KAPL-P-000159
(K97039)

DIGITIAL SIGNAL PROCESSING CONTROL OF INDUCTION MACHINE'S TORQUE AND STATOR FLUX UTILIZING THE DIRECT STATOR FLUX FIELD ORIENTATION METHOD

J. Burger Seiz

April 1997

NOTICE

KAPL ATOMIC POWER LABORATORY SCHENECTADY, NEW YORK 10701

Operated for the U. S. Department of Energy
by KAPL, Inc. a Lockheed Martin company

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible in electronic image products. Images are produced from the best available original document.

UNION COLLEGE- GRADUATE STUDIES

Schenectady, New York

DIGITAL SIGNAL PROCESSING CONTROL OF INDUCTION MACHINE'S TORQUE AND STATOR FLUX UTILIZING THE DIRECT STATOR FLUX FIELD ORIENTATION METHOD

A thesis presented to the Committee on Graduate Studies and the Department of Electrical Engineering of Union College, Schenectady, New York, in partial fulfillment of the requirements for the degree of Master of Science.

By Julie Burger Seiz — 3-17-97
Julie Burger Seiz — DATE

Approved by _____
Michael Rudko, Thesis Advisor — DATE

Approved by _____
Department Chairman — DATE

Approved by _____
Chairman- Graduate Sub-Council — DATE

Received in the Office of Graduate Studies — Date _____

INDEX

FIGURES *page*

FIGURES *page*

FIGURES *page*

TABLES

ABSTRACT: Digital Signal Processing Control of Induction Machine's Torque and Stator Flux Utilizing the Direct Stator Flux Field Orientation Method

Subject: This paper presents a review of the Direct Stator Flux Field Orientation control method. This method can be used to control an induction motor's torque and flux directly and is the application of interest for this thesis. This control method is implemented without the traditional feedback loops and associated hardware. Predictions are made, by mathematical calculations, of the stator voltage vector. The voltage vector is determined twice a switching period. The switching period is fixed throughout the analysis. The three phase inverter duty cycle necessary to control the torque and flux of the induction machine is determined by the voltage space vector Pulse Width Modulation (PWM) technique. Transient performance of either the flux or torque requires an alternate modulation scheme which is also addressed in this thesis. Appendix A.1 provides a block diagram of this closed loop system.

Objective: The objective of this thesis is to review one of the more recent advancements in the power electronic field, specifically the use of the Digital Signal Processor (DSP) to aid in determining the desired switching states of an inverter necessary to achieve the desired goal of controlling, in this case, torque and flux of the induction motor. In the past, field oriented control was extremely difficult due to the large number of algorithms and speed required to transform and process the specified signals. This argument is no longer valid with the development of the DSP. This project will focus on the stator flux field orientation method and the advantages that exist from such a control scheme.

PREFACE

The development of this thesis involved four unique phases. The sequence of the Chapters, contained herein, is consistent with the development of this thesis. The first phase , Chapter (1), reviews the use of the three phase induction motor and the characteristics associated with operating an induction motor.

The second phase, Chapter (2), is a review of the controller used to operate the induction motor which, in this case, is a six step pulse width modulation controlled inverter. With the review of the two key components (the induction motor and inverter) complete, the next Chapter focuses on simulating a control system to provide direct torque and flux control of an induction motor. The simulation tool chosen is SIMULINK, a tool used in conjunction with MATLAB. The model was developed and tested based on the stator flux field orientation method.

Chapter (3) is divided into four parts. The first part provides a general overview of the objectives and implementation of the control system chosen for this project. This project was based on the studies documented in Reference (1). The second part provides a discussion of the simulation tool SIMULINK. The third part of Chapter (3) discusses various modulation schemes used to control the induction motor as well as a discussion of the induction motor model itself. Part four of Chapter (3) provides the results of the simulation runs.

Chapter (4) is the final chapter and contains:

(a) a description of the implementation of the control scheme that was used in the previously discussed SIMULINK model using Digital Signal Processing (DSP) hardware,

(b) the results of testing the algorithms with the 21020 floating point processor, and

(c) conclusions related to the development and execution of this project.

A NOTE OF THANKS...

I am deeply indebted to Dr. Jim Lyons for the time and mentoring that he provided to me during the initial developmental phases of this project. He shared with me the principles I needed to accomplish this task as well as field experiences that were useful to the development of this project. I am grateful to Dr. Michael Rudko and Jim Hendrick for their assistance in the digital signal processing portion of this project. For my husband Manny, your patience and faith in me allowed me to successfully struggle through this. For my Lord Jesus Christ, thank you for the continued gift of learning.

Chapter 1- Induction Motor

I. COMPARISON OF AC INDUCTION MOTORS AND DC MOTORS

In order to understand how to control an ac induction motor, it is prudent to compare the characteristics of the ac induction motor and the dc motor. Understanding the positive characteristics of both machines has led to control techniques that take advantage of these positive characteristics. In the case of this project, the control technique (see References (1)-(8)) used is the direct stator flux field orientation method which is discussed in more detail later in this chapter.

The three phase ac induction motor is a brushless machine and; therefore, requires minimal maintenance. The rotor winding of the induction motor is short circuited and receives its supply by induction from the stator thereby providing the name "induction" motor. During the starting of an induction motor, the frequency of the rotor current is high and is considered to be equal to the stator frequency. The induction motor also has high starting current, but minimal torque because of a low rotor power factor (pf) at the high rotor frequency. At normal steady state conditions, frequency of the rotor current is low. The absence of the mechanical commutator in the induction motor allows this motor to operate at higher speeds than its dc motor counterpart. Additionally, armature voltage in an induction motor can be higher than what a dc motor is capable of. The transient response of the dc machine is limited by the rate of rise of armature current, which with today's machines is approximately 30 times rated current. The induction motor has no such

limit. The rate of rise in current is only limited by the leakage inductance of the machine and the amount of voltage available to force the rise of current. In addition to these desirable characteristics, the induction motor is also smaller, lighter in weight and more efficient than a dc motor. The disadvantage of an induction motor is the fact that it is difficult to control due to its highly interactive multivariable control structure. With load variation, the induction motor's space angle between the rotating stator and rotor fields will vary, which results in complex interactions. A dc machine, on the other hand, has a decoupled control structure with independent simplistic control of torque and flux.

II. THREE PHASE TO TWO PHASE TRANSFORMATION

Based on the comparisons between a dc motor and the ac induction motor provided above, the only significant drawback appears to be the overly complex control structure. This problem can be resolved by modifying the induction motor to resemble that of a dc motor. This can be accomplished by implementing a three phase to two phase transformation.The phases of an induction machine are interactive or coupled and; therefore, dependent on one another. To decouple the phases, results in simplification of control, and the phases are broken down into the real and imaginary parts representing the alpha-beta (α-β) reference frame, respectively. The objective of decoupling is achieved by using the principle of field orientation. Field orientation implements a ninety degree space angle between

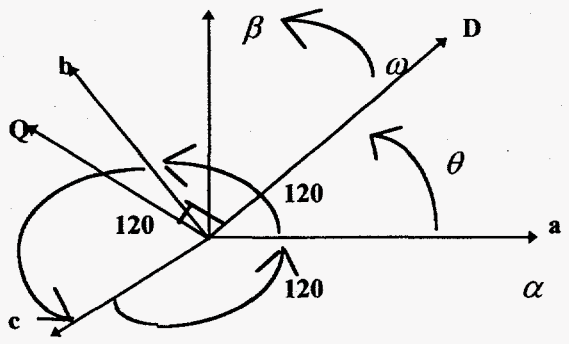
specific field components to simulate the effects of the dc machine. The action of decoupling the phases into a two phase rectangular coordinate system yields:

$$(i\alpha.\beta = i_a + e^{j2(pi)/3}\, i_b + e^{j4(pi)/3}\, i_c = i_a + j\, i_b)$$

and is equivalent to that of a dc machine which already is decoupled by its operational characteristics. It should be noted that this decoupling algorithm is only valid when the sum of the stator currents equals zero. The phase shift for phases b and c, as shown in Figure I.1, is $e^{j2(pi)/3}$ and $e^{j4(pi)/3}$, respectively. The matrix representation of this transformation from three phases to two phases is:

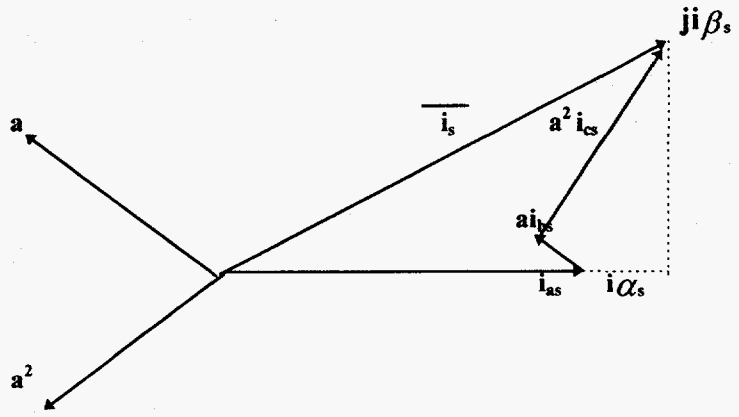
$$\begin{bmatrix} i\alpha s \\ i\beta s \end{bmatrix} = \begin{bmatrix} 1 & -1/2 & -1/2 \\ 0 & \sqrt{3}/2 & -\sqrt{3}/2 \end{bmatrix} \begin{bmatrix} ias \\ ibs \\ ics \end{bmatrix}$$

Figure I.1- Three Phase/ Two Phase Transformation



The same equations can be used for stator voltage and for the complex stator flux linkage vector. The physical interpretation of the three phases is shown in Figure I.2 .

Figure I.2-Physical Interpretation of Three Phase System



$\overline{i_s}$ defines the instantaneous magnitude and angular position of the resultant stator magnetomotive force (mmf) wave. For balanced three phase current, the vector i_s, has a constant amplitude and rotates with a constant angular velocity.

III. FIELD ORIENTED CONTROL

Field oriented control transforms the dynamic structure of the ac machine into that of a separately excited compensated dc machine. For a dc machine, the field flux is proportional to the field current assuming that the magnetic saturation term is negligible. The field current is not effected by, or is independent of, the armature current. The armature current provides direct control of torque. With the induction

motor transformed to a seperately excited dc machine, the induction motor can achieve four quadrant operation with fast torque response (the armature current provides direct control of the torque) and satisfactory performance down to standstill. The space angle of the induction motor (angle between stator and rotor fields) will vary with load and; therefore, result in complex interactions between the fields. The space angle will be controlled such that the stator input current can be decoupled into flux producing and torqueproducing components. The control action takes place in a field coordinate system using the rotating flux vector as a frame of reference for the stator voltages and currents. It is convenient for this project to use the stationary dq (two phase) reference frame. Field oriented control is the independent non-interacting control of flux and torque. Direct control of the position and magnitude of flux can be achieved by either direct measurement or derived from measurement of motor input voltage and current which is the method chosen for this project. Torque producing stator current is perpendicular to the flux and is subsequently regulated to produce the desired torque of the machine.

The flux producing stator current is controlled independently to regulate the flux inside the induction motor. Therefore, similar to the desirable characteristic of a dc machine, the decoupled or independently controlled torque and flux is achievable for the induction motor whenever the position of the flux vector is know

Any stator flux field orientation scheme suffers from the severe problem that at low speed, the stator "IR" drop becomes a significant term in determining the stator flux. The integration at low frequency gives an unreliable result. This is because the signal to noise ratio is very small.

There are three reference frames that can be used in a field orientated control system:

(1) Stationary Reference Frame: $\omega = 0$. This is the method used in this project.

(2) Rotor Reference Frame (Park's Transformation): $\omega_r = \omega$.

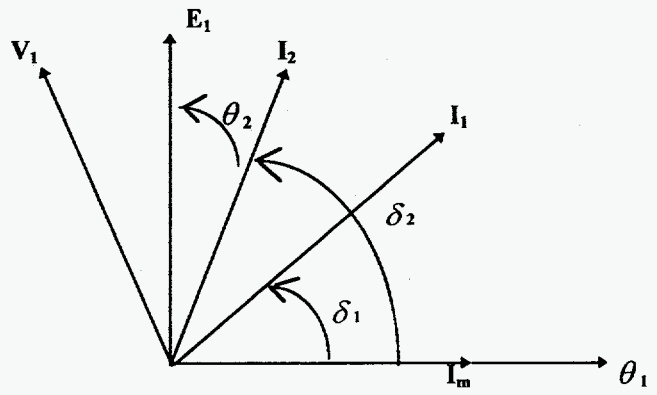
(3) Synchronous Rotating Reference Frame: $\omega_e = \omega$.

Where ω is the excitation frequency, ω_r is the rotor frequency and ω_e is the stator frequency.

The conditions of operation will determine the most convenient reference frame to use. The stationary or synchronous (sync) reference frame are generally picked to analyze balanced or symmetrical conditions. The sync rotating reference frame is convenient to use when incorporating dynamic characteristics of an induction machine into a digital computer program to observe transient conditions of a large power supply. In a pulse width modulation (pwm) drive, the pulsating torques that are developed are small in amplitude and are at high frequencies compared to the fundamental frequency. The result of this is that there are minimal speed pulsations because of motor inertia.

The phasor diagram of the motor can be represented as follows:

Figure I.3 Induction Motor -Phasor Diagram



Where:

I_m= magnetizing current (Reference Phasor)

E_1= stator emf

I_1 = net stator current

I_2 = load component of stator current which neutralizes the rotor mmf and lags E_1 by the rotor power factor angle θ_2

θ_1 = fundamental airgap flux

The flux and current phasors can also be regarded as space vectors of flux and mmf.

Torque can be expressed as follows:

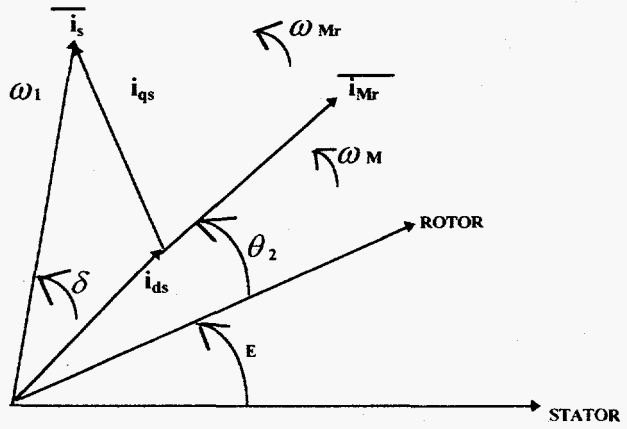
$$T = K\theta_1 I_2 \sin\delta_2 = K\theta_1 I_1 \sin\delta_1$$

IV. OPTIMAL MOTOR OPERATION

Motor operation should be restricted to regions of high torque/amp, thereby matching the motor and inverter ratings and minimizing system losses. To obtain high torque throughout the speed range, the airgap flux should be maintained constant. Below the base speed, the torque producing capability of a given motor frame size can be fully utilized by holding airgap flux constant. The induction motor torque capability is then independent of supply frequency and permits fast transient response of the drive system.

The stator current vector in field coordinates has orthogonal direct and quadrature components i_{ds} and i_{qs} which are perpendicular and parallel to i_{mr}, respectively. The current, i_{mr}, is the modified magnetizing current vector representing rotor flux as shown in Figure I.4. The current, i_{mr}, is analogous to the main field flux of the dc machine and is controlled by i_{ds}, the direct component of the stator current vector.

Figure I.4- Modified Magnetized Current Vector



Where:

ω_M= angular velocity of two pole rotor

ω_{Mr}= instantaneous angular velocity

ω_1 = synchronous angular velocity

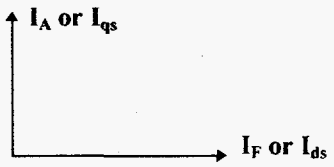
i_M= magnetizing current vector = $i_s + i_r e^{jE}$. This is the sum of the stator and rotor current vectors in a common (in this case stator) reference frame.

$\overline{i_{Mr}}$= modified magnetizing current vector representing rotor flux.

The current i_{qs} is analogous to the armature current of a dc machine and can be rapidly varied by an appropriate change in stator current to provide a fast response to a sudden change in torque demand. If i_{ds} and i_{qs} can be independently controlled, $\overline{i_{mr}}$ will behave like a dc motor with decoupled control of flux and torque. Airgap flux will control stator current which in turn controls the stator voltage. A Volts/Hertz type control system has the disadvantage that airgap flux may drift, which effects slip and torque. As the airgap flux decreases, the slip will increase for the same torque demand. For a voltage fed drive system, both torque and airgap flux are functions of voltage and frequency. The coupling of these two parameters is responsible for the sluggish response of the induction motor. For example, as frequency is increased, slip and torque will increase as flux decreases. The reduction in flux results in a reduction of torque sensitivity with slip and; therefore, lengthens the response time of the machine. Field oriented control of the machine improves the time response concern.

In a dc machine there are two currents of interest; the armature or torque component of current and the field or flux component of current. For the ac machine, we can say that I_{ds} is analogous to the flux component of current and I_{qs} is analogous to the torque component of current, as shown in Figure I.5.

Figure I.5- Torque/Flux Components of Current



I_{qs}= Torque Component I_{ds}= Flux Component

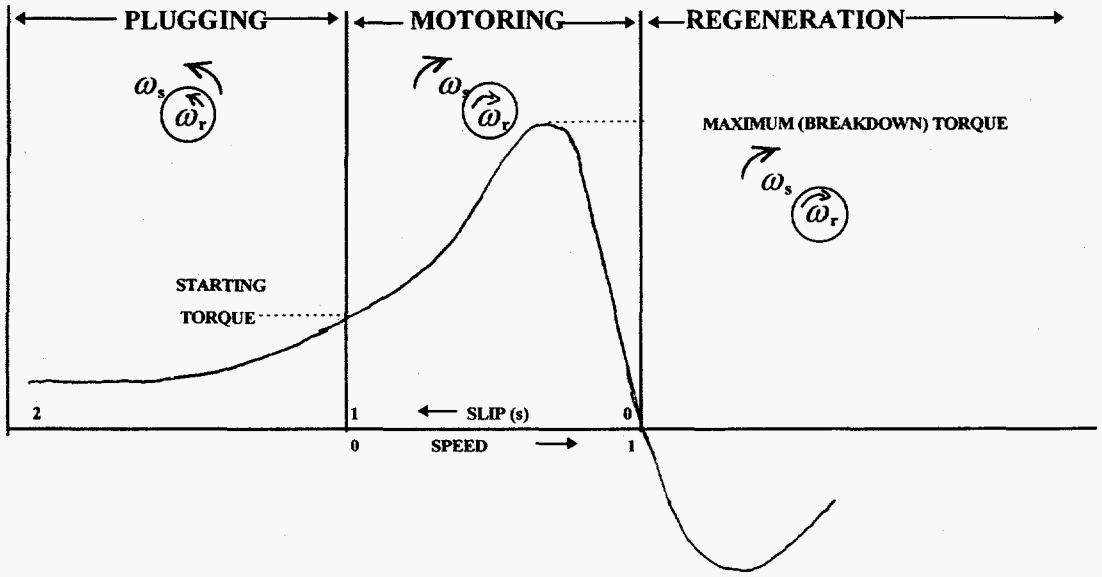
V. TORQUE

Torque is derived from the tangential force exerted on the rotor windings by the radial magnetic field produced by the stator mmf. Torque is proportional to the vector cross product of the stator current vector and the rotor current vector, and is therefore proportional to the sine of the angle between the stator and rotor current vectors.

To generate rotor current and thereby provide torque, the rotor must turn more slowly than the rotor field. The difference in rotational velocity is defined as slip. Slip is defined as f_r/f_s. Unit Slip is the ratio of the difference between the stator and rotor angular frequency (f_r) to the stator angular frequency (f_s). The airgap flux moves at the slip frequency. Synchronous speed is defined as $(60\ f_s)/p$. "p" is defined as the number of poles.

Torque pulsation is produced by the interaction of airgap flux and rotor mmf waves at different harmonic orders. Torque pulsations, if not filtered by the motor inertia, can cause speed fluctuations too. At higher frequency, speed variation becomes less pronounced . However, vibration and; therefore, noise problems are created. For fundamental frequency, or any order thereof, the phase angle between airgap flux and rotor current is constant. A harmonic component of the airgap flux will induce rotor current at the same frequency and; therefore, torque is produced in the same direction as the rotating airgap flux. The torque-speed Curve for a given voltage and frequency is documented in Figure I.6.

Figure I.6 - Torque -Speed Curve at Constant Voltage and Frequency



*Plugging : $1 \leq s \geq 2$ Motoring: $0 \leq s \geq 1$ Regeneration: $s < 0$

* Rapid reversal of the induction motor.

ω_r = rotor angular frequency ω_s= stator angular frequency

As slip of the machine increases, the speed decreases. Energy is consumed during motoring and energy is generated and given back to the source during regeneration. In terms of the dq reference frame torque is defined as follows:

$$T = \frac{3}{2}\frac{p}{2}(\lambda_{ds} I_{qs} - \lambda_{qs} I_{ds})$$

where λ_{ds}= Stator Flux in the d axis I_{ds}= Stator Current in the d axis

λ_{qs}= Stator Flux in the q axis I_{qs}= Stator Current in the q axis

VI. CONTROL OF FLUX AND TORQUE

The induction motor for this project is powered from a six step voltage source inverter (see Chapter 2 for detailed description on voltage source inverter). The objective of this project is to control airgap flux and torque. The torque producing stator current component is regulated to produce the desired torque. The flux producing stator current component is controlled independently to regulate the flux inside the induction motor. Similar to a dc motor, decoupled torque and flux control is obtained for the motor whenever the position of the flux vector is known. For high performance drive systems, precise control of airgap flux as well as fast torque response control is necessary.

This type of control system is used for such things as a traction drive in an electric vehicle or could be used as an inner loop in a speed or position controlled drive system. Terminal stator voltages and currents are monitored and then used to calculate motor torque and airgap flux. For optimum performance, the computation of motor torque and airgap flux should be unaffected by variations in machine parameters.

For this project the inverter duty cycle is directly calculated each switching period based on torque and flux errors, transient reactance and an estimated value of the voltage behind the transient reactance. The flux and torque are derived from the

stator voltage, stator current and resistance in the stator flux field orientation method. This method eliminates the need to monitor rotor speed, position or flux. The disadvantage of this method is that it results in a variable switching frequency due to hysteresis control. This method of control is used for a steady state condition. Constant torque capability throughout the speed range is achieved if the airgap flux in motors is maintained constant at all speeds. For transient conditions the torque and/or flux cannot be driven to a reference value in a single switching period, therefore an alternate control method is used which is discussed in Chapter (2).

In order to study the transient behavior of the induction motor, the simulation of the system assumes the following;

(1) The motor is a symmetrical three phase wye connected stator winding with the neutral electrically isolated.

(2) The space harmonics in the airgap flux mmf and flux density waveforms can be neglected.

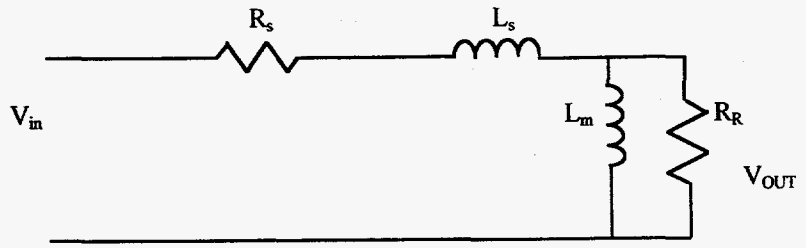
(3) The stator and rotor iron have infinite permeability.

(4) Skin effect and core losses are considered to be negligible.

(5) Slot and end effects can be ignored.

The per phase equivalent circuit of the induction motor can be depicted as follows:

Figure I.7-Per Phase Equivalent Circuit of an Induction Motor



L_s= Stator Inductance R_s= Stator Resistance

L_M= Magnetizing inductance R_R= Rotor Resistance

Balanced three phase stator windings have the same number of effective turns placed 120 degrees apart. A motor is normally delta connected or the neutral is insulated so it is insensitive to line-neutral voltage. Line-neutral voltage can, therefore, be varied with no consequence to the load side (one degree of freedom). The sum of line-line voltages must equal zero.

To express the motor in terms of the stationary reference frame, the excitation frequency component ω_e is assumed to equal zero. Voltage in the dq stationary reference frame is:

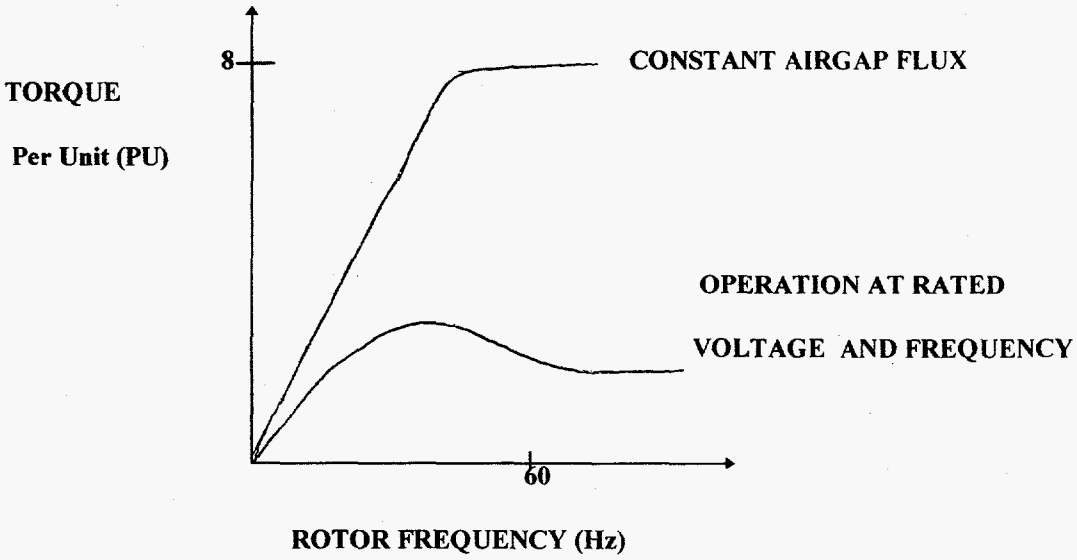
$V_{qs} = R_s I_{qs} + d\lambda qs/dt + \omega_e \lambda_{ds}$

$V_{ds} = R_s I_{ds} + d\lambda ds/dt - \omega_e \lambda_{qs}$

If airgap flux is kept constant under all operating conditions, the torque is solely determined by slip speed and not synchronous speed. The shape of the torque-speed curve (Figure I.6) for different frequencies is, therefore, the same. For constant airgap flux , the inverter output voltage, V_1; and frequency must be appropriately controlled for each operating condition. The inverter frequency is closely related to the shaft speed and must be adjusted to the value that is dictated by the speed command. Therefore, the only quantity for flux regulation is the motor voltage, V_1. Flux sensors could be used, but are normally distorted by large slot harmonics that cannot be filtered effectively because their frequency varies with the motor speed.

Airgap flux is proportional to the: (1) ratio of the airgap electromotive force (emf) to sync speed, and (2) product of the magnetizing inductance and magnetizing current.

Figure I.8- Constant Airgap Flux



Constant Airgap Flux-sustained operation may require special cooling, requiring two times the terminal voltage to achieve maximum torque.

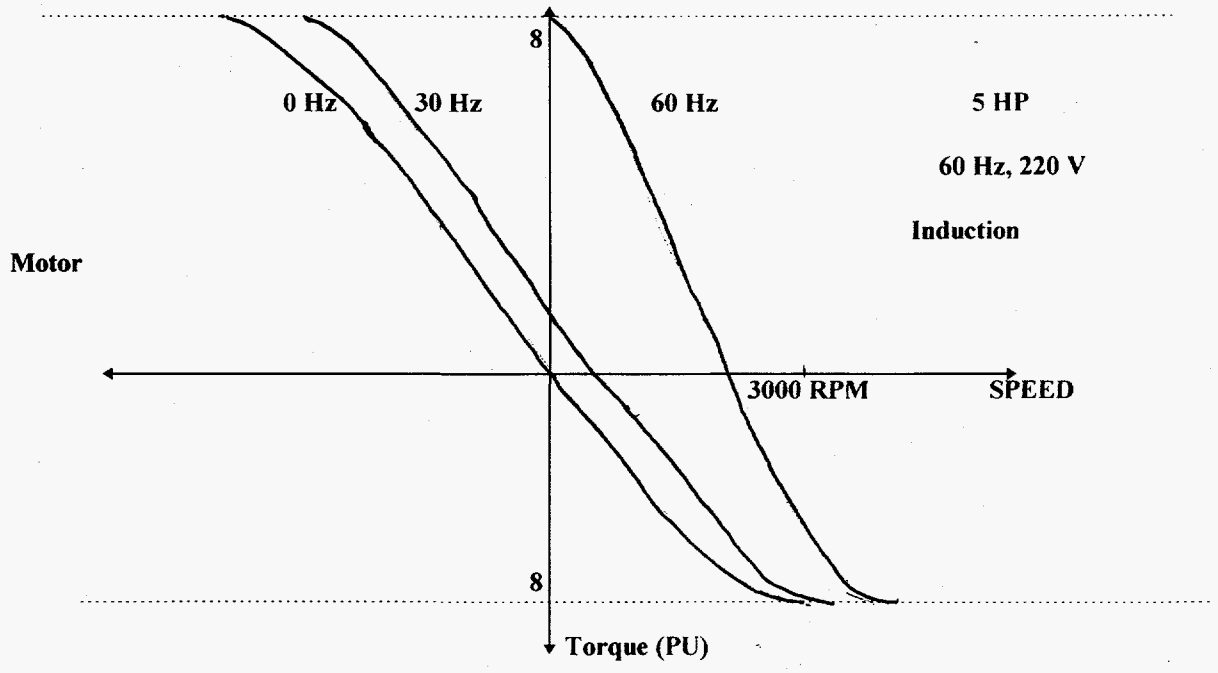
Airgap flux is maintained constant at the value corresponding to no-load operation at rated voltage and frequency. For constant flux, one can apply the universal torque equation:

$T/T_b = [\ (\omega_2/\omega_b) + (\omega_b/\omega_2)]$

ω_b= The rotor breakdown frequency = $\pm R_2 / L_2$

Breakdown torque is the same for all frequencies $T_b = \pm pm_1\ [E_1\ /\ \omega_1\ ]^2 / 2L_2$

Figure I.9- Rotor Breakdown Frequency

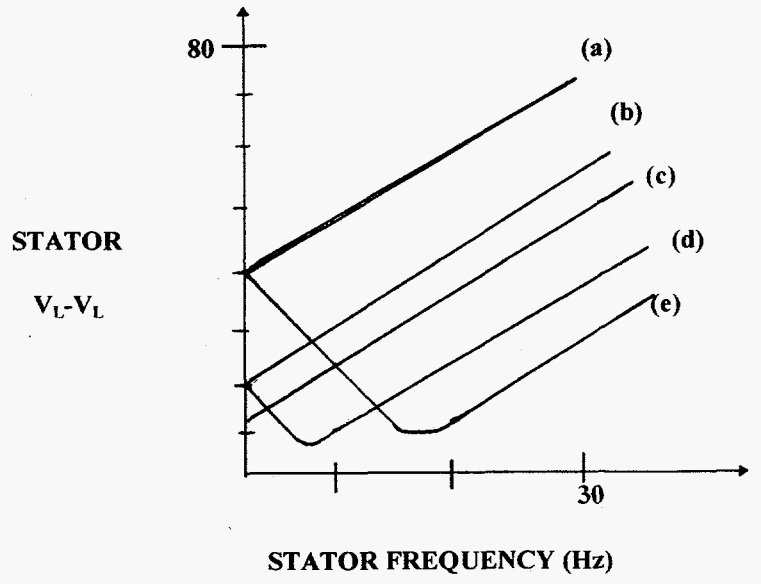


Stator current is independent of the supply frequency when airgap flux is constant. The terminal voltage (V_1) is equal to the phasor addition of the airgap emf (E_1) and the stator voltage drop, $(R_1 + j X_1) I_1$.

$$V_1 = E_1 + (R_1 + j X_1) I_1$$

E_1 varies linearly with stator frequency.The stator voltage required for constant flux operation is described in the figure below.

Figure I.10- Stator Voltage Required for Constant Flux Operation



(a) I_1= 2.0 pu- Motoring

(b) I_1= 1.0 pu- Motoring

(c) I_1= Nominal Magnetizing Current

(d) I_1= 1.0 pu- Generating

(e) I_1= 2.0 pu- Generating

The operation of an induction motor at a high power factor results in a higher efficiency provided that the rotor frequency does not exceed the breakdown torque value.

VII. PER UNIT SYSTEM OF MEASURING

The per unit system of measuring is a method of normalizing values. In the per unit system, the actual value of the parameter is expressed as a fraction of the base value. For example, the root mean square (rms) value of rated phase voltages are generally selected as the base voltage for "abc" variables. Yet, the peak value of voltage is normally selected as the base voltage for dq variables.

Base current is equal to rated full load sine wave current. Base torque is defined as the torque corresponding to a rotor current of one per unit with an airgap flux of one per unit and a rotor power factor of unity. Base torque will be greater than the rated torque of the motor. For this project, use of per unit measurements will be specified when used.

VIII. HARMONICS

In addition to controlling torque and flux directly, some effect can be imparted to improve the harmonics of the motor. Harmonics, generally occur as sidebands of

*NOTE: The project will not focus on methods for improving the harmonics associated with the oepration of this control system.

the carrier frequency and its multiples. Harmonic order is defined as $k = np \pm m$ (where m is defined as the number of sidebands and n is the carrier harmonic). If n is even, there is an odd sideband spectrum because harmonics are nonexistent when n and m are both even. For n=2, there are harmonics of order $2p\pm1$, $2p\pm3$, $2p\pm5$ in the pole voltage waveform, but harmonic amplitude diminishes rapidly with increasing values of m. The fifth harmonic is the same spatial distribution as the fundamental, but one that pulsates at five times the supply frequency. Harmonic behavior of an ac motor mimics a linear device, if magnetic saturation is neglected. The principle of superposition can be used. The result is that the motor behavior can be analyzed independently for the fundamental component and for each harmonic term. The fifth and seventh harmonic react with the fundamental airgap flux to produce a sixth harmonic, pulsating torque, which can cause irregular stepping or cogging rotation of the motor shaft especially at low speeds (less than 5 hertz). One advantage of sinusoidal vice square wave is the enhanced low speed performance. The seventh harmonic adds to the fundamental torque, but the fifth harmonic opposes it.

Harmonic slip is defined as (synchronous speed of the fundamental rotating field - the actual rotor speed) / (synchronous speed of the fundamental rotating field). When a motor is fed by a voltage source inverter with a specific output waveform at a particular frequency, harmonic currents will remain constant from a no load condition to a full load condition. If the motor has a large starting current, it will draw large harmonic currents on non-sinusoidal voltage supplies.

If the motor has very low leakage reactance, caution must be exercised to avoid excessive harmonic currents which leads to overheating of the unit.

Harmonics in the airgap result in additional harmonic torques on the rotor. There are two types:

(1) Steady State- This occurs due to reaction of harmonic airgap fluxes with harmonic rotor mmfs or currents for the same order. This harmonic torque component has a negligible effect on motor operation.

(2) Pulsating State- This occurs due to reaction of harmonic rotor mmfs with harmonic rotating fluxes of a different order. Pulsating torques have zero average value, but their existence causes angular velocity of the rotor to vary during revolution at low speeds. This is evidenced by rotor rotation being "jerky". The irregular cogging motion sets a lower limit for the useful speed range of the motor.

For sustained operation at low speeds, an improved inverter output waveform is required to help suppress low order harmonics. Harmonic currents delivered by the inverter produce a sixth harmonic voltage ripple on the dc link capacitor which in turn amplifies the harmonic voltage components in the inverter output voltage. The end result is that harmonic current flow to the motor increases and the sixth harmonic pulsating torque is magnified. A large dc capacitor will aid in minimizing this effect.

A characteristic of the Pulse Width Modulation (PWM) waveform is the large pulsations that are produced at high switching frequencies.

IX. STABILITY OF THE SYSTEM

Instability due to interaction between the inverter and the motor can occur when the inverter has an infinite source impedance (i.e., filter which smoothes the dc link supply). This will usually occur at frequencies greater than 25 hertz when energy is transformed between motor inertia and filter inductance and capacitance.

Stability of an inverter fed induction motor can be improved by:

(1) increasing the load torque and inertia,

(2) reducing the stator voltage and

(3) increasing filter capacitance and reducing filter inductance and resistance.

Pulse Width Modulation (PWM), discussed in Chapter (3), has a step-down transformer action* that reduces the effective output impedance at low frequencies and; therefore, improves the drive stability.

---

*NOTE: A six step inverter does not exhibit characteristics of a step-down transformer.

This is clear by evaluating low speed operation. The fundamental output voltage is much less than dc link voltage yet the output current is much greater than the dc link current. Therefore, PWM can give stable open loop operation at low speeds. Induction motors run asynchronously at a speed which is normally 95% of the synchronous speed. The torque developed by the induction motor at a given slip varies approximately to the square of the applied voltage, and steady state operation occurs when the motor torque balances the load torque. Output power from the induction motor is proportional to shaft torque and speed. Input power delivered to the inverter is the product of the dc link voltage and current.

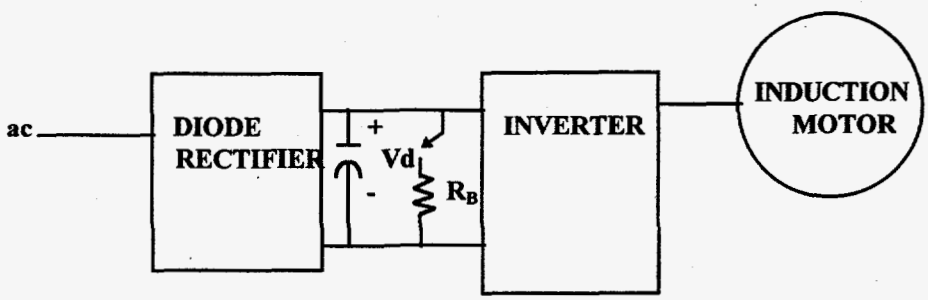
X. DYNAMIC BRAKING

Dynamic braking is a method of dissipating regenerated power and is accomplished by installing a resistor which is switched across the dc link when the dc link voltage reaches a given threshold level due to the regenerative charging of the filter capacitor (Figure I.11). The switching can be performed automatically by means of an auxiliary thyristor or transistor which is triggered from a voltage sensing circuit. There are also other means of achieving the same objective (i.e., regenerative).

Regenerative braking is energy efficient. This allows four quadrant operation. Regenerative braking utilizes a converter at the front end instead of a diode bridge rectifier. The energy recovered from the motor load is fed back to the utility supply.

However, dynamic braking circuit, with its simplistic design, can be constructed at a much lower cost than a regenerative braking design and would be used for this application.

Figure I.11 Dynamic Braking



Chapter 2-Voltage Source Inverter

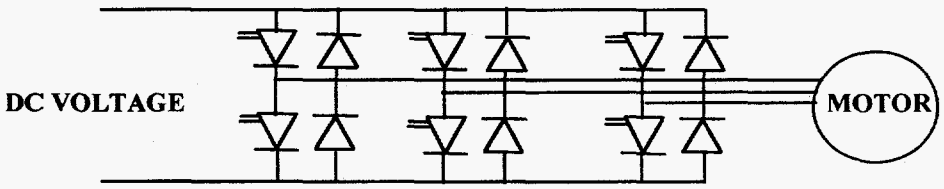
I. INVERTERS

In order to control the torque and flux of the ac induction motor discussed in Chapter (1), the stator current and voltage are sampled periodically. These analog signals are then converted to digital signals and sent to a digital signal processor (Chapter 4). The output (three phase) from the digital signal processor (dsp) is fed back to the inverter where the switching states and time spent in each state are controlled by the dsp. The inverter provides the motor with adjusted waveforms that correct for torque and flux errors. The following provides a general discussion of the voltage source inverter based on a review of References (9) - (12).

The Voltage Source Inverter (VSI) is powered from a stiff, low impedance dc voltage source (i.e., battery or rectifier with a LC smoothing filter). The inverter is an adjustable frequency-voltage source where V_o is independent of I_L. For three phase inverters, the output voltage does not depend on the load. To minimize the inverter starting current rating the controller must restrict motor operation at low slip frequency allowing stable operation at a high power factor with a high torque to stator amp ratio. Voltage source inverters are normally used with big drives with low switching frequency.

However, this method is acceptable for small or medium drives as well. With an extremely high switching frequency, approaching infinity, a filter is often used for both the dc and ac side. The energy stored in the filter is negligible since the instantaneous input equals the instantaneous output. A simplistic representation of the inverter used for this project is shown in Figure II.1.

Figure II.1- Voltage Source Inverter



DC to AC inverters are used in ac motor drives where the objective is to produce a sinusoidal ac output whose magnitude and frequency can be controlled. The control strategy for an adjustable frequency induction motor is to ensure operation is restricted to regions of high torque per ampere which match the motor and inverter ratings and minimizes system losses.

There are three general categories of voltage source inverters:

(1) Pulse Width Modulated Inverters- The dc voltage is kept constant in magnitude. The inverter controls the magnitude and frequency of the ac output voltages. This is achieved by Pulse Width Modulation (PWM) of the inverter switches (i.e., sinusoidal PWM or space vector modulation which is utilized in this project). As the motor speed increases, the modulation strategy can be altered to reduce the number of inverter switches per cycle. This aids in minimizing the inverter switching losses.

(2) Square Wave Inverters- The input dc voltage is controlled in order to control the magnitude of the output dc voltage and the inverter, therefore, only controls the frequency of the output voltage.

(3) Single Phase Inverters with voltage cancellation- The input is constant ac voltage. The single phase output controls the magnitude and the frequency of the inverter output voltage. This inverter combines the characteristics of both inverters above. However, this method does not work with three phase systems and; therefore, was not considered for use in this project.

II. LOSSES ASSOCIATED WITH INVERTER OPERATION

Harmonic loss factors are mainly copper losses (P_{loss})and can be expressed as follows:

$$I_k = V_k / k f_1 X_{pu}$$

$$P_{loss} = \sum_{k \neq 1} I_k^2 R_k$$

where f_1 is the per-unit fundamental frequency, X_{pu} is the per unit leakage reactance and R_k (I_k)is the resistance (current) of the motor.

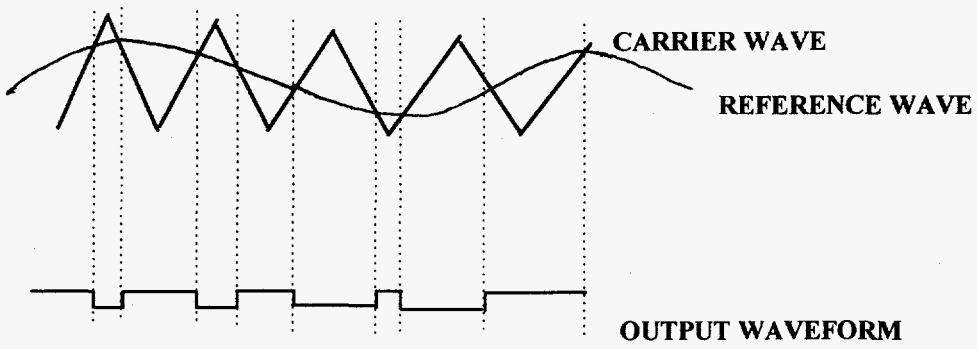
Inverter losses are a function of commutations per second.

III. SINUSOIDAL PULSE WIDTH MODULATION (PWM)

One implementation of PWM is to take a sinusoidal control signal at the desired frequency and compare it with a triangular waveform. This method was implemented/modeled using SIMULINK. The results of this simulation are discussed in Chapter 3. The frequency of the triangular waveform establishes the inverter switching frequency, and is generally constant along with the peak value of voltage waveform (v_{tri}). The switching frequency, f_s, is known as the carrier frequency. The sinusoidal fundamental waveform ($v_{control}$) is used to modulate the switch duty ratio and has frequency (f_1) which is the desired fundamental frequency of the inverter.

The frequency modulation index is defined as $m_f = f_s / f_1$. The modulation index is varied with frequency to control the volts per hertz ratio in the constant torque range. In a six step VSI, the maximum output voltage occurs when the front end thyristor rectifier circuit is phased fully on and the dc link voltage is at the maximum voltage for that application. Figure II.2 shows the relationship between the carrier wave and the control (reference) waveform.

Figure II.2- Relationship of Carrier and Reference Waveforms



The amplitude of the fundamental frequency component of the output voltage varies linearly with the modulation ratio, defined as $m_a = v_{control} / v_{tri}$. This is true as long as m_a is less than or equal to one. The linear range of operation is possible when m_a is a value from 0-1. Harmonics of the inverter output voltage waveform appear as sidebands centered around the switching frequency m_f, $2m_f$, $3m_f$. When m_f is greater than or equal to 9, harmonic amplitudes are independent of m_f even though the value of m_f defines the frequency at which they will occur.

Because of the relative ease of filtering harmonic voltages at high frequencies, it is desirable to use as high a switching frequency as possible. However, it should be noted that switching losses in the inverter switches will increase proportionally with the switching frequency.

Synchronous PWM means that the frequency modulation index is an integer. Therefore, for asynchronous PWM, m_f is a fractional number.

The induction motor voltage waveforms are determined by the impedance presented to the fundamental and harmonic components of the inverter output current.

IV. SPACE VECTOR MODULATION (SVM)

For this project, the PWM technique used is space vector modulation. The switching instants are derived from the voltage space vector. Phase voltages are represented in the alpha beta reference frame. The alpha-beta components are derived by the Park Transform, where total power and impedance remain unchanged. The mean space vector remains unchanged during the switching period. There are eight states in the space vector modulation technique. These states correspond to the eight switching positions of the inverter. The space vector control provides on the average 15% better utilization of the dc bus voltage, and 33% reduction in inverter switching frequency as compared to a conventional sinusoidal PWM technique. The harmonic content of the inverter output voltages and currents is less for the space vector modulation technique than for its sinusoidal counterpart. Consequently, the harmonic content of current and torque pulsations are significantly reduced.

The optimum pulse width modulation is achievable if the following conditions are met:

(1) The maximum deviation of the current vector for several switching states becomes as small as possible.

The deviation current vector within one switching state is defined as

$$\Delta I = \int_{t1}^{t2} (V - \overline{V}) dt$$

$\overline{V}$= mean space vector, which is constant during each switching period;

t1 is the beginning of the switching state; and t2 is the end of the switching state.

(2) the cycle time is as short as possible.

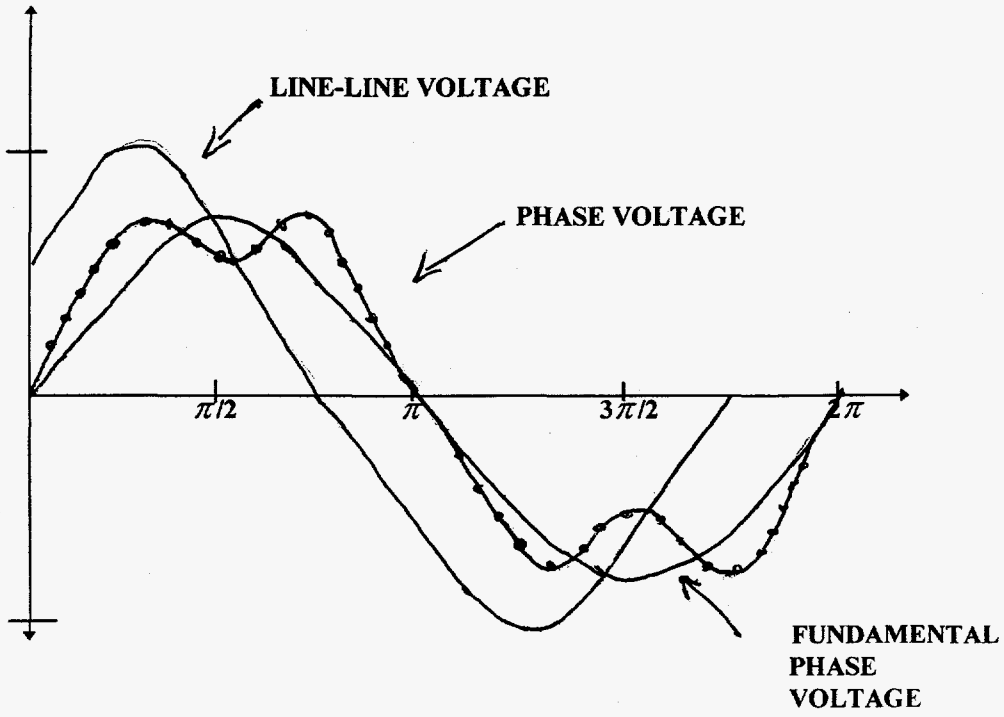
The above conditions are met when the following statements are true:

(1) only the three switching states adjacent to the reference vector are used and

(2) a cycle wherein the average voltage vector is equivalent to the reference vector which consists of three successive switching states.

