A MICROCOMPUTER CONTROLLER FOR A NYLON SPINNING MACHINE

by Terence Enfield Kirk

Submitted to the University of Capetown in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

September 1985

ABSTRACT

This thesis will show how a new type of controller for a Nylon spinning machine was developed from an initial specification. The controller is a component in a loosely coupled feedback system which reads two tachometer pulse trains and various plant interlocks, and produces two pulse trains which are used to control two solid state variable frequency variable voltage inverters and their AC motors. The specification calls for 24 controllers to be linked to a PDP 11/23 host computer which holds a library of operating parameters which can be downloaded into each control unit by an operator.

After examining the requirements of the system, a microcomputer implementation was chosen as best meeting the needs of the project. Elsewhere in the plant several earlier attempts at using micro-computers as dedicated controllers had been made, with rather poor results. Consideration of the future requirements of the company showed that there was a clear role for these controllers, and it was clear that there was a need to define standards for their development and implementation, and so a survey of the company's requirements was done, on the basis of which a standard was adopted.

The thesis covers all system related aspects of the project, from the initial selection of a microcomputer system and software development system, to the design and implementation of the controller.

ACKNOWLEDGEMENTS.

At the start of this project it was estimated that there were approximately two person years of work to be done. The project had to be commissioned within eight months, so it was necessarily a team effort. My role was that of system designer. Since there were no established standards for microcomputer systems at SANS, this role included a survey and assessment of the company's microcomputer requirements, and the establishment of standards, as well as designing and developing the application hardware and software.

I had invaluable assistance in all aspects of the software design and implementation from Dave Thalrose of the computer applications group at SANS, who also wrote the host communications module, and explained the operation of the SMT operating system to me. Don Glass had overall responsibility for the project, and made sure that all phases of the project fitted together smoothly, but still managed to find time to write the OCP module. Michel Malengret adapted the sequence control module from the Type 30 Conmac software, and Karsten Rapmund designed and built the prototype I/O board. Len Baxter and Dereck Gray built the Cabinet for the control units, as well as assisting with the assembly and testing of the computer drawers. Annelie Faure of the SANS technical library was very helpful in locating technical data required for the project.

I would not have been able to complete this work without the financial assistance provided to me by the University in the form of the J.W. Jagger scholarship, as well as the monthly remittance from SANS.

Finally I would like to thank Professor Braae of the Electrical Engineering department for the prompt and helpful supervision of the final drafts of this thesis.

I N D E X

INDEX

APPENDICES

A) Introduction to Nylon Spinning.

B) Production Specification for Machine 5B controller.

C) Overview of Inverter Control Unit System.

D) Specification of operating environment for computers at SANS.

E) Comparison of currently available microcomputer systems suitable for use in real time control applications.

F) Communications protocol for an RS-422 multidrop link between the host and ICU.

G) Software listings.

H) Hardware schematics and jumper allocations.

ILLUSTRATIONS

CHAPTER 1 : INTRODUCTION

1.1 INTRODUCTION.

The aim of this project was to control the speed of the traverse and winder motors on a Barmag High speed wind up head, according to the measured ratio of speeds of the traverse motor and a chuck driven by the winder motor. For more details see the introduction to Nylon Spinning given in Appendix A. The rest of this thesis will assume familiarity with these ideas. A glossary of terms and abbreviations is given in chapter 6.

Because of increased demand for certain types of yarn, and in anticipation of future demand, SANS decided to convert an existing Type 18 Terylene spinning machine into a Type 18 Nylon Spinning machine. One part of the project called for the installation of Barmag high speed winders on the windup floor. The control of the winder drives forms the subject of this thesis. Existing machines use two inverters to control all 24 traverse and winder drives. Inverter failure means that all 24 positions are stopped. With the advent of low cost, solid-state, variable speed AC drives, it seemed that it might be financially viable to use individual inverters for each position, and it was clear that an investigation was necessary.

Once the question of individual inverters on each position had been raised, the question of inverter controllers for each position also had to be considered. Assuming that a sophisticated controller able to give a programmable traverse frequency modulation could be produced, the requirements for the system would be as set out in the next section.

1.2 REQUIREMENTS.

1) Reduce the amount of lost production time through inverter failure, without increasing the amount of maintenance time because of a greater number of inverters and hence inverter failures.

2) Extend the range of products that can be spun with the Barmag winder. The old combination of controller and winder could only operate over a limited range, which restricted the number of products which could be spun.

3) Improve the package build of spun cakes. Research carried out by Hudgell, Sykes and others (ref 1, 2, 3, 4) indicated that improved modulation and banding avoidance techniques would reduce or even eliminate ribboning and banding problems which are responsible for wasted production because of yarn takeoff problems.

4) Simplify the updating of operational instructions to a winding position. The controllers on existing machines are analog devices which use potentiometers and switches to set operational parameters. After a machine has been set up for a new product, it usually takes several further fine adjustments to get a satisfactory standard of output - a wasteful process.

5) Monitor the operation of each position on the machine to check whether it is within preset limits, and generate an alarm if operation moves outside those limits. The idea here was to monitor the difference between desired and actual motor speeds to detect bearing failures and log unscheduled production stoppages.

INTRODUCTION

The production department conducted an investigation which revealed that the break even point for cost per inverter against lost production time through inverter failure would be about one inverter for five positions. Factors 2) to 5) called for individual inverters, and the potential production gains from having these features, coupled with the low cost of the inverters made it possible for the machine to pay for itself within the company's stipulated 2 year payback period for projects of this nature. Thus a decision was made to use two inverters and one controller per position.

This decision had other beneficial offshoots. The limited local market for certain products and the inability of the old system to produce more than one product at a time meant that some products required short runs to satisfy demand. Incorrect setting up and faulty operation of the analog controllers is one of the major sources of wasted production in similar systems elsewhere in the plant. The ability to split the machine and control individual positions means that low demand products can be set up on a few positions and left running for longer periods, thus increasing the conversion efficiency of the machine. Individual control also enables some units to be used for pre-production trials while normal production continues on the rest of the machine (it had been the practice in the past to stop the machine and wind on only a few heads while trials were in progress).

One aspect of individual control that was of particular interest was the requirement in 3) for banding avoidance or "Ribbon Breaking", which can only be done on the basis of chuck and traverse speed measurements and individual position control. Trials had been done at Barmag in Germany, FII in Canada and ICI in the UK which indicated that banding avoidance leads to improved package build and fewer customer returns (ref 1). The three systems all use dual inverters on the traverse, with some method for switching from one to the other when a ribbon point is detected. However the dual inverters added to the cost of the system, and considerable problems were experienced with switching from one inverter to the other. In addition there were problems achieving the resolution of control necessary for these schemes to work. The scheme used by SANS overcomes all of these problems.

1.3 THE INVERTER CONTROL UNIT.

Once the general requirements for the controller had been decided on, and the SANS production department had produced a specification for the controller operation (appendix B), it was necessary to describe the required behaviour of the controller exactly so that a design could be produced. The functions to be performed by the controller are described in appendix C.

The design of the controller was determined by the following factors :

1) The need for highly accurate speed control and measurement to keep yarn tension within closely defined tolerances (See Appendix B for a specification of operational parameters).

2) The need for quick, reliable accurate and repeatable alteration of operational parameters.

3) The lack of hardware development facilities at SANS. Past experience has shown that project completion dates, in- service reliability and cost are improved by buying board level components from reliable vendors. In-house design and construction is normally only justified when there is no commercially available system to perform the task, or when the scale of the project leads to savings on component count through hardware optimisation (because ready-made boards usually incorporate features which are unnecessary for a particular project, but give the greatest overall flexibility).

4) The controller has to perform a complex sequence of operations depending on the current status of the machine.

5) The controller had to be able to incorporate future improvements and changes to the machine. It was also desireable that it should be usable in similar but different sytems.

There are no ready-made analog controllers which are able to perform this control function. Given the range and complexity of the functions of the controller, and after investigating the ability of a micro-computer to perform the necessary tasks, and given the relutance of SANS to get involved in the development of component level systems, the decision was taken to develop a microcomputer controller rather than a custom made analog device. This decision meant that a whole range of ready-made systems became available.

1.4 SELECTING A REAL TIME MICROCOMPUTER SYSTEM.

Having decided to use a microcomputer, it then became necessary to choose the microcomputer hardware, operating system, and a development system to implement the controller. There were two factors which had to be considered in making the choice.

The first is that there are already a number of microprocessor controllers in operation at SANS, and it is expected that many more will be installed in future. These sytems have several disadvantages. The first is that they are mostly dedicated, with software in ROM and no source listing or development facility. This means that it is difficult to maintain and tailor systems for particular applications. In addition there are a number of different systems, which means that a wide variety of spares have to be kept, and more importantly development, maintenance and operational staff have to be familiar with a wide range of equipment. This is undesirable because of the training and familiarisation time needed for each new item. There was a clear need to try and rationalise the situation, and develop a standard for microcomputer and development systems. The system chosen had to be able to cope with all the present and projected needs of the company, and as far as possible had to be compatible with existing equipment, which implied a careful review of the company's present and future requirements for dedicated computer control. Appendix D is the result of this enquiry.

The second factor that had to be considered was the wide variety of board level computer equipment currently available on the market. Each system has drawbacks and advantages, which had to be seen in the light of the company's present and future needs. The various systems that were examined and the aspects of each that were used for comparison are presented in Appendix E.

On the basis of these enquiries, the standards adopted were: INTEL single board computers for target controller systems; and the MAGIC development package running on DEC PDP-11 minicomputers for application software development.

CHAPTER 2 : THE INVERTER CONTROL UNIT HARDWARE

2.1 THE SINGLE BOARD COMPUTER.

The reasons for using the INTEL iSBC 88/25 single board computer were dealt with in chapter 1.4. This chapter will deal with the hardware on the board, and describe how it is used to control the inverter control unit. A brief description of the single board computer will be given, followed by a detailed description of the use of the component parts of the board in this project.

2.1.1 THE iSBC 88/25 SINGLE BOARD COMPUTER.

A block diagram of the computer is given in Figure 2.1.1. The basic board contains : an 8 level Programmable Interrupt Controller (PIC); a Programmable Peripheral Interface (PPI) giving three eight bit parallel ports; a three channel Programmable Interval Timer (PIT); a Universal Synchronous / Asynchronous Receiver / Transmitter (USART) for serial communication with a local terminal; 4 kilobytes of RAM, with sockets for another 4 kilobytes; and four sockets for 2, 4, 8 or 16 kilobyte ROMs, giving a maximum capacity of 64 kilobytes of EPROM or ROM.

Because of the need for a second serial port for host computer communications and more interval timers, an iSBX 351 piggy back board was added to the mother board via one of the local bus iSBX

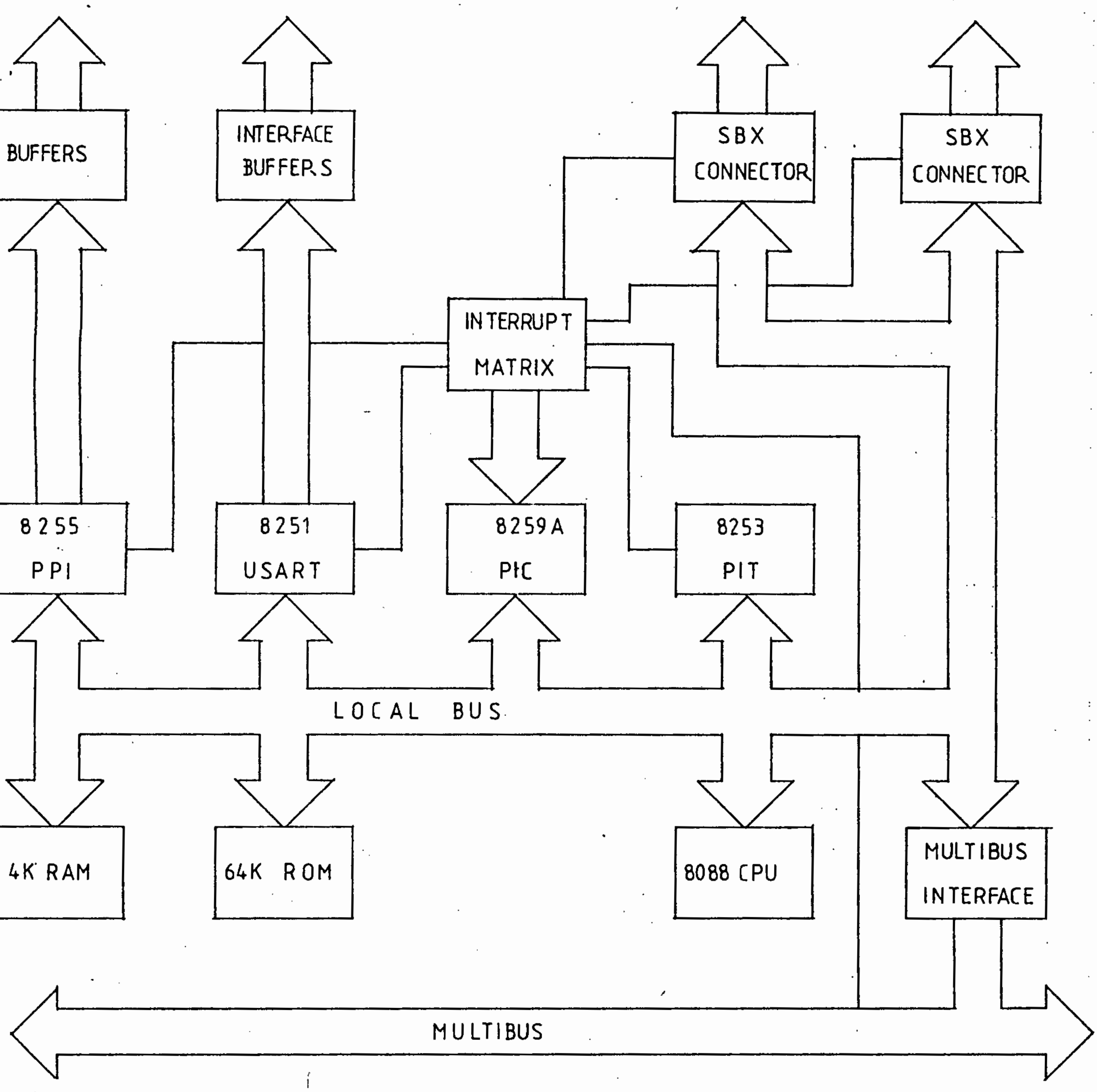


FIGURE 2.1.1: BLOCK DIAGRAM OF SBC

connectors. This board provides an extra USART with buffers for RS-232 or RS-422 (differential multidrop) serial communications, and an extra PIT, which is usually used to clock the USART.

Note that a more general term for Progammable Interval Timer (or PIT) is Real Time clock (or RTC). Both terms are used interchangeably in this thesis. Figures H.1 and H.2 in Appendix H contain full schematic circuit diagrams for the two boards.

2.1.2 USE OF PROGRAMMABLE INTERVAL TIMERS.

A requirement of the controller was that speed control be smooth, avoiding sudden speed changes (except intended changes such as P-jumps). The 8253 PIT was carefully checked for glitching at terminal counts, using a high speed logic analyser. It was selected as a candidate for timing applications only when it had been shown to have smooth, glitch free operation. There are two PIT's in the system (one on the mother board and one on the piggy back board), giving a total of six timers. The system real time clock requirements were as follows :

1 x	50Hz system clock.
2 x	USART baud rate clocks.
1 x	Traverse inverter pulse train.
1 x	Winder inverter pulse train.
1 x	Traverse tachometer.
1 x	Chuck tachometer.

Total 7	

The use of each PIT channel will now be considered in more detail.

2.1.2.1 USART BAUD RATE GENERATION.

Since both USART's had to run at the same baud rate it was possible to use one channel of the PIT for both USART clocks, which meant that six PIT channels were required in all. This need could be satisfied by the two existing 8253's. The USART requires a square wave clock for the serial / parallel shift register. To work out the frequency of the clock, the requirements of the system and the operation of the USART had to be considered. A baud rate of 9600 was required, and the PIT is clocked at 1.229 MHz. The USART has an internal divider which divides the clock by 1, 16 or 64 to give the baud rate. 1.229 MHz has to be divided by approximately 128 to give a baud rate of 9600. The USART division was arbitrarily set to 16, which meant that the PIT had to divide the basic clock by 8. Thus the PIT was set up to operate in mode 3 (square wave generator), with a count value of 8.

2.1.2.2 SHAFT SPEED MEASUREMENT.

The specification called for the chuck speed and the traverse speed to be measured. There are three common techniques for measuring shaft speeds with a computer : Analog to digital conversion of the output of a tacho generator; shaft encoders; and counting of pulses from a shaft rotation transducer (eg proximity detector) over time. Each method has its advantages and drawbacks. In this project the last method was used for four reasons :

A) Analysis showed that the pulse counting method was capable of producing the required measurement accuracy.

B) The PIT had channels available for this purpose, providing a simple and cheap method for speed measurement, the only extra hardware required is for buffering and filtering.

C) Speed measurement had to be done to a high degree of accuracy. If A-D or shaft encoding techniques had been used, high quality (and therefore expensive) hardware would have been required.

D) There was very little space on the machine to mount a tachometer or encoder, and the simplicity, small size and reliability of proximity detectors offered reduced maintenance overheads.

2.1.2.3 INVERTER PULSE TRAIN GENERATION.

The controller had to provide variable speed control for two inverters. Because of the requirement for highly accurate speed control, pulse driven inverters were used rather than the more usual reference voltage control. This meant that Emerson had to modify one of their standard inverters to accept a pulse train input (See appendix C for a description of the inverter operation). If a PIT channel was to be used to generate the pulses, it had to be able to produce the full range of operating frequencies with the required accuracy. Two factors had to be considered if this method was to work. The first is that the inherent resolution of the PIT decreases as frequency of operation increases, which had to be checked against the production requirements. The second factor is that there is a maximum count value that can be loaded into the PIT, which limits the low frequency operation of the PIT. Each of these factors will now be considered in detail.

A) HIGH FREQUENCY RESOLUTION OF THE PIT.

The traverse operates at a higher frequency than the winder, its maximum frequency being 320 Hz. The control pulse train had to be six times the required output frequency of the inverter, so the maximum output frequency required was :

$$6 * 320 \text{ Hz} = 1920 \text{ Hz}.$$

The specification called for a 0.2 Hz resolution, which translates into 1.2 Hz from the PIT. In Mode 3 the 8253 operates as a square wave generator. A count value loaded into the PIT is decremented by clock pulses from a crystal oscillator. When the count has reached half its value the state of the output is changed. When the count reaches zero the state of the output is changed back, and the count value is automatically reloaded into the counter.

Thus :

$$\text{SQUARE WAVE FREQUENCY} = \frac{\text{CRYSTAL CLOCK FREQUENCY}}{\text{COUNT VALUE.}}$$

The frequency resolution of the generated square wave is limited by the integer count value loaded into the PIT. The resolution decreases as the output frequency approaches the PIT clocking frequency. Output frequency is controlled by loading different count values into the PIT. The factory default clock rate is 1.229 MHz, thus :

$$\text{MINIMUM COUNT VALUE} = \frac{1\ 229\ 000}{1920} = 640$$

The smallest change in frequency that can be achieved at this frequency is by increasing the count value from 640 to 641. This corresponds to a frequency change from 1920 to 1917.3 Hz, ie 2.7Hz, which does not meet the specification. Fortunately it was possible to double the PIT clock rate, as the 8253 will operate up to 2.5MHz, and there was a 2.456MHz clock available. This increased the minimum count to 1280, and the worst case resolution is 0.225Hz, which was acceptable to the Production Department.

B) MAXIMUM COUNT VALUE THAT CAN BE LOADED INTO PIT.

The count value loaded into the PIT is a 16 bit integer, so :

$$\text{MINIMUM FREQUENCY} = \frac{2\ 456\ 000}{65\ 536} = 37.5\ \text{Hz.}$$

Ramping of the motors down to a standstill was achieved by using the expression :

NEW SPEED = OLD SPEED - DECELERATION RATE

If the old speed minus the new speed results in a count value requiring more than sixteen bits, then the high order bits above sixteen will be lost (ie the count "wraps around" from 65 536 to zero), and the resulting count value will be incorrect. To ensure that this situation never arises, the minimum speed should be set so that the count value of new speed is never greater than 65 536 :

OLD SPEED = NEW SPEED + DECELERATION RATE

expressing this in terms of the count value :

$$\frac{Fi}{Nold} = \frac{Fi}{Nnew} + MAXDEC$$

Where : Fi = clock rate (2.456 MHz)

Nold = previous count value

Nnew = new count value

MAXDEC = maximum deceleration rate

Rearranging :

$$Nold = \frac{Fi * Nnew}{Fi + Nnew * MAXDEC}$$

If Fi = 2.456MHz

Nnew = 65 536

MAXDEC = 4.2Hz per 100mS (Limiting inverter acceleration rate)

Then maximum count value Nold = 58 930. This corresponds to a frequency of 42 Hz.

Thus by using a clock frquency of 2.456 MHz it is possible to achieve the speed resolution required by the specification, and the useable speed range is 42 - 1920 Hz, corresponding to 7 - 320 Hz out of the inverter. Initially the aim was to get the starting speed as low as possible to limit the starting current of the (synchronous) wind up roll motor. However it was discovered that the motor was unstable at this speed, causing inverter trips, and the best practical starting speed was 13.3Hz, corresponding to 80Hz from the computer.

2.1.2.4 THE SYSTEM CLOCK.

The SMT operating system requires a 50Hz system clock which is provided by one of the PIT channels. The requirement for highly accurate speed measurement made it necessary to read the system clock "on the fly" to achieve a clock resolution of less than 20mS (see chapter 3.3.3 and 3.3.10). As a result the PIT was operated in mode 2 as a divide by N rate counter, rather than in mode 3 as a square wave generator. In mode 3 the count value is decremented by two for each clock pulse, and when the count reaches zero the state of the OUT pin is inverted. In mode 2 the count value is decremented by one each clock pulse, and when the count reaches zero the OUT pin goes low for one clock pulse. In mode 3 the state of the output pin has to be determined, as well as the count, whereas in mode 2 only the count has to be read, which is far simpler. Using mode 2 had no other ramifications for the design, and gave satisfactory performance in practice.

The clock frequency used was 1.228MHz, so the required PIT count value was :

$$\frac{1\ 228\ 000}{50} = 24\ 560$$

2.1.3 THE PROGRAMMABLE INTERRUPT CONTROLLER.

This section assumes familiarity with the 8086 interrupt mechanism, 8086 assembly language, MULTIBUS, and the operation of various Intel peripheral chips. Intel publishes numerous reference manuals which should be consulted for more information (See references 9 and 10). This section will be limited to implementation notes.

Seven interrupts were required by the system, so the on-board eight level 8259A interrupt controller was adequate, and the use of slave interrupt controllers was not necessary. The interrupt vector table is initialised in MSMTU1.RTL, the user hardware initialisation module. Interrupts Types 1 to 4 are reserved for 8086 exception handling, single stepping and Non-Maskable interrupts, and Types 5H to 19H are reserved by Intel for future use. To maintain compatability with future hardware and still leave as much RAM available as possible, the controller used Interrupt Types 20H to 27H, occupying physical memory addresses from 80H to 9FH.

SMT has no provision for rotating priority interrupt arbitration, so the 8259A was used in the fixed priority mode, where INT 20 has the highest priority and INT 27 the lowest. The version 1.0 release of the MAGIC assembler has a bug in it which makes it impossible to use embedded absolute code sections in the middle of other code segments. This meant it was not possible to use the ORG and DW pseudo assembler instructions to initialise the vector table, which had to be done with MOV instructions. The use of each interrupt will now be considered in more detail.

2.1.3.1 INT 21, THE OFF BOARD ADDRESS TRAP.

The computer´s MULTIBUS interface was not used, which means that all system addresses are on-board. The computer has address decoding ROM´s which select the peripheral chips, and also give a signal indicating whether the address is on-board or off-board. If the address is off-board, it is handled by an 8289 bus controller and an 8288 system controller dedicated to the bus. The system then goes into a WAIT state until the bus peripheral being addressed acknowledges (XACK) that it has valid data on the data bus. If there is no peripheral, the computer would go into an indefinite wait state, that is it would "crash". To prevent this from happening, a monostable is gated with the XACK signal. If no ALE signal is received by the monostable in 1mS, it times out and clears the HOLD releasing the CPU from its WAIT state. The monostable output can be latched and used to interrupt the processor.

Since there is no off board-device in this system, any attempt to address one implies that something is wrong. If this does happen, the processor traps this condition, which causes the unrecoverable error routine (RRGEL) to be called with an error number 30.

2.1.3.2 INT 22, THE SYSTEM CLOCK INTERRUPT.

This routine gets called once every 20 mS when a system clock interrupt is received from the PIT. The routine is held in the system module SMTCLK, and is declared PUBLIC so that it can be initialised in MSMTU1. The service routine sets a system event which triggers the clock task which updates all its counters.

2.1.3.3 INT 23, TRAVERSE TACHO INTERRUPT.

This routine is triggerred when the tacho PIT count reaches zero, and is the timer for the traverse tacho task. It takes the current value of NOW, reads the system clock "on the fly", and sets an event which tells the traverse tacho that the timing period has elapsed. See Chapter 3.3.3 for a more detailed explanation of the operation. The routing of the physical link is described in the chapter on the I/O board.

2.1.3.4 INT 24, CHUCK TACHO INTERRUPT.

This performs the same function as INT 23 for the chuck tacho. See chapter 3.3.10 for more details.

2.1.3.5 INT 25, HOST LINK TRANSMIT DATA INTERRUPT.

The USART for the host link is on a piggy back SBX 351 board. This board can be configured for RS-232 or RS-422 (three state differential) operation by changing some links. From the programming point of view however, there is no difference between these two modes.

As explained in chapter 3.3.12, the multidrop serial link is configured so that it produces an interrupt when the transmit buffer is empty. The interrupt service routine fetches the next character from the packet buffer, checks to see if it is the final character, and loads it into the transmit buffer (after checking to make sure it is empty). If the character is the final one, then an event is set which tells the Host link communications task that the transmission is complete. The routine is then exited without reloading the transmit buffer, and data transmission halts.

2.1.3.6 INT 26, THE LOCAL VDU RECEIVE DATA INTERRUPT.

Every time a key on the local VDU is pressed, the local VDU USART receives a character in its input buffer, causing an interrupt that calls the service routine in the system module MSMTPIO.RTL. The service routine checks the received character to see if it is control-A. If it is, it sets the system event which triggers the CTL-A task and exits. If it isn't CTL-A, it checks that the input buffer is not full, and if it isn't, places the character in the text buffer and echoes it on the VDU. If the buffer is full, then the pointers are reset and the previous data is overwritten.

2.1.3.7 INT 27, THE HOST LINK RECEIVE DATA ROUTINE.

This is similar to the transmit routine, and is fully described in chapter 3.3.12.

2.1.4 THE SERIAL LINKS.

Two serial links are provided. One is an RS-232 link used for communication with a local VDU, and the other is an RS-422 multidrop link for communication with a remote process management computer. The local VDU link uses the on board USART, while the multi drop host link uses the piggy back USART.

Both links operate at 9600 baud, and use eight data bits, with one stop bit and no parity. As explained before, the baud rate clock for the transmit and receive buffers on both USART´s are generated by the same PIT output (See section 2.1.2.1). Initialisation of the USART´s is performed in MSMTU1.RTL, and follows the sequence reccommended by INTEL, namely :

1) Disable interrupts.
2) Write four zeroes to USART control port with a 16 clock cycle settling time between writes.
3) Write a USART reset character to control port.
4) Allow USART to settle.
5) Write mode definition word and allow settling time.
6) Write command word to control port.
7) Enable interrupts.

Once an initialising sequence has started, it must be completed. To ensure that this happens in an oderly fashion, interrupts are disabled at the start, and four dummy zeroes are written to the control port, which will take the longest initialisation sequence possible to completion. The settling periods are required by the internal operation of the USART. Once any outstanding initialisation sequences have been completed, control words can be sent to the control port with appropriate settling times between writes. The reset word instructs the USART that mode and control words are about to follow. The mode word sets up the number of stop and data bits and disables parity. The command word enables the Transmit and receive buffers, and sets the DTR and RTS modem control pins. This completes the initialisation phase and the USART can receive or send data. When data is received, the interrupt routine merely has to read the USART data register. When the CPU wishes to send data, it writes it to the data port.

