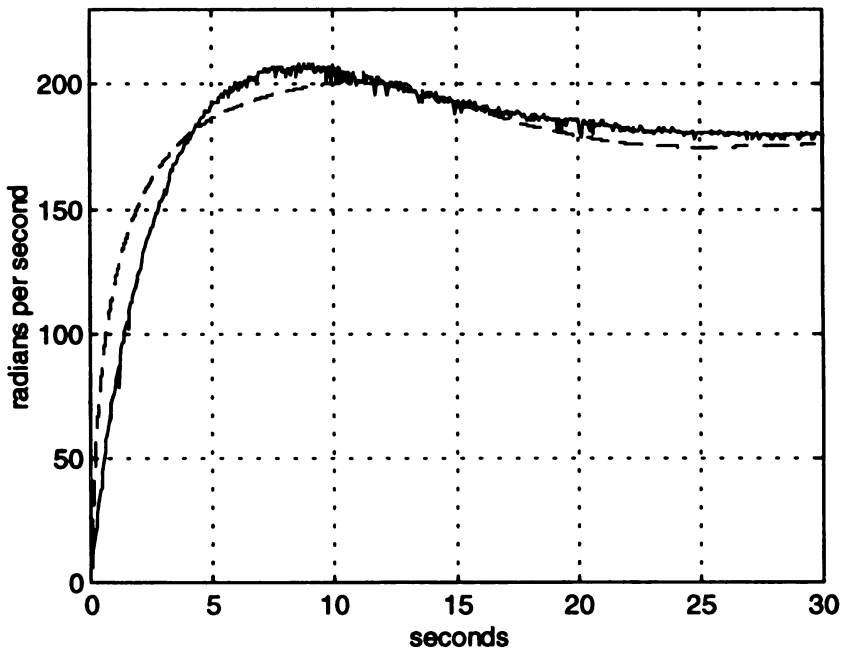

Figure 7.1 -Reference (dashed) and actual (solid) speeds with wref=180 rads/sec

7.3 EXPERIMENTAL SETUP

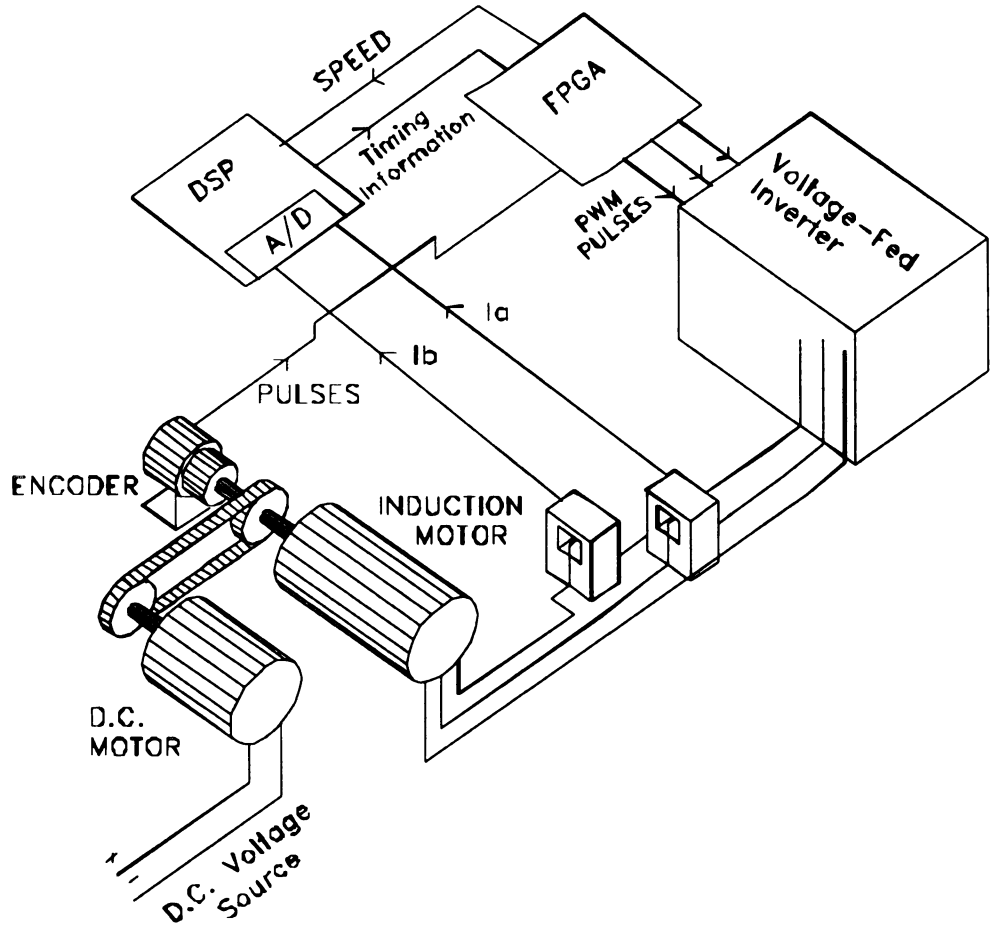

Figure 7.2 -Experimental Setup

A 40 volt separately excited d.c. motor acted (though a belt and pulley) as a load on the induction motor. Two current sensors measured the phase currents of the motor. The voltage-fed inverter was supplied by a 3-phase 208 V_{LL} source

7.4 SPEED REGULATION WITH DISTURBANCE

With switching frequency of 24kHz, the speed command for this experiment was 180 radians per second. Once the motor reached steady state a load was applied by the d.c motor. After retaining the command speed, the load was removed.

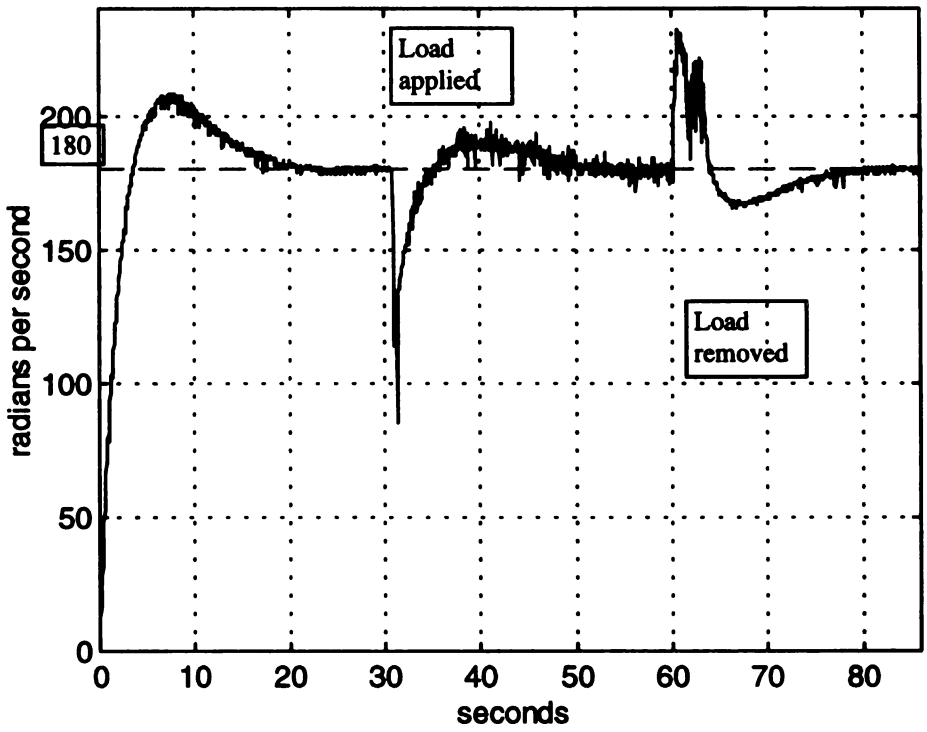

Figure 7.3 -W_{ref} = 180 rad/s with load applied and removed

7.5 STEP INPUT, SPEED COMMAND

In this experiment, the initial speed command was 180 radians per second. After reaching steady state the speed command (step input) became 40 radians per second. For the first run, no load was applied to the motor. In the second run, an initial load was applied to the motor. Again the switching frequency was 24kHz. The d.c. motor was supplied 40 volts. See Appendix A for program.