The space vector modulation technique switches the three inverter legs on and off once per switching cycle; thus, generating three independent waveforms. With the insulated neutral, the motor only senses line to line voltages, and is insensitive to the common mode component of the line to neutral voltage.

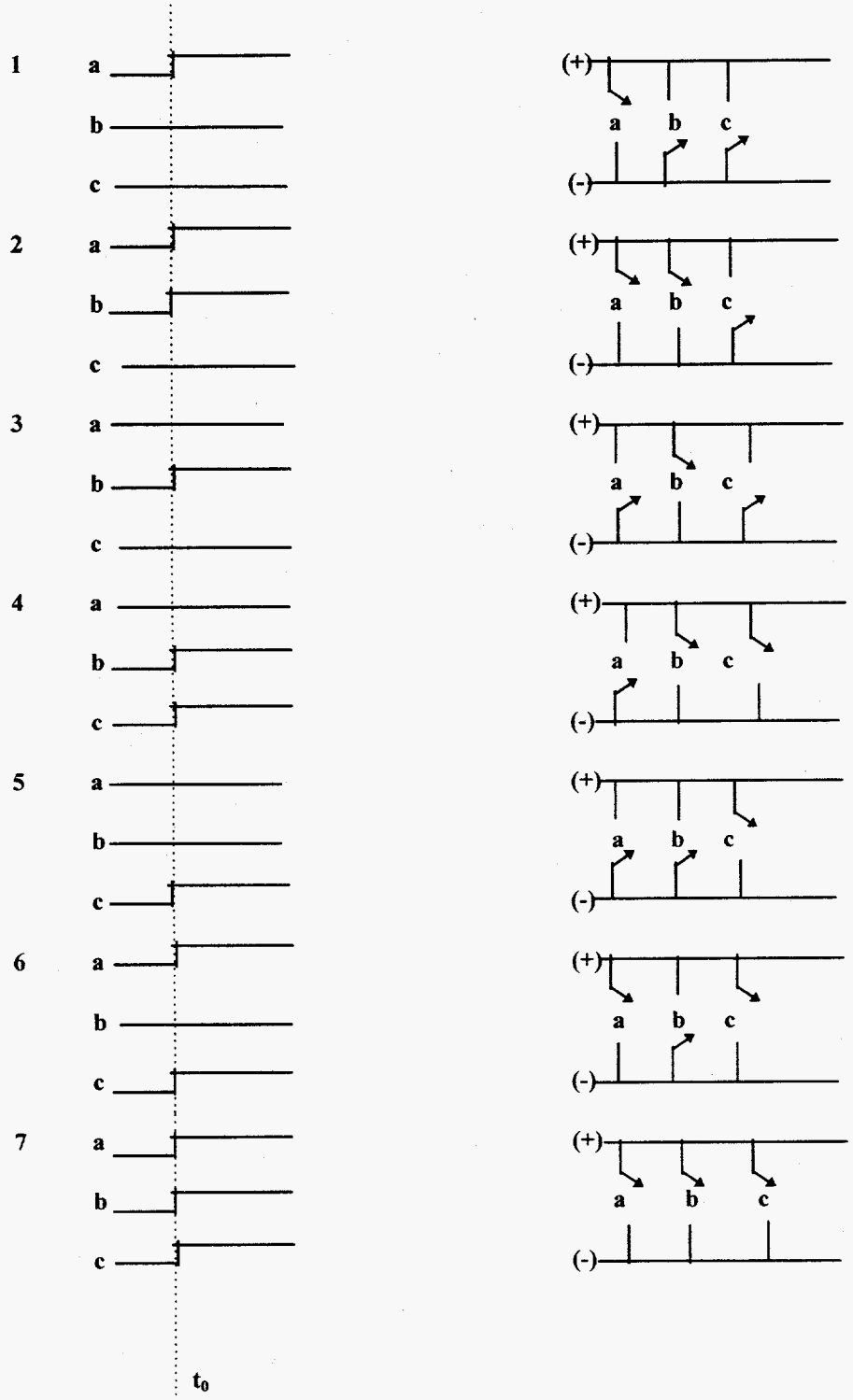
Figure II.3-Line-Line Phase Relationship



The vectors are all of the same magnitude. For space vector modulation, the line to neutral voltage is not sinusoidal; but the line to line voltage as seen by the machine is sinusoidal, as shown in Figure II.3 above.

The line to neutral voltage can, therefore, be varied without consequence to the load side. This allows one degree of freedom. If the signal is high ("1") the top switch is closed, else the signal is low ("0") and the bottom switch is closed. With all upper switches of the inverter closed [1 1 1] the output of the inverter is connected to the (+) dc rail. With all lower switches closed [0 0 0] the output of the inverter is connected to the (-) dc rail. Figure II.4 illustrates States 1-7.

Figure II.4 -Inverter States



The eight possible inverter states are defined as shown in Table II.1.

Table II.1- Inverter States

STATE	Phase A	Phase B	Phase C
0	0	0	0
1	1	0	0
2	1	1	0
3	0	1	0
4	0	1	1
5	0	0	1
6	1	0	1
7	1	1	1

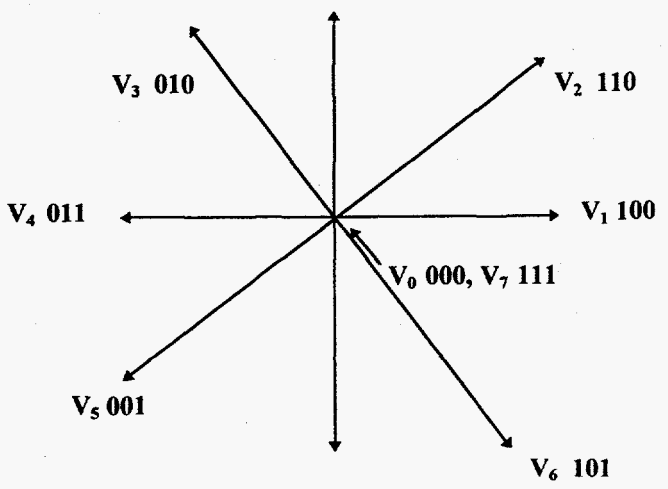
The stator voltage vector takes on one of seven values depending on the switching state "k" of the inverter.

$V_{sk} = 2/3\ V_{dc}\ e^{j(k-1)2pi/3}$ when k=1,2,...6

$V_{sk}=0$ when k=0,7

The space vector diagram is shown in Figure II.5.

Figure II.5- Space Vector Diagram



The reference voltage is realized in an average sense by computing the duty ratio (fraction of the switching period) for two voltage vectors V_{sk} and $V_{s(k+1)}$ which are the vectors adjacent to V_s. The amount of time at each state is defined as follows;

$V_sTs = (V_{sk}\,Tk) + (V_{s(k+1)}\,T(k+1))$

$Ts = T0 + Tk + T(k+1)$

T0 = time spent at the zero state

Tk = time spent on V_{sk} $\qquad Tk = \dfrac{|Vref|}{Vdc}\dfrac{\sin(60-\gamma)}{\sin 60}Ts$

$\gamma = Tan^{-1}\ \dfrac{Vq}{Vd}$

T(k+1) = time spent on $V_{s(k+1)}$ $$T(k+1) = \frac{|Vref|}{Vdc} \frac{\sin\gamma}{\sin 60} Ts$$

Ts = half the actual switching frequency

In order to switch from the zero state to two adjacent states, each inverter leg is commutated once during each half of the switching frequency. This allows control of torque and flux twice during each switching cycle. Figure III.1 shows the various states and associated times in each case.

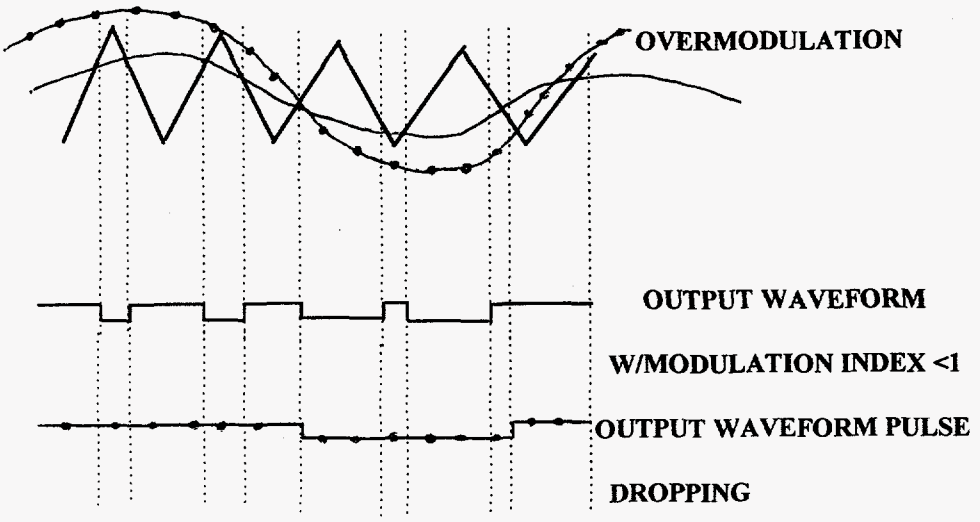
The goal for the space vector modulation technique is to minimize the deviation of stator current from its fundamental component each switching period. This optimization can be accomplished by forming the appropriate output voltage waveforms and thereby finding the optimal switching pattern. Additionally the cycle time should be as short as possible. The minimum time for the switching interval is t_0 which equates to the maximum switching frequency as; $t_{0\,min} = 1/f_{s\,max}$. Each inverter switch has an associated dead time. Inverter dead time is defined as the delay between switching off and turning on of power devices in the same inverter leg. The deadtime will be considered negligible for the purposes of modeling and; therefore, will not be incorporated into the models discussed in Chapter 3.

V. COMPARISON OF SINUSOIDAL PWM AND SPACE VECTOR MODULATION

The major difference between sinusoidal PWM and the space vector modulation technique is the reference waveform. The result of this difference is the maximum output voltage that can be obtained by the sinusoidal PWM as compared to the space vector modulation. The utilization of the dc bus voltage for the space vector modulation, as stated previously, is approximately 15% more than the sinusoidal PWM technique. The maximum modulation for the space vector method vector is $M_{max}=\frac{2}{\sqrt{3}}$ as compared to the sinusoidal pulse width modulation of $M_{max}=\frac{\sqrt{3}}{2}$ which equates to the 15% increase in the maximum modulation index. The modulation index is the ratio of the reference voltage wave amplitude to the carrier voltage wave amplitude. The magnitude of "M" determines the notch width in the modulated pole voltage waveform and; therefore, controls the fundamental output voltage of the inverter.

When the Modulation Index is greater than 1, over-modulation takes place and pulse dropping results. Pulse dropping is caused by the fact that some of the intersections between the reference and carrier waveforms are avoided due to the reference amplitude exceeding the carrier amplitude when the Modulation Index is greater than 1. The pulse is then dropped. A major disadvantage to this is that the lower order harmonics reappear in the output voltage waveform. An example of pulse dropping is shown in Figure II.6.

Figure II.6 Pulse Dropping



The amplitude of the maximum voltage fundamental reaches approximately 90% of the respective square wave fundamental. The space vector representation has the advantage of lower current harmonics and a higher modulation index as compared to the three phase sinusoidal modulation method. Additionally, the space vector modulation technique can achieve the minimum stator current distortion as well as an increased fundamental voltage capability compared with the sinusoidal PWM inverters.

Chapter 3- MATLAB/SIMULINK Simulations

I. BACKGROUND

The objective of this project was to simulate a control system to provide direct control of both flux and torque of an ac induction motor based on information contained in References (1), (13) - (15). The inverter duty cycle (three phase inverter switching pattern) is calculated twice each switching period based on torque and flux errors as well as the transient reactance (back electromotive force (emf)) of the induction motor. The desired voltage space vector is determined by using the space vector modulation technique, as was discussed in Chapter (2). This results in the inverter being switched in a deadbeat or hysteresis type manner and is based on the torque and flux instantaneous errors in the machine during that switching period. The switching period (T_s) is held constant (500 microseconds) throughout the simulation runs. With the implementation of the stator flux field orientation method (discussed in Chapter 1), no speed feedback of the motor is required to control the torque of the motor.

In the past, Proportional Integral (PI) regulators were commonly used because of easy tuning and zero steady state errors. Controller output is amplitude limited to limit any excursions. PI regulators were used to generate a dq voltage reference based on the input of torque and flux errors. From this, the voltage space vector PWM controlled the inverter. The disadvantage of using the PI regulator is that the

transient response of the control system is limited by the PI regulator. This project eliminates the use of the PI regulator by calculating a voltage space vector, which then provides direct control of torque and flux in a dead beat manner. The advantage of this method is that transient performance is improved.

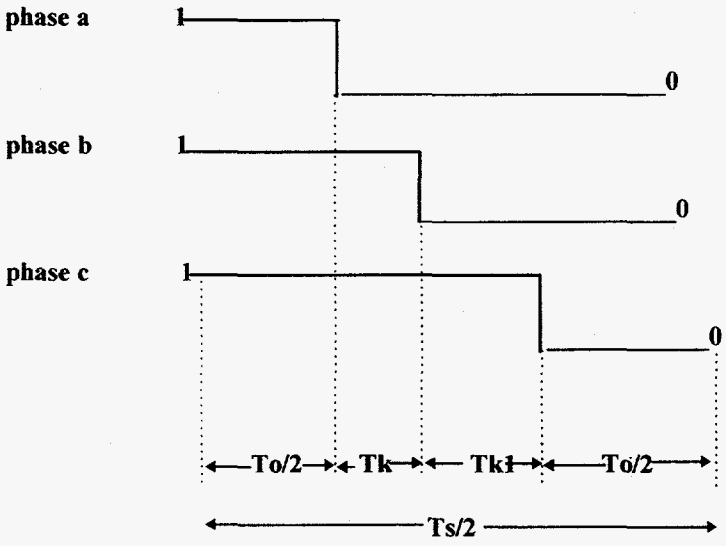
The advantages of torque control in the above discussed manner, includes the elimination of load dependent controller parameters, greater transient response as discussed above, and reduction of over current trips for a given application.

The first step of this controller is to sample the stator current and voltage, perform an analog to a digital signal conversion and transform the three phases to two phases. This provides the current and voltage in a stationary two phase dq reference frame which is used with the stator flux field orientation method. In order to calculate the stator flux vector, the stator resistance must also be known. The stator flux vector is also determined in the dq reference frame.

The expected change in torque during a given switching period is determined from the stator voltage, current and the transient reactance voltage. The change in flux for a given period is based on the stator current and voltage.

The dq voltage vector required for deadbeat control of the flux and torque is then determined. Once the voltage vector is known, the inverter states are established based on the space vector modulation technique. The amount of time in each state is calculated. If the amount of time in any one of the various states is negative, a transient condition exists and is resolved as discussed below. Otherwise, the inverter states are switched in sequence for the required time in order to control the motor in a deadbeat manner. Each phase must change state once during the period "Ts/2". An example of a non-transient (steady state) case would be as follows:

Figure III.1 Example of Steady State Three Phase Cycle



When a transient condition (the value of Tk,Tk1 and/or To is negative) exists in which the torque or flux error is greater than the allowable error, than an alternate method is used. This alternate method is described in the flow chart shown in Figure III.2.

Transient conditions are determined as:

(1) The switching times Tk and Tk1 are determined , initially assuming a steady state (non-transient) condition. If the solution is negative, a torque transient is assumed (most frequently occurring cause of transient) and the process requires performance of Step (2) below. A negative time indicates that the voltage space vector calculated is too large to be synthesized in a single switching period. The objective then becomes to drive the torque toward its reference value while not impacting the dead beat control of flux. If however, the solution results in positive times Tk, Tk1 and T_0, then a steady state condition is present for the switching period; and both torque and flux can be driven to their reference values in a controlled deadbeat manner.

(2) The inverter states are determined for a transient torque condition and the times Tk and Tk1 are calculated based on the inverter states found in Table III.1a. If the solution is positive then the torque transient condition is correct, else, a flux transient is assumed and Step (3) is performed. In order to determine the applicable vector state from Table III.1.a, the angle of the flux vector [Tan^{-1} ($\lambda q/\lambda_d$)] must be known, from which the variable "n" is determined and applied.

(3)The inverter states are determined for a transient flux condition and the times Tk and Tk1 are calculated based on the inverter states found in Table III.1b. If the

solution is positive then the flux transient condition is correct, else a torque and flux transient condition exists and the guidelines of Step (4) apply.

(4) The inverter state for a torque and flux transient is determined from Table III.1c. A single state is selected for the entire period which will tend to drive the torque and flux in the desired direction as quickly as possible.

It is important to note that for all the transient cases (Steps 2-4), the zero state T_0 is not used. This is because the torque and/or flux must be driven in one direction as quickly as possible and; therefore, the use of zero states is eliminated.

The selection of inverter states K and K+1 under transient conditions is determined as follows:

$(2n-3)\pi/6 <$ Flux Angle $[\lambda] < (2n-1)\pi/6$

T*= Reference or desired Torque

F*= Reference or desired Flux

Table III.1a -Torque Transient

Sign of (T-T*)	**K**	**K+1**
Negative	n+1	n+2
Positive	n+4	n+5

Table III.1b- Flux Transient

Sign of (abs(F) - F*)	**K**	**K+1**
Negative	n	n+1
Positive	n+2	n+3

Table III.1c-Torque and Flux Transient

Sign of (T-T*)	**Sign of (abs(F) - F*)**	**K**
Negative	Negative	n+1
Negative	Positive	n+2
Positive	Negative	n+4
Positive	Positive	n+5

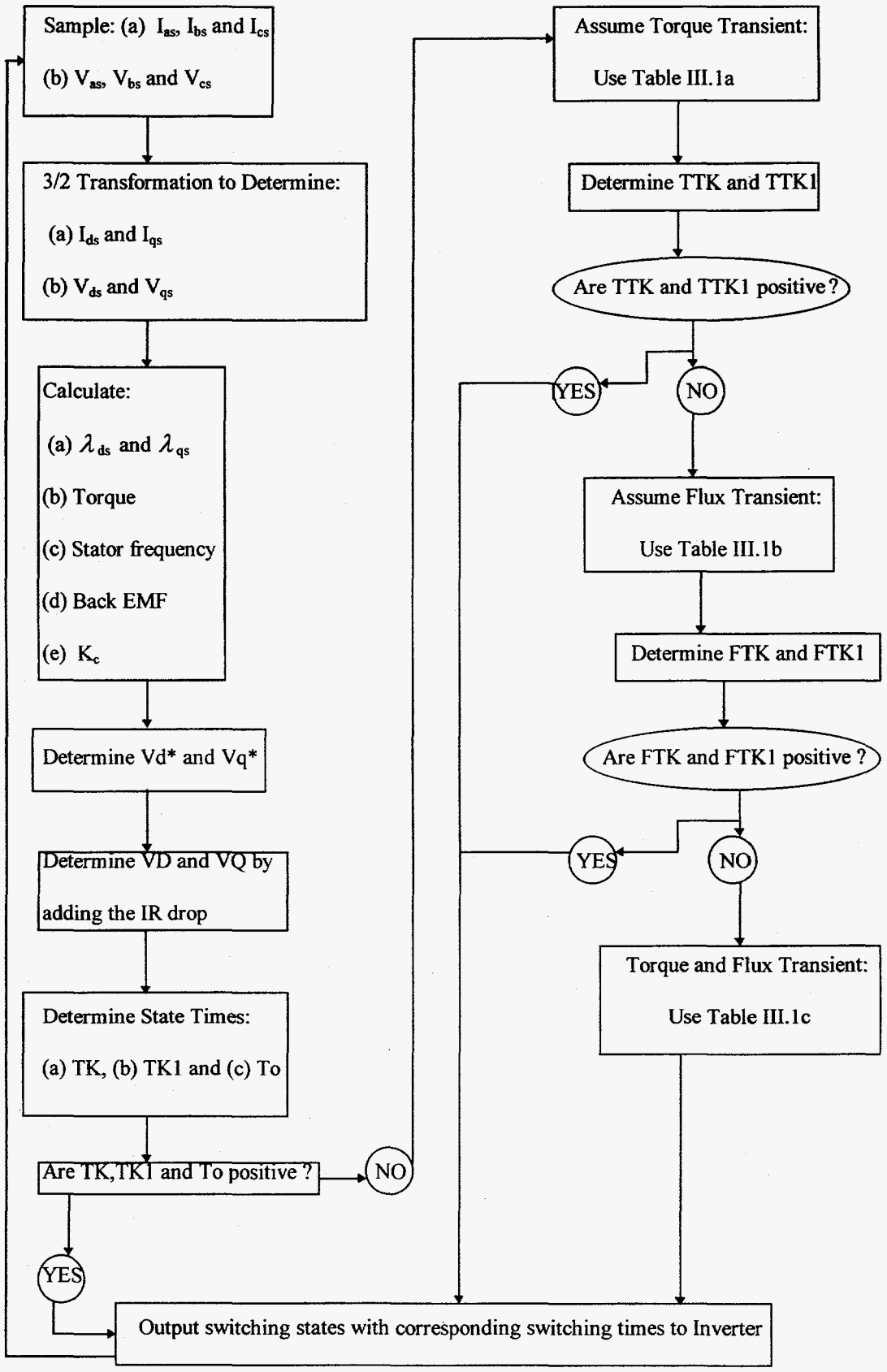
The inverter states for the above transient condition are then expressed as:

Inverter State= k $K \leq 6$

k-6 $K>6$

The flow chart for determining space vector states and associated inverter state times is shown in Figure III.2

Figure III.2- Flow Chart For Control Scheme



To better understand what occurs during a transient condition, consider the following: If during a transient torque condition (see Figure III.3), there exists a flux vector angle of twenty degrees and a stator current vector angle of eighty degrees, then the following is true. Stator voltage vectors 2 and 3 (see Figure II.5) cause the stator current vector and; therefore, torque to increase. Likewise, stator voltage vectors 5 and 6 would cause the torque to decrease. In the case of flux control, states 2 and 6 would cause the flux to increase and states 3 and 5 would cause the flux to decrease. Based on this, states 2 and 3 would be used to drive the flux to its reference value while increasing torque toward its reference value. Table III.2 provides a summary of the above. A similar scenario exists during a flux transient condition.

Figure III.3- Torque Transient Case

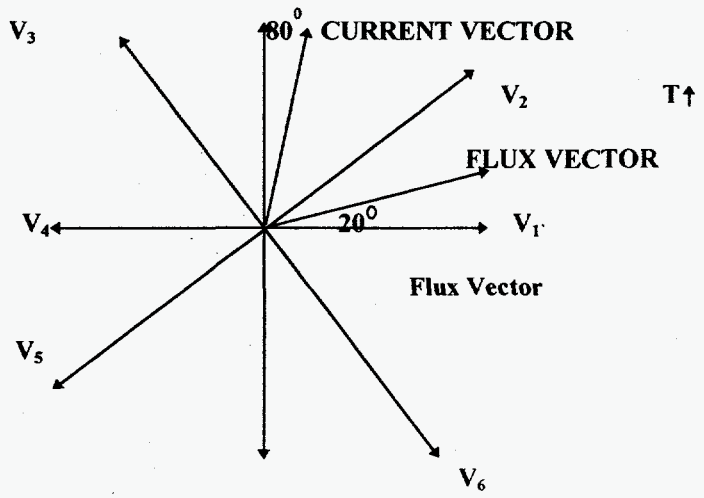


Table III.2 Example of States Used for Transient Torque Condition

STATES	2	3	5	6	Result
Torque	Increases	Increases	Decreases	Decreases	Increases
Flux	Increases	Decreases	Decreases	Increases	No Effect

II. GENERAL

As discussed above, the objective of this portion of the project was to simulate the direct torque control system for an induction motor. The tool that was utilized is SIMULINK which is a program used to simulate dynamic systems. SIMULINK is an extension of MATLAB. The two phases involved in creating a working model consist of:

(1) A Model Definition and

(2) A Model Analysis

Model definition involves the composition of each building block . For example, the induction motor is represented as a simple subsystem block (see Figure III.4) with input and output ports. This is created from individual blocks as shown in Figure III.5.

Figure III.4- Induction Motor-Subsystem Block Diagram

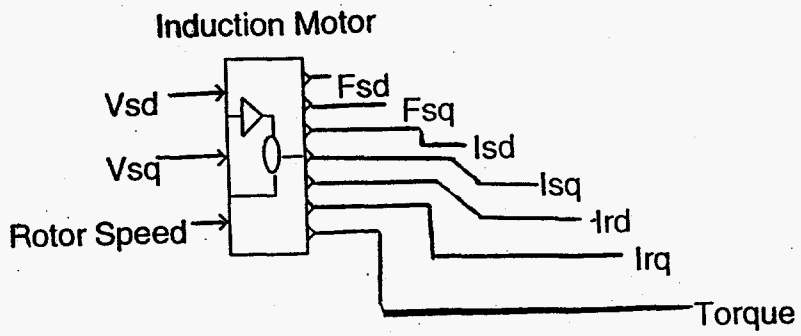
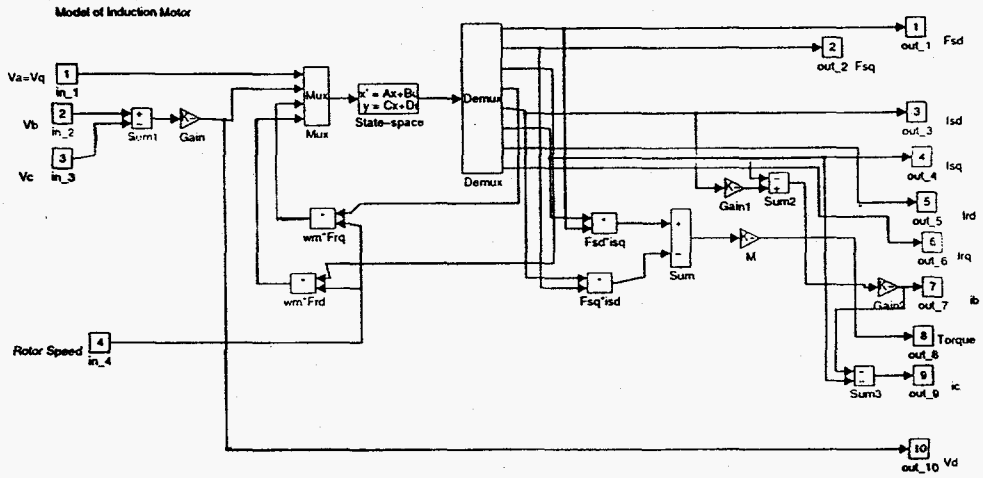


Figure III.5- Induction Motor-Individual Block Diagram Derived from Subsystem Block



The model analysis allows the operator to observe the dynamic response of the system. The progress of a simulation can be viewed while the simulation is running (i.e., using the scope for viewing) and the final results can be viewed using

MATLAB "plots" at the completion of the simulation. The operator has the ability to define the following parameters prior to initiating the simulation:

(1) Start Time

(2) Stop Time

(3) Minimum/Maximum Step Size

(4) Tolerances

(5) Method used for analysis (i.e., Euler, Runge-Kutta, Gear, etc.)

The final model for this project was developed by evaluating less complex models and applying lessons learned to the final model.

(1) *MODEL I* - Modeled direct torque control of induction motor using the MATLAB program only. This allowed a preliminary check of the algorithms that were to be used in the SIMULINK model (see Appendix A.2). Sample plots were generated to ensure that expected results were achieved.

(2) *MODEL II* - Model of Induction Motor using simple transfer function with three phase sinusoidal input.

(3) *MODEL III* - Complex model of the induction motor based on the information provided by Reference (16) as shown in Figures III.4 and III.5.

(4) *MODEL IV* - Modeled sinusoidal pulse width modulation switching scheme as input to the Induction Motor (Model III above).

(5) *MODEL V* - Modeled Space Vector Modulation switching scheme as input to inverter to control the induction motor (Model (III) above).

The above models are discussed in greater detail in the following sections of this Chapter. The purpose of modeling the above was to identify as many "bugs" as possible prior to implementation of the control system using Digital Signal Processing (DSP) hardware detailed in Chapter 4.

The Time Domain technique is used since all time harmonics are automatically incorporated in the description waveforms (i.e., exact solutions of current and flux).This method requires less computing time than the harmonic analysis (Fourier Analysis) method.

Analytical solutions to linear differential equations are used. Linearity is assumed due to the following assumptions: (1) constant rotor speed in the 2-axis voltage equations formulated in the appropriate reference frame and (2) effects of saturation are neglected.

The advantages of digital simulation versus analog simulation include:

(1) changes in software are easier, (2) parameters can easily be modified,

(3) quicker response for obtaining solutions, and (4) the simulation can be speed dependent.

III. MODELS

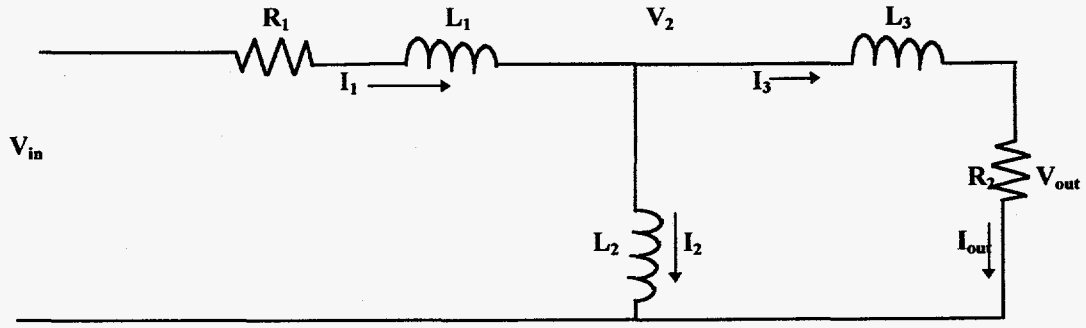
(a) *MODEL I - MATLAB MODEL*

The MATLAB model of a direct torque controlled system is found in Appendix A.2. This model contains the algorithms necessary to calculate the desired voltage space vector states and the switching times associated with those states. This provided a basis for creating the SIMULINK block diagram (Model V).

(b) *MODEL II - TRANSFER FUNCTION MODEL OF INDUCTION MOTOR*

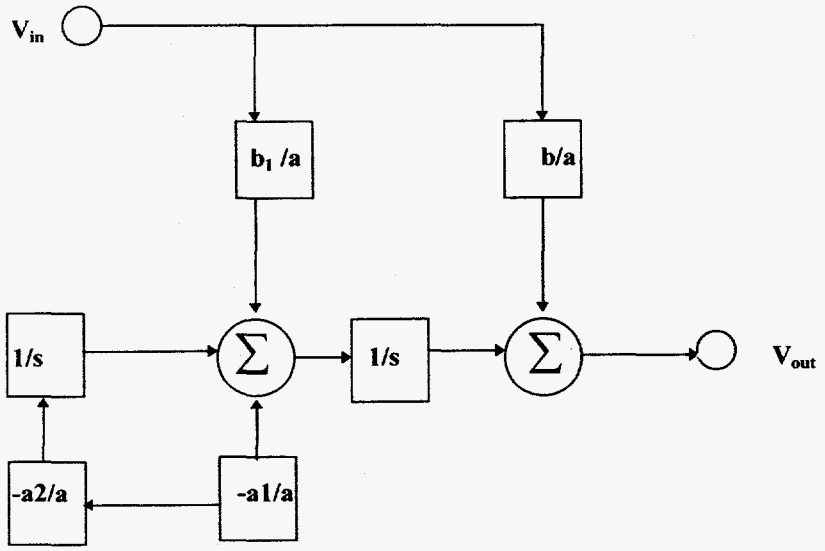
The original induction motor was modeled from the parameters provided in Reference (1).The equivalent single phase representation of the induction motor is as shown in Figure III.6.

Figure III.6-Induction Motor-Single Phase Representation



The Observer Canonical form of this model is shown in Figure III.7.

Figure III.7- Induction Motor-Observer Canonical Form



The transfer function for the induction motor can be represented using the following equations:

$$H(s)= V_{out}/ V_{in} = [s^2 (b) + s(b_1)] / [s^2(a) + s(a_1) + a_2]$$

$$b = L_m L_r \qquad b_1 = L_m R_r$$

$$a = L_s L_m + L_s L_r + L_m L_r \qquad a_1 = R_s L_M + R_s L_r + L_s R_r + L_m R_r \qquad a_2 = R_s R_r$$

where the parameters, per Reference (1), are defined as:

L_1=2.72 mH=L_s= Stator Inductance

L_2=84.33 mH=L_M = Magnetizing Inductance

L_3=3.3 mH=L_R = Rotor Inductance

R_1 =.371 ohms=R_S =Stator Resistance

R_2=.415 ohms=R_R= Rotor Resistance

$$H(s)= [s^2 (2.783e\text{-}04) + s(0.03499)]/[s^2(5.16e\text{-}04) + s(0.0686) + (0.154)]$$

A more realistic representation (as demonstrated using the SIMULINK tool) of this system can be shown by evaluating the output current vice output voltage.

$H(s)= I_{out}/ V_{in}$

$I_1 = I_2 + I_3 = I_{in} = I_2 + I_{out}$

$[V_{in} - V_2] /[R_1 + L_1 s] = V_2/[L_2 s] + [V_2 - V_{out}]/[L_2 s]$

$V_2=V_{in}(L_2 s)/[R_1 + L_1 s + L_2 s]$

$V_2= I_{out} L_3 s + Iout R_2 = I_{out} (L_3 s + R_2)$

$I_{out} (L_3 s + R_2)=V_{in}(L_2 s)/[R_1 + L_1 s + L_2 s]$

$H(s)= L_2 s / [s^2(L_1 L_3 + L_2 L_3) + s(R_1 L_3 + R_2 L_1 + L_2 R_2) + R_2 R_1]$

The resulting transfer function using the above values results in:

$H(s)= 84.33 \text{ e-}03/ [s^2 (2.873 \text{ e-}04) + s(.0373) + .1539] = I_{out}/V_{in}$

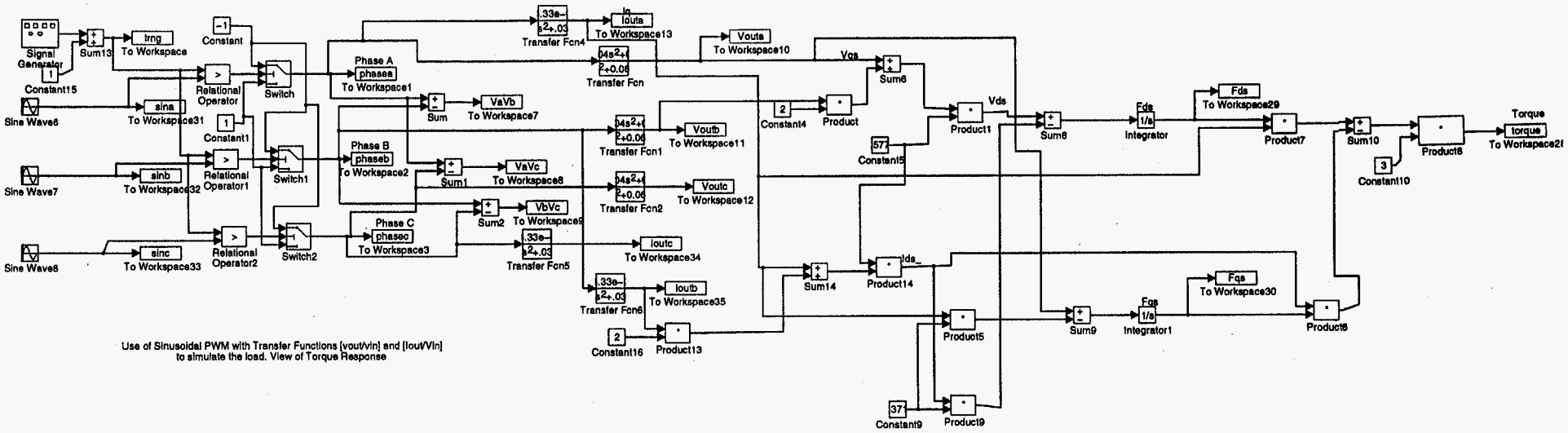
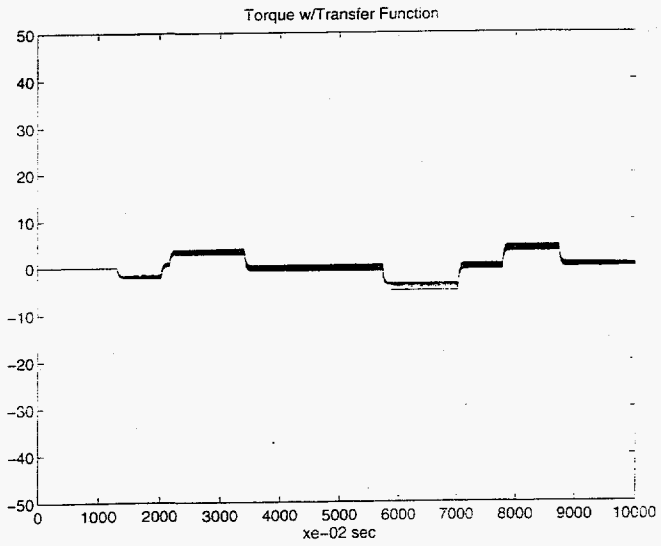


Figure III.8 Transfer Function Model of Three Phase Induction Motor

Figure III.8 represents the SIMULINK Block diagram of the Transfer Function Model of the three phase induction motor. Figure III.9 shows the output torque's behavior, from the Figure III.8 model, cycles around a steady state point of zero, representing a no load condition. The model provides an initial attempt at modeling an induction motor. For transient cases, the results of this model are limited. A more detailed model as described later in this chapter provides a better representation of an induction motor.

Figure III.9 Torque Based on Transfer Functions



(c) *MODEL III - DETAILED MODEL OF INDUCTION MOTOR*

A more complex induction motor model was added replacing the transfer function used in III.b above.

The parameters of the induction motor used in the SIMULINK program are specified as:

Rsm=.007565 [Stator Magnetizing Resistance]

Rrm=.00596 [Rotor Magnetizing Resistance]

Lhm=4.559e-03 [Magnetizing Inductance of the Motor]

IsIm=.1187e-03 [Stator Magnetizing Current]

LrIm=.1692e-03 [Rotor Magnetizing Inductance]

ppairs=3

Lsm=(Lhm)(IsIm) [Stator Inductance] Lrm=(Lhm)(LrIm) [Rotor Inductance]

Det=(Lrm)(Lsm)-Lhm^2 Mm=3 (ppairs)/2

MI1=Lrm/Det MI2=Lsm/Det MI3=-Lhm/Det

RSMI1=(-Rsm)(MI1) RSMI2=(-Rsm)(MI2) RSMI3=(-Rsm)(MI3)

RRMI1=(-Rrm)(MI1) RRMI2=(-Rrm)(MI2) RRMI3=(-Rsm)(MI3)

From the above parameters a state space model as shown below can be created:

STATE SPACE MODEL

$$\dot{X} = AX + BU$$

$$Y = CX + DU$$

where A,B,C and D are defined as:

$$A = \begin{bmatrix} RSMI1 & 0 & RSMI3 & 0 \\ 0 & RSMI1 & 0 & RSMI3 \\ RRMI3 & 0 & RRMI2 & 0 \\ 0 & RRMI3 & 0 & RRMI2 \end{bmatrix} = \begin{bmatrix} -26.75 & 0 & 25.796 & 0 \\ 0 & -26.75 & 0 & 25.796 \\ 20.32 & 0 & 20.85 & 0 \\ 0 & 20.32 & 0 & 20.85 \end{bmatrix}$$

$$B = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

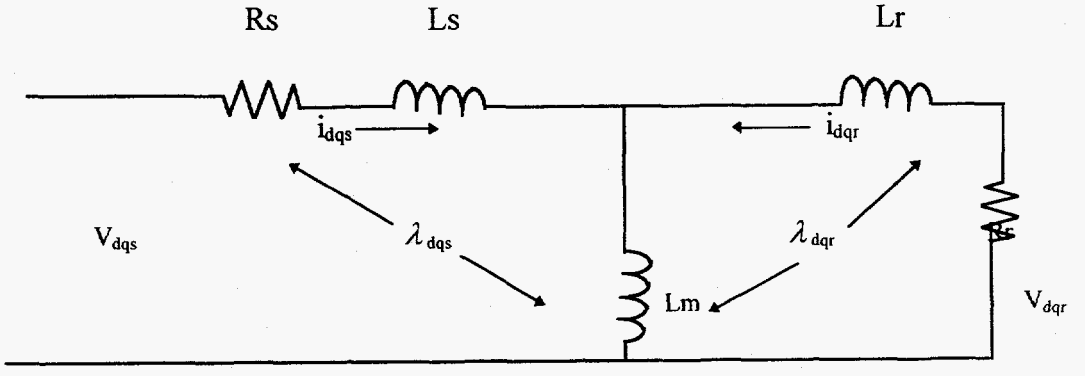
$$C=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ MI1 & 0 & MI3 & 0 \\ 0 & MI1 & 0 & MI3 \\ MI3 & 0 & MI2 & 0 \\ 0 & MI3 & 0 & MI2 \end{bmatrix}$$

$$C=\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 3536.42 & 0 & -3409.87 & 0 \\ 0 & 3536.42 & 0 & -3409.87 \\ -3409.87 & 0 & 3498.65 & 0 \\ 0 & -3409.87 & 0 & 3498.65 \end{bmatrix}$$

$$D=\begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$X=\begin{bmatrix} Fsd \\ Fsq \\ Frd \\ Frq \end{bmatrix} \qquad Y=\begin{bmatrix} Fsd \\ Fsq \\ Frd \\ Frq \\ Isd \\ Isq \\ Ird \\ Irq \end{bmatrix} \qquad U=\begin{bmatrix} Vd \\ Vq \\ Wm * Frq \\ Wm * Frd \end{bmatrix}$$

Proof of the State Space Equations can be derived as:



Per the state space equations we know that:

$$i_{sd} = MI1\,\lambda_{sd} + MI3\,\lambda_{rd}$$

$$i_{sq} = MI1\,\lambda_{sq} + MI3\,\lambda_{rq}$$

and from Reference (17) we know that:

$$\lambda_{dqs} = L_s\, i_{dqs} + L_M\, i_{dqr/}$$

$$\lambda_{dqr} = L_M\, i_{dqs} + L_r\, i_{dqr}$$

therefore,

$$i_{dqs} = \lambda_{dqs}/L_s - L_M\, i_{dqr}/L_s$$

$$i_{dqr} = \lambda_{dqr}/L_r - L_M\, i_{dqs}/L_r$$

and,

$$i_{dqr} = \lambda_{dqr}/L_r - L_M/L_r\,(\lambda_{dqs}/L_s - L_M i_{dqr}/L_s)$$

$$i_{dqr} = \lambda_{dqr}/(L_r(1-L_M^2/L_rL_s)) - L_M\,\lambda_{dqs}/\,(L_rL_s(1-L_M^2/L_rL_s))$$

From definitions provided above we know that

$$MI3 = -Lhm/(Lhm+IsIm)\,(Lhm+LrIm) - Lhm^2 = -L_M/\,(L_sL_r - L_M^2)$$

$$MI2 = Lsm/DET = Lhm+IsIm/Lrm*Lsm-Lhm^2 = 1/\,(L_r(1-L_M^2/L_rL_s)) = 1/[Lr-Lm^2/Ls]$$

given that,

L_M=Lhm L_S=Lhm + IsIm L_R=Lhm +LrIm

$\dot{\lambda}_{sd}$ = Fsd = (RSMI1)(Fsd) + RSMI3(Frd) + Vd

$\dot{\lambda}_{sq}$ = Fsq = (RSMI1)(Fsq) + RSMI3(Frq) + Vq

$\dot{\lambda}_{rd}$ = Frd = (RRMI3)(Fsd) + RRMI2(Frd) -WmFrd

$\dot{\lambda}_{rq}$ = Frq = (RRMI3)(Fsq) + RRMI2(Frq) +Vd +WmFrq

Stator flux is determined by using the following equation $\lambda = \int (V_s - I_sR_s)$;

therefore,

$d\lambda = V_s - I_sR_s$

$\dot{Fsd}$= Vd + (RSMI1)(Fsd) +(RSMI3)(Frd)

- [(RSMI1)(Fsd) +RSMI3(Frd)]=Ids Rs

$\dot{Fsq}$= Vq + (RSML1)(Fsq) + RSML3(Frq)

- [(RSML1)(Fsq) + RSML3(Frq)]= Iqs Rs

The inputs to the induction motor model are identified as V_d , V_q and rotor speed. The dq transformed voltages are derived from phase voltages V_a , V_b and V_c as: $V_q=V_a$ and $V_d=(V_b-V_c)/\sqrt{3} = (V_a+2V_b)/\sqrt{3}$. As stated previously in Chapter (1), transformation from a coupled three phase system into a decoupled dq frame (similar to a dc machine) simplifies control of the system since the complex interactions of a three phase system no longer apply. The output of the induction motor is then transformed back to a three phase complex system ((2/3) transformation).

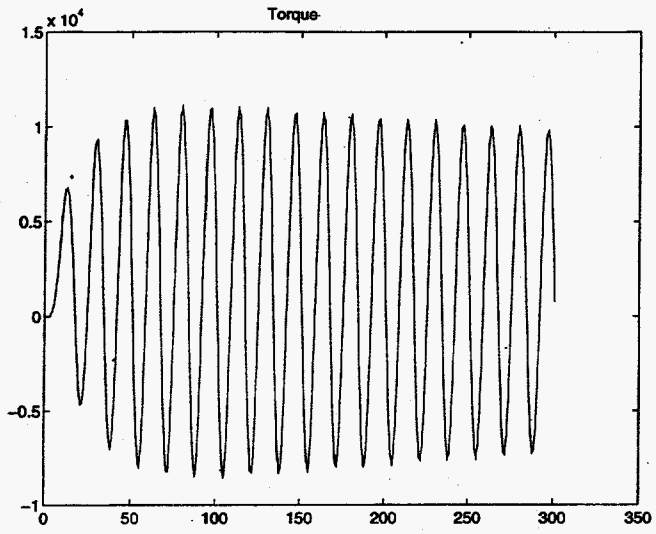
From the above state space equations, the torque can be determined using the following equation which is contained within the sub-block induction motor model (labeled as output 8 on Figure III.5).