2.1.6 DIGITAL I/O.

The SBC 88/25 computer has an on board 8255 Programmable Peripheral Interface (PPI) chip, providing 24 lines of digital I/O. The system digital requirements were for six inputs and four outputs to the outside world, and one output for on board use, as follows :

INPUTS

1) Four address selection bits.
2) Position ready to start.
3) Tailing button. (not used in this system)

OUTPUTS

1) Position ready.
2) Winder inverter run contact.
3) Traverse inverter run contact.
4) Software error.
5) Off board memory address latch clear.

Each of these signals is fully described in the section on the I/O board (section 2.2), and the overall operation of these signals is described in Appendix C. This section will be limited to a description of the PPI.

The outputs had to be latched, but the inputs did not require latching. Port A of the PPI has an 8287 bi-directional buffer, whereas the other ports have facilities for output buffers only.

So port A was made an input port, and the Transmit/Receive pin of the 8287 buffer was pulled low to put it into the input mode. Port B was configured as an output port, buffered by 74LS00 NAND gates in sockets XU9 and XU10. Both ports were initialised as basic input/output mode 0 ports. The off board memory address latch required bit set/reset facilities, so one bit of port C was used, since port C supports bit set/reset operation. Once again the requirement was for simple latched operation, so port C was configured as a mode 0 output port. Thus the PPI was initialised as follows :

PORT A	INPUT	MODE 0
PORT B	OUTPUT	MODE 0
PORT C	OUTPUT	MODE 0

The code which initialises the PPI is in MSMTU1.RTL.

2.1.6 RAM AND ROM.

Since the 8086 interrupt vector table occupies the first kilobyte of physical address space, RAM is always mapped into the lowest memory locations. The basic SBC 88/25 board comes with 4k of RAM fitted. SMT requires approximately 2.5 k, the vector table uses 256 bytes, and the applications tasks need approximately 1.5 k giving a total requirement of 4.25k. It might have been possible to reduce this to under 4k by carefully tuning the system, but because of the project time limits extra RAM was added to the sockets provided on the board. The factory default setting maps this RAM from physical addresses 3000 to 3FFF (hex). It was re-mapped to be contiguous with the on board RAM by cutting a track from the address decoding ROM, and soldering a length of wire from a different address select line (See Appendix H for the details).

The 8086 reset vector resides at address FFFF0H, so on board ROM is always mapped to the highest memory locations. SMT occupies approximately 16 kilobytes of code, whilst the applications tasks have nearly 32 kilobytes of code. This makes

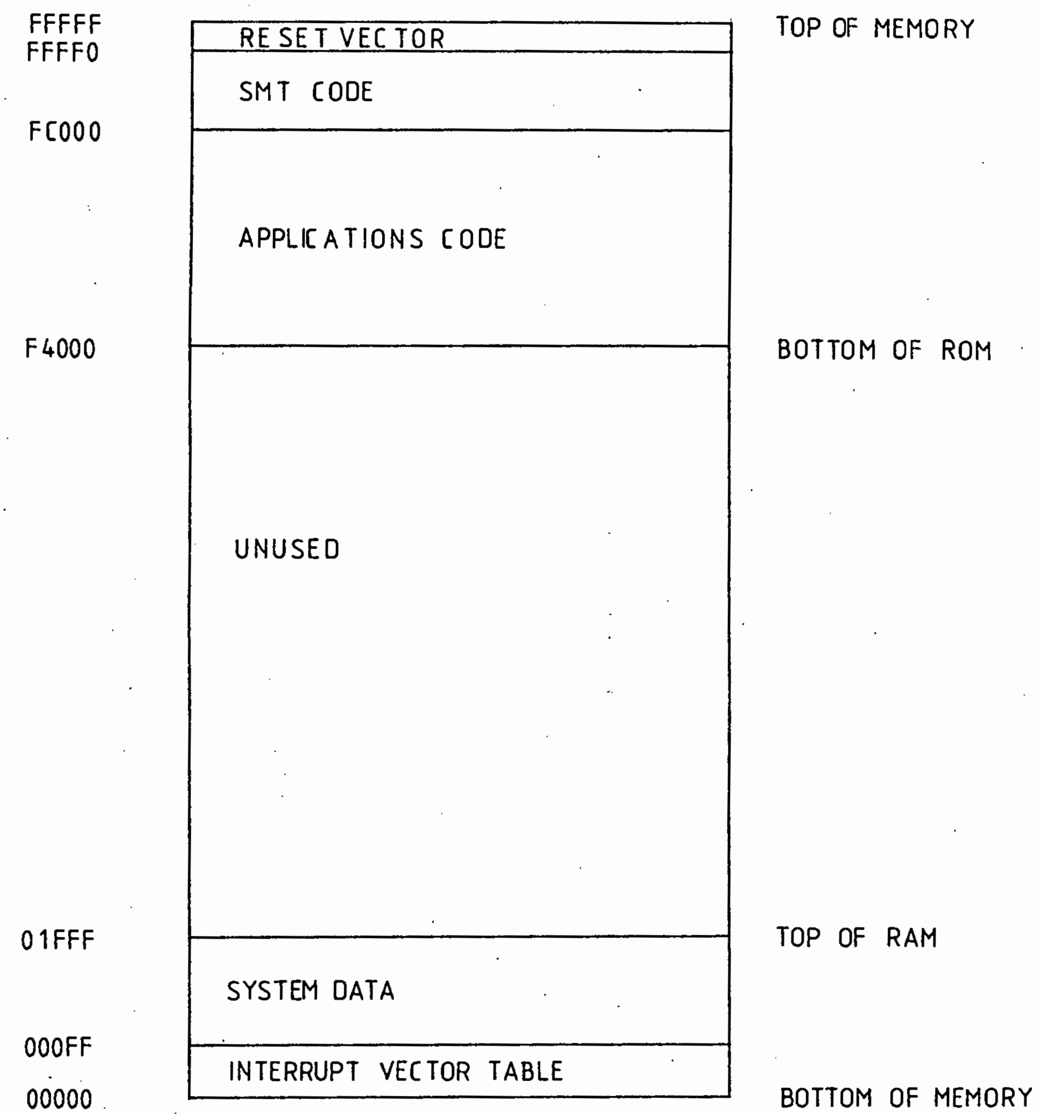


FIGURE 2.1.2: SYSTEM MEMORY MAP

the total code requirement 48k. There are four ROM slots on the board that can take all standard ROM sizes from 2 kilobytes up to 16 kilobytes. Four 8k ROMs give 32k, so it was necessary to use three 16k ROMs. The RESET vector at FFFF0H contains a jump to the SMT entry point. Figure 2.1.2 is a memory map of the system, showing how RAM and ROM are used by the controller.

2.2 THE I/O BOARD.

The function of the I/O board is to level shift and buffer signals from the plant and the computer. The board also logically AND's the winder interlock signals together and produces a Go/Nogo signal for the computer. Each interlock drives an indicator LED through a latch, so that its status can be monitored. The signals handled by the I/O board will now be described. Appendix A should be consulted for a description of a Nylon Spinning machine.

2.2.1 THE PLANT INTERLOCKS.

These are voltage free normally open contacts used to monitor the state of the plant. The computer supplies 24V to one side of the contact, and the other side goes to ground through an opto-coupler and a limiting resistor. If the contact is closed, the output of the opto-coupler goes high, and this output is ANDed with all the other outputs, and latched for an LED display. For the winder to start or continue running, all these contacts must be open. The start button is similar, except that the opto-coupler output is latched before being ANDed with the other interlocks.

2.2.1.1 INPUTS FROM THE PLANT.

A) STOP - this is a push button mounted next to each winder position that when pressed causes the winder motors to ramp to a standstill.

B) START - this is a push button mounted next to each winder position that causes the winder motors to ramp up to running speed when pressed, provided none of the other interlocks are enabled.

C) EMERGENCY STOP - There is one emergency stop button located at either end of the machine. If either of these buttons is pressed, all 24 winders ramp to a standstill.

D) WRAP DETECT - A sensor mounted close to the winder and traverse rolls is triggered if yarn gets wrapped around either of them. This usually happens when the thread line has broken, and the machine cannot operate under these conditions.

E) OIL MIST FAIL - The bearings of the motors and the chuck are lubricated by oil droplets carried by air. If the lubrication system fails, all 24 winder positions are ramped to a standstill.

F) THERMISTOR TRIP - Thermistors mounted in the stator windings detect whether the motor is overheating, which signals imminent motor failure, and ramps the winder to a standstill.

G) TRAVERSE INVERTER READY - Each traverse inverter has a set of contacts which are closed as long as the inverter is not in a tripped state, such as over or under voltage or current limit. If an inverter does trip, the winder is ramped to a standstill.

H) WINDER INVERTER READY - As for the traverse inverter.

I) TAILING BUTTON - See Appendix A for a picture of the transfer tail. Operation of the transfer tail indicates that a wind up period has begun. The ability to monitor this condition was included in case it became necessary to know this start point in future modifications to the system. However it is not used in this application.

2.2.1.2 OUTPUTS TO THE PLANT.

A) HEAD LIFT SOLENOID - Whenever a fault condition occurs or the winder is stopped for any reason, the winder roll must be lifted clear from the package surface. The winder roll is mounted in the head which is held in contact with the yarn package by compressed air. The head lift solenoid switches the flow of air so that the head is lifted clear of the package. This signal is generated by ANDing all the input interlocks together, and is generated completely independently of the computer software.

B) POSITION READY - This signal is fed to an indicator lamp mounted next to each winder position, and indicates when the motors are up to operating speed and ready for use.

C) WINDER RUN - This is a set of relay contacts which energises the winder roll inverter.

D) TRAVERSE RUN - As for the winder.

E) HOUR METER - This is a set of relay contacts which drive an hour meter which records how long the unit has been in operation. It is used for routine maintenance.

2.2.2 THE PULSE TRAINS.

There are four pulse trains in all. Two go from the computer via the I/O board to the inverter, whilst the other two come from variable reluctance probes mounted on the shafts of the traverse motor and chuck, and are used for speed sensing. The inverter drive pulses are optically isolated and then buffered by transistors. The tacho pulse trains are filtered and shaped, and then optically isolated.

2.2.3 THE INDICATOR LED's.

Fourteen indicator LED's are mounted on the front panel of each inverter control unit to give information about the status of the plant interlocks and the operation of the computer system. Each indicator is briefly described.

L1 - Winder inverter is not ready, ie it has tripped.

L2 - Traverse inverter has tripped.

L3 - Thermistor trip has occurred.

L4 - Emergency stop button has been pressed.

L5 - Oil mist supply has failed.

L6 - Wrap detected on roll.

L7 - Stop button has been pressed.

L8 - Software watch dog.

L9 - CPU running.

L10 - Data transmitted to host from ICU.

L11 - ICU received data from host.

L12 - Position winder up to operating speed.

L13 - Position started.

L14 - Position ready to start.

L1 to L7 are red and reflect the state of the plant interlocks. If any are lit the winder is stopped or stopping. L8 to L11 are amber, and reflect the activities of the computer. L8 is driven by the sequencing task (see chapter 3.3.6.2), and will flash with a two second period if all the system tasks are functioning correctly. L9 is connected (via a buffer) to the 8288 system controller Address Latch Enable pin, and indicates whether the CPU is running. L10 and L11 are connected to the host link USART transmit and receive interrupt lines. If the USART is handling data, the LED´s will flash. L12 to L14 are green, and the winder must be used only when all three are on. There is also a pushbutton mounted on the front panel which resets the interlock LED latches. THIS RESET BUTTON ONLY AFFECTS THE LED´s, AND HAS NO EFFECT WHATSOEVER ON THE OPERATION OF THE CONTROLLER.

2.2.4 THE LINK WITH THE COMPUTER.

To simplify maintenance and housing , the I/O circuitry was mounted on a board with a MULTIBUS form factor, which was manufactured to a specification supplied by SANS. The MULTIBUS P-1 connector was not used at all, and the P-2 auxiliary connector was used for bringing power onto the I/O board and connecting the I/O board with the plant. The J-1 connector was used for connecting the I/O board signals to the computer. Figure 2.2.1 shows a logic diagram of the I/O board, whilst figure H.4 in Appendix H shows a detailed schematic diagram of the board.

2.2.5 THE MARK II VERSION OF THE I/O BOARD.

Two problems were experienced with the first version of the I/O board. The first was that no limiting resistor was placed on the outputs of the inverter pulse train transistors, so if this output was short circuited the transistors were destroyed. The second problem was the lack of a driver transistor for the "Winder Ready" signal lamp.

A second I/O board was produced with the necessary additions, and these boards have been installed in ICU drawers that have had repeated failures due to these ommissions.

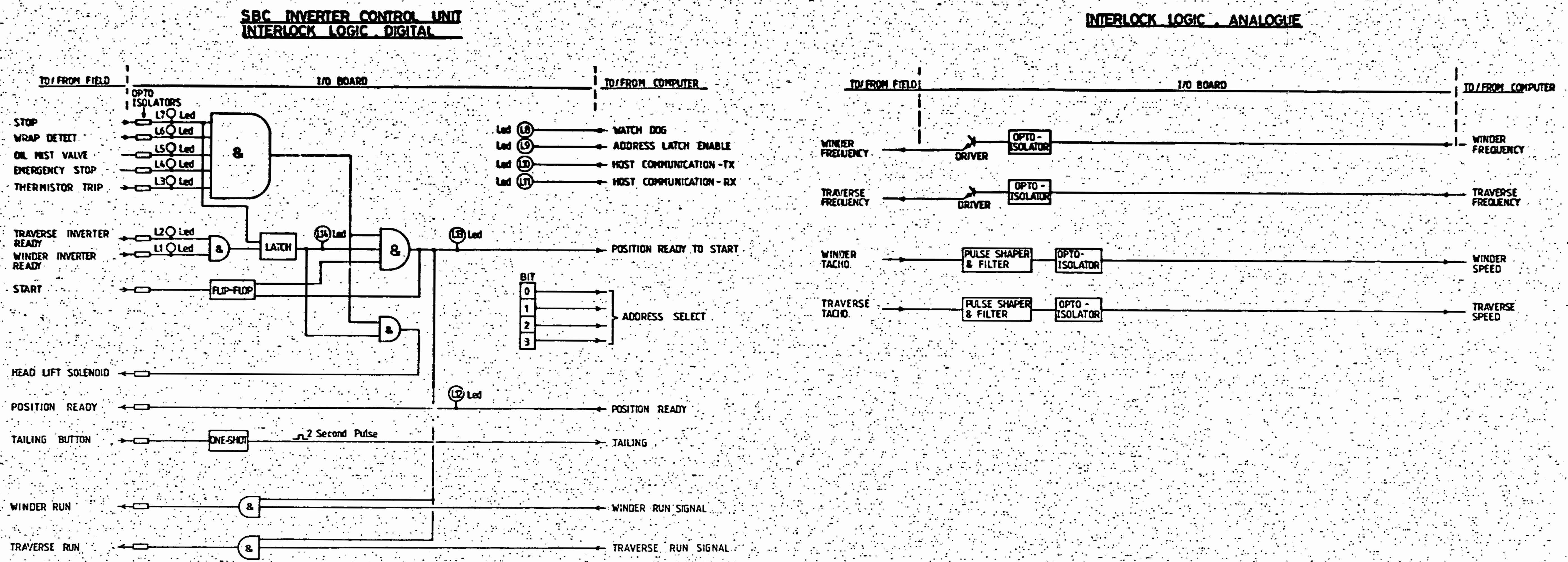


FIGURE 2.21

SOUTH AFRICAN NYLON SPINNERS (PTY) LTD
BELLVILLE

SCALE		
DRAWN		
TRACED	R.T.	JUNE '84
CHECKED	T.A.N.	[illegible]
APPROVED ENGINEER		
APPROVED (SANS)	D.L.G.	[illegible]

TITLE: ICU
INTERFACE BOARD LOGIC DIAGRAM

DWG No. SAB 1663

CODE No. P-00-01-49D

REV	DESCRIPTION	DRWN DATE	CHKD DATE	APP'D DATE	REV	DESCRIPTION	DRWN DATE	CHKD DATE	APP'D DATE	REV	DESCRIPTION	DRWN DATE	CHKD DATE	APP'D DATE	REFERENCE	DWG No.

2.3 THE INVERTER CONTROL UNIT CABINET.

The complete Inverter Control System consisted of 48 inverters and 24 computers, all of which had to be mounted in the same space as the original two inverters and one controller of the old machine. Thus space was at a premium, and the computer cabinet had to fit into a space which was two 19 inch rack spaces wide, and no more than eight feet high. In addition the cabinet had to be mounted against a wall with other cabinets on either side. However individual computers had to be easily accessible for maintenance.

2.3.1 THE CABINET.

Each computer is housed in a 19 inch rack mounting drawer with removable front and back panels. The indicator LED's are mounted on the front panel, and all the interconnection sockets are mounted on the back panel. The cabinet is made like a cupboard, with two doors opening outwards. Twelve computer drawers were mounted in each door of the cabinet, with hanging connections going from the back of the drawers to terminal strips on the back of the cabinet. The field wiring was brought in through the top of the cabinet and runs in a spine between the terminal strips. Figure 2.3.1 shows a front view of the cabinet with the doors opened.

This technique makes it possible to access the terminal strips or the back of the drawers by opening the cabinet doors. However it is also possible to remove individual I/O or computer boards without opening the cabinet, simply by removing the front panel of a drawer, and sliding the relevant board out. Figure 2.3.2 is an exploded diagram of the drawer showing the layout and orientation of all the drawer components.

2.3.2 THE POWER SUPPLIES.

The maximum current consumption from the 5V supply for each Single Board Computer, its SBX piggy back board and I/O board is 6A.

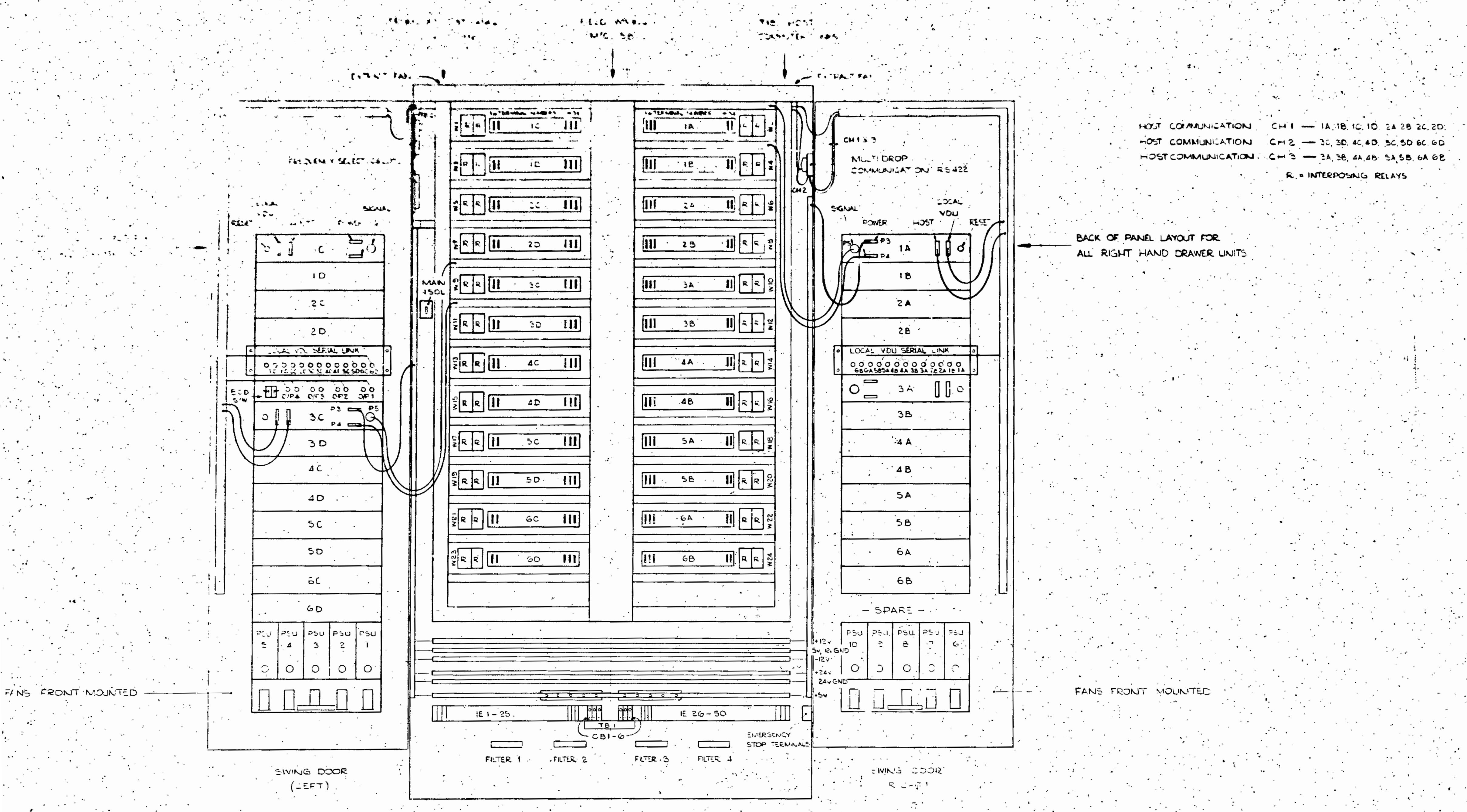


FIGURE 23.1

SOUTH AFRICAN NYLON SPINNERS (PTY) LTD BELLVILLE

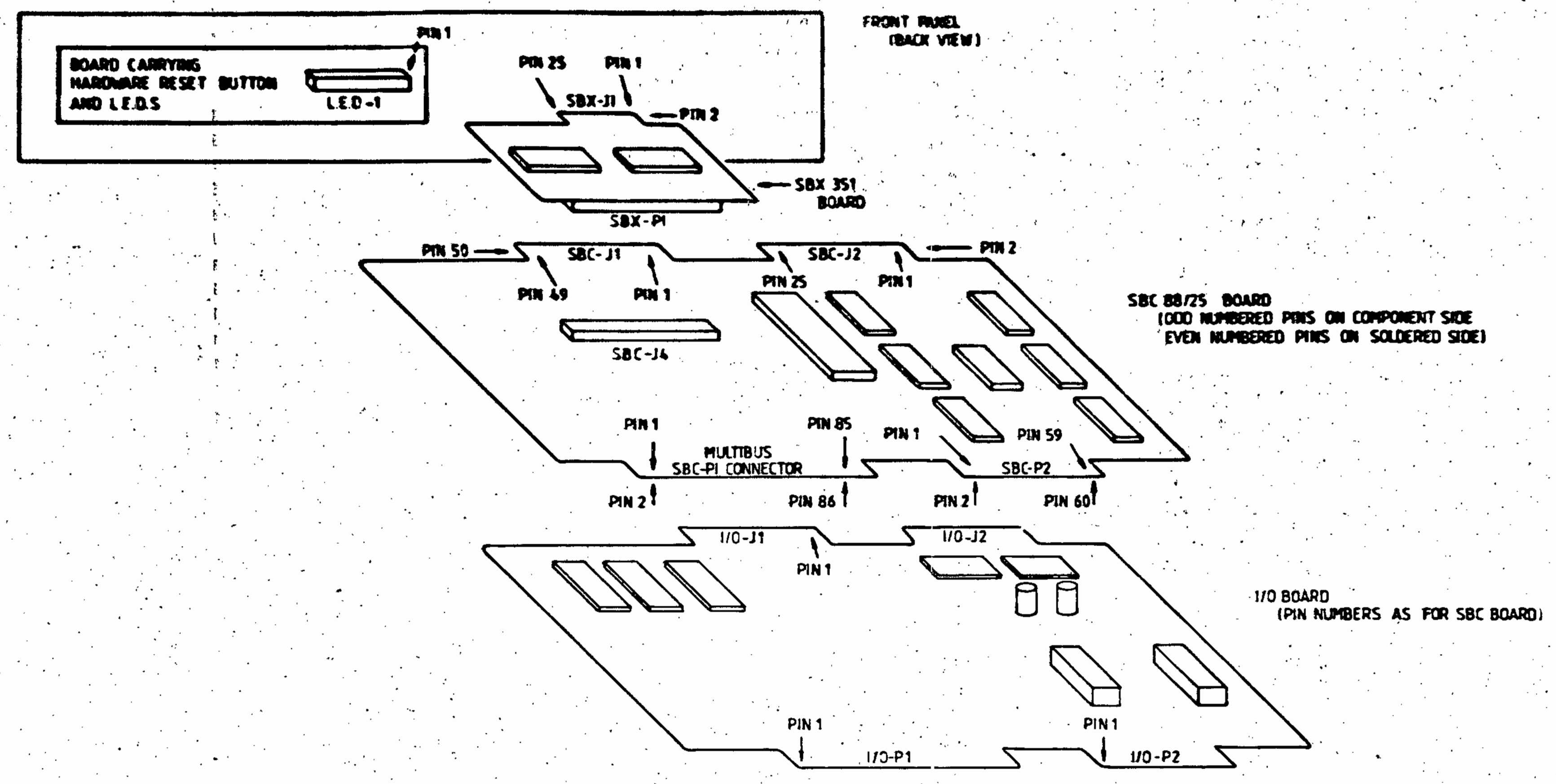
DWG No SAB 1656

SAB 1656

LEGEND OF LED MEANING

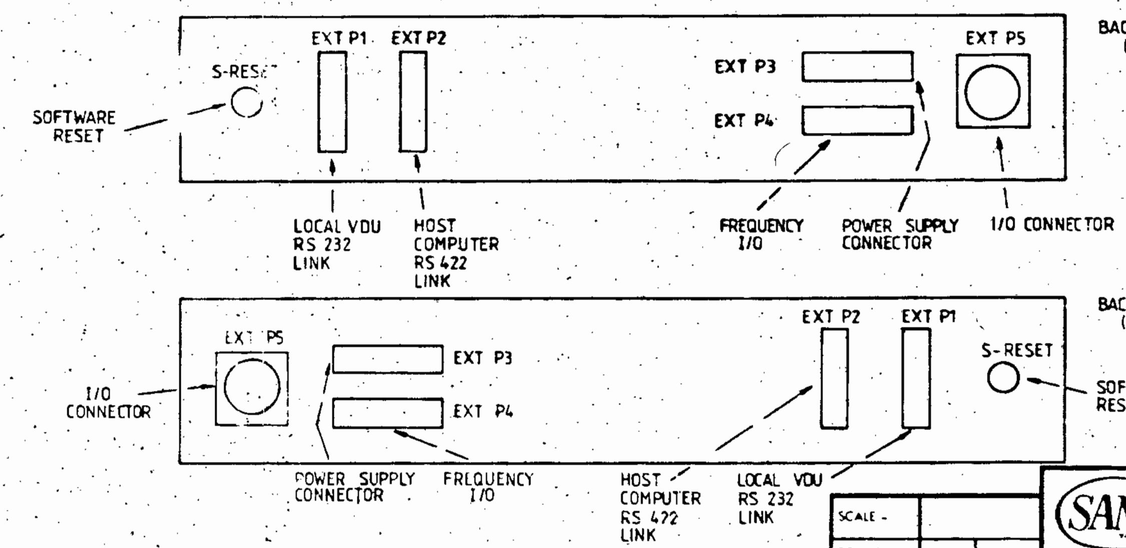
SANS L1 L2 L3 L4 L5 L6 L7 L8 L9 L10 L11 L12 L13 L14 RESET

LED	Meaning	Colour
L1	WINDER INVERTER NOT READY	RED
L2	TRAVERSE INVERTER NOT READY	RED
L3	THERMISTOR TRIP	RED
L4	EMERGENCY STOP ACTIVATED	RED
L5	OIL MIST FAILURE TIME-OUT	RED
L6	WRAP DETECTED	RED
L7	STOP ACTIVATED	RED
L8	RECEIVE ACTIVE	AMBER
L9	TRANSMIT ACTIVE	AMBER
L10	CPU RUNNING	AMBER
L11	WATCHDOG TIMER	GREEN
L12	POSITION READY TO WIND	GREEN
L13	START ACTIVATED	GREEN
L14	POSITION READY TO START	GREEN



PLUG INTERCONNECTIONS

I/O:
I/O-P2 — EXT-P5
COMPUTER TO I/O BOARD:
SBC-J1 — I/O-J1
COMPUTER TO SBX BOARD:
SBC-J4 — SBX-P1
L.E.D. AND HARDWARE RESET:
L.E.D.-1 — I/O-J2
LOCAL VDU:
SBC-J2 — EXT-P1
HOST COMMS:
SBX-J1 — EXT-P2
POWER SUPPLIES:
EXT-P3 — I/O-P2
EXT-P3 — SBC-P1
SOFTWARE RESET:
S-RESET — SBC

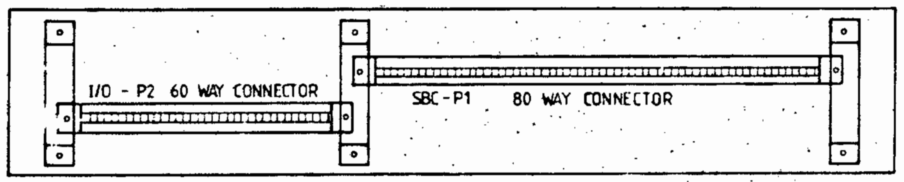


BACK PANEL (REAR VIEW) LEFT HAND DRAWER

NOTE: ALL OTHER INTERNAL PLUGS, SOCKETS AND BOARDS REMAIN IN A CONSTANT POSITION

BACK PANEL (REAR VIEW) RIGHT HAND DRAWER

FIGURE 2.3.2



BACKPLANE DETAILS (VIEWED FROM FRONT)

SOUTH AFRICAN NYLON SPINNERS (PTY) LTD BELLVILLE

TITLE: INVERTER CONTROL UNIT LOCATION OF CONNECTORS IN INVERTER CONTROL UNIT

DWG No: SAB 1593

CODE No: P-00-01-49D

SCALE		
DRAWN	H.W.	1.5.84
TRACED		
CHECKED		
APPROVED ENGINEER		
APPROVED SANS		

0 ISSUED FOR COMMENT

THIS DRAWING IS A PRIVATE AND CONFIDENTIAL COMMUNICATION AND THE PROPERTY OF SOUTH AFRICAN NYLON SPINNERS (PTY) LTD. IT MUST NOT BE COPIED OR LOANED WITHOUT THE WRITTEN CONSENT OF SOUTH AFRICAN NYLON SPINNERS (PTY) LTD AND MUST BE RETURNED IMMEDIATELY ON THE COMPLETION OF TENDER OR CONTRACT.