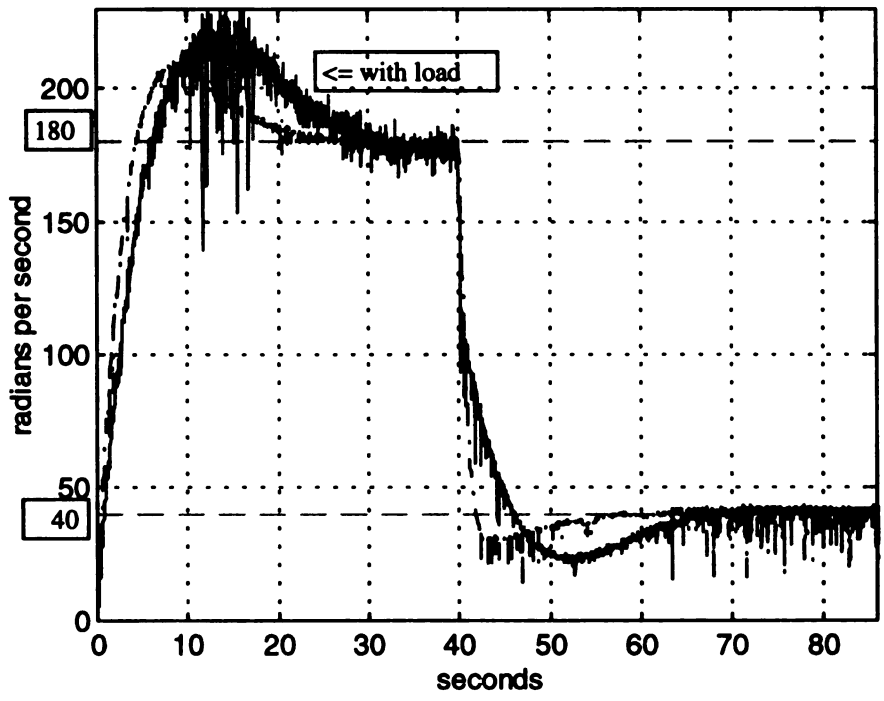

Figure 7.4 -Without load (dashed) and without load (solid)

7.6 FREE ACCELERATION

The gain values of the PI controller were adjusted to provide a fast response to a speed command, over a range of speeds from 20 to 220 radians/second. No load was applied.

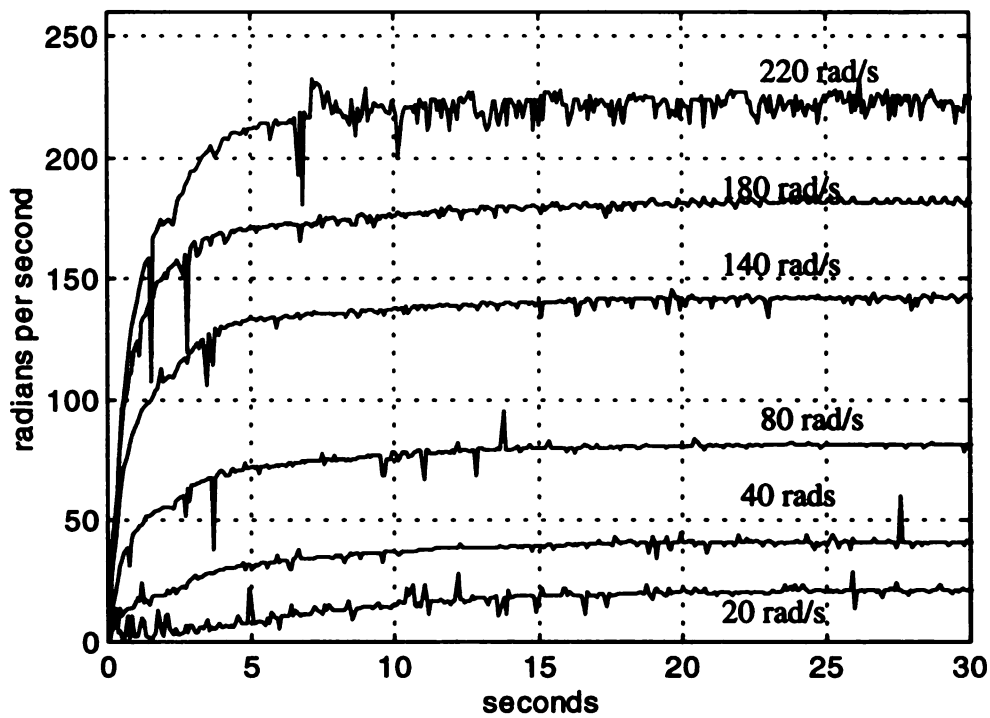

Figure 7.5 -Free Acceleration, w_{ref} 20-220 rads/sec

7.7 PROGRAM TIMING

Table 7.2 is an estimation of the total time of the DSP program. The total time is broken down between program parts. For example, the time of the DSP SPWM program is 9.7μs, which is 26.7 % of the total program time.

Table 7.2 -DSP Control and SPWM program timing specifications

Controller Program	Time approximation	
Stator current measurements	1.8 μs	5.0%
Speed measurements	4.5 μs	12.5%
3 - 2 Phase Transformation	1.0 μs	3.0%
ISD Loop		
Flux Observer	2.0 μs	5.5%
AB => DQ Coord. Transform.	6.3 μs	17.4%
Flux Controller	1.7 μs	5.0%
Isd Controller	1.2 μs	3.3%
ISQ Loop		
Speed Controller	1.3 μs	3.6%
Torque Controller	1.9 μs	5.2%
Isq Controller	1.4 μs	3.9%
VDQ => VAB	1.1 μs	3.0%
Storage	2.2 μs	6.1%
Controller Total	26.4 μs	73.1%
PWM Program	9.7 μs	26.7%
Total	36.1 μs	

The actual program time was 40.1 μs, because the IGBTs were rated at 25kHz switching frequency .

Table 7.3 is an estimation of the total time of the FPGA program. All timing simulations were conducted using Xilinx's ViewSim program. However, once the data is latched by

Table 7.3 -FPGA SPWM program timing specifications

FPGA Program	Time Approximation
8-bit Adder	14.5ns

the *FD32RE* latch (Figure 3.1) the most significant delay in the critical path delay is from the *Adder/Subtracter* component which perform the operation t_c-$t_{u,v,w}$. All other mathematical operations occur simultaneously with the outputting of the PWM pulse. Thus, relative to the DSP, the FPGA's time delays are insignificant.

Chapter 8

SUMMARY AND CONCLUSIONS

The main goal of this thesis was to explore the feasibility of implementing an induction motor control program and the Pulse Width Modulation program on separate dedicated hardware. The benefits of this hardware partitioning would be the computational alleviation of the control processor. Once freed of the responsibly of the PWM program, the control processor could implement more sophisticated control schemes or increase inverter switching frequency, for improved response.

Due to the memoryless and feedback independent characteristics of the PWM program, a field programmable gate array was chosen as the target hardware for PWM implementation. Although total program separation was found to be impractical, partial separation between the control program and PWM program significantly decreased processor computational time allowing for higher frequency inverter switching.

An indirect rotor flux field orientation control scheme was chosen to test the idea of program partitioning. Experimental results of Chapter 7 are discussed in this chapter.

An additional benefit of the PWM partitioning is the placement of time constraints on the PWM pulse without a computational time penalty. The time constraints on the PWM pulse ensures operation within inverter specification. However it was found that the modification to the PWM signal, from pulse time constraints, resulted in increased harmonic effect on stator currents.

8.1 CONTROL AND PWM PROGRAM HARDWARE ALLOCATION

Total implementation of the Space Vector Pulse Width Modulation algorithm on an FPGA was found to be impractical. Although the SPWM algorithm appears more comparable to a combinational logic circuit as opposed to the sequential circuit nature of the control program, the SPWM consists of several arithmetic operations which if implemented in a combinational fashion would not be cost effective. A FPGA CORDIC processor was developed to handle the trigonometric operations of the SPWM algorithm. However, the CORDIC algorithm and its controller consumed over eighty-five percent of the CLBs of the targeted FPGA with only eight bits of resolution. A trigonometric look-up table was not considered because handling an addressable matrix look-up table is more applicable to a microprocessor; thus creating a complex, two-microprocessor system. Therefore, the responsibility of the SPWM trigonometric operation was shifted back to the control processor. This shift eliminates the inherent redundancy in performing trigonometric calculations in a system with total hardware separation between control and PWM programs.

In a total hardware separated system the control processor must perform an arctangent operation to find the voltage space vector angle. The angle is passed to the FPGA, where the cosine of the angle is calculated. With the partially separated system the actual angle of the voltage space vector is never calculated, rather the cosine and sine of the angle is calculated with a simple multiplication operation, eliminating the timing consuming task of performing the arctangent operation.

With the partial PWM hardware implemented system, an inverter switching frequency of 24 kHz was achieved, while the applying an indirect rotor flux field orientation control scheme.