T= 3 p/4 (Fsd*Isq -Fsq*Isd)

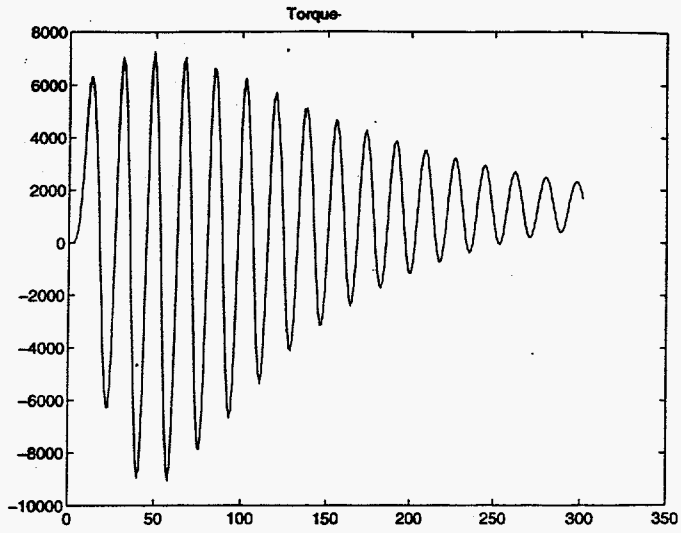
The speed of the rotor was varied from 0 to 1.2 per unit. As was shown in Figure I.6, previously, when the speed of the rotor is in the range of 0 -1, torque continually increases until breakdown torque is reached. After breakdown torque is reached, torque rapidly decreases until the speed reaches 1. This region (0-1) is known as the motoring region of operation. Once the speed increases above 1, a negative torque results. This is known as the regeneration region. The results of speed adjustment and its effect on torque are illustrated in Figure III.10.

Figure III.10 Speed (Slip) Adjustments- Effect on Torque Response (Sheet 1 of 2)

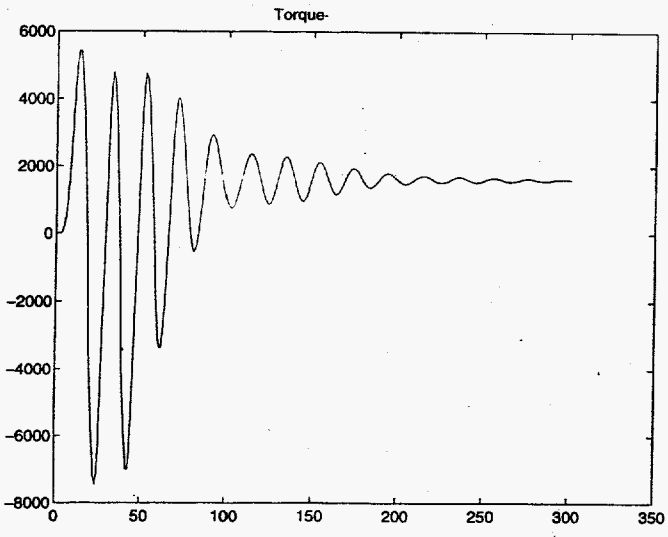
a. Speed=0



b. Speed=0.1



c. Speed=0.2



d. Speed=0.5

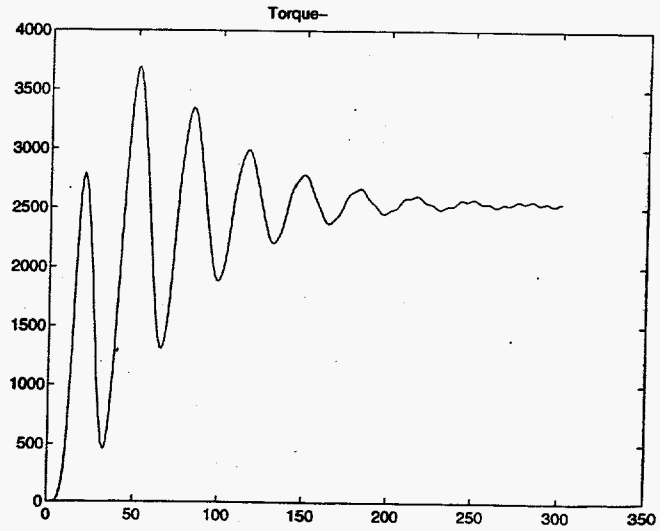
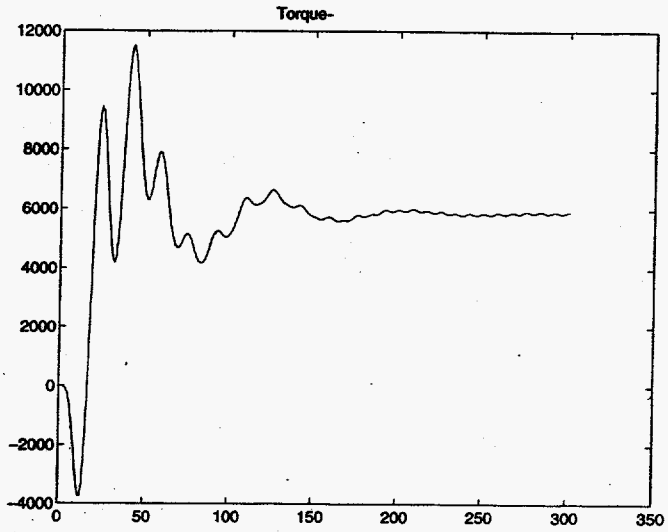
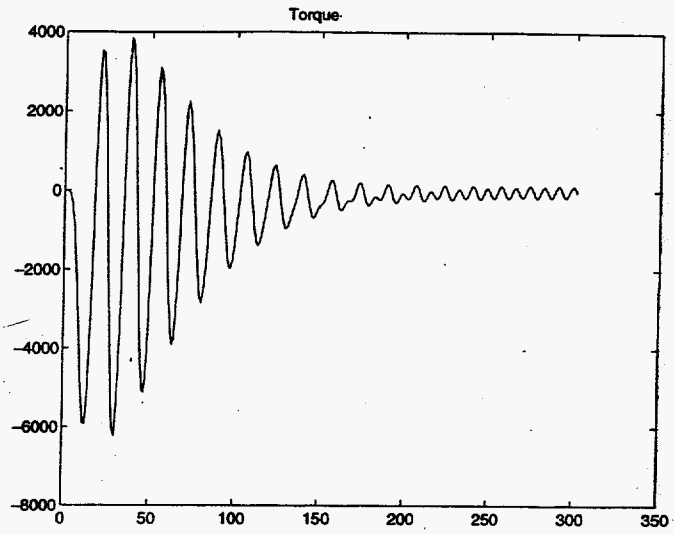


Figure III.10 Speed (Slip) Adjustments- Effect on Torque Response (Sheet 2 of 2)

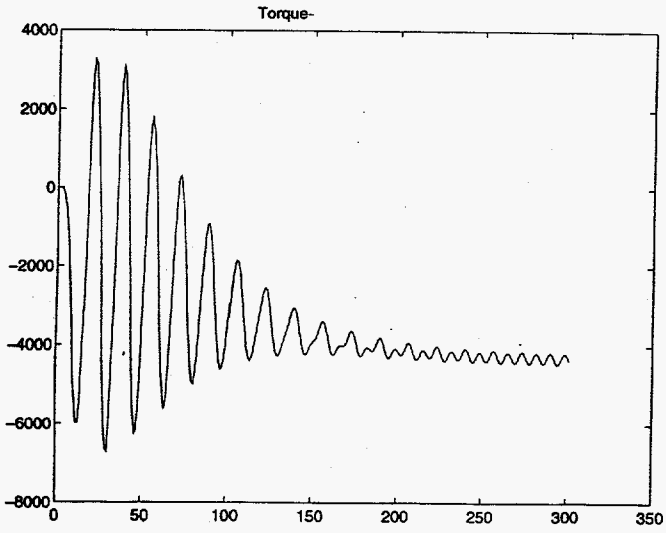
e. Speed=0.8



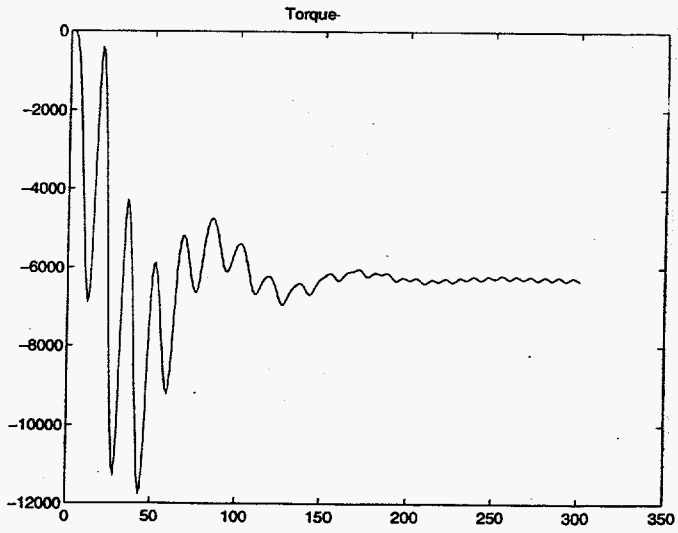
f. Speed=1.0



g. Speed=1.01036



h. Speed=1.2



(d) MODEL IV - SINUSOIDAL PULSE WIDTH MODULATION

Figure III.11 represents the block diagram of this model. The same induction motor model that was used in III.c above is used for this model. The signal generator provided the carrier wave (triangular wave) which determines the switching frequency. Sina, sinb and sinc provide the control waveform for phases a, b and c, respectively; and represent the desired fundamental or modulating frequency (60 Hertz). As the switching frequency is increased (varied from 15x -33x the fundamental frequency) , the filtering of harmonics becomes easier but results in more switching losses. This project did not focus on determining the magnitude of these switching losses. The amplitude of Vtri was maintained greater than Vcontrol , thereby maintaining the modulation index less than one. This eliminates over-modulation or pulse dropping. However, it should be noted that over-modulation is commonly used with induction motors.

The advantage of over-modulation is that the maximum possible value of the fundamental voltage will be obtained. The disadvantage of dropping pulses is that . low order harmonics will be apparent in the output voltage waveform. The addition of filters eliminates the concern for low order harmonics in the output voltage waveform.

The frequency modulation index, M_f, (as discussed in Chapter 2) was varied as stated above. Based on the fact that synchronous PWM was used in this model, an odd integer value was chosen for M_f (i.e., 15, 21, 33). The control and carrier waveforms are shown in Figure III.12. Note that the carrier wave peak value varies from approximately 7.8 to 9.8. In reality the carrier wave was observed to have an amplitude of 10.0. It was noted that as the frequency of the carrier wave was increased (i.e. from 15x-33x the fundamental frequency) the more distorted the wave became. This could not be adjusted and appears to be a function of the SIMULINK program. Therefore, the maximum frequency used for the carrier wave was 7916.8135 rads/sec or 21x the fundamental frequency (376.99 rads/sec). The amplitude of the fundamental waveforms (phases a, b and c) was set at 6.

Figure III.11- Sinusoidal Pulse Width Modulated SIMULINK Model

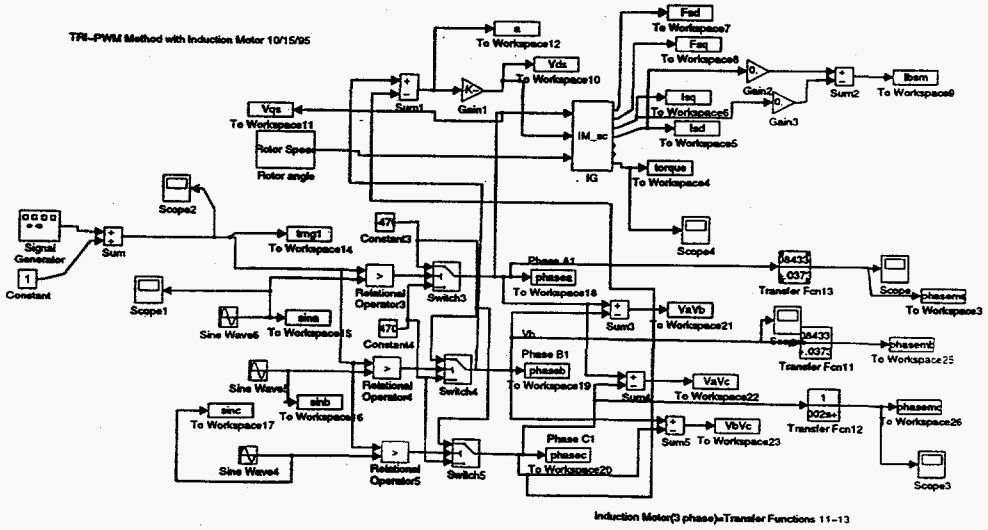
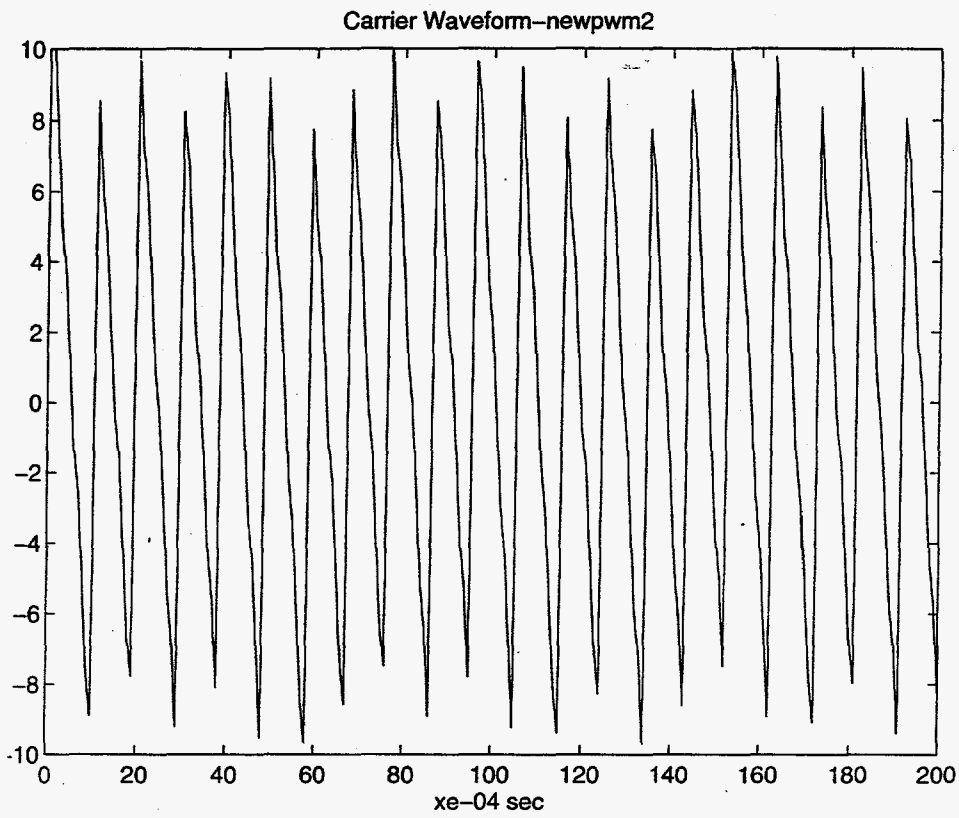
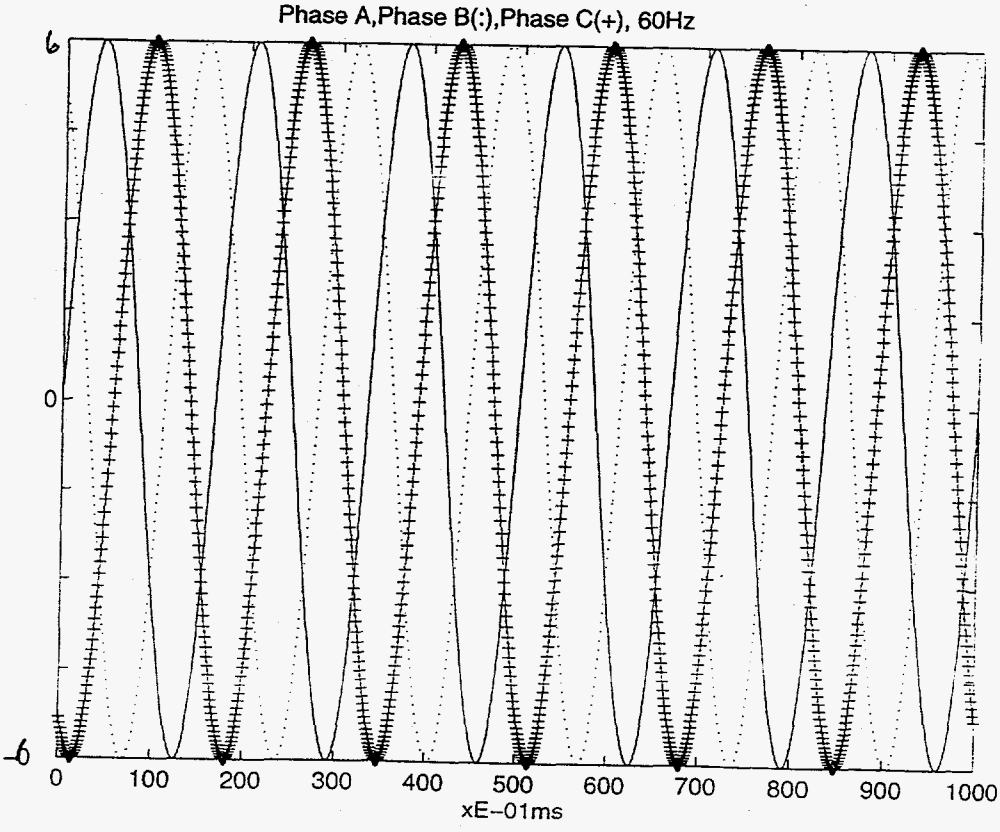


Figure III.12- Carrier and Control Waveforms for Sinusoidal PWM

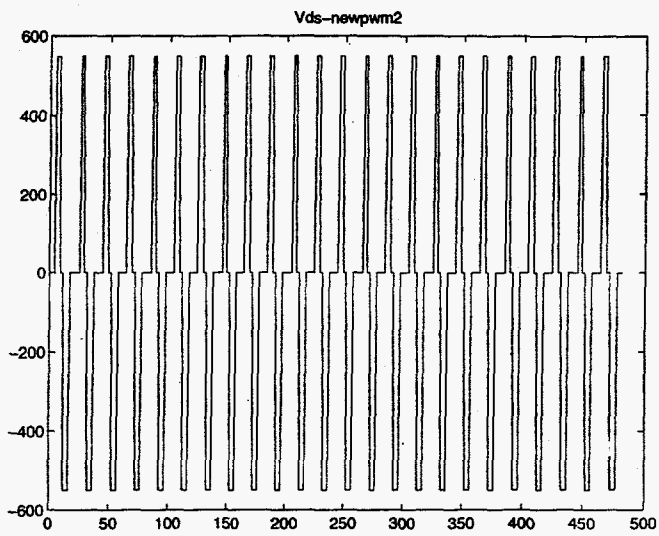


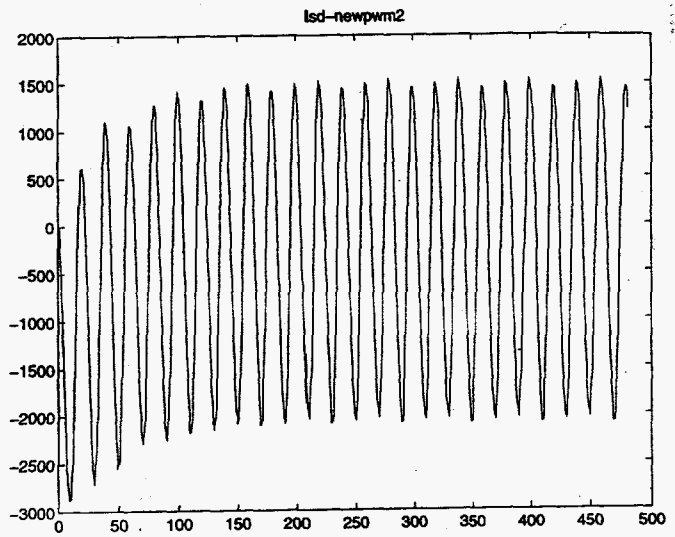


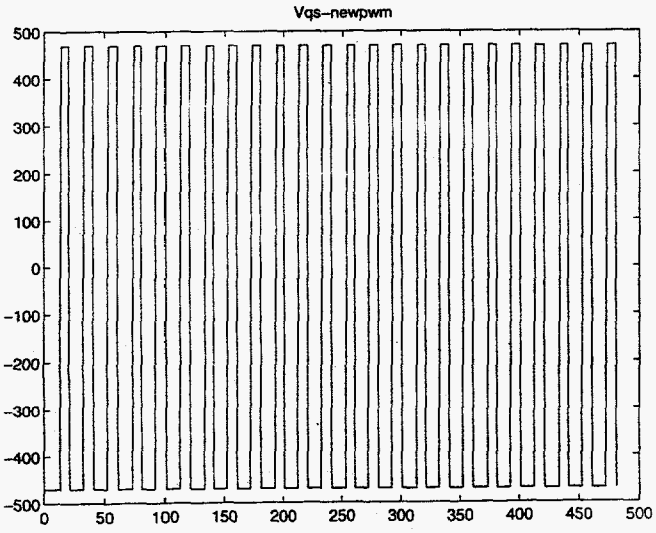
The three phase input sinusoidal waveforms are 120 degrees apart from one another. This was previously identified in Chapter (1) as one of the requirements for transformation into the dq reference frame.

The result of this sinusoidal switching scheme for phases a, b and c is shown in Figure III.13 respectively. Transfer functions 11, 12 and 13 represented output filters. The addition of filters provided minimal effect on the simulated output and were subsequently deleted for the SIMULINK model only. Outputs VaVb, VaVc and VbVc provide the line-line voltages (see Figure III.14). The line-line voltage provides a sinusoidal type response. The torque and flux response (Figure III.15) shows initial settling time is required (approximately 200 msec) for the torque until a steady state band results (in this case approximately -700 ft-lbs).

Figure III.13 Sinusoidal PWM Output Voltage/Current Waveforms







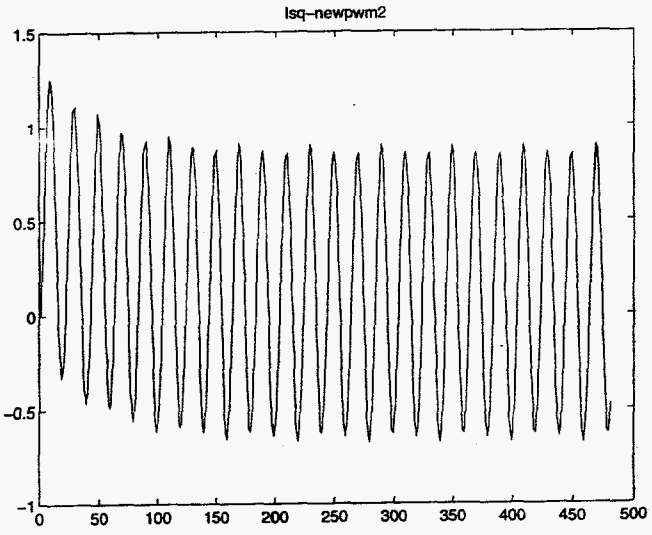


Figure III.14- Sinusoidal PWM Line-Line Output Voltage Waveforms

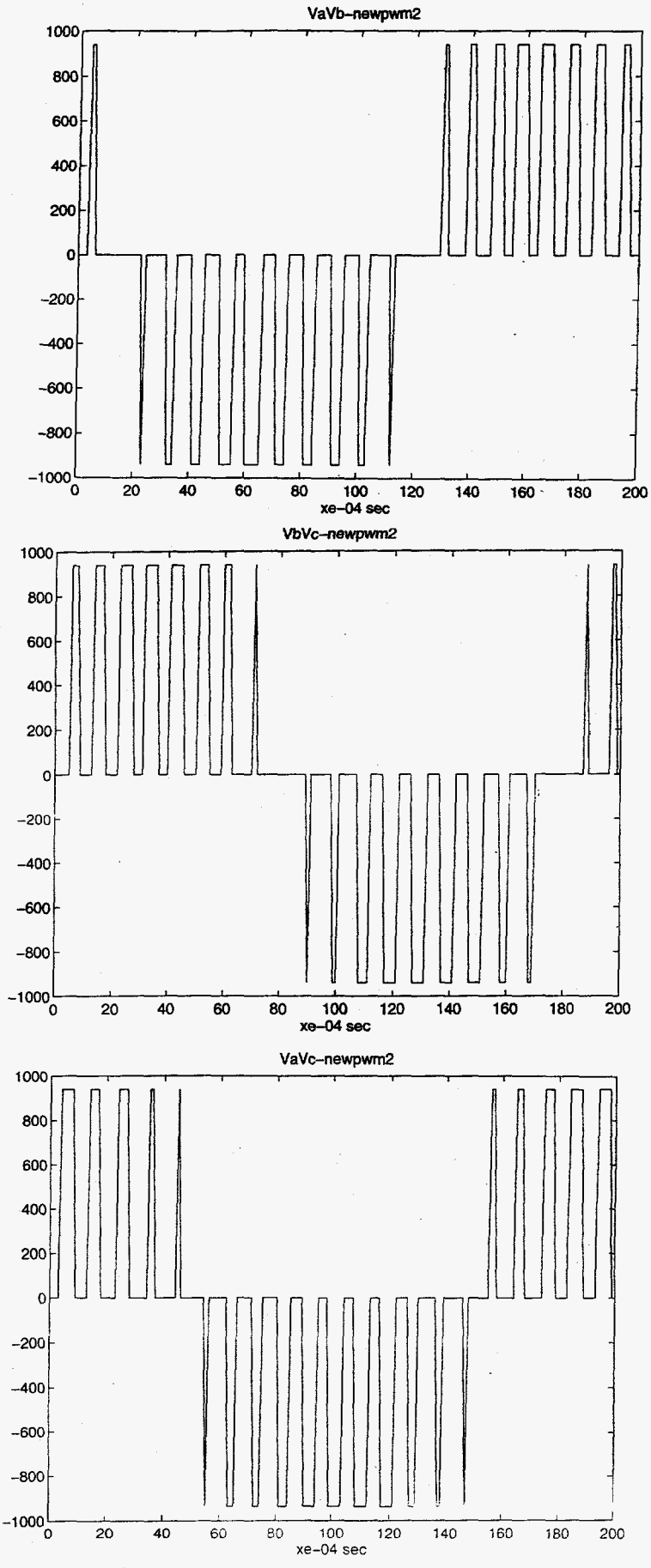
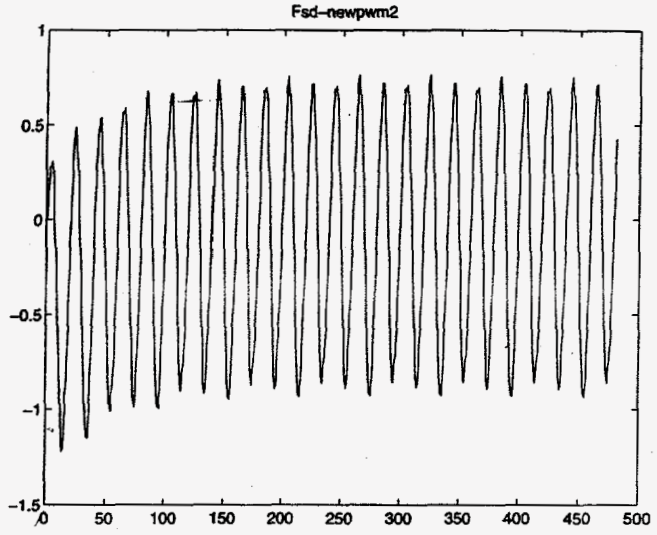
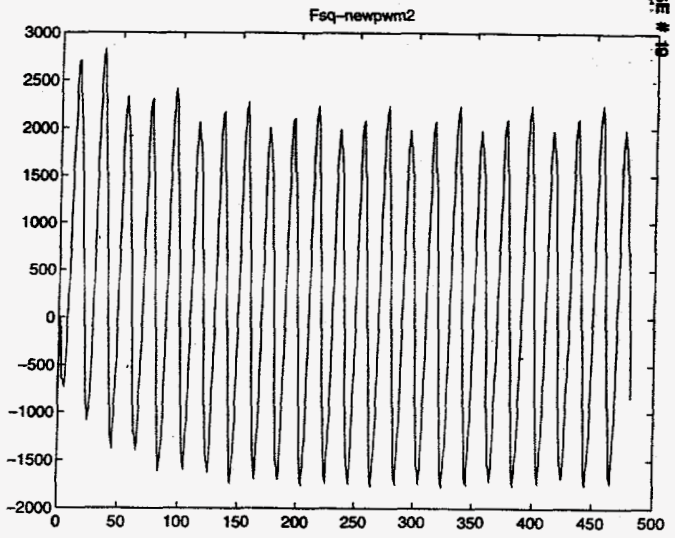
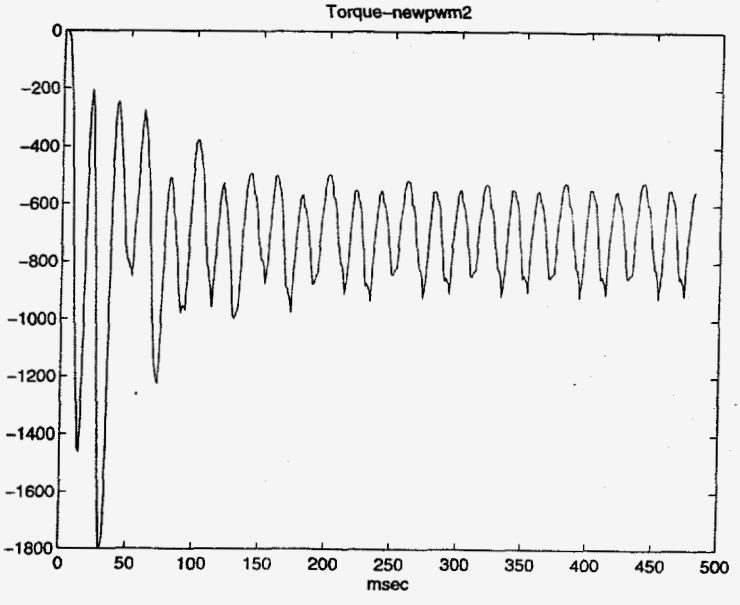


Figure III.15- Torque and Flux Response with Sinusoidal PWM







(e) MODEL V - SPACE VECTOR MODULATION SWITCHING SCHEME

The objective of this model is to calculate the voltage space vector twice each switching period. The switching period remains fixed during the entire analysis. The flux (dq), current (dq) and torque output of the motor are used in several algorithms as shown in Appendix B. Once the voltage space vector and associated switching times are calculated, the a, b and c phase states (0 or 1) are known and fed back to the inverter switches to control torque and flux.

Based on Reference (1), a Block Diagram was created as shown in Figure III.16: The SIMULINK model of the Direct Stator Flux Field Orientation Controller as represented in Figure III.17 was constructed based on this Block Diagram. The individual sub-blocks as labeled in Figure III.17 are found in Appendix B.

Initially, for a fixed period of time the simulation is run by feeding a sinusoidal PWM signal into the induction motor (this is model IV). This allows the induction motor to reach a steady state condition. The time should be long enough to have torque reach a steady state condition (for this case approximately 150 msec).

The SIMULINK Technical Manual discusses the use of algebraic loops and states: "Algebraic or implicit loops occur when two or more blocks with direct feed through of their inputs form a feedback loop. When this occurs, SIMULINK must perform iterations at each step to determine whether there is a solution to this problem. Algebraic loops considerable reduce the speed of a simulation and may be unsolvable..."

Initially, there was some difficulty with the model based on the large number of algebraic loops created in the model. In order to effectively break these loops, a delay time of one cycle and a zero order hold was added to several parameters. This results in calculating voltage space vectors and associated times in one cycle and applying them to the inverter during the next cycle, as would be the case for the actual implementation of this control scheme. Due to the large number of algorithms that are exercised each cycle, the simulation run itself takes a long period of time to complete (i.e., approximately sixty minutes for a 400 msec run). This problem will not exist when implemented on the DSP 21020, since one of the major advantages of the DSP is the speed at which it can compute values to feed back to the inverter.

Two hundred msec after the simulation commenced, the inverter was switched to the space vector modulation scheme. As results indicate, the torque levels off very quickly, to the new desired level.

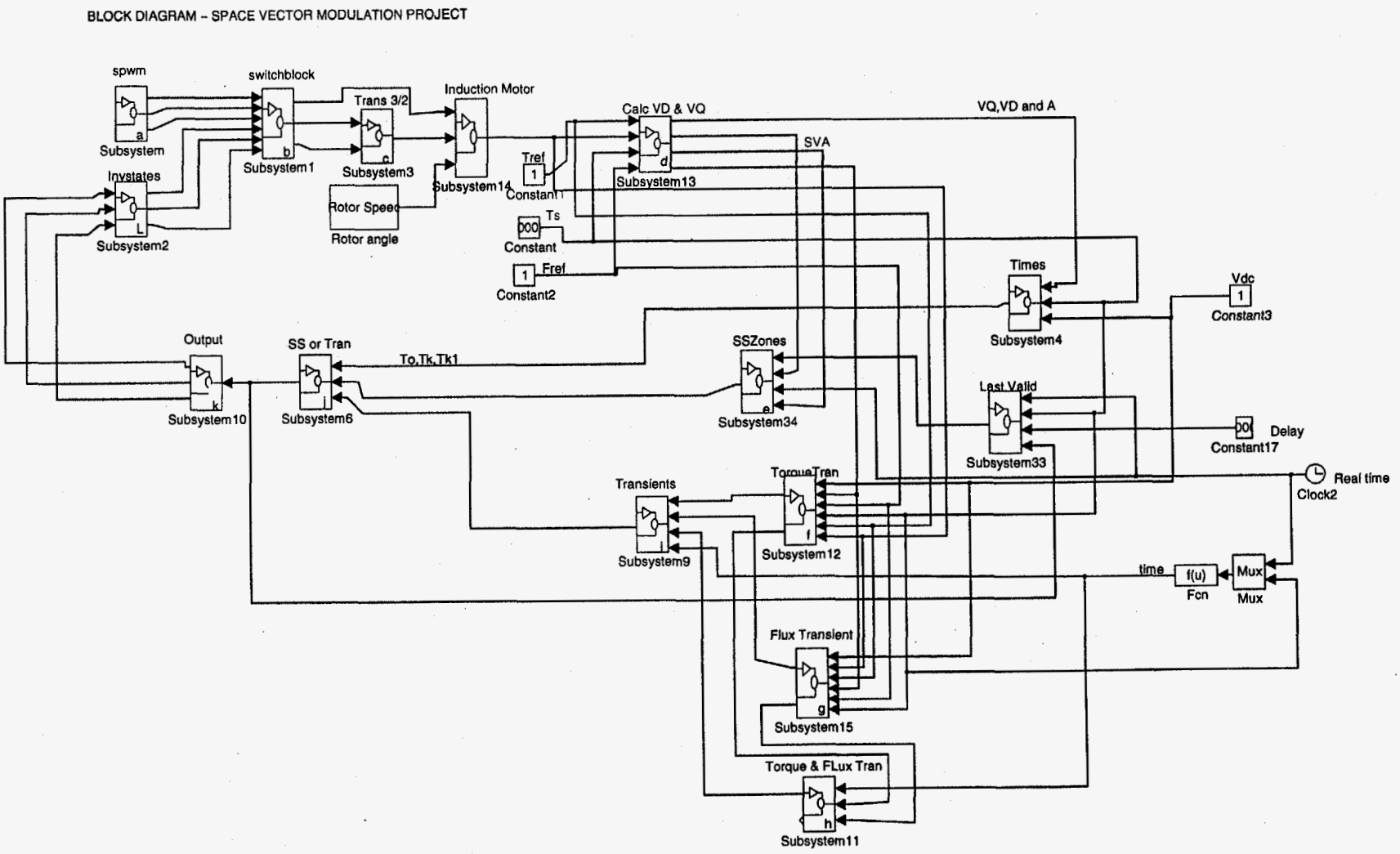


Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 1 of 13)

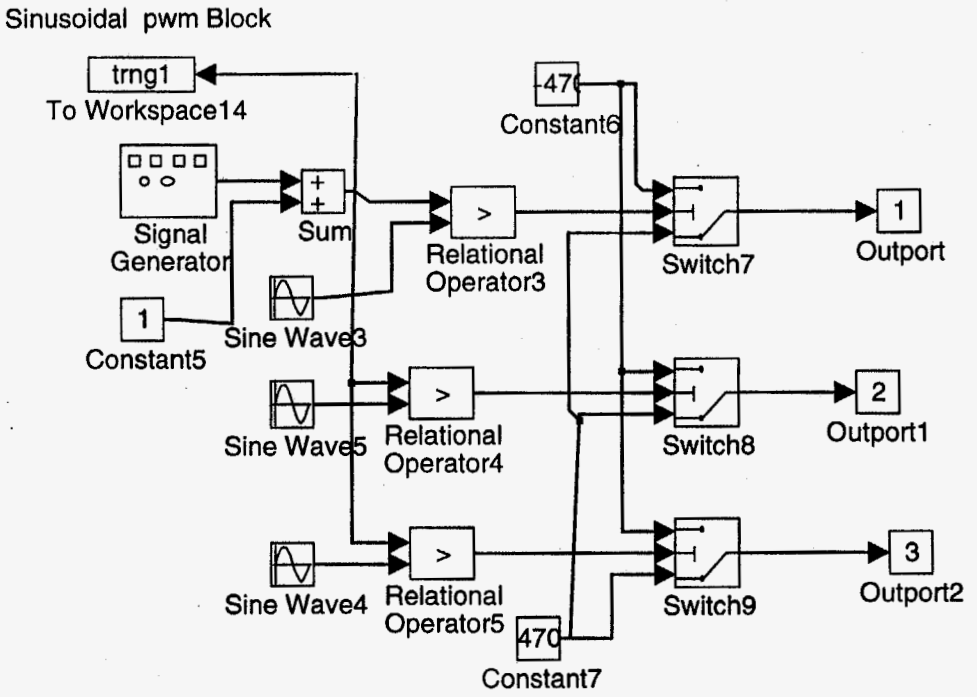


FIGURE III.16a–Sinusoidal PWM Control

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 2 of 13)

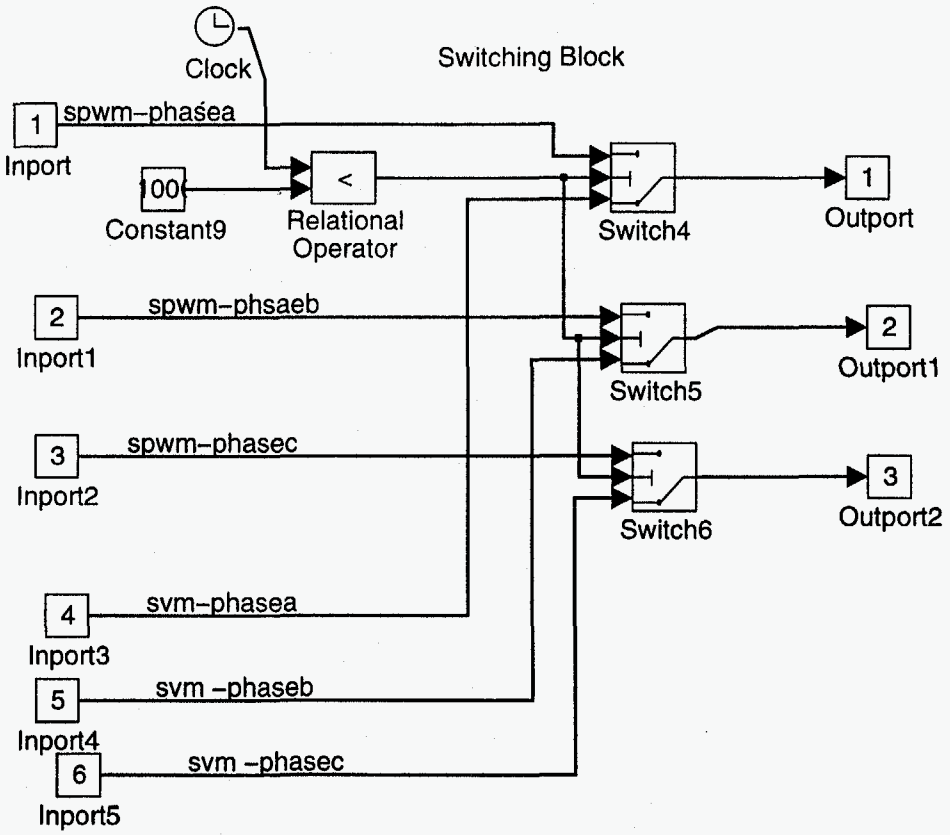


FIGURE III.16.b– Switching between sinusoidal pwm abd space vector modulation

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 3 of 13)

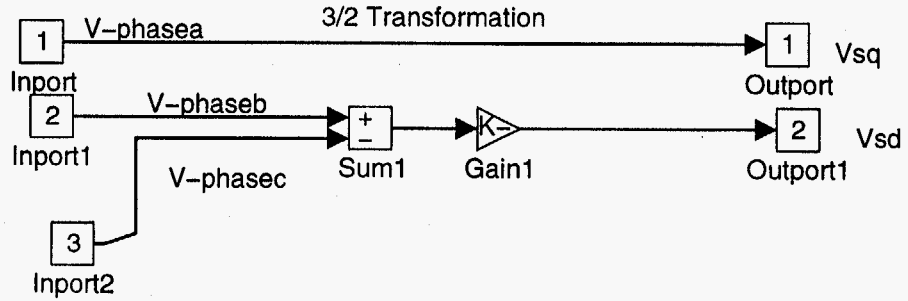


FIGURE III.16c– Transformation from Three Phase to Two Phase System

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 4 of 13)

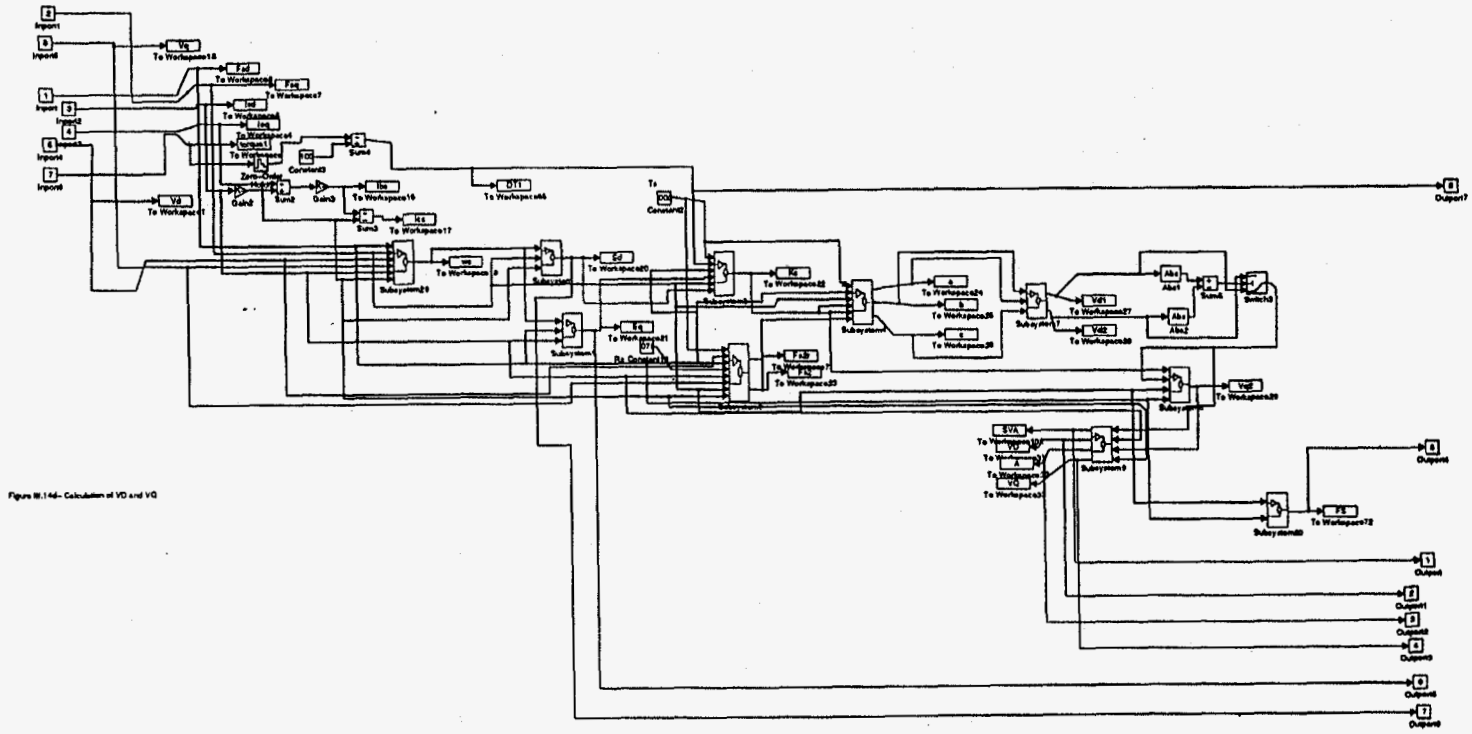


Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 5 of 13)

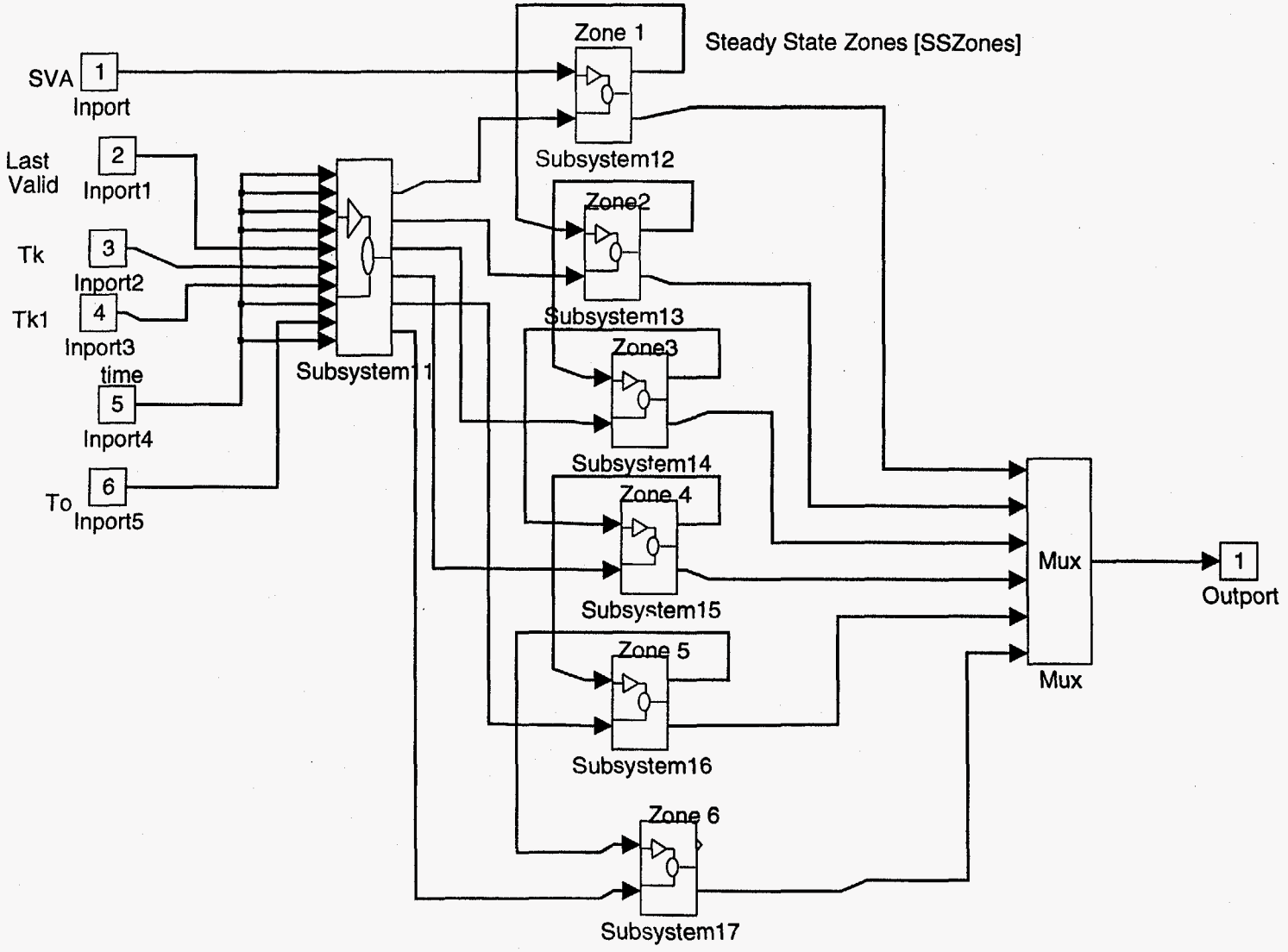


FIGURE III.16e–Determination of States for Steady State Condition

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 6 of 13)