REV 0

REV	DESCRIPTION	DRWN DATE	CHKD DATE	APPD DATE	REV	DESCRIPTION	DRWN DATE	CHKD DATE	APPD DATE	REV	DESCRIPTION	DRWN DATE	CHKD DATE	APPD DATE	REFERENCE	DWG No

There are 24 computers, so the total current supply required is 144 Amps. In addition, +/- 12V supplies were required for the RS-232 serial communications and the I/O board, and 24V was needed for the interlock contacts. This gave power supply requirements as follows :

Voltage	Current
+5V	144A
+12V	1.2A
-12V	1.2A
+24V	5A

The controller specification called for normal operation with up to two seconds power loss from ESCOM, and protection against power supply failure. A ring main UPS (Uninterruptible Power Supply) is available at SANS, so the power supplies were run off this for protection against ESCOM failure. For protection against power supply failure, a backup power supply was needed. 144A power supplies are expensive, so ten cheaper and more readily available 20A power supplies were wired in parallel. With this arrangement, it is possible to operate with three power supplies out of action.

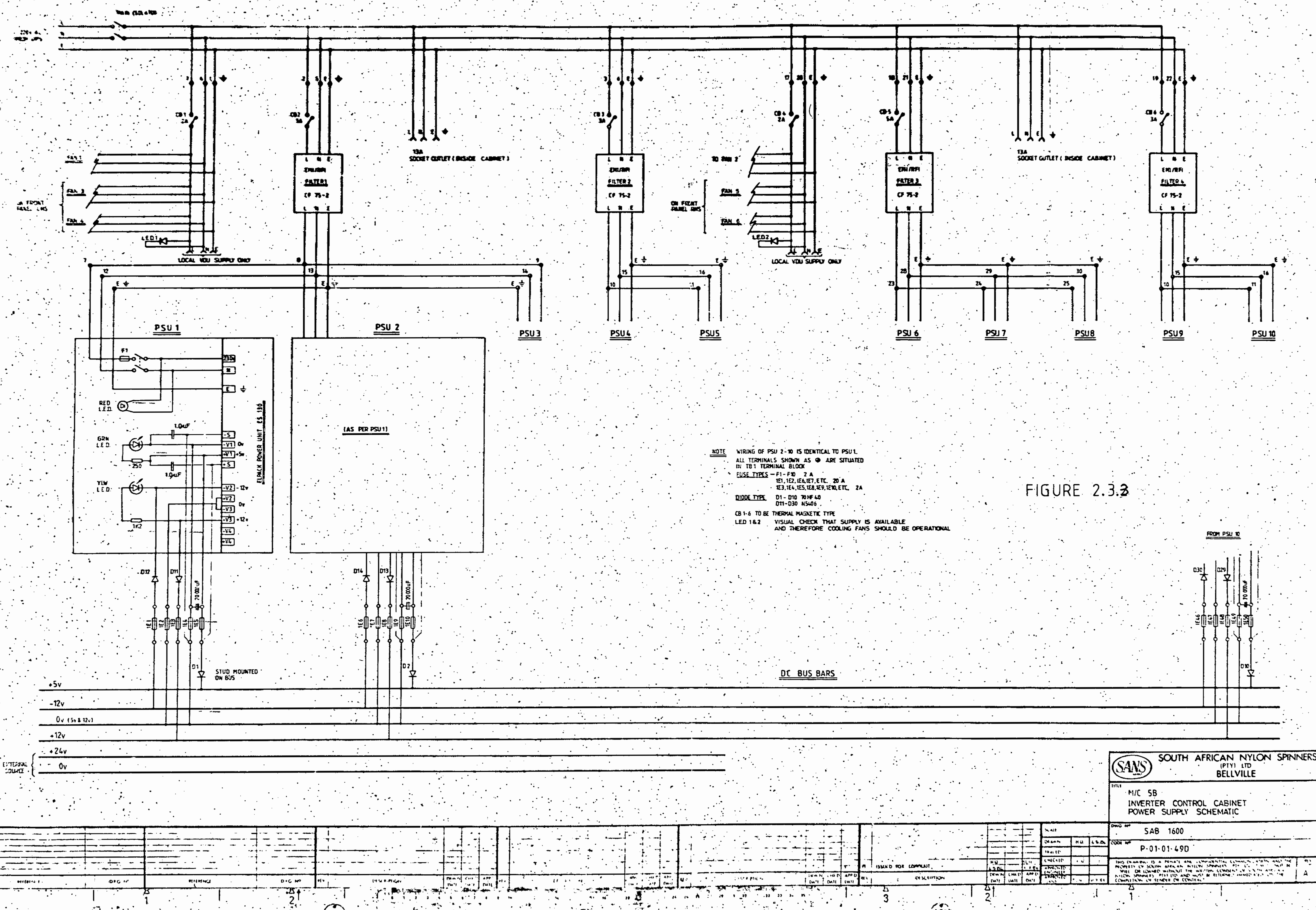
Each power supply is connected to a busbar system through a diode, to isolate it from the rest of the system if it fails. The diodes are 20A stud mounting types mounted directly onto the busbar for cooling. Each power supply has individual mains fuses, and 22 000 micro Farad capacitors on the 5V supply before the diodes. Voltage sensing is done at the diode anodes, which are as close to the load as possible. An attempt was made to sense directly on the busbar itself, so that the voltage drop of the diode would be automatically compensated for, but it proved impossible to distribute the current evenly between the power supplies, and the system was extremely unstable. Load balancing is done using a DC current clamp to measure the current in individual supplies, and adjusting the voltage setting pot on the power supply until it is supplying approximately 14A. This process has to be done iteratively two or three times until all the supplies are sharing the load evenly.

Each supply has three LED´s mounted on the front panel, indicating the presence of mains, 5V and +/- 12V. Each group of three power supplies is fed through an EMI/RFI filter to remove noise and spikes on the mains. The power supplies are mounted in the door of the cabinet beneath the computers, so that they can be adjusted without opening the cabinet door. All connections to and from the busbars are made via spade connectors.

ELPAC ES130 switced mode power supplies were used. These have isolated supplies of 5V at 20A, and 2 x 12V at 1A which can be connected in series to give the +/- 12V required. The 24V supply is derived from an external UPS driven supply used for the instrumentation and interlocks on the rest of the machine. Considerable problems were experienced initially with all ten power supplies going into fold-back current limiting for no apparent reason. The problem was traced to the ELPAC overvoltage crowbar circuit, which is extremely sensitive to noise. This was cured by paying very careful attention to earthing, and by increasing the capacitor on the crowbar thyristor gate, the philosophy being that it is cheaper to risk destroying a power supply than it is to stop production on the machine. Figure 2.3.3 is a schematic diagram of the power supply system, and Figure H.5 in appendix H shows a detail of the power supply connection to an inverter control unit drawer.

2.3.3 THE COMMUNICATIONS LINKS.

Each ICU had to be able to communicate with a local VDU, and also with a remote process control computer. The local VDU connections were made via 24 GPO stereo jacks mounted on the front panel of the computer cabinet, any particular ICU being addressed by plugging a GPO plug into the appropriate jack. The jacks had to be the stereo type to accommodate the transmit and receive line of the RS-232 protocol.



PSU 1
PSU 2
PSU 3
PSU 4
PSU 5
PSU 6
PSU 7
PSU 8
PSU 9
PSU 10
(AS PER PSU 1)
ELNACK POWER UNIT ES 130
RED L.E.D.
GRN L.E.D.
YLW L.E.D.
13A SOCKET OUTLET (INSIDE CABINET)
LOCAL VDU SUPPLY ONLY
STUD MOUNTED ON BUS
DC BUS BARS
FROM PSU 10
+5v
-12v
0v (5v & 12v)
+12v
+24v
0v
NOTE WIRING OF PSU 2-10 IS IDENTICAL TO PSU 1.
ALL TERMINALS SHOWN AS ● ARE SITUATED IN TB1 TERMINAL BLOCK
FUSE TYPES — F1-F10 2 A
1E1, 1E2, 1E6, 1E7, ETC. 20 A
1E3, 1E4, 1E5, 1E8, 1E9, 1E10, ETC. 2A
DIODE TYPE D1-D10 70HF40
D11-D30 N5406
CB 1-6 TO BE THERMAL MAGNETIC TYPE
LED 1&2 VISUAL CHECK THAT SUPPLY IS AVAILABLE AND THEREFORE COOLING FANS SHOULD BE OPERATIONAL
SOUTH AFRICAN NYLON SPINNERS (PTY) LTD BELLVILLE
M/C 5B
INVERTER CONTROL CABINET
POWER SUPPLY SCHEMATIC
DWG No SAB 1600
CODE No P-01-01-490
A

FIGURE 2.3.3

At the host computer end, the total requirement was for a local RS-232 link to an LA 120 hard copy terminal, and RS-422 link(s) to the ICU's. The host computer (which is part of the plant process management system, called T18 OLDMAC) was a PDP 11/23 computer, with no spare serial ports. The options were to get one RS-232 board and one RS-422 board for the system, or to get a single multi channel RS-232 board and attach RS-232 to RS-422 converters to it. The RS-422 board (with two serial channels) was unable to drive all 24 ICU's, so two boards would have been required. This made the cost of the first solution much higher than the second, so an 8 channel BML was chosen, with three RS-232 to RS-422 converters, which meant that one channel controlled eight ICU's (the converters had drive capacity for up to eight ICU's).

Figure H.6 in Appendix H shows details of the communications links.

2.3.4 WIRING AND INTERCONNECTIONS.

Appendix H.7 contains a schedule of all connections in the ICU drawers. Figure H.8 shows details of the field wiring connections, and Figure H.9 shows modifications that have to be made to a new SBC received from the manufacturers in order for it to work as an inverter control unit.

CHAPTER 3 : THE INVERTER CONTROLLER SOFTWARE

3.1 MAGIC, A SOFTWARE DEVELOPMENT SYSTEM.

Magic is a software development package which converts programs written in RTL/2 into machine code for four different microprocessors. Development is done on a multi-user host computer, which produces object code which can then be loaded into the target microprocessor system. This allows modular applications development by a team of programmers. There are facilities for testing and debugging programs and producing ROM format object code files. The package is designed to operate on DEC computers running the RSX-11 operating system.

Two run-time environments are supported by MAGIC. The first is a stand alone Base program called BASEM. This provides a minimal environment with a single task entry to a single user procedure. It sets up the RTL/2 stack and provides simple housekeeping and I/O functions. The second environment is SMT, which provides a real time multitasking environment as described in chapter 3.2. Both environments can operate in the free standing mode, or in the host /target link mode.

The MAGIC system consists of five utilities and the source code for the run-time environments. The utilities are : A compiler; a target processor assembler; a linker; a target test controller; and a ROM object code formatter. The utilities can be configured for the users requirements by specifying different options at build time.

Figure 3.1.1 shows the application program development cycle in schematic form. The RTL/2 source program is submitted to the compiler which produces three output files : An assembler source file, a cross reference file and a compiler list file which contains error messages and compilation statistics. On successful compilation, the assembler source file and cross reference file are submitted to the assembler, which produces two output files : An object code file and an assembler list file which contains error messages and assembly statistics. On successful assembly the object file, along with other applications code and executive object files are submitted to the linker which : resolves inter and intra module references; allocates physical address space to all the object modules; splits the object code into data and procedure blocks for allocation to ROM or RAM in the final system; and verifies the link. The linker produces two files : A link file which is a loadable HEX-ASCII image of the system; and a map file which gives the locations of all brick level objects in the system and entry points. At this stage there are two options. The first is to use the test link controller utility (MAG) to load and test the link file into RAM on the target. The second option is to produce a ROM image using the ROM formatting utility (FDM), which can be loaded into an EPROM programmer and blown into EPROM for the final debugged version.

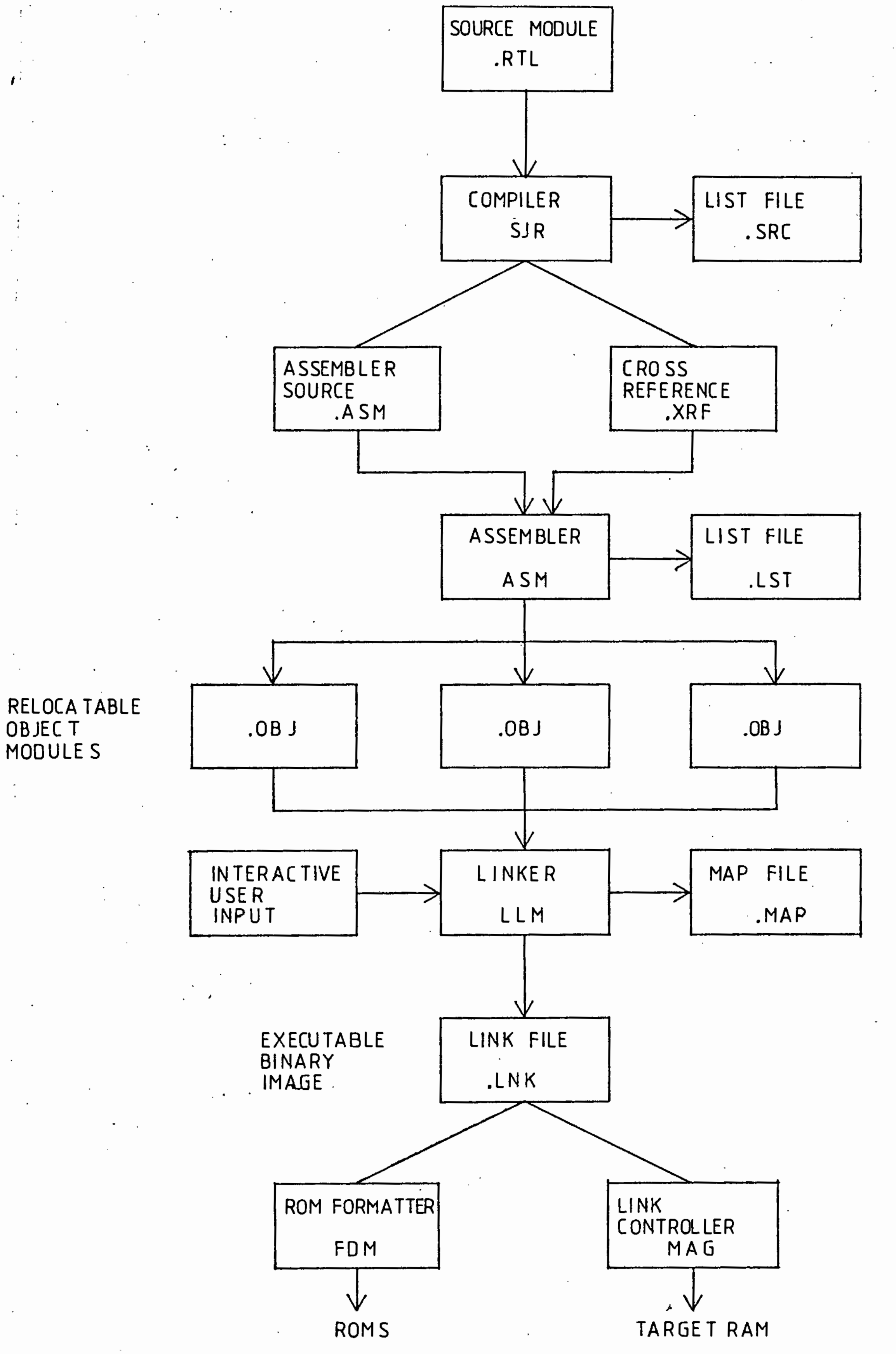


FIGURE 3.1.1: MAGIC DEVELOPMENT CYCLE

3.2 THE SMT OPERATING SYSTEM.

The source modules for SMT are supplied as part of the MAGIC package. SMT is a real time multitasking operating system suitable for use in small microcomputer systems. Only those parts of the executive required for the final system need be included, so its size will vary between applications, but generally it uses between 10 and 16k of code, and about 2 to 3k of RAM. Because the source code for the system is supplied, it can be easily modified for use by the applications code. It is modular so that only the features required in the final system need be added. It offers :

1) Support for up to 255 tasks
2) Task scheduling
3) Task priorities
4) Interrupt handling
5) Events
6) Timing
7) Facility control
8) Run-time monitor
9) System error traps
10) Run-time checking.

Each task can be in one of two states, either GO or NOGO. These two states can be further subdivided as follows :

1) GO
   - Running. Only one task can be running at any given time. This will be the task with the highest priority that is in the GO state.

   - Pending. Any task which is ready to run but has a lower priority than other pending or running tasks will be placed in the pending state.

2) NOGO - Stopped. The task will not run at all.

- Suspended. A task can be suspended for three reasons : Waiting for an event; securing a facility; delayed for a time interval.

Every time the currently executing task enters the "suspended" or "stopped" state or a hardware interrupt occurs, the scheduler puts the highest priority "pending" task into the "running" state. If there is no "pending" task, it activates the fallback task which has the lowest system priority and simply executes a "jump to itself" instruction. The executive itself appears as several system tasks, with negative task numbers to distinguish them from the application tasks.

Run-time checks are designed to provide protection between tasks, to try and prevent tasks from corrupting other tasks, and also to provide an oderly recovery mechanism from a detected error. Checks are performed on a range of activities such as array bound checking, divide by zero, stack overflow, floating point underflow or overflow and a range of other system errors.

Each of the procedures that provide system services will be briefly described.

1) Timing

DELAY for an integer multiple of 20 mS

2) Event handling

SET an event flag
RESET or clear an event flag
WAIT for an event flag to be set
TWAIT wait for event flag to be set or for a delay timeout

3) Facility control

SECURE a system facility, suspend task if not available
RELEASE a system facility
TSECURE attempt to secure facility within timeout period
TSTSCR test whether facility is secured

4) Task control

START a task
STOP a task and release its secured facilities

5) Scheduling

LOCK disable task scheduling
UNLOCK enable task scheduling

6) Interrupts

HLOCK disable interrupts
HUNLOCK enable interrupts

7) Error handling

RRGEL unrecoverable error trap
CLEANUP release task's secured facilities
ERPRIN print error message

8) System default procedures

DFERP recoverable error trap
DEFIN stream input
DEFOUT stream output
RRNUL null procedure (fallback task)
ME returns own task number

Note : The MAGIC package used was version 1.0. This implementation is called "SMT+" which has some features not found in other versions of SMT. These additional features were not used in order to maintain compatibility with other systems in use at SANS. In addition this version had to be modified to comply with the version 18 SMT used by the company.

3.3 THE INVERTER CONTROL UNIT SOFTWARE MODULES.

3.3.1 SOFTWARE OVERVIEW.

The software for the Inverter control unit has been broken into ten tasks. The relationships between the tasks and the data in the system is illustrated in Fig 3.3.1. The control parameter database holds all the operational values and parameters that the computer controller must use to produce the required yarn, as well as all global flags and data bricks required for inter task communication.

Machine operational conditions are entered into the database either by an operator typing them in from a local VDU connected to the controller, or from the PDP-11 host via an RS-422 multidrop serial link. The host communication task and local VDU task convert the stream of serial data from either source into the appropriate data type expected by the computer hardware, and stores it in a buffer in the data base update task. The update task loads these values into the database at the correct point in the machine´s operational cycle.

The winder and traverse tacho tasks take a stream of pulses from the transducers fitted to the motor shafts and convert them into speeds which are stored in the database. The traverse tacho task also has to calculate the ratio of the winder and traverse speeds and initiate banding avoidance, and so it is called the "ratio" task.

The traverse and winder control tasks generate speed values which are used by the corresponding drive tasks to generate the pulse trains which drive the inverters. 13 seconds worth of speed values are generated every 11 seconds, and stored in the control tables. The drive tasks take speed values from the table every 100mS and convert them into appropriate square waves.

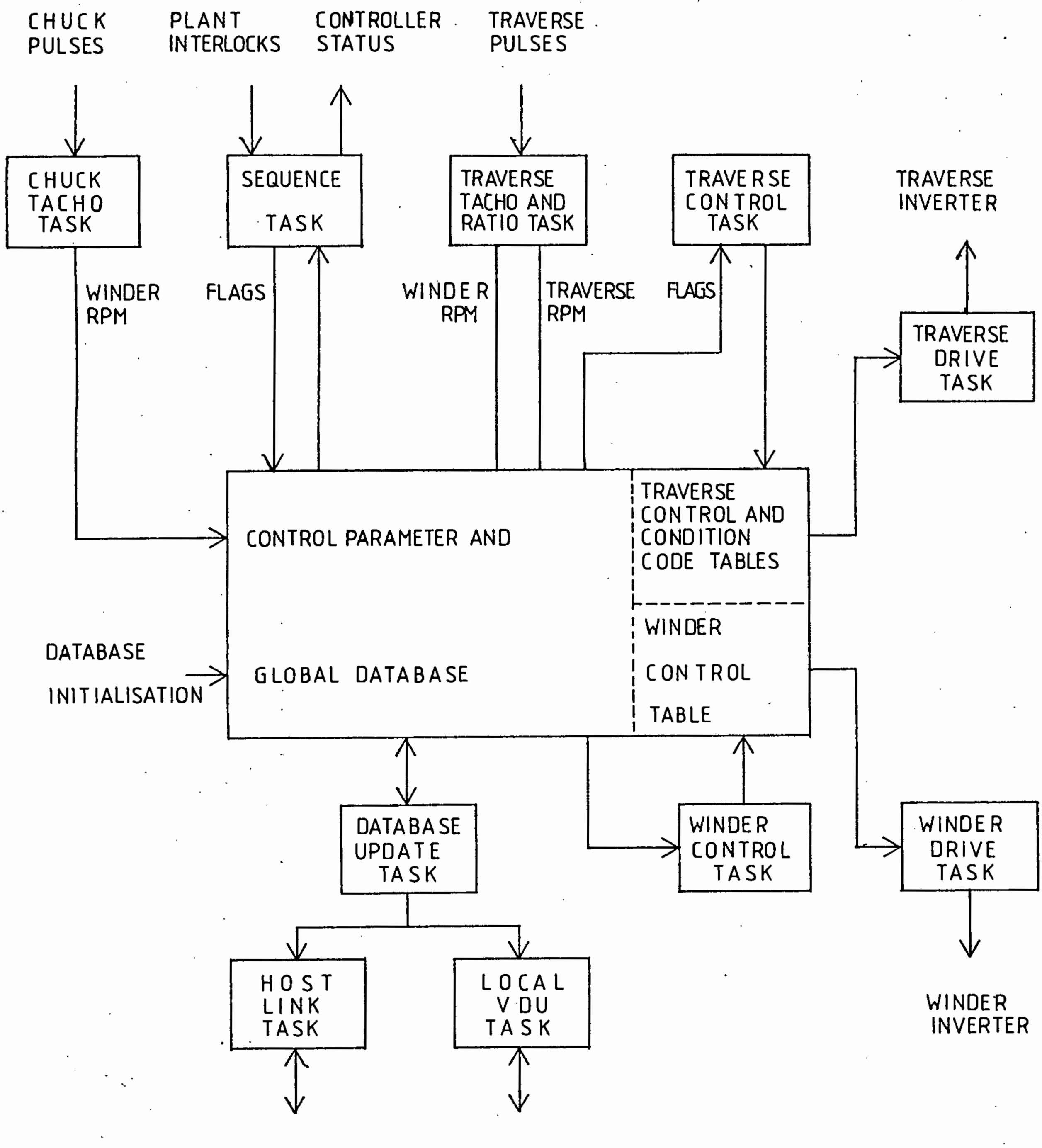


FIGURE 3.3.1: OVERVIEW OF ICU TASKS AND DATA

The sequence task monitors plant interlock signals and controls the system state by setting flags and events which are used by other tasks in the system. It also provides status information for the benefit of plant operators.

Each of the tasks will now be considered in more detail, and the design decisions and task algorithms will be discussed. The tasks will be discussed in the same order as the linker processes them. The emphasis will be on design criteria, and the system will be explained at flowchart level. Familiarity with RTL/2 and SMT is assumed, although a brief introduction to SMT is given in Chapter 3.2. The RTL/2 listings in appendix G should be consulted for details. In the software listings, variables and data relating to the winder are prefixed with a "W", whilst those relating to the traverse are prefixed with a "T". As far as possible, names have been made self explanatory.

3.3.2 SYSTEM STARTUP AND INITIALISATION.

Initialistion of the database is performed by the SMT procedure RRFILL in the module RRFILC which loads data held in a ROM table into a specified area of RAM. The linker creates a "data template" consisting of all the data declared in linker data segments (not to be confused with 8086 segment registers). This template is held in a data brick called RRDITP. RRFILL loads the contents of this brick into the specified RAM at sytem startup.

Hardware initialisation is performed by an SMT module called MSMTU1.RTL . This module is written in 8086 assembly language by the applications programmer, and sets up interrupt vectors, as well as initialising both serial link controllers, the 8255 parallel I/O chip, the six 8253 timer channels and the 8259 interrupt controller. MSMTU1 is called by the operating system before any system tasks are started. A listing of MSMTU1.RTL as used in this system is contained in Appendix G. In addition, the interrupt servicing routines are kept in MSMTU1

When the system is RESET or powered up, the CPU starts executing whatever code is at physical address FFFF0 Hex. A module called STARTUP.COD was placed at this address, which executes a jump to the SMT entry point (RRXEQ) in the system module ROMCBA. This module is coupled with the SMT segment of the system at link time, so that the external references between the system and startup code can be resolved. A hardware testing routine to check the system hardware at startup could be called first if required.

3.3.3 THE RATIO TASK.

The function of this task is to convert a pulse train from a speed transducer into RPM, calculate the ratio of the traverse and winder speeds and flag the rest of the system to take banding avoidance if required.

The speed of the traverse was found by loading a count value into an 8253 Real Time Clock chip, and then using the tacho pulses to decrement the count value and interrupt the processor when the count reaches zero. The processor measures the time interval between interrupts and calculates the shaft rotational speed by dividing the count value by the time taken to count that number of pulses.

To get the accuracy required by the specification, the count value loaded into the RTC and the measurement of the time between interrupts had to be considered very carefully. Problems arise because the traverse is frequency modulated and because the system clock has an inherent accuracy and resolution.

Both of these problem areas will be considered in detail, and then the results will be combined to calculate the actual tacho speed in RPM. Once an expression for the tacho speed in RPM has been derived, the requirements for banding avoidance will be considered.

3.3.3.1 DETERMINING THE COUNT VALUE.

As discussed previously (Chapter 2.1.2.2), the speed measurement technique chosen was that of counting pulses over a time interval. There are two approaches to this technique : the number of pulses in a fixed time interval can be measured (variable pulse method); or the time interval for a fixed number of pulses can be measured (variable time method). The latter method was chosen because the sytem clock is clocked at a much higher rate than the tacho RTC, so the inherent resolution is much higher. Analysis showed that the measurement interval would have to be unacceptably long to achieve the desired accuracy of measurement with the former method.

