A Microprocessor Control Scheme for Switched Reluctance Motor Drives

by

Ameesh R. Oza

Thesis submitted to the Faculty of the

Virginia Polytechnic Institute and State University

in partial fulfillment of the requirements for the degree of

Master of Science

in

Electrical Engineering

APPROVED:

Dr. Krishnan Ramu, Chairman

Dr. Charles E. Nunnally

Dr. Saifur Rahman

August, 1987

Blacksburg, Virginia

A Microprocessor Control Scheme for Switched Reluctance Motor Drives

by

Ameesh R. Oza

Dr. Krishnan Ramu, Chairman

Electrical Engineering

(ABSTRACT)

A microprocessor control scheme for variable speed switched reluctance motor(SRM) drives is discussed. A particular implementation derived from first principles of the SRM is presented. The Intel 8088 microprocessor is used for the design implementation. It is shown that given the control requirements of the SRM like firing different phases according to rotor position and phase currents, a microprocessor controller is a good choice. The controller is economical since it uses standard TTL chips. The slow response at low speeds is also discussed. Experimental results performed on a static inductive load using a simulated position feedback are presented, showing how the current control available at lower speeds is lost at higher speeds, due to limited dc bus voltage. A listing of the controller software with adequate comments and the circuit diagrams are appended.

Acknowledgements

I would like to acknowledge the constant guidance, support and encouragement of Dr. Ramu throughout this project and my coursework. I have learned a lot from my association with him. I thank Dr. Nunnally for loaning equipment and giving useful tips. It has been a pleasure knowing him and working with him. I thank Dr. Rahman for agreeing to be on the committee.

If ever there was a collection of nice people in one place, it is the EE Shop at Virginia Tech. This project would not have seen the light of day had it not been for the constant support of the EE Shop personnel. Whether it was a hardware problem or an urgent need for parts and equipment they always came through. I am particularly grateful to for his support and advice on the use of Laboratory equipment. I learned a lot about hardware and electronics from talking to who was always willing to discuss a problem.

I also am grateful to all my colleagues in the Motor Drives group at Virginia Tech. A lot of ideas used in the project were given by . is a true gentleman and guide when you need one. Last but not the least I would like to thank my parents and and sister for keeping my spirits up. If it were not for them I would not be here. To them I dedicate this thesis.

Table of Contents

List of Illustrations

1.0 Introduction

1.1 Historical Background

The past two decades have seen a revival of interest in a class of machines called the Switched Reluctance Motor(SRM). The SRM of today is a modern version of the 'Electromagnetic Engine' of the 1830's. The first American patent for such a machine was given in 1838. The early machines faced a host of problems. Structural problems and large pulsating forces were observed when running these machines. The un-laminated core caused high losses. Another problem was that position sensing and power switching was done with mechanical cam-driven switches thus limiting the speed of the machine. The arrival of the d.c. machines relegated the SRM to history, or so it seemed.

The 1960's saw the use of high-power semiconductor switches. These were fast and reliable. Angular position could also be measured using position encoders now. These advances rekindled an interest in the SRM. An improved understanding of the

magnetics led to the machines being driven to saturation to get better efficiencies[3]. Inspite of the progress, the main drawback of the machine is a pulsating torque which precludes it from high-performance applications. Efforts are on to design motors, converters and controllers to give a smooth torque output.

1.2 Literature Survey

The literature available on the SRM may be broadly classified into three segments:

1. Motors.
   a. Design -

      The theory and potentials of SRMs are explored in detail in[2]. The authors describe the basic principles and obtain performance characteristics of the motor. Optimal design decisions and the nonlinearities involved are also discussed. A simple design procedure for a SRM is described in [11]. The authors use the required output and magnetic characteristics of the material to methodically derive the design dimensions of the SRM. A verification method for the design is also discussed.

   b. Analysis -

      The SRM characteristics are highly nonlinear, as will be discussed later. Finite element analyses of SRMs, to derive the inductance profiles of the machine under various conditions, is discussed in[5]. A paper by the same

authors[16] also compares two different kinds of SRMs using finite element analysis.

2. Converters.

Though the converter required to drive an SRM is conceptually simple, various configurations are possible depending on current requirements, switching speeds, kinds of windings, etc. A regenerative "C-Dump" converter is described in[9]. The trapped magnetic energy is dumped into a capacitor and returned to the dc source. A system approach is taken in[8]. The authors design a thyristor inverter, the analysis of which is used to provide guidance for motor design. This is possible because the SRM is not an established motor. Thus, the motor and converter are designed together to get optimum performance. The paper also compares experimental and theoretical results. A third paper[14] considers various converter circuit options for single winding and bifiliar-wound motors. Component ratings are calculated for the various circuits. Experimental results for a traction-drive are also included.

3. Control.

A microprocessor based controller is discussed in[6]. It uses a thyristor based converter circuit. This control is inherently more complicated because of the need to commutate thyristors. Another approach using a microcomputer control of a transistor based converter for a SRM is discussed in[7]. This controller is for a machine with four phases. A speed control loop and torque control loop are implemented. A paper on sensorless control[13], discusses a very simple control

circuit without using position sensors. The machine efficiencies, using such a control were found to be extremely low and only suitable for low-end aplications.

1.3 Objective

The SRMs are inherently variable-speed drives, since they need a converter for its operation. The control strategy may be implemented using analog or digital components. A digital control scheme enables the use of a microprocessor. The objective of this project is to design, build and test a microprocessor controller on a prototype drive system. This would allow data to be collected for various parameter and design changes via software, for understanding the motor, converter and controller and their integrated performance.

1.4 Outline of the thesis

The second chapter describes the switched reluctance motor and relevant performance equations are derived. Chapter three considers the design of the controller and its requirements. Chapters four and five describe in detail the prototype hardware and software, respectively. Some experimental results are presented in chapter six.

The conclusions and recommendations the future study are given in chapter seven. Software listing and system hardware circuit diagrams are given in the appendices.

2.0 The Switched Reluctance Motor

2.1 Introduction

The switched reluctance motor(SRM) may be thought of as a brushless d.c. machine without field excitation. The machine is 'doubly salient' i.e. it has salient poles on the stator and rotor (figure 1). The SRM has no windings on the rotor and the windings on the stator are simple. Thus the motor is economical to manufacture.

The motor has a wide speed variation at constant power output, high torque capability at low speeds, four quadrant operation and simple control. The machine is rugged, reliable, and has a comparable power density to other machines[1].

Additionally, the windings need unidirectional current for operations in any four quadrants. Thus the power circuit is inexpensive and reliable. Since the switches are in series with the phase windings, the shoot through faults are limited. SCRs may be replaced by power transistors making control even simpler.

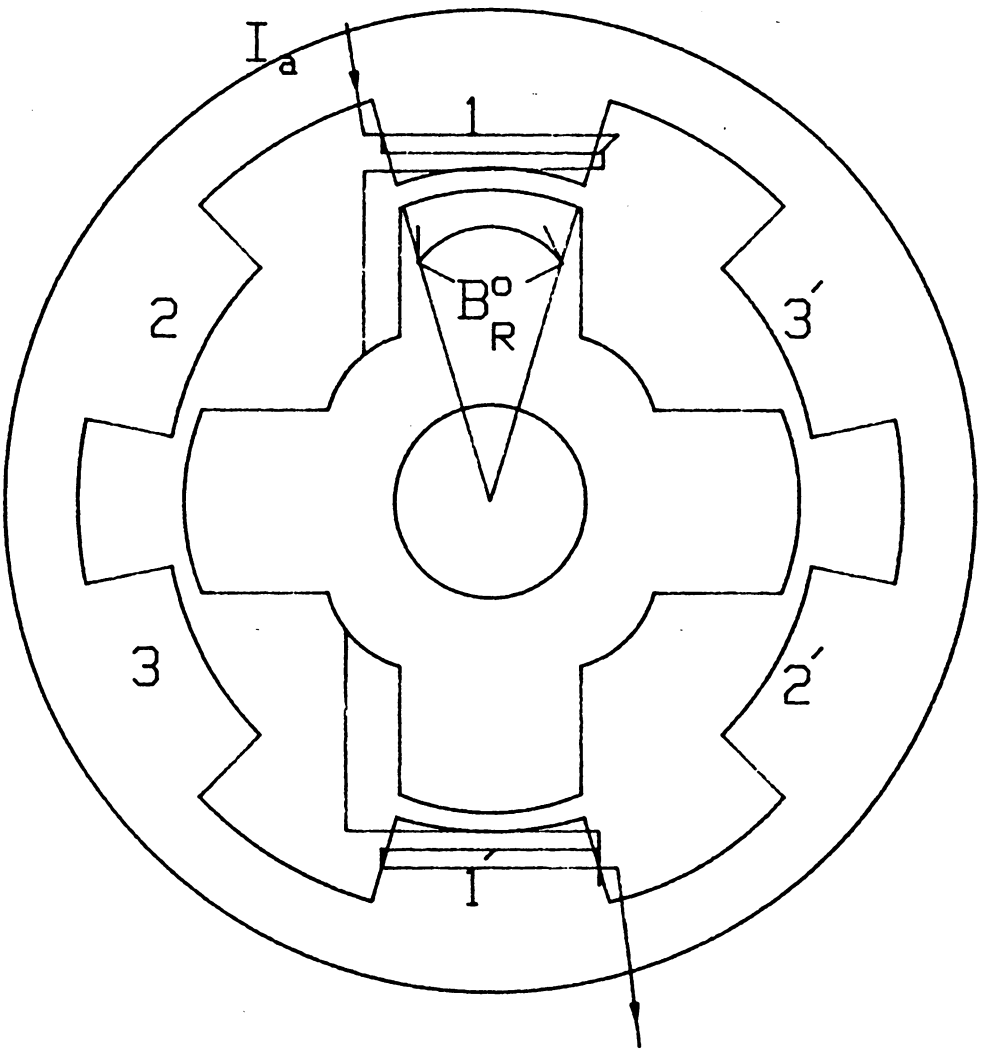


Figure 1. CS of a 6/4 SRM

2.2 *Torque and power*

A magnetic circuit tends toward a configration of minimum reluctance when excited. In a SRM when one stator phase is excited, it magnetises the rotor pole which tends to move under the stator pole where reluctance is minimum. Figure 2 shows the variation of inductance with position, for one phase. The profiles are similar for other phases but shifted by the difference between the stator and rotor pole pitches. Thus by exciting the stator phases in a proper sequence and at the right instances, a continuous torque is developed.

The profile shown in figure 2 is an idealized one. The actual inductance is a function of the position and current. Thus it is non-linear. This nonlinearity is due to the fact that the flux saturates. A sample plot of flux linkages versus current is shown in figure 3. For the purpose of a conceptual treatment a linear profile is assumed. The torque equation is derived as follows.

If the stator winding is modelled as a resistance R and an inductor L in series, the voltage equation may be written as,

$$v = Ri + \frac{d\psi}{dt} \tag{2.1}$$

where,

v = applied voltage
ψ = flux linkages
i = instantaneous current

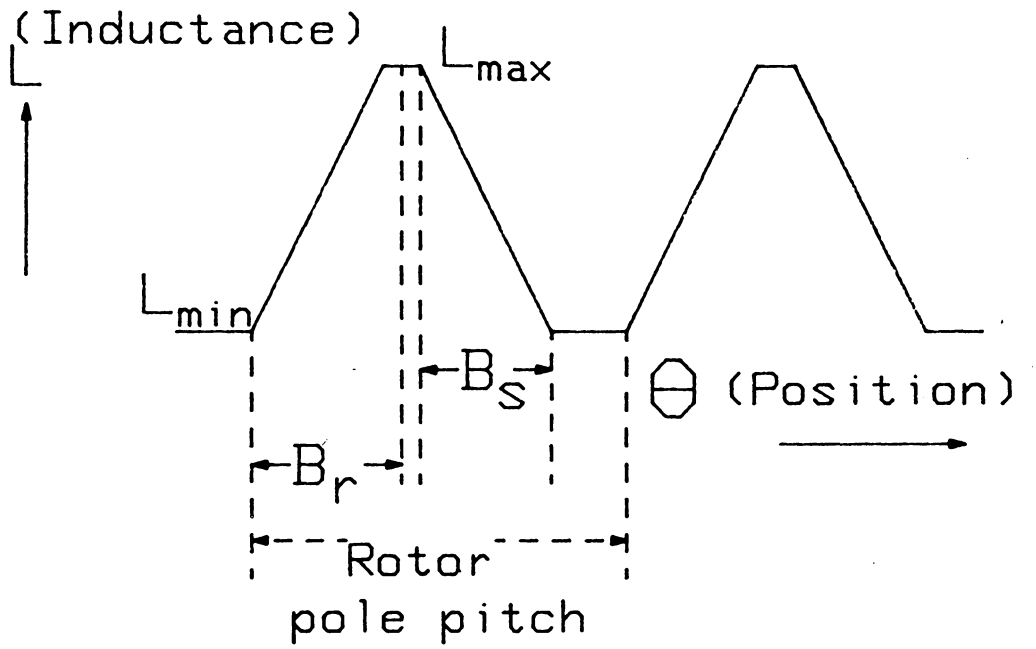


Figure 2. Inductance Profile for One Phase

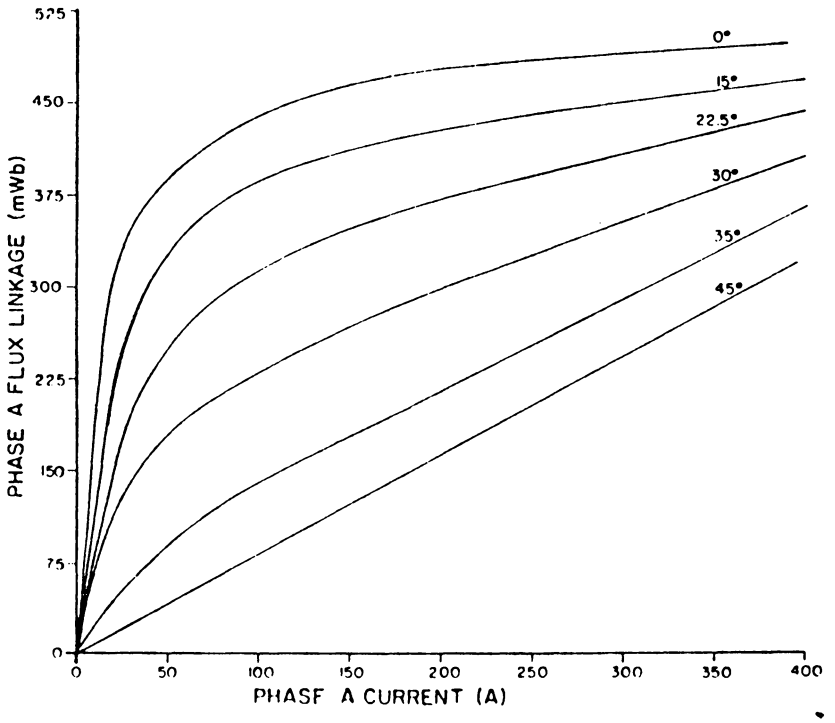


Figure 3. Flux Linkages v/s Current for SRM

Assuming magnetic linearity and negligible resistance,

$$v = \frac{d(Li)}{dt} = L\frac{di}{dt} + i\frac{dL}{dt} \tag{2.2}$$

which may be rewritten as,

$$v = L\frac{di}{dt} + \frac{i\ dL}{2\ dt} + \frac{i\ dL}{2\ dt} \tag{2.3}$$

Thus the rate of flow of energy is

$$vi = \left[Li\frac{di}{dt} + \frac{i^2\ dL}{2\ dt}\right] + \frac{i^2\ dL}{2\ dt}$$

The term in brackets may be rewritten as $\frac{d}{dt}\left(\frac{1}{2}Li^2\right)$, and the term $\frac{d\theta}{dt}$ is the angular speed ω. Thus,

$$vi = \frac{d}{dt}\left(\frac{1}{2}Li^2\right) + \frac{i^2\ dL}{2\ d\theta}\omega \tag{2.4}$$

The term $\left(\frac{1}{2}Li^2\right)$ is recognized as the increase in stored magnetic energy, and the term $\left(\frac{i^2}{2}\frac{dL}{d\theta}\right)$ as the power converted to mechanical work. But, mechanical power is

$$\omega_{mech} = T\omega \tag{2.5}$$

$$\text{And hence,} \quad T = \frac{i^2\ dL}{2\ d\theta} \tag{2.6}$$

Since a linear inductance rise is assumed, $\frac{dL}{d\theta}$ is a constant.

$$T = K_t i^2 \tag{2.7}$$

where,

$$K_t = \frac{1}{2}\frac{dL}{d\theta} \tag{2.8}$$

2.3 Design Considerations

The non linearity of the motor's magnetic property makes it difficult to develop a steady state model as in the case of other electric machines. Thus the design of the motor is also difficult. For details the reader is referred to references [2] and [11]. For the present a small summary should be adequate.