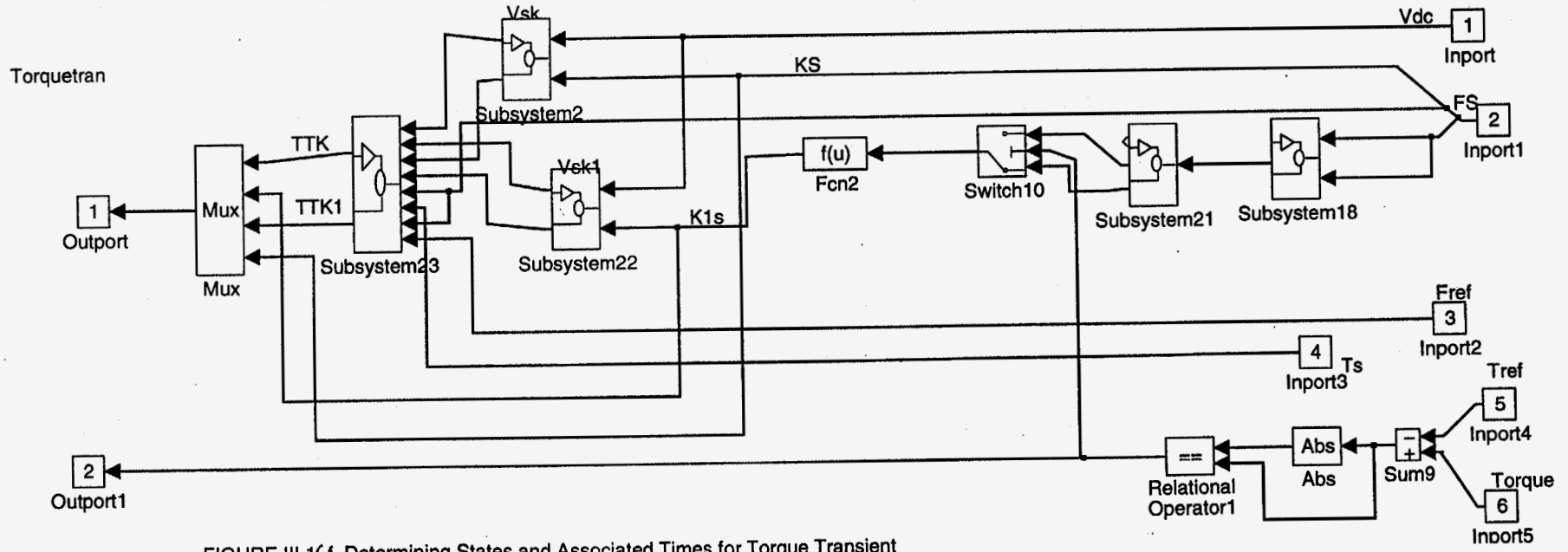


FIGURE III.16f–Determining States and Associated Times for Torque Transient

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 7 of 13)

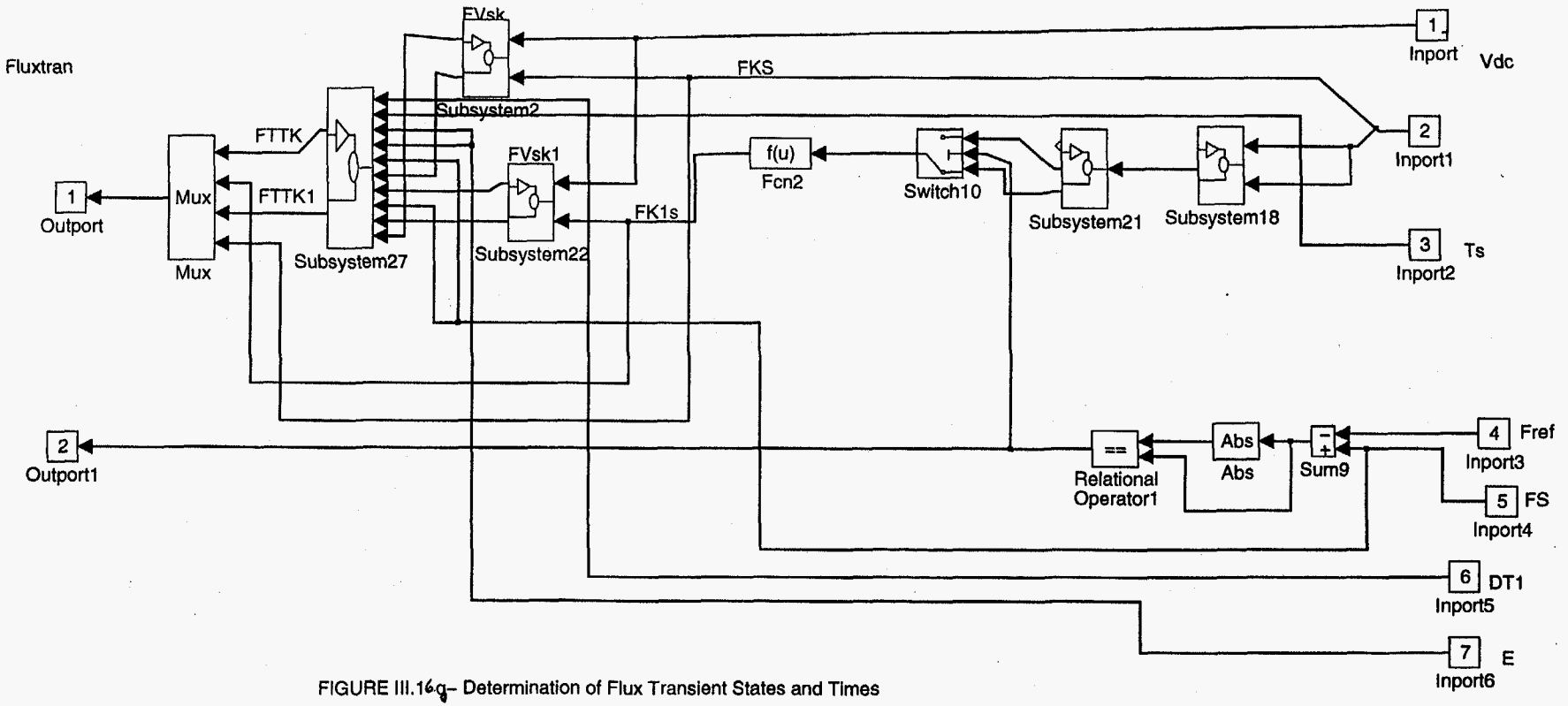


FIGURE III.16g– Determination of Flux Transient States and Times

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 8 of 13)

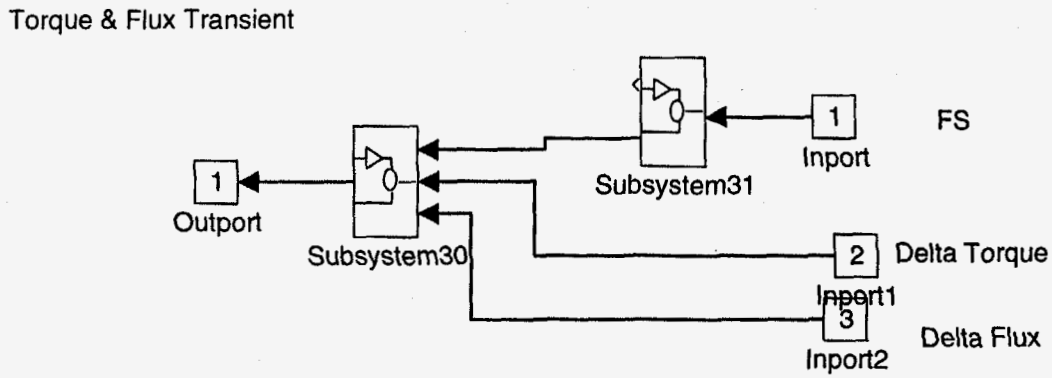


Figure III.16h–Determining State for Torque and FLux Transient

Figure III.14 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 9 of 13)

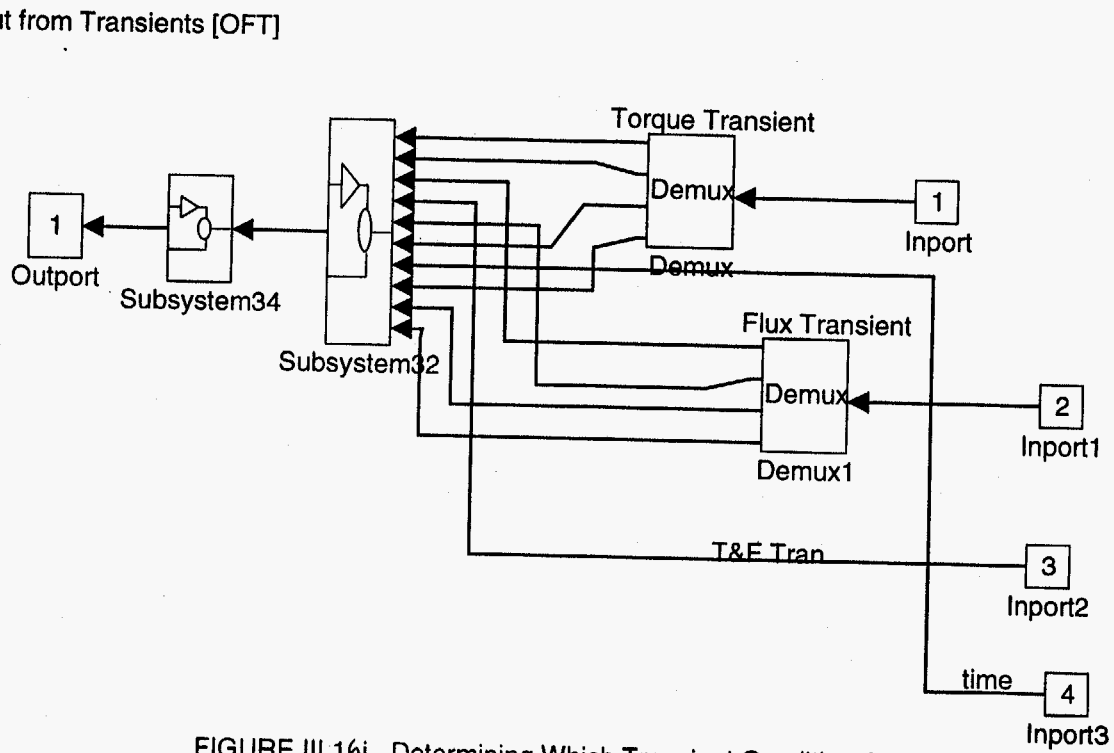


FIGURE III.16i– Determining Which Transient Condition Applies

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 10 of 13)

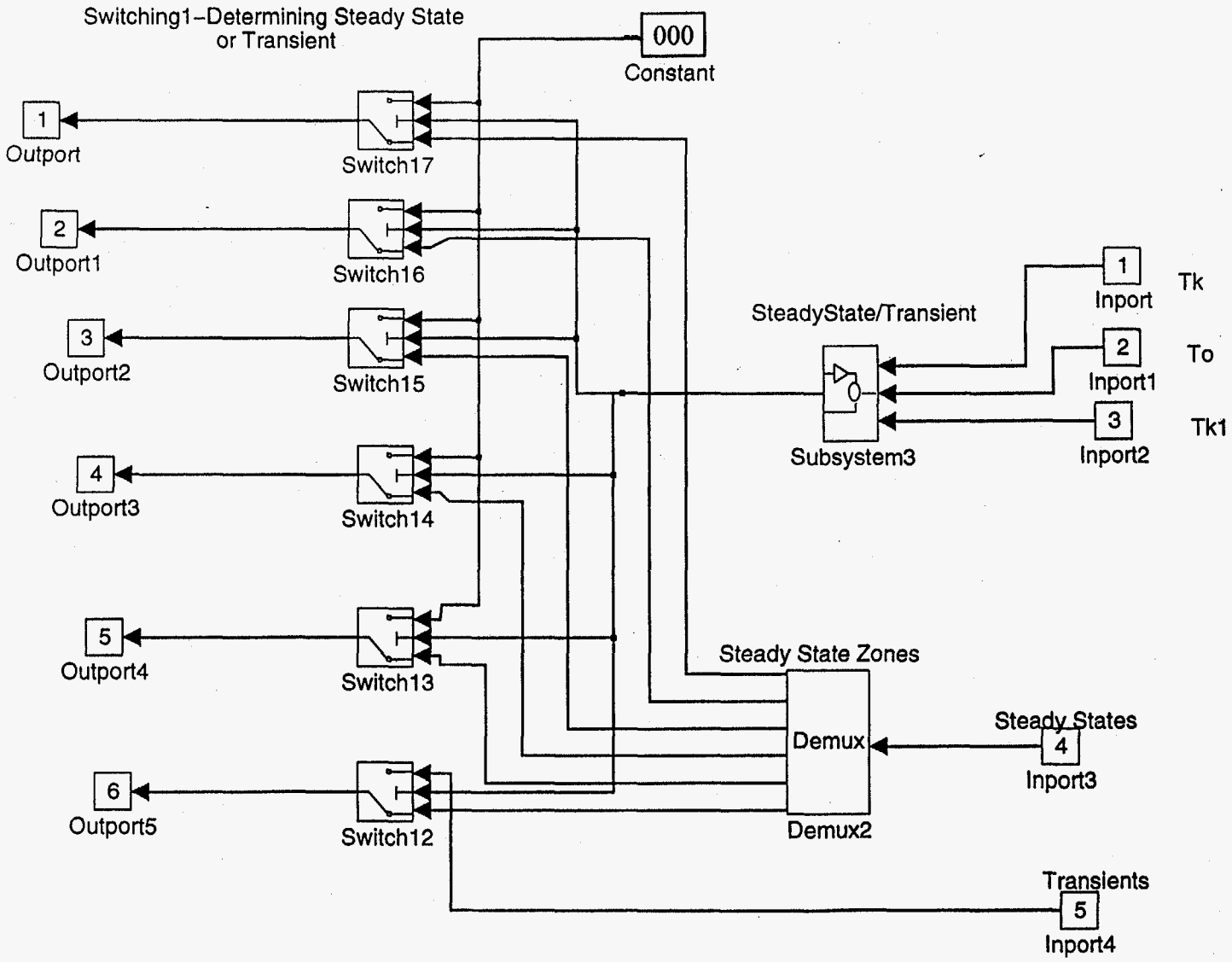


FIGURE III.16j– Steady States or Transient Condition

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 12 of 13)

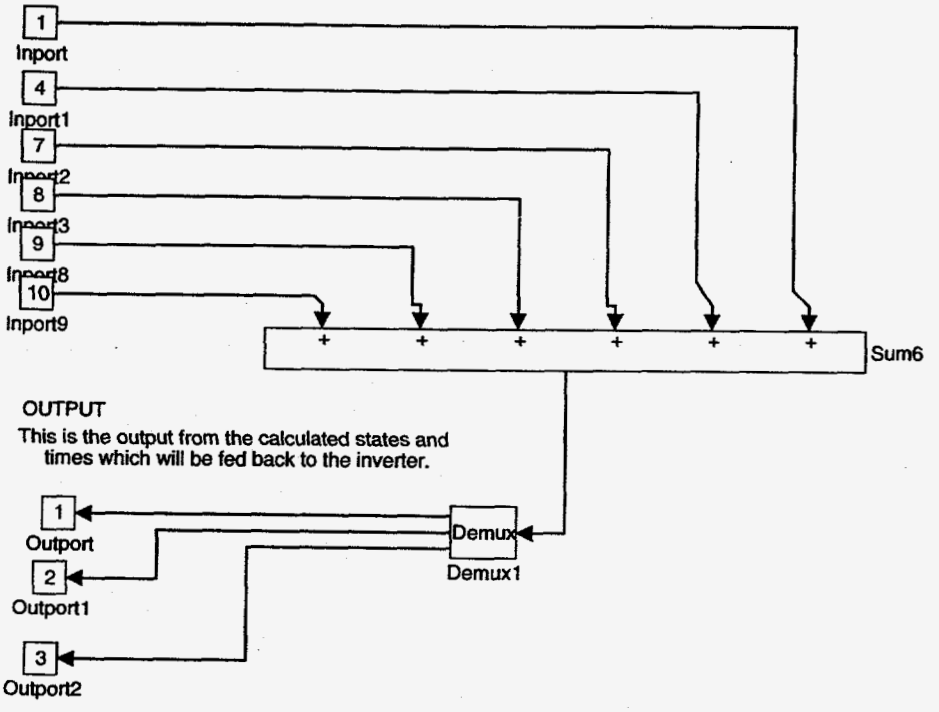


Figure III.16k– Output of phases a,b,c

Figure III.16 Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor (Sheet 13 of 13)

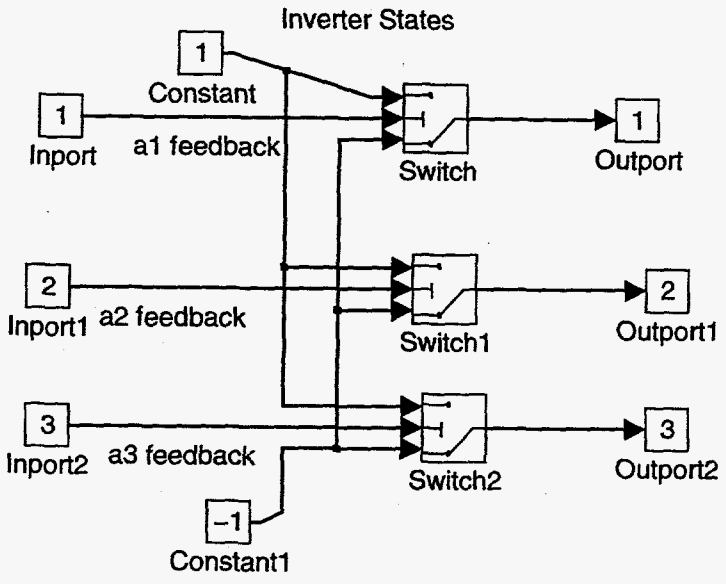


Figure III.16L– Determining Inverter States

IV. RESULTS OF SPACE VECTOR MODULATION SIMULATION RUNS

a. PARTITIONED MODEL:

Initially, the model was partitioned into smaller sections in order to test specific portions of the model prior to testing the entire model. Specifically, VD and VQ were replaced with two sinusoidal waves with a defined phase shift between the two. For the results shown below the phase shift between VD and VQ was 60 degrees. This provided a cyclic pattern and allowed us to evaluate both the steady state and transient conditions. In order to ensure a steady state condition (i.e., Tk and Tk1 must be positive values) the space vector angle " A" must be greater than 0 but less than approximately 1.

For the first 0.25 milliseconds the output is zero. This is based on the one cycle delay. For the next seven cycles (0.25-2.0 msec) a transient condition was in place. This was followed by three steady state cycles (2.0-2.75 msec). The remaining segment of the simulation (2.75-3.0 msec) resulted in a transient condition. The results of this simulation are documented in Figure III.18. Feedback to the inverter for this run (phases a, b and c) is documented in Table III.3. Each period (Ts/2) provided 25 samples or a time span of 0.25 msec.

The Space Vector Angle (SVA) for this example varied from 0-88 degrees. Therefore, when the steady state condition was applicable, states 0, 1, 2 and 7 were used (see Figure II.5). It should be noted that SVA was varied from 0-360 degrees during subsequent runs in order to ensure that all potential sequences were observed.

VD2 and VQ2 represent VD and VQ discussed above with a zero order hold (250 usec) applied to each parameter.

Tk,Tk1 and To represent the times calculated for the steady state condition. Note that for this particular run Tk1 was always positive. When Tk and/or To was negative, a transient condition resulted.

Comparing Tables II.1 and III.2 with the values of SVA (see Figure III.18, steady state only), KS (Torque Transient State), Fks (Flux Transient State) or the Torque and Flux Transient case, the results of the partitioned model are as expected.

Specifically the following is observed:

(1) For the steady state cases, States 0 and 7 are used. Each phase changes state once during the entire cycle and occurs one phase at a time.

(2) For the transient cases, States 0 and 7 are not used. For the torque and flux transient, one state remains in place for the entire 250 usec. While for the torque or flux transient case, two states are used.

Again with the exception of the torque and flux transient cases, phases are changed one at a time. For example, if the last valid state was 0 (000), and States 2 (110) and 3 (010) were to be used, then State 3 would be sent first followed by State 2. Likewise, if the last valid input was 7 (111), then State 2 would be sent to the inverter first, followed by State 3.

Figure III.17 SIMULINK Model of Direct Stator Flux Field Orientation Controller

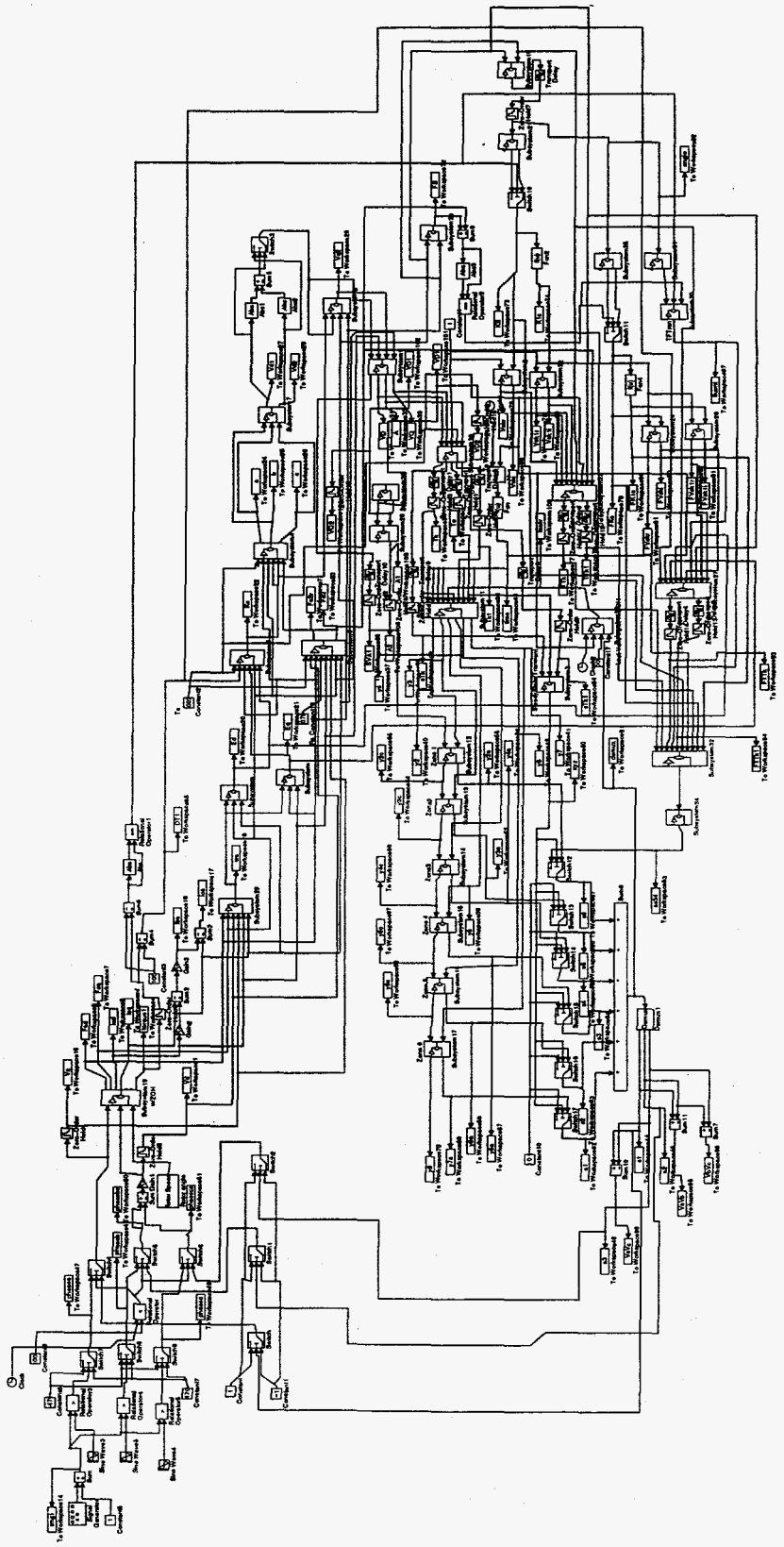
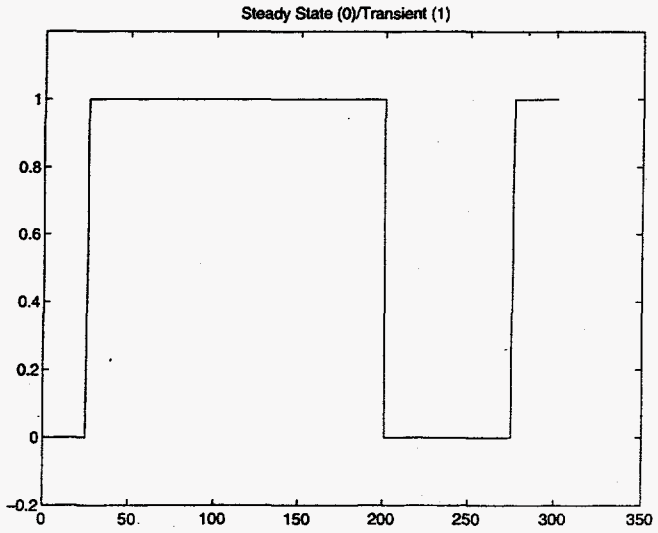
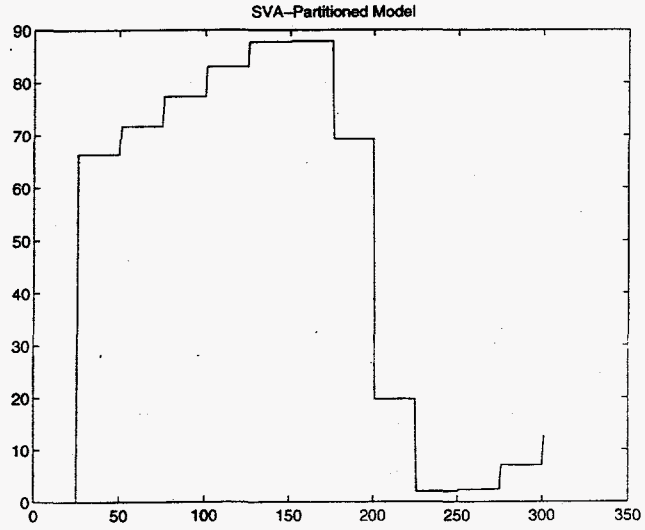


Figure III.18- Results from the Partitioned Model (Sheet 1 of 4)





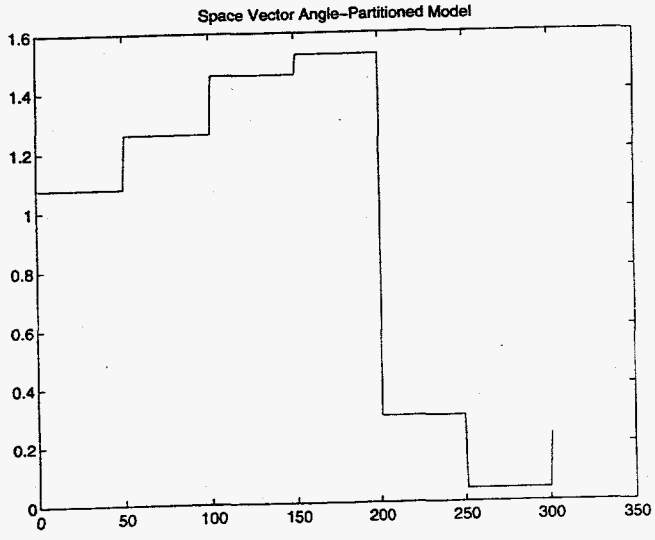
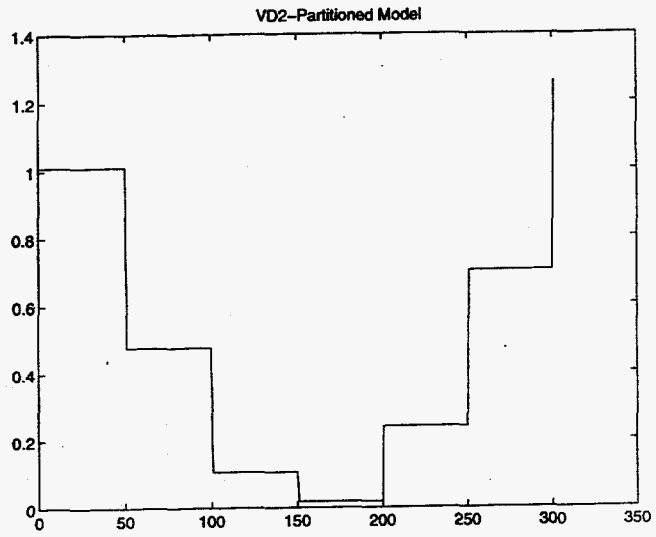


Figure III.18- Results from the Partitioned Model (Sheet 2 of 4)



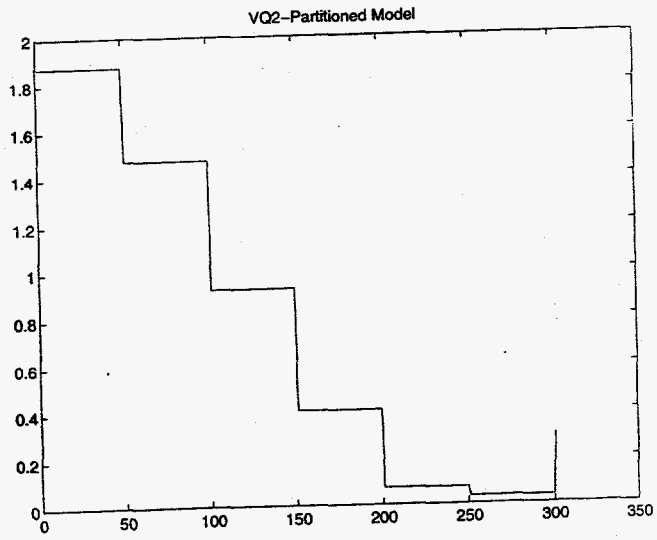


Figure III.18- Results from the Partitioned Model (Sheet 3 of 4)

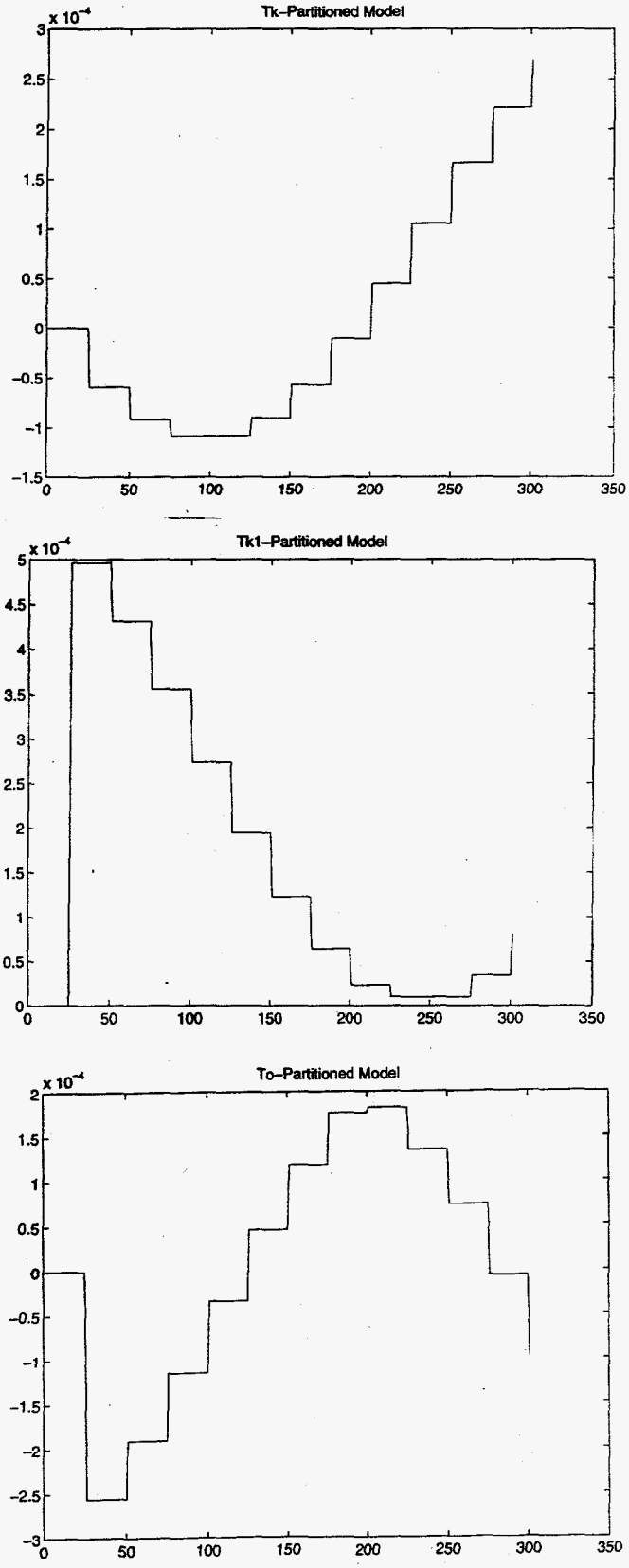
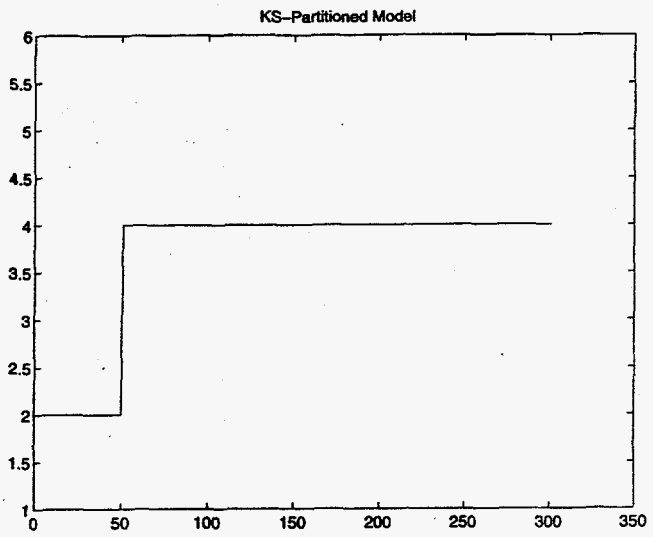
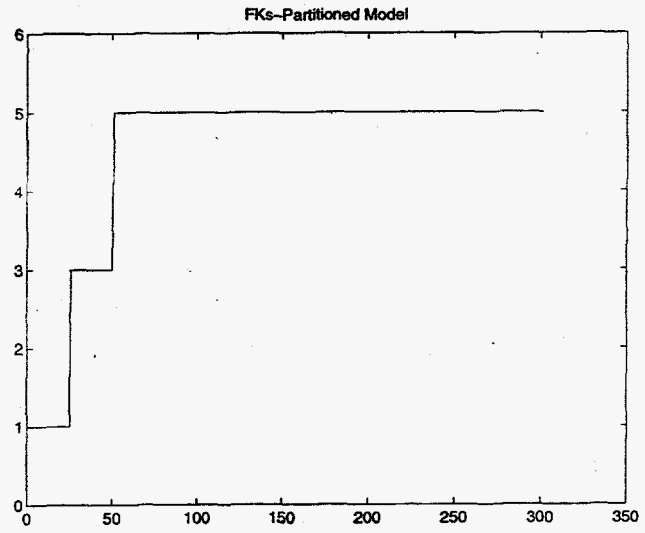


Figure III.18- Results from the Partitioned Model (Sheet 4 of 4)





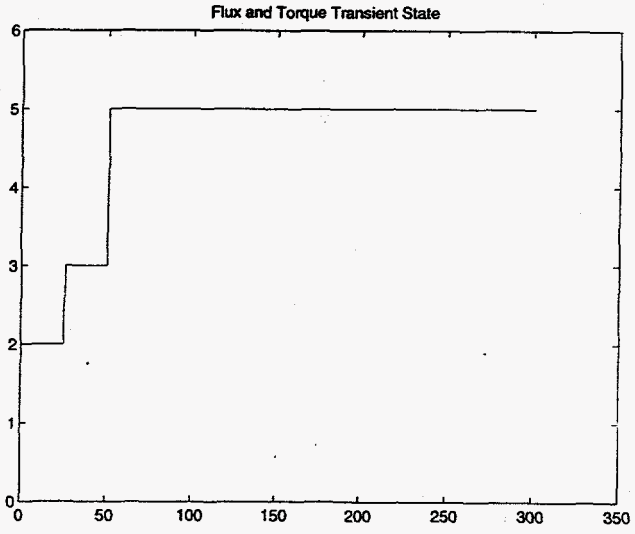


Table III.3-Three Phase Output to Inverter from Partitioned Model

Mode (Time is Simulation)	States(Output of Demux)	# of times
(0-0.25 msec)	000	x25
Flux Transient	010	x1
(0.25-0.5 msec)	011	x24
Torque & Flux Transient	001	x150 (6 Cycles)
(0.5-2.0 msec)		
Steady State	000	x10
(2.0-2.25 msec)	100	x4
	110	x2
	111	x9
Steady State	111	x7
(2.25-2.5 msec)	110	x1
	100	x11
	000	x6
Steady State	000	x4
(2.5-2.75 msec)	100	x17
	110	x1
	111	x3
Torque & Flux Transient	001	x25
(2.75-3.0 msec)		

Note: The Space Vector Angle (SVA) is only valid during the steady state condition, since Table III.1 is used for the transient cases.

After confirming that the algorithms required to determine both: (1) desired space vector states and (2) associated times for either the steady state or transient condition, were implemented correctly, the next step was to provide actual feedback to the inverter.

b. SPACE VECTOR MODULATION (SVM) MODEL

As an interim step, Model V was run with only the sinusoidal pulse width modulation scheme. Figure III.19 provides the torque response of the system for this run and is equivalent to the results shown in Figure III.15.

Model V was then modified to initially start the motor using the sinusoidal pulse width modulation for approximately 2.0×10^{-5} seconds. Once the motor was running in a steady state condition, the modulation scheme was switched to SVM as evidenced by the step change shown in Figure III.20.

However, once the system is switched to the SVM scheme, the torque value does not reflect the desired torque value and attempts to adjust the torque reference value were not successful. The system was in a locked condition. Discussions with SIMULINK's technical assistance personnel confirmed that based on the magnitude of algorithms contained in the model and the large number of internal loops, SIMULINK is unable to handle this condition and results in a locked state. Attempts to use zero order holds to effectively break the internal loops were unsuccessful. Although direct torque and flux control could not be demonstrated with this model, the actual control portion (sampling of stator current and voltage, calculating a space vector, determining steady state or transient state vectors, and observing the output states along with the state times) operated satisfactorily as was demonstrated with the partitioned model.

The next phase of this project (Chapter 4) focuses on the control portion of the model as described above. Given the results of the SIMULINK models, comparisons can easily be made with the results of the digital signal processor (dsp) and in fact several layers of complexity were added to the implementation of the dsp control system (i.e., implementation of interrupt handlers, optimization of code- which results in reducing cycle times, use of timers, etc.).

Figure III.19- Torque Response Using Sinusoidal PWM (Model V)

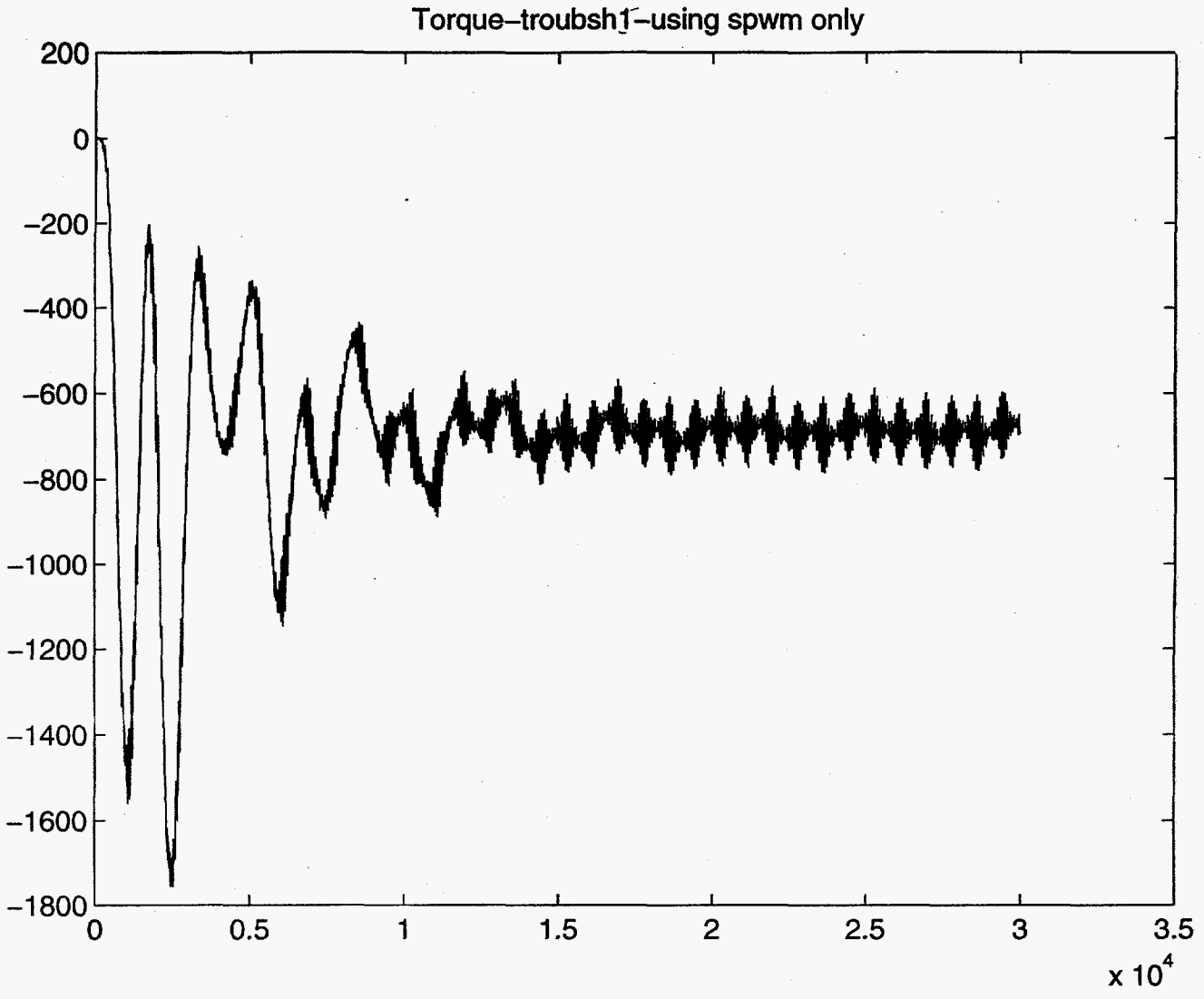
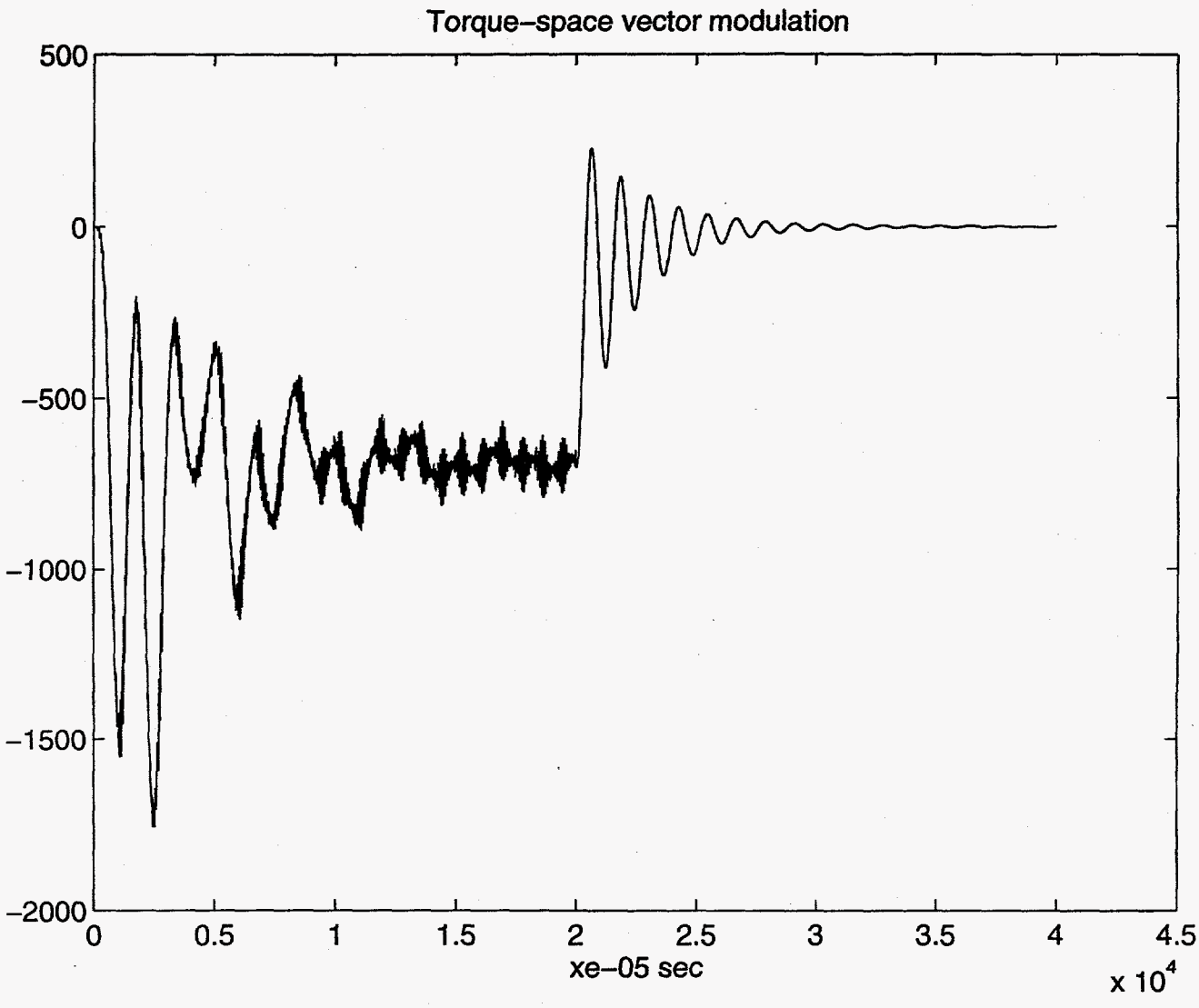


Figure III.20- Torque Response Using Sinusoidal PWM and SVM (Model V)



Chapter 4- ADSP (Analog Digital Signal Processor) 21020: Implementation of the SVM Technique

I. Introduction

The digital signal processor used for this project was the Analog Device (AD) Model 21020KG-133. Reference (18) provides a description of the processor, operating instructions, troubleshooting guidance, example programming, etc. The 21020 is a floating point processor. This processor has a 30 ns instruction cycle time or a 33.3 Mhz instruction rate. The ADSP package is a 223 lead ceramic pin grid array and is rated for ambient conditions in the range of 0C to 70C. This processor is mounted on an ADSP 21020 board known as the EZ-LAB evaluation board. EZ-LAB includes on board memory (RAM). Pushbuttons and indicator lights are connected directly to the processor.