8.2 INDIRECT ROTOR FLUX FIELD ORIENTATION CONTROL

The proven indirect rotor flux field orientation control scheme was chosen to test the partial PWM hardware implemented system. The control scheme was first simulated in a continuous fashion, varying PI controller gain values for desired response. The actual control was written using AT&T DSP32C assembly code. The control program required three hundred lines of codes. In conducting experimental runs of Chapter 7, the focus was less on the actual control response but more on the feasibility of the overall system structure. However, the fact that the system delivered desired control response is a validation to the actual feasibility of the system.

8.2.1 SIMULATION VS. EXPERIMENTAL RESULTS

Simulation of the control scheme was conducted using MATLAB. Figure 7.1 shows the comparison between the actual response and simulated response of the system. The response profile of reaching the desired speed of 180 rad/s differs slightly between the actual and simulated results. This difference may be due to the variation in the measured motor parameters, implying the need for a more robust control scheme. Also,

the simulation did not take into account the inertia of the encoder, whose effect may not be insignificant for such a small motor.

8.2.2 EXPERIMENTAL RESULTS

Figure 7.3 shows the control program correcting for an applied load. The settling time is approximately seven seconds. After ten seconds the load is removed and again the controller performs a correction.

Figure 7.2 shows the controller responding to a speed change command. The set speed was changed from 180 to 40 rad/s. The first run was without a load. The settling time is approximately twelve seconds with an overshoot of 10 rad/s. For the second run an initial load was applied to the induction motor. As expected, the settling time of the speed change increased, approximately 23 seconds. There is increased noise on the speed signal of the second run. This is probably caused by the increased current which created increase noise on the speed sensor.

Figure 7.5 shows the speed range capability of the controller. The speed PI Controller gains k_i and k_p were increased for faster response. No load was applied. The minimum speed obtained was 20 rad/s.

8.3 SPWM PULSE TIME CONSTRAINTS

Imposing time constraints on the PWM pulse results in a distorted PWM signal which, in turn, results in distorted sinusoidal voltage waveforms. However, for increased switching frequencies, pulse constraints are especially necessary. Faster switching results in greater occurrences of smaller pulse "on/off" times, putting the inverter,s IGBTs more at risk. Decreasing modulation can also act as a pulse time constraint. However, maximum voltage utilization from the inverter is usually desired. Due to the lack of a computational time penalty, imposing time constraints via the FPGA SPWM program is desirable.

8.4 FUTURE DIRECTION

Assuming increasing computational speeds and decreasing cost of processors, full implementation of a PWM algorithm on a FPGA should not be pursued even with the introduction of FPGA dsp tools.

The idea of varying the switching frequency should be explored. The primary disadvantage of increased switching frequency is increased switching power loss. An optimal scheme would have switching frequency increase for transient response and have the switching frequency decrease for steady state operation. The DSP SPWM developed for this thesis has the ability to change t_c, the switching period, from pulse to pulse.