Theoretically the machine may have any number of stator poles and rotor poles. Any number of stator phases are also possible. There is however , a definite optimum choice of numbers, depending among other things on:

1. Elimination of mutual inductance between phases.
2. Need to minimize L_u, the inductance in unaligned position.
3. Patterns of repeatability between relative stator pole -rotor pole positions.
4. Self-starting capability of motor.
5. Minimum switching frequency.

The first two conditions will give maximum output; the next two will enable easier control; the last one reduces the skin effect and eddy current losses. Lower switching frequencies also mean that almost all of the available power devices are eligible for use in the converter to control this motor.

The dimensions of the machine are determined by the required output. The present trend is to fit the design to a standard induction motor frame to compare their relative performance and to save the additional cost involved in casting a new frame. A machine with 6 stator poles and 4 rotor poles is considered for study in this thesis.

2.4 Converter Configuration

The SRM has to be driven using a converter. The converter is essentially a switch in series with each phase. The type of converter used however has a strong impact on the design of the controller. The converter configuration used for this project is shown in figure 4 [12]. Note the simplicity and minimum use of switches.

The design uses MOSFETs. The converter essentially operates in three modes. Each phase is independent of the others and hence the operation is described for one phase only. In the first mode the main and phase switches are on, and current flows through the inductor and produces torque. The second mode starts when the phase switch is switched off while chopping. The main switch remains on in this mode. The snubber capacitor C1 charges up till it opposes the supply voltage. Then the diode D_0 turns on and the phase freewheels. The operation toggles between modes 1 and

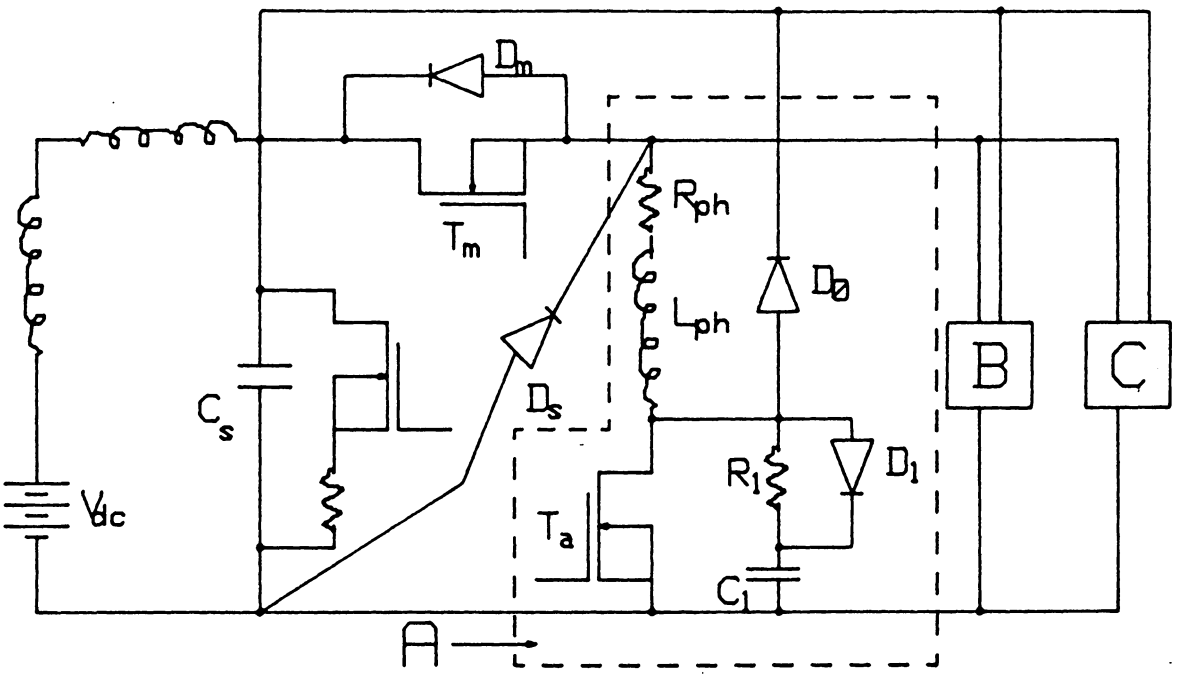


Figure 4. Converter Configuration used

2 while the current is being chopped. The third phase starts when the phase and main switches are turned off before the next phase is activated. The diodes D_0 and D_s turn on, returning power to the source capacitor. The voltage in the capacitor builds up and is discharged when the next phase turns on. If the capacitor voltage builds up beyond a certain voltage it is discharged through a resistor in parallel with it. The main switch is required for the purpose of returning power to the source. It is also used to freewheel the phase during chopping. The source capacitor serves to smooth out the dc link voltage.

3.0 The Controller: Design Aspects

3.1 Introduction

This chapter describes the control requirements of the SRM. It introduces the parameters to be controlled and the modes of operation of the SRM. The effects of motor construction on the controller and the advantages of a microprocessor controller are shown. Hardware-software tradeoffs are discussed in the final section.

3.2 Control requirements

The SRM is inherently a variable speed machine. Hence, the first requirement of the controller is that it should control the speed, as required by the application, ideally from zero to the maximum design speed of the machine. Another requirement is that the motor is self starting and is able to operate in all the four quadrants. This means

that the motor should be able to run in either direction and that in both directions regenerative braking should be possible, i.e. while braking the mechanical energy stored in the machine should be returned to the power supply. A supplemental function of the controller would be to monitor the vital signs of the motor and converter and warn the operator of any malfunction.

A block diagram of the control system is shown in figure 5. The speed is fed back to the controller. Depending on the error, a PI controller is implemented to give the required torque command. Using equation 2.6 the current requirement is calculated. The phases are turned on or off according to the position of the rotor. The value of the actual current is the maintained between a certain 'window'. This type of current control is known as hysteresis control.

3.3 Control Parameters

3.3.1 Advance Angle

As shown in figure 6 the current is switched on during the period of rising inductance for motoring action. To get maximum torque the phase current is maintained a constant for the period of rising inductance. Since the circuit is inductive it takes some time for the current to rise to the required level, after the phase switch is turned on. Hence, to get the desired current at the instant of rising inductance, the phase is turned on at an advanced angle $\hat{\theta}_0$, given by

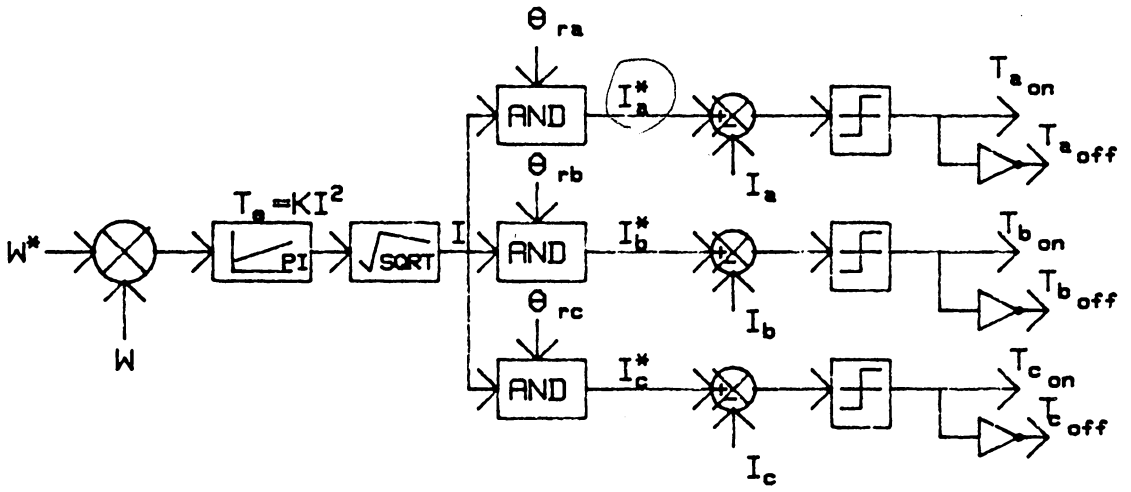


Figure 5. Block Diagram of Control Scheme

$$\hat{\theta}_0 = I \times \frac{L_u \omega_r}{V_d} \tag{3.1}$$

where,

I = phase current, A
L_u = Inductance in unaligned position, H
ω_r = rotor speed, rad/s
V_d = dc supply voltage, V

$\hat{\theta}_0$ is w.r.t. the corner point of rising inductance.

This is an approximate equation, assuming the linear inductance variation shown in figure 6. But since L_u is constant for all current values, the calculated $\hat{\theta}_0$ is accurate.

3.3.2 Modes of Operation

The base speed of the SRM, ω_b , may be defined as the highest speed upto which the torque is maintained at a constant value. Analogous to the d.c. motor, two modes of operation may be defined, one above the base speed and another below base speed. These may be called constant torque region for speeds below ω_b and the constant power region above ω_b

The constant torque is maintained from very low speed until base speed by current chopping as shown in figure 6(a). The current setting is calculated as described in

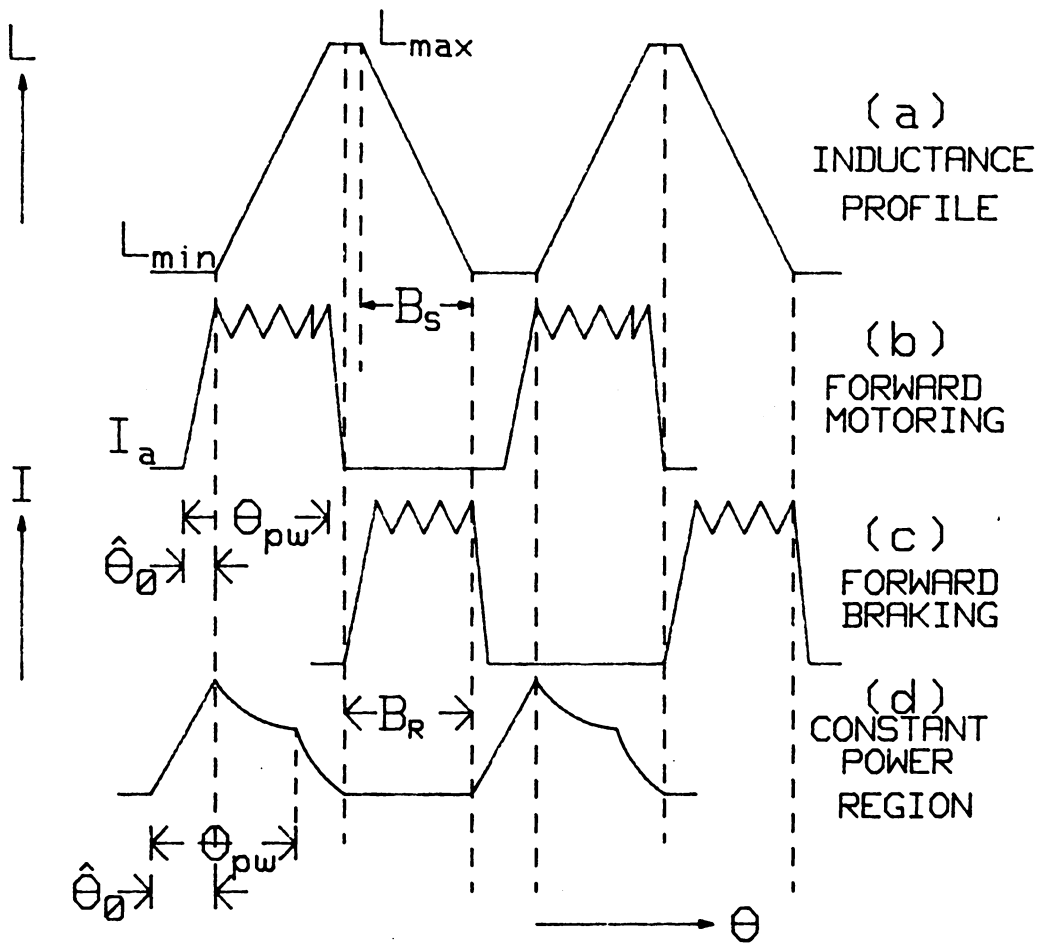


Figure 6. Phase Currents

section 3.1 and maintained between a window centered about the setting. Beyond ω_b, the current control is lost because the current pulses are too small (in time), so that the phase has to be turned off before it can be chopped. This is shown in figure 6(c). Also, the advance angle $\hat{\theta}_0$ cannot be increased beyond a certain angle to prevent it from overlapping with the falling inductance of the previous phase. If the current requirement in this region exceeds a certain peak value, chopping may still be needed.

3.4 *Effects of Motor construction*

Another important factor that affects the design choices in a controller is the construction of the stator and rotor. Since we are considering a machine with 6 stator poles , 4 rotor poles and 3 phases, we find that

(i) angle between consecutive stator poles = 60°

(ii) angle between consecutive rotor poles = 90°

Thus each phase has to be fired every 90° revolution of the rotor, which is when a rotor pole approaches a stator pole. Since there are three phases and each is triggered four times per revolution, there is a total of twelve firings per revolution.

The arcs of the stator β_s and rotor β_r also play an important part in controller design. β_r must be greater than the difference (90 - 60) = 30°. This guarantees an overlap at rest. This is essential to make the machine self-starting from any position.

The stator pole arc β_s decides the time for which the inductance rises. This dimension is important, since if β_s and β_r are equal or close, the flat portion of the profile vanishes (figure 7). The current must now be turned off at a point such that it does not extend into the portion of falling inductance. If it does, a negative or braking torque is produced (since $\frac{dL}{d\theta}$ is negative), thus reducing the average torque.

Since the currents have to be fired at correct positions, a position feedback has to be incorporated in the controller. From the position, the speed may also be calculated.

3.5 *Microprocessor Control*

3.5.1 Advantages of Microprocessor control

The control requirements described in the foregoing sections are realizable using a microprocessor and some additional hardware. The microprocessor enables calculation of the advanced angle $\hat{\theta}_0$ of equation (3.1). The microprocessor will allow connection to a CRT, thus enabling interactive running of the machine. Commands from the keyboard may be used to vary speed, change direction etc.