The 21020 has two modes of operation. The first is an in-circuit emulator known as EZ-ICE. Although not used for this project, the EZ-ICE is a valuable tool since its function is to control EZ-LAB while on line. EZ-ICE allows stopping/starting a program in process, altering registers and memory, and conducting other debug operations. The EZ-ICE disconnects the path that would normally exist between the controller (ADSP 2111) and the ADSP 21020. EZ-LAB also contains stereo audio ports. However this function was not needed for the development of this project. The other mode, which was used in this project, controls the board from a PC connected via a RS-232 link. This mode has the

capability to download and upload programs, and reset and initiate execution, all of which is driven from the PC.

Based on the results of modeling this system in SIMULINK (see Chapter 3 for details), a program was written in ANSI C (Appendix C.1). The objective of this program was to calculate the appropriate vector states and associated times that are then fed back to the inverter to directly control torque and flux of an induction motor. This program was designed to mimic the SIMULINK block diagram documented in Chapter 3. The inputs are the simulated values of VD,VQ, Fds, and Fqs. For the actual application, the stator voltage (Va,Vb, Vc) would be sampled and fed through an A/D converter at the beginning of each switching period. From the phase voltage values the voltage vector (VD and VQ) are determined using a simple algorithm. The stator flux (Fds and Fqs) would then be calculated via hardware vice software. The assumption is that analog hardware would be used for the analog integration of stator flux since this is easily achieved using hardware and relieves the additional burden on the processor.

II. ADSP 21020 Architecture

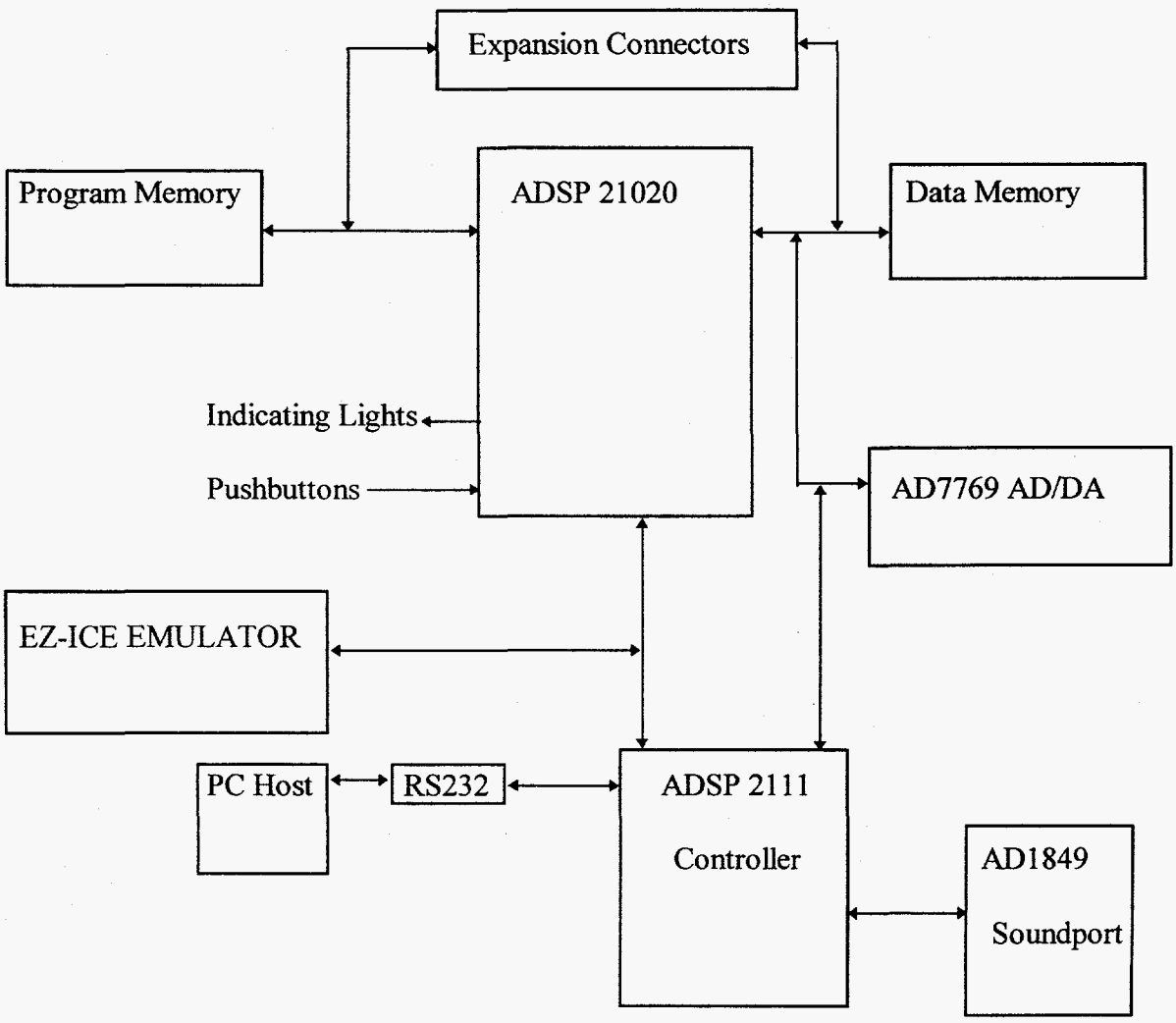
Figure IV.1 provides an overview of the EZ-LAB evaluation board and its components. The ADSP 2111 microcomputer controls the ADSP 21020 through the JTAG access port.

The functions of the ADSP 2111 are to:

(1) Transfer data between the 21020 and the AD1849 SOUNDPORT (not used in this project).

(2) Handle all host communication including downloading and uploading programs.

Figure IV.1- EZ-LAB Evaluation Board Block Diagram

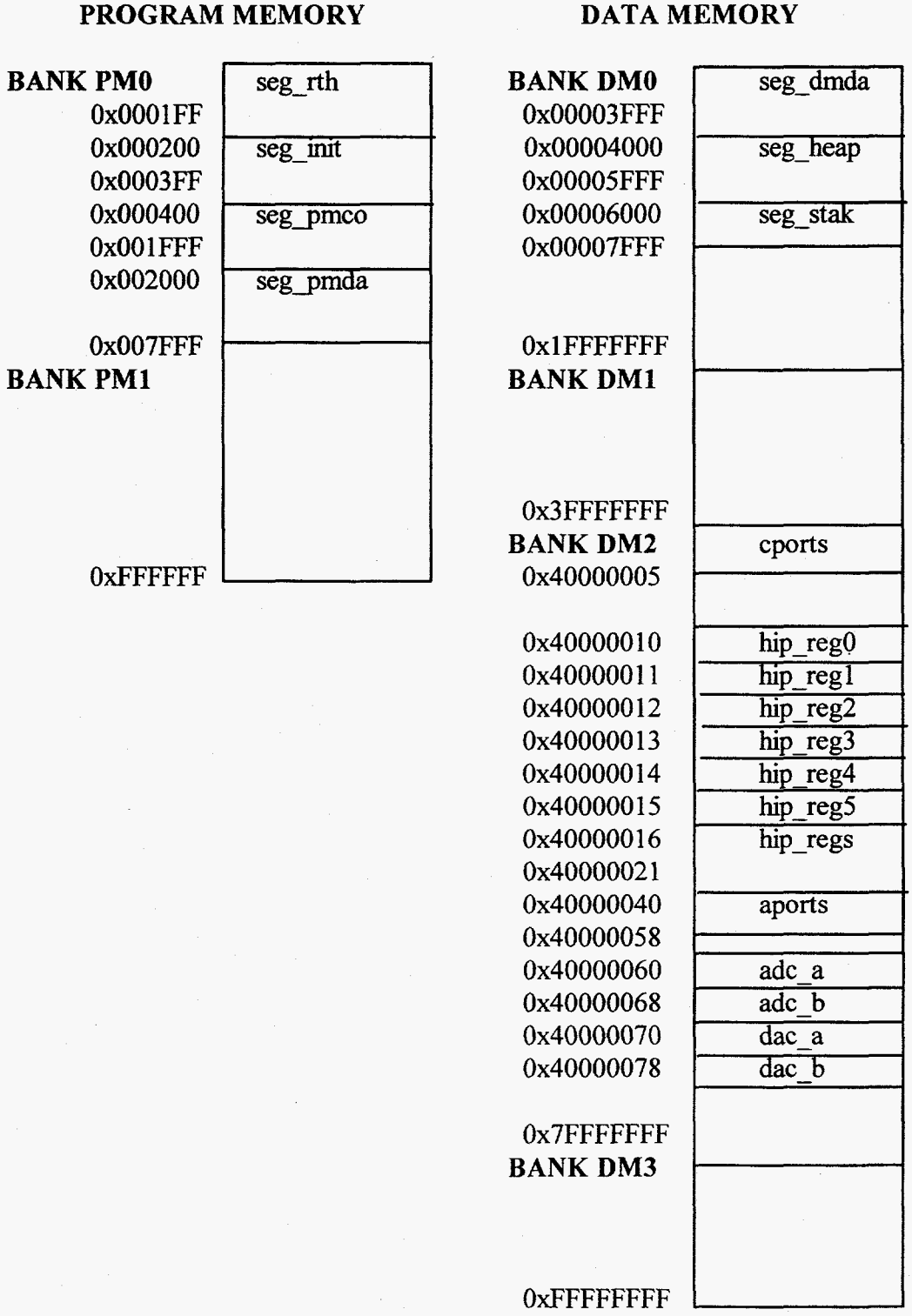


The AD7769 two channel AD/DA also interfaces directly with the 21020 through I/O ports. There are two ports for input and two for output.

The ADSP interfaces directly with both the program and data memory chips. The program and data memory space is divided into two separate regions. There are two banks for program memory and four banks for data memory. Figure IV.2 provides a visual representation of these memory locations as used for this project. The architecture file (file.ach) is responsible for allocating sections of memory for all functions required for use for a given program. Data can be stored in either program or data memory. The architecture file is used to map or specify the location of data. Data stored in the data memory bank(s) are designated as "dm". Likewise, data stored in the program memory bank(s) are designated as "pm".

The architecture file (jb7.ach) developed for this project was based on the EZ-LAB evaluation board, since ultimately the program is downloaded and runs the EZ-LAB evaluation board. The jb7.ach file is located in Appendix C.2. The corresponding map file is documented in Appendix C.3.

Figure IV.2- Program and Data Memory Maps



III. OPERATION of EZ-LAB and PC

To run the ADSP 21020, the following actions are required:

(1) Use of the command "lab21k" connects the EZ-LAB to the PC host. The interface program starts communicating with the ADSP 2111, assuming power is available to the EZ-LAB.

(2) The program displays a menu of program options on the PC monitor. These items include:

a. Download to ADSP 21020 memory. This loads a specific program from the PC to program and data memory on the EZ-LAB board. Downloaded programs are in byte stacked format, known as file.stk.

b. Reset and Run the ADSP 21020. This resets the processor which then allows the program to commence execution.

c. Upload from the ADSP 21020 memory. This selection results in stopping the ADSP 21020 and loading a program from either the program or data memory on the EZ-LAB board to the PC.

d. Information related to the software version is displayed.

e. The final choice from this menu is a return to DOS which leaves the program and returns to the normal operating system.

Various commands and associated switches were used to accomplish tasks such as compiling the source code, assembling the code etc. Table IV.1 provides a summary of the commands used for the development of this program.

Table IV.1- Commands for the 21020 and Associated EZ-LAB /PC Interface

COMMAND	FUNCTION	REMARKS
g21k	C Compiler for 21020	various switches listed below provide additional functions
-g (switch)	produces debuggable code for CBUG	
-map (switch)	generates map file	See Appendix C.3
-save-temps (switch)	stores the usual "temporary" intermediate files	saves compile, assemble, object code files
-0 or -02 (switch)	Optimize code; 0:reduce code size & execution time 02:increase in both compile timing and the performance of generated code.	This switch is not used during CBUG operations. Observed negligible results for this project with this switch, so did not use.
-lsamp (switch)	Searches the library named samp when linking	Used with CBUG operations
sim21k	Invokes CBUG	

COMMAND	FUNCTION	REMARKS
spl21k	Uses a PROM splitter to create byte-stacked files (file.stk)	Needed to create this file in order to download to EZ-LAB board. Order of listing switches in command line is very important. Example in manual did not work.
lab21k	Interface program between PC and EZ-LAB	Five options are displayed as discussed above in Section III.

EXAMPLES:

-To compile source code txt.c for CBUG : *g21k txt.c -a txt.ach -o txt.exe -g -lsamp*

-To compile source code txt.c for download to EZ-LAB: *g21k txt.c -a txt.ach -o txt.exe*

-To invoke CBUG: *sim21k -a txt.ach -e txt.exe*

-To create txt.stk: *spl21k -pm -dm -ram -f B -a txt.ach txt.exe*

-To interface with EZ-LAB from PC: *lab21k*

IV. DEVELOPMENT OF THE PROGRAM

This program, written in ANSI C, was designed with simulated inputs (VD, VQ, Fds, and Fqs). Output phases a, b and c, were presented using the status of indicating lights located on the EZ-LAB. VD and VQ represent the quadrants of the stator voltage vector "Vs" where VQ lags VD by angle θ. The stator flux (Fds and Fqs) in reality would be calculated using hardware vice software. The program was run with all calculated states and associated times being stored in an array entitled "RESULT". In order to be able to visually observe the changing status of lights, the switching time was changed from 250 micro-seconds to 9 seconds. Obviously for the real application where feedback is fed to an inverter and not to status lights, the cycle time would be approximately 250 micro seconds. Twice each switching period the input is processed through the various functions of the program in order to determine if the condition represents:

(1) Steady state [STEAD_STATE] -where four states and four associated times are calculated and used in the following cycle.

(2) Torque transient [TORQUE_TRAN] -where two states and two associated times are calculated and used in the following cycle.

(3) Flux transient [FLUX_TRAN] -where two states and two associated times are calculated and used in the following cycle.

(4) Torque and Flux transient [TORQUE_FLUX_TRAN] - where a single state is determined and the full cycle is spent in this state.

Each row in the array entitled "RESULT" has nine columns. The ninth column (not shown below) represents the condition (steady state,torque transient, etc.) that exists for that case. Specifically, "1" represents steady state , "2" represents a torque transient, "3" represents a flux transient, and "4" represents a torque and flux transient.

For the remaining columns, the following is true:

RESULT	1-TIME	2-STATE	3-TIME	4-STATE	5-TIME*	6-STATE*	7-TIME*	8-STATE
STEAD_STATE1	T01	STI	Tk	ST1	Tk1	ST2	T01	STF
STEAD-STATE2	T01	STI	Tk1	ST2	Tk	ST1	T01	STF
T_TRAN1	Ttkt	KT	Ttk1t	KT1	0	1	0	STF
T_TRAN2	Ttk1t	KT1	Ttkt	KT	0	1	0	STF
F_TRAN1	Tkf	FKF	Tk1f	FKF1	1	0	1	STF
F_TRAN2	Tk1f	FKF1	Tkf	FKF	1	0	1	STF
TF_TRAN	9	KTF	0	1	0	1	0	STF

T_TRAN= Torque transient case (1 or 2 is contingent on the previous last valid state (STF))

F_TRAN= Flux transient case (1 or 2 is contingent on the previous last valid state (STF))

TF_TRAN= Torque and Flux transient case. Since the entire cycle time is spent in the one state KTF, the full cycle time (9 seconds) is specified.

* For torque, flux and torque_flux transient cases, default values are placed in columns only to be used as an identifier or place holder.

Initially, once the RESULT array was full, a separate function stepped through the array using the timer and interrupt handlers to determine when to change states. For demonstration purposes, the LEDs on the board were used as: FLAG1 represents phase a, FLAG2 represents phase b, and FLAG3 represents phase c feeding back to the inverter. When the LED is ON for a given phase, the feedback to the inverter for that phase would be "1" else if the LED is OFF, feedback to the inverter would be "0".

Once this was successfully demonstrated, the program was revised to reflect the desired control scheme. Specifically, the program must calculate the new state and associated state times in parallel with providing an output to the inverter. As was done with the original program, the output is represented by the three LEDs changing state to reflect the three phases being fed back to the inverter. For the actual application, the EZ-LAB board outputs would be provided to the inverter, switching phases (High "1" or Low "0") appropriately to provide the direct control of both torque and flux. This switching would

be at a much higher frequency (in the range of 4000 Hz). The output is based on the previous cycle's calculations. Again the use of interrupts and timing options was utilized to accomplish this task. Appendix C.1 provides the documentation of the program with associated comments. For the steady state condition, the status of phases a, b and c change four times within the allotted time (9 seconds). In order to accomplish this the following interrupts are used:

(1) timer_isr: This interrupt handler is used for all possible scenarios (steady state, torque transient, etc.). For the steady state case, the counter is set using the interrupt handler routine to count from the value of time T01 (T0/2) to zero seconds. Once the timer_isr routine is complete and prior to the timer reaching zero, the program returns to the main "for" loop and the calculations begin using inputs of the next sample. However, it should be noted, that at time 0 seconds of the cycle, the first thing to occur is an initial output which is provided to the inverter representing values calculated at -Ts seconds or in this case -9 seconds. Therefore the actual sampling time for the stator voltage and current will occur at time t1, where t1 represents the amount of time it takes to service the timer_isr interrupt and return to the main program where the stator voltage and current are sampled. When the timer reaches zero, the program services the timer1_isr interrupt routine.

The above is also true for the other possible conditions with the following exceptions: For the torque transient case and the flux transient case the counter is set for Ttkt (Ttk1t) or Tkf (Tk1f), respectively, vice T01. For the torque and flux transient case:

(a) the timer is set for the full cycle or 9 seconds and

(b) the timer_isr interrupt is the only interrupt handler routine used during this cycle.

(2) timer1_isr: This interrupt handler is only used for the steady state condition, the torque transient condition or the flux transient condition. For the steady state case, the counter is set using the interrupt handler routine to count from the value of time Tk(Tk1) to zero seconds. Once the timer1_isr routine is complete and prior to the timer reaching zero, the program returns to the main "for" loop and if all the calculations have not been completed, the program will continue where it exited to handle the timer1_isr routine. If all the required calculations have been completed and stored in the array "RESULT" , the program will remain in the "idle" condition and wait for the timer to reach 0 seconds. The above is also true for the torque transient case or the flux transient case with the exception that the counter is set for Ttk1t (Tkt) or Tk1f (Tkf), respectively , vice Tk (Tk1).

Once the timer1_isr reaches 0 seconds, the interrupt handler routine timer2_isr will be serviced if the condition is steady state. If the condition was either the torque transient or flux transient case, the interrupt handler timer_isr will be serviced which marks the beginning of the next cycle.

(3) timer2_isr: This interrupt handler is only used for the steady state condition. The counter is set using the interrupt handler routine to count from the value of time Tk1(Tk) to zero seconds. Once the timer2_isr routine is complete and prior to the timer reaching zero, the program returns to the main "for" loop and if all the calculations have not been completed, the program will continue where it exited to handle the timer2_isr routine. If all the required calculations have been completed and stored in the array "RESULT" , the program will remain in the "idle" condition and wait for the timer to reach 0 seconds. Once the timer2_isr reaches 0 seconds, the interrupt handler routine timer3_isr is serviced.

(4) timer3_isr: This interrupt handler is only used for the steady state condition. The counter is set using the interrupt handler routine to count from the value of time T01(T0/2) to zero seconds. Once the timer3_isr routine is complete and prior to the timer reaching zero, the program returns to the main "for" loop and if all the calculations have not been completed, the program will continue where it exited to handle the timer3_isr routine. If all the required calculations have been completed and stored in the array "RESULT" , the program will remain in the "idle" condition and wait for the timer to reach 0 seconds. Once the timer3_isr reaches 0 seconds, the interrupt handler routine timer_isr is serviced. This marks the beginning of the next cycle.

The timer command used in the interrupt handler routine is timer_set (TPERIOD,TCOUNT). TPERIOD specifies the frequency of the timer interrupts. The number of cycles between timing interrupts is TPERIOD +1 or a maximum value of 2^{32} -1 (4294967295 * 30 nsec=128.8 seconds). TCOUNT is the register containing the timer counter. The timer decrements the TCOUNT register each clock cycle. When TCOUNT reaches zero, the interrupt is generated.

On the next clock cycle after TCOUNT reaches zero the timer automatically reloads the TCOUNT register from the TPERIOD register.

In addition to viewing the lights on the EZ-LAB board, the results of the program can be uploaded from the board to the PC. The values found in the array RESULT can then be compared to that observed visually on the board. In order to be able to run this program indefinitely, once the array "RESULT" is filled, the program returns to Row 0 and begins writing over previous values that are no longer needed.

V. Troubleshooting Aid-CBUG

The CBUG emulator allows the operator to troubleshoot the compiled program in C. This tool allows you to step through a program line by line and view the expected parameter values. CBUG uses the executable file for debugging. CBUG also supports debugging of the C-source interrupt handlers. In order to do this a breakpoint at the interrupt handler is inserted. The command "CONTINUE" is used and run until the program reaches the interrupt handler routine. Breakpoints are used throughout the program in order to allow the program to run through several lines of code without interruption.

VI. Conclusion

For those with any language programming experience, a review of Appendix C.1 will quickly reveal that this code was not in any manner optimized. Execution time of the full program (single loop time for worse case torque and flux transient condition) took approximately 100 microseconds which is less that the 250 microseconds which would be used for this project's direct torque and flux control application. Therefore this program, as written, is satisfactory for use.

The most difficult part of the program was to get the interrupts and associated clock cycles working satisfactorily. Overall the ADSP 21020 is very easy to use, and very forgiving for the inexperienced operator. A great deal of knowledge about the functions of the program and how they are executed can be gained by using the CBUG troubleshooting tool.

The next step in the development of this project would be to use sampled filtered inputs from am induction motor to the digital signal processor. The output would then be fed back to the inverter controlling the induction motor. Reference torque and flux values would be altered to verify the system responds to the desired inputs appropriately. This is needed to validate the assumptions that these algorithms will be able to control torque and flux directly for any given torque or flux reference value.

REFERENCES

(1) Direct Torque Control of Induction Machines Using Space Vector Modulation - 1991 IEEE Habetier, Profumo, Tolbert and Pastorelli

(2) ECE 711- Dynamics and Control of AC Drives: TA Lipo,DW Novotny, Univ. of Wisconsin 1992

(3) Handbook of Operational Amplifier Circuit Design - David Stout/Milton Kaufman

(4) Direct Torque Control of Induction Machines Over a Wide Speed Range-1/92 IEEE Habetier,Profumo and Pastorelli

(5) Space Vector Modulation -An Engineering Review: PG Handley, JT Boys, Univ of Aukland, New Zealand

(6) Space Vector Modulation and Field Oriented Control Using Transputers and Signal Processing for Speed Control of Induction Motors: Webster, Levy, Diana and Harley 1989 IEEE

(7) Induction Machine Field Orientation Along Airgap and Stator Flux: Erdman, Hoft IEEE 3/90

(8) Evaluation of a High Performance Induction Motor Drive Using Direct Torque Control: Abe, Habetler, Profumo, Griva IEEE 9/93

REFERENCES (CONT)

(9) Power Electronic Control of AC Motors- JMD Murphy & FG Turnbull

(10) Power Electronics-Circuits, Devices and Applications -2nd Edition- Muhammad Rashid

(11) The General Relationship between Regular- Sampled Pulse Width Modulation and Space Vector Modulation for Hard Switched Converters: Donald Holmes 1/92 IEEE

(12) A High Performance Induction Motor Drive System Based on a Non-Linear Resonant Pole Soft Switching Inverter- Burgers, Van Wyk, Case IEEE 3/1992

(13) Microprocessor Control of Motor Drives and Power Converters IEEE 3 rd- Tutorial Course

(14) Electrical Machines and Drives- A Space Vector Theory Approach- Peter Vas

(15) Analysis and Realization of PWM Based on Voltage Space Vectors-IEEE 1986- HW VanderBoeck

(16) Analysis of Electric Machinery- Paul Krause 1986

(17) Power Electronics and AC Drives- BK Bose 1986

REFERENCES (CONT)

(18) ADSP-21020- Analog Devices Manuals (1993)

- User's Manual

- Applications Handbook Volume I

- Assembler Tools & Simulator Manual

- C Runtime Library Manual

- C Tools Manual

- IXLibs-21K User's Manual

- EZ-LAB Evaluation Board Manual

APPENDIX A.1- Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor

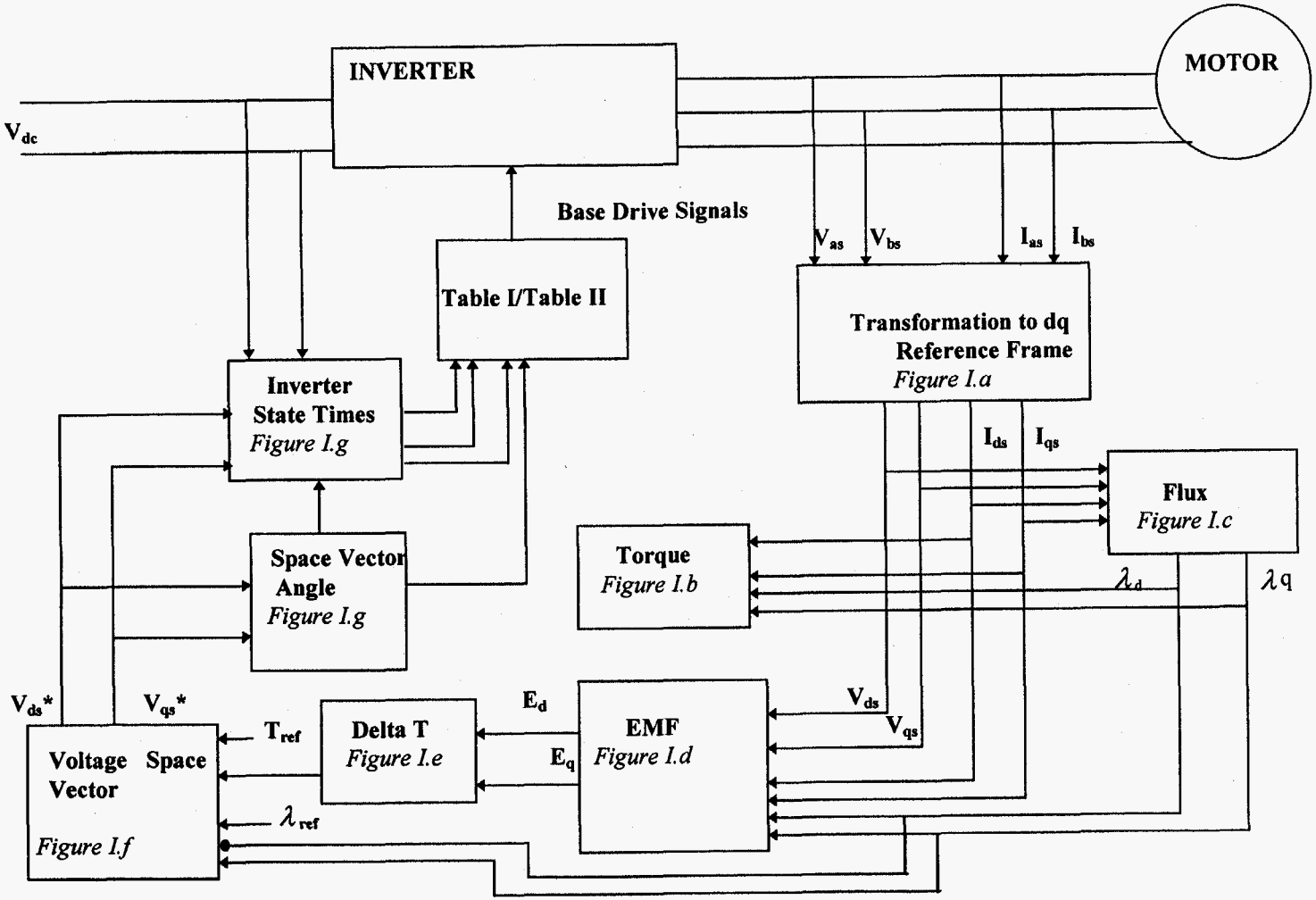


FIGURE I- **Block Diagram of Direct Stator Flux Field Orientation Controlled Induction Motor**

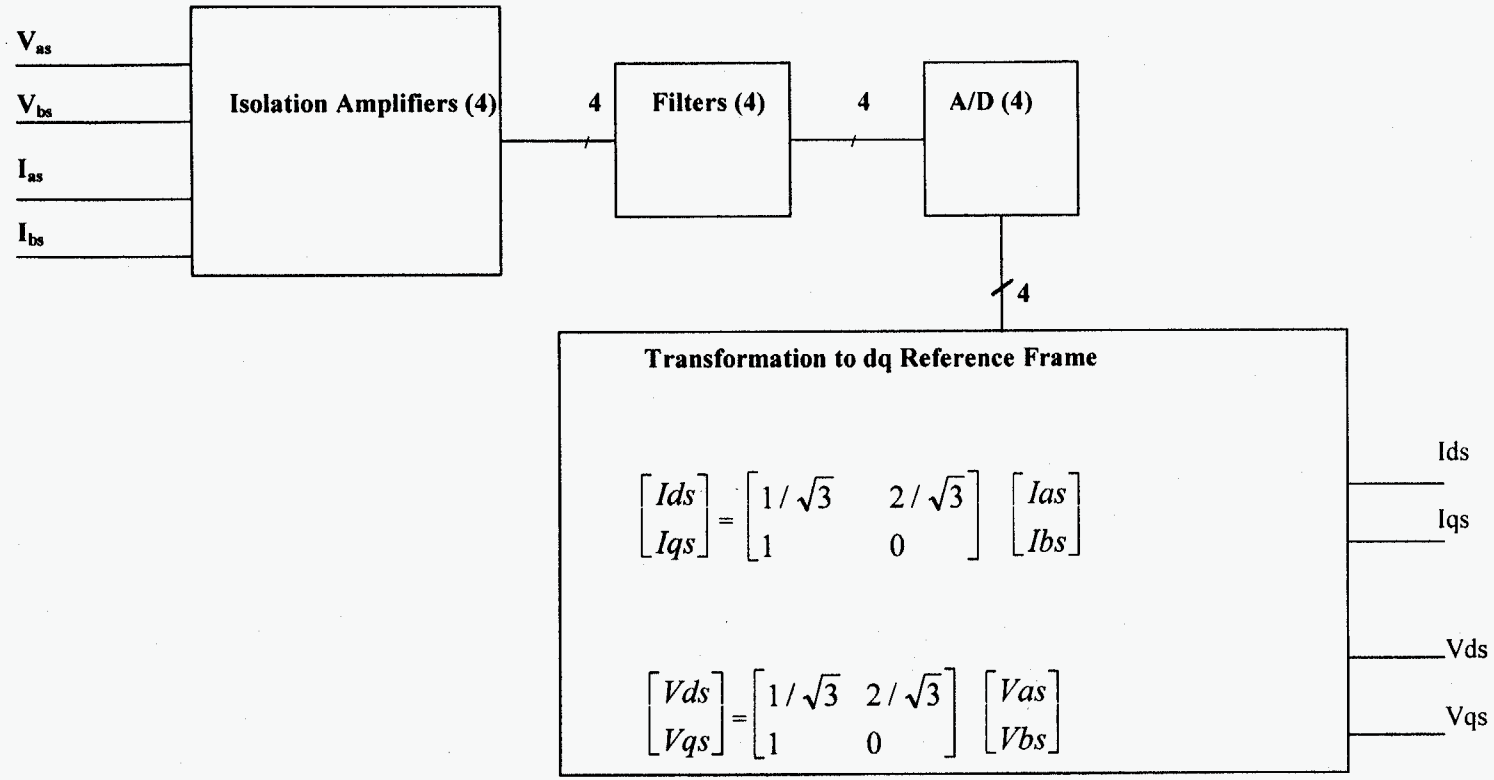


Figure I.a- **Transformation of System Quantities to a Stationary dq Reference Frame**

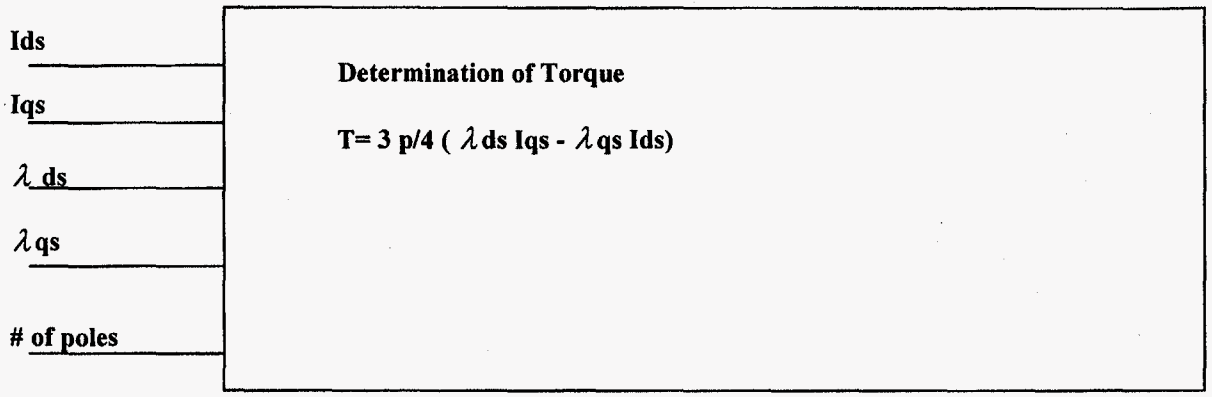


Figure I.b- **Determination of Torque**

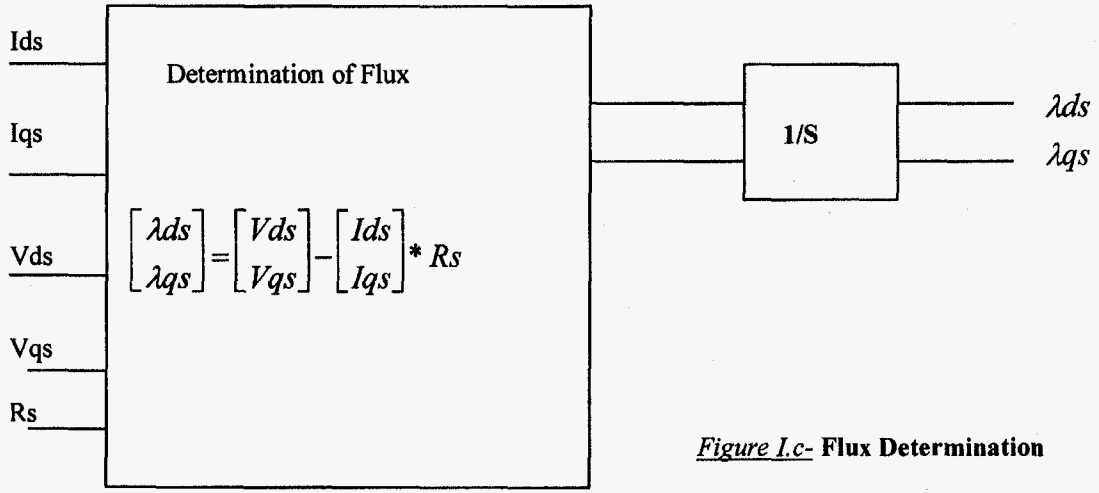


Figure I.c- **Flux Determination**

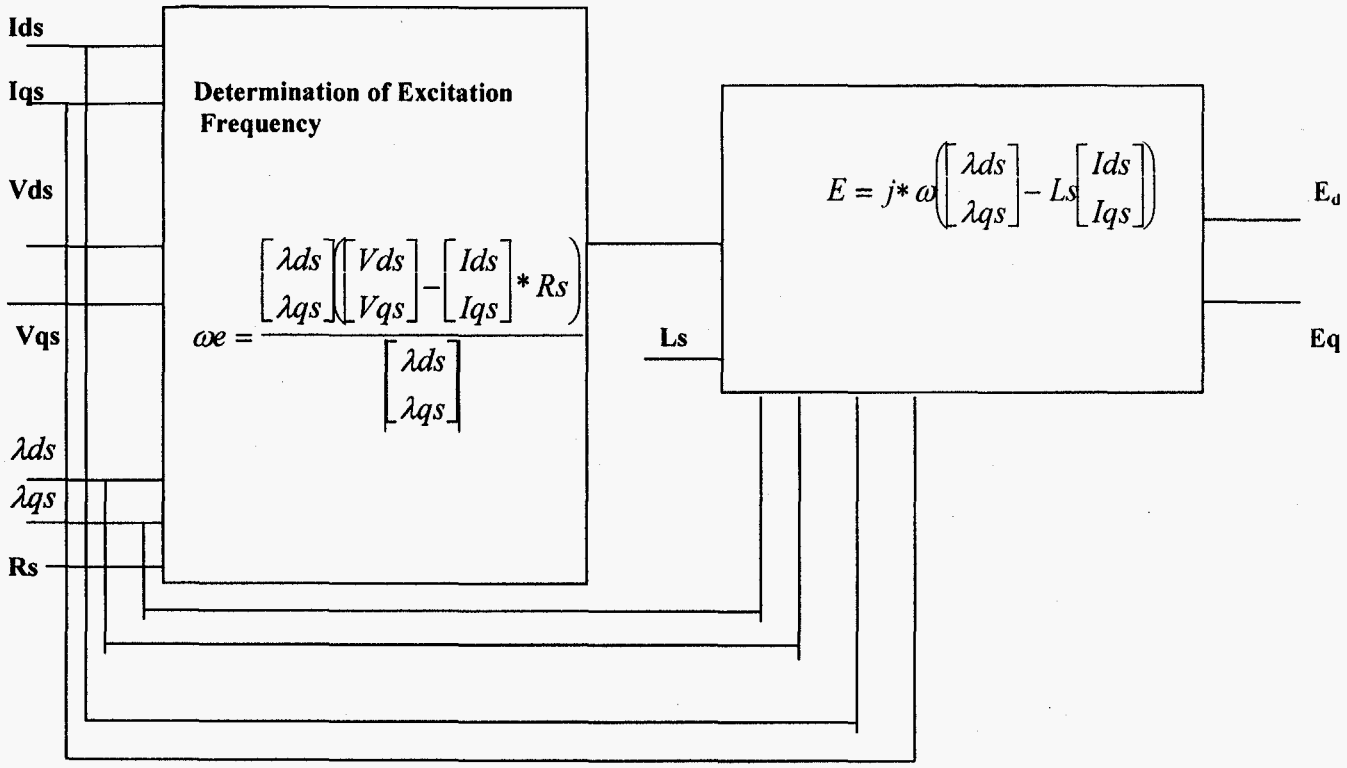


Figure I.d- **Determination of EMF**

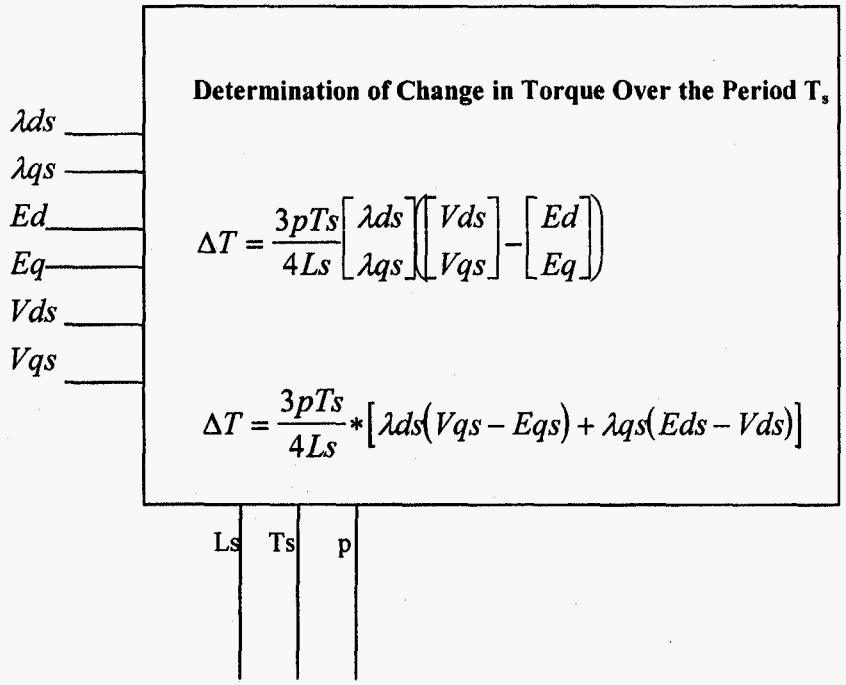


Figure I.e- **Determination of Change in Torque**

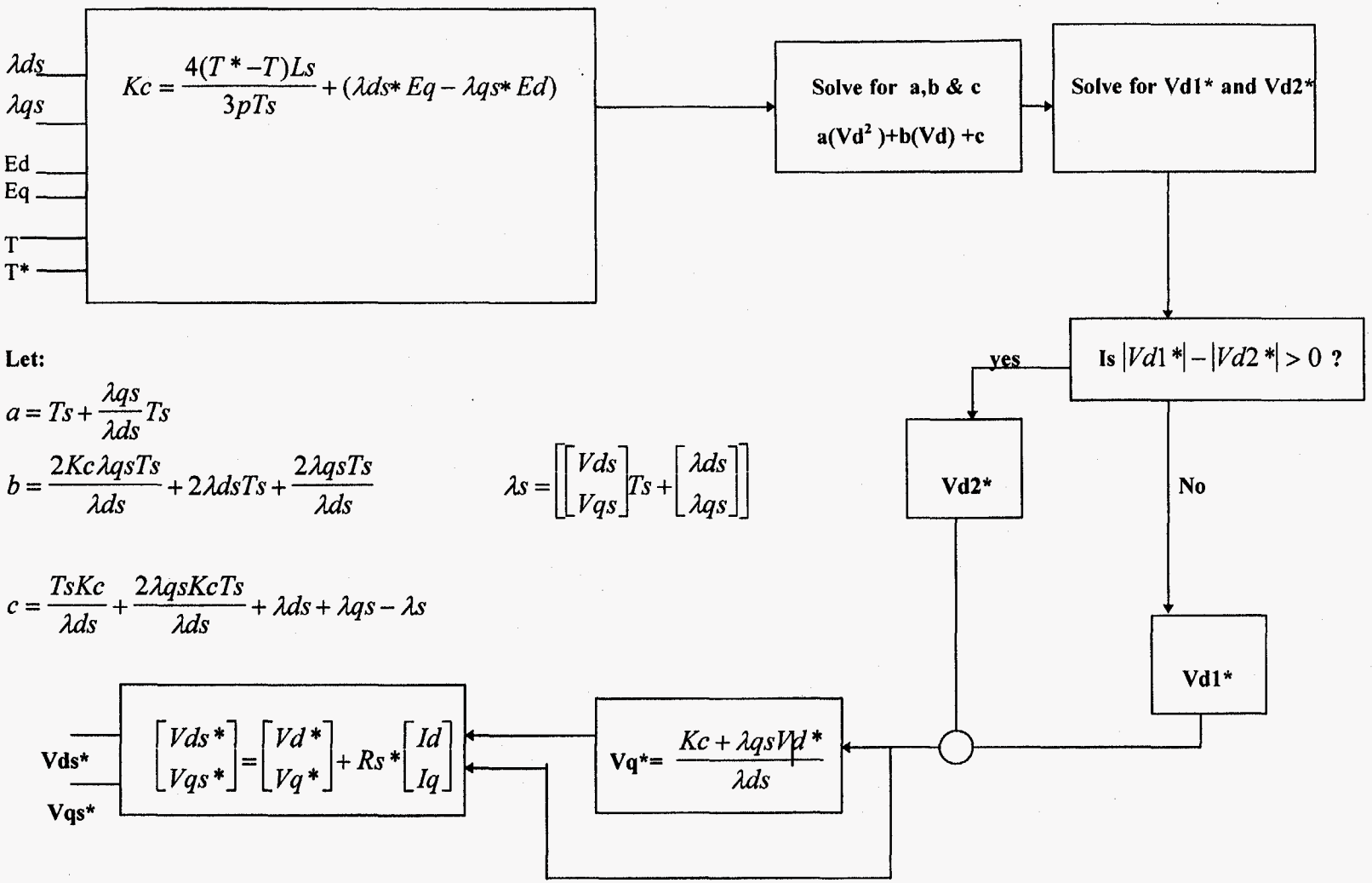


Figure I.f- **Determination of Space Vector**

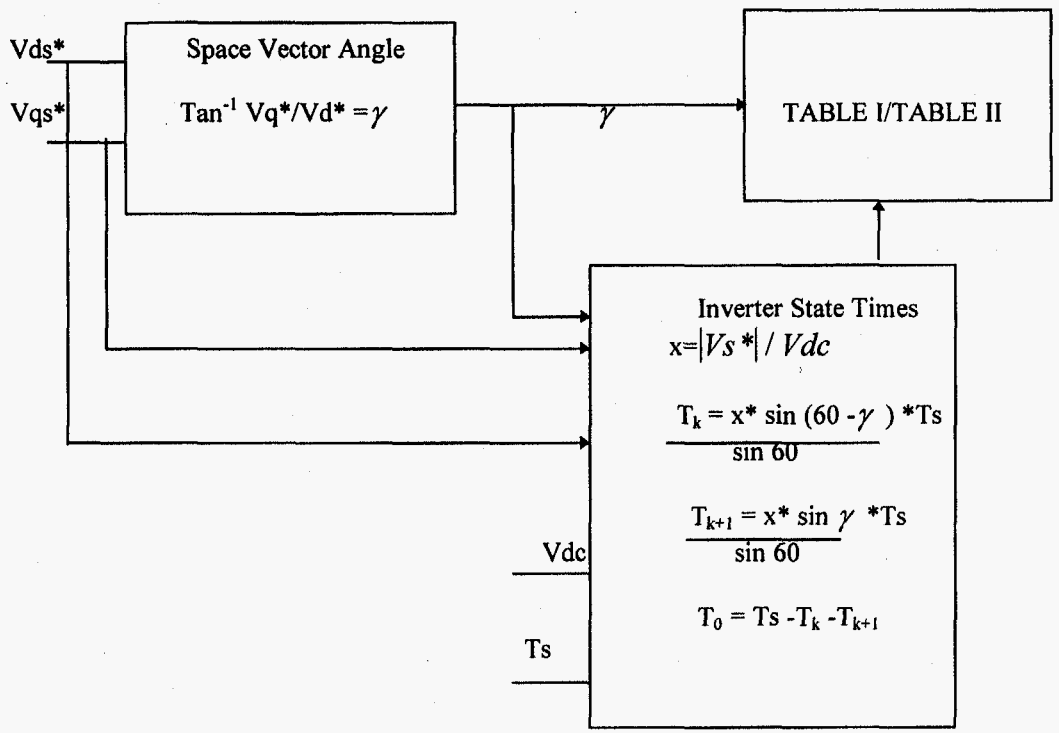


Figure I.g- **Determination of Inverter State Times**

TABLE I- STEADY STATE VALUES (T_k, T_{k+1} AND T_0>0)

γ-SPACE VECTOR ANGLE	INITIAL STATE	K STATE	K+1 STATE	FINAL STATE
$0 < \gamma < 60$	0[000]	1[100]	2[110]	7[111]
$60 < \gamma < 120$	7[111]	2[110]	3[010]	0[000]
$120 < \gamma < 180$	0[000]	3[010]	4[011]	7[111]
$180 < \gamma < 240$	7[111]	4[011]	5[001]	0[000]
$240 < \gamma < 300$	0[000]	5[001]	6[101]	7[111]
$300 < \gamma < 360$	7[111]	6[101]	1[100]	0[000]

TABLE II- TRANSIENTS (T_k, T_{k+1} AND/OR $T_0 < 0$)