APPENDIX A

APPENDIX A

The DSP Code

The DSP Code is written in AT&T's DSP32c Assembly language. The program is composed of the indirect rotor flux orientation control program and the partial SPWM program.

```
/*************************************************************************/
/* INDIRECT ROTOR FLUX FIELD ORIENTATION CONTROL PROGRAM                 */
/*                  WITH PARTIAL SPWM PROGRAM    (cc18.s )               */
/*                                  by                                   */
/*                              John W. Kelly                            */
/* Copyright 1997                                                        */
/*************************************************************************/

        #define   ivtpe      r22e

          .global __start
          .global __control
          .global __end
          .global __intr

          .rsect  ".start"
          goto    __start
          nop

          .rsect  ".prog"
          .align  4

__start:  ivtpe = itable
          ioc = 0x020fd3          /* Disable any Serial DMAs */
                                  /* 32 bit word output      */
                                  /* output rate 6.25 Mhz    */
                                  /* OLD out                 */
                                  /* ILD in                  */
                                  /* OCK out */

         r1  = 0x4432
          pcw = r1

          r2e = 0x200008
          r1  = 0xfe66      /* interupt time  41us*/
          *r2 = r1
          nop
          r13e = 0x60000



/* ========================= Start Loop ========================== */
/* ------------------------------------------------------------------ */
/* ------------------------------------------------------------------*/
__control: r2e = _gg        /*                                       */
          r3e = _pos1       /*                                       */
          *r2 = a1 = *r3    /*                                       */
          nop
          nop
/* ================================================================ */
```

```
/*                                                                    */
/* ================ Stator Current Measurments ====================*/
/* ------------------------------------------------------------------*/
/* measurment of stator current of phases A and B                     */
/* calculation of stator current in phase C                           */
/*                                                                    */
/* fast LEM current sensors                                           */
/*                                                                    */
/* cur_a = float(channel A)*f_in*f_cur + offset                       */
/* cur_b = float(channel B)*f_in*f_cur + offset                       */
/* cur_c = - cur_a - cur_b                                            */
/* ------------------------------------------------------------------*/
        r1e = 0x200000        /* analog channel A                     */
        r2e = 0x200004        /* analog channel B                     */
        r3e = f_in            /* 1V * f_in = 1.0                      */
        r4e = cur_a           /* current phase A                      */
        r5e = cur_b           /* current phase B                      */
        r6e = cur_c           /* current phase C                      */
       a0 = float(*r1)        /* read analog channel A                */
        a1 = float(*r2)       /* read analog channel B                */

        r7e = f_ino
        a0 = *r7 + a0 * *r3
        a1 = *r7 + a1 * *r3

        r9e = offs_2
        r8e = f_cur2
        *r4 = a0 = *r9 + a0 * *r8    /* store cur_a                   */
        *r5 = a1 = *r9 + a1 * *r8    /* store cur_b                   */
        r7e = loops
        nop
        *r4 = a0 = a0 * *r7
        *r5 = a1 = a1 * *r7
        nop
        *r6 = a2 = -a1 - a0    /* store cur_c                         */
        nop
        nop
/* 23 */
/* ================================== ========================== */
/*                                                                    */
/* ======================= Speed Measurement ====================== */
/* ------------------------------------------------------------------ */
/* The number of clock tics between speed encoder pulses is           */
/* continuously uploaded to the DSP. The resolution of the encoder    */
/* is 1024 pulses per rev. The Speed Equation:                        */
/*       wr = (2*pi/1024)*(clk freq./ clk tics)                       */
/* ------------------------------------------------------------------ */
          a3 = float(ibuf)      /* read serial buffer                 */

          r1e = _wr_new
```

```
        r2e = _wr_oldh      /* Upper speed limit                          */
        r4e = _wr_old       /*                                          */
        r3e= _zero
        a2 = *r3
        if (ale) goto __SpdZero   /* count = zero                         */
        a1 = a2                   /* speed = zero                       */

        r2e = _2n           /* dummy varible                              */
        *r2 = a2 = seed(a3) /* -- Inversion Routine: _invf --             */
        r3e = _invfa        /* Source: AT&T Appl. Soft. Lib.              */
        nop                 /* replace nop  of _invf                      */
        a0 = *r3++ - a3 * a2  /* _invf                                    */
        a1 = *r2 * *r3++      /* _invf                                    */
        a2 = *r3++ - a3 * *r2 /* _invf                                    */
        a0 = *r3++ +  a0 * a0 /* _invf                                    */
        a1 = a0 * a1          /* _invf                                    */
        a2 = a2 * a1          /* _invf                                    */
        nop                 /* replace return(r14) of _invf               */
        a0 = a2 + a0 * a1     /* _invf                                    */

        r2e = _wr_oldh      /* (nop) Upper speed limit                    */
        r4e = _wr_old       /* (nop)                                    */

        r6e = _2pi_fclk     /* (nop) (2*pi/1024)*6.25e6                   */
        *r1 = a1 = a0 * *r6 /* (2xpi/1024)x6.25e6/_epulse_new              */
        nop

__SpdZero:
        a2 = *r4            /* <-|                                      */
        r3e = _wr_oldl      /*   |     Lower speed limit                 */
        a0 = *r2 - a1       /*   |<-|  Upper limit - speed               */
        a3 = *r3 - a1       /*   |  |<-| Lower limit - speed             */
        if (aeq) goto __LL  /* <-|  |  | WrOld = 0                        */
        r6e = _spd_tol      /*    |  | Speed tolerance                   */
        if (alt) goto __OldSpd /*<-|  | Upper limit-speed<0               */
        nop  /* a2=a0+a1 */
__LL:   nop                 /*       |                                  */
        nop
        if (agt) goto __OldSpd /*   <-| Lower limit-speed>0               */
         nop /* a2=a3+a1*/
        nop
        nop
        nop
        a2 = *r1
        nop
__OldSpd: nop
        nop
        nop
        *r1 = a1= a2
        nop
```

```
        nop
        *r2 = a3 = a1 + *r6 * a1   /* Upper speed limit                    */
        *r3 = a0 = a1 - *r6 * a1   /* Lower speed limit                    */
        nop
        nop
        *r4 = a1 = a2          /* WrOld = WrNew                            */
        nop
/* 56 */
/* ================================================================== */
/*                                                                    */
/* ======================= 3=>2 Transformation ======================= */
/* ------------------------------------------------------------------ */
/*          3-phase to 2-phase transformation                         */
/*                                                                    */
/*          ia = 2/3*cur_a - cur_b/3 - cur_c/3                        */
/*          ib = 1/sqr(3)*(cur_c - cur_b)                             */
/* ------------------------------------------------------------------ */
        r1e = _isa_new          /* stator current  q_axis             */
        r2e = _isb_new          /* stator current  d_axis             */
        r3e = _p_fac            /* 2/3   1/3   1/sqr(3)               */
        r4e = cur_a             /* current phase A                    */
        r5e = cur_b             /* current phase B                    */
        r6e = cur_c             /* current phase C                    */

        a0 = *r3++ * *r4
        a0 = a0 - *r3 * *r5
        *r1 = a0 = a0 - *r3++ * *r6   /* ia=2/3*a - b/3 - c/3         */
        a1 = *r3 * *r6
        *r2 = a1 = a1 - *r3 * *r5     /* ib=1/sqr(3)*(c-b)            */
        nop
        nop
/* 13 */
/* ================================================================== */


/* ========================== ISD LOOP ============================== */
/*                                                                    */
/* ======================== Flux Observer =========================== */
/* psiahdNew = -ITr*psiahNew - p*WrNew*psibhNew + ITr*Lm*IsaNew       */
/* psibhdNew = -ITr*psibhNew + p*WrNew*psiahNew + ITr*Lm*IsbNew       */

        r3e = _pole
        r7e = _wr_new

        a1 = *r7 * *r3          /*  pole*WrNew                        */

        r4e = _Ybh_new
        r5e = _Yah_new

        a2 = -a1 * *r4          /* pole*WrNew*YbhNew                  */
```

```
        a3 = a1 * *r5          /* pole*WrNew*YahNew                              */
        r3e = _ITr

        a1 = a2 - *r3 * *r5    /* -pole*WrNew*YbhNew - ITr*YahNew                */
        a2 = a3 - *r3 * *r4    /* +pole*WrNew*YahNew - ITr*YbhNew                */

        r3e = _c1              /* ITr*Lm                                         */
        r5e = _isa_new
        a3 = *r5 * *r3         /* (ITr*Lm)*Isanew                                */


        r4e = _isb_new
        a0 = a3 + a1
                    /* -pole*WrNew*YbhNew-ITr*YahNew+(ITr*Lm)*Isanew   */

        a1 = *r4 * *r3          /* (ITr*Lm)*Isbnew                               */
        nop

        a3 = a1 + a2
                   /* +pole*WrNew*YahNew-ITr*YbhNew+(ITr*Lm)*Isbnew   */

        r3e = _time

        a1  = *r3 * a0         /* HT*(YahdNew)                                   */
        a2  = *r3 * a3         /* HT*(YbhdNew)                                   */

        r4e = _Yah_new
        *r4 = a3 = a1 + *r4    /* Yahold+HT*(YahdNew)                            */

        r5e = _Ybh_new
        *r5 = a1  = a2 + *r5   /* Ybhold+HT*(YbhdNew)                            */
        nop
        nop
/* 25 */
/* ================================================================== */
/*                                                                    */
/* ============Coordinate Transformation  AB => DQ ================== */
/* ------------------------------------------------------------------ */
/* ------------------------------------------------------------------ */
        a2 = a3 * a3           /* Yahnew^2                                       */
        a0 = a1 * a1           /* Ybhnew^2                                       */
        nop

        r1e = _invsqr_in       /*                                                */
        *r1 = a1 = a0 + a2     /* Yd^2 = Yah^2 + Ybh^2                           */

        call _invsqr (r21)     /*  Call Inverse of Square Root                   */
        nop
        r9e = _invsqr_out      /*  1/(Yd)^.5                                     */