Additionally, for a prototype system as the one under consideration the experimenter may change the values of some of the parameters and constants such as the PI control constants, torque constant K_t (equation 2.6). This would allow an experimental calculation of some of the parameters of the machine. Ease of debugging and

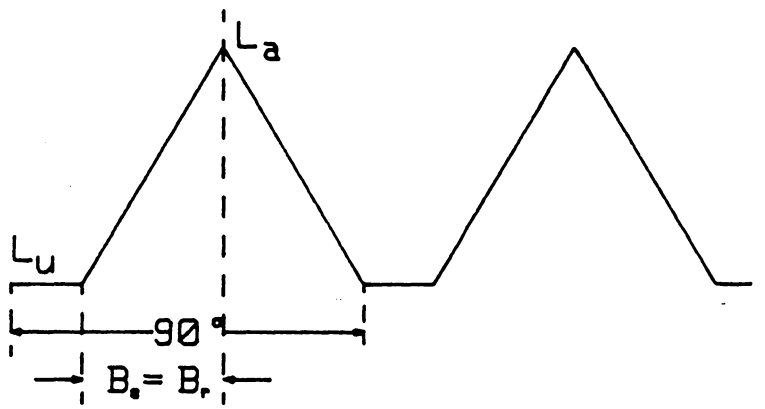


Figure 7. Indutance profile for equal pole arcs.

low cost of the peripheral chips used in the circuit make a microprocessor controller an attractive proposition.

3.5.2 Hardware software tradeoffs

At the first stage of designing the controller, certain decisions had to be made regarding the best use of the microprocessor. It was decided to build both the angle and chopping control schemes in dedicated hardware under software control. The software is responsible for loading the appropriate counters and registers with appropriate values. This choice was made since both the angle and chopping control are time critical. The implementations of the PI controller and diagnostic routines are in software.

4.0 The Controller - Hardware

4.1 Introduction

This chapter discusses the particular hardware design used to implement the controller described in chapter 3. The reasons for choosing the particular processor used and the various blocks of hardware are described. A block diagram of the hardware is shown in Fig 8.

4.2 Choice of Processor

The following guidelines were used for selecting a processor for SRM control:

1. The estimated timing requirements of the circuit
2. The cost of the processor, peripherals and subsystems.

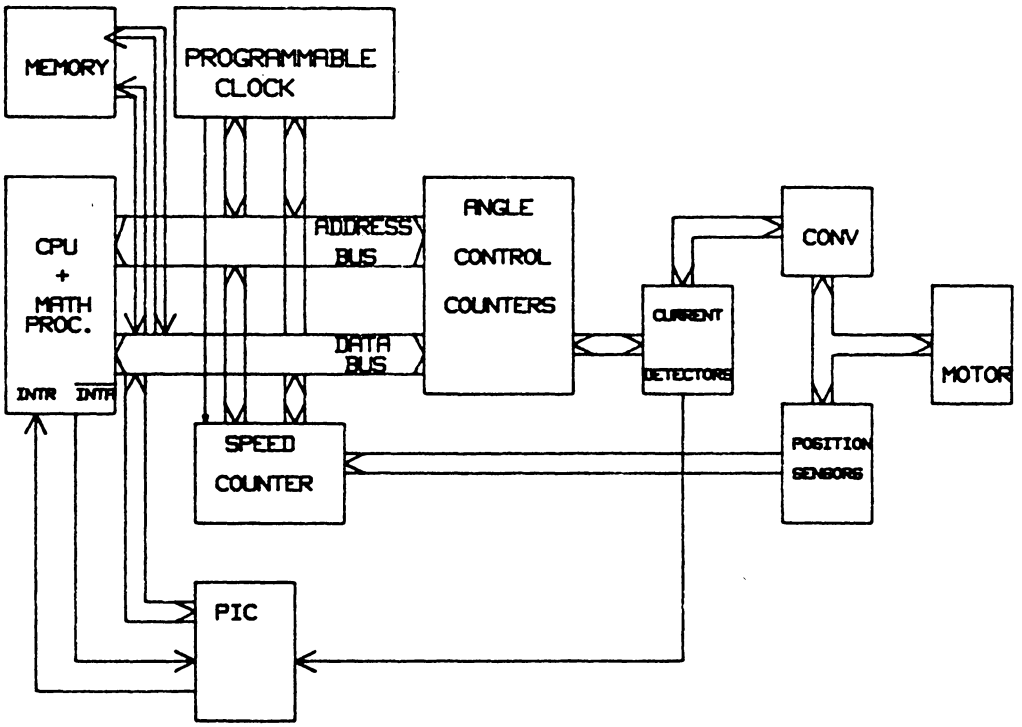


Figure 8. Hardware Block Diagram

3. The availability of a development system and other support within the research environment.

These factors led to the choice of an 8088 processor. A CPU card using the 8088 on an STD bus was readily available. A PC connected to the system via a USART card served as the development system. Peripheral chips and support systems were available with a short lead time. The STD bus is an industry standard bus, details of which are given in APPENDIX E. The 8-bit accuracy offered by the 8088 was considered sufficient for this project.

4.3 Rotor Position Measurement

The rotor position is measured using a 10 bit absolute position encoder. The encoder reverses output in the reverse direction i.e. it counts down from high to low. Thus the outputs of the position EPROM for reverse direction have to be programmed on three other bits of the EPROM output to give proper signals. The signals are multiplexed and switched as required.

4.4 Resolution of Angles

As discussed in Chapter 3, the control scheme requires a position feedback. This is measured as discussed in the preceding section. The LSB of the digital position re-

peats at an angular interval of 0.7° . This pulse train may be used as a clock for the angle counting scheme described in Section 4.5 . Thus the resolution of angles is 0.7°. This accuracy is sufficient for the angle measurements.

4.5 *Speed Measurement*

The digital position bits form the address to an EPROM as shown in Fig 9. The output of the EPROM is as shown. This output serves two purposes.

1. It is used in starting the motor.
2. It triggers three monostables, one for each phase. These are triggered precisely at the point where a rotor pole begins to move under a stator pole of that phase. For an ideal machine, this translates to the point where the inductance in that phase just begins to rise. Thus each monostable gives a narrow pulse for every 90° of rotation.

A logical OR of the three monostable pulses gives a series of pulses 30° apart, since firing of each phase is displaced by 30° from firing of last phase. These are fed to a 8253 counter. A high-frequency clock is accumulated between two consecutive pulses. Knowing the frequency of the clock, the elapsed time between the two pulses can be extrapolated to 1 revolution. This is the time-period corresponding to one revolution. Its inverse gives the speed.

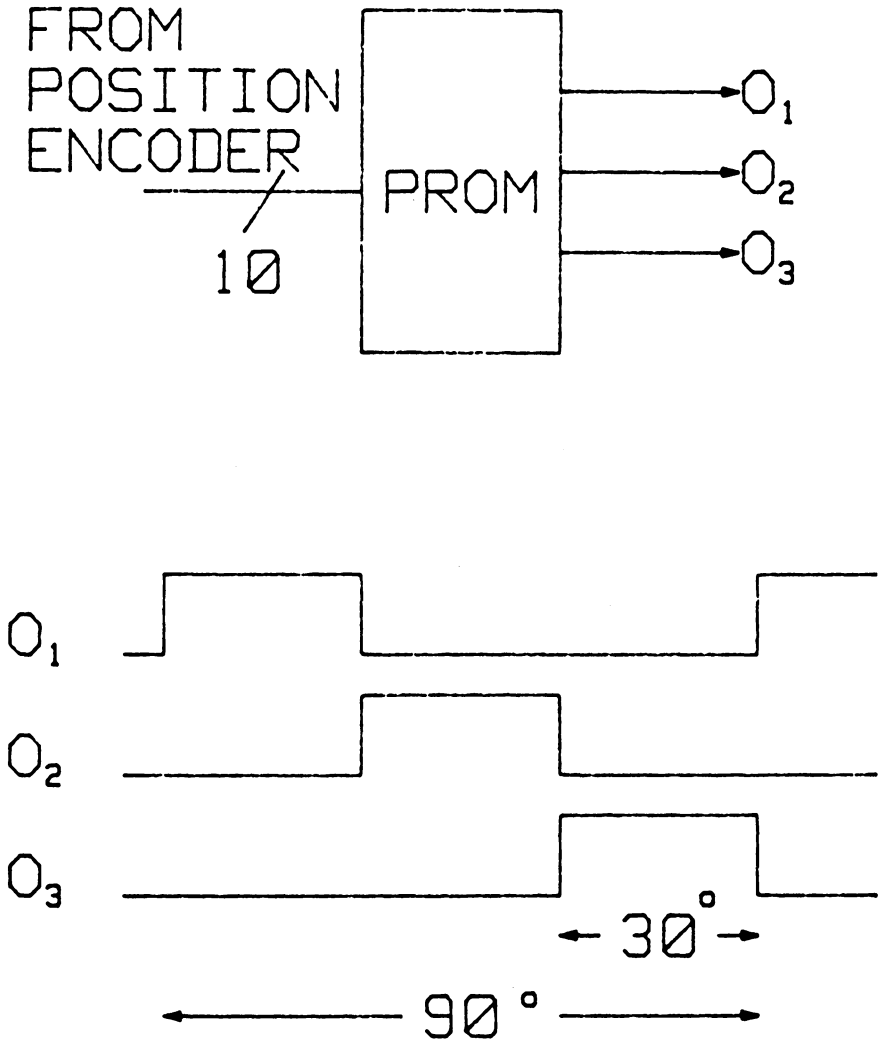


Figure 9. Output of Position EPROM

4.6 Angle Control

Angle control is achieved by using dedicated hardware in the form of two 8253 counters for each phase. This scheme is shown in Fig 10. The first counter, triggered by the monostable for that phase, counts down the delay angle $\theta_0 = 90 - \hat{\theta}_0$, where $\hat{\theta}_0$ is given by equation 3.1. At the end of the count it activates the phase switch. Simultaneously it also triggers the second counter which counts down the angle for which the phase remains on.

Note that the advanced angle is now a delay angle i.e., the present position pulse is used to trigger the counter for the next firing of that phase. This is because the counter cannot predict the occurence of the 90° pulse from the monostable. The counting is done using the 0.16° pulse train generated by the encoder as described in section 4.3.

4.7 Chopping Control

The current has to be maintained within a window centered about the required level. The chopping circuit used is shown in Fig 11. The current feedback is converted to a digital word using an A/D converter. This digital word is compared with the upper and lower limits of the window, stored in two buffers. Two digital comparators are used for this purpose. One comparator turns on the phase when the lower limit is

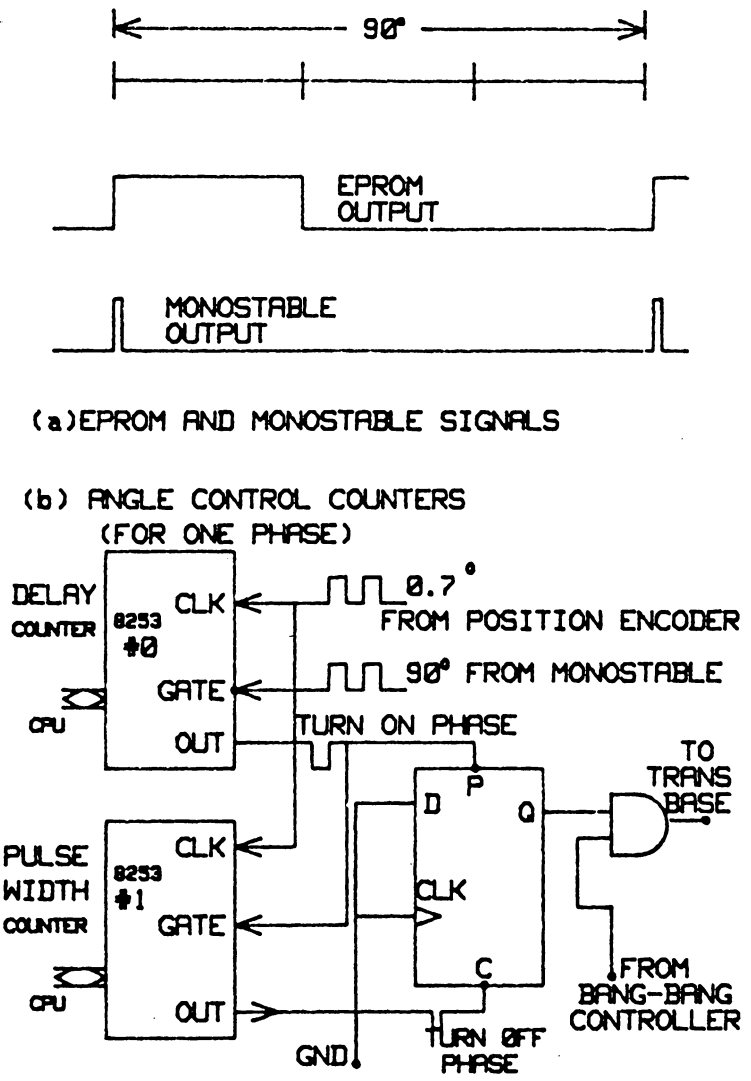


Figure 10. Angle Control Scheme

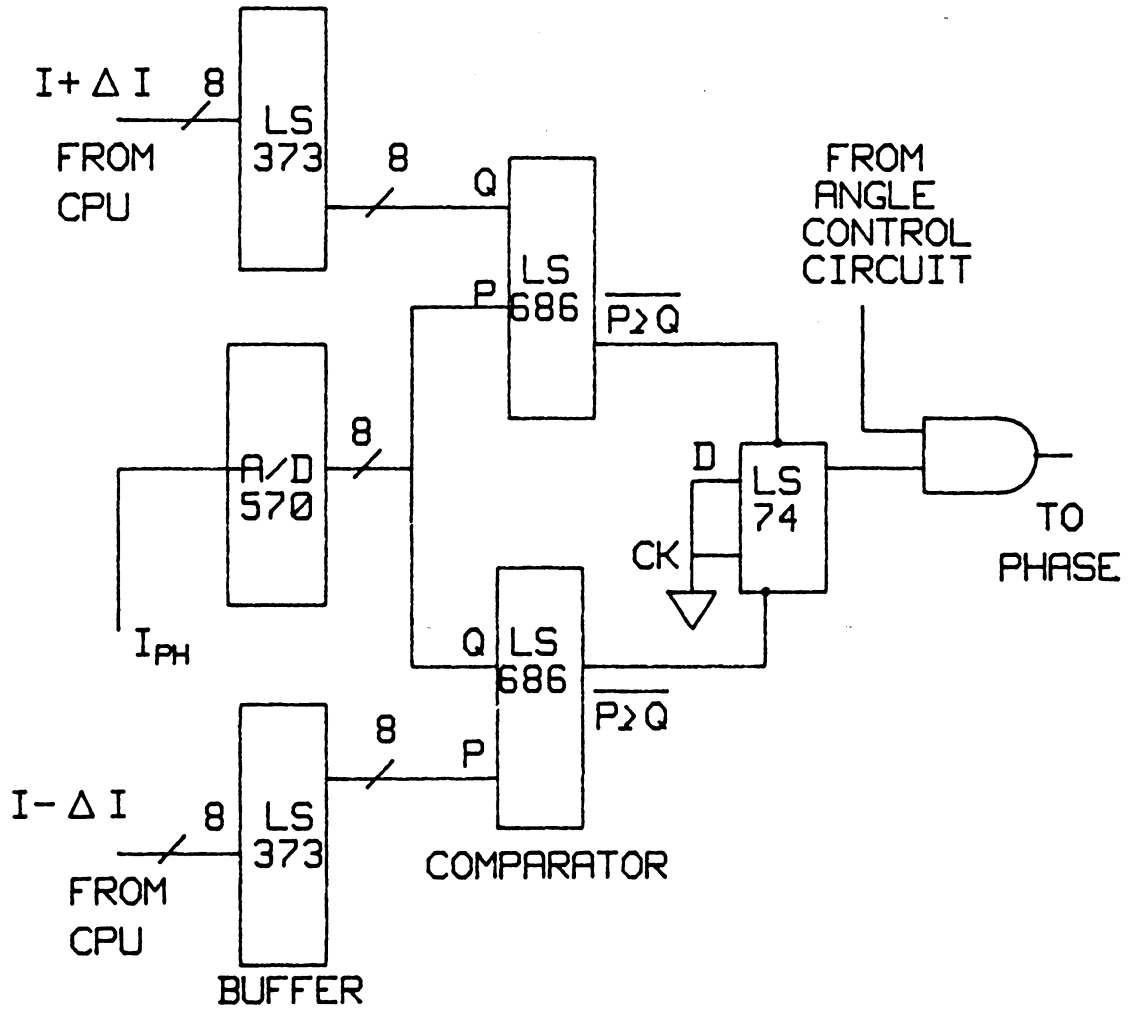


Figure 11. Bang-Bang Control Scheme

reached. The other comparator turns it off when the upper limit is reached. The control signals required to operate the A/D and the comparator chips are generated using a PAL. This effectively reduces the chip count of the circuit.