$(2N-3)PI/6 < \text{FLUX ANGLE}(\lambda) < (2N-1)PI/6$

$\lambda = TAN^{-1} (\lambda QS/\lambda DS)$

TABLE II.A-TORQUE TRANSIENT

SIGN OF (T-T*)	K	K+1
NEGATIVE	N+1	N+2
POSITIVE	N+4	N+5

TABLE II.B- FLUX TRANSIENT

SIGN OF ($\lvert\lambda\rvert$ - λ*)	K	K+1
NEGATIVE	N	N+1
POSITIVE	N+2	N+3

TABLE II.C- TORQUE AND FLUX TRANSIENT

SIGN OF (T-T*)	SIGN OF ($\lvert\lambda\rvert$ - λ*)	K
NEGATIVE	NEGATIVE	N+1
NEGATIVE	POSITIVE	N+2
POSITIVE	NEGATIVE	N+4
POSITIVE	POSITIVE	N+5

APPENDIX A.2- Matlab- Algorithms for Direct Torque Control of Induction Motor

```
%JB Seiz Thesis-DSP Controlled Induction Motor
%The name of this file is invertbu.m
%update of this file last occurred 08/28/95

%Reference (1) Direct Torque Control of Induction
%Machines
%Using Space Vector Modulation dated 05/91


      %Initial Conditions
ts=clock;
Tref=100        %Reference or Command value
                %of Torque
Ts=250e-06;     %Switching frequency
p=4;            % Number of poles
Vr=[0 0 0];     %Resulting Space Vector State

                %for each switching period
t=0:0.5:4*pi;
Va=10; %sin(t);
Vb=20; %sin(t-2*pi/3);
Vc=30; %sin(t-4*pi/3);
Fm=60*1.01036;    %Rotor speed, slip can be
                  %varied.
Rsm=.007565;      %Stator magnetizing
                  %resistance
Ia=Va/Rsm;
Ib=Vb/Rsm;
Ic=Vc/Rsm;
Rrm=.00596;     %Rotor magnetizing resistance
Lhm=4.559e-03     %Magnetizing inductance of
                  %motor
IsIm=.1187e-03;   %Stator/magnetizing current
LrIm=.1692e-03;
ppairs=3;
Lsm=Lhm+IsIm;      %Stator Inductance
Lrm=Lhm+LrIm;      %Rotor Inductance
Det=Lrm*Lsm-(Lhm)^2;
MI1=Lrm/Det;
MI2=Lsm/Det;
MI3=-Lhm/Det;
RSMI1=-Rsm*MI1;
RSMI2=-Rsm*MI2;
RSMI3=-Rsm*MI3;
Mm=3*(ppairs/2);
RRMI3=-Rrm*MI3;
RRMI2=-Rrm*MI2;
```

```
RRMI1=-Rrm*MI1;
Vdc=120;        %Input Voltage to inverter



%Output of inverter
%Determining dq reference values for stator
%voltage, from EQN (1) of Ref (1)

Vds=Vb-Vc/((3)^0.5)
pause
Vqs=Va
pause



 %A,B,C,D values provided from
 %geb15.m program
 %A=[RSMI1 0 RSMI3 0;0 RSMI1 0 RSMI3;
 %   RRMI3 0 RRMI2 0;0 RRMI3 0 RRMI2]

  A=[-26.75 0 25.796 0;0 -26.75 0 25.796;
    20.32 0 20.85 0;0 20.32 0 20.85];

  B=[1 0 0 0;0 1 0 0;0 0 -1 0;0 0 0 1];

  %C=[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1;
  %  MI1 0 MI3 0;0 MI1 0 MI3;0 MI3 0 MI2]

  C=[1 0 0 0;0 1 0 0;0 0 1 0;0 0 0 1;
 3536.42 0 -3409.87 0;0 3536.42 0 -3409.87;
-3409.87 0 3498.65 0;0 -3409.87 0 3498.65];

  D=[0 0 0 0;0 0 0 0;0 0 0 0;0 0 0 0;
  0 0 0 0;0 0 0 0;0 0 0 0;0 0 0 0];


%The root locus of the system is as follows
%[r,k]=rlocus(A,B,C,D);

%Equation #1- Determine dq reference values
%for stator current,from EQN (1)
% of Ref (1).
          I=[1/3^0.5 2/3^0.5; 1 0]*[Ia;Ib];
          Ids=I(1,1);
          Iqs=I(2,1);
```

```
%Equation #2- Determine stator flux
% "F"-Trapezoidal Rule
I0ds=0;V0ds=0;I0qs=0;V0qs=0;
mIds=(Ids-I0ds)/Ts;
mIqs=(Iqs-I0qs)/Ts;
mVds=(Vds-V0ds)/Ts;
mVqs=(Vqs-V0qs)/Ts;
t1=0;
for i=1:10,
T1=t1+Ts/10;
ids(1,i)=(mIds*T1)+I0ds;
iqs(1,i)=(mIqs*T1)+I0qs;
vds(1,i)=(mVds*T1)+V0ds;
vqs(1,i)=(mVqs*T1)+V0qs;
dFds(1,i)=(vds(1,i)-Rsm*(ids(1,i)));
dFqs(1,i)=(vqs(1,i)-Rsm*(iqs(1,i)));
t1=T1;
end

Fds=Ts/10*(sum(dFds)-0.5*(dFds(1,1))
-0.5*(dFds(1,10)));

Fqs=Ts/10*(sum(dFqs)-0.5*(dFqs(1,1))
-0.5*(dFqs(1,10)));

F=Fds+j*Fqs;



%Equation #3- Determine Torque, EQN (3) of
% Ref (1)
T=(3*p/4)*((Fds*Iqs)-(Fds*Ids));

%Equation #4a- Determine excitation frequency
% "w",from EQN (8) of Ref (1)
%w=(Fds*Vqs-Fqs*Vds)-Rsm*(Fds*Iqs-Fqs*Ids)
/(abs(Fds+j*Fqs))^2;

wreal=[((Vqs*Fds)-(Vds*Fqs))-((Iqs*Fds)
-(Ids*Fqs))*Rsm*((Fds^2)-(Fqs^2))]
/[((Fds^2)-(Fqs^2))^2+(4*(Fds^2)*(Fqs^2))];

wimag=[((Vds*Fqs)-(Vqs*Fds))+((Fds*Iqs)
-(Fqs*Ids))*Rsm*2*Fds*Fqs]
/[((Fds^2)-(Fqs^2))^2+(4*(Fds^2)*(Fqs^2))];
```

```
%Equation #4b- Determine slip frequency
%"sw" defined in the sync rotating
%reference frame aligned with the stator
% flux.Fqs^e=0;T=Fds^e*Iqs^e.
% This is for info only not used in the
% rest of simulation
%sw=(LS*Rr*Iqs)/LR*(Fds-Ls*Ids);

%Equation #5- Determine back emf of the
%motor "E", from EQN (7) of Ref (1)
%E=j*w*((Fds+j*Fqs)-Ls*(Ids+j*Iqs));
%Ed=real(E);
%Eq=imag(E);
Ed=-(wreal*Fqs)-(wimag*Fds)+Lsm*
[(wreal*Iqs)+(wimag*Ids)];

Eq=(wreal*Fds)-(wimag*Fqs)+
Lsm*[(-wreal*Ids)+(wimag*Iqs)];

E=Ed+j*Eq;

%Equation #6- Determine Change in Torque
%"DT",from EQN (12) of Ref (1)
%DT=(3*p*Ts/4*Ls)*((-Fds*Eq+Fqs*Ed)+
%(Fds*Vqs-Fqs*Vds));


%Equation #7-Determining Command Value of
% Stator Flux,from EQN (16) of Ref (1)

Frefd=Ts*(Vds-Rsm*Ids)+Fds;
Frefq=Ts*(Vqs-Rsm*Iqs)+Fqs;
Fref=((Frefd)^2+(Frefq)^2)^0.5;

%Equation #8- Determining Change in torque
%which will ensure dead beat control
%over a constant period.
DT1=Tref-T;


%Equation #9- Determining Kc, from
%EQN (13) of Ref (1)
Kc=(4*DT1*Lsm/3*p*Ts)+(Fds*Eq-Fqs*Ed);

%Equation #10- Determining the command
% value of stator flux magnitude
```

```
%deleted

%Equation #11- Defining the quadratic
%variables, EQN (18) of Ref (1)
%Where (a)*Vd^2 + (b)*Vd +(c)
%will be used to determine the two possible
%solutions for Vd. Once Vd is known Vq
%can be determined- See Equation #13
a=Ts^2+((Fqs)^2/(Fds)^2)*Ts^2;

b=((2*Kc*Fqs*Ts^2)/Fds^2)+2*Fds*Ts
+(2*(Fqs^2)*Ts)/Fds;

c=((Ts^2)*(Kc^2))/(Fds^2)+
(2*Fqs*Kc*Ts/Fds)+(Fds^2)+(Fqs^2)-Fref^2;

%Equation #12- Lets find out what the
% values of Vd are
p1=[a b c];
r=roots(p1);
r1=abs(r);
Vd1=r1(1,1)
Vd2=r1(2,1)

h=Vd1-Vd2;
if h<0, Vd=Vd1;
else Vd=Vd2;
end

%Equation #13- Determining the value of Vq,
% from EQN (14) of Ref (1)
Vq=(Kc+Fqs*Vd)/Fds;

%Equation #14- Determining the voltage space
% vector, from EQN(19) of Ref (1)
Vs=(Vd+j*Vq)+Rsm*(Ids+j*Iqs);
VD=real(Vs);
VQ=imag(Vs);

%Equation #15- Determining the Space Vector Angle
A=angle(Vs);


if Vs-abs(Vs)==0, SVA=A*180/pi;
elseif VQ==abs(VQ), SVA=A*180/pi;
else SVA=(A*180/pi)+360;
end
SVA;
```

```
%Determining n for Steady State
if SVA>=0 & SVA<=60,  n=1;
elseif SVA>60 & SVA<=120, n=2;
elseif SVA>120 & SVA<=180, n=3;
elseif SVA>180 & SVA<=240, n=4;
elseif SVA>240 & SVA<=300, n=5;
else   SVA>300 & SVA<=360, n=6;
end

%Equation #16- Determining inverter state times
x=(abs(Vs))/Vdc;
Sts= [0 000 1 100 2 110 7 111;
        7 111 2 110 3 010 0 000;
        0 000 3 010 4 011 7 111;
        7 111 4 011 5 001 0 000;
        0 000 5 001 6 101 7 111;
        7 111 6 101 1 100 0 000];
Tk=((x*sin(pi/3-A)*Ts)/sin(pi/3));
Tk1=((x*sin(A)*Ts)/sin(pi/3));

%For steady state condition
T0=Ts-Tk-Tk1;
if Tk>=0 & Tk1>=0 & T0>=0,
disp('Steady State condition'),
pause


if Sts(n,2)==Vr, Vi=Sts(n,2);
Vk=Sts(n,4);Vk1=Sts(n,6);Vr=Sts(n,8);
else Vi=Sts(n,8);Vk=Sts(n,6);
Vk1=Sts(n,4);Vr=Sts(n,2);
end


else disp('Transient Condition')
pause
end

B=(T-Tref);
C=(abs(F)-Fref);
AF=angle(F);
if F-abs(F)==0,FVA=AF*180/pi;
elseif Fqs==abs(Fqs), FVA=AF*180/pi;
else FVA=(AF*180/pi)+360;
end
```

```
%Determining value of N for
%Transient Conditions
if FVA>270 & FVA<=330, N=0;
elseif FVA>330 & FVA<=360, N=1;
elseif FVA>=0 & FVA<=30, N=1;
elseif FVA>30 & FVA<=90, N=2;
elseif FVA>90 & FVA<=150, N=3;
elseif FVA>150 & FVA<=210, N=4;
else   FVA>210 & FVA<=270, N=5;
end

%The Torque Transient Case
if B==abs(B),K=N+4;
else K=N+1;
K1=K+1;
end

%Determining Values for VK
%and Vk1 for Torque Transient case
VKt=(2/3)*Vdc*((cos((K-1)*
2*pi/3)+j*sin((K-1)*2*pi/3)));

VK1t=(2/3)*Vdc*((cos((K1-1)*
2*pi/3)+j*sin((K1-1)*2*pi/3)));

VKtr=real(Vkt);
VKti=imag(VKt);
VK1tr=real(VK1t);
VK1ti=imag(VK1t);

%Determining the switching times for
% each state
Tkt=(Fref-((Vk1t*Ts)+F))/(Vk1t-Vkt);
FTkt=(([real(Tkt)]^2)+
([imag(Tkt)]^2))^0.5;

Tk1t=Ts-FTkt;

if Tkt<0,disp('Flux Transient Condition')
elseif Tk1t<0,
disp('Flux Transient Condition')
else disp('Torque Transient Condition')
pause
end

%The Flux Transient Case
if C==abs(C),Kf=N+2;
else Kf=N;
```

```
Kf1=Kf+1;
end

%Determining Values of Vk and
% Vk1 for Flux Transient Case
VKf=(2/3)*Vdc*((cos((Kf-1)*
2*pi/3)+j*sin((Kf-1)*2*pi/3)));

VK1f=(2/3)*Vdc*((cos((Kf1-1)*
2*pi/3)+j*sin((Kf1-1)*2*pi/3)));

VKfr=real(VKf);
VKfi=imag(VKf);
VK1fr=real(VK1f);
VK1fi=imag(VK1f);

%Determining the switching times for
% each state
Tkf=[((DT1*4*Lsm)/(3*p*F))+
(E*Ts)-(VK1f*Ts)]/[VKf-VK1f]

FTkf=(([real(Tkf)]^2)+
([imag(Tkf)]^2))^0.5;

Tk1f=Ts-FTkf;

if Tkf<0,
disp('Torque and Flux Transient Condition')
elseif Tk1f<0,
disp('Torque and Flux Transient Condition')
else disp('Flux Transient Condition')
pause
end

%Determining Values of Single State
% for Torque and FLux Transient
if B==abs(B) & C==abs(C),  Ktf=N+5;
elseif B==abs(B), Ktf=N+4;
elseif C==abs(C), Ktf=N+2;
else Ktf=N+1;
end

%I0ds=Ids; I0qs=Iqs; V0ds=Vds;
V0qs=Vqs;g=G;ZIds(G)=Ids;
%ZIqs(G)=Iqs;
%ZFds(G)=Fds;
%ZFqs(G)=Fqs;
%Zw(G)=w;
```

```
%ZT(G)=T;
%Zsw(G)=sw;
%ZEd(G)=Ed;
%ZEq(G)=Eq;
%ZVd1(G)=Vd1;
%ZVd2(G)=Vd2;
%ZVq(G)=Vq;
%ZTk(G)=Tk;
%ZTk1(G)=Tk1;
%ZT0(G)=T0;
%Zz(G)=z;

%Computation of Per Phase Motor Output Voltage
%aa=[2.873E-04 .0373 0.154];
%bb=[0.03499 0 0];
%ww=logspace(-1,1);
%hh=freqs(bb,aa,ww);
%mag=abs(hh);phase=angle(hh);
%mv=hh*Vas;
%Zmv(G)=mv;
%f=ww/(2*pi);

%Computation of Per Phase Motor Output Current
%aaa=[84.33E-03 0]
%bbb=[0.08766 .415]
%www=logspace(-1,1);
%hhh=freqs(bbb,aaa,www);
%mi=hhh*Ias;
%Zmi(G)=mi;
%ff=www/(2*pi);

end
```

APPENDIX B- SIMULINK Individual Block Diagrams

APPENDIX B- LISTING OF SUBSYSTEMS

Subsystem- Determination of E_d (Transient Reactance Voltage -d axis Reference frame)

Subsystem 1-Determination of E_q (Transient Reactance Voltage -q axis Reference frame)

Subsystem 2-Determination of Vkr and Vki (Torque Transient Stator Voltage Vector -real and imaginary)

Subsystem 3-Determination of whether steady state or transient condition

Subsystem 4-Determination of abc- (For quadratic equation- Solution for Vd*)

Subsystem 5-Determination of K_c

Subsystem 6-Determination of F_s (Command Value of stator flux magnitude)

Subsystem 7-Determination of V_d* (smallest d axis voltage necessary to drive torque and flux to their reference value)

Subsystem 8-Determination of V_q (smallest q axis voltage necessary to drive torque and flux to their reference value)

Subsystem 9- Determination of SVA, A, VD and VQ (Space Vector Angle [Tan $^{-1}$ Vq*/Vd*], Angle in radians, VD and VQ is the smallest d&q axis voltage plus the stator IR drop)

Subsystem 10-Determination of Tk, Tk1 and T0 (Steady State times for stator voltage vector Vk and Vk1)

Subsystem 11-Determination of state and times

Subsystem 12-Determination of Zone 1 for steady state

Subsystem 13-Determination of Zone 2 for steady state

Subsystem 14-Determination of Zone 3 for steady state

Subsystem 15-Determination of Zone 4 for steady state

Subsystem 16-Determination of Zone 5 for steady state

Subsystem 17-Determination of Zone 6 for steady state

Subsystem 18-Determination of the flux angle

Subsystem 19- not used

Subsystem 20-Determination of the absolute value of the flux vector

Subsystem 21-Determination of value of "n" for torque transient

Subsystem 22-Determination of TVK1r and TVK1i (Torque Transient Stator Voltage Vector Vk1 real and imaginary parts)

Subsystem 23-Determination of TTK and TTK1(Torque Transient times)

Subsystem 24-Determination of FVsKr and FVsKi (Flux Transient Stator Voltage Vector Vk real and imaginary parts)

Subsystem 25-Determination of "n" for flux transient (Used in Lookup Table)

Subsystem 26-Determination of FVsK1r and FVsK1i (Flux Transient Stator Voltage Vector Vk1 real and imaginary parts)

Subsystem 27-Determination of FTK and FTK1(Flux Transient times)

Subsystem 28- Not used

Subsystem 29- Determination of W_s (Excitation Frequency)

Subsystem 30- Determination of transient torque and flux (Table 1c)

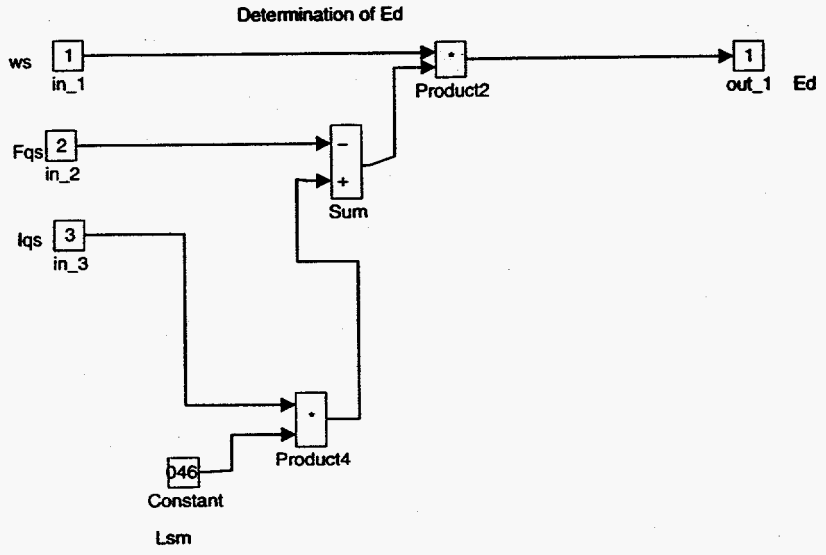
Subsystem 31-Determination of "n" for transient torque and flux condition

Subsystem 32-Resulting state times with stator voltage states for transient cases

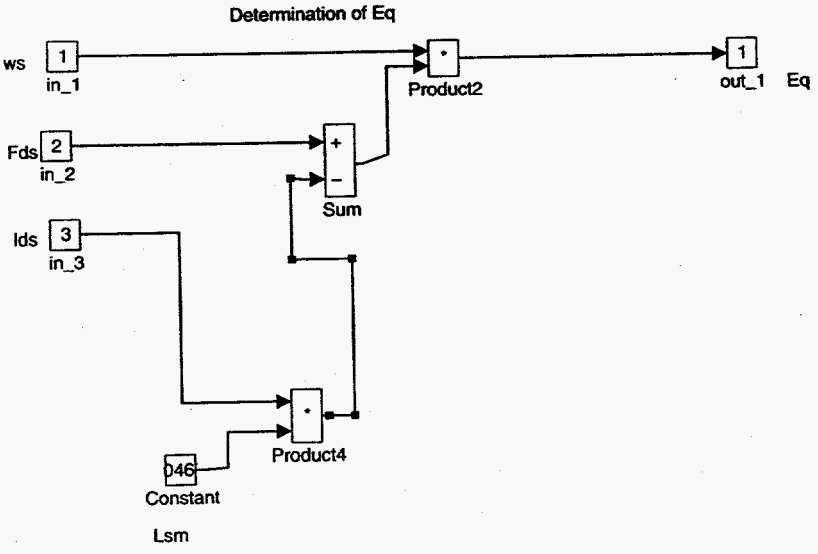
Subsystem 33-Determination of last valid input for steady state case