        nop
```

```
        a3  = *r9
        r2e = _2n              /* dummy varible                                */
        *r2 = a2 = seed(a3)    /* -- Inversion Routine: _invf --               */
        r3e = _invfa           /* Source: AT&T Appl. Soft. Lib.                */
        nop                    /* replace nop  of _invf                        */
        a0 = *r3++ - a3 * a2   /* _invf                                        */
        a1 = *r2 * *r3++       /* _invf                                        */
        a2 = *r3++ - a3 * *r2  /* _invf                                        */
        a0 = *r3++ +  a0 * a0  /* _invf                                        */
        a1 = a0 * a1           /* _invf                                        */
        a2 = a2 * a1           /* _invf                                        */
        r7e = _Yd_new          /* replace return(r14) of _invf                 */
        a0 = a2 + a0 * a1      /* _invf                                        */

        r2e = _Yah_new
        r3e = _Ybh_new
        r4e = _cos_pr
        r5e = _sin_pr

        *r7 = a3 = a0          /* Store Yd                                     */

        *r4 = a1 = *r2 * *r9   /* cosPr = Yahnew / Yd                          */
        *r5 = a2 = *r3 * *r9   /* sinPr = Ybhnew / Yd                          */

        r2e = _isa_new
        r3e = _isb_new

        a0 = *r2 * a1          /* IsaNew * cosPr                               */
        a3 = *r3 * a2          /* IsbNew * sinPr                               */
        nop

        r6e = _isd_new
        *r6 = a0 = a3 + a0     /* IsqNew=IsaNew*cosPr+IsbNew*sinPr             */
        nop

        a3 =  a2 * *r2         /* -IsaNew * sinPr                              */
        a0 =  a1 * *r3         /* IsbNew * cosiPr                              */


        r7e = _isq_new
        *r7 = a2 = a0 - a3     /* IsdNew=-IsaNew*sinPr+IsbNew*cosPr            */
        nop
        nop
/* 45 + 34*/
/* ============================================================== */
/*                                                                */
/* ====================== Flux Controller ======================= */
/* -------------------------------------------------------------- */
/* -------------------------------------------------------------- */
```

```
        r2e = _Yd_new
        r3e = _Yd_ref_new

        a0 = *r3 - *r2          /* YdError = YdRefNew - YdNew                    */
        nop

        r5e = _time
        a2 = a0 * *r5           /* H*YdError                                     */

        nop
        r6e = _Yd_int_old
        *r6 = a1 = a2 + *r6     /*YdIntOld = YdIntOld+H*YdError                  */

        r2e = _isd_kp
        a2 = *r2 * a0           /* YdError * Kp                                  */

        r3e =  _isd_ki
        a3 = a1 * *r3           /* YdIntNew * Ki                                 */
        nop

        a0 = a3 + a2

        r4e = _ILm              /* 1/Lm                                          */
        r5e = _isd_ref_new
        *r5 = a1 = a0 * *r4     /* Yd/Lm = Imr                                   */
        nop
        nop
        nop
/* 21 */
/* ================================================================== */
/* ================== Isd Current Controller ======================== */
/* ------------------------------------------------------------------ */
/* ------------------------------------------------------------------ */

        r2e = _isd_new
        r3e = _isd_ref_new

        a0 = *r3 - *r2          /* IsdError = IsdRefNew - IsdNew                 */
        nop

        r5e = _time
        a2 = a0 * *r5           /* H*IsdError                                    */

        r6e = _isd_int_old
        *r6 = a1 = a2 + *r6     /*IsdIntOld = IsdIntOld+H*IsdError               */

        r2e = _usd_kp
        a2 = *r2 * a0           /* IsdError * Kp                                 */
```

```
        r3e = _usd_ki
        a3 = a1 * *r3          /* IsdIntNew * Ki                                  */

        r4e = _usd_ref
        *r4 = a0 = a3 + a2
        nop
/* 15 */
/*================================================================== */
/*                                                                    */
/* ========================== ISQ LOOP ========================== */
/*                                                                    */
/* ======================= Speed Controller ======================= */
/* ------------------------------------------------ ---------------- --*/
/* ------------------------------------------------------------------ */
        r2e = _wr_new
        r3e = _wr_ref

        a0 = *r3 - *r2         /* WrError = WrRef-WrNew                           */
        nop

        r5e = _time
        a2 = a0 * *r5

        r6e = _wr_int_old
        *r6 = a1 = a2 + *r6  /*WrIntOld=WrIntNew=WrIntOld+H*WrErrorNew      */

        r2e = _te_kp
        a2  =  *r2 * a0        /* WrError * Kp                                    */

        r3e = _te_ki
        a3  = a1 * *r3         /* WrIntNew * Ki                                   */

        r4e = _te_ref_new
        *r4 = a0 = a3 + a2     /*TeRefNew=WrErrorNew*Kp+WrIntNew*Ki          */
        nop
        nop

/*16 */
/* ================================================================== */
/* ====================== Torque Controller ====================== */
/* ------------------------------------------------------------------ */
/*            te = 1.5*p*Lm*Yd*Isq/Lr                                 */
/* ------------------------------------------------------------------ */
        r2e = _isq_new
        r3e = _Yd_new
        a0 =  *r2 * *r3
        nop

        r4e = _k               /* 1.5*Pole*Lm                                     */
        a1 = a0 * *r4          /* TeNew = 1.5*p*Lm*Yd*Isq/Lr                      */
```

```
        nop
        r3e = _te_ref_new
        a0 = *r3 + a1           /* TeError = TeRef-TeNew                          */
        nop

        r5e = _time
        a2 = a0 * *r5
        nop

        r6e = _te_int_old
        *r6 = a1 = a2 + *r6     /*TeIntOld=TeIntNew=TeIntOld+H*TeErrorNew        */

        r2e = _isq_kp
        a2  =  *r2 * a0         /* TeError * Kp                                   */

        r3e = _isq_ki
        a3  = a1 * *r3          /* TeIntNew * Ki                                  */

        r4e = _isq_ref_new
        *r4 = a0 = a3 + a2      /*IsqRefNew=TeErrorNew*Kp+TeIntNew*Ki            */
        nop
        nop
        nop
/* ================================================================== */
/* ================== Isq Current Controller ========================= */
/* ------------------------------------------------------------------- */
/* ------------------------------------------------------------------- */

        r2e = _isq_new
        r3e = _isq_ref_new
        r7e = _isq_error
        *r7 = a0 = *r3 - *r2    /* IsqError = IsqRefNew - IsqNew                  */
        nop

        r5e = _time
        a2 = a0 * *r5           /* H*IsqError                                     */

        r6e = _isq_int_old
        *r6 = a1 = a2 + *r6     /* IsqIntOld=IsqIntOld+H*IsqError                 */

        r2e = _usq_kp
        a2 = *r2 * a0           /* IsqError * Kp                                  */

        r3e =  _usq_ki
        a3 = a1 * *r3           /* IsqIntNew * Ki                                 */

        r4e = _usq_ref
        *r4 = a0 = a3 + a2
        nop
```

```
        nop
        nop
/* 18 */
/* ============================================================== */
/*                                                                */
/* ============================================================== */
/*                                                                */
/* ============== Voltage Coordinate Transformation ============= */
/* -------------------------------------------------------------- */
/*          Vsa = UsdRef*cos_pr-UsqRef*sin_pr                     */
/*          Vsb = UsqRef*cos_pr+UsdRef*sin_pr                     */
/* -------------------------------------------------------------- */

        r2e = _usq_ref
        r3e = _sin_pr
        a0 = *r2 * *r3          /* UsqRef * sin_pr                */

        r4e = _usd_ref
        r5e = _cos_pr
        a1 = *r4 * *r5          /* UsdRef * cos_pr                */

        a2 = *r3 * *r4          /* UsdRef * sin_pr                */

        a3 = *r5 * *r2          /* UsqRef * cos_pr                */
        r7e = _Vsa
        *r7 = a0 = a1 - a0      /*Vsa = UsdRef*cos_pr-UsqRef*sin_pr */
        r6e = _Vsb
        *r6 = a2 = a3 + a2      /*Vsb = UsqRef*cos_pr+UsdRef*sin_pr */
        nop
        nop       /* ? */
/* 14 */
/* ============================================================== */
/*                                                                */

/* ===================== Store Speed & Current ================== */
        r3e = _done
        a1 = *r3
        r7e = _wr_new
        r4e = _count
        r5e = _pos1             /* initial = 1 then =0            */
        *r4 = a2 = *r4 + *r5
        r2e = _event
        if (agt) goto __ok
        nop
        a3 = *r2 - a2
        nop
        r6e = _storcnt
        r3e = _zero
        if (agt) goto __ok
        nop
```

```
        *r6 = a2 = *r6 + *r5
        *r13++ = a1 = *r7
        nop
        r9e = _stor
        a3 = *r9 - a2
        nop
        *r4 = a0 = *r3
        r6e = _done
        if (agt) goto __ok
        nop
        *r6 = a1 = *r5          /* _stpos1 = 0                                  */

__ok:   nop
/* 28 */
/* ========================= Start PWM Program ========================= */
/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------ */
        r6e = _Vsa
        a0 = *r6

        r7e = _Vsb
        a2 = *r7
        nop

        a3 = a0 * a0            /* Va^2                                     */
        a1 = a2 * a2            /* Vb^2                                     */

        nop
        r1e = _invsqr_in        /* Vd                                       */
        *r1 = a2 = a3 + a1      /* _Vd^2 = Va^2 + Vb^2                      */
        call _invsqr (r21)      /*                                          */
        nop                     /*                                          */
        r1e = _invsqr_out       /*                                          */
        r2e = _Vsa              /*                                          */
        r3e = _cosV             /*                                          */
        *r3++ = a1 = *r1 * *r2 /* cos(a) = Vsa/Vd  pt to _sinV               */
        r2e = _Vsb              /*                                          */
        *r3 = a2 = *r1 * *r2    /* sin(a) = Vsb/Vd                          */
        nop
/* 12+34 = 45 */
/* ====== Calculate tc and factor which includes voltage modulation ========*/
/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------ */
        r1e = _fout             /*                                          */
        r3e = _2n_fclk          /* samples/fclk                             */
        a3 = *r1 * *r3          /* fout*samples/fclk                        */
        r1e = _invsqr_out       /*                                          */
        r5e = _cosV             /* for "Determine Sector" was nop           */
        a3 = a3 * *r1           /* fout*samples/fclk/Ud                     */
        r2e = _2n               /*                                          */
```

```
        *r2 = a2 = seed(a3)   /*  INVERSION OF PREV. LINE                */