4.8 Additional hardware requirements

These include a 8259 Programmable Interrupt Controller. This is to interrupt the processor every 30° to execute the PI algorithm of the speed controller. A current buffer stores the digital value of the feed-back current for diagnostic purposes. Supplementary hardware in the form of buffers, drivers, address decoders, flip-flops and logic gates are also used. Hardware requirements and detailed diagrams appear in APPENDIX D.

5.0 The Controller: Software

5.1 Introduction

This chapter is concerned with the software portion of the controller. Sequencing of the controller is described as also the strategy used to start the motor. The PI control algorithm is also described.

5.2 Main Functions

The software is designed with the following objectives:

1. To start the machine.
2. To vary speed and change direction as required.
3. To implement the PI control algorithm in speed controller

4. To enable interactive changes in variables and constants in motor and controller
5. To perform diagnostics.

5.3 Starting

The starting of a switched reluctance motor requires the position of the motor at rest. For easier starting it also needs an overlap as described in chapter 3. The position EPROM has a certain output at start since the position at rest is available. As seen from the output profile in Fig 8. the rotor is in a position such that a current flows through one phase in any position to provide a motoring torque. To acheive starting, the three outputs of the EPROM are directly fed to the base drive circuits of the MOSFET phase switches. This ensures that the motor will start from any position.

For reverse starting, three signals similar to the three for forward starting, are programmed on three other bits of the 8-bit EPROM. These lines are multiplexed onto the three lines as shown and switched according to the starting direction required.

5.4 Sequencing

The software is essentially divided into four modes of operation. A simplified sequence diagram is shown in figure 12. At power-up the processor enters the neutral mode. Here, the PI constants K_p and K_i and variables like commanded speed and

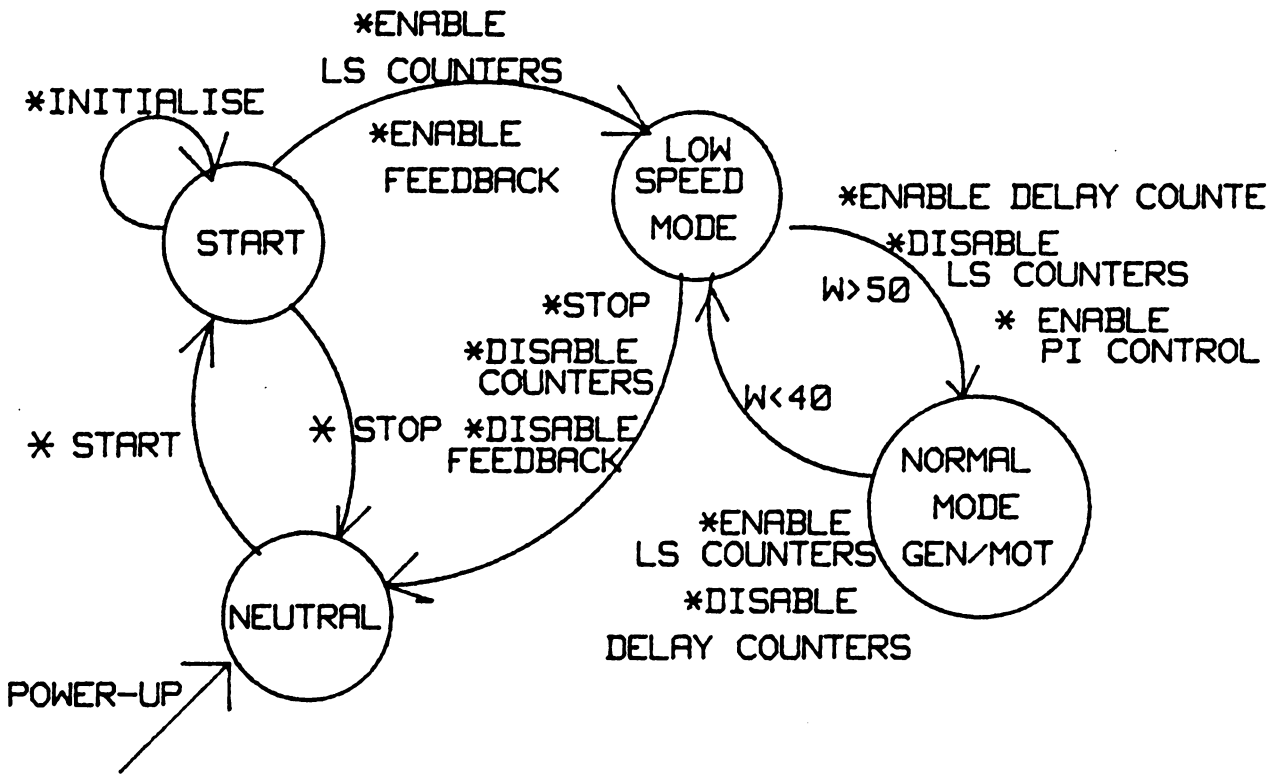


Figure 12. Simplified Sequence Diagram

direction may be entered by the user. At a start command it goes into the start-up mode where it initializes the required registers, counters, switches etc. It then enters the low-speed mode. It stays in this mode till a certain minimum speed is achieved. The current setting corresponds to the maximum allowable to achieve a high starting torque. Next, it enters the normal mode. Here, the PI speed control is enabled and current control is enforced.

5.5 The PI Algorithm

The control signals to the SRM are periodic in angular position but not in time. Hence, it was decided to implement the PI algorithm using an angular interval. For the maximum speeds specified for the machine a period of 30° is considered a good choice. Thus the torque at the kth sample is given by

$$T^*(k) = K_p e(k) + K_2 \sum_{j=1}^{k} e(j)\delta t_j \qquad (5.1)$$

where $e(j)$ = error at j^{th} sample
$= \omega_j^* - \omega_j$
ω_j^* = set speed
ω_j = feed-back speed at j^{th} sample

Note that since the time-step is varying, δt_i is not constant. Since a certain count, say x_i number of pulses of a high-frequency clock are accumulated between two pulses 30° apart.

Writing,

$$\delta t_i = x_i t_c \tag{5.2}$$

where t_c is the period of high-frequency clock.

Thus,

$$\omega_j = \frac{\Pi}{6\delta t_i} = \frac{\Pi}{6x_i t_c} = \frac{K_R}{x_i} \tag{5.3}$$

where, K_R is a constant for constant clock frequency. The proportional speed error is,

$$\begin{aligned} e(k) &= \omega_j^* - \omega_j \\ &= \omega^* - \frac{K_R}{x_i} \end{aligned} \tag{5.4}$$

for which the torque command is,

$$\begin{aligned} T^*(k) &= K_p\left(\omega_j^* - \frac{K_R}{x_i}\right) + K_2\sum_{i=1}^{k}\left(\omega_j^* - \frac{K_R}{x_i}\right)x_i t_c \\ &= K_p\left(\omega_j^* - \frac{K_R}{x_i}\right) + K_2 t_c\sum_{i=1}^{k}\left(\omega_j^* - \frac{K_R}{x_i}\right)x_i \\ &= K_p\left(\omega_j^* - \frac{K_R}{x_i}\right) + K_i\sum_{i=1}^{k}\left(\omega_j^* - \frac{K_R}{x_i}\right)x_i \end{aligned} \tag{5.5}$$

This algorithm is slightly more complicated than if a fixed time step had been used. The additional computational overhead is supported by the math-coprocessor. It performs the math operations to a high degree of precision and hence gives very

accurate results. Then using the equation $T = K_t i^2$ the required current is determined.

5.5.1 Diagnostics

An important function of the software is to oversee the operation of the subsystems and the entire system. The diagnostic routines periodically check for over-current, over-voltage and over-speed. In case of abnormal operation, a warning is flashed on the screen. The operation of the machine is halted and the processor returns to the neutral state awaiting user commands.

6.0 RESULTS

6.1 *Introduction*

This chapter discusses probable sources of errors in the controller and their effects on the performance. The problems faced in the practical implementation are briefly discussed. It gives the results obtained by implementing the controller on a converter as described in chapter 2, using a static RL load to simulate the motor.

6.2 *Errors*

The microprocessor implementation of any control application has certain inherent errors. The most important of these is the rounding off errors. Though most of the calculations in this implementation are done using a math coprocessor the final out-

puts are all integers. While preventing intermediate round-off errors, the implementation cannot avoid some rounding error in

1. The current command
2. The delay and pulse-width angle command

Some of the intermediate computations are also done on integers for simplicity.

Another source of error is the resolution of the angles. The 10-bit position sensor gives a fastest clock of 0.7° on the LSB. Thus all angles have to be measured in steps no smaller than 0.7°. This however, is not expected to affect the performance of the controller since most of the angles involved are larger than 0.7°.

A third source of error is the fact that a varying time step has been used in the PI control implementation. At low speeds, the time taken to traverse the 30°, which is the sampling period, is large. Thus any jitter will not be detected by the controller, leading to slightly inaccurate results. At higher speeds it may become necessary to average out the speed over a few samples to smooth the jitter.

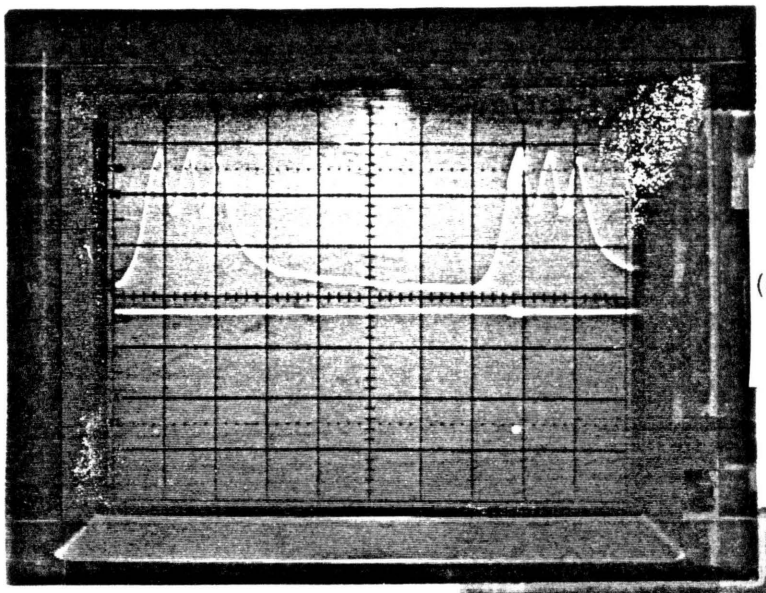
The counters for each phase are triggered every 90°. Once they are triggered the loading of a new count does not affect the current count. Any update takes effect only for the next triggering. Thus for positive torque command, there is a minimum delay of 120° before an update takes effect. This may cause a poor response at low speeds. For negative torque commands the delay is 30° since the counters are reinitialised each time. A small source of error is also due to the large speed variation required of the drive. As discussed in section the clock used to measure the speed is varied over blocks of the speed range. The way the counters are configured this takes place

'on the fly'. One sample is incorrect during a period of change over from one clock to another. This sample is discarded and hence introduces an error in the current calculation.

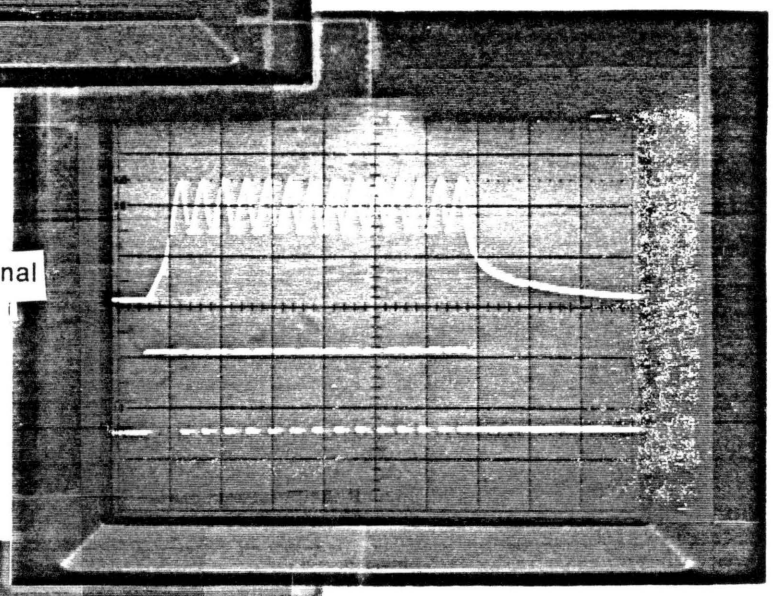
6.3 Results

The hardware and software were tested using a simulated position feedback signal. Both performed correctly from zero to maximum design speed. The converter was tested using a static RL load. Some output waveforms are shown in Fig 13.

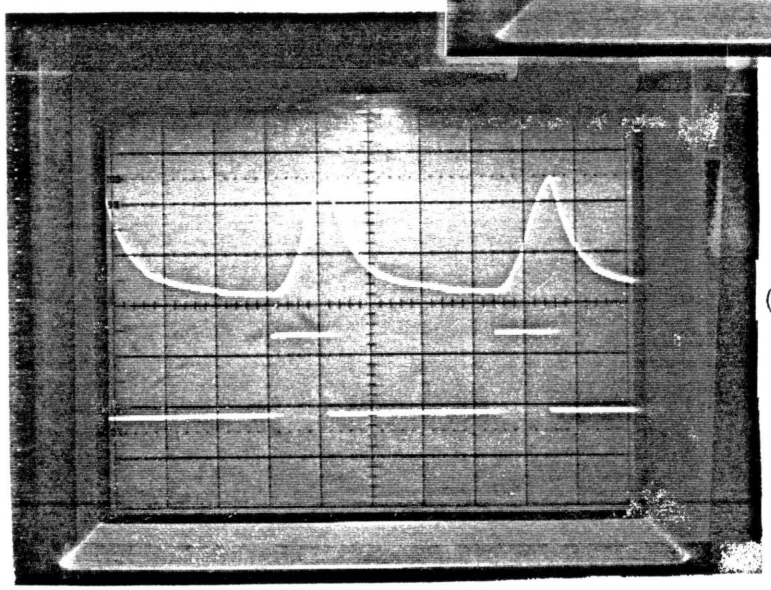
Fig 13a shows the concept of 'advanced angle'. The lower waveform is the narrow position pulses from the triggering monostable. This is the point where inductance begins to rise. Figure 13b shows the current and corresponding base drive signal at low speeds. The current is chopping between 12 amperes and 8 amperes to give an average of about 10 amperes. The waveform in fig 13c is for higher speeds. Note that the current pulses are too narrow, so that chopping ceases. The current is about 4 amperes. This is below the requirement, but the speed of the machine prevents it from going higher. It must be noted that the converter was operated at 180 volts when these photographs were taken. The actual machine will work at 300 volts, thus pushing the current higher. This will enable the machine to go to speeds of 3000 rpm which is the maximum design speed of the controller.