Subsystem 34-Determination of states for transients

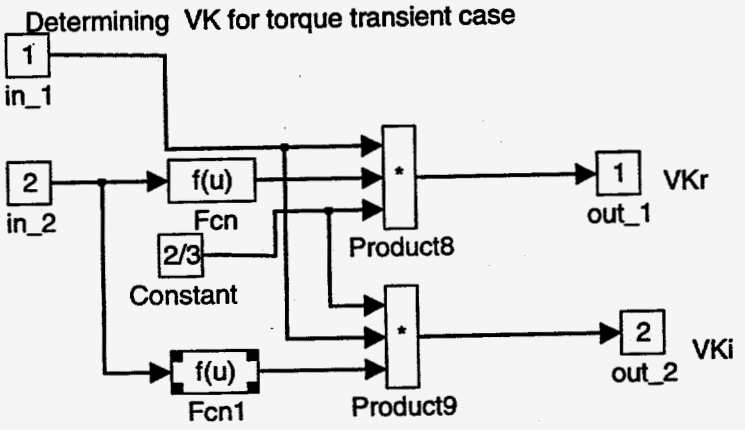
Subsystem



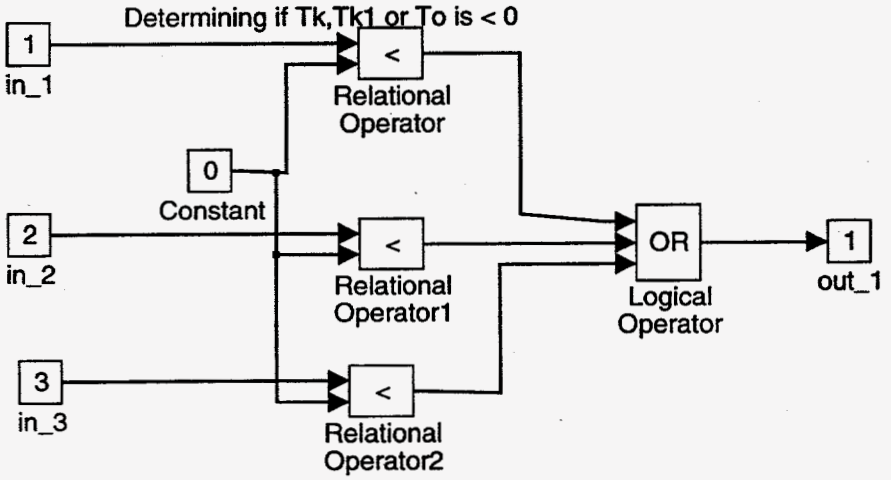
Subsystem 1



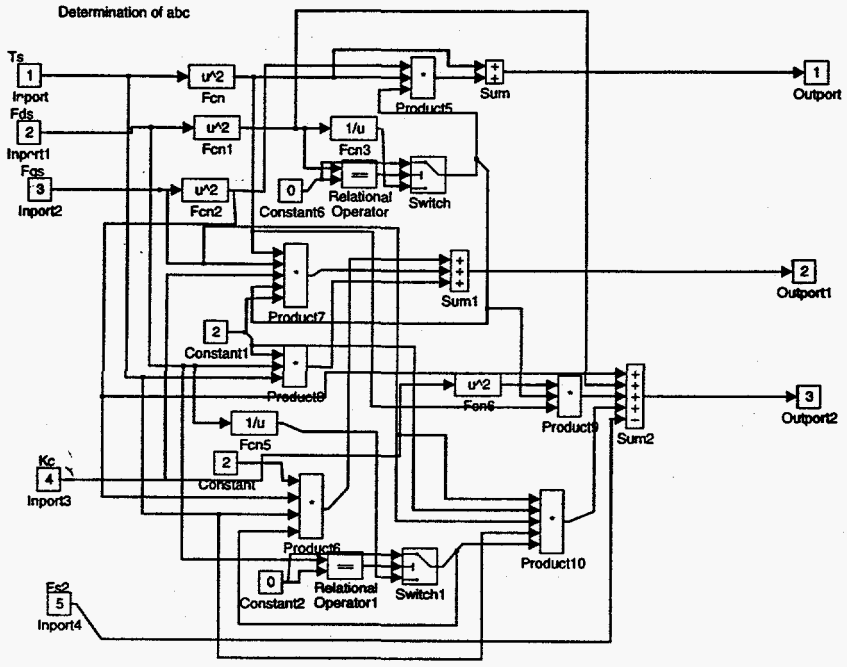
Subsystem 2



Subsystem3

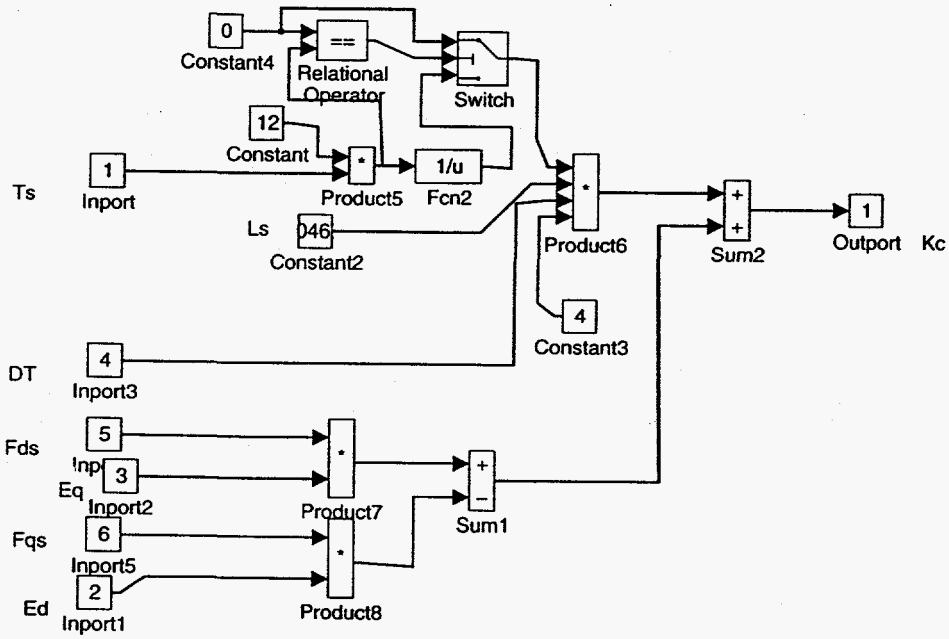


Subsystem 4

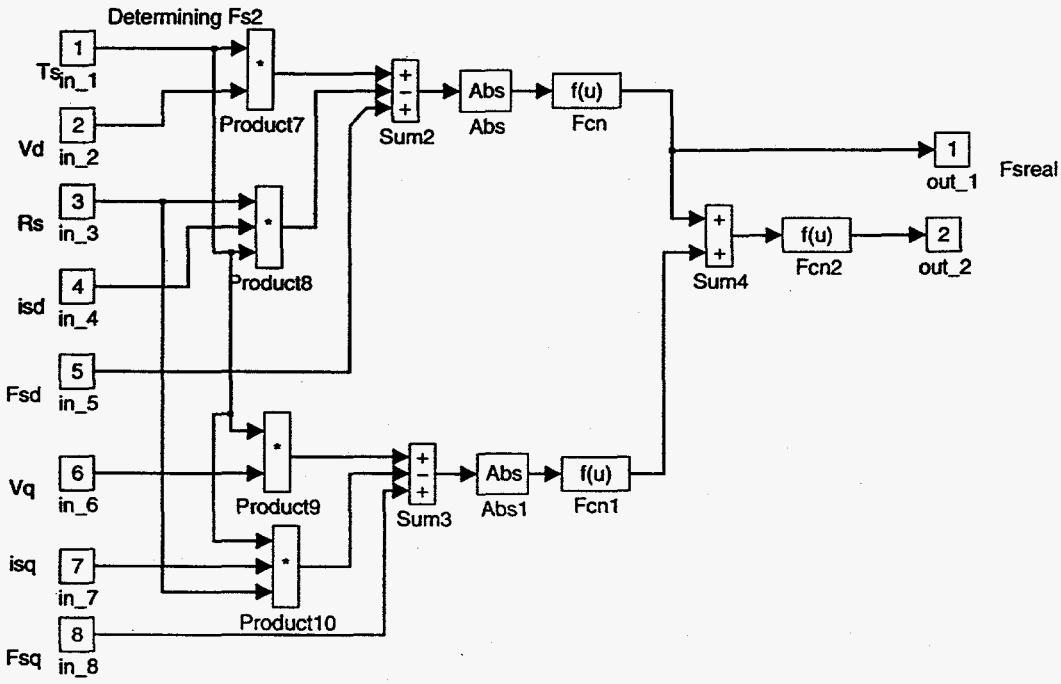


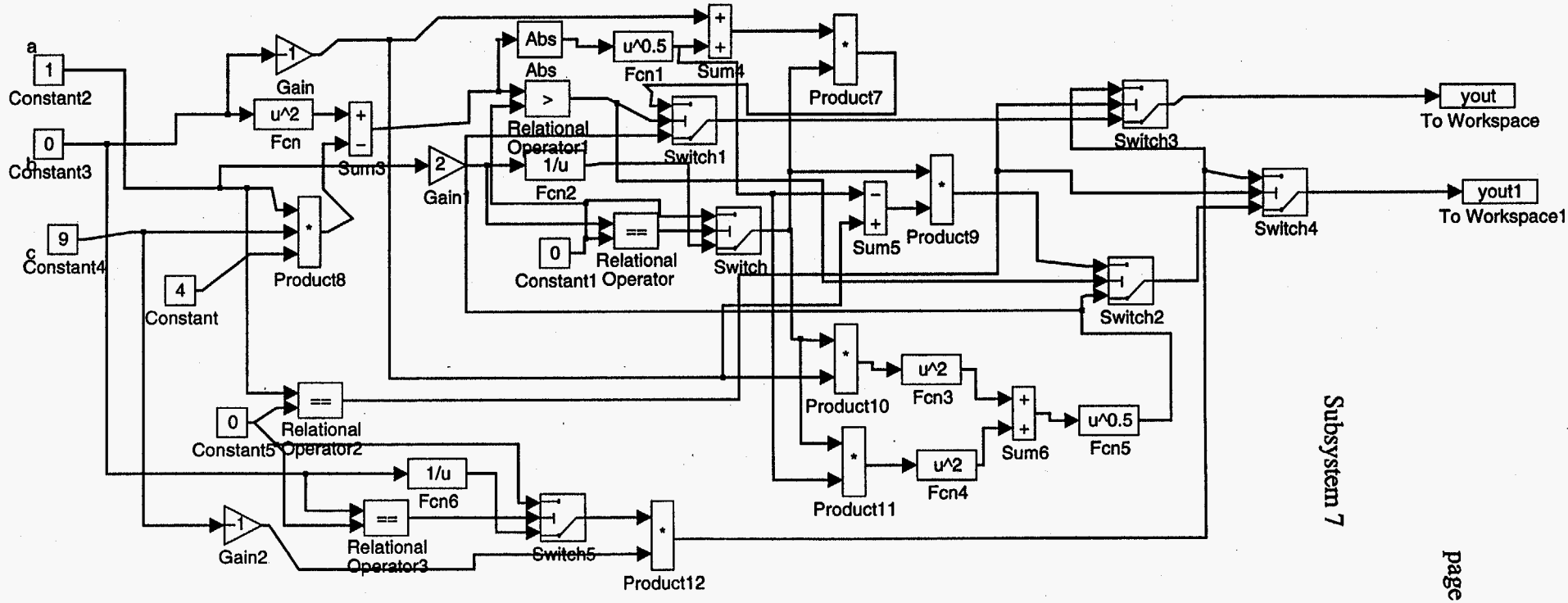
Subsystem 5

Determination of Kc



Subsystem 6

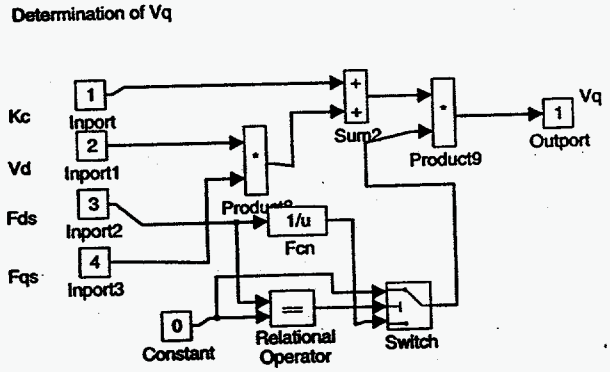




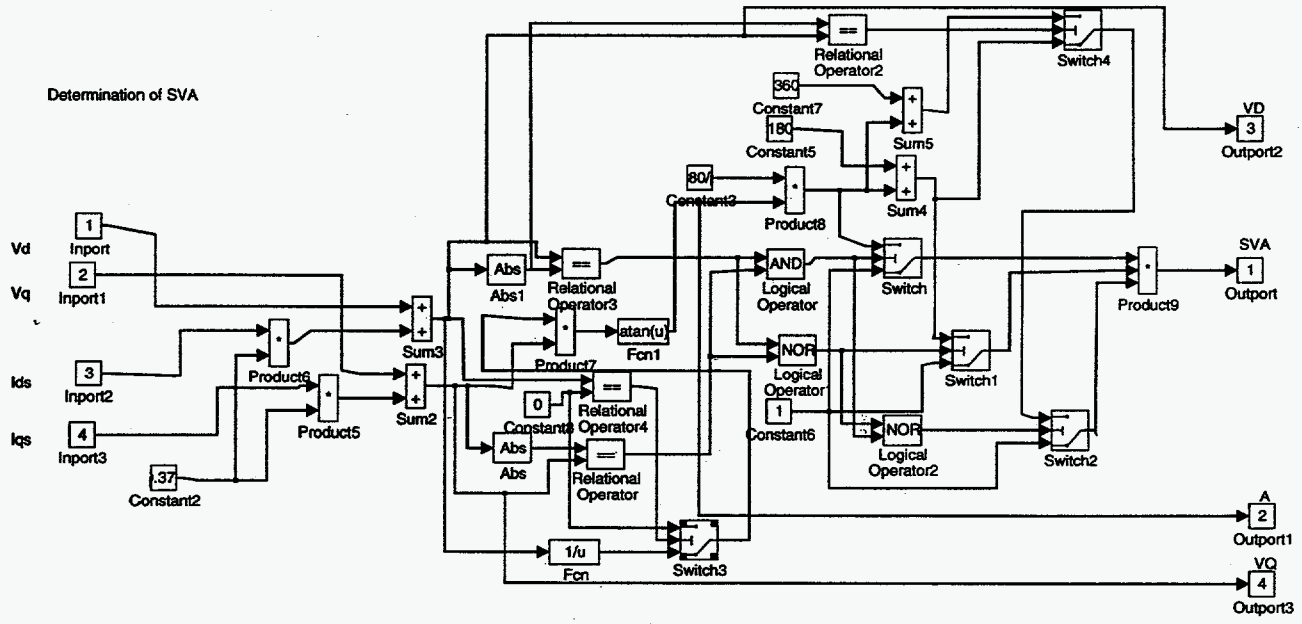
Subsystem 7

Determination of Vd*

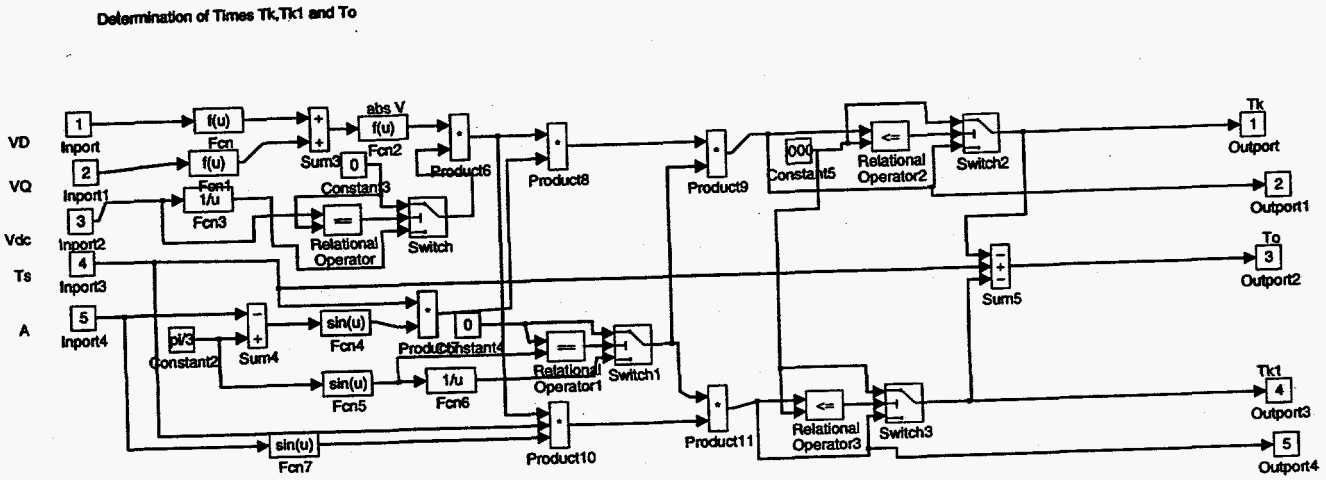
Subsystem 8



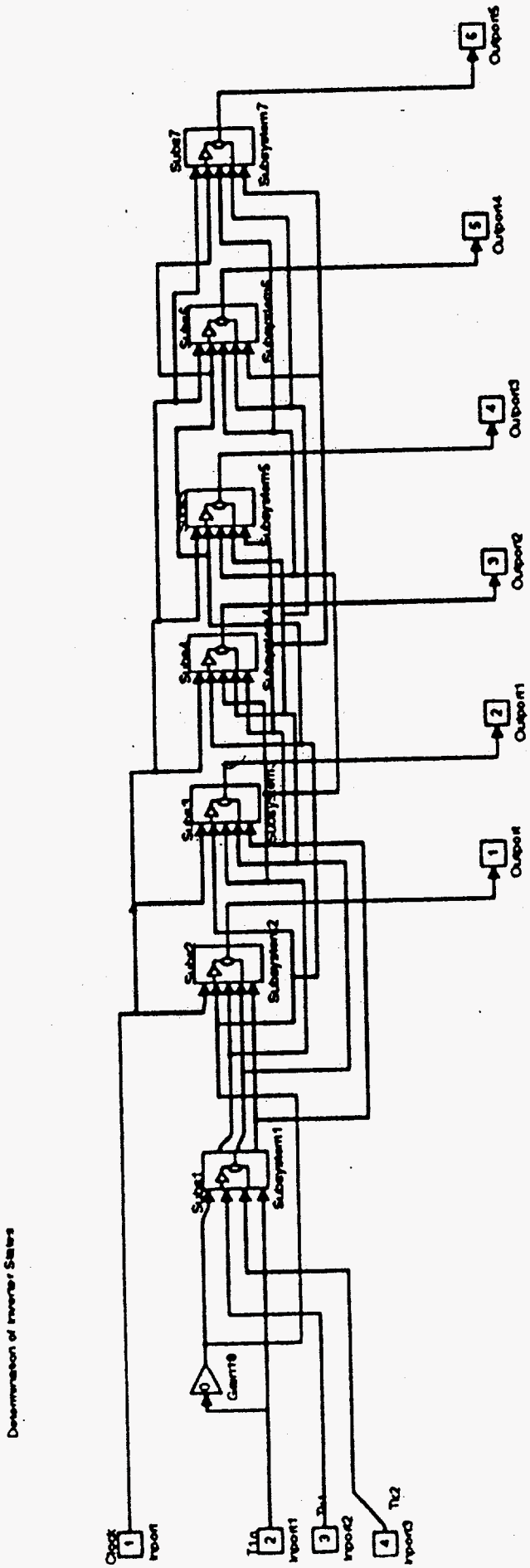
Subsystem 9



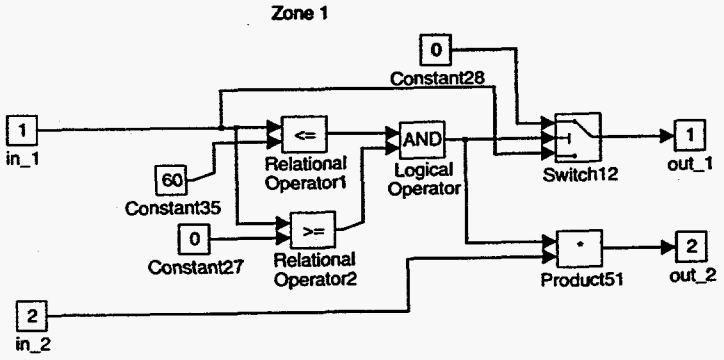
Subsystem 10



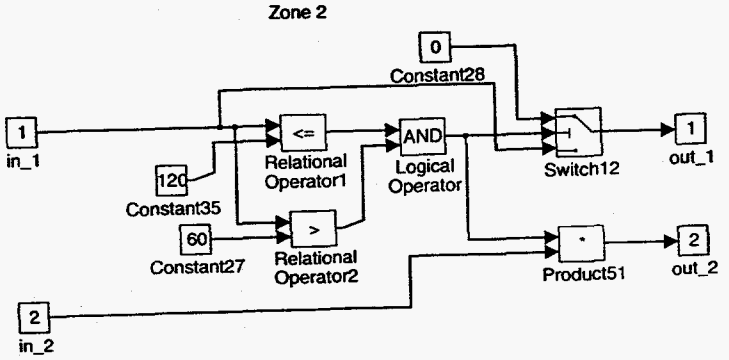
Subsystem 11



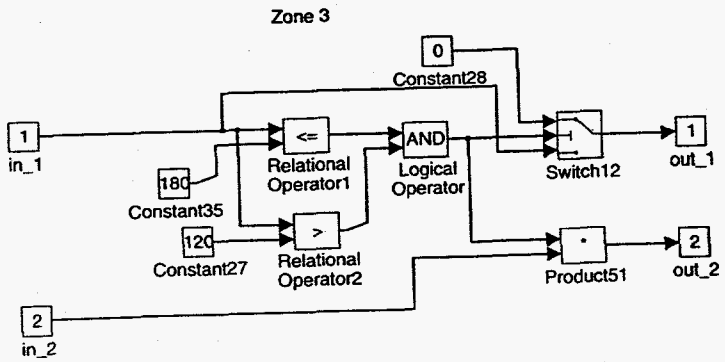
Subsystem 12



Subsystem 13

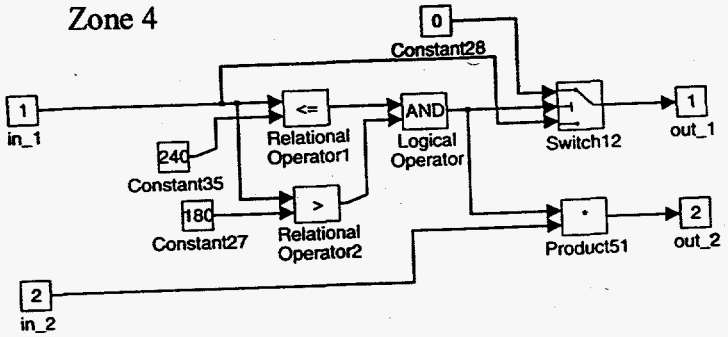


Subsystem 14



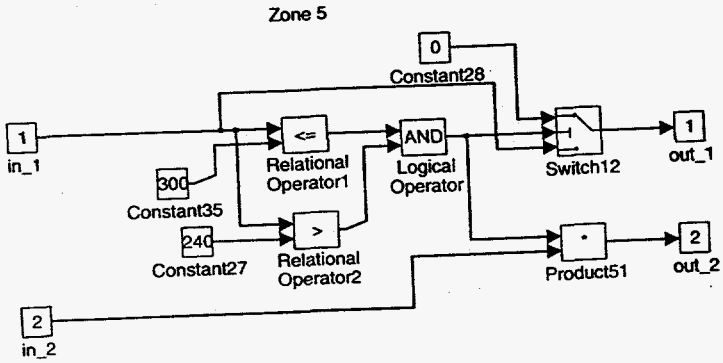
Subsystem 15

Zone 4



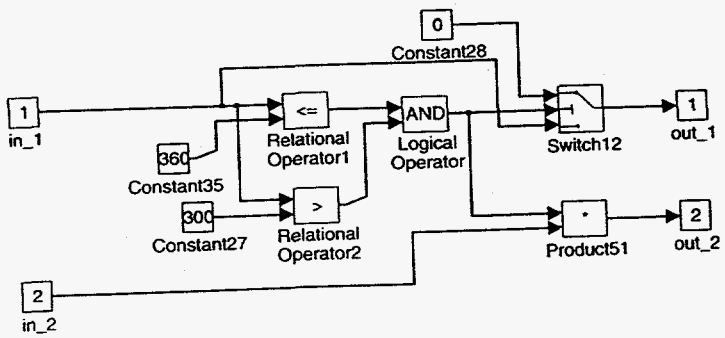
Subsystem 16

Zone 5

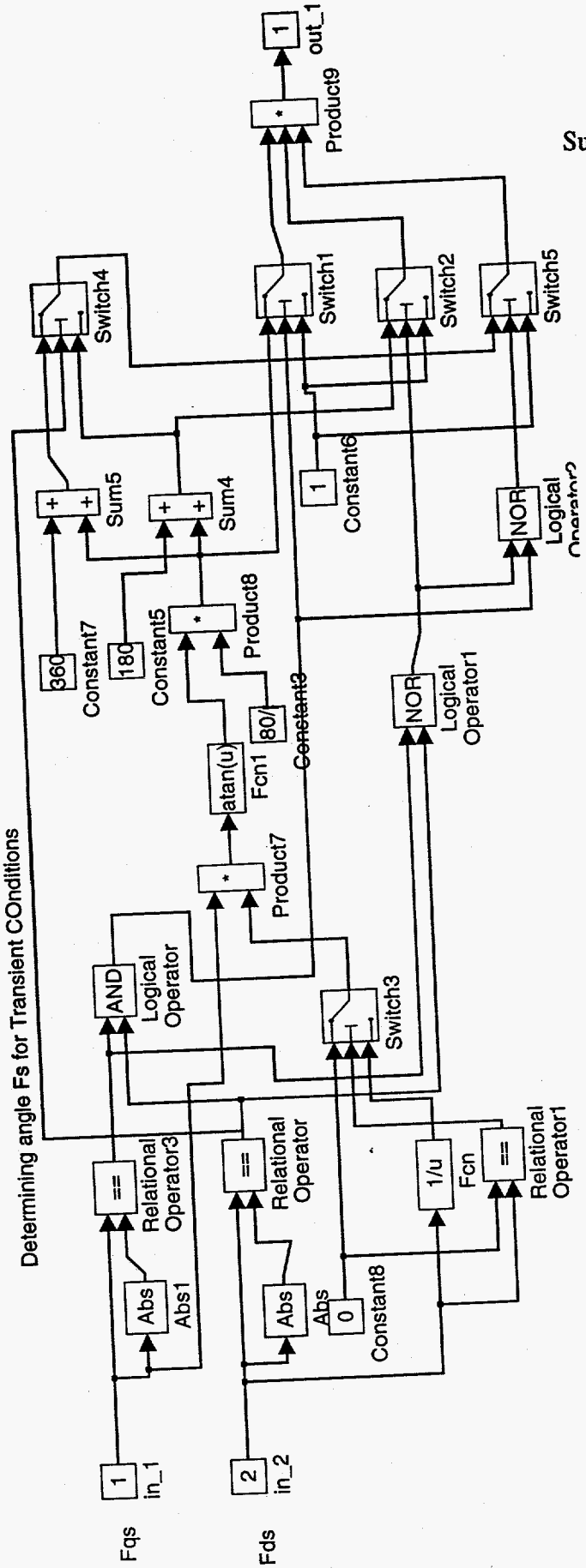


Subsystem 17

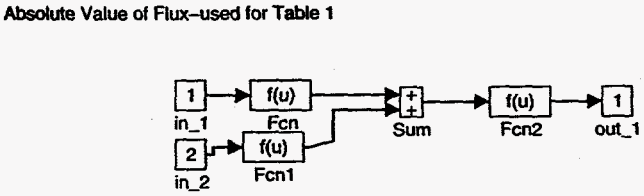
Zone 6



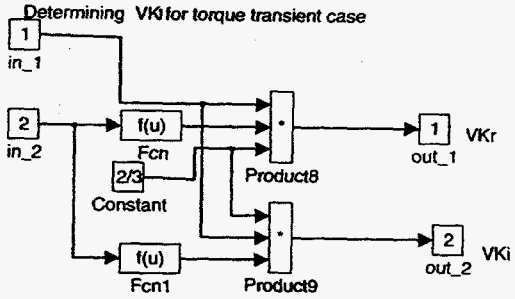
Subsystem 18



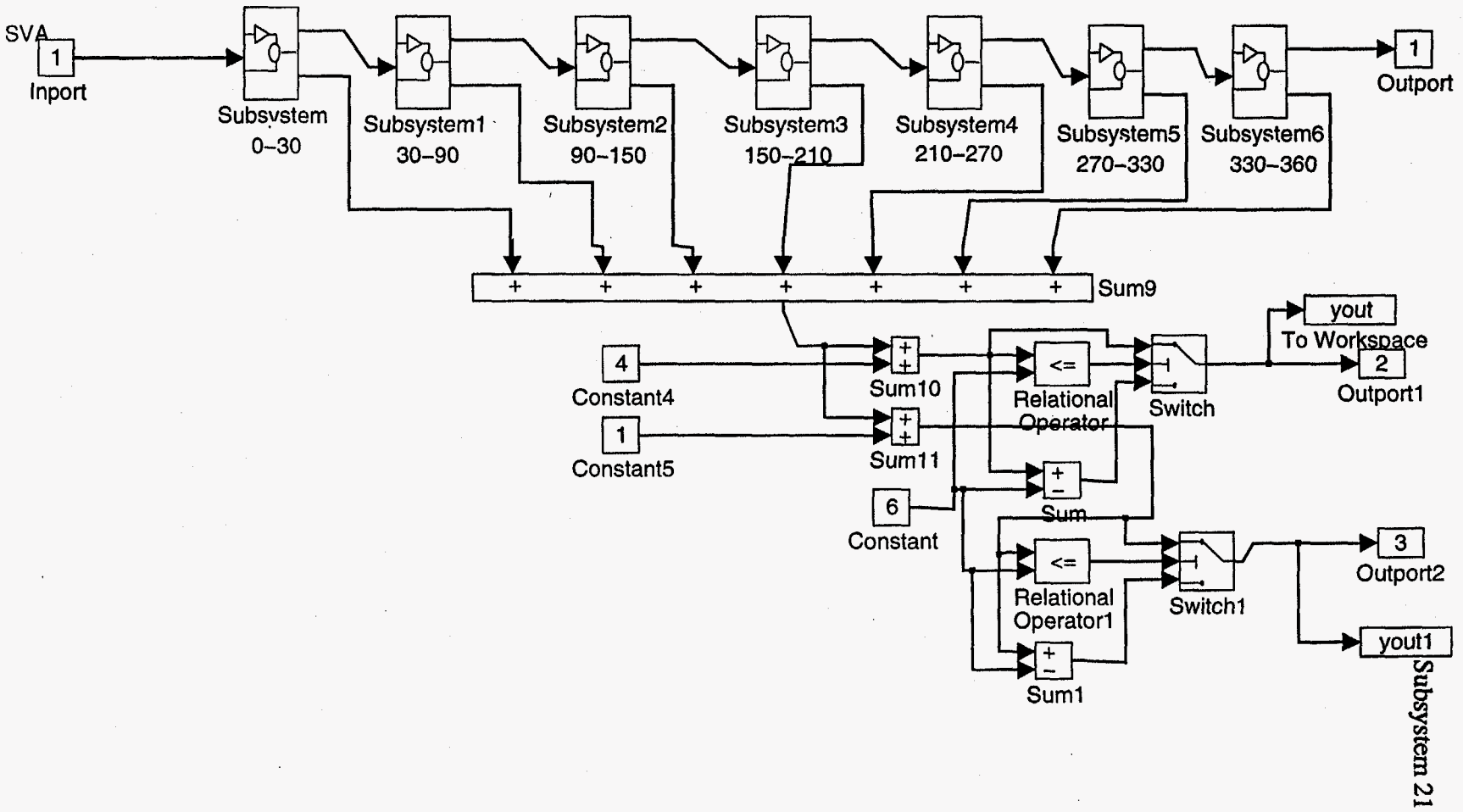
Subsystem 20

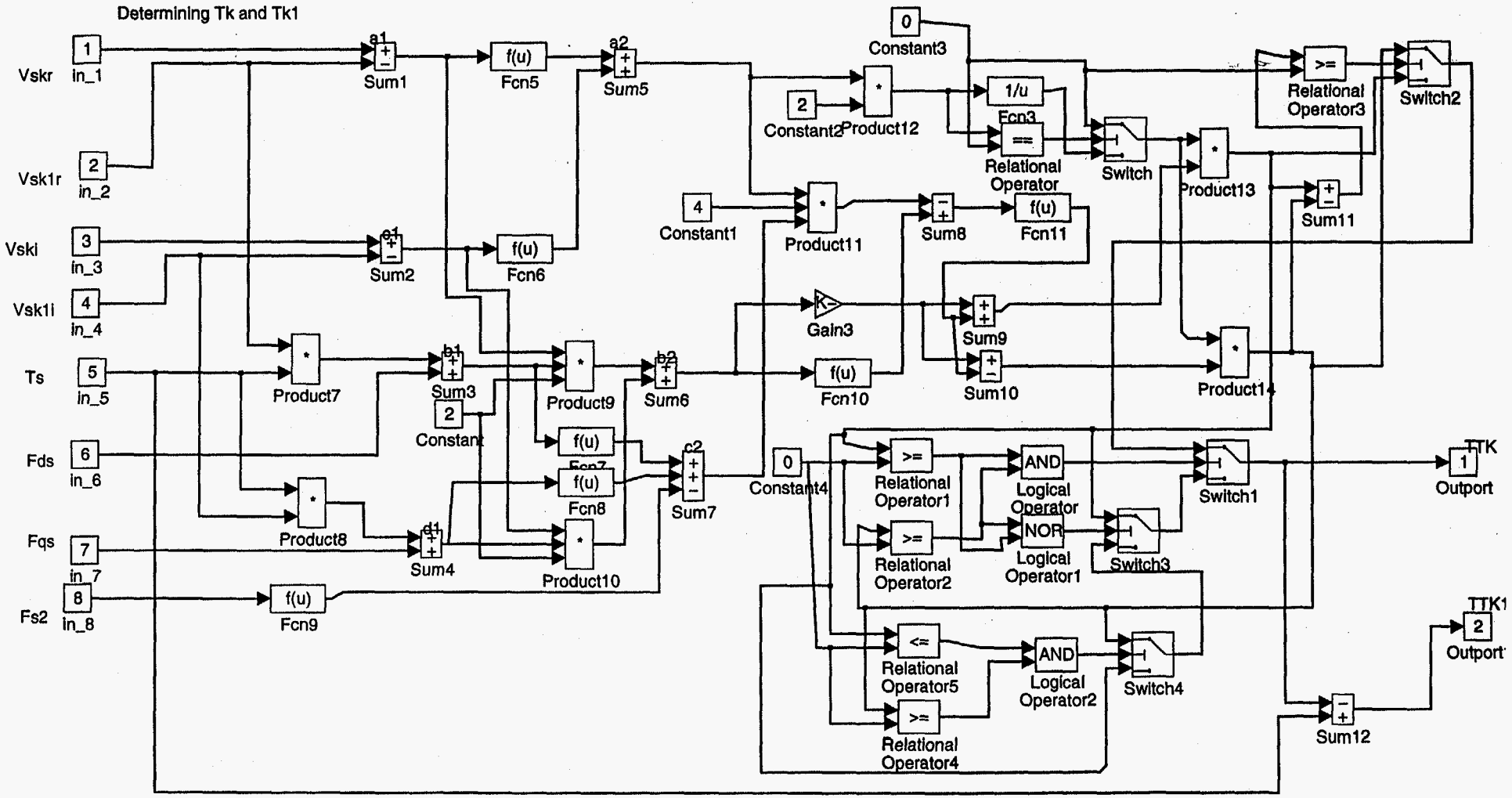


Subsystem 22,24,26



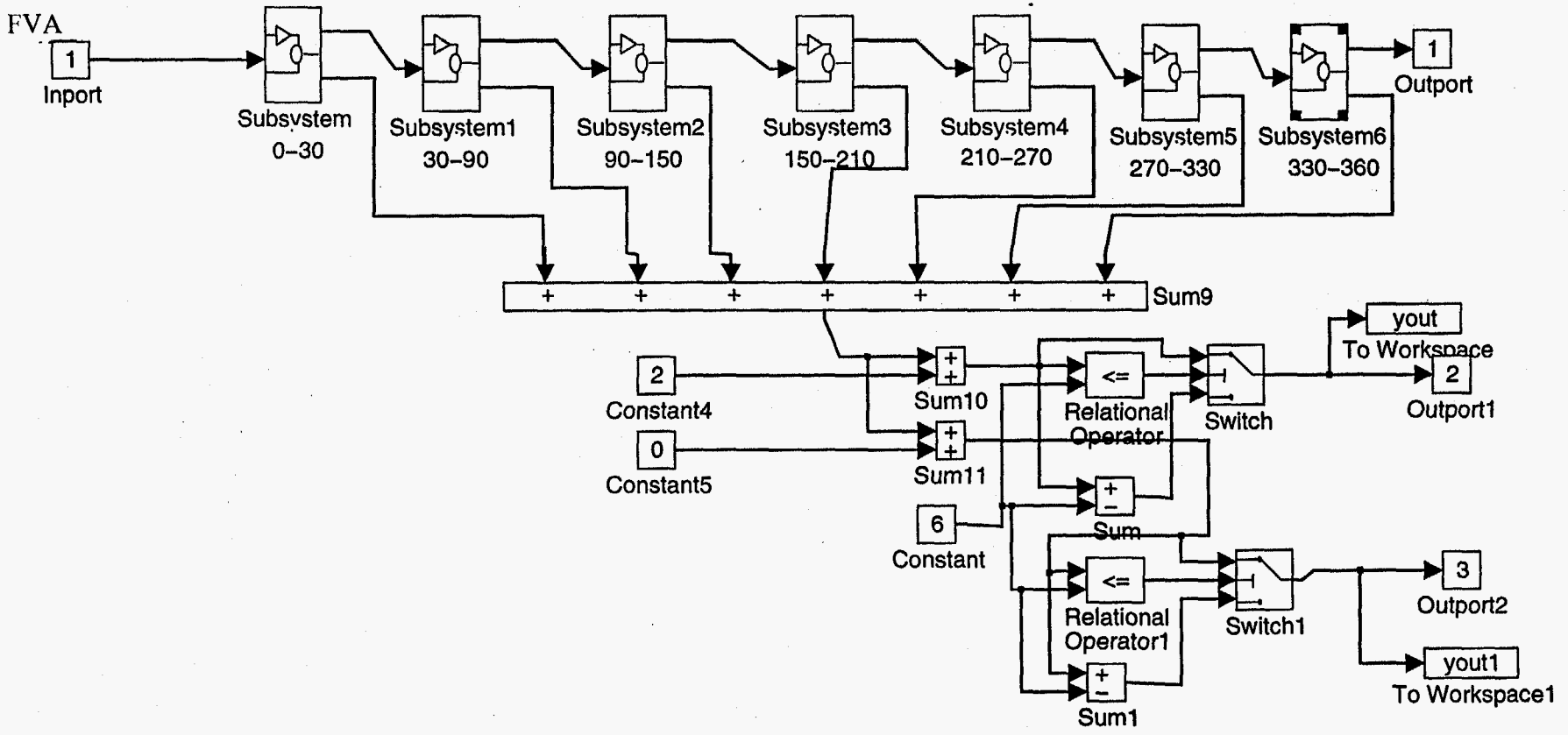
Determination of n for Torque Transient



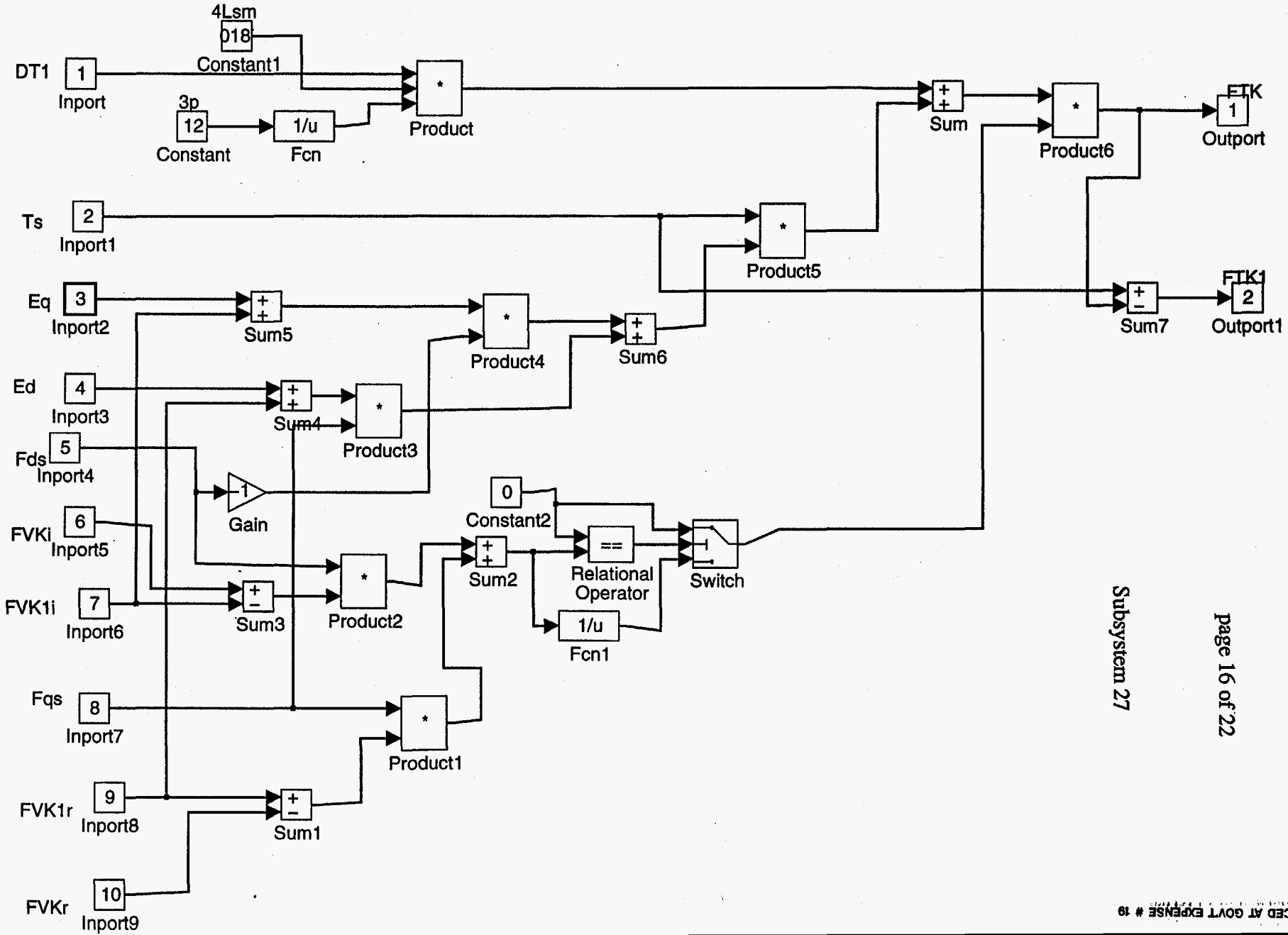


Subsystem 23

Determination of n for the Flux Transient

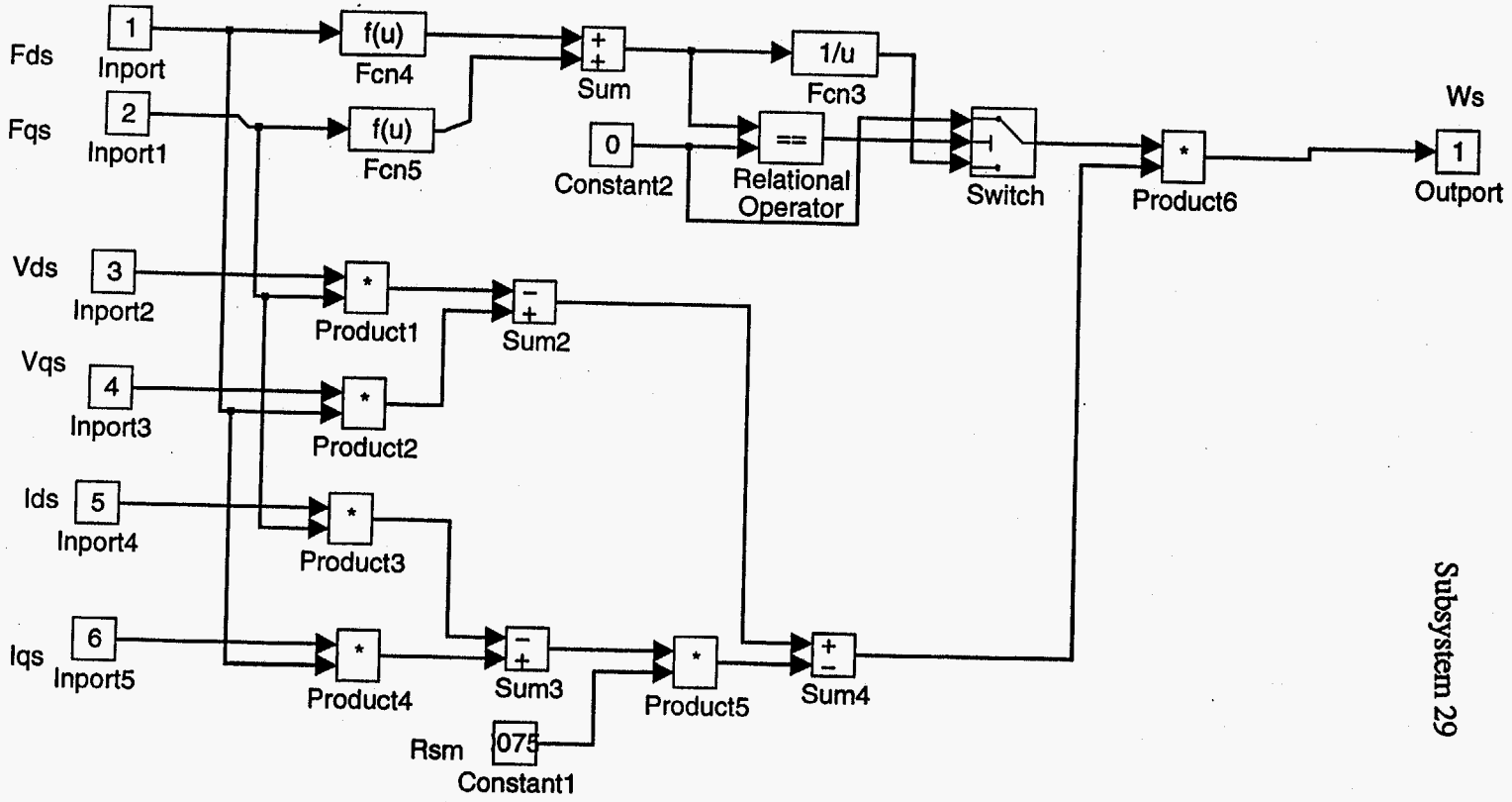


Subsystem 25

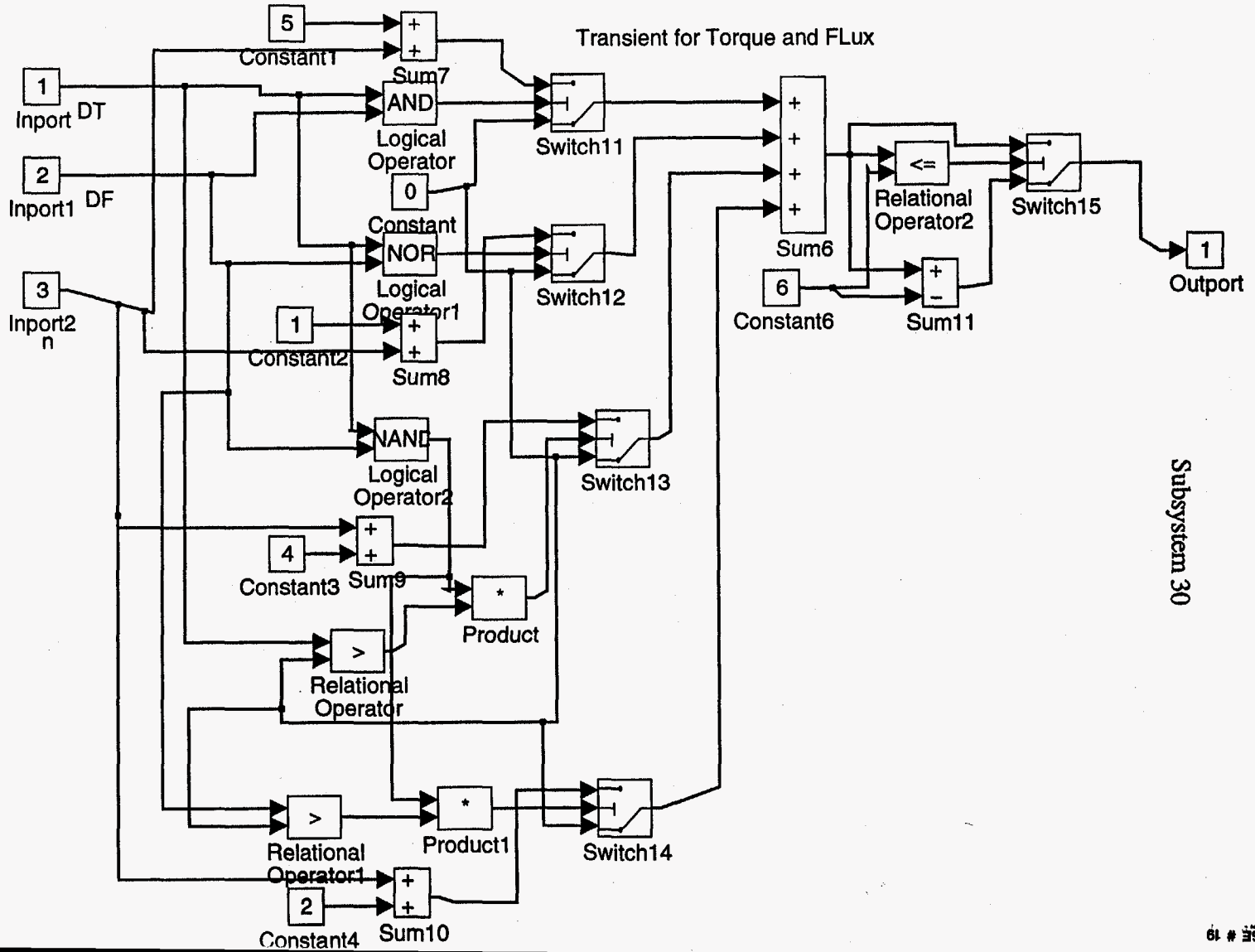


Subsystem 27

Determination of W_s



Subsystem 29



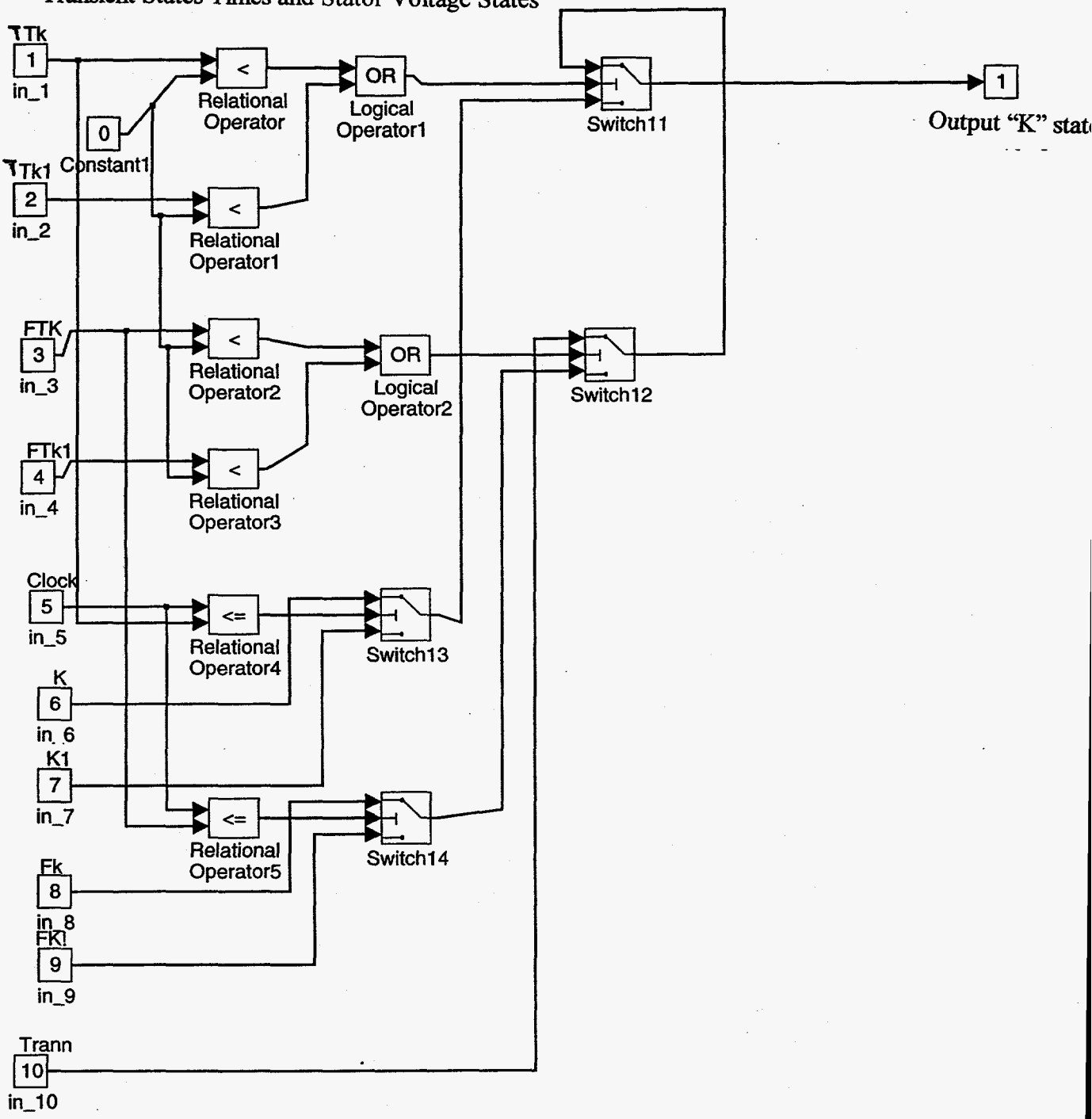
Subsystem 30

Determination of n for transient torque and flux

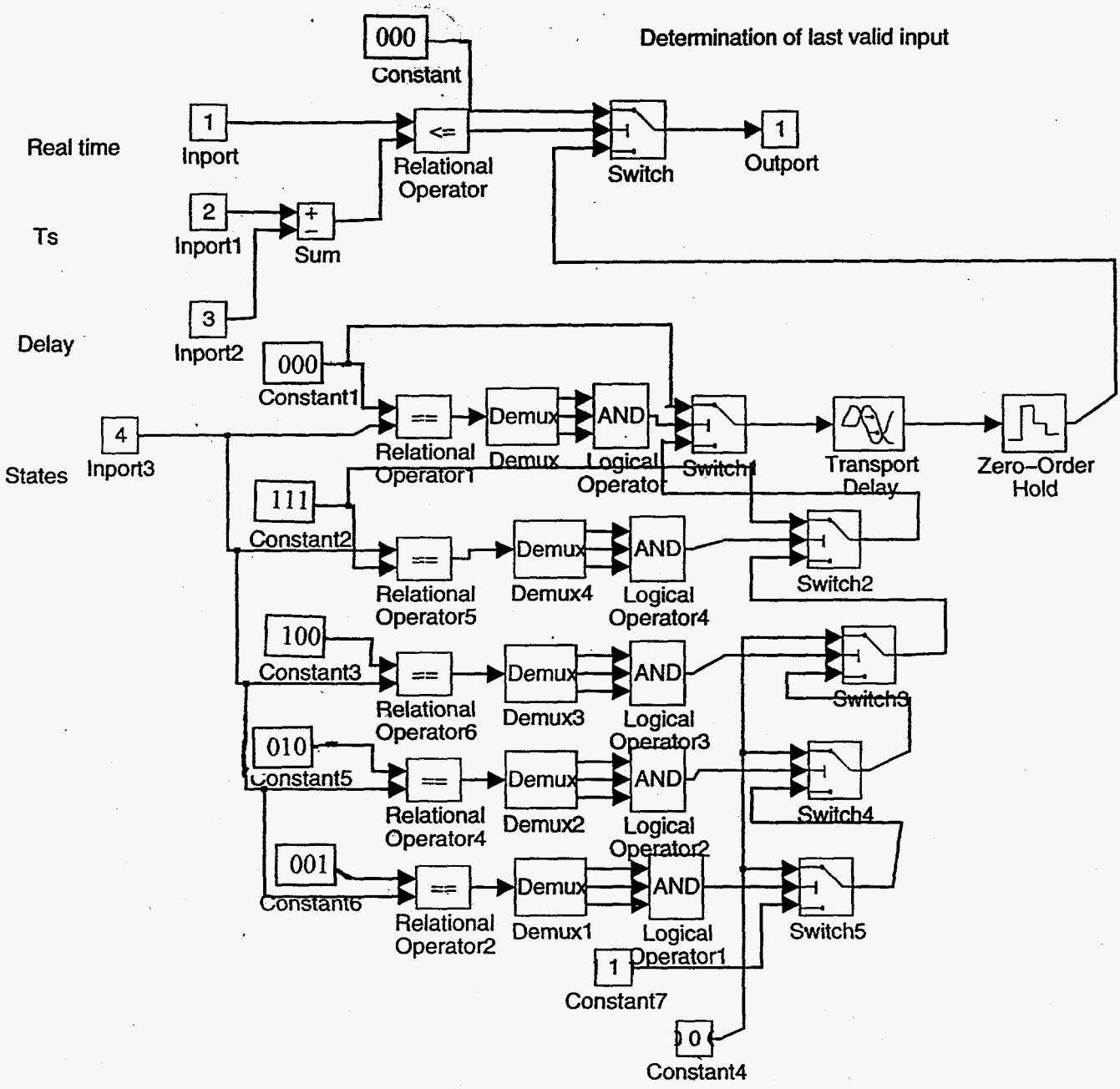
Subsystem 31

Subsystem 32

Transient States Times and Stator Voltage States

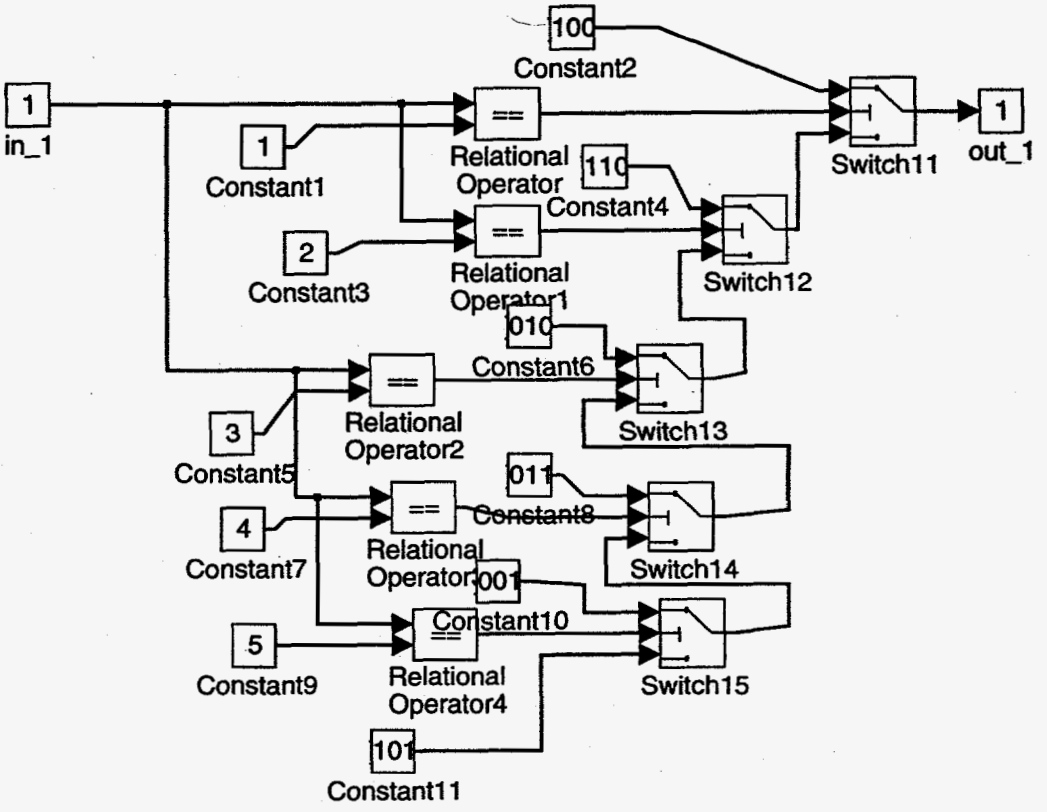


Subsystem 33



Subsystem 34

Determination of states for transients



APPENDIX C.1- ANSI C Program Code (JULIE7.c)

```
/*03/14/97: C Program developed to simulate inputs (VD and VQ),
calculate states and corresponding times to be fed back to
the inverter(phases a,b and c) which feeds the
induction motor.*/

        /*Setting up initial conditions-needed libraries*/
        #include "stdio.h"
        #include "stdlib.h"
        #include "math.h"
        #include "mathl.h"
        #include "miscl.h"
        #include "util.h"
        #include "21020.h"
        #include "signal.h"
        #define PI 3.1416

/*Defining the Functions for this program*/
void START();
void STEAD_STATE();
void TORQUE_TRAN();
void FLUX_TRAN();
void TORQUE_FLUX_TRAN();

/*Global Variables*/
volatile int mann;
volatile int mann1;
volatile int z;
int u=0;
int u1=0;
int y1=0;
int y1a=0;
int y3=0;
int y3a=0;
int y5=0;
int y5a=0;
int y7=0;
int y7a=0;

int SLVAL=0;      /*This is the initial "last valid state" to be
                    used for various funtions*/
```

```
int TAB[8][5]={{0,0,1,3,5},        /*TAB will be used later in the torque*/
               {1,0,6,1,2},       /*and flux transients*/
               {2,7,1,2,3},
               {3,0,2,3,4},
               {4,7,3,4,5},
               {5,0,4,5,6},
               {6,7,5,6,1},
               {7,7,2,4,6}
                        };

float RESULT[10][9]={0,0,0,0,0,0,0,0,0,0};   /*Initial Condition of RESULT defined
                                               States/corresponding times will be
                                               stored in this matrix*/


int RESULT1[10][4]={0,0,0,0}; /*RESULT1 will be used to store converted
                                times that will be used by the timer_set
                                instruction*/


                  void timer_isr(int t);    /*Declaring timers for this program*/
                  void timer1_isr(int t);
                  void timer2_isr(int t);
                  void timer3_isr(int t);

void main( )
                   /*Start of main program, defining variables to be used*/
{  float Vdc;      /*Vdc is the dc input voltage to the inverter*/
   float Ts;       /*  Ts is the switching period(sec)- time in which to sample input
                      to induction motor,calculate new values for matrix RESULT
                     while executing previous switching period RESULT*/

   float VD;   /*d axis portion of the calculated space vector*/
   float VQ;   /*q axis portion of the calculated space vector*/
   float Tref; /*This is the torque reference value*/
   float T;    /*This is the simulated value of torque as observed
                              during the execution of the simulink model*/
   float F;    /* This is the simulated value of flux as observed
                                  during the execution of the simulink model*/
   float F1;   /*absolute value of F*/
   float F2;   /* F*F */
   float F3;
   float Fref;       /* This is the flux reference value*/
   float AF;         /* This is the simulated flux angle as observed in the
                                  simulink model*/
   float Fds;        /* This is the d axis portion of the calculated flux angle*/
   float Fds1;
   float Fds2;
```

```
float Fqs;          /* This is the q axis portion of the calculated flux angle*/
float Fqs1;
float DT1;  /* This is the change in torque from the reference value*/
float Ed;   /* This is the transient reactance in the d axis*/
float Eq;   /* This is the transient reactance in the q axis*/
float Lsm;  /* This is the stator inductance of the modeled induction motor*/
float p;    /* This is the number of poles for the modeled induction motor*/
float y;    /* The "y" values will be used to convert times(sec) to values
                    to be used with the timer_set instruction*/

float y2;
float y4;
float y6;
float t1;   /*t1 will be used for the for loop found in the main program*/
float t2=0;
int t;
Lsm=0.0046777;
p=4.0;
mann=0;
mann1=1;
Vdc=10.0;
Ts= 0.000250;  /*Period for this program is 250 usec or a switching frequency
                                of 4000 Hz*/
Tref=100.0;
T=1000.0;
F=250;
Fref=300;
Ed=400;
Eq=200;
DT1=Tref-T;

timer_set(8334,8334);  /*Setting the initial timer for 250 usec[8334*30nsec]
                                where 30 ns is the cycle time for the dsp*/
interrupt(SIG_TMZ0, timer_isr);       /*This is the initial interrupt which directs
                                that the interrupt handler "timer_isr" be serviced
                                at the end of the 250 usec*/
timer_on();     /* The timer starts counting down*/

set_flag(SET_FLAG1,CLR_FLAG);
    set_flag(SET_FLAG2,CLR_FLAG);             /*Setting initial*/
set_flag(SET_FLAG3,CLR_FLAG);                    /*flags for viewing*/

/*The flags are used to represent the output of the dsp which will feed the
  six step inverter. Phase a will be flag1,phase b is flag2 and phase c is
  flag3. When the phase is high(1) then the associated light will be on.
  Likewise when the phase is low(0) the associated light will be off.*/
```

```
for(;;)          /*Start of the infinite for loop*/
 {
    t2++;
    t=mann1;
    t1=mann1*PI/6;
    VD=sin(t1+PI);    /*Simulating the calculated space vector*/
    VQ=sin(t1+PI*2.0/3.0);
    AF=sin(t1);
    Fds=sin(t1);
    Fds1=fabs(Fds);
    if (Fds1 !=0)
    Fds2=Fds1*Fds1;
    else Fds2=0;

    F1=fabs(F);
    if (F1 !=0)
    F2=F1*F1;
    else F2=0;

    F3=F2-Fds2;
    if (F3 !=0)
    Fqs=pow((F3),0.5);
    else Fqs=0;

    if (Fds1 !=0 && F3 !=0)
    AF=tan (Fqs/Fds);
    else AF=0;


START(VD,VQ,Vdc,Ts,T,Tref,F,Fref,AF,Fqs,Fds,Lsm,DT1,Eq,Ed,p,t);
        /*START will calculate times to determine if the condition is a steady
          state condition or a potential torque transient condition*/

y=RESULT[mann1][0];          /*Calculating times for timer_set*/
y1=(y/0.000000030);
RESULT1[mann1][0]=y1;

u=RESULT[mann1][8];          /* Additional timer interrupts may be needed
                                    depending on the resulting condition*/
```

```
switch (u)
{

        case 3:      /*This is the flux transient case*/
        case 2:     /* This is the torque transient case*/
       y2=RESULT[mann1][2];
       y3=(y2/0.000000030);
       RESULT1[mann1][1]=y3;
       break;

        case 1:     /* This is the steady state case*/
       y2=RESULT[mann1][2];
       y3=(y2/0.000000030);
       RESULT1[mann1][1]=y3;

       y4=RESULT[mann1][4];
       y5=(y4/0.000000030);
       RESULT1[mann1][2]=y5;

       y6=RESULT[mann1][6];
       y7=(y6/0.000000030);
       RESULT1[mann1][3]=y7;
       break;

        default:
}


        if (z==1)     /*Interrupts are obviously asynchronous, therefore
                        it is necessary to have idle conditions established
                        to ensure that the interrupts are handled properly
                        and at the right time.Each of these idle conditions
                        ensures that the program stays in an idle condition
                        until it is time to service the next interrupt handler*/
        idle();
        if(z==2)
        idle();
        if (z==3)
        idle();
        if (z==0)
        idle();
  }
}
```

```
void timer_isr(int t) /*This is the first interrupt handler and is
                                    used for all four cases(steady state,torque transient,
                                flux transient or torque and flux transient).*/

{ if (mann==8)
  {mann=0;}    /*This sets up RESULT to fill the next row in the matrix
 else mann++;    and creates a loop in the matrix*/
 if (mann1==8)
  {mann1=0;}
 else mann1++;

    u1=RESULT[mann][1];    /*This value is the first state (State 1)
                                                calculated*/
    switch (u1)
    {              /*The lights will be used to reflect the appropriate
                     state*/

     case 0:       /*For example, phases a,b and c are all low(0), therefore
                                the lights shall be off.*/
    set_flag(SET_FLAG1,CLR_FLAG);
    set_flag(SET_FLAG2,CLR_FLAG);
    set_flag(SET_FLAG3,CLR_FLAG);
    break;

    case 1:
   set_flag(SET_FLAG1,SET_FLAG);
   set_flag(SET_FLAG2,CLR_FLAG);
   set_flag(SET_FLAG3,CLR_FLAG);
   break;

   case 2:
   set_flag(SET_FLAG1,SET_FLAG);
   set_flag(SET_FLAG2,SET_FLAG);
   set_flag(SET_FLAG3,CLR_FLAG);
   break;

   case 3:
  set_flag(SET_FLAG1,CLR_FLAG);
  set_flag(SET_FLAG2,SET_FLAG);
  set_flag(SET_FLAG3,CLR_FLAG);
  break;

   case 4:
  set_flag(SET_FLAG1,CLR_FLAG);
  set_flag(SET_FLAG2,SET_FLAG);
  set_flag(SET_FLAG3,SET_FLAG);
  break;
```

```
 case 5:
set_flag(SET_FLAG1,CLR_FLAG);
set_flag(SET_FLAG2,CLR_FLAG);
set_flag(SET_FLAG3,SET_FLAG);
break;

 case 6:
set_flag(SET_FLAG1,SET_FLAG);
set_flag(SET_FLAG2,CLR_FLAG);
set_flag(SET_FLAG3,SET_FLAG);
break;

 default:
set_flag(SET_FLAG1,SET_FLAG);
set_flag(SET_FLAG2,SET_FLAG);
set_flag(SET_FLAG3,SET_FLAG);
}

timer_off();   /*Shutting the timer off in preparation for starting a new
                                timer*/
clear_interrupt(SIG_TMZ0);  /*Clearing the interrupt in preparation for
                                    establishing a new interrupt handler routine*/
y1=RESULT1[mann][0];        /*This is the associated time for State 1*/
y1a=y1-270;                 /*"270" represents the cycle time needed to execute
                                    the instructions from the beginning of this
                                   interrupt,  prior to starting the clock*/

timer_set(y1a,y1a);         /*This instruction established the lenth of time
                                    the new timer will run prior to executing the
                                    interrupt handler routine*/

u=RESULT[mann][8];          /*This determines which condition we calculated
                                    during the LAST switching period(-250usec)*/
if(u==4)                    /*If u=4, then it was the torque and flux transient
                                case*/
{interrupt(SIG_TMZ0,timer_isr);     /*This means for the entire time"y1a"
                                        STATE 1 will be displayed*/
 z=0;}                      /*Resets the value of z in preparation for the idle
                                instruction*/

else
        {interrupt(SIG_TMZ0,timer1_isr);    /*If the condition is steady state or
                                                the torque or flux transient case,
                                                STATE 1 is only used until that
                                                time expires(<250 usec) at which
                                                time STATE 2 will be displayed*/
z=1;}
 timer_on();                /*The timer is started and counts down from the value y1*/
}
```

```
void timer1_isr(int t)          /*This is the second interrupt handler routine which
                                    is only used for the steady state, torque or flux
                                    transient conditions*/
{
 u1=RESULT[mann][3];  /*Determining STATE 2 as calculate during the
                                    previous cycle*/
 switch (u1)                  /*Lights will reflect 3 phase input to inverter*/
 {    case 0:
    set_flag(SET_FLAG1,CLR_FLAG);
    set_flag(SET_FLAG2,CLR_FLAG);
    set_flag(SET_FLAG3,CLR_FLAG);
    break;

    case 1:
   set_flag(SET_FLAG1,SET_FLAG);
   set_flag(SET_FLAG2,CLR_FLAG);
   set_flag(SET_FLAG3,CLR_FLAG);
   break;

   case 2:
   set_flag(SET_FLAG1,SET_FLAG);
   set_flag(SET_FLAG2,SET_FLAG);
   set_flag(SET_FLAG3,CLR_FLAG);
   break;

   case 3:
  set_flag(SET_FLAG1,CLR_FLAG);
  set_flag(SET_FLAG2,SET_FLAG);
  set_flag(SET_FLAG3,CLR_FLAG);
  break;

   case 4:
  set_flag(SET_FLAG1,CLR_FLAG);
  set_flag(SET_FLAG2,SET_FLAG);
  set_flag(SET_FLAG3,SET_FLAG);
  break;

   case 5:
  set_flag(SET_FLAG1,CLR_FLAG);
  set_flag(SET_FLAG2,CLR_FLAG);
  set_flag(SET_FLAG3,SET_FLAG);
  break;

   case 6:
  set_flag(SET_FLAG1,SET_FLAG);
  set_flag(SET_FLAG2,CLR_FLAG);
  set_flag(SET_FLAG3,SET_FLAG);
  break;
```

```
            default:
            set_flag(SET_FLAG1,SET_FLAG);
            set_flag(SET_FLAG2,SET_FLAG);
            set_flag(SET_FLAG3,SET_FLAG);
            }

 timer_off();        /*Shutting timer off in preparation for starting new timer*/
 clear_interrupt(SIG_TMZ0);   / *Clearing interrupt in preparation for
                                    setting new timer interrupt*/
 y3=RESULT1[mann][1];        /*This will be used to establish the time that we
                                 remain in STATE 2*/

 y3a=y3-253;         /*"253" represents the cycle time needed to
                           execute the instructions from the beginning
                           of the interrupt, prior to starting the clock.*/
 timer_set(y3a,y3a);    /*Setting up on the timer to count from time "y3a" down to zero
                             where the next interrupt handler will be run*/
 u=RESULT[mann][8];   /* Determining which condition we calculated during
                                 the LAST switching period(<250usec)*/
 if (u!=1)                /*If the condition was either the torque or flux transient
                             case then we go back to the timer_isr interrupt handler routine*/
 {interrupt(SIG_TMZ0,timer_isr);
  z=0;}
  else                  /*The previous condition was steady state, therefore we will use the
                             timer2_isr interrupt handler*/
  {interrupt(SIG_TMZ0,timer2_isr);
  z=2;}            /*Setting up the program to idle  until timer2_isr is reached*/
  timer_on();             /*The timer is turned on and counts from time "y3a"*/
}


void timer2_isr(int t)  /*This interrupt handler is only used for the
                           steady state condition*/
{
 u1=RESULT[mann][5];         /*Determining STATE 3*/
 switch (u1)              /*Lights will reflect phases a-c to the inverter*/
 {    case 0:
    set_flag(SET_FLAG1,CLR_FLAG);
    set_flag(SET_FLAG2,CLR_FLAG);
    set_flag(SET_FLAG3,CLR_FLAG);
    break;

    case 1:
   set_flag(SET_FLAG1,SET_FLAG);
   set_flag(SET_FLAG2,CLR_FLAG);
   set_flag(SET_FLAG3,CLR_FLAG);
   break;
```

```
 case 2:
 set_flag(SET_FLAG1,SET_FLAG);
 set_flag(SET_FLAG2,SET_FLAG);
 set_flag(SET_FLAG3,CLR_FLAG);
 break;

 case 3:
set_flag(SET_FLAG1,CLR_FLAG);
set_flag(SET_FLAG2,SET_FLAG);
set_flag(SET_FLAG3,CLR_FLAG);
break;

 case 4:
set_flag(SET_FLAG1,CLR_FLAG);
set_flag(SET_FLAG2,SET_FLAG);
set_flag(SET_FLAG3,SET_FLAG);
break;

 case 5:
set_flag(SET_FLAG1,CLR_FLAG);
set_flag(SET_FLAG2,CLR_FLAG);
set_flag(SET_FLAG3,SET_FLAG);
break;

 case 6:
set_flag(SET_FLAG1,SET_FLAG);
set_flag(SET_FLAG2,CLR_FLAG);
set_flag(SET_FLAG3,SET_FLAG);
break;

 default:
set_flag(SET_FLAG1,SET_FLAG);
set_flag(SET_FLAG2,SET_FLAG);
set_flag(SET_FLAG3,SET_FLAG);
 }
timer_off();        /*Timer is shut off in preparation for starting the next timing
                      sequence*/
clear_interrupt(SIG_TMZ0);   /*Interrupt is cleared in preparation for
                               setting up the next (timer3_isr) interrupt handler*/
y5=RESULT1[mann][2]; /*The timer will be set based on time"y5a"*/
y5a=y5-229;         /*"229" represents the cycle time needed to
                      execute the instructions from the beginning of
                      the interrupt, prior to starting the clock.*/
timer_set(y5a,y5a);          /*Setting up the timer to count from time "y5a"*/
interrupt(SIG_TMZ0,timer3_isr); /*Next interrupt to be handled will be
                                  timer3_isr*/
z=3;                  /*Program will idle intil interrupt handler timer3_isr is
                        reached*/
timer_on();             /*Timer starts counting from "y5a"*/
}
```

```
void timer3_isr(int t)              /*This interrupt handler is only used for the steady
                                      state condition*/
{
 u1=RESULT[mann][7];  /*Determining STATE 4 from the previous switching period
                        (<250 usec)*/
 switch (u1)                        /*Lights to be set to reflect STATE 4-By definition STATE
                                      4 can only be all phases high(111) or all phases low(000)*/
 {    case 0:
    set_flag(SET_FLAG1,CLR_FLAG);
    set_flag(SET_FLAG2,CLR_FLAG);
    set_flag(SET_FLAG3,CLR_FLAG);
    break;

     default:
    set_flag(SET_FLAG1,SET_FLAG);
    set_flag(SET_FLAG2,SET_FLAG);
    set_flag(SET_FLAG3,SET_FLAG);
 }

timer_off();          /*Shutting timer off in preparation for starting new timer*/
 clear_interrupt(SIG_TMZ0);   /*Clear interrupt in preparation for starting
                                new interrupt handler routine*/
 y7=RESULT1[mann][3];         /*New timer value will be based on time "y7a"*/
 y7a=y7-205;                  /*"205" represents the cycle time neede to execute
                                the instructions from the beginning of the interrupt,
                                prior to starting the clock.*/
 timer_set(y7a,y7a);          /*Setting the timer for time "y7a"*/
 interrupt(SIG_TMZ0,timer_isr);  /*Next interrupt handler to be serviced will
                                   be back to the original timer_isr which begins
                                   the new cycle.*/
 z=0;
 timer_on();                  /*Starts the new timer based on time "y7a"*/
}
```

```
/*.....*/
/*This funtion will determine the times for a given set of conditions
and determine if the conditionis a steady state condition or if it is one
of the three transient conditions*/

void START(VD,VQ,Vdc,Ts,T,Tref,F,Fref,AF,Fqs,Fds,Lsm,DT1,Eq,Ed,p,t1)
   float VD,VQ,Vdc,Ts,T,Tref,F,Fref,AF,Fqs,Fds,Lsm,DT1,Eq,Ed,p,t1;

{
 float A,x,x1,Tk,TK,Tk1,TK1,T0,VD1,VD2,VQ1,VQ2;
        /*A is the space vector angle in radians*/
if (VD !=0.0)
  {A=atan(VQ/VD);VD1=fabs(VD);VD2=VD1*VD1;}
 else
  {A=0.0;VD2=0;}

if (VQ !=0.0)
  {VQ1=fabs(VQ);VQ2=VQ1*VQ1;}
 else
  VQ2=0;

 x1=VD2+VQ2;
 if (x1 !=0)
  x=pow((x1),0.5)/Vdc;   /*[((VD)^2+(VQ)^2)^0.5]/Vdc*/
 else
  x=0;

 Tk=(x*sin((PI/3.0)-A)*Ts)/sin(PI/3.0);       /*Calculated time for State 2(Vk)*/
 if (Tk>0.00 && Tk<0.00001)
 TK=0.00001;
 else TK=Tk;

 Tk1=(x*sin(A)*Ts)/sin(PI/3.0);      /*Calculated time for State 3 (Vk1)*/
 if (Tk1>0.00 && Tk1<0.00001)
 TK1=0.00001;
 else TK1=Tk1;

 T0=Ts-TK-TK1;              /*Remaining time in period Ts to be split between States
                                1 and 4*/
 if (T0>0.00 && T0<0.00001)
 T0=0.00001;

 if (TK>0 && TK1>0 && T0>0)   /*If all three values are positive, then a
                                steady state condition exists*/
    {
     STEAD_STATE(VD,VQ,A,T0,TK,TK1);
     }
 else        /*If any time is negative, then a transient condition exists*/
  TORQUE_TRAN(Vdc,Ts,T,Tref,F,Fref,AF,Fqs,Fds,Lsm,DT1,Eq,Ed,p,t1);

}
```

```
/*.....*/
/*This function will determine the appropriate States for the steady state
condition*/
void STEAD_STATE(VD,VQ,A,T0,TK,TK1)
float VD, VQ, A, T0, TK, TK1;
{
 float SVA,T01;
 int  n, us, STI, ST1, ST2, STF;

/*Matrix to be used to define the states*/

 int Sts[6][4]={{0,1,2,7},
                {7,3,2,0},
                {0,3,4,7},
                {7,4,5,0},
                {0,5,6,7},
                {7,6,1,0}};

/*Determining the Space Vector Angle(SVA)*/
 if(VD==fabs(VD) && VQ==fabs(VQ))
  SVA=A*180.0/PI;
 else if (VQ==fabs(VQ))
  SVA=A*180.0/PI;
 else
  SVA=(A*180.0/PI)+360.0;



 if    (SVA>=0.0 && SVA<=60.0)  n=0;
 else if (SVA>60.0 && SVA<=120.0)  n=1;
 else if (SVA>120.0 && SVA<=180.0) n=2;
 else if (SVA>180.0 && SVA<=240.0) n=3;
 else if (SVA>240.0 && SVA<=300.0) n=4;
 else n=5;

 us=1;

/*To determine the sequence for displaying states, the following instructions
are used. The objective is to ensure that only ONE phase changes state for each
update. For a given period, all three phases will change state once as observed
below*/

 SLVAL=TAB[SLVAL][1];
 if (Sts[n][0]==SLVAL) /*SLVAL is the last valid sequence and reflects
                         the condition of all three phases prior to
                         executing the results of this cycle.*/
```

```
  {
   STI=Sts[n][0];    /* To be used as STATE 1 in the interrupt timer_isr*/
   ST1=Sts[n][1];    /* To be used as STATE 2 in the interrupt timer1_isr*/
   ST2=Sts[n][2];    /* To be used as STATE 3 in the interrupt timer2_isr*/
   STF=Sts[n][3];    /* To be used as STATE 4 in the interrupt timer3_isr*/
   RESULT[mann1][2]=TK; /*This time will be used in timer1_isr to determine timer_set
                          value*/
   RESULT[mann1][4]=TK1;    /*This time will be used in timer2_isr to determine timer_set
                          value*/
  }
 else
  {
   STI=Sts[n][3];    /* To be used as STATE 1 in the interrupt timer_isr*/
   ST1=Sts[n][2];    /* To be used as STATE 2 in the interrupt timer1_isr*/
   ST2=Sts[n][1];    /* To be used as STATE 3 in the interrupt timer2_isr*/
   STF=Sts[n][0];    /* To be used as STATE 4 in the interrupt timer3_isr*/
   RESULT[mann1][2]=TK1;    /*This time will be used in timer1_isr to determine timer_set
                          value*/
   RESULT[mann1][4]=TK;     /* This time will be used in timer2_isr to determine timer_set
                          value*/
  }


T01=T0/2;      /*This time will be used in timer_isr and timer3_isr for the timer_set value*/

 SLVAL=STF;   /*Setting up last valid State for next cycle*/

 RESULT[mann1][0]=T01;
 RESULT[mann1][1]=STI;
 RESULT[mann1][3]=ST1;
 RESULT[mann1][5]=ST2;
 RESULT[mann1][6]=T01;
 RESULT[mann1][7]=STF;
 RESULT[mann1][8]=us;          /*This will identify in the timer interrupt routines
                                 that when us=1, the condition is steady state*/
}

/*.....*/
/*This function initially assumes a torque transient case. However,
if the calculated times are negative, the program jumps to the flux
transient condition*/

void TORQUE_TRAN(Vdc,Ts,T,Tref,F,Fref,AF,Fqs,Fds,Lsm,DT1,Eq,Ed,p,t1)
          float Vdc,Ts,T,Tref,F,Fref,AF,Fqs,Fds,Lsm,DT1,Eq,Ed,p,t1;

 {
  int us,N,K,K1,KT,KT1,ip1,ip2,hilo,hilo1;
  float B,C,FVA,TVKtr,TVKti,TVK1tr,TVK1ti,a1,b1,c1,d1,a2,b2,c2,a3,b3,
  c3,d3,a4,b4,c4,d4,TTK1,TTK2,h1,TTkt,TTk1t,e1,e2,e3,e4,e5,e6;
```

```
/*Determine sign(+ or -) for Table 1a-1c*/
 B=(T-Tref);
 C=(fabs(F)-Fref);

/*Determine flux vector angle*/
 if (F-fabs(F)==0)
       FVA=AF*180.0/PI;
 else if (Fqs==fabs(Fqs))
       FVA=AF*180.0/PI;
 else
       FVA=(AF*180.0/PI)+360.0;


/*Determine value of N for transient condition*/
 if (FVA>270.0 && FVA<=330.0)
       N=0;
 else if (FVA>330 && FVA<=360.0)
       N=1;
 else if (FVA>=0.0 && FVA<=30.0)
       N=1;
 else if (FVA>30.0 && FVA<=90.0)
       N=2;
 else if (FVA>90.0 && FVA<=150.0)
       N=3;
 else if (FVA>150.0 && FVA<=210.0)
       N=4;
 else N=5;



if (B=fabs(B))
       K=N+4;
 else K=N+1;
 K1=K+1;

 ip1=(K-1)/6;          /*This is equivalent to the rem function in MATLAB*/
 ip2=(K1-1)/6;

 KT=((K-1)-ip1*6)+1;
 KT1=((K1-1)-ip2*6)+1;

/*Determine imag and real parts of States VK and VK1 for the torque transient
 case*/
 TVKtr=(2.0/3.0)*Vdc*cos((KT-1)*2.0*PI/3.0); /*real*/
 TVKti=(2.0/3.0)*Vdc*sin((KT-1)*2.0*PI/3.0); /*imag*/

 TVK1tr=(2.0/3.0)*Vdc*cos((KT1-1)*2.0*PI/3.0); /*real*/
 TVK1ti=(2.0/3.0)*Vdc*sin((KT1-1)*2.0*PI/3.0); /*imag*/
```

```
/*Determine switching times for torque transient case*/
a1=TVKtr-TVK1tr;
b1=Ts*TVK1tr+Fds;
c1=TVKti-TVK1ti;
d1=Ts*TVK1ti+Fqs;
  if (a1 !=0)
        {a3=fabs(a1);a4=a3*a3;}
       else a4=0;
  if (c1 !=0)
        {c3=fabs(c1);c4=c3*c3;}
       else c4=0;
a2=(a4+c4);
b2=((2.0*a1*b1)+(2.0*c1*d1));
  if (b1 !=0)
        {b3=fabs(b1);b4=b3*b3;}
       else b4=0;
  if (d1 !=0)
        {d3=fabs(d1);d4=d3*d3;}
       else d4=0;

c2=(b4)+(d4)-(Fref*Fref);
/*Solving quadratic equation*/
e1=b2*b2;
e2=4.0*a2*c2;
e3=2.0*a2;
e4=e1-e2;
if (e4 !=0)
e5=pow(e4,0.5);
else e5=0;

if (e3 !=0)
TTK1=-b2+e5/e3;
else TTK1=-b2;
e6=((b2*b2)-(4.0*a2*c2));
if (e6 !=0 && a2 !=0)
TTK2=-b2 -(pow(e6,0.5)/2.0*a2);
else TTK2=0;

/*Determine smaller value*/
h1=TTK1-TTK2;

 if (TTK1>=0.0 && TTK2>=0.0 && h1<0.0)
       TTkt=TTK1;
 else if (TTK1>=0.0 && TTK2>=0.0 && h1>0.0)
       TTkt=TTK2;
 else if (TTK1>=0.0)
       TTkt=TTK1;
 else if (TTK2>=0.0)
       TTkt=TTK2;
 else TTkt=-1.0;
```

```
if (TTkt>0 && TTkt<0.00001)
  TTkt=0.00001;

 TTk1t=Ts-TTkt;

if (TTk1t>0 && TTk1t<0.00001)
 TTk1t=0.00001;

hilo=0.0;
hilo1=1.0;
us=2;          /*This will be used in the interrupt handler routines
                        to identify that this is a torque transient condition*/


if(TTkt>0 && TTk1t>0)
  { RESULT[mann1][4]=hilo;          /*The values for [mann1][4-6] are place holders*/
       RESULT[mann1][5]=hilo1;
       RESULT[mann1][6]=hilo;
       RESULT[mann1][8]=us;

       if (SLVAL==KT)
        {SLVAL=KT1;
        RESULT[mann1][0]=TTkt;
        RESULT[mann1][1]=KT;
        RESULT[mann1][2]=TTk1t;
        RESULT[mann1][3]=KT1;
        RESULT[mann1][7]=SLVAL;}

       else if (SLVAL==KT1)
        {SLVAL=KT;
        RESULT[mann1][0]=TTk1t;
        RESULT[mann1][1]=KT1;
        RESULT[mann1][2]=TTkt;
        RESULT[mann1][3]=KT;
        RESULT[mann1][7]=SLVAL;}

       else if (TAB[SLVAL][2]==KT)
        {SLVAL=KT1;
        RESULT[mann1][0]=TTkt;
        RESULT[mann1][1]=KT;
        RESULT[mann1][2]=TTk1t;
        RESULT[mann1][3]=KT1;
        RESULT[mann1][7]=SLVAL;}

       else if (TAB[SLVAL][4]==KT
        {SLVAL=KT1;