        r3e = _invfa          /*                                         */
        r6e = _sinV           /* for "Determine Sector" was nop          */
        a0 = *r3++ - a3 * a2  /*                                         */
        a1 = *r2 * *r3++      /*                                         */
        a2 = *r3++ - a3 * *r2 /*                                         */
        a0 = *r3++ +  a0 * a0 /*                                         */
        a1 = a0 * a1          /*                                         */
        a2 = a2 * a1          /*---------------------------------------*/
        r4e = _tc             /* point to _tc                            */
        r3e = _factor         /* r3 point to _factor                     */
        a0 =a2+a0*a1          /* store  _tc                              */
        r1e = _invsqr_out     /* r1e points to 1/Ud                      */
        r8e = _pwm            /* r8 points tp _pwm                       */
        *r4 = a3 = a0 * *r1   /* a2 = tc*fclk*Ud/Ud                      */
        r11e = _m             /* r5 pts to 1/(Uref*sin(pi/3)/.86         */
        *r3 = a0 = a0 * *r11  /* tc*fclk/(Uref*sin(pi/3)/.866            */
        *r8 = a3 = int24(a3)  /* store tc  a0  = int(tc*fclk)            */
/* ==== 35 lines=============================================================*/
/*                                                                         */
/* ==================== Determine Sector ===================== */
/* ------------------------------------------------------------------------*/
/* _cosV resides in a1 & _sinV resides in a2; last operation               */
/* ------------------------------------------------------------------------ */
        a1 = *r5              /*                                         */
        a2 = *r6              /* <-|                                     */
        r3e = _cos_pi_3       /*   |                                     */
        r2e = _sin_pi_3       /*   |                                     */
        a3 = *r2 + a2         /*   |  <-|                                */
        if (ale) goto __xx    /* <-|    |   <-|                          */
        a3 = a1               /*   |<-| |     |                          */
        a3 = *r2 - a2         /* <-|  | |     |                          */
        nop                   /*   |  | |  |  |                          */
        a3 = a1               /*   |<-|  | |  |                          */
        a3 = a1 * *r2         /* sin(pi/3)*cos(a)  |                     */
        if (alt) goto __sec2  /* <-| | | |  |                          */
        a0 = a2 * *r3         /* cos(pi/3)*sin(a)  |                     */
        if (alt) goto __sec3  /*   <-| | |  |                          */
        nop                   /*       | |  |                          */
        goto __sec1           /*       | |  |                          */
__xx:   a3 = a1 * *r2         /* sin(pi/3)*cos(a)<-|                     */
        if (ale) goto __sec5  /*      <-| |                              */
        a0 = a2 * *r3         /* cos(pi/3)*sin(a)                        */
        if (ale) goto __sec4  /*         <-|                             */
        nop                   /*                                         */
        goto __sec6           /*                                         */
        nop                   /*                                         */
/*======================== max lines: 14 ========================*/
__sec1: a0 = -a0 + a3         /*-cos(pi/3)*sin(a)+sin(pi/3)*cos(a)       */
        r3e = _factor         /* r3 points _factor                       */
```

```
        a3 = a2 * *r3          /* load _t2 with _factor*sin(_vang)          */
        a0 = *r3 * a0          /* load _t1 w/_factor*sin(pi/3-a)            */
        r2e = _half            /* r2 points _half                           */
        a1 = a0 + a3           /* a1 = _t1+ (_t2)                           */
        r3e = _tc              /* r3 points _tc                             */
        a2 = -a1 + *r3         /* _tc -(_t1 + _t2) = _t0*2                  */
        r7e = _tu              /*                                           */
        r15 = 0x0001           /* for address _pwm increment                */
        a3 = a2  * *r2         /* _tu = a3 = _t0*2 ( *.5)                   */
        r6e = _tw              /*                                           */
        r5e = _tv              /*                                           */
        a2 = a3 + a1           /* _tw = a2 = _t0 + _t1+_t2                  */
        a0 = a3 + a0           /* _tv = a0 = _t0 +_t1                       */
        *r7 = a3 = int24(a3)   /* int24(tu)                                 */
        *r6 = a2 = int24(a2)   /* int24(tw)                                 */
        *r5 = a0 = int24(a0)   /* int24(tv)                                 */
        r8e = _pwm + 1         /*                                           */
        r7 = *r7               /*                                           */
        r6 = *r6               /*                                           */
        r5 = *r5               /*                                           */
        *r8++r15 = r6l         /* tw to _pwm+1                              */
        *r8++r15 = r5l         /* tv to _pwm+2                              */
        pcgoto __end           /*                                           */
        *r8 = r7l              /* tu to _pwm+3                              */
/*================== 26 lines  test w/ .5 0615232a ==================*/
__sec2: a1 = a0 + a3           /*cos(pi/3)*sin(a)+sin(pi/3)*cos(a)         */
        a2 = a0 - a3           /*cos(pi/3)*sin(a)-sin(pi/3)*cos(a)         */
        r3e = _factor          /* r3 points _factor                         */
        r2e = _half            /* r2 points _half                           */
        a0 = *r3 * a1          /* load _t1 w/_factor*sin(pi/3-a)            */
        a3 =  a2 * *r3         /* load _t2 with _factor*sin(_vang)          */
        r5e = _tv              /*                                           */
        a1 = a3 + a0           /* a1 = _t1+ (_t2)                           */
        r3e = _tc              /* r3 points _tc                             */
        a2 = -a1 + *r3         /* _tc -(_t1 + _t2) = _t0*2                  */
        r6e = _tw              /*                                           */
        r7e = _tu              /*                                           */
        a0 = a2  * *r2         /* _tv = a3 = _t0*2 ( *.5)                   */
        r15 = 0x0001           /* for address _pwm increment                */
        a2 = a0 + a1           /* _tw = a2 = _t0 +_t1+_t2                   */
        a1 = a0 + a3           /* _tu = a0 = _t0 +_t2                       */
        *r5 = a0 = int24(a0)   /* int24(tv)                                 */
        *r6 = a2 = int24(a2)   /* int24(tw)                                 */
        *r7 = a1 = int24(a1)   /* int24(tu)                                 */
        r8e = _pwm +1          /*                                           */
        r5 = *r5               /*                                           */
        r6 = *r6               /*                                           */
        r7 = *r7               /*                                           */
        *r8++r15 = r6l         /* tw to _pwm+1                              */
        *r8++r15 = r5l         /* tv to _pwm+2                              */
```

```
        pcgoto __end          /*                                          */
        *r8 = r7l             /* tu to _pwm+3                             */
/*================== 27 lines  test w/ 1.32745 0f07232a ================*/
__sec3: a0 = a0 + a3          /*cos(pi/3)*sin(a)+sin(pi/3)*cos(a)         */
        r3e = _factor         /* r3 points _factor                        */
        a3 =  a2 * *r3        /* load _t1 with _factor*sin(_vang)         */
        a0 = -*r3 * a0        /* load _t2 w/_factor*sin(pi/3-a)           */
        r2e = _half           /* r2 points _half                          */
        a1 = a0 + a3          /* a1 = _t1+ (_t2)                          */
        r3e = _tc             /* r3 points _tc                            */
        a2 = -a1 + *r3        /* _tc -(_t1 + _t2) = _t0*2                 */
        r5e = _tv             /*                                          */
        r15 = 0x0001          /* for address _pwm increment               */
        a0 = a2  * *r2        /* _tv = a3 = _t0*2 ( *.5)                  */
        r6e = _tw             /*                                          */
        a3 = a0 + a3          /* _tw = a0 = _t0 +_t1                      */
        a2 = a0 + a1          /* _tu = a2 = _t0 + _t1+_t2                 */
        r7e = _tu             /*                                          */
        *r5 = a0 = int24(a0)  /* int24(tv)                                */
        *r6 = a3 = int24(a3)  /* int24(tw)                                */
        *r7 = a2 = int24(a2)  /* int24(tu)                                */
        r8e = _pwm +1         /*                                          */
        r5 = *r5              /*                                          */
        r6 = *r6              /*                                          */
        r7 = *r7              /*                                          */
        *r8++r15 = r6l        /* tw to _pwm+1                             */
        *r8++r15 = r5l        /* tv to _pwm+2                             */
        pcgoto __end          /*                                          */
        *r8 = r7l             /* tu to _pwm+3                             */
/*================== 26 lines  test w/ 2.1745 21071f2a ================*/
__sec4: a0 = -a0 + a3         /*-cos(pi/3)*sin(a)+sin(pi/3)*cos(a)        */
        r3e = _factor         /* r3 points _factor                        */
        r2e = _half           /* r2 points _half                          */
        a3 = -a2 * *r3        /* load _t2 with _factor*sin(_vang)         */
        a0 = -*r3 * a0        /* load _t1 w/_factor*sin(pi/3-a)           */
        r7e = _tw             /*                                          */
        a1 = a0 + a3          /*  a1 = _t1+ (_t2)                         */
        r3e = _tc             /*  r3 points _tc                           */
        a2 = -a1 + *r3        /*  _tc -(_t1 + _t2) = _t0*2                */
        r5e = _tu             /*                                          */
        r6e = _tw             /*                                          */
        a0 = a2  * *r2        /* _tw = a0 = _t0*2 ( *.5)                  */
        r15 = 0x0001          /* for address _pwm increment               */
        a2 = a0 + a1          /* _tu = a2 = _t0 +_t1+_t2                  */
        a1 = a0 + a3          /* _tv = a1 = _t0 +_t2                      */
        *r7 = a0 = int(a0)    /* int24(tw)                                */
        *r5 = a2 = int(a2)    /* int24(tu)                                */
        *r6 = a1 = int(a1)    /* int24(tv)                                */
        r8e = _pwm +1         /*                                          */
        r7 = *r7              /*                                          */
```

```
        r5 = *r5            /*                                    */
        r6 = *r6            /*                                    */