(a) Upper Trace: Phase Current
Lower Trace: Monostable Pulse
Scale:2V/div
Time Scale: 2ms/div



(b) Upper Trace: Phase Current
Lower Trace: Base Drive Signal
Scale:2V/div
Time Scale: 2ms/div
Speed: @330 rpm



(c) Upper Trace: Phase Current
Lower Trace: Base Drive Signal
Scale:2V/div
Time Scale: 2ms/div
Speed: @1770 rpm

Figure 13. Experimental Results

6.4 *Noise problems in the practical implementation*

One of the biggest problems faced during the entire implementation was due to the noise. Both, the digital and analog circuits are susceptible to noise. Solutions included decoupling capacitors, resistors to ground(to drive the buffers 'hard'), short leads where possible, buffered signals etc. These worked with limited success. The EMI noise induced by the inductive circuit being chopped is picked up easily and it is beyond the scope of this project to get rid of it. Thus, another probable source of error is the noise in the current feed back circuit. Designing better filters in the analog feedback circuit may eliminate some of the noise. A linear power supply instead of the switched mode power supply used will reduce the switching noise.

7.0 Conclusion

7.1 Introduction

This chapter discusses the conclusions of the research and recommendations for future study

7.2 The Advantages of microprocessor control

The microprocessor controller is a flexible tool to study the prototype. The constants may be changed and the controller tuned for various machines. It is also possible to rewrite portions of the software to make it flexible and modular and enable different algorithms to be implemented using the same hardware. The microprocessor controller may also serve as a data-logging tool to enable a study of the dynamic performance of the machine.

Another benefit of the controller is that it can be used to 'program' a sensorless controller being currently developed by the Motor Drives group at Virginia Tech and compare it's performance to the controller using rotor position feedback.

The size of the software was larger than originally estimated. The code was about 2Kb long and the data about 1Kb. The debugging was done using a simple debugger included in a EPROM on the CPU card. This was not found to be the best method, specially for hardware debugging. It may be worth mentioning here that the 8088 is not the best of processors for a controller. A dedicated controller like the Intel 8051 may have been a better choice. The 8088 was chosen simply because of the availability of support systems and debugging facilities. However, the advantages of using a microprocessor controller are still evident.

7.3 *Limitations of the implementation*

The implementation in its present form has some limitations. The speed of the processor may be increased and some of the functions incorporated into the software. This will reduce the hardware and consequent problems. Another limitation is its inability to operate correctly at low speeds because of the varying sampling time used. The sampling time becomes too large at low speeds making the controller response poor.

A minor limitation is that the controller is not independent of the peculiarities of the position sensors. Since it was decided to use an absolute encoder, only position

sensors which give 10 bits (or higher) of absolute position may be used in this scheme.

It is concluded that the advantages offered by the microprocessor far outweigh the drawbacks in the development of a prototype SRM drive system. The present implementation is a first iteration and can be tuned in the future. The working of the controller was found to be satisfactory.

7.4 Recommendations for Future Study

7.4.1 A VLSI Implementation

Semiconductor technology continues to shrink the size of integrated circuits. This advance in technology can improve the cost-effectiveness and quality of controllers for the SRM. Once the main aspects of the SRM control are studied and understood, a VLSI implementation is a logical next step. There are numerous reasons for this:

1. Many features of the general purpose microprocessor remain unused.
2. For low-end applications a very economical controller is required.
3. VLSI implementation of the control circuit will also
    a. Increase the speed of operation and hence the maximum design speed of the machine.
    b. Reduce size of the controller by a large factor.
    c. Reduce power requirements of the controller circuit.

4. Additionally the controller can be made reconfigurable, i.e. it may be used for various kinds of motors eg. a 6/4 machine or a 8/6 machine.

Thus a VLSI implementation appears promising.

7.4.2 Sensorless Control Methods

One of the drawbacks of the control scheme discussed here is that it needs a position feedback both at rest and in motion. Any scheme to sense the position, specially to the high degree of accuracy required by the scheme, is inherently expensive adding to the cost of the drive system. Thus, a 'sensorless' control scheme which predicts the position of the motor, instead of using an actual position feedback, is also one of the aims of building a cost -effective SRM drive.

Bibliography

1. L.E.Unnewehr and W.H.Koch, "An Axial Air-Gap Reluctance Motor for Variable Speed Applications", IEEE Trans., 1974, PAS-93, pp 367-376

2. P.J.Lawrenson, J.M.Stephenson, P.T.Blenkinsop, J.Corda and N.N.Fulton, "Variable speed Switched Reluctance Motors", IEE Proc. Vol 127, Pt B, No.4, Jul 1980, pp 253-265

3. J.V.Byrne, "Tangential Forces in Overlapped Pole Geometries Incorporating Ideally Saturable Materials", IEEE Trans on Magnetics, Mag-8, No. 1, 1972, pp 2-9

4. J.V.Byrne and J.G.Lacy, "Electrodynamic System Comprising a Variable Reluctance Machine", British Patent No. 1321110, 1973

5. J.F.Lindsay, R.Arumugam and R.Krishnan, "Finite-Element Analysis charecterisation of a switched reluctance motor with multi-tooth per stator pole", IEE Proc., Vol.133,Pt.B,No.6,Nov 1986, pp 347-353

6. P.H.Chappel, W.F.Ray and R.J.Blake, "Microprocessor control of a Switched Reluctance Motor", IEEE-IAS 1985, pp 542-547

7. B.K.Bose, T.J.E.Miller,P.M.Szczesny and W.H.Bicknell, "Microcomputer Control of a Switched Reluctance Motor", IEEE IAS 1985, pp 542-547

8. W.F.Ray and R.M.Davis, "Inverter Drive for Doubly Salient Reluctance Motor: Its Fundamental Behaviour, Linear Analysis and Cost Implications", Electric Power Applications, Vol.2, Dec 1979, pp 185-193

9. T.J.E Miller, "Converter Volt-Ampere Requirements of the Switched Reluctance Motor Drive", Conf. Rec. 1984 IEEE/IAS Annual Meet., pp 813-819

10. J.V.Byrne, M.F.Mcmullin and J.B.Odwyer, "A High Performance Variable Reluctance drive: A new brushless servo", Proc. of Motorcon, Oct 1985, pp 147-160

11. R.Krishnan, R.Arumugam and J.F.Lindsay, "Design Procedure for Switched Reluctance Motors", Conf. Record, IEEE-IAS Annual Meeting, Colorado,Oct '86, pp 858-863

12. R.Krishnan, "Analysis of Electroncally Controlled Machines", Class Notes, EE Dept, VPI&SU, Blacksburg, VA, 1986.

13. J.T.Bass, M.Ehsani and T.J.E Miller, "Simplified Electronics for Torque Control of a Sensorless Switched Reluctance Motor", IEEE Transcations in Industrial Electronics, Vol IE-34, No. 2, May '87, pp 234-239.

14. R.M.Davis,W.F.Ray and R.J.Blake, "Inverter Drive for Switched Reluctance Motor: Circuits and Component Ratings", IEEE Proc., Vol.128, Pt. B,No. 2,Mar '81,pp 126-136.

15. A.R.Oza,R.Krishnan and S.Adkar, "A Microprocessor Control Scheme for Switched Reluctance Motor Drives", accepted for presentation at IEEE-IECON, Nov '87.

16. R.Arumugam,J.F.Lindsay and R.Krishnan,"A Comparison of the Performance of Two Different Types of Switched Reluctance Motors", Electric Machines and Power Systems, 12, 1987,pp281-286.

17. iAPX 86,88 Users Manual, Intel Corp., Santa Clara, CA, 1981

18. IBM Macro Assembler Users Manual, IBM Corp., Boca Raton, FL, 1984.

19. IBM Macro Assembler Reference, IBM Corp., Boca Raton, FL, 1984.

Appendix A. Dimensions of the SRM

This appendix gives the dimensions and other details of the SRM under study. Figure 14 shows the variables for reference.

A.1 Dimensions

Shaft radius	S_r	= 1.428 cm (0.5625inch)
Stator poles		= 6
Rotor poles		= 4
Stator outside radius	S_{or}	= 9.4996 cm (3.74inch)
Rotor core outside radius	r_{or}	= 6.103 cm (2.403inch)
Rotor core inchner radius	r_{ir}	= 3.426 cm (1.349inch)
Rotor back iron thickness	b_{yr}	= 2.0 cm (0.787inch)
Stator back iron thickness	b_{ys}	= 1.0515 cm (0.414inch)
Stator pole height	h_s	= 2.31 cm (0.913 inch)

Stator pole arc angle	θ_s	= 23.91°
Rotor pole arc length	θ_r	= 35.92°
Airgap length		= 0.025 cm (0.010 inch)
Wire size		= AWG# 19
Number of turns per pole		= 268 turns
Stack length	L	= 6.037 cm (2.3768inch)

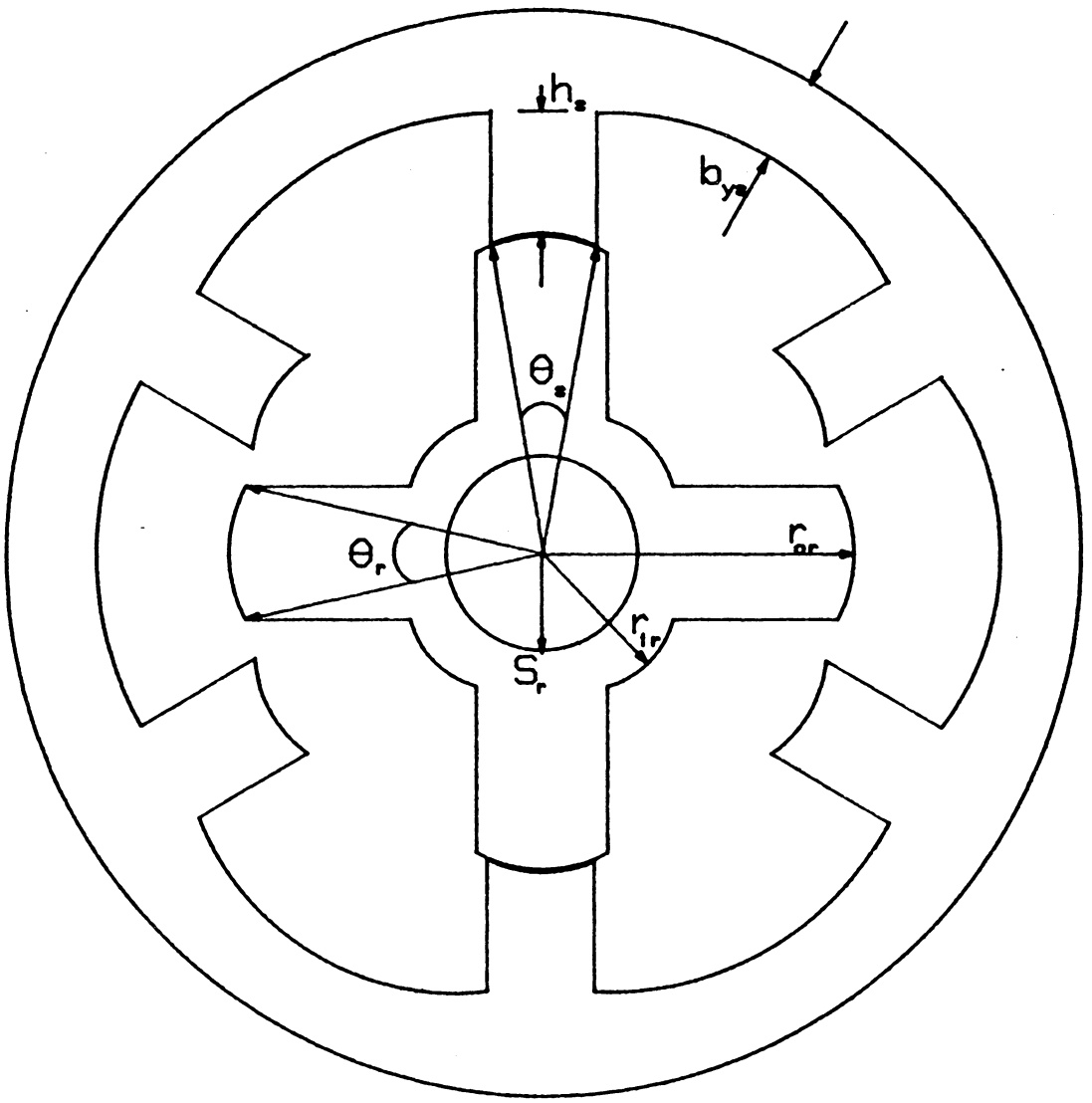


Figure 14. Dimensions of the SRM

Appendix B. Software Listing

This appendix gives a listing of the software used in the controller. A listing of the program for the PAL chip used in the chopping control is also included.

The software is written entirely in assembly language. The reader is referred to[17] and [18] for details of the 8088 assembly language programming. These also give details of programming the 8087 math coprocessor.