        RESULT[mann1][0]=TTkt;
        RESULT[mann1][1]=KT;
        RESULT[mann1][2]=TTk1t;
        RESULT[mann1][3]=KT1;
        RESULT[mann1][7]=SLVAL;}
```

```
        else
         {SLVAL=KT;
         RESULT[mann1][0]=TTk1t;
         RESULT[mann1][1]=KT1;
         RESULT[mann1][2]=TTkt;
         RESULT[mann1][3]=KT;
         RESULT[mann1][7]=SLVAL;}

         /*RESULT[mann1][0]=This is the time which will be used in timer_isr
           to set the timer_set value.
            RESULT[mann1][1]= This is STATE 1 to be used in timer_isr.
            RESULT[mann1][2]= This is the time which will be used in timer1_isr
            to set the timer1_set value.
            RESULT[mann1][3]= This is STATE 2 to be used in timer1_isr.
            RESULT[mann1][7]= This sets up the last valid state for the next cycle.*/

         }
          else
             FLUX_TRAN(Vdc,Ts,Fqs,Fds,Lsm,DT1,Eq,Ed,p,B,C,N,t1);
         }



         /*.....*/
         /*This function initially assumes that the transient case is flux transient.
         However, if the calculated times are negative, the case is a torque and flux
         transient case and the program jumps to the torque_flux_transient case*/

void FLUX_TRAN (Vdc,Ts,Fqs,Fds,Lsm,DT1,Eq,Ed,p,B,C,N,t1)
float Vdc,Ts,Fqs,Fds,Lsm,DT1,Eq,Ed,p,B,C,t1;
int N;
{
   int us,Kf,Kf1,FKf, FKf1,ip2,ip3,hilo;
   float FVKfr,FVKfi,FVK1fr,FVK1fi,Tkf,Tk1f,g1;


 hilo=0;
 us=3;       /*This will be used later in the interrupt handlers to
                   identify that the case is a flux transient case*/

 if (C==fabs(C))
   Kf=N+2;
 else Kf=N;
   Kf1=Kf+1;

 ip2=(Kf-1)/6;      /*This is similar to the rem function in MATLAB*/
 ip3=(Kf1-1)/6;
  FKf=((Kf-1)-ip2*6)+1;
  FKf1=((Kf1-1)-ip3*6)+1;
```

```
/*Determining real and imag parts of Vk and VK1 for the flux transient
condition*/
FVKfr=2.0/3.0*Vdc*cos((FKf-1)*2.0*PI/3.0);/*real*/
FVKfi=2.0/3.0*Vdc*sin((FKf-1)*2.0*PI/3.0);/*imag*/
FVK1fr=2.0/3.0*Vdc*cos((FKf1-1)*2.0*PI/3.0);/*real*/
FVK1fi=2.0/3.0*Vdc*sin((FKf1-1)*2.0*PI/3.0);/*imag*/


g1=(Fds)*(FVKfi-FVK1fi)+(Fqs*(FVK1fr-FVKfr));
if (g1 !=0)
Tkf=(4.0*Lsm*DT1/3.0*p)+(Ts*(-Fds)*(Eq+FVK1fi)+(Fqs)*(Ed+FVK1fr))/g1;
else Tkf=0;

Tk1f=Ts-Tkf;


if (Tkf>0 && Tkf<0.00001)
   Tkf=0.00001;
if (Tk1f>0 && Tk1f<0.00001)
    Tk1f=0.00001;

 if (Tkf>0 && Tk1f>0)
   { RESULT[mann1][4]=hilo;   /*RESULT[mann1][4-6] are place holders*/
     RESULT[mann1][5]=hilo;
     RESULT[mann1][6]=hilo;
     RESULT[mann1][8]=us;      /*Identifies the condition(flux tran) for the
                                    interrupt handler*/

 if (SLVAL==FKf)
  {SLVAL=FKf1;
  RESULT[mann1][0]=Tkf;
  RESULT[mann1][1]=FKf;
  RESULT[mann1][2]=Tk1f;
  RESULT[mann1][7]=SLVAL;}

 else if(SLVAL==FKf1)
  {SLVAL=FKf;
  RESULT[mann1][0]=Tk1f;
  RESULT[mann1][1]=FKf1;
  RESULT[mann1][2]=Tkf;
  RESULT[mann1][7]=SLVAL;}

 else if (TAB[SLVAL][2]==FKf)
  {SLVAL=FKf1;
  RESULT[mann1][0]=Tkf;
  RESULT[mann1][1]=FKf;
  RESULT[mann1][2]=Tk1f;
  RESULT[mann1][7]=SLVAL;}
```

```
else if (TAB[SLVAL][4]==FKf)
 {SLVAL=FKf1;
 RESULT[mann1][0]=Tkf;
 RESULT[mann1][1]=FKf;
 RESULT[mann1][2]=Tk1f;
 RESULT[mann1][7]=SLVAL;}

else
 {SLVAL=FKf;
 RESULT[mann1][0]=Tk1f;
 RESULT[mann1][1]=FKf1;
 RESULT[mann1][2]=Tkf;
 RESULT[mann1][7]=SLVAL;}

        /*RESULT[mann1][0]=This is the time which will be used in timer_isr
          to set the timer_set value.
           RESULT[mann1][1]= This is STATE 1 to be used in timer_isr.
           RESULT[mann1][2]= This is the time which will be used in timer1_isr
           to set the timer1_set value.
           RESULT[mann1][3]= This is STATE 2 to be used in timer1_isr.
           RESULT[mann1][7]= This sets up the last valid state for the next cycle.*/

}
        else
   TORQUE_FLUX_TRAN(B,C,N);
        }



        /*.....*/
        /*This function is the remaining condition. The torque_flux
        transient case results in a single state for the entire Ts period*/

        void TORQUE_FLUX_TRAN(B,C,N)
        float B,C;
        int N;
        {int us,Ktf,ip4,KTF,hilo,hilo1;
        hilo=0;
        hilo1=1;
        us=4;              /*This is used in the interrupt handler routines to identify
                                   that the case is a torque_flux transient case*/

        if (B==fabs(B) && C==fabs(C))
          Ktf=N+5;
        else if (B==fabs(B))
          Ktf=N+4;
        else if (C==fabs(C))
          Ktf=N+2;

        else Ktf=N+1;
```

```
ip4=(Ktf-1)/6;
KTF=((Ktf-1)-ip4*6)+1;

SLVAL=KTF;   /*Setting up last valid for next cycle*/

RESULT[mann1][0]=0.000250 ;          /*Ts*/
RESULT[mann1][1]=KTF;          /*This is STATE 1 for timer_isr*/
RESULT[mann1][2]=hilo;        /*RESULT[mann1][2-6] are place holders*/
RESULT[mann1][3]=hilo1;
RESULT[mann1][4]=hilo;
RESULT[mann1][5]=hilo1;
RESULT[mann1][6]=hilo;
RESULT[mann1][7]=SLVAL;
RESULT[mann1][8]=us;
}

/*END OF PROGRAM*/
```

APPENDIX C.2- Architecture file (jb7.ach)

```
.SYSTEM       jb7;
.PROCESSOR =   ADSP21020;

.SEGMENT /PM   /RAM  /BEGIN=0x000000 /END=0x0001FF   seg_rth;
.SEGMENT /PM   /RAM  /BEGIN=0x0200   /END=0x03FF     seg_init;
.SEGMENT /PM   /RAM  /BEGIN=0x0400   /END=0x1FFF     seg_pmco;
.SEGMENT /PM   /RAM  /BEGIN=0x2000   /END=0x7FFF     seg_pmda;

.SEGMENT /DM   /RAM  /BEGIN=0x0000   /END=0x3FFF     seg_dmda;
.SEGMENT /DM   /RAM  /BEGIN=0x4000   /END=0x5FFF/cheap seg_heap;
.SEGMENT /DM   /RAM  /BEGIN=0x6000   /END=0x7FFF      seg_stak;

.Bank /DM0 /wtstates=0/wtmode=internal/BEGIN = 0x00000000;
.Bank /DM1 /wtstates=0/wtmode=internal/BEGIN = 0x20000000;
.Bank /DM2 /wtstates=1/wtmode=internal/BEGIN = 0x40000000;
.Bank /DM3 /wtstates=0/wtmode=internal/BEGIN = 0x80000000;

.Bank /PM0/wtstates=0/wtmode=internal/BEGIN = 0x000000;
.Bank /PM1/wtstates=0/wtmode=internal/BEGIN = 0x008000;


.SEGMENT /PORT /BEGIN=0x40000000 /END=0x40000005 /DM  cports;
.SEGMENT /PORT /BEGIN=0x40000010 /END=0x40000010 /DM  hip_reg0;
.SEGMENT /PORT /BEGIN=0x40000011 /END=0x40000011 /DM  hip_reg1;
.SEGMENT /PORT /BEGIN=0x40000012 /END=0x40000012 /DM  hip_reg2;
.SEGMENT /PORT /BEGIN=0x40000013 /END=0x40000013 /DM  hip_reg3;
.SEGMENT /PORT /BEGIN=0x40000014 /END=0x40000014 /DM  hip_reg4;
.SEGMENT /PORT /BEGIN=0x40000015 /END=0x40000015 /DM  hip_reg5;
.SEGMENT /PORT /BEGIN=0x40000016 /END=0x40000021 /DM  hip_regs;

.SEGMENT /PORT /BEGIN=0x40000040 /END=0x40000058 /DM  aports;
.SEGMENT /PORT /BEGIN=0x40000060 /END=0x40000060 /DM  adc_a;
.SEGMENT /PORT /BEGIN=0x40000068 /END=0x40000068 /DM  adc_b;
.SEGMENT /PORT /BEGIN=0x40000070 /END=0x40000070 /DM  dac_a;
.SEGMENT /PORT /BEGIN=0x40000078 /END=0x40000078 /DM  dac_b;

.ENDSYS;
```

APPENDIX C.3- Map file (julie7.map)

Architecture Description: jb7

Segment	Start	End	Length	Memory Type	Attribute
seg_rth	000000	0001ff	512	Program Memory	RAM
seg_dmda	00000000	00003fff	16384	Data Memory	RAM
seg_init	000200	0003ff	512	Program Memory	RAM
seg_pmco	000400	001fff	7168	Program Memory	RAM
seg_pmda	002000	007fff	24576	Program Memory	RAM
seg_heap	00004000	00005fff	8192	Data Memory	RAM
seg_stak	00006000	00007fff	8192	Data Memory	RAM
cports	40000000	40000005	6	Data Memory	PORT
hip_reg0	40000010	40000010	1	Data Memory	PORT
hip_reg1	40000011	40000011	1	Data Memory	PORT
hip_reg2	40000012	40000012	1	Data Memory	PORT
hip_reg3	40000013	40000013	1	Data Memory	PORT
hip_reg4	40000014	40000014	1	Data Memory	PORT
hip_reg5	40000015	40000015	1	Data Memory	PORT
hip_regs	40000016	40000021	12	Data Memory	PORT
aports	40000040	40000058	25	Data Memory	PORT
adc_a	40000060	40000060	1	Data Memory	PORT
adc_b	40000068	40000068	1	Data Memory	PORT
dac_a	40000070	40000070	1	Data Memory	PORT
dac_b	40000078	40000078	1	Data Memory	PORT

Memory Usage (Actual):

Segment	Start	End	Length	Memory Type	Attribute
seg_rth	000000	0000ff	256	Program Memory	RAM
seg_init	000200	00020e	15	Program Memory	RAM
seg_pmco	000400	001790	5009	Program Memory	RAM
seg_pmda	******	******	0	Program Memory	RAM
seg_dmda	00000000	0000021e	543	Data Memory	RAM
seg_heap	********	********	0	Data Memory	RAM
seg_stak	********	********	0	Data Memory	RAM
cports	********	********	0	Data Memory	PORT
hip_reg0	********	********	0	Data Memory	PORT
hip_reg1	********	********	0	Data Memory	PORT
hip_reg2	********	********	0	Data Memory	PORT
hip_reg3	********	********	0	Data Memory	PORT
hip_reg4	********	********	0	Data Memory	PORT
hip_reg5	********	********	0	Data Memory	PORT
hip_regs	********	********	0	Data Memory	PORT
aports	********	********	0	Data Memory	PORT
adc_a	********	********	0	Data Memory	PORT
adc_b	********	********	0	Data Memory	PORT
dac_a	********	********	0	Data Memory	PORT
dac_b	********	********	0	Data Memory	PORT

Memory Usage Summaries:

Memory Type	Attribute	Total
Program Memory	ROM	0
Program Memory	RAM	5280
Program Memory	PORT	0
Data Memory	ROM	0
Data Memory	RAM	543
Data Memory	PORT	0

Cross Reference for file -> C:\ADI_DSP\21k\lib\020_hdr.obj:

Symbol	Type	Address	Class
___lib_CB15I	PM	000060	static
___lib_CB7I	PM	000058	static
___lib_FIXI	PM	000078	static
___lib_FLTII	PM	000090	static
___lib_FLTOI	PM	000080	static
___lib_FLTUI	PM	000088	static
___lib_IRQ0I	PM	000040	static
___lib_IRQ1I	PM	000038	static
___lib_IRQ2I	PM	000030	static
___lib_IRQ3I	PM	000028	static
___lib_RSTI	PM	000008	static
___lib_SOVFI	PM	000018	static
___lib_TMZI	PM	000070	static
___lib_TMZOI	PM	000020	static
___lib_USI15I	PM	0000e8	static
___lib_USR0I	PM	0000c0	static
___lib_USR1I	PM	0000c8	static
___lib_USR2I	PM	0000d0	static
___lib_USR3I	PM	0000d8	static
___lib_USR4I	PM	0000e0	static
___lib_USR6I	PM	0000f0	static
___lib_USR7I	PM	0000f8	static
___lib_int_table	DM	000000e9	external
(C:\ADI_DSP\21k\lib\libc.a:seg_dmda)			
___lib_prog_term	PM	00000e	global
___lib_setup_environment	PM	001287	external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)			
___lib_setup_hardware	PM	00127b	external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)			
___lib_setup_processor	PM	001224	external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)			

_main PM 000400 external (julie7.o:seg_pmco)

Cross Reference for file -> julie7.o:

Symbol	Type	Address	Class
_FLUX_TRAN	PM	000f85	global
_L1	PM	000576	static
_L10	PM	0004a2	static
_L11	PM	000548	static
_L115	PM	0008cf	static
_L116	PM	0008ac	static
_L117	PM	0007ef	static
_L118	PM	000807	static
_L119	PM	00081f	static
_L12	PM	0004fa	static
_L120	PM	000837	static
_L121	PM	00084f	static
_L122	PM	000867	static
_L123	PM	00087f	static
_L124	PM	000897	static
_L125	DM	000000af	static
_L13	PM	0004fa	static
_L14	PM	00050f	static
_L15	PM	000548	static
_L152	PM	00093c	static
_L153	PM	000919	static
_L154	PM	0008ec	static
_L155	PM	000904	static
_L165	PM	000a54	static
_L166	PM	00096a	static
_L167	PM	00096e	static
_L168	PM	00097d	static
_L169	PM	00097f	static
_L17	PM	000553	static
_L170	PM	0009a1	static
_L171	PM	0009a3	static
_L172	PM	0009d5	static
_L173	PM	0009d7	static
_L174	PM	000a06	static
_L175	PM	000a08	static
_L176	PM	000a1b	static
_L177	PM	000a33	static
_L178	PM	000a54	static

_L18	PM	00055e	static
_L187	PM	000bd7	static
_L188	PM	000a8e	static
_L189	PM	000abc	static
_L19	PM	000569	static
_L190	PM	000aa9	static
_L191	PM	000abc	static
_L192	PM	000acc	static
_L193	PM	000b12	static
_L194	PM	000add	static
_L195	PM	000b12	static
_L196	PM	000aee	static
_L197	PM	000b12	static
_L198	PM	000aff	static
_L199	PM	000b12	static
_L2	PM	000443	static
_L20	PM	000573	static
_L200	PM	000b10	static
_L201	PM	000b12	static
_L202	PM	000b5d	static
_L203	PM	000b95	static
_L205	PM	000f76	static
_L206	PM	000c12	static
_L207	PM	000c40	static
_L208	PM	000c2d	static
_L209	PM	000c40	static
_L210	PM	000c51	static
_L211	PM	000ca7	static
_L212	PM	000c62	static
_L213	PM	000ca7	static
_L214	PM	000c72	static
_L215	PM	000ca7	static
_L216	PM	000c83	static
_L217	PM	000ca7	static
_L218	PM	000c94	static
_L219	PM	000ca7	static
_L220	PM	000ca5	static
_L221	PM	000ca7	static
_L222	PM	000cb7	static
_L223	PM	000cba	static
_L224	PM	000d86	static
_L225	PM	000d88	static
_L226	PM	000d97	static
_L227	PM	000d99	static
_L228	PM	000db6	static
_L229	PM	000db8	static
_L230	PM	000dc7	static

_L231	PM	000dc9	static
_L232	PM	000df2	static
_L233	PM	000df4	static
_L234	PM	000e1a	static
_L235	PM	000e1d	static
_L236	PM	000e44	static
_L237	PM	000e46	static
_L238	PM	000e5e	static
_L239	PM	000e88	static
_L240	PM	000e72	static
_L241	PM	000e88	static
_L242	PM	000e7c	static
_L243	PM	000e88	static
_L244	PM	000e86	static
_L245	PM	000e88	static
_L246	PM	000e95	static
_L247	PM	000ea6	static
_L248	PM	000f59	static
_L249	PM	000f24	static
_L250	PM	000f56	static
_L251	PM	000f76	static
_L261	PM	001165	static
_L262	PM	000fa6	static
_L263	PM	000fa8	static
_L264	PM	001090	static
_L265	PM	001092	static
_L266	PM	0010a3	static
_L267	PM	0010b0	static
_L268	PM	00115d	static
_L269	PM	001128	static
_L270	PM	00115a	static
_L271	PM	001165	static
_L279	PM	00121f	static
_L280	PM	001194	static
_L281	PM	0011b5	static
_L282	PM	0011a3	static
_L283	PM	0011b5	static
_L284	PM	0011b2	static
_L285	PM	0011b5	static
_L3	PM	000576	static
_L35	PM	0006a3	static
_L36	PM	000660	static
_L37	PM	0005a3	static
_L38	PM	0005bb	static

```
        _L39                    PM    0005d3 static
        _L4                     PM    000573 static
        _L40                    PM    0005eb static
        _L41                    PM    000603 static
        _L42                    PM    00061b static
        _L43                    PM    000633 static
        _L44                    PM    00064b static
        _L45                    DM    000000a1 static
        _L46                    PM    000697 static
        _L47                    PM    0006a0 static
        _L5                     PM    000479 static
        _L6                     PM    00047b static
        _L7                     PM    00048a static
        _L75                    PM    0007c9 static
        _L76                    PM    000786 static
        _L77                    PM    0006c9 static
        _L78                    PM    0006e1 static
        _L79                    PM    0006f9 static
        _L8                     PM    00048c static
        _L80                    PM    000711 static
        _L81                    PM    000729 static
        _L82                    PM    000741 static
        _L83                    PM    000759 static
        _L84                    PM    000771 static
        _L85                    DM    000000a8 static
        _L86                    PM    0007bd static
        _L87                    PM    0007c6 static
        _L9                     PM    0004a0 static
        _LC0                    DM    000000b6 static
        _RESULT                 DM    0000001f global
        _RESULT1                DM    00000079 global
        _SLVAL                  DM    00000006 global
        _START                  PM    000940 global
        _STEAD_STATE            PM    000a5a global
        _TAB                    DM    00000007 global
        _TORQUE_FLUX_TRAN       PM    001172 global
        _TORQUE_TRAN            PM    000bdd global
        ___clear_interrupt020   PM    0015a6 external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
        ___divsi3               PM    0012e0 external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
        ___interrupt020         PM    001515 external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
        ___memcpyDD             PM    001399 external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
        _atanf                  PM    001492 external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
```

```
        _cosf                   PM    001463 external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
                _exit                   PM    00137e external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
        _idle                   PM    0015bd external (C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
        _main                    PM    000400 global
        _mann                    DM    000000e6 global
        _mann1                   DM    000000e7 global
        _powf                   PM    0013d3 external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
        _set_flag               PM    001558 external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
        _sinf                   PM    001466 external
 (C:\ADI_DSP\21k\lib\libc.a:seg_pmco)


        _timer1_isr              PM    0006a8 global
        _timer2_isr              PM    0007ce global
        _timer3_isr              PM    0008d4 global
        _timer_isr               PM    00057a global
        _u                      DM   00000000 global
        _u1                     DM   00000001 global
        _y1                     DM   00000002 global
        _y3                     DM   00000003 global
        _y5                     DM   00000004 global
        _y7                     DM   00000005 global
        _z                      DM   000000e8 global


        Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

        Symbol                   Type Address  Class

        ___inits                PM    000208 external (C Support Object:seg_init)
        ___lib_clear_cache       PM    00122b static
        ___lib_clear_irptl      PM    001228 static
        ___lib_dmbank1           PM    000202 external (C Support Object:seg_init)
        ___lib_setup_hardware     PM    00127b global
        ___lib_setup_memory      PM    00124e static
        ___lib_setup_modes       PM    00124a static
        ___lib_setup_processor    PM    001224 global
        ___lib_setup_registers   PM    00122f static
        blk_inits_dm             PM    00126f static
        blk_inits_pm             PM    001279 static
        end_setup               PM    001246 static
        finish_inits            PM    00127a static
        init_blk_dm              PM    00126e static
        init_blk_pm              PM    001278 static
```

```
init_dm                 PM    001266 static
init_pm                 PM    001270 static
zero_blk_dm               PM    00125b static
zero_blk_pm               PM    001264 static
zero_dm                 PM    001254 static
zero_inits_dm             PM    00125c static
zero_inits_pm             PM    001265 static
zero_pm                 PM    00125d static


Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol                  Type Address  Class

___lib_exit_table           DM   000001c8 external
(C:\ADI_DSP\21k\lib\libc.a:seg_dmda)
___lib_heap_space             PM    000209 external (C Support Object:seg_init)
___lib_int_table            DM   000000e9 external
(C:\ADI_DSP\21k\lib\libc.a:seg_dmda)
___lib_rand_seed            DM   000001c7 external
(C:\ADI_DSP\21k\lib\libc.a:seg_dmda)
___lib_setup_args           PM    0014fc external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)
___lib_setup_environment        PM    001287 global
___lib_setup_errno            PM    0012a6 static
___lib_setup_exit           PM    001293 static
___lib_setup_heaps            PM    0012b5 static
___lib_setup_ints           PM    001298 static
___lib_setup_rand             PM    00128f static
___lib_setup_stacks           PM    0012aa static
___lib_stack_length           PM    000201 external (C Support Object:seg_init)
___lib_stack_space            PM    000200 external (C Support Object:seg_init)
___lib_sure_rts             PM    0012dc global
_errno                  DM   0000021e external
(C:\ADI_DSP\21k\lib\libc.a:seg_dmda)
default_int_table           PM    0012a4 static
loop_thru_heaps             PM    0012b7 static
pm_heap                 PM    0012cf static
zero_exit_table             PM    001296 static
```

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___lib_int_table	DM	000000e9	global

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___divsi3	PM	0012e0	global
___dtoi	PM	00135d	global
___dtoui	PM	001366	static
___fixdfsi	PM	00135d	global
___fixunsdfsi	PM	001366	global
___float_divide (C:\ADI_DSP\21k\lib\libc.a:seg_pmco)	PM	0015c2	external
___ldtoi	PM	00135d	global
___ldtoui	PM	001366	static
___lib_dtoi (C:\ADI_DSP\21k\lib\libc.a:seg_pmco)	PM	00166b	external
___lib_dtoui (C:\ADI_DSP\21k\lib\libc.a:seg_pmco)	PM	001678	external
___modsi3	PM	001321	global
___sdiv	PM	0012e0	global
___sldiv	PM	0012e0	global
___slmod	PM	001321	global
___smod	PM	001321	global
___truncdfsi2	PM	00135d	global
___udiv	PM	0012fd	global
___udivsi3	PM	0012fd	global
___uldiv	PM	0012fd	global
___ulmod	PM	001339	global
___umod	PM	001339	global
___umodsi3	PM	001339	global
fix_neg1	PM	0012ee	static
fix_neg2	PM	00132d	static
fixed_up1	PM	0012f2	static
fixed_up2	PM	001331	static
get_outta_here	PM	00131b	static
restore_state	PM	0012f7	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___lib_rand_seed	DM	000001c7	global
_rand	PM	001373	global
_srand	PM	00136f	global
restore_state	PM	00137b	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___lib_exit_table	DM	000001c8	global
___lib_prog_term	PM	00000e	external
(C:\ADI_DSP\21k\lib\020_hdr.obj:seg_rth)			
_exit	PM	00137e	global
execute_routine	PM	001389	static
find_end	PM	001387	static
finished_table	PM	001396	static
found_end	PM	001388	static
label_1	PM	001391	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___memcpyDD	PM	001399	global
___memcpyDP	PM	00139c	global
___memcpyPD	PM	00139f	global
___memcpyPP	PM	0013a2	global
_memcpy	PM	001399	global
copy_core	PM	0013a5	static
dm_dm	PM	0013bf	static
dm_pm	PM	0013c5	static
dm_to_dm_copy	PM	0013bb	static
dm_to_pm_copy	PM	0013c1	static
pm_dm	PM	0013cb	static
pm_pm	PM	0013d1	static
pm_to_dm_copy	PM	0013c7	static
pm_to_pm_copy	PM	0013cd	static
restore_state	PM	0013b4	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___float_divide	PM	0015c2	external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)			
_errno	DM	0000021e	external
(C:\ADI_DSP\21k\lib\libc.a:seg_dmda)			
_powf	PM	0013d3	global
a1_values	DM	000001e8	static
a2_values	DM	000001fa	static
check_y	PM	00145d	static
determine_R	PM	001403	static
determine_Z	PM	00143a	static
determine_g	PM	0013e3	static
determine_m	PM	0013e0	static
determine_mp	PM	001431	static
determine_p	PM	0013e4	static
determine_u1	PM	00140e	static
determine_u2	PM	001409	static
determine_w	PM	001413	static
determine_z	PM	0013f6	static
flow_to_a	PM	00142c	static
overflow	PM	00145f	static
power_array	DM	00000203	static
restore_state	PM	001448	static
underflow	PM	001455	static
x_neg_error	PM	001459	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___sin_path	PM	001473	static
_cosf	PM	001463	global
_sinf	PM	001466	global
compute_R	PM	001483	static
compute_f	PM	00147d	static
compute_modulo	PM	001477	static
compute_poly	PM	001486	static
compute_sign	PM	00148a	static
reg_save	PM	001468	static
restore_state	PM	00148b	static
sine_data	DM	0000020d	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___float_divide (C:\ADI_DSP\21k\lib\libc.a:seg_pmco)	PM	0015c2	external
_atan2f	PM	001495	global
_atanf	PM	001492	global
adjust_by_AN	PM	0014e6	static
do_divide	PM	00149c	static
do_division	PM	0014ab	static
get_f	PM	0014af	static
input_error	PM	00149f	static
overflow	PM	0014f6	static
overflow_tst	PM	0014a2	static
restore_state	PM	0014ef	static
save_stack	PM	001497	static
tst_N	PM	0014d8	static
tst_f	PM	0014b8	static
tst_for_eps	PM	0014c6	static
tst_sign_x	PM	0014ec	static
tst_sign_y	PM	0014e9	static
underflow	PM	0014f9	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___lib_arg1	DM	00000219	static
___lib_argv	DM	00000218	static
___lib_setup_args	PM	0014fc	global
___lib_setup_argv	PM	001502	static
clear_argv	PM	001506	static
init_arg1	PM	001509	static
init_argv	PM	001507	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___interrupt020	PM	001515	global
___interrupt020f	PM	00150f	global
___interrupt020s	PM	001512	global
___lib_faster_int_cntrl	PM	00173a	external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)			
___lib_int_cntrl	PM	0016ba	external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)			
___lib_int_table	DM	000000e9	external
(C:\ADI_DSP\21k\lib\libc.a:seg_dmda)			
___lib_super_int_cntrl	PM	001774	external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)			
___lib_sure_rts	PM	0012dc	external
(C:\ADI_DSP\21k\lib\libc.a:seg_pmco)			
continue	PM	001517	static
default_int	PM	001540	static
error_return	PM	001555	static
ignore_int	PM	001537	static
real_func	PM	00152b	static
restore_state	PM	001550	static
return_address	PM	00154c	static
stop_ints_def	PM	001543	static
stop_ints_ignor	PM	00153a	static
stop_ints_real	PM	00152e	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
_set_flag	PM	001558	global
error_return	PM	001560	static
flg0	PM	001562	static
flg1	PM	001572	static
flg2	PM	001582	static
flg3	PM	001592	static
restore_state	PM	0015a2	static
tst_clr_flg0	PM	001567	static
tst_clr_flg1	PM	001577	static
tst_clr_flg2	PM	001587	static
tst_clr_flg3	PM	001597	static
tst_flg0	PM	00156f	static
tst_flg1	PM	00157f	static
tst_flg2	PM	00158f	static
tst_flg3	PM	00159f	static
tst_tgl_flg0	PM	00156b	static
tst_tgl_flg1	PM	00157b	static
tst_tgl_flg2	PM	00158b	static
tst_tgl_flg3	PM	00159b	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___clear_interrupt020	PM	0015a6	global
error_return	PM	0015ba	static
restore_state	PM	0015b7	static
stop_ints_real	PM	0015b1	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
_idle	PM	0015bd	global
idle_loop	PM	0015be	static
restore_state	PM	0015bf	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___float_divide	PM	0015c2	global
___lib_dadd	PM	001625	global
___lib_dmult	PM	0015e7	global
___lib_dsub	PM	001624	global
___lib_dtof	PM	0015ca	global
___lib_dtoi	PM	00166b	global
___lib_dtoui	PM	001678	global
___lib_dtox	PM	001699	global
___lib_ftod	PM	0015d6	global
___lib_ftold	PM	0015d6	global
___lib_itod	PM	001687	global
___lib_itold	PM	001687	global
___lib_ldtof	PM	0015ca	global
___lib_ldtoi	PM	00166b	global
___lib_ldtoui	PM	001678	global
___lib_ldtox	PM	001699	global
___lib_xtod	PM	0016a7	global
___lib_xtold	PM	0016a7	global
add_core	PM	001641	static
check_MSW	PM	00164a	static
extract_y_mant	PM	00163a	static
invert_result	PM	001646	static
invert_x	PM	001637	static
invert_y	PM	00163e	static
large_prod	PM	001620	static
normalize	PM	001645	static
place_exp_sign	PM	00165a	static
shift_x_op	PM	00162c	static
small_result	PM	00165d	static
swap_ops	PM	001629	static
zero_result	PM	001621	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
_errno	DM	0000021e	global

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___lib_DO_NOT_EMU_STEP1	PM	001714	static
___lib_DO_NOT_EMU_STEP2	PM	001717	static
___lib_DO_NOT_EMU_STEP3	PM	00171a	static
___lib_DO_NOT_EMU_STEP4	PM	00171d	static
___lib_DO_NOT_EMU_STEP5	PM	001720	static
___lib_DO_NOT_EMU_STEP6	PM	001723	static
___lib_int_cntrl	PM	0016ba	global
empty_stack	PM	001726	static
int_cont	PM	001706	static
stop_ints	PM	00170d	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___lib_faster_int_cntrl	PM	00173a	global
int_cont	PM	001757	static

Cross Reference for file -> C:\ADI_DSP\21k\lib\libc.a:

Symbol	Type	Address	Class
___lib_super_int_cntrl	PM	001774	global
int_cont	PM	001788	static

Cross Reference for file -> C Support Objects:

Symbol	Type	Address	Class
___inits	PM	000208	global
___lib_dmbank1	PM	000202	global
___lib_dmbank2	PM	000203	global
___lib_dmbank3	PM	000204	global
___lib_dmwait	PM	000205	global
___lib_heap_space	PM	000209	global
___lib_pmbank1	PM	000206	global
___lib_pmwait	PM	000207	global
___lib_stack_length	PM	000201	global
___lib_stack_space	PM	000200	global