Tacho speeds are measured by taking the time for a fixed number of pulses to be counted. Since the traverse is modulated with a symmetrical waveform, tacho clocking pulses must be measured for an integral multiple of the traverse modulation period, so the measured value is the mean value of the traverse speed. The count value which will produce an integral multiple of the traverse modulation period can be calculated from the expression :

COUNT = MODULATION PERIOD * TACHO PULSE FREQUENCY

The modulation period is given, and the tacho speed should be the same as the motor speed (ignoring motor slip), and motor speed is given by :

$$\text{TRAVERSE MOTOR SPEED} = \frac{\text{TRAVERSE MOTOR RTC CLOCK FREQUENCY}}{\text{TRAVERSE MOTOR RTC COUNT VALUE}}$$

Note that there is one RTC used for generating motor control signals, and another used for measuring the tacho speed.

Several scaling factors are needed :

1) The tacho pulses are derived from a shaft driven through a 2:3 gear train from the traverse shaft.
2) The shaft gives two tacho pulses for every revolution.
3) The computer output frequency is 6 times the inverter output frequency.

Thus the required scaling factor is 2/3 * 1/6 * 2 = 2/9 and the required count value for the traverse tacho RTC is :

$$\text{COUNT} = \frac{2 * \text{MODULATION PERIOD} * \text{TRAVERSE MOTOR RTC CLOCK FREQUENCY}}{9 * \text{TRAVERSE MOTOR RTC COUNT VALUE}}$$

A factor was included to make the measurement period greater than 8 seconds, but still an integral multiple of the modulation period. This factor is a leftover from an earlier attempt to measure the speed. However, it guarantees that the required accuracy will be achieved, so it was left in. Thus :

```
         2 * PERIOD * TRAVERSE FREQUENCY
COUNT = -------------------------------- * TIME FACTOR
               9 * RTC COUNT VALUE
```

Where :

TIME FACTOR = 1 + (15 - MODULATION PERIOD) :/ (MODULATION PERIOD)

:/ is the integer divide function which returns no remainder. Thus in terms of the variables and constants used in the Inverter control system software :

```
              2.0 * TMPERIOD * CLOCKFREQ
       TCNT = -------------------------- * TNCYCCNT
                9.0 * TPITVAL(TPOINT)
```

Where TMPERIOD = Required modulation period.

CLOCKFREQ = Clocking frequency of RTC producing inverter pulses.

TPITVAL(TCNT) = Count loaded into inverter output pulse RTC.

TNCYCCNT = Factor to make duration of measurement greater than or equal to 8 secs.

The traverse motor is an induction motor so there will be some slip. This will result in a speed measurement error because pulses will be coming from the tacho slightly slower than expected, so the measurement interval will be slightly longer than the modulation period, and as a result the measured speed will not be the true mean speed. Note that this problem could have been avoided by measuring a variable number of pulses for a fixed time. However as pointed out before this method could not be used because the measurement intervals became unacceptably long.

If the traverse has a slip of e% of set speed, then the measurement interval will be e% longer than expected. To get an idea of the error involved a triangular waveform with amplitude a% of set speed without P-jumps was analysed (see figure 3.3.2a). The mean value of any waveform over some interval is given by the area under the curve. The mean value of the modulation waveform over the interval 0 - t_1 is zero. The error due to the measurement interval being too long will be given by the shaded portion under the curve. Figure 3.3.2b shows the error portion of the curve with the axes shifted to simplify the analysis. The slope of the curve is :

$$\frac{a}{t_1/4} = \frac{4a}{t_1}$$

and the mean value of the shaded portion is :

$$\frac{1}{et_1 - 0}\int_0^{et_1} \frac{4a}{t_1}\, t\, dt = \frac{4a}{et_1^2}\left[\frac{t}{2}\right]_0^{et_1} = 2ae$$

This result confirms the intuitive expectation that the error will be proportional to the slip and the modulation amplitude. The slip will vary according to the rate of acceleration or decelaration of the traverse, and can even have a negative value ("regenerative" braking). According to Hudgell (reference 2) the slip under no load conditions is 0.5%, whilst the average slip with modulation can be up to 0.8% at 200 Hz. Assuming a maximum slip of 1% and a maximum modulation amplitude of 4%, the worst case speed measurement error will be :

0.04 * 0.01 * 2 = 0.08% of mean speed

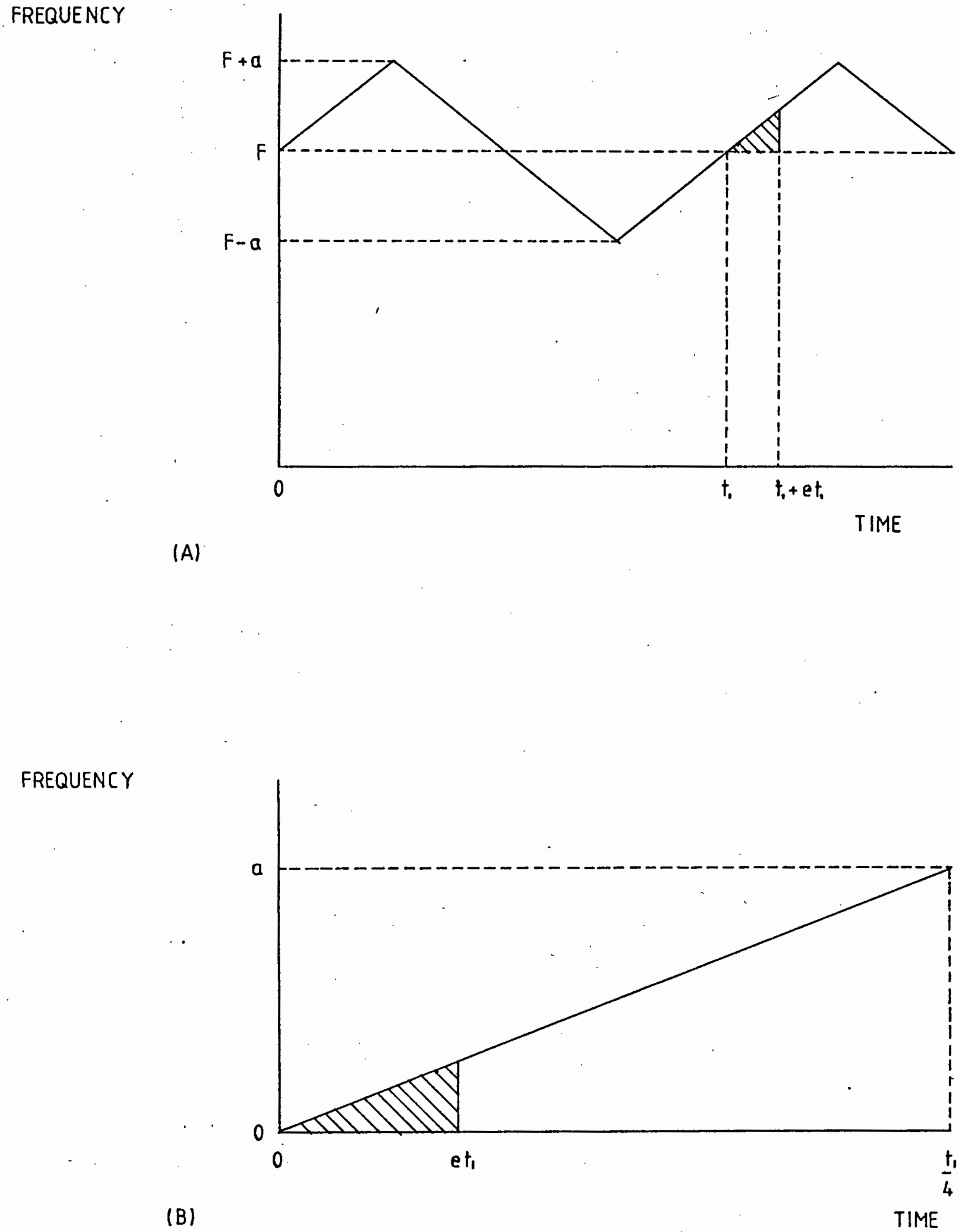


FIGURE 3.3.2: TRAVERSE SPEED MEASUREMENT ERROR

The specification called for measurement accuracies of 0.1%, so this measurement technique was accepted as meeting the design requirements.

3.3.3.2) DETERMINING THE TIME BETWEEN RTC INTERRUPTS.

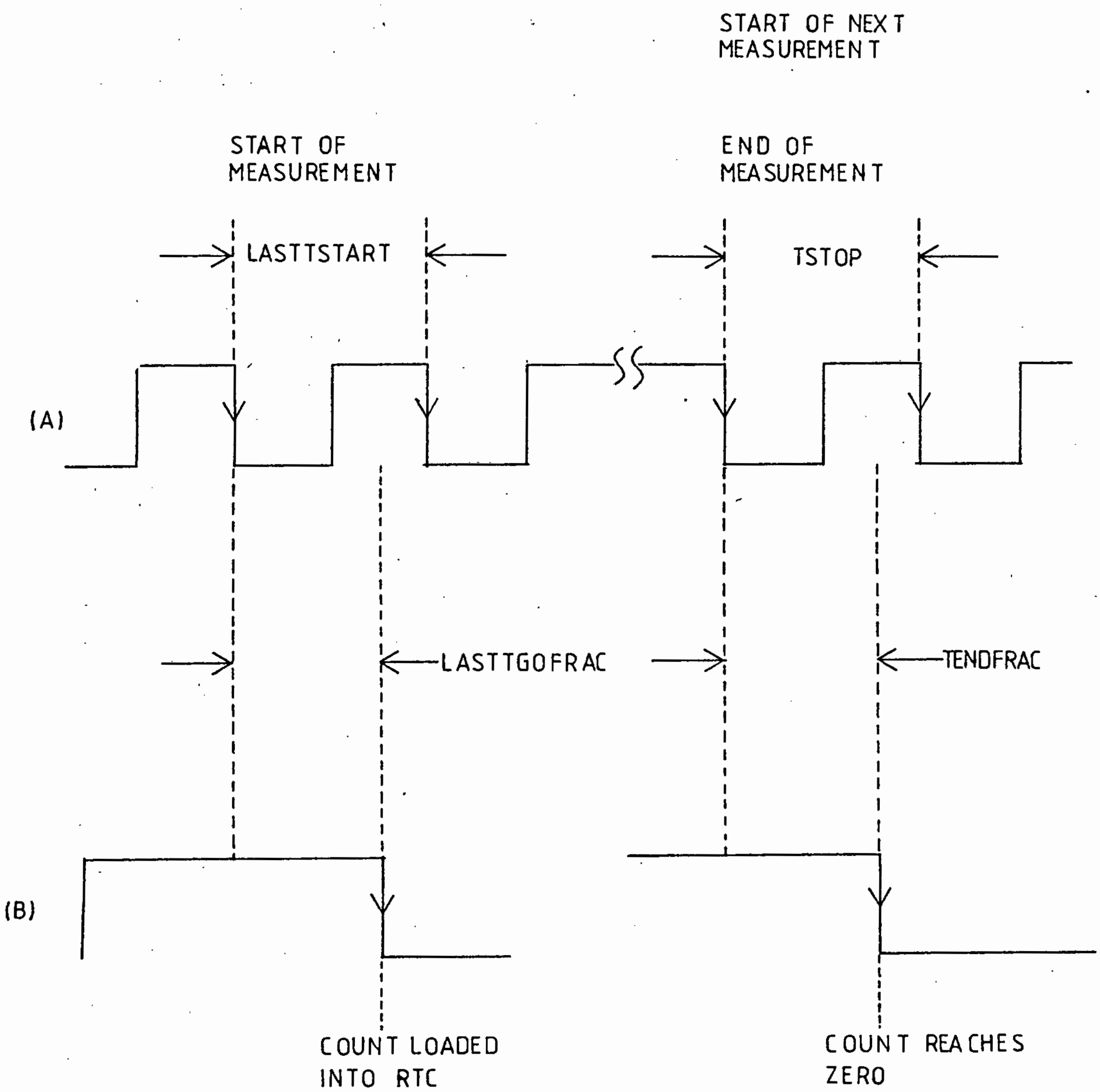
An examination of figure 3.3.3 shows that the accuracy of the measured speed depends on how accurately time is measured. The system operates by interrupting the CPU when the required number of pulses has been counted, and recording the current time. The time of the previous interrupt is known, so the time duration between interrupts can be calculated. The system clock operates at 50 Hz which means that its resolution is 20 mS. To achieve a 0.1% accuracy the measurement period would have to be at least 20 seconds. This was too slow for the chuck speed measurement (see chapter 3.3.10), which works on the same principles, so a better resolution had to be achieved.

It is possible to read the 8253 RTC count value at any time between interrupts. By this method it was possible to get a fractional count value with a resolution approaching that of the crystal driving the RTC. With reference to Figure 3.3.3 :

TIME BETWEEN INTERRUPTS =

TIME AT END OF COUNT- TIME AT START OF COUNT.

SMT provides a counter that gets incremented every 20 mS. The counter is an integer variable called NOW, whose value increases from 0 to 65536 and then "wraps around" back to zero. The interrupt service routine simply records the current value of NOW and the fractional value of the RTC. These values are saved until the next interrupt so that the end of one interrupt period is also the start of the next period. With variables defined as follows :



A-SYSTEM CLOCK PULSES

B-TACHO PULSES

FIGURE 3.3.3: RELATIONSHIP BETWEEN SYSTEM CLOCK PULSES AND TACHO PULSES

LASTTSTART = the value of NOW at the previous interrupt.
LASTGOFRAC = count value read from RTC at the previous interrupt.
TSTOP = the value of NOW at current interrupt. (Becomes LASTSTART)
TENDFRAC = count value read from RTC at current interrupt.
IT50HZ = total RTC count value to give a 50Hz pulse output.

Thus :

TIME AT END OF COUNT - TIME AT START OF COUNT

= (TSTOP + FRACTIONAL COUNT) - (LASTTSTART + FRACTIONAL COUNT)

$$= TSTOP + \frac{IT50HZ - TENDFRAC}{IT50HZ} - LASTTSTART + \frac{IT50HZ - LASTTSTART}{IT50HZ}$$

TENDFRAC and LASTTSTART are subtracted from IT50HZ because IT50HZ is loaded into the RTC and decremented. Simplifying this expression gives :

TIME BETWEEN INTERRUPTS =

$$TSTOP - LASTTSTART + \frac{LASTTGOFRAC - TENDFRAC}{IT50HZ}$$

If a situation occurs where a tacho interrupt occurs while a clock interrupt is being serviced, the tacho interrupt will be serviced immediately after the clock interrupt, before the value of NOW has been updated (NOW is updated by an S-task which is triggered by an event set by the interrupt H-task). This means that the value of NOW picked up by the tacho service routine could be wrong. To prevent this happening, the fractional part of the count is tested to see if the count is close to its start or end. If it is then that speed measurement is discarded. This is the function of the test with INTLIMIT in the listing.

3.3.3.3 DERIVING THE TACHOMETER SPEED.

Now that we have an expression for the count value to be loaded into the RTC and a method for accurately determining the time period between interrupts, it is possible to calculate the actual tacho speed. Remember that this is found from :

$$\text{TACHO SPEED} = \frac{\text{COUNT VALUE LOADED INTO RTC}}{\text{TIME TAKEN TO COUNT THAT NUMBER OF PULSES}}$$

Various constants of proportionality have to be considered. Time measurements are in 1/50ths of a second. The traverse shaft sensor is mounted on a secondary shaft driven through a 3:2 gear ratio, and the sensor produces 2 pulses for every revolution of the shaft. Finally the tacho speed must be converted to RPM by multiplying the frequency by 60. Thus the required constant is :

$$(60 * 50 * 3/2) / 2 = 2250$$

In the terms used in the program :

$$\text{TTACHO} = \frac{2250 * \text{TCNTLAST}}{\text{TTIM}}$$

Where :

TTACHO	=	Traverse tacho speed in RPM.
TCNTLAST	=	Count value for last time interval.
TTIM	=	Period between previous two interrupts from RTC.

3.3.3.4 BANDING AVOIDANCE.

The Chuck tacho speed is measured in a similar way to the traverse tacho speed (see chapter 3.3.10). By definition, the Ribbon ratio is given by :

$$\text{RIBBON RATIO} = \frac{6 * \text{CHUCK TACHO SPEED}}{\text{TRAVERSE TACHO SPEED}}$$

After the Traverse tacho speed has been calculated, the Ratio task calculates the Ribbon ratio, and checks to see if it is less than or equal to the critical value where ribboning starts or ends. If a critical point has been reached the flags which control the Traverse control task are altered so that banding avoidance starts or ends. The flow diagram in Fig 3.3.4 and the listing of the Ratio task in Appendix H give details of operation.

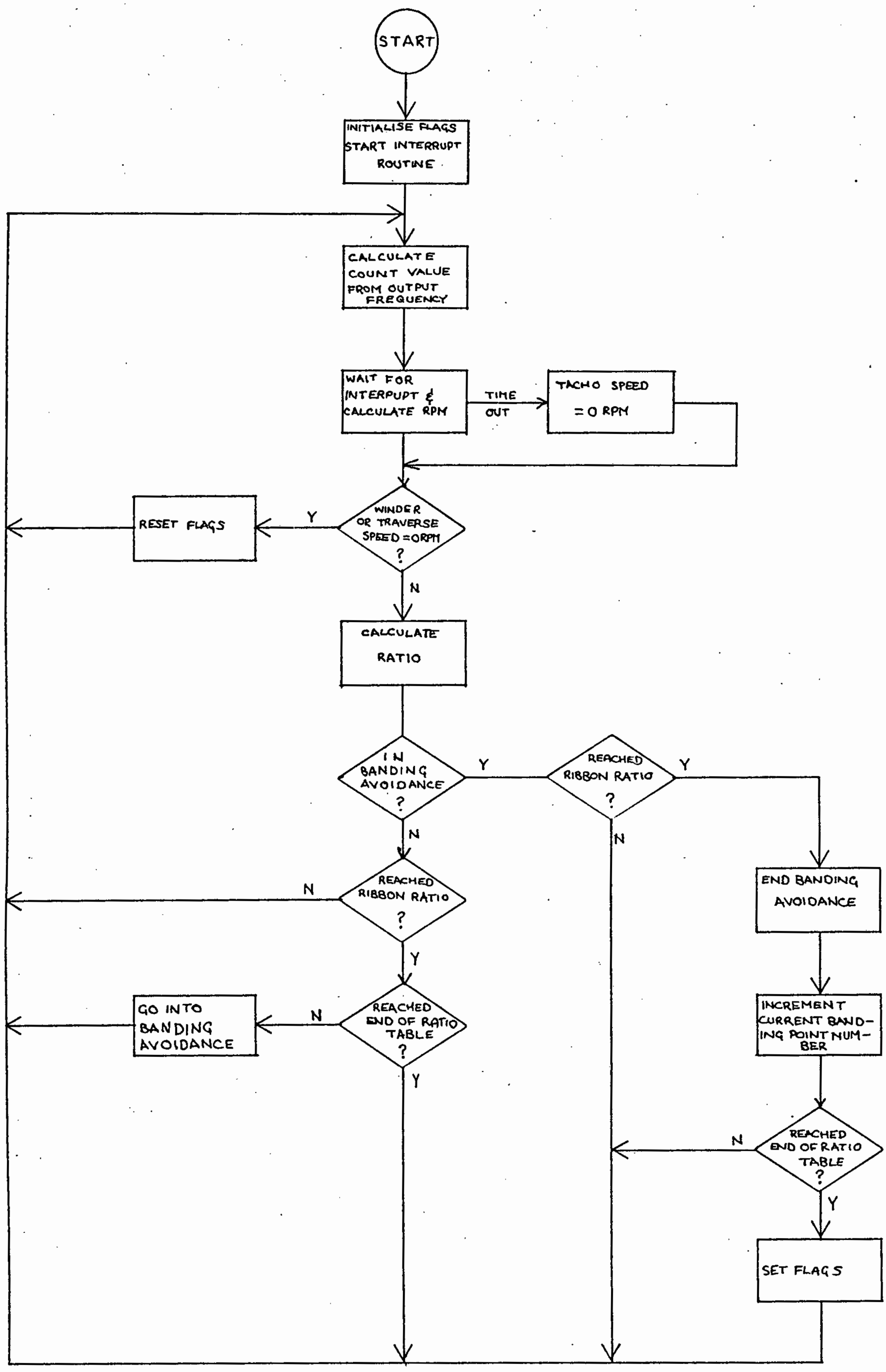


FIGURE 3.3.4: RATIO TASK FLOW DIAGRAM

3.3.4 WINDER DRIVE TASK.

This task reads a value from the winder control table every 100mS and outputs it to the winder drive RTC (See chapter 2.1 for a description of the RTC). The control table values are generated by the winder control task which calculates the required output speeds and fills the table. The winder and traverse drive tasks directly control the speed of the inverter and hence control the behaviour of the winding process. Because of their important function these two tasks have the highest priorities in the system. To prevent them from holding out other lower priority tasks they were made as compact and efficient as possible. Their only other duty is to ensure that maximum acceleration or deceleration rates are not exceeded, and that the maximum frequency for the winder drive is not exceeded. If they are exceeded for any reason, they are limited to safe values. Figure 3.3.5 shows a flow diagram of the winder drive task.

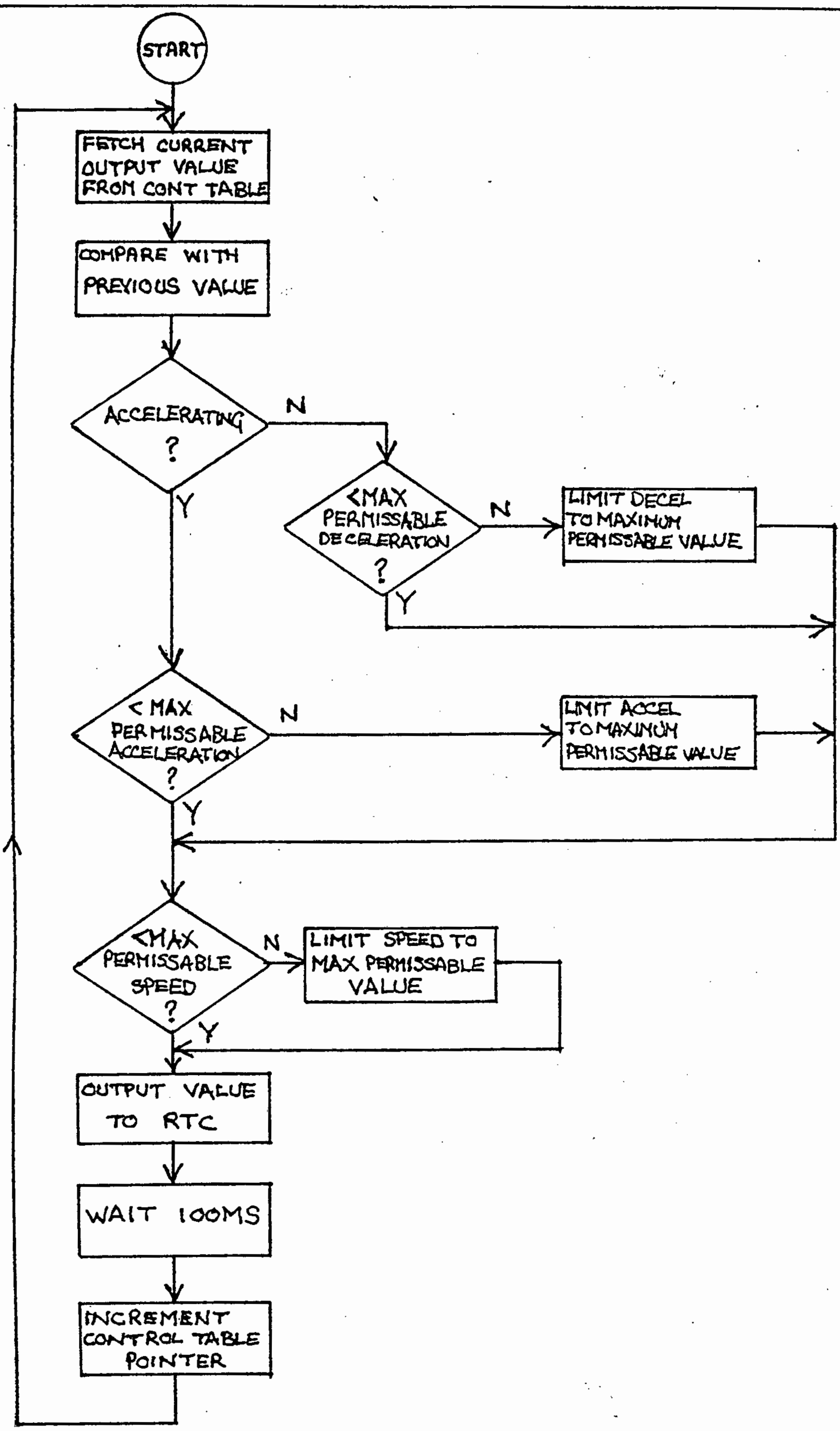


FIGURE 3.3.5: WINDER DRIVE TASK FLOW DIAGRAM

3.3.5 WINDER CONTROL TASK.

This task generates winder control table values according to the state of the system (as defined by the sequencing task), and responds rapidly and appropriately to system state changes. It generates 13 seconds worth of control table values every 11 seconds, except when the state changes (eg from STOPPED to STARTING), when it responds immediately. The drive task reads a value from this table every 100mS and outputs it to the RTC.

3.3.5.1 SYSTEM CONSTRAINTS.

This technique for generating speed values was arrived at after considering the limitations of the system as a whole. These limitations fell into two broad categories : those of the inverters and their associated motors; and those of the computer system. They placed conflicting demands on the computer, because on the one hand the motors needed to have their RTC output values updated as often as possible, but at the same time the amount of calculation to be done by the computer had be kept to a minimum. The solution was a compromise between the two requirements.

In practice it would have been extremely difficult to predict exactly what the timings and requirements of the various tasks would be, as they all run asynchronously with completely different cycle times, and the number of possible states of the system is vast. So empirical methods were used to determine some of the timings. The values that were arrived at were 100mS for the RTC update period, and a 128 value control table updated every 11 seconds. (Note that if one value is read from the control table every 100mS, then it will take 12,8 seconds to read the entire table). Figure 3.3.6 shows the broad principles of the system used, whilst figure 3.3.7 is a flow diagram of the control task.

Each of the areas of limitation will now be considered in more depth, showing the factors that led to the final design.

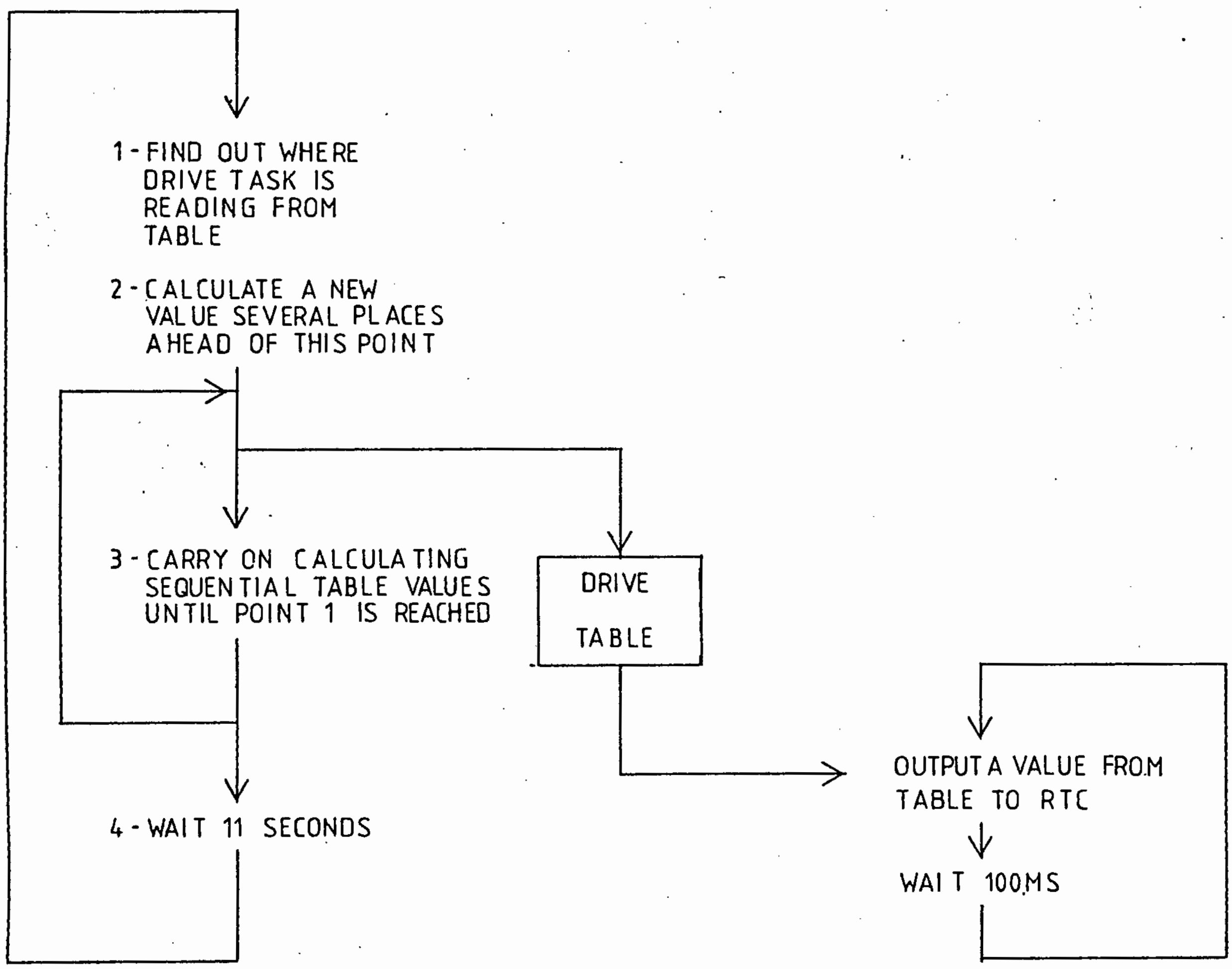


FIGURE 3.3.6: OVERVIEW OF TASKS PRODUCING INVERTER PULSE TRAINS.