B.1 The Control Program

```
page     ,132
;THIS PROGRAM IS SOFTWARE FOR SRM CONTROLLER
;
;AUTHOR   :      AMEESH  R.  OZA
group_all group code,data,stack
assume  cs : code,ds : data,ss : stack
8087
;
data    segment
v_msg   db      'TECON SRM V 1.0',0DH,0Ah,'$' ;version message
slowing_down db  'MOTOR IS SLOWING DOWN',0DH,0AH,'$'  ;slowing down message
too_fast  db     'THE MACHINE IS TOO FAST',0DH,0AH,'$' ;
opt_msg   db     '1 : CHANGE SPEED',13,10,10,
        db      '2 : CHANGE DIRECTION',13,10,10,
```

```
        db    '3 : STOP MACHINE',13,10,10,
        db    'YOUR CHOICE : ','$'
opt_err  db     CR,LF,LF,'SORRY! TRY AGAIN : ','$'

inp_speed db     13,10,'INPUT NEW SPEED : (ESC TO ABORT)',CR,LF,'$'
esc_msg  db     13,10,10,'HIT ESC TO INPUT',13,10,10,'$'
inp_err  db    CR,LF,'INVALID INPUT. TRY AGAIN',CR,LF,'$'
lrg_inp  db    CR,LF,'CANNOT BE GREATER THAN 3000',CR,LF,'$'
hlt_msg  db     CR,LF,'MACHINE IS NOW IN HALT MODE',CR,LF,'$'
;
 pll0  equ    0ff88h              ;counter 0
 pll1  equ    pll0+1               ;       1
 pll2  equ    pll1+1               ;       2
 pllm  equ     pll2+1               ;mode register
;
dly_ctr_1 equ  0ff85h
wdth_ctr_1 equ 0ff86h
mode_1  equ    0ff87h
dly_ctr_2 equ  0ff8eh
wdth_ctr_2 equ 0ff84h
mode_2  equ    0ff8fh
dly_ctr_3 equ  0ff8ch
wdth_ctr_3 equ 0ff8dh
;
;mde_wrd0 equ  36h
;mde_wrd1 equ  7ah
;mde_wrd2 equ  0b6h
;
dly_mode_1 equ 7ah
pls_mode_1 equ 0bah
dly_mode_2 equ 0bah
pls_mode_2 equ 03ah
dly_mode_3 equ 03ah
pls_mode_3 equ 7ah
;
delay  equ    1c7h
pls_wdth equ   08eh
;
crrnt_prt equ  0ff90h              ;port to read current
diagnostics equ 0ff90h
hi_crrnt_prt equ 0ff94h             ;port for upper window
lo_crrnt_prt equ 0ff98h             ;port for lower window
;
lo_spd_crnt_hi equ 0cch
lo_spd_crnt_lo equ 99h
;
master_en equ   0ffb8h              ;master enable port
master_dis equ  0ffbch              ;master disable port
;
start_mode_port equ 0ff9ch           ;
norm_mode_port equ  0ffa0h           ;
```

```
reverse_port   equ  0ffa4h
forward_port   equ  0ffa8h
;
mask_int1 equ    01h              ;masks interrupt ir0
halt_clk     equ 1250
div_by_n_slo  equ 125
div_by_n_med  equ  25
div_by_n_fast  equ  2
fast_e_nuf    equ 150
start_count  equ 10000
mask_dir     equ 01
;
slo_flg   db     00
fast_flg  db     00
dir_flag  db    00
slo_spd_count dw 00
status    dw     00
dec_flg   db     00
dec_delay equ    42
acc_delay equ    90
theta_cap_hi equ  8
theta_cap_lo equ  2
deg_2_rad dd     57.2958          ;so many degrees = 1 rad
theta_cap dw     00
theta_del dw     00
icmd      dw     0                ;command current
i_true    dw     0
i_act     dw     0                ;current loaded in comparators
hi_crnt   db     00
lo_crnt   db     00
crrnt_fact dd    25.6
halt_flag  db   00
int_flag   db   00
last_inp   dw   00
last_dig   dw   00
i_real     dd    0.0
;
div_n   dw      08h
omega_temp dw    00
;
esc       equ    1bh
CR        EQU    13
LF        EQU    10
option    db     00
opt_flag  db     00
esc_flag  db     00
rev_flag  db     00
inp_flag  db     00
timing_count  equ 1000
;
count   equ     0ffffh            ;count for delay rout.
```

```
;
;this data block is for PI control storage
;
kt        dq     6.2404E-2
kr        dq     0.0
kp        dq     0.2
ki        dq     0.8                ;this is -ve power of 2
ki_mod    dq     0.0                ;modified ki of algorithm
w_fb      dq     0.0                ;actual feed_bak speed
w_set     dq     41.8879
sum_ei    dq     0.0                ;integral error
ti        dq     0.0                ;torque command
i_lo      equ    15h
i_hi      equ    0c0h
hi_spd    equ    3000
lo_spd    equ    50
sample_new dw    00
delta_i   db     0eh
Imin      dq     21.34e-3
Vd        dw     150
kr1       dq     52359.877
kr2       dq     654498.4695
theta_res dq     0.17578125
k2        dq     8.0e-7
k1        dq     1.0e-5
rpm_2_rad dd     0.10472            ;to convert input speed to rad/s
;
;this is structure defined to input user values
;
u_in_val  struc
;
inp_msg   db     'PLEASE INPUT KONSTANT:'
end_of_msg db    '$'
ch_count  db     0
count_after db   0
ch_flag   db     0
up_lim    dd     0.0
lo_lim    dd     0.0
val_def   dq     0.0
val_input dw     0
real_val  dq     0.0
;
u_in_val  ends
;
;define dummy structure
dummy     u_in_val <'PLEASE INPUT DUMMY:',,,,,20.0,10.0,10.0,0,0.0>
omega     u_in_val <'PLEASE INPUT SPEED:',,,,,3000,50,1800.0,0,0.0>
;
ten       dw     10
null      dw     00
val_divisor dw   0
```

```
x1      dw    0
x2      dw    0
x3      dw    0
sum     dd    0
sum1    dd    0
prop    dd    0
torq    dd    0
longjump  dd   0ffff0000h
;
;
error_msg db    0dh,0ah, 'sorry! invalid value. try again:','$'
data   ends
;
stack  segment
fill   dw     128 dup(00)
stack_top label word
stack  ends
;
code   segment
;
;this is macro to call the message subroutine
scrn_msg  macro  message_name
        mov    bx,offset message_name ;start_of_message is in bx
        call   message
        endm
;
;this is macro to input user defined values
;
input    macro  var_nam_struc
        mov    bx,offset  var_nam_struc
        call   user_value
        endm
;
;service_routine
start_serv:

        push   ax
        push   bx
        push   cx
        push   dx

        mov    dx,pllm
        mov    al,40h
        out    dx,al              ;latch count
;
        mov    dx,pll1
        in     al,dx
        xchg   al,ah
        in     al,dx
        xchg   al,ah               ;input count
;
```

```
        xor    ax,count
        mov    sample_new,ax        ;store count

        mov    int_flag,01          ;set int flag
        mov    al,halt_flag
        add    al,pi_flag
        cmp    al,00
        jz     control
        jmp    div_by_zero
;
;this is procedure to implement PI control loop
;
control:
;pi_control proc
;
        mov    ax,sample_new
        cmp    ax,00
        jne    not_zero             ;can't div by zero
        jmp    div_by_zero
not_zero: jg     calc_trq
        mov    sample_new,7fffh     ;limit  count
;
;this part is executed by 8087
;
calc_trq:  fld      kr              ;load kr
         fidiv    sample_new       ;div by Xi
         fst      w_fb             ;store feedback speed

         fld      w_set            ;load W*
         fsubr                     ;W* - W = ep
         fld      st(0)            ;duplicate stack top
         fimul    sample_new       ;
         fmul     ki_mod           ;form integral part
         fadd     sum_ei           ;form integral sum
         fst      sum_ei           ;store new sum
         fxch
         fmul     k_prop.real_val  ;form proportional part
         faddp    st(1),st(0)      ;form torque
         fst      ti               ;store torque
        ftst                       ;test for sign of current
        fstsw  status
        fwait
        mov    al,byte ptr status + 1
        and    al,01h
        cmp    al,01
        je     neg_trq
        mov    dec_flg,00           ;reset dec_flg
        jmp    pos_trq
neg_trq:  fchs
        fwait
```

```
        mov    dec_flg,01          ;set dec_flg
pos_trq:
         fdiv     k_torque.real_val ;divide by constant
         fsqrt
;       fst    i_real              ;store in real format
;       fist   i_true              ;store required current
        fmul   crrnt_fact            ;convert to a/d equivalent volts
        fistp  icmd
        ffree                      ;restore empty stack
        fwait
        mov    ax,icmd              ;get command current
        cmp    ax,i_hi
       jle    crnt_not_2_hi
        mov    ax,i_hi             ;if current too high
        jmp    crnt_ok             ;set to max
crnt_not_2_hi: cmp ax,i_lo
        jge    crnt_ok             ;if current too low
        mov    ax,i_lo             ;set to min
crnt_ok:  mov    i_act,ax              ;store for theta calc
        mov    cl,al
        mov    bl,delta_i
        add    al,bl
        mov    dx,hi_crrnt_prt
        out    dx,al
        mov    hi_crnt,al
        mov    al,cl
        sub    al,bl
        mov    dx,lo_crrnt_prt
        out    dx,al               ;set window
        mov    lo_crnt,al
        call   theta_calc
;
div_by_zero:                        ;jump here to avoid divide by zero
        pop    dx
        pop    cx
        pop    bx
        pop    ax                  ;restore registers
;       ret
leave:    iret                     ;return from interrupt
;

;pi_control endp
;
;this procedure calculates the advance (delay) angle
;
theta_calc      proc

        fild   i_act               ;calculate advance angle
        fdiv   crrnt_fact           ;get true current from equivalent volts
        fmul   lmin
        fmul   w_fb
```

```
        fidiv  Vd
        fmul   deg_2_rad          ;convert to degrees
        fistp  theta_cap          ;store theta_cap
        fwait
        mov    ax,theta_cap       ;limit the advanced angle
        cmp    ax,theta_cap_hi
        jle    theta_not_hi
        mov    ax,theta_cap_hi
        jmp    theta_ok
theta_not_hi:
        cmp    ax,theta_cap_lo
        jge    theta_ok
        mov    ax,theta_cap_lo
theta_ok: mov    theta_cap,ax      ;store actual value of theta_cap
        mov    al,dec_flg          ;check for deceleration
        cmp    al,01               ;
        jne    no_dec
decelerate:
        mov    dx,mode_1           ;reload mode registers
        mov    al,dly_mode_1
        out    dx,al
        mov    dx,mode_2
        mov    al,dly_mode_2
        out    dx,al
        mov    al,dly_mode_3
        out    dx,al
        mov    ax,dec_delay
        sub    ax,theta_cap
        call   load_cntrs
        jmp    exit_theta
no_dec:  mov    ax,acc_delay
        sub    ax,theta_cap
        call   load_cntrs
exit_theta: ret

theta_calc      endp
;
;this procedure loads angle counters
;
load_cntrs proc

        mov    theta_del,ax
        fild   theta_del
        fdiv   theta_res
        fistp  theta_del
        fwait
        mov    ax,theta_del
        mov    dx,dly_ctr_1
        out    dx,al
        xchg   ah,al
        out    dx,al
```

```
        xchg   ah,al
        mov    dx,dly_ctr_2
        out    dx,al
        xchg   ah,al
        out    dx,al
        xchg   ah,al
        mov    dx,dly_ctr_3
        out    dx,al
        xchg   ah,al
        out    dx,al
        ret

load_cntrs endp
;
;this is interrupt service routine 0 used during start up
;
start_int0:
          mov  slo_flg,01          ;set slo_flag
          mov  int_flag,01         ;set int_flag
          iret
;
;this is interrupt service routine 1 used during start up
;
start_int1: cmp  slo_flg,01          ;if slo_flg is set
         jz    too_slo             ;do nothing
set_fst_flg:
          mov  fast_flg,01         ;set fast_flg
too_slo:    mov  slo_flg,00          ;reset slo_flag
        mov    int_flag,01
          iret
;
; this procedure is called to output a carriage return and line feed
;
crlf     proc

st_chk1:  in     al,stat_port
        and    al,01
        cmp    al,01
        jnz    st_chk1
        mov    al,CR
        out    data_port,al
st_chk2:  in     al,stat_port
        and    al,01
        cmp    al,01
        jnz    st_chk2
        mov    al,LF
        out    data_port,al
        ret

crlf     endp
;
```

```
;this procedur to check input status of crt
;
stat_in_chk proc

status_chkin:
        in     al,stat_port
        and    al,02
        cmp    al,02
        jnz    status_chkin
        in     al,data_port
        ret

stat_in_chk endp
;
;this procedure checks output status of crt
;
stat_out_chk proc

status_chkout:
        in     al,stat_port
        and    al,01
        cmp    al,01
        jnz    status_chkout
        ret

stat_out_chk endp
;
;this is procedure to output messages to screen
;all messages should end with '$' (the almighty dollar !)
;entry conditions : bx <-- start_of_message
;exit conditions  : none
;

message   proc
stat_port equ    0e4h              ;status port address
data_port equ 0e3h                  ;data port address
;
       push   ax                 ;save ax register
stat_read:
            in      al,stat_port  ;;read usart status
            and     al,01         ;mask high seven
            cmp     al,01          ;is transmit enabled?
         JNZ stat_read
         mov     al,[bx]           ;get next character of message
         cmp     al,'$'           ;is it end_of_message?
      JZ quit_msg
         out    data_port,al        ;no, output charater to screen
         inc     bx               ;point to next character
      JMP SHORT stat_read
quit_msg: pop    ax                 ;restore ax
```

```
        ret                     ;return from proc
message   endp                      ;end of procedure
;
;this procedure is to input user_input values
;
;bx contains start address of structure
;
user_value      proc
;
        push   ax               ;save registers
        push   cx
        push   dx
;
        push   bx
        call   message          ;output message
        pop    bx
        mov    ax,0
        mov    cx,0
zero:    mov    [bx].count_after,0   ;initialise to 0
        mov    [bx].ch_flag,0       ;
        mov    [bx].val_input,0
        mov    [bx].ch_count,0      ;
        fild   null
        fstp   [bx].real_val
;
stat_chk1: in    al,stat_port       ;check status for input
        and    al,02h
        cmp    al,02h
        jnz    stat_chk1
        in     al,data_port        ;if ready, input character
        xchg   ah,al
stat_chk2: in    al,stat_port       ;check status for output
        and    al,01h
        cmp    al,01h
        jnz    stat_chk2
        xchg   ah,al               ;if ready, output on crt
        out    data_port,al
        cmp    al,0dh              ;return?
        jnz    no_cr               ;
        jmp    car_ret
no_cr:   cmp    al,30h             ;
        jge    chk_hi
        jmp    point_chk
chk_hi:  cmp    al,39h
        jng    no_error
        jmp    error
no_error: mov    ah,[bx].ch_flag      ;check if after decimal input
        cmp    ah,01
        jz     post_dec
pre_dec: sub    al,30h
        mov    ah,0
```

```
        mov    last_dig,ax
        fild   [bx].val_input
        fimul  ten
        fiadd  last_dig
        fcom   [bx].up_lim
        fstsw  status
        fwait
        mov    al,byte ptr status+1
        and    al,41h              ;
        cmp    al,00
        jz     error
        fist   [bx].val_input
        fstp   [bx].real_val
        fwait
        inc    [bx].ch_count
        jmp    stat_chk1

post_dec: sub    al,30h
        mov    ah,0
        mov    last_dig,ax
        inc    [bx].ch_count
        inc    [bx].count_after
        mov    cl,[bx].count_after
        fild   last_dig
div:    fidiv  ten
        loop   div
        fadd   [bx].real_val
        fcom   [bx].up_lim
        fstsw  status
        fwait
        mov    al,byte ptr status+1
        and    al,41
        cmp    al,00
        jz     error
        fstp   [bx].real_val
        fwait
        jmp    stat_chk1

point_chk: cmp   al,46
        jnz    error
        mov    ah,[bx].ch_flag
        cmp    ah,01
        jz     error
        mov    [bx].ch_flag,01      ;
        jmp    stat_chk1
;
val_2_big:
error:   push   bx
        scrn_msg error_msg          ;output error message
        pop    bx
```

```
        jmp    zero
car_ret:  mov    ah,[bx].ch_count
        cmp    ah,0
        jnz    chk_lo                  ;if no thing input
        FLD    [bx].val_def              ;use default value
        fstp   [bx].real_val
        fwait
        jmp    exit_val
chk_lo:   fld    [bx].real_val
        fcom   [bx].lo_lim
        fstsw  status
        fwait
        mov    al,byte ptr status+1
        and    al,01
        cmp    al,01
        jz     error
exit_val:
        call   crlf
        pop    dx
        pop    cx
        pop    ax
        ret
user_value      endp
;
;this procedure inputs user commands while the motor is running
;
dyn_input proc

        mov    cx,timing_count      ;load cx with timing count
        mov    al,esc_flag
        cmp    al,01              ;if escape flag is set
        jz     input_option         ;input option
        mov    al,opt_flag
        cmp    al,01
        jnz    jump7
        jmp    input_option
jump7:    mov    al,inp_flag
        cmp    al,01
        jnz    stat_chk_1
        jmp    stat_chk_2
stat_chk_1: in   al,stat_port        ;
        and    al,02h             ;wait for input command
        cmp    al,02h
        loopnz stat_chk_1
        jz     jump1
        jmp    exit_input
jump1:    in     al,data_port
        cmp    al,esc
        jnz    stat_chk_1

set_esc_flag:
```

```
        mov    esc_flag,01          ;set esc_flag
output_options:
        scrn_msg  opt_msg           ;output screen message
        jcxz   jump2               ;if time⅛s up exit routine
        jmp    input_option
jump2:  jmp    exit_input

input_option:
        in     al,stat_port
        and    al,02h              ;
        cmp    al,02h
        loopnz input_option
        jz     jump3
        jmp    exit_input
jump3:  in     al,data_port
        mov    esc_flag,00
        xchg   ah,al
opt_2_scrn: in   al,stat_port
        and    al,01
        cmp    al,01
        jnz    opt_2_scrn
        xchg   ah,al
        out    data_port,al
        mov    option,al
        mov    opt_flag,01
        jcxz   jump4
        jmp    chk_opt
jump4:  jmp    exit_input
chk_opt: mov   al,option
        sub    al,30h
        cmp    al,01
        jne    jump9
        jmp    input_speed
jump9:  cmp    al,02
        jz     rev_dir
        cmp    al,03
        jz     hlt_mc