        *r8++r15 = r7l      /* tw to _pwm+1                       */
        *r8++r15 = r6l      /* tv to _pwm+2                       */
        pcgoto __end        /*                                    */
        *r8 = r5l           /* tu to _pwm+3                       */
/*================== 26 lines   test 3.2745 220b082a ==============*/
__sec5: a1 = a0 + a3        /*cos(pi/3)*sin(a)+sin(pi/3)*cos(a)   */
        a2 = a0 - a3        /*cos(pi/3)*sin(a)-sin(pi/3)*cos(a)   */
        r3e = _factor       /* r3 points _factor                  */
        r2e = _half         /* r2 points _half                    */
        a0 = -*r3 * a1      /* load _t1 w/_factor*sin(pi/3-a)     */
        a3 =  -a2 * *r3     /* load _t2 with _factor*sin(_vang)   */
        r7e = _tw           /*                                    */
        a1 = a3 + a0        /* a1 = _t1+ (_t2)                    */
        r3e = _tc           /* r3 points _tc                      */
        a2 = -a1 + *r3      /* _tc -(_t1 + _t2) = _t0*2           */
        r5e = _tv           /*                                    */
        r6e = _tu           /*                                    */
        a3 = a2  * *r2      /*   a3 = _t0*2 ( *.5)                */
        r15 = 0x0001        /* for address _pwm increment         */
        a2 = a3 + a1        /* _tv = a2 = _t0 +_t1+_t2            */
        a0 = a3 + a0        /* _tu = a0 = _t0 +_t1                */
        *r7 = a3 = int24(a3)  /* int24(tw)                        */
        *r5 = a2 = int24(a2)  /* int24(tv)                        */
        *r6 = a0 = int(a0)  /* int24(tu)                          */
        r8e = _pwm +1       /*                                    */
        r7 = *r7            /*                                    */
        r5 = *r5            /*                                    */
        r6 = *r6            /*                                    */
        *r8++r15 = r7l      /* tw to _pwm+1                       */
        *r8++r15 = r5l      /* tv to _pwm+2                       */
        pcgoto __end        /*                                    */
        *r8 = r6l           /* tu to _pwm+3                       */
/*================== 27 lines   test 4.32745 1d22072a ============*/
__sec6: a0 = a0 + a3        /*cos(pi/3)*sin(a)+sin(pi/3)*cos(a)   */
        r3e = _factor       /* r3 points _factor                  */
        r2e = _half         /* r2 points _half                    */
        a3 =  -a2 * *r3     /* load _t2 with _factor*sin(_vang)   */
        a0 = *r3 * a0       /* load _t1 w/_factor*sin(pi/3-a)     */
        r7e = _tu           /*                                    */
        a1 = a0 + a3        /*  a1 = _t1+ (_t2)                   */
        r3e = _tc           /*  r3 points _tc                     */
        a2 = -a1 + *r3      /*  _tc -(_t1 + _t2) = _t0*2          */
        r5e = _tv           /*                                    */
        r15 = 0x0001        /*  for address _pwm increment        */
        a3 = a2  * *r2      /*  a3 = _t0*2 ( *.5)                 */
        r6e = _tw           /*                                    */
        a0 = a3 + a0        /*  _tw = a0 = _t0 +_t1               */
        a2 = a3 + a1        /*  _tv = a2 = _t0 +_t1+_t2           */
```

```
            *r7 = a3 = int24(a3)    /* int24(tu)                                */
            *r6 = a0 = int24(a0)    /* int24(tw)                                */
            *r5 = a2 = int24(a2)    /* int24(tv)                                */
            r8e = _pwm +1           /*                                          */
            r7 = *r7                /*                                          */
            r6 = *r6                /*                                          */
            r5 = *r5                /*                                          */
            *r8++r15 = r6l          /* tw to _pwm+1                             */
            *r8++r15 = r5l          /* tv to _pwm+2                             */
            pcgoto __end            /*                                          */
            *r8 = r7l               /* tu to _pwm+3                             */
/*=================== 26 lines   test 5.32745 07 22 0b 2a ============= */
/*                                                                              */
/* ========================= Wait Loop ======================== */
/* ------------------------------------------------------------------------ */
/* ------------------------------------------------------------------------ */
__end:      r2e = _gg
            a1 = *r2
            3*nop
            if (ale) goto __control
            nop
            goto   __end
            nop
/* ============================================================== */
/*                                                                          */
/*======================= Inverse Square ======================= */
_invsqr:    r2e = _x1               /* dummy varible                            */
            r1e = _invsqrC
            *r2++ = a0 = *r1++
            *r2-- = a0 = *r1
            r1e = _invsqr_in
            r3e = _invsqrB
            r4l = *r1
            nop
            r4 = -r4
            *r2 = r4l
            a2 = *r1 * *r2
            r4 = r4 >> 1
            if (cc) pcgoto _invsqrA
            a1 = a2 * a2
            r2e = r2 + 4
_invsqrA:
            a0 = a2 * *r3++
            a0 = a0 + *r3++
            a0 = a0  +a1 * *r3++
            a1 = a2 * *r3++
            a1 = a1 + *r3++
            a0 = a1 + a1 * a0
            a2 = a2 * *r3++
            r4 = r4 - 64
```

```
            a1 = a0 * a0
            *r2 = r4l
            a0 = a0 * *r2
            a1 = *r3 - a1 *a2
            r1e = _invsqr_out
            nop
            *r1 = a0 = a0 * a1
            nop
            return (r21)
            nop
/*================ 34 lines ================================== */

/* ======================== Interupt Routine ==================== */
/* ------------------------------------------------------------------------- */
/* ------------------------------------------------------------------------- */
__intr:   r2e = _pwm
            obufe = *r2
            nop
            r2e = _gg
            r3e = _neg1
            *r2 = a1 = *r3
            nop             /* required */
            nop             /* required */
            ireturn
            nop
/* ============================================================== */


            .rsect ".table"
itable:   2*nop
            goto 0x8004d4
            nop
            2*nop
            2*nop
            2*nop
            goto __intr
            nop


            .rsect  ".data"

_pwm:      float
_2pi:     float 6.2831853
_2n:      float 8.26e2
_n:       float 4.13e2
_2n_fclk:  float 2.6432e-4

_cos_pi_3: float 0.50000000
_sin_pi_3: float 0.866025404
/* ======================== Inversion ========================*/
```

```
_invfa:   float 1.4074074347, 0.81,2.27424702,-0.263374728
/*=============================================================*/
_cosV:    float 0.0              /*                             */
_sinV:    float 0.0
_tc:      float 0.0
_tu:      float 0.0
_tw:      float 0.0
_tv:      float 0.0
_vang:    float 0.0
_m:       float 6.04e-3          /*  pi/(2*Vdc)*sin(pi/3))      */
_factor:  float 0.0
_half:    float 0.5
_neg1:    float -1.0
_pos1:    float 1.0
_zero:    float 0.0
_Yd_new:  float 0.0
/* ========================= INVSQR ========================== */
_invsqrB: float -0.42990081, 1.2869825, 0.057866799, -2.0122938
          float  2.0972557, 0.5, 1.5000001
_invsqrC: float  1.0, 1.4142136
_x0:      float 0.0
_x1:      float 0.0
_x2:      float 0.0
_invsqr_in:  float 0.0
_invsqr_out: float 0.0
/* ============================================================ */
_gg:      float 0.0
_ang:     float 0.0
_fclk:    float 3.125e6          /* clock of pulse generaters   */
_fout:    float 3.0e1
/* ============== CURRENT MEASUREMENT PARAMETERS ====== */
cur_a:    float 0.0
cur_b:    float 0.0
cur_c:    float 0.0
f_in:  float -0.000081197        /* analog channel:  - 0.00009184   */
offset: float 132.5              /* offset of current measurement   */
f_cura:  float 99.0              /* 125 Amp / 1.25 V                */
f_curb:  float 99.0              /* 125 Amp / 1.25 V                */
f_ino:   float -0.036539
offs_2: float -0.0482            /* offset of current measurement   */
f_cur2: float -20.086            /* 1 Amp / 17.7 mV                 */
loops:  float 0.142857           /* 1/wire loops          XXOXOXOXOX */

/* ============== SPEED MEASUREMENT PARAMETERS========= */
_2pi_fclk: float 3.83495e4       /*  (2*pi/1024)*6.25e6             */
_wr_new:  float 0.0
_wr_old:  float 0.0
_wr_oldh:  float 0.0
_wr_oldl:  float 0.0
_spd_tol:  float 0.05            /* Speed tol. +/-        XXOXOXOXOX */
```

```
/* ================== 3=>2 Coord. Transformation ================= */
_p_fac: float 0.66667, 0.33333, 0.57735 /* 2/3 1/3 1/sqr(3)                */
_isa_new: float 0.0
_isb_new: float 0.0
/* ================== FLUX OBSERVER PARAMETER========== */
_pole:  float 2.0              /*              XXOXOXOXOX          */
_ITr:   float 3.24e2           /*              XXOXOXOXOX          */
_c1:    float 6.1              /* ITr*Lm       XXOXOXOXOX          */
_time:  float 82.0e-6          /*              XOXOXOXOX           */

_Yahd_new: float 0.0
_Ybhd_new: float 0.0

_Yahd_old: float 0.0
_Ybhd_old: float 0.0

_Yah_old:  float 0.0
_Yah_new:  float 0.0
_Ybh_old:  float 0.0
_Ybh_new:  float 0.0

_Yd_ref_new: float 0.02        /*              XXOXOXOXOXOX        */
_1_Yd:     float 0.0

_cos_pr: float 0.0
_sin_pr: float 0.0
_isq_new: float 0.0
_isd_new: float 0.0
/* ================== STORAGE PARAMET================= */
_count:   float 0.0
_event:   float 1230.0         /*              XXOXOXOXOXOX        */
_testc:   float 0.0
_storcnt: float 0.0
_stor:    float 3.20e4
_done:    float -1.0
/* ================== CHANGE SPEED ================ */
_countspd: float 0.0
_eventspd: float 2.0e4
_speed1:  float 0.0
_speed2:  float 140.0
_donespd: float -1.0
/* ================== FLUX CONTROLLER ================ */
_Yd_int_old: float 0.0
_isd_kp:  float 0.35           /*              XXOXOXOXOXOX        */
_isd_ki:  float 0.001          /*              XXOXOXOXOXOX        */
_ILm:     float 53.0           /*              XXOXOXOXOXOX        */
_isd_ref_new: float 0.0
/* ================ Isd CURRENT CONTROLLER ============== */
_isd_int_old: float 0.0
_usd_kp:  float 0.25           /*              XXOXOXOXOXOX        */
```

```
_usd_ki:  float 0.0          /*              XXOXOXOXOXOX      */
_usd_ref:  float 0.0
/* ================== SPEED CONTROLLER ================ */
_wr_ref:   float 210.0       /*              XXOXOXOXOXOX      */
_wr_int_old: float 0.0
_te_kp1:  float 0.7
_te_kp2:  float 0.7
_te_kp:   float 1.17         /*              XXOXOXOXOXOX      */
_te_ki:   float 0.15         /*              XXOXOXOXOXOX      */
_te_ref_new: float 0.0
/* ===================== TORQUE CONTROLLER ============== */
_k:        float 3.9         /* 3*p*Lm/2*Lr      XXOXOXOXOXOX   */
_te_int_old:  float 0.0
_isq_kp:    float 1.05       /*              XXOXOXOXOXOX      */
_isq_ki:    float 0.15       /*              XXOXOXOXOXOX      */
_isq_ref_new: float 0.0
/* ================ Isq CURRENT CONTROLLER ============== */
_isq_int_old: float 0.0
_isq_error:  float 0.0
_usq_kp:  float 0.25         /*              XXOXOXOXOXOX      */
_usq_ki:  float 0.0          /*              XXOXOXOXOXOX      */
_usq_ref:  float 0.0

_Vsa:    float 0.0
_Vsb:    float 0.0
```