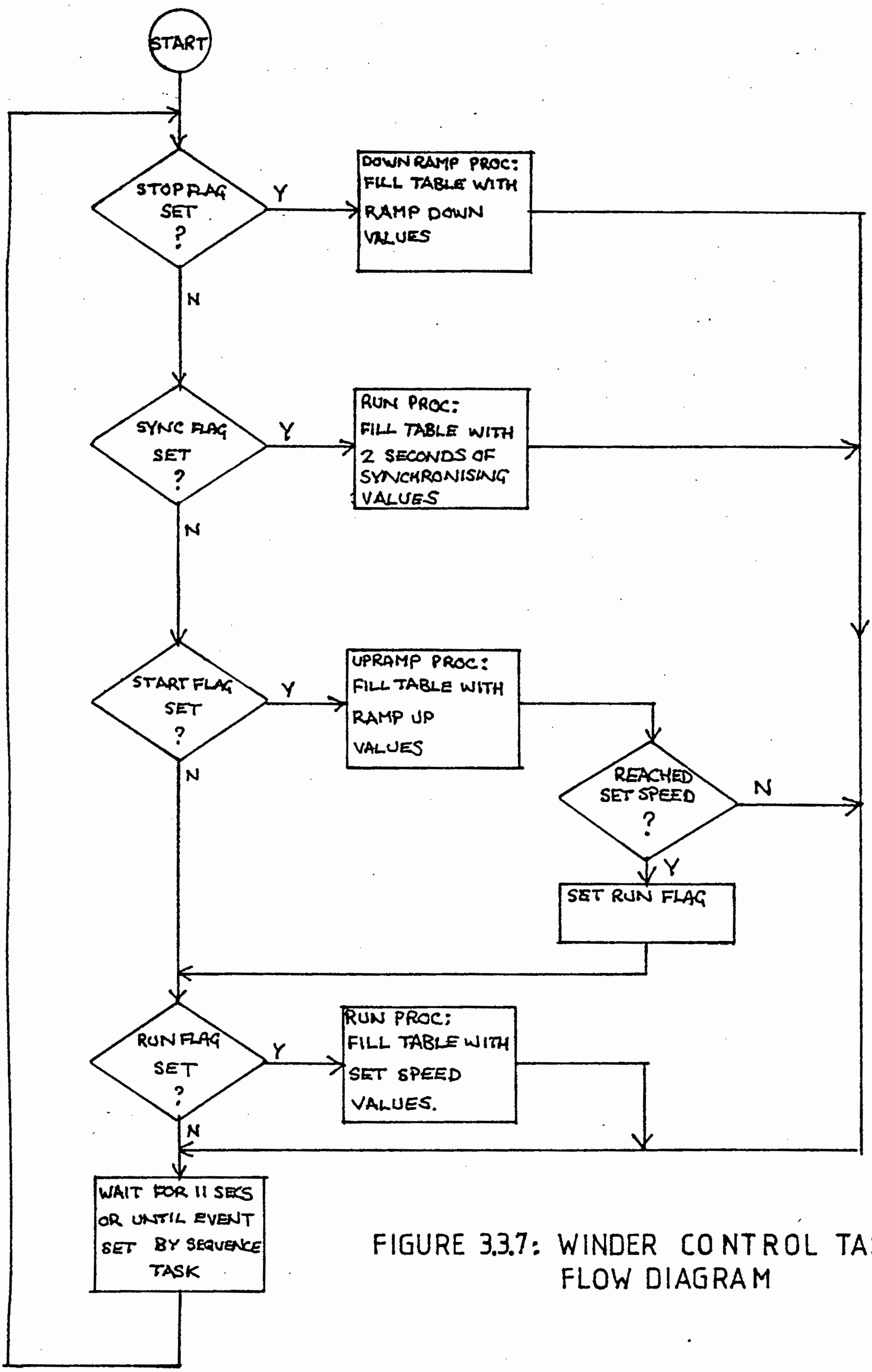


FIGURE 3.3.7: WINDER CONTROL TASK FLOW DIAGRAM

3.3.5.2 INVERTER AND HARDWARE LIMITATIONS.

The traverse and winder drive output have to be updated often enough to ensure smooth control of the winder and traverse rolls. There should be no sudden changes in frequency (apart from P-Jumps), which would shorten inverter life by causing large current surges, and cause unacceptable changes in yarn tension.

Since the traverse is frequency modulated, it is more important in deciding system values than the winder, so the limitations it places on the system will now be considered. Trials with sweep generator control of the traverse inverter indicated that the maximum continuous acceleration and deceleration rate that the inverter could maintain without tripping is 3 Hz/sec, whilst the maximum short duration acceleration and deceleration (less than five seconds) was 7 Hz/sec. Trials with computer control of the inverter showed that an RTC update period of 100mS enabled the specified modulation of the traverse to be achieved without exceeding the maximum acceleration rates of the inverter, as well as giving satisfactory inverter performance with negligible current spikes and smooth acceleration.

Similar trials with the winder inverter showed that the maximum continuous acceleration or deceleration it could reliabily sustain was 1Hz/sec. Trials with step changes were not conducted because they were not required, and because synchronous motors behave unpredictably (and sometimes destructively!) when they lose synchronism. The traverse and winder motors could accelerate at far greater rates than they could decelerate (mainly because the inverters can supply more energy than they can absorb - through regenerative braking - without tripping). Since the modulation waveform had to be symmetrical under all conditions, the limiting acceleration and deceleration rates had to be made the same, which meant that the overall acceleration / deceleration limit was determined by the deceleration limit.

3.3.5.3 COMPUTER SYSTEM LIMITATIONS.

There were three areas of the computer system that had to be examined : Firstly there was a limited amount of RAM available in the system, so efficient use had to be made of memory; Secondly there were other tasks in the system requiring CPU time, so the drive tasks could not prevent lower priority tasks being serviced; thirdly the control table had to have valid data before the drive task tried to read values from it. Each of these limitations will now be considered in turn.

A) RANDOM ACCESS MEMORY LIMITATIONS.

The drive table had to use RAM economically, as there was only 8k available. However the size of the table also had to be large enough to reduce the frequency of value calculation, so that lower priority tasks could get their fair share of CPU time. The communication and tachometer tasks had lower priorities, and cycle times (where appropriate) of about 5 to 15 seconds. A table update period of about 10 seconds (100 values) gave no detectable interference with other tasks. A value of 128 was eventually settled on because it made hexadecimal manipulation of the table array index simpler.

B) MULTITASKING LIMITATIONS.

The winder and traverse drive tasks were given the highest priorities in the system because the purpose of the controller was to supply a smooth supply of pulses to the inverters. Reliable real time generation of values cannot be guaranteed because of the nature of multitasking systems. The standard method for producing fast real time output values from a computer is by using look up tables. A static table required too much memory, as the range of operating conditions is extremely wide. An interpolated table would have increased the real time processing overhead once more. The solution was to use a dynamically generated table, the data values being generated at a low priority when the system did not have more important tasks to service.

A second aspect of the operating system that had to be considered was the 50Hz system clock. The SMT task delay mechanism is controlled by this clock, so delays are multiples of the clock period. Task switching often occurs after a system clock interrupt, which places a limit of 20 mS on the drive task switching time. In practice the drive task cycle period should be much greater than this to avoid excessive system overhead. A 100mS update period means that the drive tasks are only activated once every 5 clock periods, which gave a practically acceptable system overhead.

C) GENERATION OF CONTROL TABLE VALUES.

A 12.8 S cycle time left enough time for drive table value calculation. The time taken by the system to calculate 128 values under a wide range of conditions was measured, and it never exceeded 600mS. (These measurements were done by printing out the difference in the system clock variable NOW on entrance to and exit from the calculation procedure.)

To prevent the drive task reading invalid data (from 12.8 seconds previously), new data had to be generated before the drive task reached the end of the table. A 300% safety margin was allowed on the 600mS maximum generation time, so new values had to be generated 1.8 seconds before the end of the table was reached, that is every 11 seconds. Thus 12.8 seconds worth of values are produced every 11 seconds.

3.3.5.4 WINDER CONTROL TASK OPERATION.

The relationship between RTC count value and output frequency is given by :

$$\text{RTC OUTPUT FREQUENCY} = \frac{\text{RTC CLOCK FREQUENCY}}{\text{COUNT VALUE}}$$

The RTC has a 2,456 MHz clock, so the desired output frequency can be obtained by calculating the corresponding count value. Note that the output frequency and count value are reciprocally related, which means that frequency resolution is inversely proportional to output frequency, and that the range of output frequencies is limited. This factor coupled with the high resolution demanded by the specification indicated that a careful consideration of clock frequency was required. See Chapter 2.1 for a description of how this was determined.

The winder motor has four states of operation :

1) Stopped or ramping down to standstill.
2) Start up synchronisation.
3) Ramp up to operational speed.
4) Normal operation.

As explained in Appendix C (Overview of ICU system), the inverters have a minimum start up frequency of 13.3Hz. At startup, the output frequency is held at this value for 2 seconds to allow the (synchronous) winder motor to synchronise itself. The 2 second period was determined by trial and error. After the synchronisation period the motor has to be smoothly accelerated up to the set speed. Once at set speed it has to stay there until the stop button is pressed or a fault occurs (such as a winder or groove roll wrap, head lift through air supply failure, or inverter failure). When commanded to stop the motors must smoothly ramp down to a standstill.

Once down to the idling speed the system stays in the ramp down condition, although an internal test limits the minimum speed to 13.3 Hz. No distinction is drawn between the RAMPING DOWN and STOPPED states as a safety precaution, so that as long as the system is in this state it will try to ramp down to a standstill.

This prevents the motors stopping at an intermediate speed if an error occurs where there is confusion between a STOPPED and a RAMPING DOWN state.

Since phases 1,2 and 3 of the traverse and winder motors are identical, and since SMT is specifically designed to support re-entrant code, the drive table value generation procedures were held in a globally accessible module (COMPROC.RTL) which could be called by both the winder and traverse control tasks. The sequencing tasks TCONT.RTL and WCONT.RTL keep track of where the drive tasks are reading values from and call the appropriate table value generation procedures with the correct parameters.

PHASE 1 : STOPPING OR STOPPED

In this phase the table has to be filled with values that correspond to 13.3Hz, which corresponds to an RTC count value of

$$\frac{2\ 456\ 000}{6 * 13.33} = 30\ 708$$

The required output frequency has to be multiplied by six because the inverters require an input frequency six times greater than the desired output frequency. (See Appendix C for a description of the inverter operation). This function is handled by a procedure in COMPROC called DOWNRAMP. It has four parameters passed to it :

a) The name of the control table for which values are to be generated. (PITVAL in the program listing)
b) The current value of the pointer that the drive task is using to read values from. (STARTP in the program listing)
c) The value of pointer which table entries are to be generated from (CALCP in the program listing). This will usually be three or four more than the value passed in b).
d) The deceleration rate in Hz / 100mS. (DEC in the program listing)

Figure 3.3.8 is a flow diagram of the Ramp down algorithm DOWNRAMP. The RTC count value is derived as follows :

Required new speed = Current speed - deceleration rate.

Thus : $Fc / Nnew = Fc / Nold - DEC$.

Rearranging : $$Nnew = \frac{Fc * Nold}{Fc - Nold * DEC}$$

Where Fc = RTC clock frequency (2,456 MHz)
Nold = Previous RTC count value in table.
Nnew = Required RTC count value.
DEC = Deceleration rate in Hz / 100mS.

PHASES 2 & 3 : STARTUP SYNCHRONISATION AND RAMP UP TO RUN

When an operator presses the start button, the system goes into the Synchronisation phase for two seconds, and then moves into the Ramp up phase. The sequencing task controls the change from one state to the next by setting various flags which indicate what state it should be in. The control task monitors these flags and calls the appropriate procedure with the correct parameters for that phase of operation. In the synchronisation phase the RUN procedure is called which fills the control table with idling speed values (13,3Hz). Two seconds later the control flags are altered and the ramp up procedure is called. The next section describes the operation of the RUN procedure. The Ramp up procedure (UPRAMP in the common procedures module COMPROC.RTL) works in essentially the same fashion as the DOWNRAMP procedure described in the last section, except that instead of decreasing the speed, it is increased :

Required new speed = Current speed + acceleration rate

Thus : $Fc / Nnew = Fc / Nold + ACC$

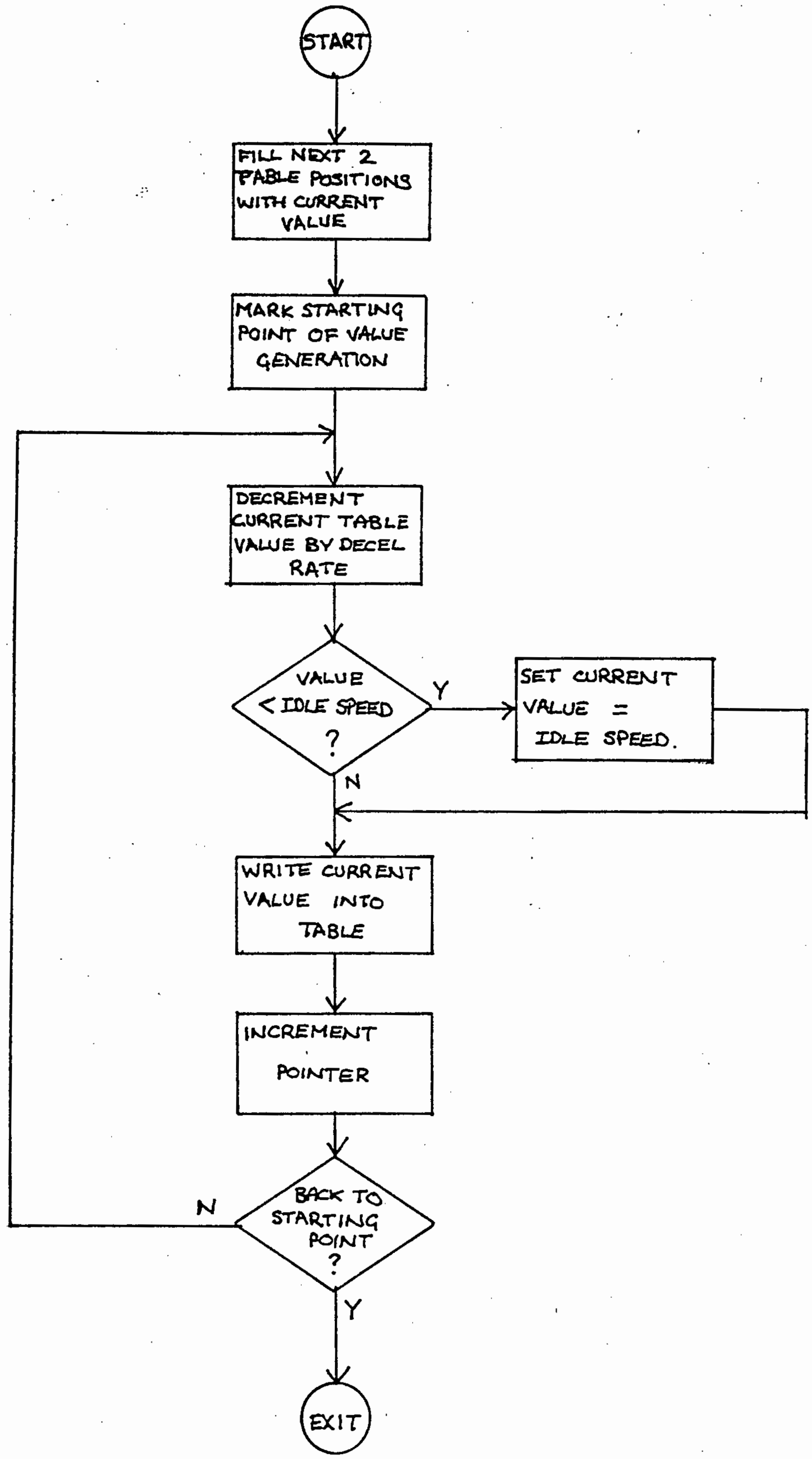


FIGURE 3.3.8: RAMP DOWN PROCEDURE ALGORITHM

Rearranging :

$$Nnew = \frac{Fc * Nold}{Fc + Nold * ACC}$$

Where Fc = RTC clock frequency (2,456 MHz).
Nold = Previous RTC count value in table.
Nnew = Required RTC count value.
ACC = Acceleration rate in Hz / 100mS.

The Ramp up procedure also has to check on whether the required operating speed has been reached. This check is performed every time a new table value is calculated. Once the normal run speed has been reached, the RUN procedure is called with the appropriate parameters to fill the rest of the table with run values, and the run flag (WRUNF) is set. The sequencing task clears the start flag (STARTF) when it detects that both motors are up to speed.

Figure 3.3.9 shows a flow diagram of the Ramp up procedure UPRAMP.

PHASE 4 : NORMAL RUN OPERATION

Once the motor has reached the required operational speed, the control table is filled with the required run value. This is very simply achieved, and figure 3.3.10 shows the flow diagram of the procedure. The RUN algorithm fills the control table with the speed value passed as a parameter, which means that it can be used in the synchronisation phase to fill the table with startup values.

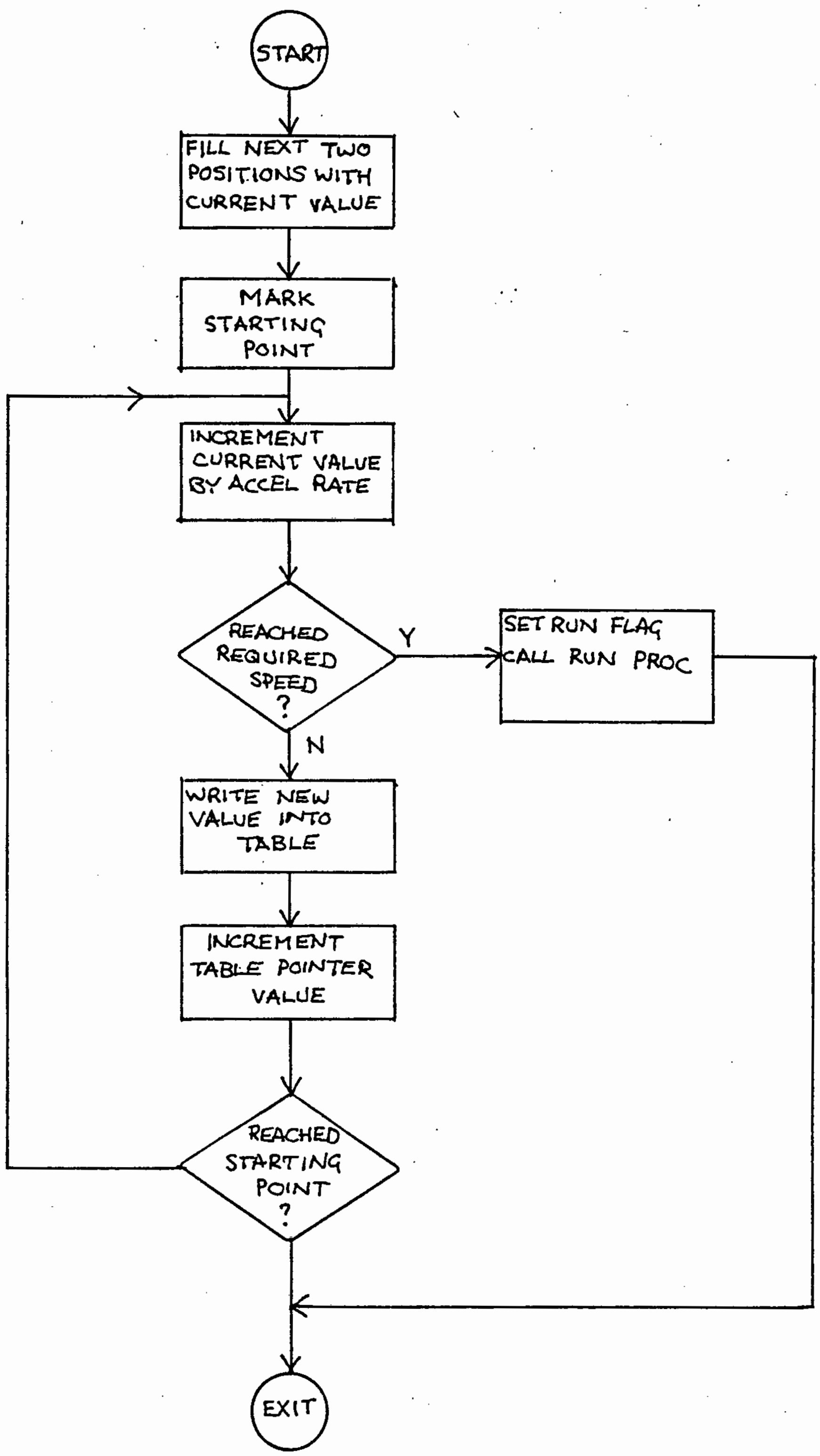


FIGURE 3.3.9: RAMP UP PROCEDURE ALGORITHM

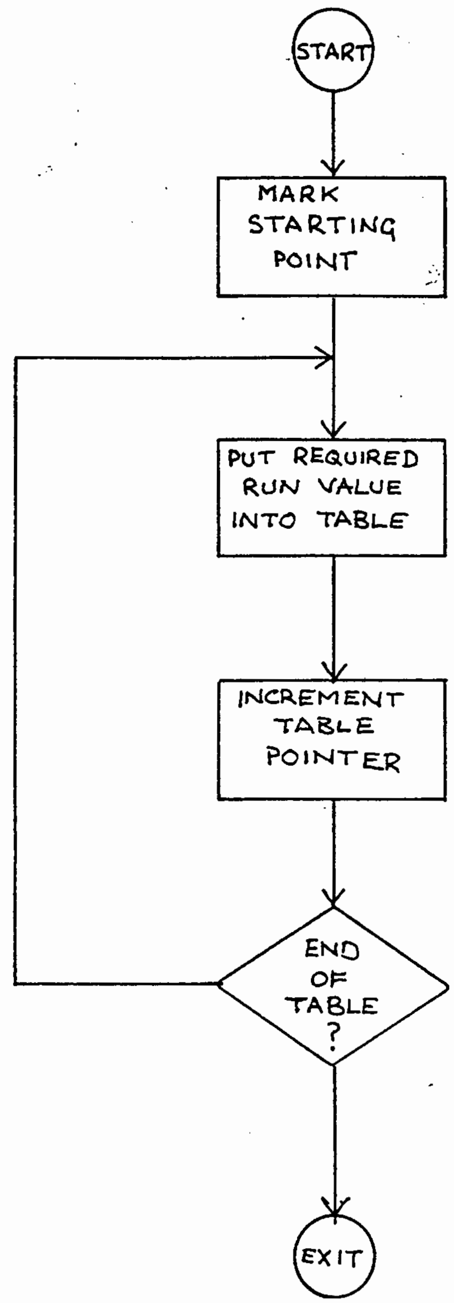


FIGURE 3.3.10: RUN PROCEDURE ALGORITHM

3.3.6 SEQUENCING TASK.

The sequencing task is the interface between the various digital signals required by the hardware, and the internal state of the software. There are three functions that have to be performed :

1) Monitor plant interlocks (such as the ON and OFF buttons), and modify the state of the system accordingly.

2) Output status information about the state of the system, and open and close interlock relays.

3) Monitor the internal state of the system software, take appropriate action and issue information if illegal states occur.

3.3.6.1 THE SANS STANDARD SEQUENCE TASK.

This task was adapted from a standard module developed by SANS for sequence control on PDP-11 computers. The use of standard modules reduces software development time, and makes software maintenance and readibility by other programmers far simpler. The task actually consists of three modules in the development stage, and two modules in the final system, namely :

1) A sequence timing module which ensures that the correct sequence gets called at the correct time. The module was developed for a multi-tasking process control system, and so there is provision for as many sequences as required by a system. Each sequence can be executed at a regular interval set by the sequence. Each step in a sequence is timed and has a timeout limit, and each sequence can be stopped, held or adjusted. Figure 3.3.11 shows the flow diagram of this module.

2) Sequence execution modules. These form the body of each sequence and consist of main sequence steps, and sequence sub steps. A main step corresponds with the state of the machine eg stopping, synchronising or ramping up. Sub-steps are the

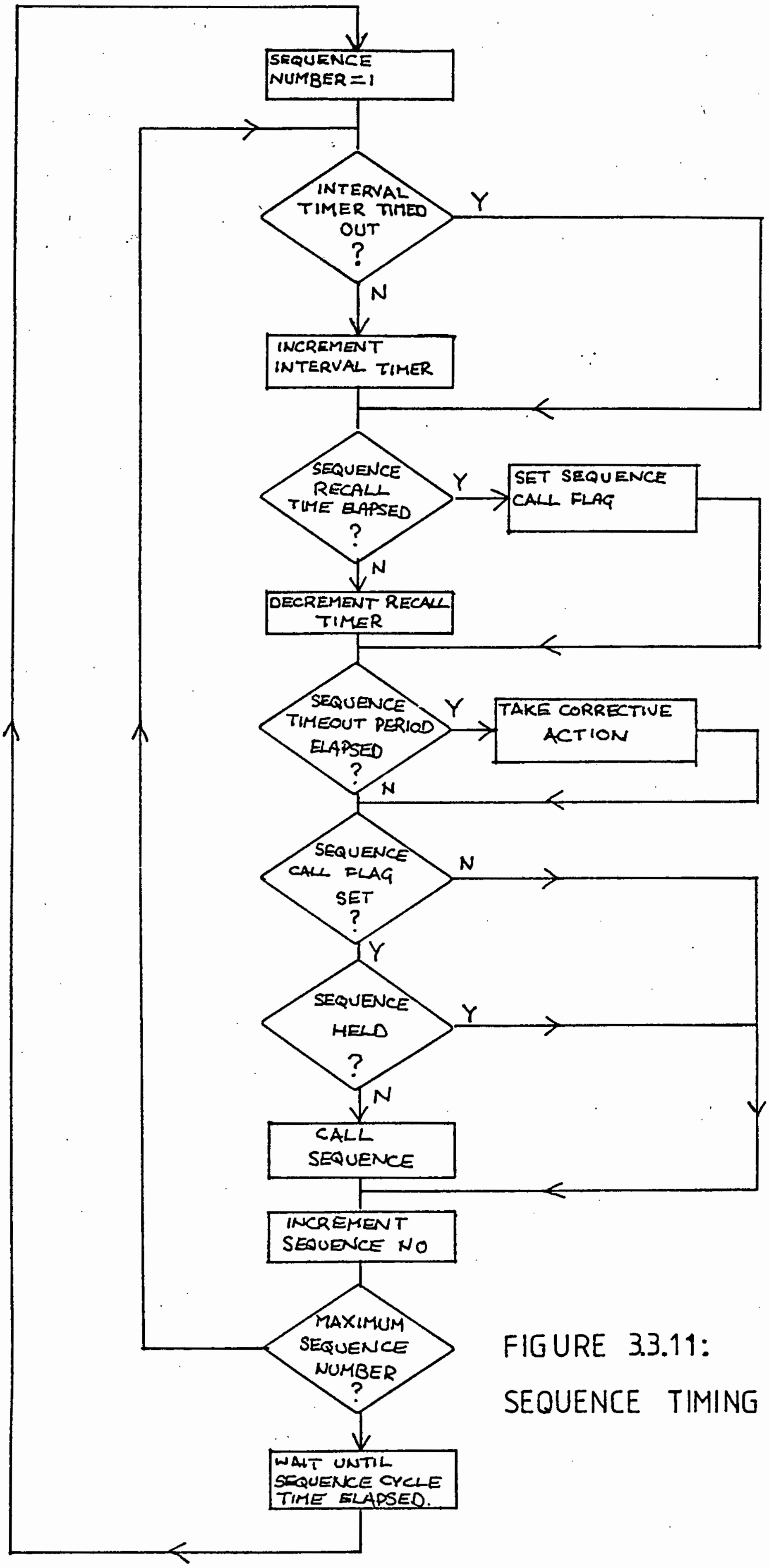


FIGURE 3.3.11:

SEQUENCE TIMING MODULE

individual steps that have to be performed eg check that the emergency stop button has not been pressed, close the start relay contacts, flash the warning light. Each sequence execution module sets its own Main and sub step values according to the information it receives from the plant interlocks. Figure 3.3.12 shows the flow diagram of this module.