        scrn_msg  opt_err
        mov    esc_flag,01
        mov    opt_flag,00
        jmp    input_option

rev_dir: mov   al,rev_flag         ;set reverse flag
        xor    al,01
        cmp    al,00
        jnz    rev_mc
        mov    rev_flag,al
        mov    al,hi_crnt
        sub    al,delta_i
        mov    ah,0
        mov    i_act,ax
```

```
        call  theta_calc
        mov   al,hi_crnt          ;reload old currents
        mov   dx,hi_crrnt_prt
        out   dx,al
        mov   al,lo_crnt
        mov   dx,lo_crrnt_prt
        out   dx,al
        mov   halt_flag,00
        scrn_msg esc_msg
        ret
rev_mc:  mov   rev_flag,al
hlt_mc:  scrn_msg hlt_msg
        mov   halt_flag,01        ;set halt flag
        mov   al,i_hi
        mov   bl,al
        add   al,delta_i
        mov   dx,hi_crrnt_prt
        out   dx,al
        mov   al,bl
        sub   al,delta_i
        mov   dx,lo_crrnt_prt
        out   dx,al
        mov   dec_flg,01
        mov   i_act,i_hi
        call  theta_calc          ;to load counters with '-ve' current
        ret
input_speed:
        mov   opt_flag,00         ;reset opt_flag
        mov   inp_flag,01
        scrn_msg inp_speed
        jcxz  jump5
        jmp   stat_chk_2
jump5:  jmp   exit_input

stat_chk_2: in  al,stat_port      ;input character
        and   al,02h
        cmp   al,02h
        loopnz stat_chk_2
        jz    jump8
        jmp   exit_input
jump8:  in    al,data_port
        xchg  al,ah
out_char: in    al,stat_port
        and   al,01
        cmp   al,01
        jnz   out_char
        xchg  al,ah
        out   data_port,al        ;output character to screen
chk_char: cmp   al,1bh
        je    abrt_inp
        cmp   al,cr
```

```
        je    end_o_inp
        cmp   al,30h
        jl    err_inp
        cmp   al,39h
        jg    err_inp
        sub   al,30h
        mov   bl,al
        mov   bh,00
        mov   ax,omega_temp
        mov   dx,ten
        mul   dx
        add   ax,bx
chk_val: cmp   ax,3000
        jg    inp_2_lrg
        mov   omega_temp,ax
        jmp   stat_chk_2
abrt_inp: mov   inp_flag,00
        mov   omega_temp,00
        scrn_msg  esc_msg
        ret
err_inp:  scrn_msg  inp_err
        mov   inp_flag,00
        mov   omega_temp,00
        jmp   input_speed
inp_2_lrg: scrn_msg  lrg_inp
        mov   inp_flag,00
        mov   omega_temp,00
        jmp   input_speed
end_o_inp: fild  omega_temp
        fst   omega.real_val
        fmul  rpm_2_rad
        fst   w_set
        fwait
        mov   ax,omega_temp
        mov   omega.val_input,ax
        mov   omega_temp,00
        mov   inp_flag,00
        scrn_msg esc_msg
exit_input:
        ret
dyn_input endp
;
;this is procedure to initialise the 8259 Programmable
;Interrupt controller
;entry conditions : none
;exit conditions  : none
;
pic_init  proc
;
icw1    equ   13h               ;init command word 1
icw2    equ   20h               ;               2
```

```
icw4    equ   0fh              ;              3
pic0    equ   0ff80h           ;8259 port a
pic1    equ   pic0+1             ;        b
;
        push  ax               ;save registers
        push  dx
;
        mov   al,icw1          ;init command word 1
        mov   dx,pic0          ;to port a
        out   dx,al
        mov   al,icw2          ;init command word 2
        mov   dx,pic1          ;to port b
        out   dx,al
        mov   ax,icw4          ;init command word 4
        out   dx,al            ;to port b
;                              (note that icw3 is not issued)
        pop   dx
        pop   ax               ;restore registers
        ret
pic_init  endp

;this is procedure to initialise speed tracking 8253
;this 8253 also provides clock for PAL of bang-bang controller

speed_init      proc
clock   equ   0ff88h           ;divided clock
speed_cntr equ  clock+1          ;position differentiator
speed_clock equ  clock+2         ;spare counter
speed_mode equ   clock+3           ;mode register for chip

mde_wrd_clk equ  36h             ;to set clock register
mde_wrd_spd equ  7ah             ;to set speed register
spd_clk_mde equ  0b6h
div_by_8  equ   08h

        push  ax               ;save registers
        push  dx

        mov   dx,speed_mode
        mov   al,mde_wrd_clk       ;set /n mode for clock reg
        out   dx,al
        mov   al,mde_wrd_spd       ;set hardware triggered mode
        out   dx,al            ;for speed calculation
        mov   al,spd_clk_mde       ;set /n mode for speed_clock
        out   dx,al
        mov   dx,clock         ;clk is
        mov   ax,div_by_8        ; 1/8 of clock of std bus
        out   dx,al
        xchg  ah,al
        out   dx,al
        mov   dx,speed_clock       ;initialise slow clock
```

```
        mov    ax,div_by_n_slo
        out    dx,al
        xchg   ah,al
        out    dx,al

        pop    dx
        pop    ax                ;restore registers
        ret
speed_init      endp
;
;initialise delay and pulse width counters
;
delay_init      proc
      mov    dx,mode_1
      mov    ax,dly_mode_1
      out    dx,al
      mov    al,pls_mode_1
      out    dx,al
      mov    al,pls_mode_2
      out    dx,al
      dec    dx
      mov    ax,pls_wdth
      out    dx,al
      xchg   ah,al
      out    dx,al
      mov    ax,delay
      dec    dx
      out    dx,al
      xchg   ah,al
      out    dx,al
      dec    dx
      mov    ax,pls_wdth
      out    dx,al
      xchg   ah,al
      out    dx,al
;
      mov    dx,mode_2
      mov    al,dly_mode_2
      out    dx,al
      mov    al,dly_mode_3
      out    dx,al
      mov    al,pls_mode_3
      out    dx,al
      dec    dx
      mov    ax,delay
      out    dx,al
      xchg   ah,al
      out    dx,al
      dec    dx
      mov    ax,pls_wdth
      out    dx,al
```

```
        xchg    ah,al
        out     dx,al
        dec     dx
        mov     ax,delay
        out     dx,al
        xchg    ah,al
        out     dx,al

        ret

delay_init      endp
;
;this procedure is to perform diagnostics
;
diagnose  proc

        mov     dx,diagnostics
        in      al,dx
        mov     ah,al
        and     al,01
        xor     al,dir_flag
        cmp     al,00
        jz      dir_ok
wrong_dir: mov  dx,master_dis
        out     dx,al
        scrn_msg dir_wrong
        jmp     warning
dir_ok:  mov    al,ah
        and     al,dia_mask
        cmp     al,00
        jz      a_ok
not_ok:  mov    dx,master_dis
        out     dx,al
        mov     ah,al
        and     al,02
        cmp     al,00
        jnz     over_crnt
over_volt:
        scrn_msg over_voltage
        jmp     warning
over_crnt: scrn_msg over_current
warning:  call  flash
        jmp     begin

a_ok:    ret

diagnose  endp
;
;this subroutine flashes lights when problems are detected
;
flash    proc
```

```
        mov    dx,lights_port
        mov    al,00
counter: xor    al,0ffh
        mov    cx,0ffffh
        out    dx,al
        push   ax
spot1:   in     al,stat_port
        and    al,02
        cmp    al,0                ;if keypad has been hit
        jnz    exit_flash          ;exit flashing
        loop   spot1
        pop    ax
        jmp    counter

exit_flash: mov  al,0ffh
          out  dx,al               ;turn off lights
          pop  ax
          ret

flash     endp
;       this procedure is entered during halting at 6rpm
;
halt      proc

        cli
        mov    ax,halt_clk
        mov    dx,speed_clock
        out    dx,al
        xchg   ah,al
        out    dx,al
        mov    dx,speed_cntr
        mov    ax,start_count
        out    dx,al
        xchg   ah,al
        out    dx,al
        mov    slo_flg,00
        sti
        hlt
loop_hlt: call   dyn_input
        call   diagnose
        cmp    halt_flag,01
        jnz    undo
        cmp    int_flag,01
        jnz    loop_hlt
        mov    al,slo_flg
        cmp    al,01
        jnz    loop_hlt
        cli
        mov    halt_flag,00
        mov    al,rev_flag
```

```
        cmp    al,01
        jnz    reset
        mov    al,mask_int1
        mov    dx,pic1
        out    dx,al              ;mask int1
        mov    al,dir_flag
        xor    al,01
        cmp    al,00
        jz     rev_seq
for_seq: mov    dx,forward_port      ;reversing to for phase sequence(1-
        out    dx,al
        jmp    chk_dir
rev_seq: mov    dx,reverse_port      ;reversing to rev phase seq(1-2-3)
        out    dx,al              ;set reverse direction signals
chk_dir: sti
        hlt
        mov    dx,diagnostics
        in     al,dx
        and    al,mask_dir
        xor    al,dir_flag
        jz     chk_dir
        xor    al,01              ;reverse direction flag
        mov    dir_flag,al
        mov    rev_flag,00          ;reset reverse flag
undo:    ret
                                ;return
reset:   pop    ax                 ;clear stack
        pop    ax
        mov    dx,master_dis
        out    dx,al
        jmp    begin

halt     endp
;

;
;this procedure does the  starting of the machine upto 50 rpm
;
start_up  proc

restart:  mov    fast_flg,00          ;reset fast_flg
        mov    al,00
        mov    dx,pic1
        out    dx,al              ;enable all interrupts
        mov    slo_flg,00           ;preset slo_flg
        mov    dx,speed_cntr
        mov    ax,start_count
        out    dx,al              ;initialise counter with 10000
        xchg   ah,al
        out    dx,al
        mov    ax,offset start_int0
```

```
        add    ax,200h              ;initialise interrupt addresses
        mov    cs:80h,ax
        mov    ax,offset start_int1
        add    ax,200h
        mov    cs:84h,ax
        mov    al,lo_spd_crnt_hi
        mov    dx,hi_crrnt_prt
        out    dx,al
        mov    al,lo_spd_crnt_lo    ;set current requirement
        mov    dx,lo_crrnt_prt
        out    dx,al
        mov    dx,start_mode_port
        out    dx,al                ;set mode to start
        mov    dx,master_en
        out    dx,al                ;enable controller outputs
        mov    pi_flag,01
;motor    should start now
;if it does not say your prayers
;(and yes, check if the power to motor is on)
        scrn_msg esc_msg            ;esc_msg
still_too_slo: sti
loop_start: call dyn_input
        call   diagnose             ;
        cmp    halt_flag,01
        jz     start_exit
        cmp    int_flag,01
        jnz    loop_start
        cli
        mov    int_flag,00
        mov    al,fast_flg
        cmp    al,01
        jnz    still_too_slo
        mov    dx,pic1
        mov    al,mask_int1
        out    dx,al                ;mask  interrupt 1
        mov    ax,offset start_serv
        add    ax,200h
        mov    cs:84h,ax
        mov    ax,count
        mov    dx,speed_cntr
        out    dx,al
        out    dx,al
        mov    halt_flag,00
start_exit:
        ret
start_up  endp
;
;this is low speed mode (10rpm < speed < 50 rpm )
;
low_speed proc
```

```
        sti
        scrn_msg esc_msg            ;esc_msg
loop_lo1: call  dyn_input
        call  diagnose
        cmp   int_flag,01
        jnz   loop_lo1
        mov int_flag,00
lo_spd_mode: sti                    ;enable interrupts
loop_lo:  call  dyn_input
        call  diagnose
        cmp   int_flag,01
        jnz   loop_lo
        cli                     ;disable interrupts
        mov   int_flag,00
        mov   ax,sample_new         ;if count is above 7fffh
        cmp   ax,00               ;then too slow
        jl    speed_up
        cmp   ax,2000             ;if 50 rpm or more
        jle   to_norm_mode          ;then go to normal mode
        cmp   ax,16666            ;if between 6 and 50 rpm
        jl    lo_spd_mode         ;then stay in low speed mode
speed_up: mov   al,00h              ;else
        mov   dx,pic1             ;re-enable interrupt 1
        out   dx,al
        mov   al,halt_flag
        cmp   al,01
        jz    no_msg
        scrn_msg  slowing_down      ;tell operator something's wrong
no_msg:  mov   slo_flg,01
        mov   fast_flg,00
        jmp   exit_lo_mode
to_norm_mode:
        mov   pi_flag,00          ;enable pi control
        mov   dx,speed_clock
        mov   ax,div_by_n_med
        out   dx,al
        xchg  ah,al
        out   dx,al
        mov   fast_flg,01
        mov   slo_flg,00
        fld   kr1
        fstp  kr                ;move kr1 to kr
        fld   k1
        fmul  k_int.real_val
        fstp  ki_mod
        fwait
        mov   dx,norm_mode_port     ;switch to normal mode
        out   dx,al
exit_lo_mode:
        ret
low_speed endp
```

```
;
;this procedure is for the medium speed mode (50rpm < speed < 500rpm)
;
med_speed proc
        scrn_msg esc_msg
        sti
        hlt
        mov    int_flag,00
loop_med: sti
        call   dyn_input
        call   diagnose
        cmp    int_flag,01
        jne    loop_med
        cli
        mov    int_flag,00
        mov    ax,sample_new
        cmp    ax,00
        jl     back_to_lo_speed
        cmp    ax,12500
        jg     back_to_lo_speed
        cmp    ax,1000
        jl     to_hi_speed
        jmp    loop_med
back_to_lo_speed:
        mov    dx,speed_clock
        mov    ax,div_by_n_slo      ;change clock
        out    dx,al
        xchg   ah,al
        out    dx,al
        mov    slo_flg,01           ;set slow flag
        mov    fast_flg,00          ;reset fast flag
        ret
to_hi_speed: mov  dx,speed_clock       ;change clock
        mov    ax,div_by_n_fast     ;
        out    dx,al
        xchg   ah,al
        out    dx,al
        mov    slo_flg,00           ;reset slow flag
        mov    fast_flg,01          ;set fast flag
        fld    kr2
        fstp   kr                   ;move kr2 to kr
        fld    k2
        fmul   k_int.real_val
        fstp   ki_mod
        fwait
        ret
med_speed endp
;
;this procedure is for high speed mode (450 < speed < 3000 )
;
hi_speed proc
```

```
        scrn_msg  esc_msg          ;
        sti
        hlt
        mov    int_flag,00
loop_hi: sti
        call   dyn_input
        call   diagnose
        cmp    int_flag,01
        jne    loop_hi
        cli
        mov    int_flag,00
        mov    ax,sample_new
        cmp    ax,00
        jl     back_to_med_speed
        cmp    ax,13899
        jg     back_to_med_speed
        cmp    ax,2000
        jl     speed_2_hi
        jmp    loop_hi
back_to_med_speed:
        mov    dx,speed_clock
        mov    ax,div_by_n_med
        out    dx,al
        xchg   ah,al
        out    dx,al
        mov    slo_flg,01
        mov    fast_flg,00
        fld    kr1
        fstp   kr                  ;move kr1 to kr
        fld    k1
        fmul   k_int.real_val
        fstp   ki_mod
        fwait
        ret
speed_2_hi:
        mov    slo_flg,00
        mov    fast_flg,01
        ret

hi_speed  endp
;
;main program
begin:  mov     ax,data            ;initialise segment registers
       add     ax,20h
       mov     ds,ax
       mov     ax,0f8h
       mov     ss,ax
       mov     sp,offset stack_top
       mov     dx,master_dis       ;
       out     dx,al              ;issue master disable
```

```
      scrn_msg v_msg              ;give version message
;wait
contin:  scrn_msg con_msg          ;

esc_chk: call  stat_in_chk        ;check if input status ready
       cmp   al,esc
       jnz   esc_chk
;

;user asked to input parameters

parm_input: scrn_msg instr_msg       ;give instructions
       input  k_prop             ;kp
       input  k_int             ;ki
       input  k_torque            ;kt
       input  omega               ;speed command

       fld   omega.real_val
       fmul  rpm_2_rad
       fstp  w_set
       fwait
dir_inp:  scrn_msg dir_msg          ;ask for direction
dir_chk:  call  stat_in_chk
       xchg  ah,al
out_dir: call   stat_out_chk
       xchg  ah,al
       out   data_port,al
       cmp   al,CR
       jz    set_for
       cmp   al,30h
       jle   dir_inp
       cmp   al,32h
       jg    dir_inp
       cmp   al,31h
       jne   set_rev
set_for:  mov   dir_flag,00        ;reset direction flag
       mov   dx,forward_port      ;set forward direction
       out   dx,al
       jmp   wait_esc
set_rev:  mov   dir_flag,01        ;set direction flag
       mov   dx,reverse_port      ;set reverse direction
       out   dx,al
wait_esc: scrn_msg wait_msg
wait_chk: call  stat_in_chk
       cmp   al,esc
       jnz   wait_chk
;
;initialise variables and constants and flags to zero
      finit                    ;initialise 8087
       mov   esc_flag,00
       mov   opt_flag,00
```

```
        mov    rev_flag,00
        mov    ax,00
        mov    icmd,ax
        fild   null
        fst    kr
        fst    sum_ei
        fst    ki_mod
;
;initialise the various counters and the pic
;
        call   pic_init
        call   speed_init          ;initialize speed tracking counter chip
        call   delay_init
;
;call start_up routine
;
rev_up:   call start_up
        cmp    halt_flag,01
        jnz    lo_mode
        call   halt
        jmp    rev_up
lo_mode:  call low_speed              ;low speed mode
        mov    al,halt_flag
        cmp    al,01
        jnz    no_halt
        call   halt
        jmp    rev_up
no_halt:  mov    al,slo_flg           ;if motor is slowing down
        cmp    al,01                ;rev it up
        je     rev_up
med_mode: call   med_speed
        mov    al,slo_flg
        cmp    al,01
        je     lo_mode
        call   hi_speed
        mov    al,slo_flg
        cmp    al,01
        je     med_mode
        scrn_msg  too_fast
        jmp    begin
;
code    ends
      end     begin
```