APPENDIX B

```
%*******************************************************
%  Simulation program of Indirect Rotor Flux Field Orientation
%                        by John W. Kelly
% Copyright 1997
%*******************************************************

%-----------------------------------------------------------------
global YahNew YahOld YbhNew YbhOld YahdOld YbhdOld z TE;
global Vsa Vsb;
YahNew = 0.00;
YahOld = 0.00;
 YbhNew = 0;
YbhOld = 0;
 YahdOld = 0.0;
 YbhdOld =0.00;
 Vsa = 0;
 Vsb =0;
 z =0;

[t,y]=ode23('mtrr12',0,40,[0.02 0.0 0.0 0.0 0.0 0.0 0.02 0.0 0 0 0 0 0]');
%-----------------------------------------------------------------------



function [xdot] = mtrr12 (t,x,Vsa,Vsb)
global YahNew YahOld YbhNew YbhOld YahdOld YbhdOld z TE;
global Vsa Vsb;
%z=1+z
Wrref = 180;
p=2;
j=.0006;
Rs=8;
Rr=4.1668;
Lr= .0128875;
Ls= .0128875;
Lm = .018833;
segr = Ls/Lm - 1/Lm;
%segr = -52.414;
seg = 1 - 1/(1+segr)^2;
%seg = .999622;
ITs = Rs/Ls;
%ITs = 620.76;
ITr = Rr/Lr;
%ITr = 323.32;
beta=Lm/(seg*Ls*Lr);
%beta=113.43;
eta=1/seg;
%eta=1.00;
gamma=1/(seg*Ls);
%gamma = 77.594;
```

```
mu=p*Lm/(j*Lr);
%mu=29.292;
k=3.9;
TL=0.00;

% ------------ Induction Motor Model in Stator Coordinates ------------------------------------
xdot(1)=-ITr*x(1)-p*x(5)*x(2)+ITr*Lm*x(3);                    %dYra/dt
xdot(2)=-ITr*x(2)+p*x(5)*x(1)+ITr*Lm*x(4);                    %dYrb/dt
xdot(3)=beta*ITr*x(1)+beta*p*x(5)*x(2)-(ITs*eta+ITr*beta*Lm)*x(3)+gamma*Vsa;%dIsa/dt
xdot(4)=beta*ITr*x(2)-beta*p*x(5)*x(1)-(ITs*eta+ITr*beta*Lm)*x(4)+gamma*Vsb;%dIsb/dt
xdot(5)=mu*(x(2)*x(3)-x(1)*x(4));
xdot(6)=x(5);
%--------------- Second Order Flux Observer ----------------------------------------------------
xdot(7) = -ITr*x(7)-p*x(5)*x(8)+ITr*Lm*x(3);
xdot(8) = -ITr*x(8)+p*x(5)*x(7)+ITr*Lm*x(4);


WrNew = x(5) ;
%W = [ WrNew t];
IsaNew = x(3);
IsbNew = x(4);

% ---------------------------- Control Scheme ---------------------------------

Yd = sqrt(x(7).^2 +x(8).^2);
cosPr=x(7)./Yd;
sinPr=x(8)./Yd;

%-----------------------------Coord. Transformation -------------------------

IsdNew = IsaNew*cosPr+IsbNew*sinPr;
IsqNew = -IsaNew*sinPr+IsbNew*cosPr;

%+++++++++++++++ Usdref ++++++++++++++++++++++++++++++++

%----------------------------- Flux Controller -------------------------------
Kp = .35;
Ki = .001;
Yrref = .02;
YrrefENew = (Yrref - Yd)./Lm;
xdot(9)= YrrefENew;
IsdrefNew = Kp.*YrrefENew + Ki.*x(9);



%------------------------ Mag. Current Controller  Isd ----------------------------

Kp = 0.25;
Ki = 0;
```

```
IsdrefENew = IsdrefNew - IsdNew;
xdot(10) = IsdrefENew;
UsdrefNew = Ki.*x(10) + Kp.*IsdrefENew;
%-------------------------------------------------------------------------------

%++++++++++++++++++++ Usqref ++++++++++++++++++++++++++++++

%----------------------------Speed  Controller ------------------------------------
Kp = 1.170;
Ki = 0.15;
WrrefENew = Wrref - WrNew ;
%WE(z,1)=WrrefENew;
xdot(11) = WrrefENew;
%WE(z,2)=x(11);
TrefNew =Kp.*WrrefENew+Ki.*x(11) ;
%WE(z,3)=TrefNew ;
%po=[WrrefENew x(11) TrefNew x(5)];
%-------------------------------------------------------------------------------
%-------------------------- Torque Controller ---------------------------------
Kp = 1.05;
Ki = 0.05;
TrefENew  = TrefNew - IsqNew.*Yd.*k;
%TE(z,1)=TrefENew;
xdot(12) = TrefENew;
%TE(z,2)=x(12);
IsqrefNew = Kp.*TrefENew + Ki.*x(12);
%TE(z,3)=IsqrefNew ;
%-------------------------------------------------------------------------------
%-------------------------- Current Controller Isq ----------------------------
Kp =.25;
Ki= 0;
IsqrefENew = IsqrefNew -IsqNew;
xdot(13) = IsqrefENew;
UsqrefNew = Kp.*IsqrefENew + Ki.*x(13);
%TE(z,4)=UsqrefNew ;
%-------------------------------------------------------------------------------
UsqrefENew = UsqrefNew;
UsdrefENew = UsdrefNew;

%------------------- Cordinate Transformatiom DQ=>AB--------------------
Vsa = UsdrefENew.*cosPr- UsqrefENew.*sinPr;
Vsb = UsdrefENew.*sinPr + UsqrefENew.*cosPr;
```

BIBLIOGRAPHY

[1] DSP32C Digital Signal Processor Information Manual, AT&T documentation, 1990

[2] The Programmable Logic Data Book, Xilinix, 1994

[3] J. Volder, "The CORDIC Trigonometric Computing Technique," IRE Transactions on Electronic Computers, pp 330-333, September 1959

[4] Muhammad H. Rashid, "Power Electronics Circuits, Devices, and Applications," Englewood Cliffs, N.J. Prentice Hall, 1993.

[5] H. W. Van der Broeck, H. Skudelny, G. Stanke, "Analysis and Realization of a Pulse Width Modulator Based on Voltage Space Vector," IEEE Transaction on Power Electronics, 1986

[6] DSP32c Application Software Library Reference Manual, AT&T 1990

[7] F. Blaschke, " Das prinzip der fieldorientierung, die grundlage futr dir transvector-regelung von asynchronmaschinen," Siemens Zeitschrift, p. 757, 1971

[8] W. Leonhard, "Control of Electrical Drives," Springer Verlag, 1985

[9] B. Aloliwi, H. Khalil, E.G. Strangas, "Robust Speed Control of Induction Motor," Michigan State University, 1996