3) Sequence Operator Command Processor (OCP). This module is only used in the testing and debugging phases. Through a local terminal, each sequence can be stopped, held where it is or have a hold set for a future step. It also enables the sequence main and sub step values to be set as required, so that the detailed operation of each step can be tested. The module is straight forward and directly manipulates the sequence data, so it will not be described further.

The data for the sequence task is held in two data bricks. One is local to the sequence timing module and holds the basic cycle time for the timing module, an array of the sequence execution procedures, and a matching array of delay, timeout and hold counters. There is a second data brick in the global data base which holds an array of step variables and flags for each sequence.

3.3.6.2 THE INVERTER CONTROL UNIT SEQUENCING TASK.

This task is made up of the sequence timing module and a sequence execution module. It has the following features :

1) There are no events in the plant which have to be synchronised with internal system events, so the sequence timeout feature of the timing module is not used.

2) There is only one sequence execution module.

3) The basic cycle time of the timing module is one second. This choice resulted from the need for a reasonably quick response time to plant interlocks (eg the emergency stop button).

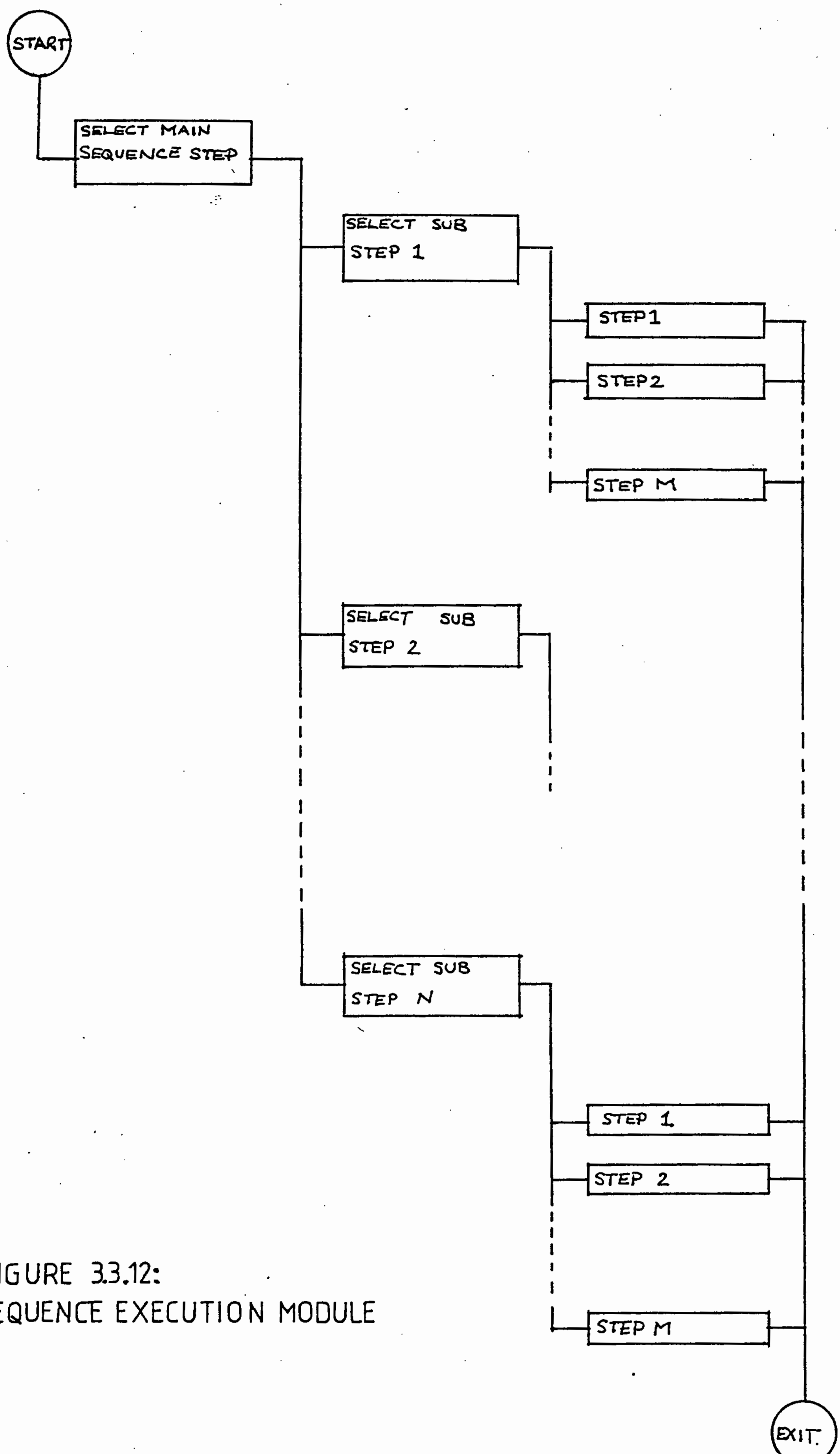


FIGURE 3.3.12:
SEQUENCE EXECUTION MODULE

4) A watchdog timer has been added to the timing module, which turns an indicator LED on and off every second if six critical tasks in the system are functioning correctly. It uses the fact that every task has to cycle within a certain period. If a task fails to complete a cycle in that period, then there must be something wrong with it. Each of the six tasks loads a count value into an array each time it completes a cycle. When the sequence task cycles, it decrements all the elements of the array and checks to see if any of them have reached zero. If any have reached zero it means that they have been held up for longer than the allowable period, so an error message is printed saying which task has failed, the LED is turned off, and the sequence alarm flag is set which causes the motors to ramp to a stop.

3.3.6.3 SEQUENCE TASK INPUTS AND OUTPUTS.

As mentioned previously, this task forms the interface between the plant and the computer system. Each of the signals and conditions that affect the state of the system will now be considered.

A) INPUTS FROM THE PLANT.

1) THE READY SIGNAL. As described in Chapter 2.2, the start, stop, emergency stop, oil mist fail, thermistor trip, wrap detector, winder inverter trip and traverse inverter trip inputs are AND'ed together into a single digital input. If this input is "TRUE" the winder is running, and must ramp up to operating speed. If it is "FALSE" the winder is stopping, and must ramp down to a standstill.

2) THE WINDER TACHOMETER. The sequence task uses this tachometer to set a system event which causes any new operating parameters which have been sent to the controller to become the current operating parameters. Updating of the running parameters has to occur when the chuck is stationary ie when the winder is being "doffed". This means that new operating conditions are synchronised with the start of a new package, rather than in the

middle of a package.

B) OUTPUTS TO THE PLANT.

1) WINDER RUNNING INDICATOR. The computer turns this LED on once both the winder and traverse rolls are at their operating speeds. This indicator is relayed to the machine head, and tells the operator that the machine is ready for use.

2) WINDER INVERTER RUN RELAY. This signal closes a set of contacts in the inverter which enables it to be started. The contacts are closed when the READY signal is TRUE, and opened when the READY signal is FALSE.

3) TRAVERSE INVERTER RUN RELAY. As for the winder run relay.

C) INPUTS FROM THE COMPUTER SYSTEM.

The sequence task monitors the frequency of the pulses being fed to the inverters for two reasons :

1) To ensure that the inverters do not start unless the output is at idling frequency. This means that once the stop button has been pressed, the motors cannot be restarted until they have ramped to a standstill.

2) To switch the RUN indicator on when both the winder and the traverse are up to speed.

D) OUTPUTS TO THE COMPUTER SYSTEM.

These are binary flags which signal to the rest of the system what phase of operation is required. The sequence task controls three of these flags. When any of them are set, the others will be cleared.

1) The STOP flag. This is set whenever the READY signal is FALSE, and signals to the system that the stopping phase has been

entered. All other flags are cleared when the STOP flag is set.

2) The SYNC flag. This is set for the first two seconds after the READY signal has been set, and allows the synchronous winder motor to get itself into lock.

3) The START flag. After two seconds, the SYNC flag is cleared and the START flag is set. This results in a steadily ramping pulse train being fed to the inverters.

Once both inverters have reached their required operating speed, all three of these flags are cleared, and the ratio task takes over control of the internal state of the controller.

Figure 3.3.13 shows a flow diagram of the combined operation of the sequence execution and sequence timing modules for the controller.

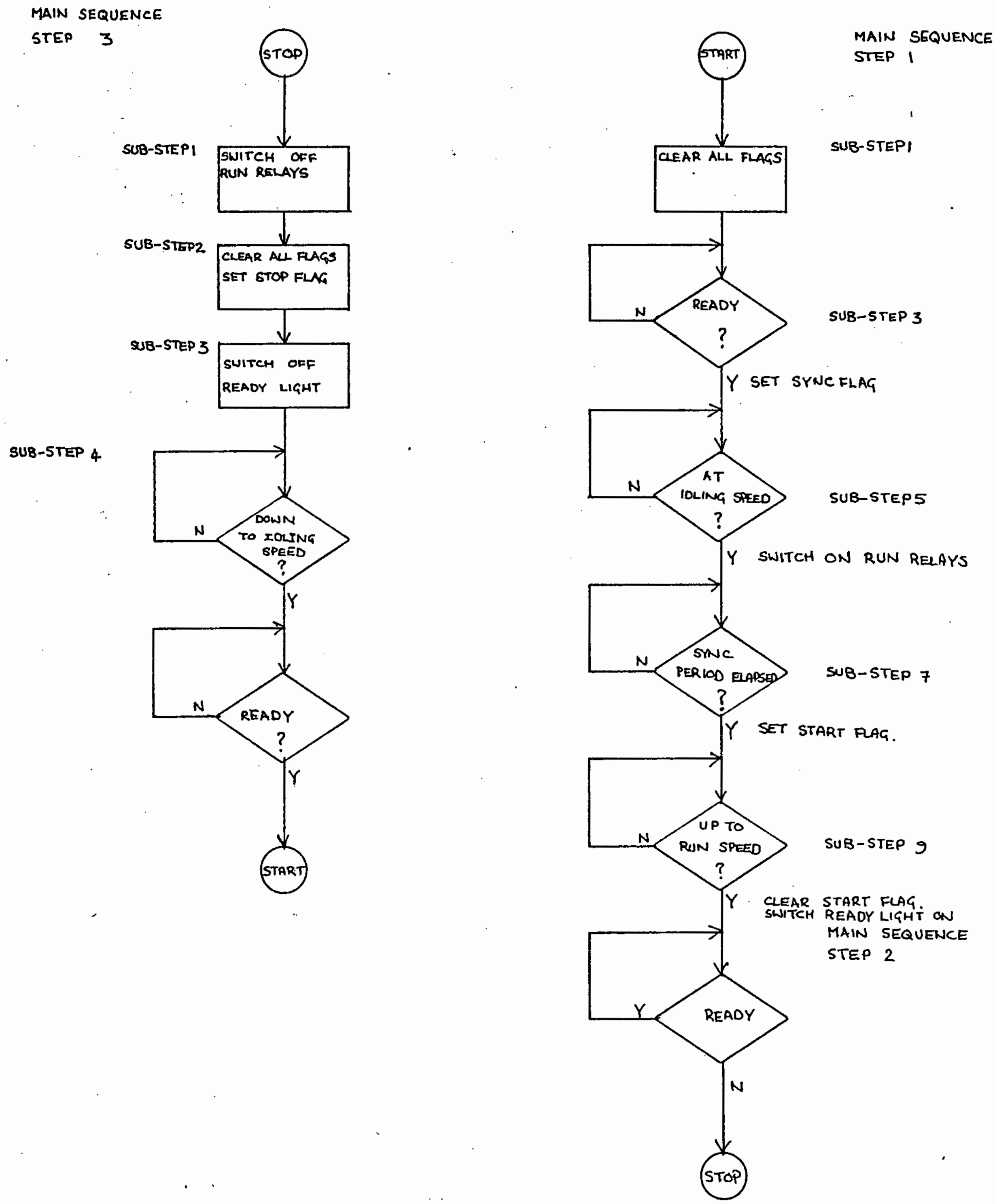


FIGURE 3.3.13: SEQUENCE TASK OPERATION

3.3.7 THE OPERATOR COMMAND PROCESSOR (OCP) TASK.

This task allows an operator to enter operating parameters into the controller via a local VDU. It converts the parameters from a form familiar to the operator into the form required by the computer. For example, the operator is used to setting the traverse and winder speeds in (REAL) Hertz. However the computer uses the number of (INTEGER) counts of a 2.456MHz clock to represent this frequency. Clearly there has to be a conversion between the two.

SANS has a large commitment to computer systems for process management and process control, and there are numerous terminals on the factory floor which are used by operators. As a result the company has attempted to evolve a standard menu driven operator interface. This section will examine the thinking underlying the OCP task, and then it will look at the operation of the software.

3.3.7.1 REQUIREMENTS FOR AN OCP.

The standards for operator communication adopted by SANS were based on those developed by an ICI team in England, who encountered problems when numerous projects were developed by many different people over a long time. There are two groups of people who stand to gain from standardisation, namely operators and programmers. Most machine operators are unfamiliar with computers, so it is desirable to establish a simple and consistent approach to operator communication to minimise training, and given the complexity of some plants, reduce the risk of error. From the perspective of the programmer, standardisation reduces development time and makes it easier to understand and maintain a piece of software written by another person.

The ICI development team found that there was no consistency of approach between projects, and much effort was wasted through duplication and having to rewrite software for each job. The result was programs that were often difficult to use and read. To overcome these problems the software had to be versatile enough to handle all situations, had to have standard and self explanatory procedure names, had to be in globally accessible modules, and had to avoid hardware specific features.

Experienced operators should be able to interact quickly and efficiently, but when mistakes are made or inexperienced operators use the system, additional information should be available. The solution was a menu driven system that lists the available options and prompts for a reply. Typing "X" (for e"X"planation) provides additional information. Validity checking is performed on operator entries, and invalid entry produces an error message and analysis. To prevent unauthorised use of the system, security checks have to be passed before the data can be altered.

3.3.7.2 OCP SOFTWARE.

Because operating parameters can only be updated when the winder is "doffed" (see Appendix A : "Introduction to Nylon Spinning" for a description of terms used in Nylon Spinning), and because the computer uses different forms of the parameters from the operator, it was necessary to have three sets of data for the OCP. Figure 3.3.14 shows a block diagram of the relationship between these three data bricks, each of which will now be decribed in detail.

A) CURRENTLY ACTIVE PARAMETERS.

Firstly there is the data that is active at present. For efficiency it is held in the form most suitable for the computer hardware, so that conversion only has to occur once when the data is entered, rather than having to be converted every time it is required. This has the added advantage of reducing the amount of RAM needed, as computer hardware generally uses byte and word orientated data, whereas people tend to use real data, which requires four bytes for each variable. This resulted in a 75%

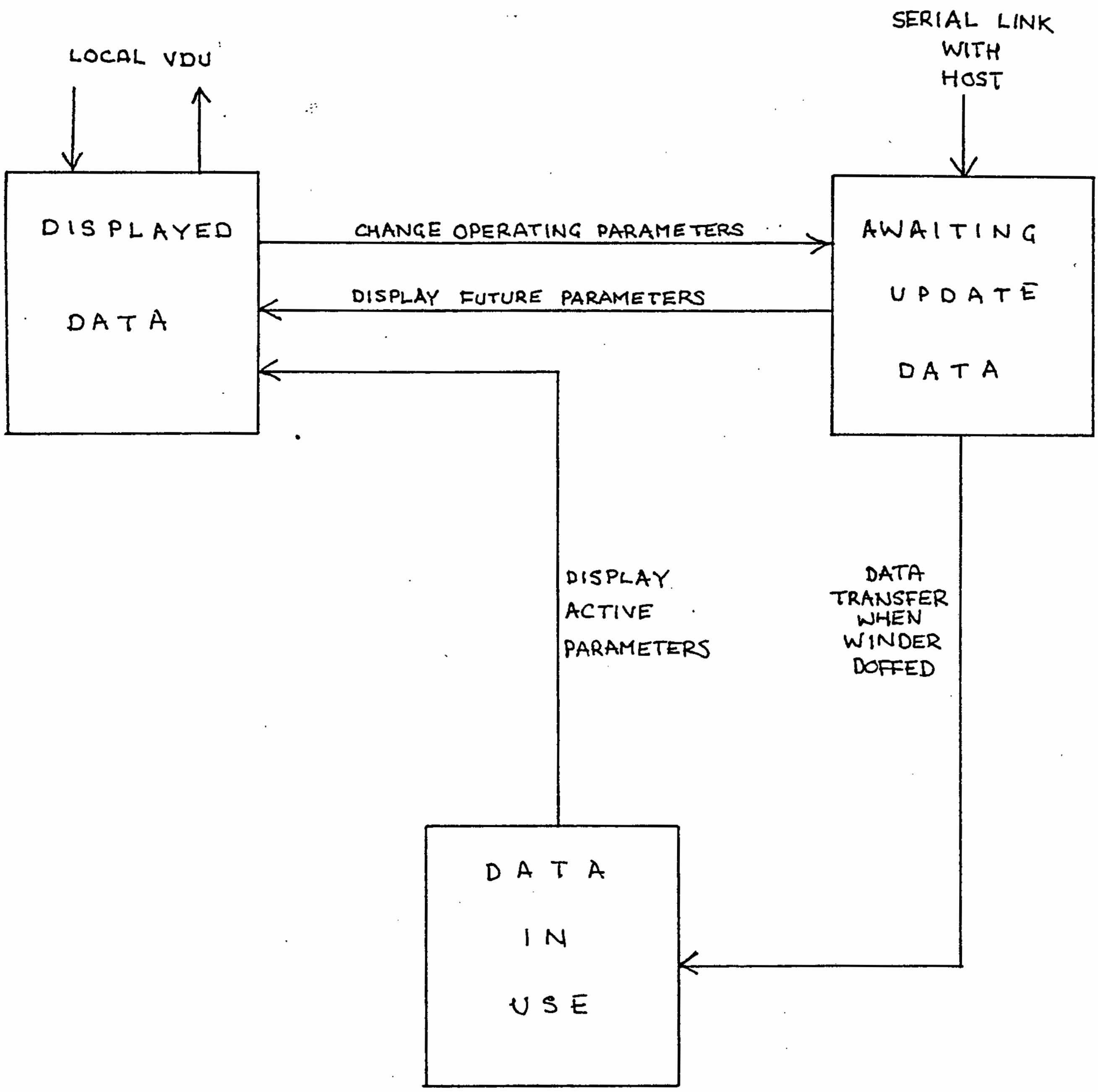


FIGURE 3.3.14: OVERVIEW OF OCP DATA

reduction in the amount of RAM required for the traverse and winder control tables, which have 128 entries each.

B) FUTURE PARAMETERS.

The second data brick is a temporary storage area that holds the newly entered operating parameters until the machine is doffed and the new parameters can become the operating parameters, which ensures that a change of operating conditions does not occur in the middle of a product run. This data brick is updated either by an operator entering values through a local VDU, or via the serial multidrop link with the host. It is an exact replica of the main data brick, using the same variable names prefixed with "AW" to indicate that they are "AW"aiting update. There is a set of flags which indicate which variable (if any) has been altered. When the speed of the chuck falls below 1500 RPM, the sequence task sets an event which triggers a database update task, which copies the contents of the awaiting update data into the main data brick.

C) DISPLAYED PARAMETERS.

The third data brick is a set of variables that hold the data in a form suitable for the operator. Valid parameters entered by the operator are converted into the form required by the computer and copied into the awaiting update data brick when the operator has passed a security check. The operator can view the parameters in the computer by the inverse conversion. There is one further refinement in that the most recently entered parameters are displayed. If these are the running parameters, then a message on the VDU informs the operator that he is viewing the "ACTIVE PARAMETERS". If these parameters have been altered but the main data base has not yet been updated, then the message indicates that these are the "FUTURE PARAMETERS".

The operation and use of the OCP is fully described in Chapter 4.2. The software for the OCP task is fairly straightforward, and is illustrated in the flowchart in Figure 3.3.15.

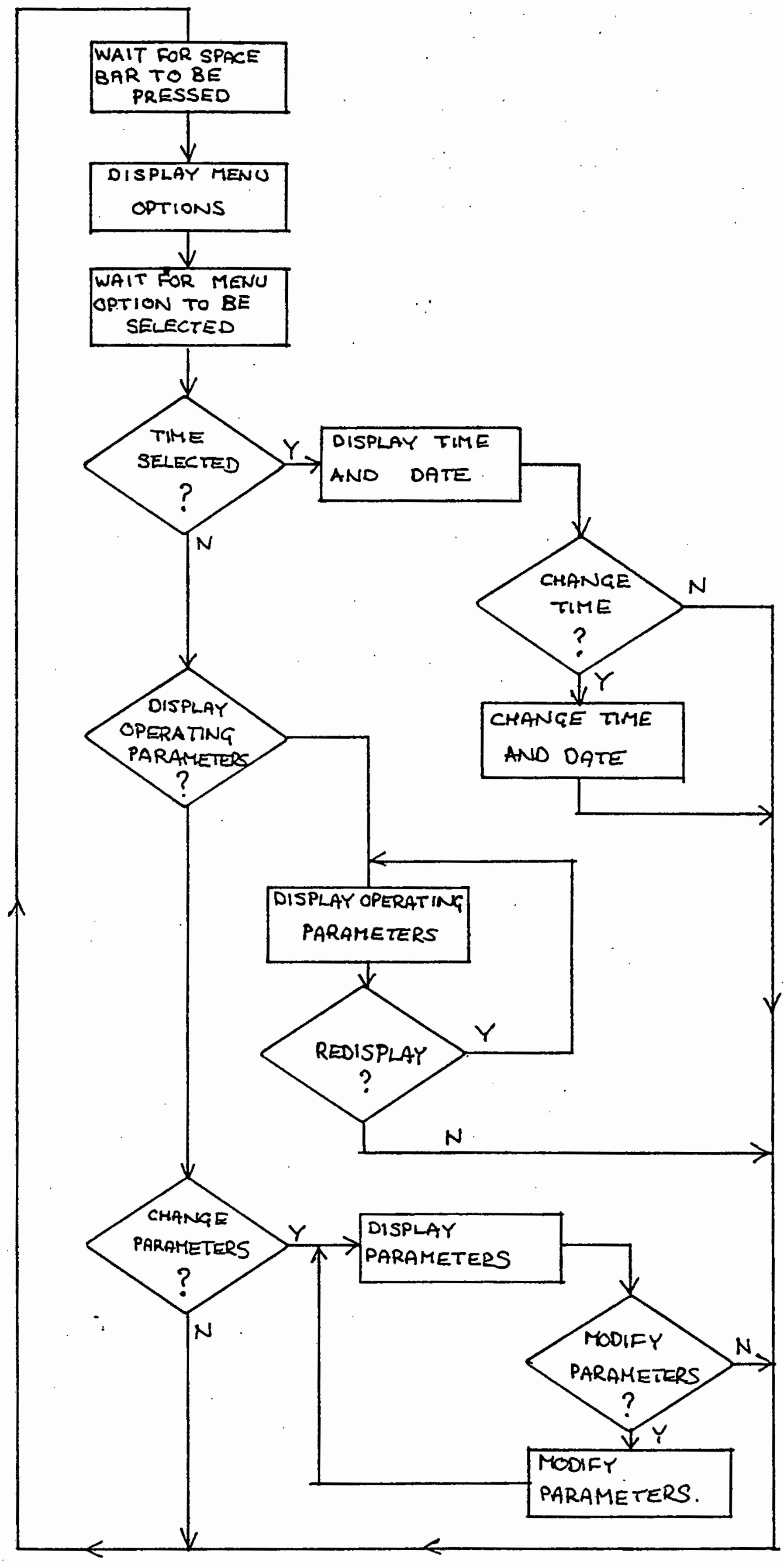


FIGURE 3.3.15:
OCP TASK ALGORITHM

3.3.8 TRAVERSE DRIVE TASK.

This task is identical in operation to that of the winder control task described in chapter 3.3.4. It reads a value from the traverse control table every 100mS and outputs it to the traverse drive RTC. The control table is filled with values by the traverse control task.

The only difference from the winder drive task is that allowance has to be made for traverse P-jumps. The maximum step acceleration needed for the P-jump is far greater than the limiting acceleration rate of the inverter. This meant that the inverter would not have been able to achieve the desired modulation characteristics. However, the inverters can withstand an instantaneous step acceleration much larger than the greatest continuous acceleration. The inverters have two current overload trips : a maximum instantaneous current trip; and an average current trip derived by integrating the load current. The integrated current trip is the one which limits the continuous acceleration rate, and the maximum current trip is the one which limits the step acceleration. If the average of the peak currents is more than the average current set point, the inverter will trip. Thus empirical tests were run on the inverters to determine the limiting repetition rate for the maximum P-jump required. In practice the inverters were able to meet the worst conditions called for by the specification

In the software for the traverse drive task, the check for the maximum acceleration or deceleration is modified so that the limiting rate is set to a different value if a P-jump is being executed. Figure 3.3.16 shows a flow diagram of the traverse drive task.

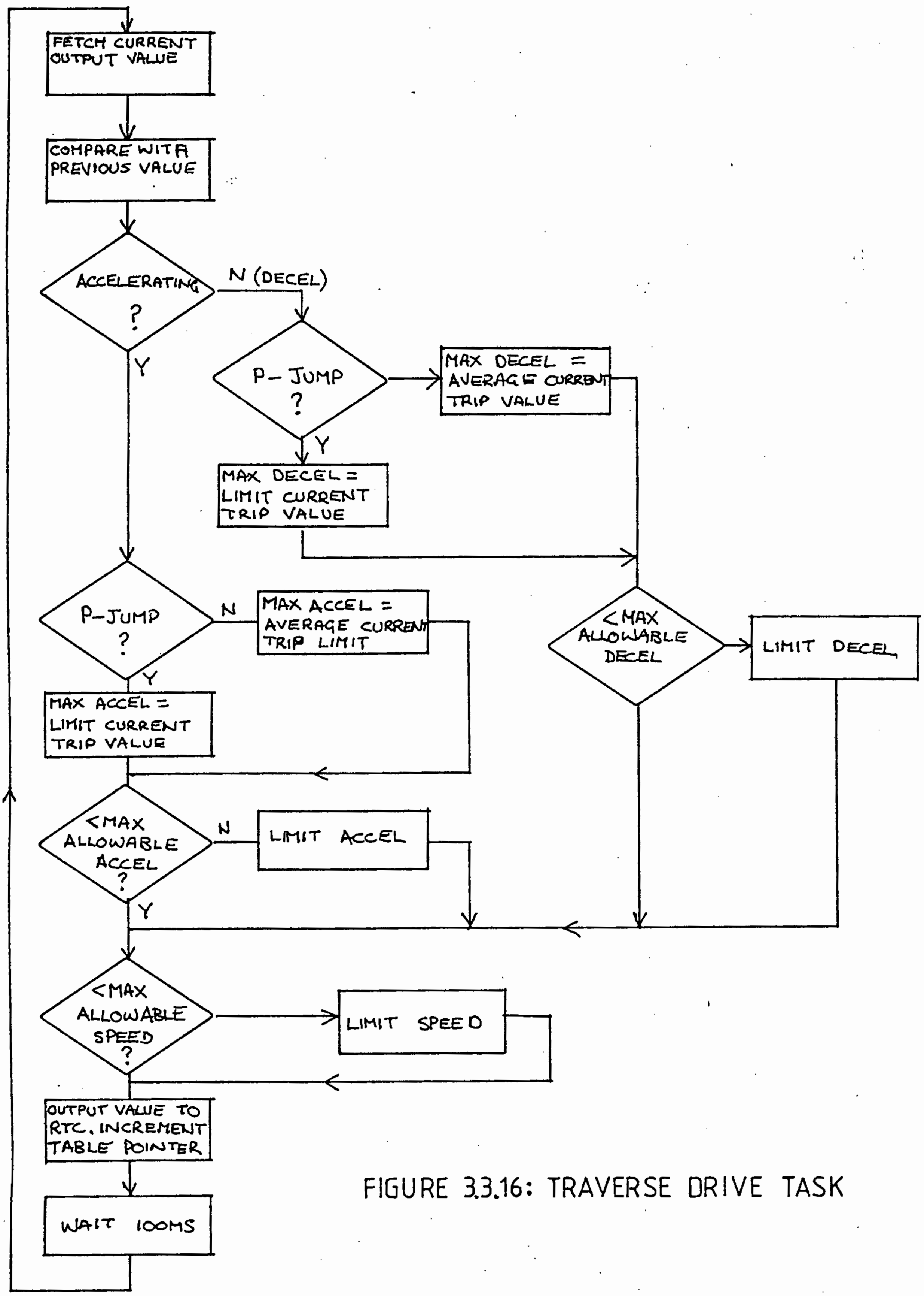


FIGURE 3.3.16: TRAVERSE DRIVE TASK

3.3.9 TRAVERSE CONTROL TASK.

This task is essentially the same as the Winder control task described in section 3.3.5, except that traverse modulation and programmable banding avoidance must be provided in place of the normal "RUN" operation of the winder. Section 3.3.5 should be consulted for a description of the requirements and limitations that led to the final design of the control task software. Figure 3.3.17 is a flow diagram of the traverse control task.