B.2 The PAL program

```
PAL16R4                                          PAL DESIGN SPECS
PAT001                                                 AMEESH OZA
BANG-BANG CONTROLLER PAL
SRM CONTROLLER
PCLK P1' P2' P3' DR' NC NC NC NC GND
OE' NC NC NC EN1' B&C' T PHI' NC VCC
;THESE ARE PIN DEFINITIONS

;OUTPUT EQUATIONS

/T := PHI'*/T  +
     PHI'*/DR'  +
     T*/DR'

/B&C' := PHI'  +
       B&C'  +
 T

/EN1' := /PHI'*/DR'*EN1'

/PHI' = /P1' + /P2' + /P3'


DESCRIPTION:
```

Appendix C. Circuit Details

This appendix gives detailed circuit diagrams for reference and debugging of the circuit. The circuit has been divided into three sections to enable ease of drawing.

1. The STD Bus interface circuit
2. The Angle Control and Position Sensing circuits
3. The Chopping control circuit

The first deals with the interface chips like buffers, inverters and decoders. The second one deals with the angle control circuit and position circuit, descriptions of which are given in chapter 4. The last one is the current chopping circuit used to control phase currents.

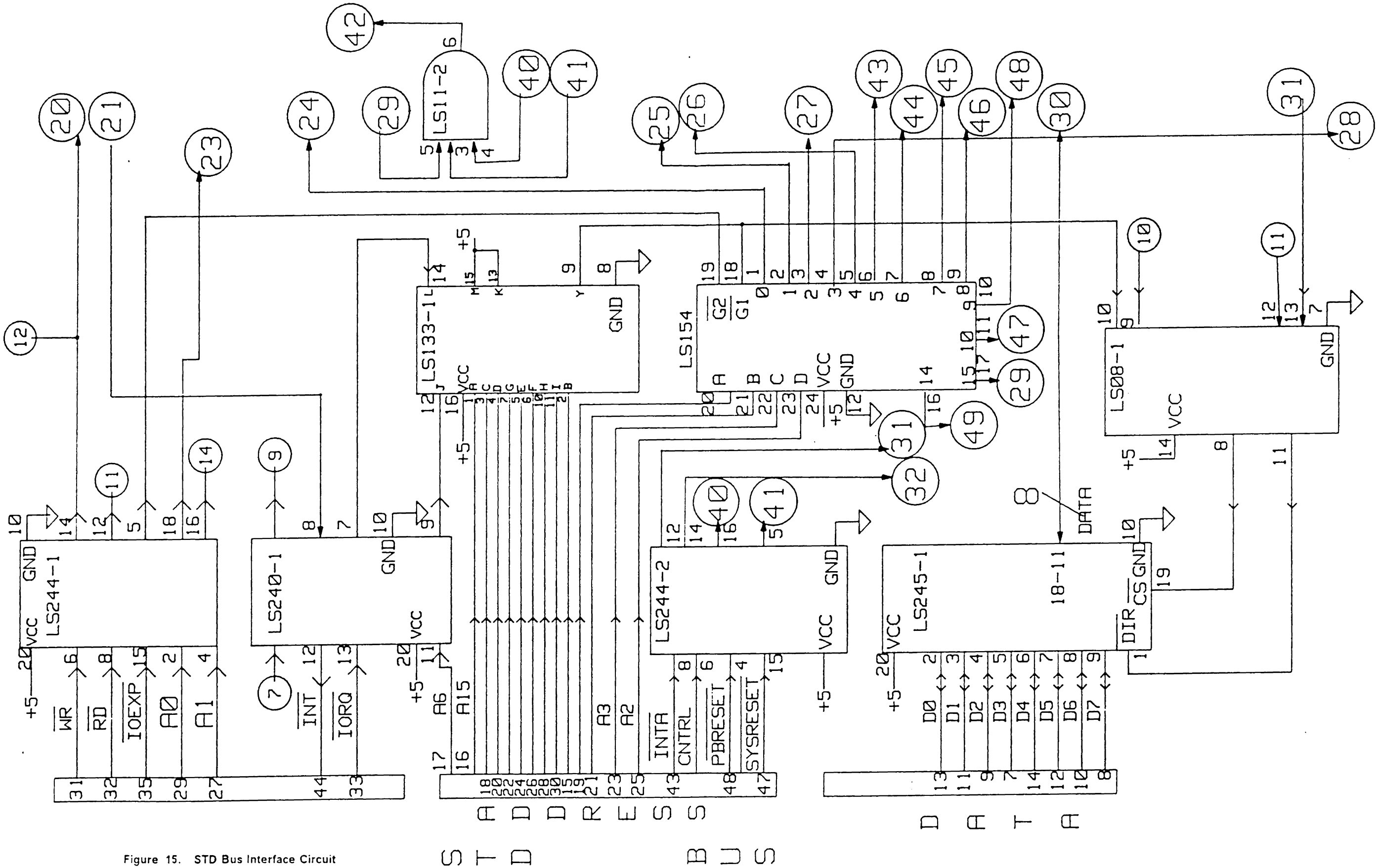


Figure 15. STD Bus Interface Circuit

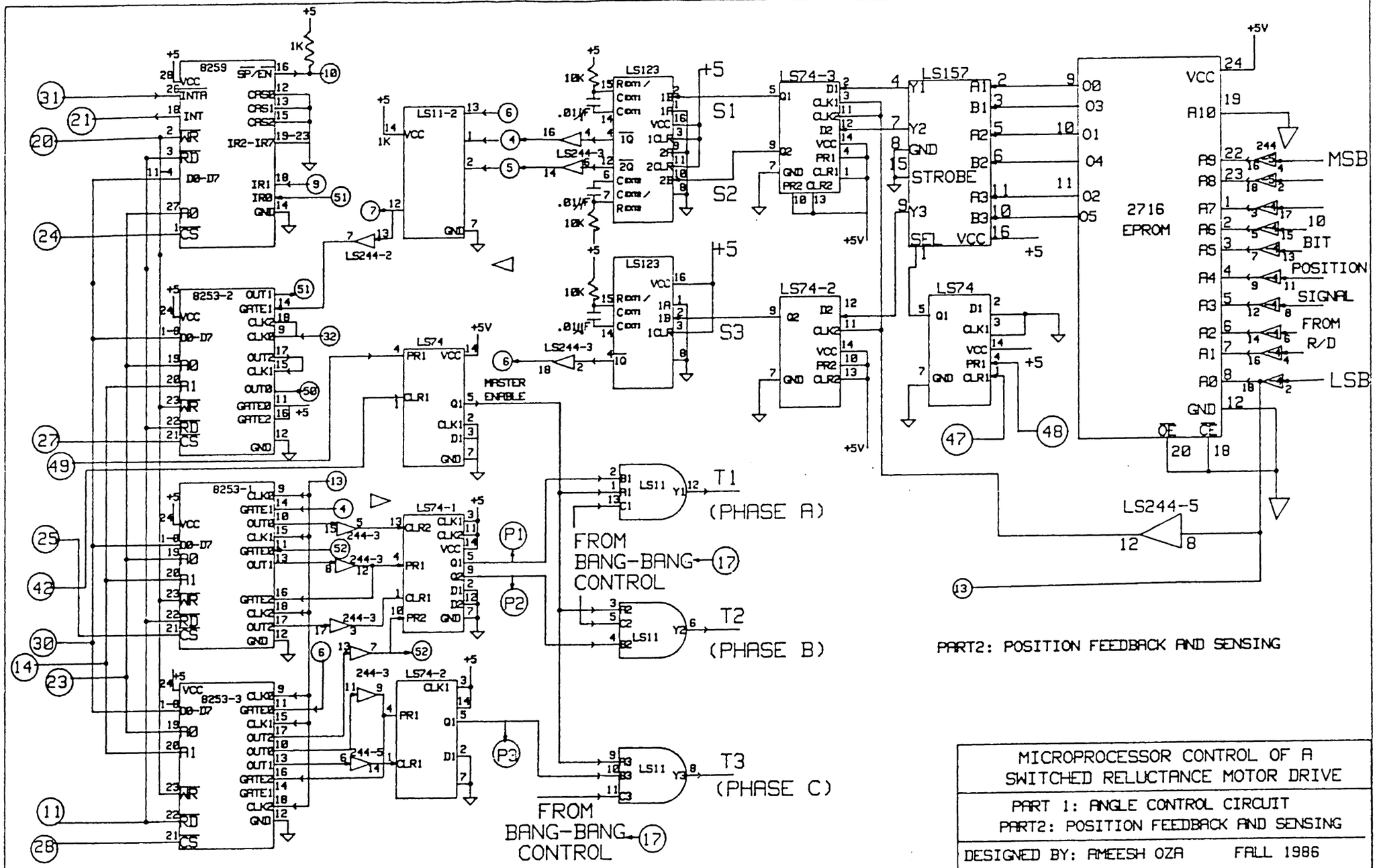


Figure 16. Angle Control and position sensing circuits

NOTES:

1: INPUTS ARE 10 BITS FROM ROTOR POSITION SENSORS

2: OUTPUTS ARE T1,T2,T3 TO BASE DRIVE CIRCUITS

3: ALL PINS NOT SHOWN ARE NOT CONNECTED

MICROPROCESSOR CONTROL OF A SWITCHED RELUCTANCE MOTOR DRIVE	
PART 1: ANGLE CONTROL CIRCUIT PART2: POSITION FEEDBACK AND SENSING	
DESIGNED BY: AMEESH OZA	FALL 1986
DRAWN BY: AMEESH OZA	NOV 10, 1986
REVISION 1: AMEESH OZA	9 MAR 1987
REVISION 2: AMEESH OZA	3 SEP 1987
VA TECH MOTOR DRIVES GROUP EE DEPT, VA TECH, BLACKSBURG, VA 24061.	

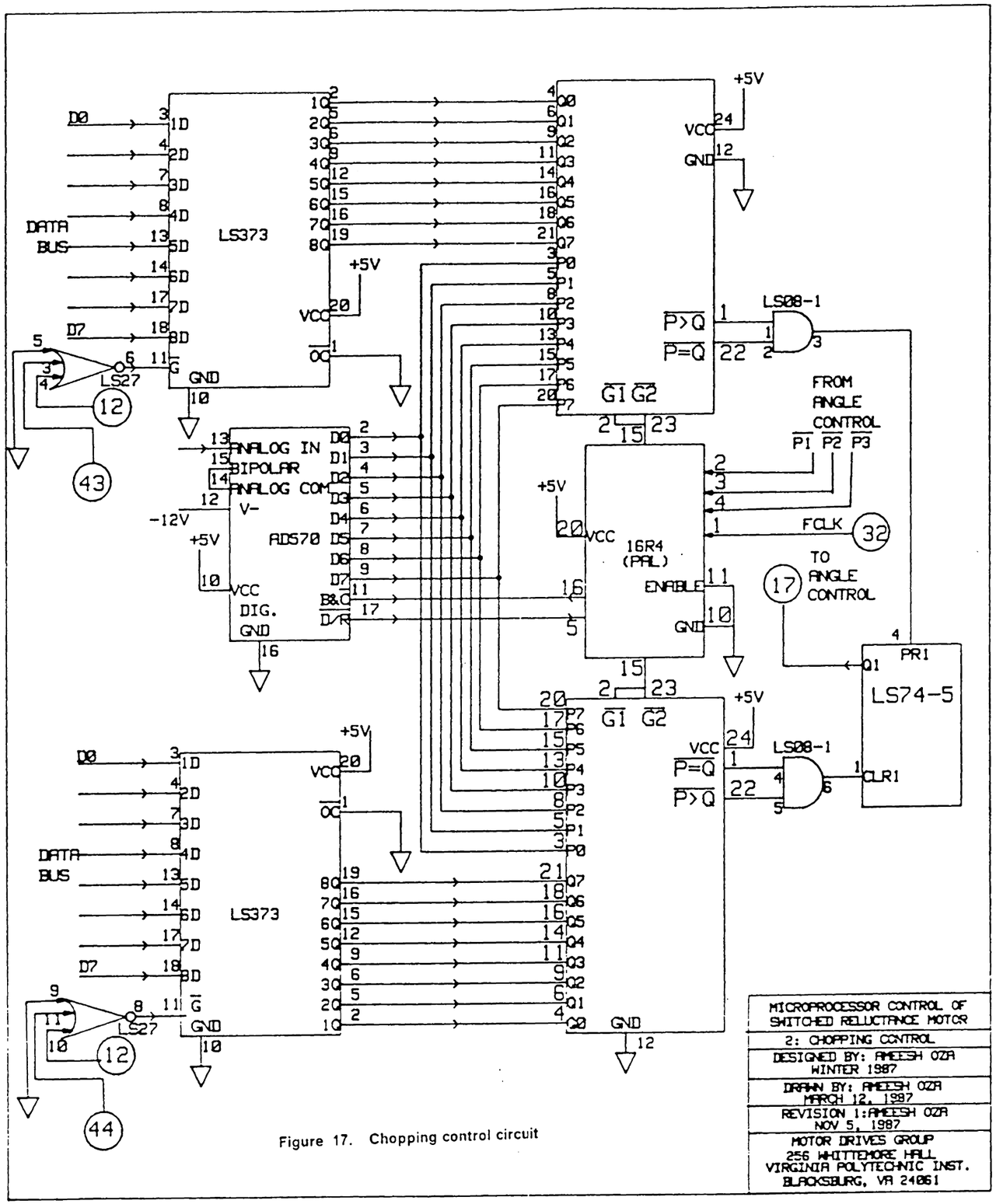


Figure 17. Chopping control circuit

The vita has been removed from the scanned document