The traverse has five states of operation :

1) Stopped or ramping down to a standstill.
2) Start up synchronisation.
3) Ramp up to operating speed.
4) Traverse modulation.
5) Banding avoidance.

The first three states have been described in chapter 3.3.5. Figure 3.3.18 is a plot of frequency vs time showing all these phases of operation. The modulation amplitude and banding avoidance speed change have been greatly exagerrated for clarity. Each of the labelled points of operation in Figure 3.3.18 will now be examined in more detail.

A) Shows the start up synchronisation phase. The traverse runs at 13.3 Hz until the winder motor has achieved synchronisation, and then ramp up occurs.

B) Traverse ramps up to operating speed.

C) Traverse reaches normal operating speed. This is known as the "F1" speed.

D) Traverse modulation does not start until the winder roll has reached its operating speed. (At this point the sequence task switches the "WINDER READY" light on, which signals to the

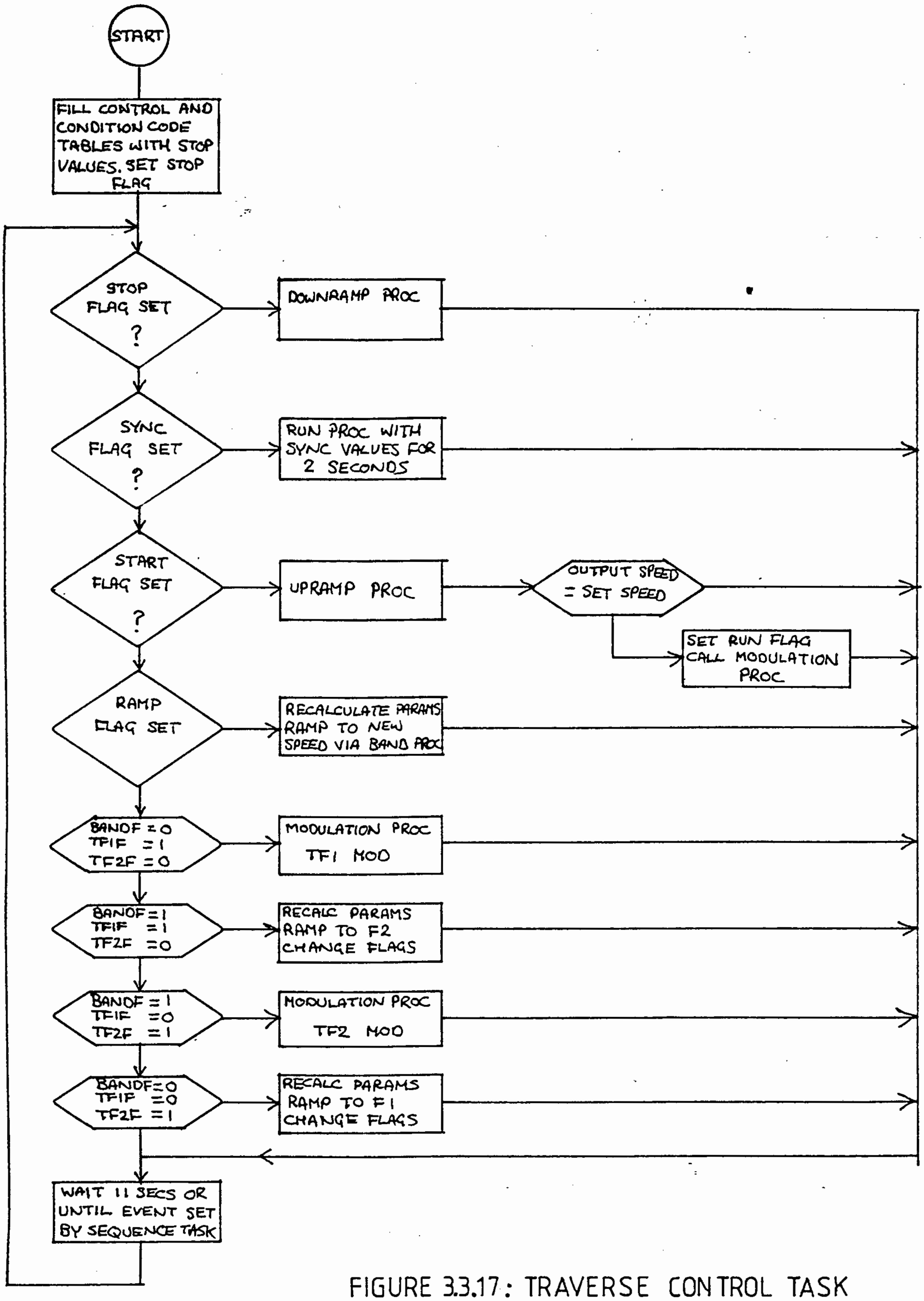


FIGURE 3.3.17: TRAVERSE CONTROL TASK

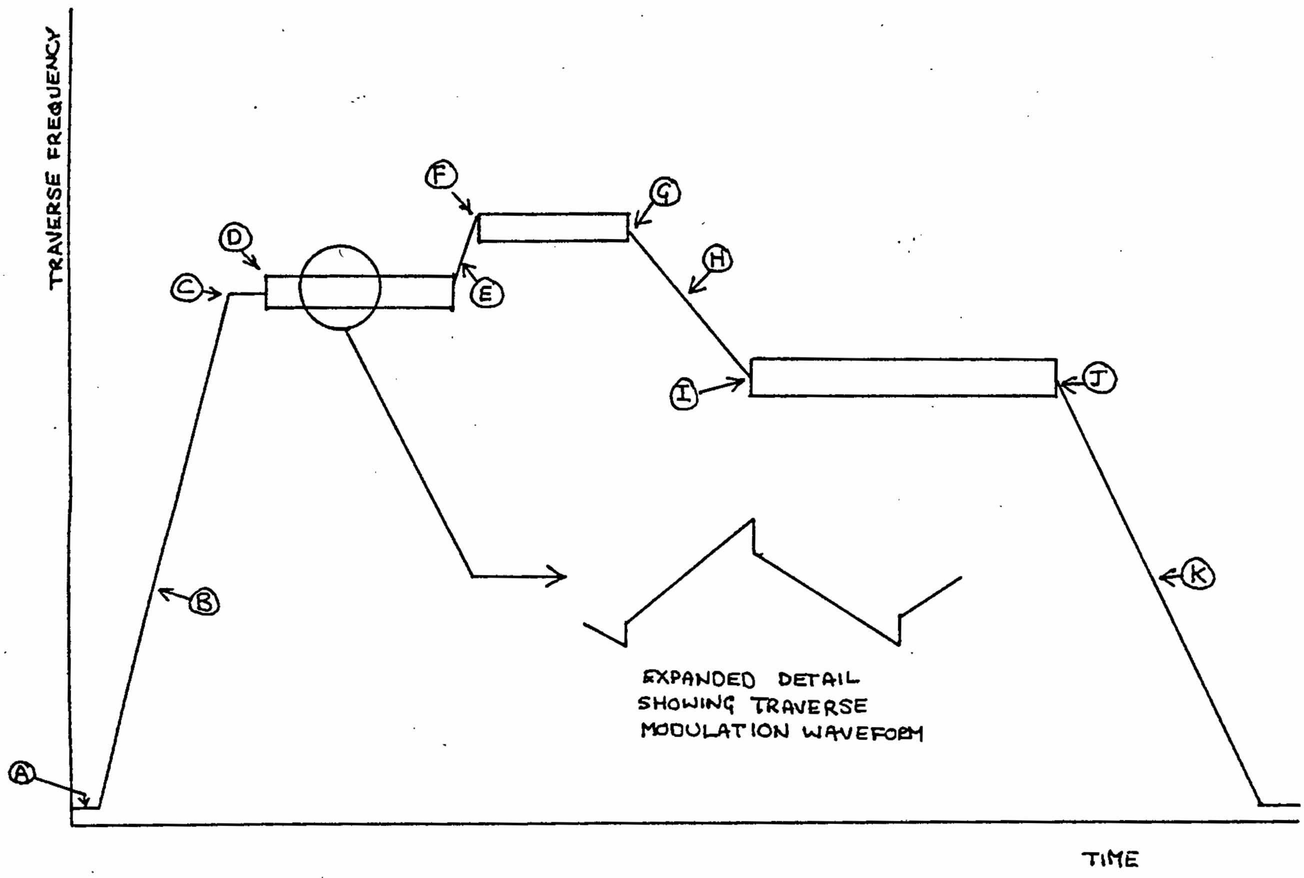


FIGURE 3.3.18: TRAVERSE SPEED VS TIME

operator that the machine can be used.)

E) Banding avoidance begins. The ratio task has detected that the ratio of the chuck and traverse speeds has reached a critical point where banding is about to occur, and has set the banding avoidance flag. Modulation is stopped, and the traverse ramps to its new speed. The rate of ramping is one of the process variables set up by an operator via the local VDU or host link.

F) The banding avoidance speed is reached. This is called the "F2" speed. This speed can take any value, and could have been less than the F1 value. F2 modulation begins.

G) Banding avoidance ends. The ratio task has detected that the need for banding avoidance has ended, and has cleared the banding avoidance flag.

H) Modulation stops, and the traverse ramps to the new F1 frequency. This ramp rate is also a process variable set by an operator.

I) The new F1 frequency is reached, and F1 modulation begins again. The new F1 speed is also a process variable which can be set to any value, and does not have to be the same as the first F1 speed at D.

J) The machine is stopped either by the operator or because a fault condition has occurred.

K) Modulation stops and the traverse ramps to a standstill.

Only one banding avoidance point was shown. In practice up to 15 points can be specified, each with its own F1 and F2 set speeds. This enables a close control of wind on tension and package build to be obtained. See Appendix A for a description of why this traverse behaviour is required.

3.3.9.1 THE CONTROL FLAGS.

During the normal run phase, the traverse has four phases of operation :

1) F1 modulation.
2) Ramp to F2 operation
3) F2 modulation.
4) Ramp to F1 operation.

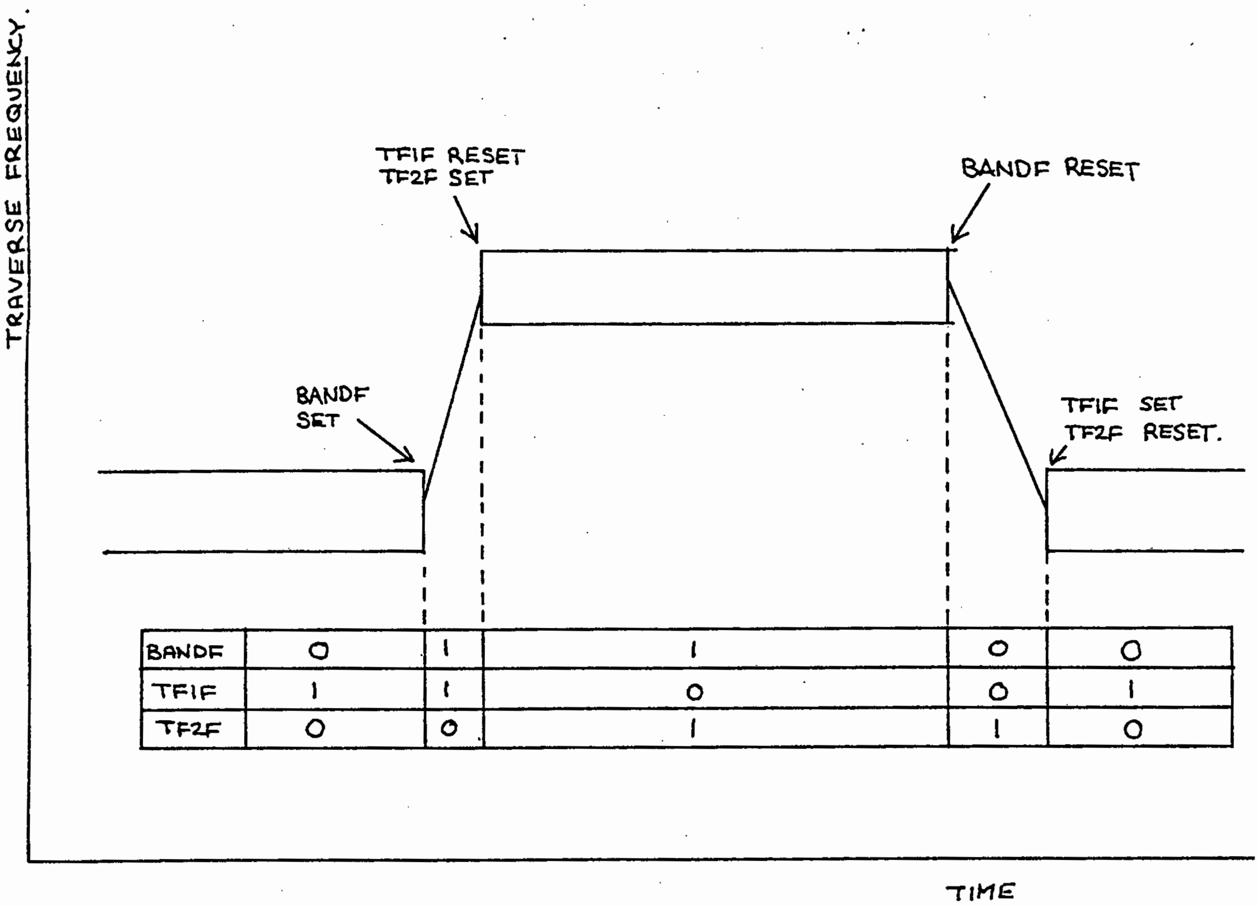
Three flags were required to uniquely specify each phase of operation : one to indicate F1 operation; two to indicate F2 operation; three for banding avoidance. Figure 3.3.19 shows the flag settings for each phase. The control task examines these flags and calls the appropriate procedure according to their status.

Another flag called the "RAMP" flag was required. When the operating parameters have been altered and the winder has to operate at a new speed, the large step changes at P-jumps allowed by the traverse drive task must be disabled while the traverse ramps to its new set point. When the ramp flag is set the control task calls the banding procedure with dummy parameters so that a banding avoidance speed change is performed. This results in a smooth ramp to the new operating speed.

3.3.9.2 THE MODULATION PROCEDURE.

The modulation waveform has four components : a positive speed ramp to the modulation amplitude maximum; a negative P-jump; a negative speed ramp to the modulation amplitude minimum; a positive P-jump. The control task finds the point where the drive task is reading values from the control table, and starts generating new values a few places ahead. Since this point is arbitrary, it is necessary to determine the modulation phase being executed, as well as the stage of the phase that has been reached.

Various methods were tried. All of them involved scanning the control table values and trying to fit the observed pattern with



BANDF	0	1	1	0	0
TFIF	1	1	0	0	1
TF2F	0	0	1	1	0

FIGURE 3.3.19 TRAVERSE MODULATION CONTROL FLAGS

reference patterns, which proved to be clumsy and time consuming. Eventually each phase was given a number, and a "condition code" table was used. Each entry in this table matches the control table and holds the modulation phase or condition code. The condition code values are as follows :

0 - No modulation ie mean speed change in progress.
1 - Positive speed ramp to maximum modulation amplitude.
2 - Negative P-jump.
3 - Negative speed ramp to minimum modulation amplitude.
4 - Positive P-jump.

Figure 3.3.20 is a detailed diagram of the modulation waveform, showing the condition code values for the different phases. With this table the phase of operation can be determined immediately, and then it is only necessary to find how many 100mS steps from the end of the phase the calculation point is. This can be be done by comparing the output speed at the calculation point with the target speed for that phase.

3.3.9.3 GENERATING THE TRAVERSE MODULATION WAVEFORM.

The parameters given in the operating instruction are TFXSPEED for the required mean speed, TFXAMP for the modulation amplitude, TFXPJ for the P-jump amplitude, and TMPERIOD for the modulation period. All the other values are derived from these four. The simplest way to generate the control table values is to fix each of the end points in the waveform and then interpolate between them. Figure 3.3.20 shows these end-points and the variable names used for them in the software, namely TFXUPSTART, TFXUPSTOP, TFXDOWNSTART and TFXDOWNSTOP, whose names are self explanatory. TRAMPTIME is the ramp duration (in 100mS counts). THOLD is the P-jump hold period (also in 100mS counts).

The prefix "TFX" is used because these are "T"raverse variables, and can be set for either non banding (F1) or banding (F2) operation. Variables not given in the operating instruction are calculated by a procedure called PARAM (in module COMPROC) which

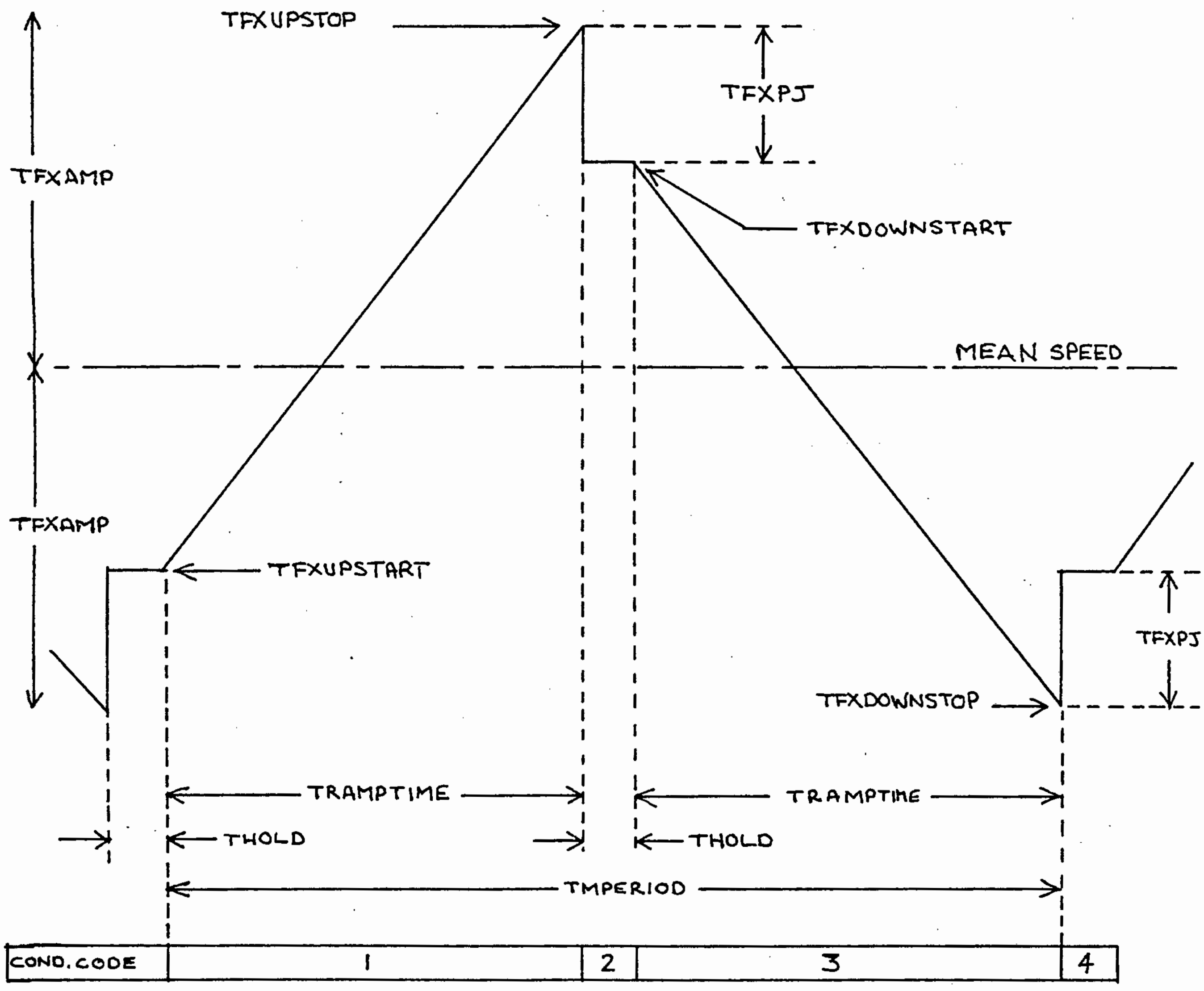


FIGURE 3.3.20 TRAVERSE MODULATION VARIABLES

gets called with the parameters for the current phase of operation. In what follows it must be remembered that the modulation waveform is symmetrical about the mean frequency, and that the modulation period remains the same for F1 and F2 operation. Each of the variables will now be derived.

A) TIME DURATIONS.

To prevent the traverse inverter from tripping when the maximum value of P-jump permissable is immediately followed by a continuous ramp, an inverter recovery time THOLD was inserted after the P-jump. It was made variable so that it could be dynamically altered to cope with different situations if required. In practice a value of 1 proved to be satisfactory for all cases. Thus :

Modulation period = (2 * TRAMPTIME) + (2 * THOLD)

Modulation period is held as seconds, so in 100mS counts :

10 * TMPERIOD = (2 * TRAMPTIME) + (2 * THOLD)

Rearranging and remembering that THOLD is given :

TRAMPTIME = (5 * TMPERIOD) - THOLD

B) CALCULATION OF RAMP UP PARAMETERS.

From figure 3.3.20 :

Start of upramp = mean speed - modulation amplitude + P-jump.

The modulation amplitude and P-jump are both held as percentages of the mean speed, thus :

Start frequency of upramp =

mean speed * (1 - mod amp / 100 + P-j amp / 100)

= mean speed * (100 - mod amp + P-j amp) / 100

Speed values are held as RTC counts (see section 3.3.5.3), and :

$$\text{RTC output frequency} = \frac{\text{RTC clock frequency}}{\text{RTC count}}$$

Thus :

$$\frac{\text{Clock freq}}{\text{start count}} = \frac{\text{Clock freq}}{\text{Mean count}} * \frac{(100 - \text{mod amp} + \text{P-j amp})}{100}$$

Rearranging :

$$\text{start count} = \frac{\text{Mean count} * 100}{(100 - \text{mod amp} + \text{P-j amp})}$$

Using the variable names of the program listing :

$$\text{TFXUPSTART} = \frac{100 * \text{TMFXSPEED}}{(100 - \text{TMFXAMP} + \text{TMFXPJ})}$$

Where : TMFXSTART = Start count of modulation upramp.
TMFXSPEED = Count corresponding to mean speed of traverse.
TMFXAMP = Modulation amplitude as % of mean speed.
TMFXPJ = P-jump amplitude as % of mean speed.

The end point of the ramp can be found in the same way since :

End point of upramp = mean speed + mod amp and :

$$TFXUPSTOP = \frac{100 * TMFXSPEED}{(100 + TMFXAMP)}$$

Ramping between the start and end points was achieved by finding the total speed difference and dividing it by the number of 100mS steps between the two points. Each individual step was found from :

speed at step = speed at previous step + step speed increment.

Thus step speed increment =

$$TFXUPINC = \frac{TFXUPSTOP - TFXUPSTART}{TRAMPTIME}$$

This yields a negative value as TFXUPINC, TFXUPSTOP and TFXUPSTART are all RTC count values which are inversely proportional to frequency, so to increase frequency, the count value must be decreased. Since smooth ramping is required under all conditions TFXUPINC was held as a REAL variable, and the ramp calculation was done as a REAL, conversion to INTEGER count only taking place when the calculation was complete.

C) CALCULATION OF RAMP DOWN PARAMETERS.

The same techniques were used for the ramp down phase of the modulation waveform, so :

start of downramp = mean speed + mod amp - P-jump amp or :

$$TFXDOWNSTART = \frac{100 * TMFXSPEED}{(100 + TMFXAMP - TMFXPJ)}$$

and :

end of downramp = mean speed - mod amplitude.

$$TFXDOWNSTOP = \frac{100 * TMFXSPEED}{(100 - TMFXAMP)}$$

and :

$$TFXDOWNDEC = \frac{TFXDOWNSTOP - TFXDOWNSTART}{TRAMPTIME}$$

This yields a positive value since the count value of TFXDOWNSTOP is greater than that of TFXDOWNSTART.

D) THE P-JUMPS.

An examination of Figure 3.3.20 shows that the P-jump values are really TFXDOWNSTART and TFXUPSTART held for THOLD 100mS steps. So once the ramp end points were reached a step change was made to the ramp start point.

E) THE TRAVERSE MODULATION PROCEDURE.

Figure 3.3.21 shows the flow diagram of this procedure. It is called with the traverse control table pointer and the desired mean speed passed as parameters. The modulation parameters are picked up from the global database.

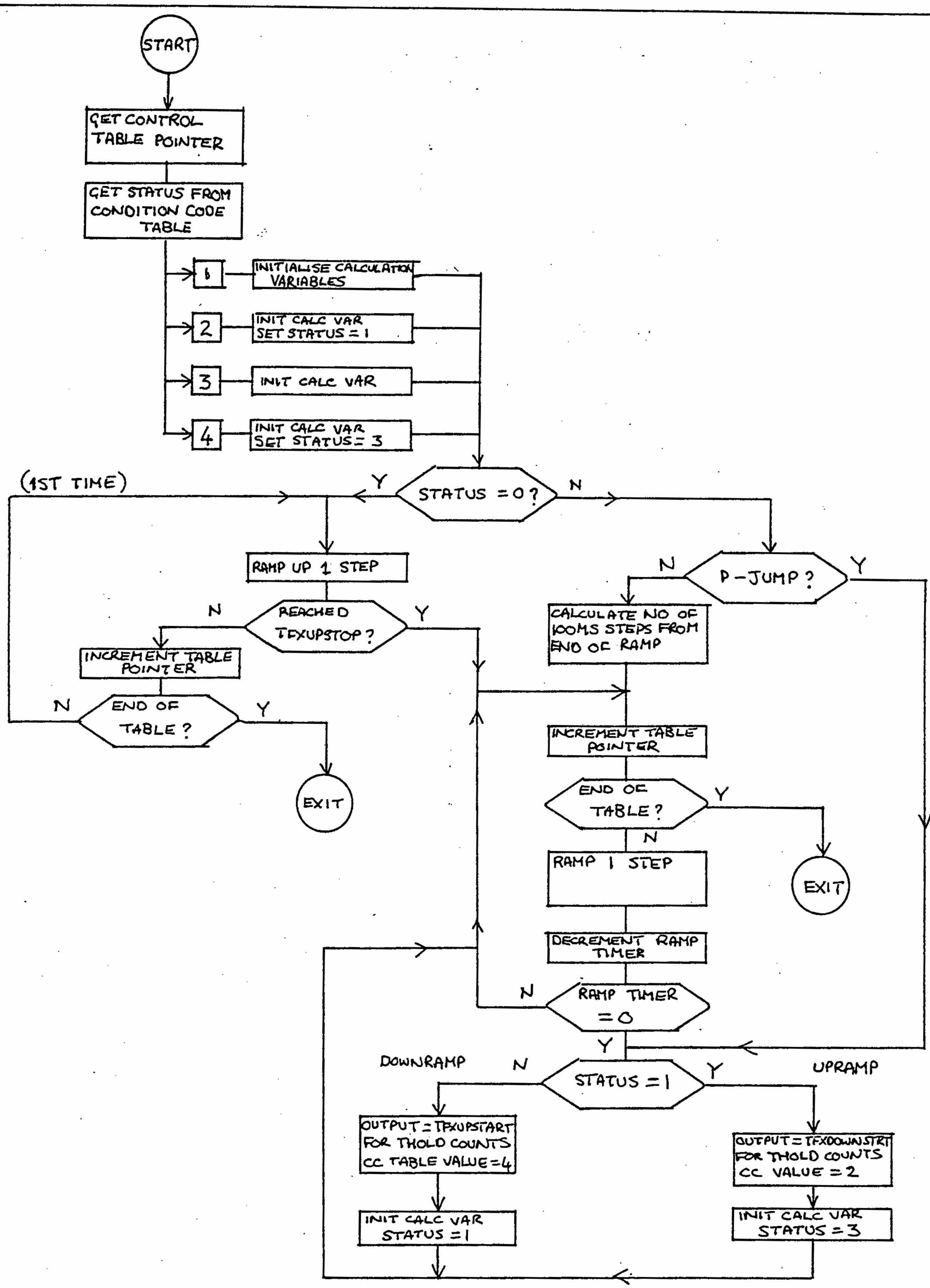


FIGURE 3.3.21: TRAVERSE MODULATION PROCEDURE

3.3.9.4 BANDING AVOIDANCE.

The banding avoidance procedure is held in the module COMPROC along with the other winder and traverse control procedures. It is used whenever the mean speed of the traverse has to be altered, either during banding avoidance, or when a new operating instruction requiring a different mean speed is entered. It is called with the target mean speed, the acceleration rate and control flags passed as parameters.

While ramping is in progress the corresponding condition code table values are set to zero. The procedure ramps the traverse speed at a rate set by one of the parameters passed to the procedure. This ramp rate is in Hz / sec, and so the corresponding rate in counts / sec must be determined. Thus :

unit ramp rate = (new frequency - old frequency) in unit time.

Converting the frequencies into their equivalent count values :

$$\text{rate} = \frac{\text{clock freq}}{\text{new count}} - \frac{\text{clock freq}}{\text{old count}}$$

Rearranging in terms of new count :

$$\text{new count} = \frac{\text{clock freq} * \text{old count}}{\text{rate} * \text{old count} + \text{clock freq}}$$

Ramp rate in terms of old counts only is given by :

$$\text{Unit ramp rate} = \text{old count} - \text{new count}$$

$$= \text{old count} - \frac{\text{clock freq} * \text{old count}}{\text{rate} * \text{old count} + \text{clock freq}}$$

or in terms of the variable names used in the listings :

$$SLOPE = TPITVAL(TCALCP) * \left(1 - \frac{CLOCKFREQ}{RATE * TPITVAL(TCALCP) + CLOCKFREQ}\right)$$

Where : SLOPE = ramp rate counts per 100mS.
RATE = ramp rate in Hz per 100mS.
TPITVAL(TCALCP) = control table value being output.
CLOCKFREQ = RTC clock frequency. (2.456 MHz)

A flow diagram of the banding procedure is given in figure 3.3.22.

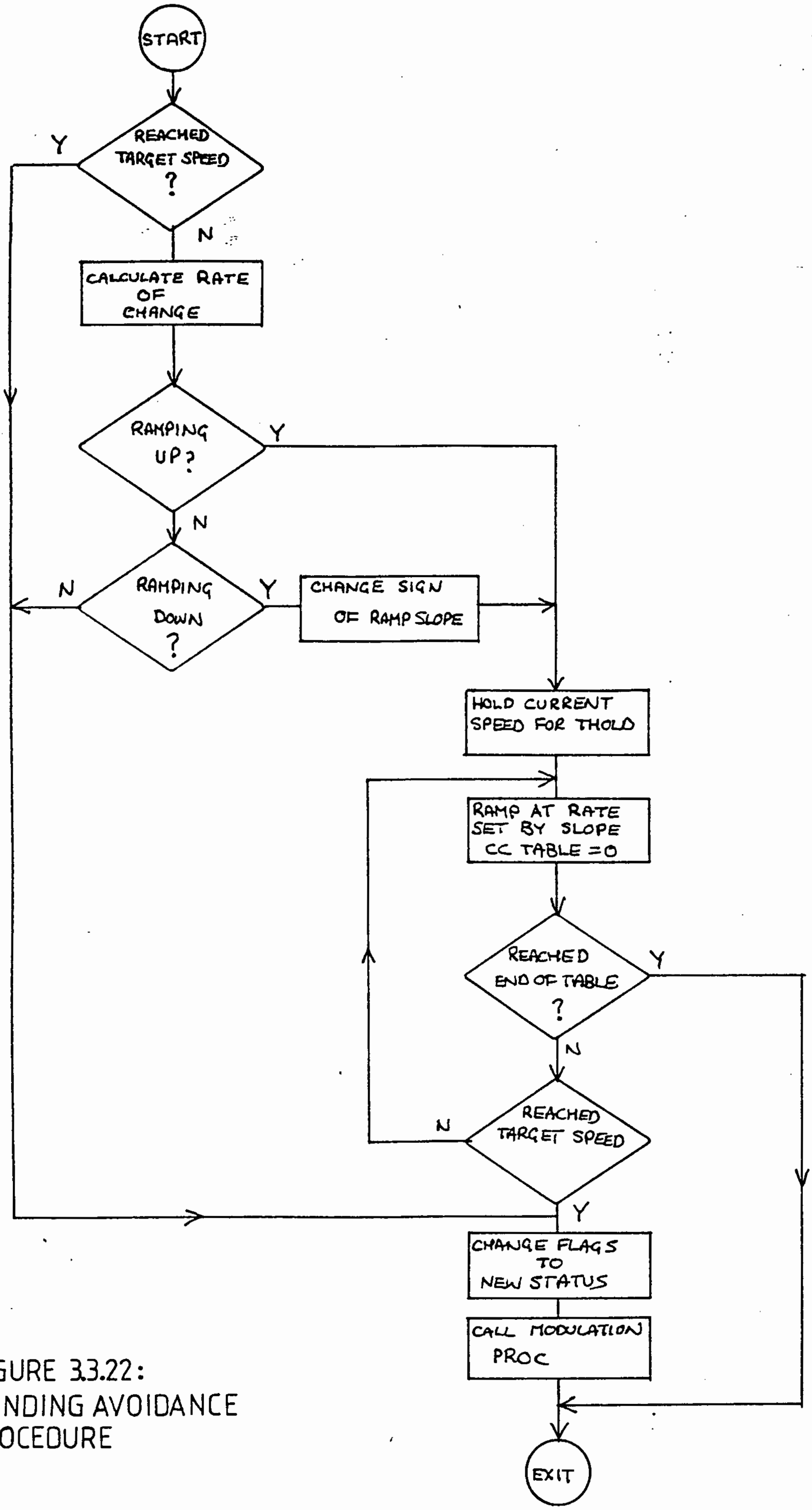


FIGURE 3.3.22:
BANDING AVOIDANCE
PROCEDURE

3.3.10 THE CHUCK TACHOMETER TASK.

This task calculates the speed of the chuck in RPM, for use during banding avoidance action. The operation of the task is identical to that of the traverse tachometer task descibed in section 3.3.3. A variable reluctance probe mounted above the chuck produces a pulse every time one of two holes drilled in the shaft passes it. The pulses are filtered and shaped, and are then used to clock an 8253 RTC. This RTC interrupts the CPU every time the count value reaches zero. The interrupt service routine reloads the counter and notes the time that the interrupt occurred.

The major difference from the traverse tachometer task lies in the length of time that pulses are counted, and hence the count value which is loaded into the PIT. The chuck speed changes continually from the time the machine is "strung up" until it is "doffed". To ensure that the required measurement accuracy is met, the measurement interval will have to be considered first. Following that an expression for the corresponding count value will be derived.

3.3.10.1 RELATIONSHIP BETWEEN CHUCK SPEED AND TIME.

The production specification called for a measurement accuracy of 0.1%. The speed measurement technique used is essentially an integrating one, so the result is the mean speed of the chuck. For a small segment of the speed/time curve, the change of speed is approximately linear, so the mean speed is half the start and end speeds. If the measured speed is to be accurate to 0.1%, the actual speed must not change by more than 0.2% during the measurement interval. This places an upper bound on the measurement interval. In order to calculate this limit, an expression relating rate of change of chuck speed with time will have to be found.

Figure 3.3.23 represents a cake of yarn, the weight of the annulus of thickness dD is :

weight = Volume of annulus * density of yarn.

$$= \frac{PI}{4} * [(D+dD)^2 - D^2] * s * p * 10^{-6}$$

Where : PI = 3.142
D = inner diameter of annulus in mm.
dD = thickness of annulus in mm.
s = length of stroke of traverse tip (ie length of annulus) in mm.
p = density of yarn in gcm^3

The weight can also be found from :

weight = length of yarn * weight per unit length.

= velocity * time * weight per unit length.

$$= \frac{v * Y * 10^{-7}}{60} * dt.$$

Where : v = wind up speed in metres per minute.
Y = decitex of yarn in g per 10^4 metres.
dt = the time interval of measurement.

These two expressions can be equated to find the change of thickness of the annulus dD in a time interval dt. Note that the units are those traditionally used in yarn technology, which accounts for the unusual constants of proportionality. Thus :

$$\frac{PI}{4} * [(dD + D)^2 - D^2] * s * p * 10^{-6} = \frac{v * Y * 10^{-7}}{60} * dt$$

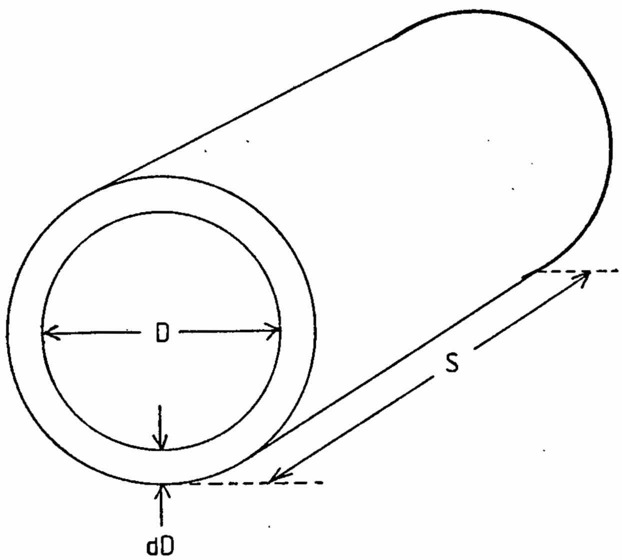


FIGURE 3.3.23: DEFINITION OF VARIABLES USED TO CALCULATE RATE OF CHANGE OF CHUCK SPEED WITH TIME.

Rearranging this in terms of dt, and assuming that the term dD^2 vanishes as dD -> 0 gives :

$$dt = \frac{300 * PI * s * p * D * dD}{Y * v}$$

The chuck speed and the diameter of the package are inversely related; the diameter starts small and increases, whereas the speed is large to begin with, and decreases during the wind up period. This is illustrated in figures 3.3.24 and 3.3.25. The speed must not change by more than 0.2% in the measurement interval. We are trying to determine the shortest time interval dt which corresponds to a 0.2% change in chuck speed. It is clear that dt is going to have its minimum value at the very start of the run. A speed decrease of 0.2% is almost equal to a diameter increase of 0.2%, so the calculation will be done in terms of diameter rather than chuck speed. The worst situation will occur when s, p and D have their minimum values, and Y and v have their maximum values.

The stroke s : has a fixed value of 120 mm
The diameter D : has a minimum value of 87 mm
The density p : has a value of 1.2 gcm^3 for most yarns.
A 0.2% change in the diameter at 87 mm makes dD = 0.174 mm.

Decitex Y, and velocity v are interrelated. Some values and their products are tabulated below :

Decitex (Y) ($g / 10^4 m$)	velocity (v) (m / min)	Product (Y * v)
20	4800	96 000
26	4600	119 600
42	6000	252 000 (1)
206	3600	741 600 (2)

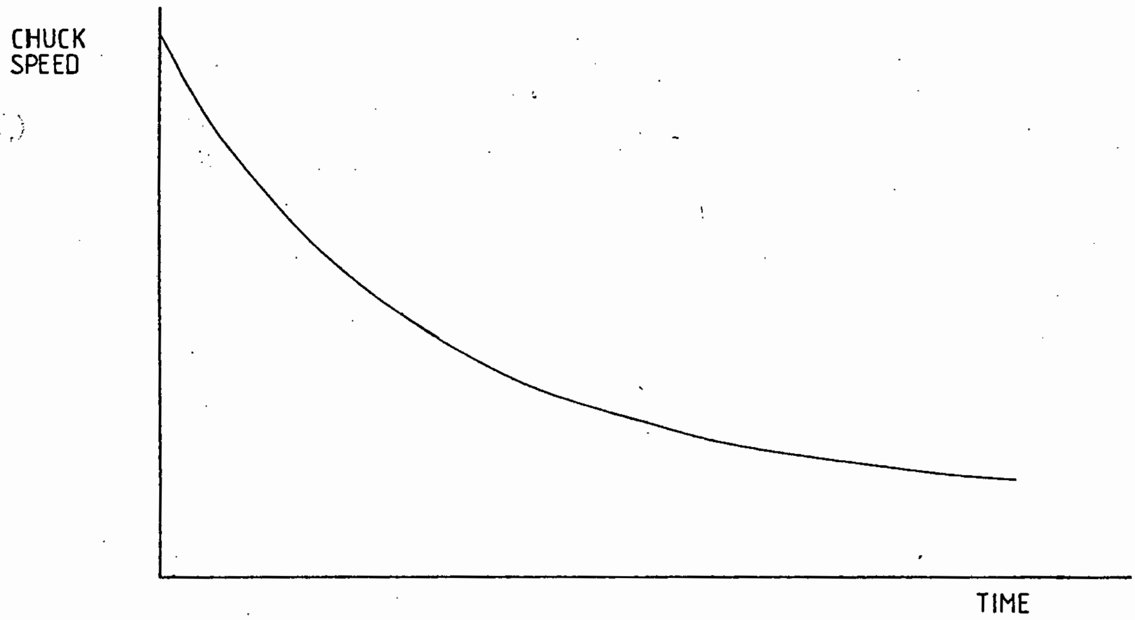


FIGURE 3.3.24: CHUCK SPEED VS TIME

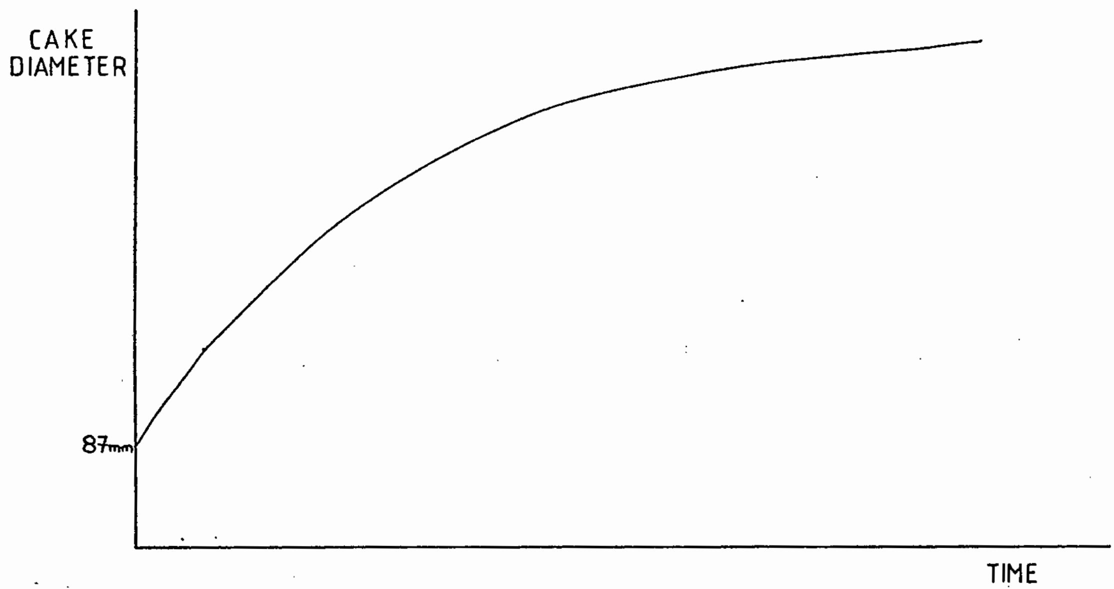


FIGURE 3.3.25: CAKE DIAMETER VS TIME

NOTES :

(1) This product cannot be produced at present because of technical production limitations, but it is hoped that it will be possible to produce it at some stage in the future.
(2) This product is the "spun" yarn, made by twisting several threadlines together.

Substituting these values for all four Yv products gives dt´s of :

Y * v	dt (secs)
96 000	21.40
119 600	17.18
252 000	8.15
741 600	2.77

The final Yv product of 741 600 is the largest that will ever be encountered, so the measurement period should be less than 2.8 seconds to guarantee that the required accuracy is met for all products. Unfortunately, no allowance was made for the last product at the design phase, and a measurement period of 4 seconds was chosen (giving a 100% safety factor). If this product is produced in future, this measurement period may have to be altered. With the chosen value of four seconds the worst speed measurement error will be slightly less than 0.15%.

There is one other source of error to be considered. The system clock places a lower limit on the time measurement. Of necessity the clock interrupt has to have the highest priority. The S-task and H-task stack switching procedures PTORTL and RETFIN take a combined total of about 150 micro-seconds to execute. Most interrupt service routines take something like 50 micro-seconds to execute. This means that the worst case latency for the clock is of the order of 200 micro-seconds. To achieve a 0.1% measurement accuracy the measurement period must be greater than 0.2 seconds, so there should be no problem with errors from this source.

3.3.10.2 DERIVATION OF COUNT VALUE.

To get a measurement period of four seconds, it is necessary to predict the number of pulses expected to arrive from the tacho in four seconds, that is the current speed of the chuck must be predicted. This can be derived from the speed of the chuck at start up and the current ribbon ratio. So :

$$\text{current chuck speed} = \text{initial chuck speed} * \frac{\text{current ribbon ratio}}{\text{initial ribbon ratio}}$$

A) initial chuck speed :

$$= \frac{1}{6} * \text{winder output frequency} * \frac{\text{circumference of winder roll}}{\text{circumference of tube}}$$

$$= \frac{1}{6} * \frac{\text{CLOCKFREQ}}{\text{WPITVAL(WPOINT)}} * \frac{3}{2}$$

Note that the initial ratio of 3:2 for the circumferences is an approximation.

B) current ribbon ratio :

$$= \frac{6 * \text{WTACHO}}{\text{TTACHO}}$$

C) initial ribbon ratio :

= approximately 7

Thus approximate current chuck speed :

$$= \frac{1}{6} * \frac{CLOCKFREQ}{WPITVAL(WPOINT)} * \frac{3}{2} * \frac{6 * WTACHO}{TTACHO} * \frac{1}{7} \text{ Hz}$$

Required count value :

= chuck speed * time interval * no of pulses / rev

= chuck speed * 4 * 2

$$= 1.714 * \frac{CLOCKFREQ}{WPITVAL(WPOINT)} * \frac{WTACHO}{TTACHO}$$

Where :

CLOCKFREQ = Winder pulse RTC clock frequency (2.456 Mhz)
WPITVAL(WPOINT) = Current count being loaded into RTC.
WTACHO = Current chuck speed (in RPM)
TTACHO = Current traverse speed (in RPM).

There will be a problem when either WTACHO or TTACHO are zero (either when the machine is first started or after a doff). To overcome this problem, a dummy count value of 100 is used until both speeds exceed 500 RPM. The measured speed during this time will be extremely inaccurate. However, accurate speed measurement is only required when both the traverse and winder motors are up to their normal operational speed.

3.3.10.3 DERIVING THE CHUCK SPEED.

This is done in exactly the same manner as for the traverse tacho speed, by dividing the number of pulses counted by the time taken to count them. Once again the possibility of the value of NOW changing while the fractional part of the count is read "on the fly" is catered for, by discarding speed measurements whose fractional values are close to the count reload time. Figure 3.3.26 shows the flow diagram for this task.

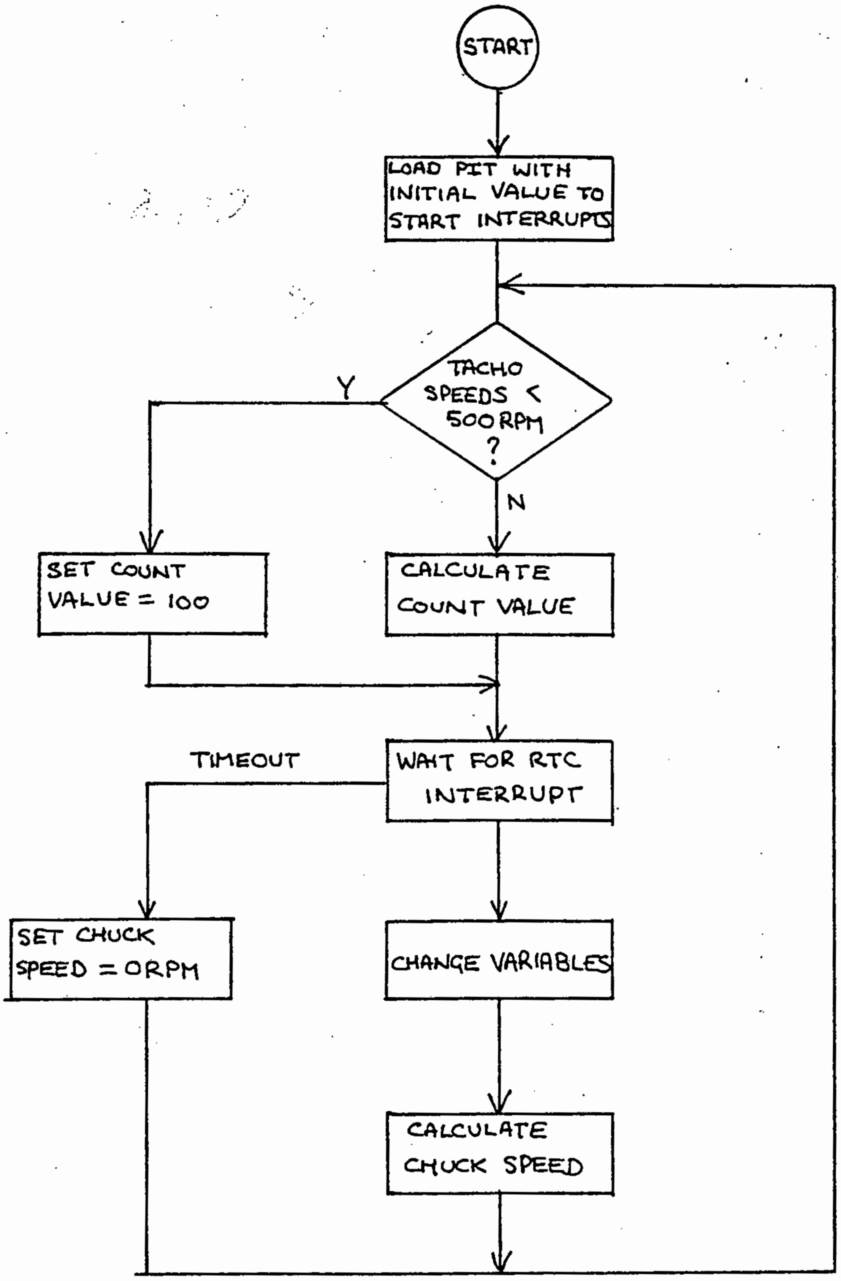


FIGURE 3.3.26: CHUCK SPEED MEASUREMENT TASK

3.3.11 THE DATA BASE UPDATE TASK.

This task is very straight forward - it simply transfers data from the awaiting update data bricks into the system data bricks, after performing any necessary conversions. The update is triggered by a system event which is set when the sequence task detects that the chuck speed has fallen below 1500 RPM, indicating that a doff is in progress. When any data is altered (by the OCP or host link tasks) a flag is set to show which block of data has been altered. The update task scans these flags and updates the appropriate data. The OCP and host link communications tasks should be consulted for further information on updating operating data.

3.3.12 THE HOST COMMUNICATIONS TASK.

The function of this task is to wait for commands from the host computer (a DEC PDP-11 Process management computer), and to execute the commands when they are received. The communication requirements of the system are for the transfer of operational information between the host and the inverter control units. This requirement arose for two reasons :

1) An operating instruction can have up to 70 parameters, and there are 24 controllers. Entering all these parameters is extremely tedious and error prone. In addition the host can hold a "library" of operating instructions for different processes. In time operating instructions for most processes required will be built up, and will not need retyping each time.

2) The host computer can monitor the status of each position and provide both immediate alarms, and long term logging and statistical information on the system. The host computer monitors all the stages of the process, and can thus provide detailed breakdowns of how the process is functioning.

The Nylon Spinning industry often requires precise control of a single process, repeated a large number of times. The advent of low cost industrial computers has raised the possibility of sophisticated control for large numbers of machines. There is a clear need for a host computer communicating with large numbers of dedicated control computers. Multi-dropping reduces the amount of wire and serial channels required, and simplifies installation. As a result, a standard communications protocol that would satisfy the needs of all applications that could be foreseen was developed. The results are reproduced in Appendix F.

The communication requirements for the inverter control unit are for the host to be able to send operating instructions to the controllers, and for the host to be able to interrogate the controllers about their current status. There will never be any need for control units to communicate with each other, and all the applications foreseen involve communication between a supervisory computer and dedicated controllers. Thus a master / slave protocol was chosen since it is considerably easier to implement than full station to station asynchronous communication. With this protocol the host always initiates data transfers, so there are no problems with busy or jammed lines.

Each controller on the line has a unique station address set up by a four bit DIP switch on the I/O board. Any communication from the host is received by all the controllers, which scan the fourth byte of the received data packet for the destination address, and compare it with the station address.

3.3.12.1 THE COMMUNICATIONS SOFTWARE.

This task consists of two modules: One for the hardware driver routines; and one for the packing and unpacking of data into packets. Since simultaneous reception and transmission of data will never occur, the same buffer is used for transmission and reception of data. The buffer is 138 bytes long, which is an optimal length for the amount of data to be sent. The communications task, called LINKDA.RTL, waits for an event which is set by the received data interrupt routine. This event is set once an entire packet destined for that station has been successfully received. LINKDA.RTL then decodes the packet type by looking at the fifth byte and decides whether the host is requesting or sending data.

If data is being requested by the host, it is fetched from the data base, converted into the format expected by the host, and packed into the data buffer along with appropriate header, end of message and checksum data. Once the packet has been assembled, the interrupt service routine is invoked through software by using the INT machine code instruction. The interrupt routine then automatically sends the buffer to the host, until the end of message byte has been transmitted. The interrupt service routine then stops sending data, and sets a system event which tells the communications task that the buffer has been sent. If this event is not set within a time limit, the task prints an error message and clears the USART transmit and receive enable flags. This ensures that the USART does not jam the multidrop link data lines. The host polls all stations once every five minutes, and prints out an error log message if no answer is received from a station. To give added protection against jamming, the host sends a "Z" type message if no answers are received from any stations after information has been solicited. This message type clears the USART enable flags.

If the host is sending data to the controller, the communications task performs the reverse process: it checks the header and checksum, unpacks the data from the packet in the data buffer, converts from host to controller format, and places the received data in the data base. Because of the large amount of data contained in an operating instruction, the host sends a total of four packets to the controller, each packet containing part of the data. The controller always acknowledges a host communication, indicating whether it was successful or not. DEC has a slightly different representation of floating point reals from INTEL, so it was necessary to modify REAL variables passed between the systems. This conversion was performed by the controller. Figure 3.3.27 shows a flow diagram of the communications control task.

The Serial link USART is configured so that the reception of a data byte from the host interrupts the CPU, which unloads the USART data buffer. The Received data interrupt routine keeps track of the number of bytes received from the host and checks to see if it is part of the header, the data block or the end of message. If the data is part of the header and the packet is addressed to that station the counters and pointers for the packet are initialised. If the packet is not destined for the station, the counters are set up in such a way that when the end of message byte is received the message length byte of the header does not tally with the buffer pointer, so the value is discarded. If the received byte is part of the data part of the packet, it is put into the appropriate place in the buffer. If the byte is an "end of message" character and the buffer pointer tallies with the length

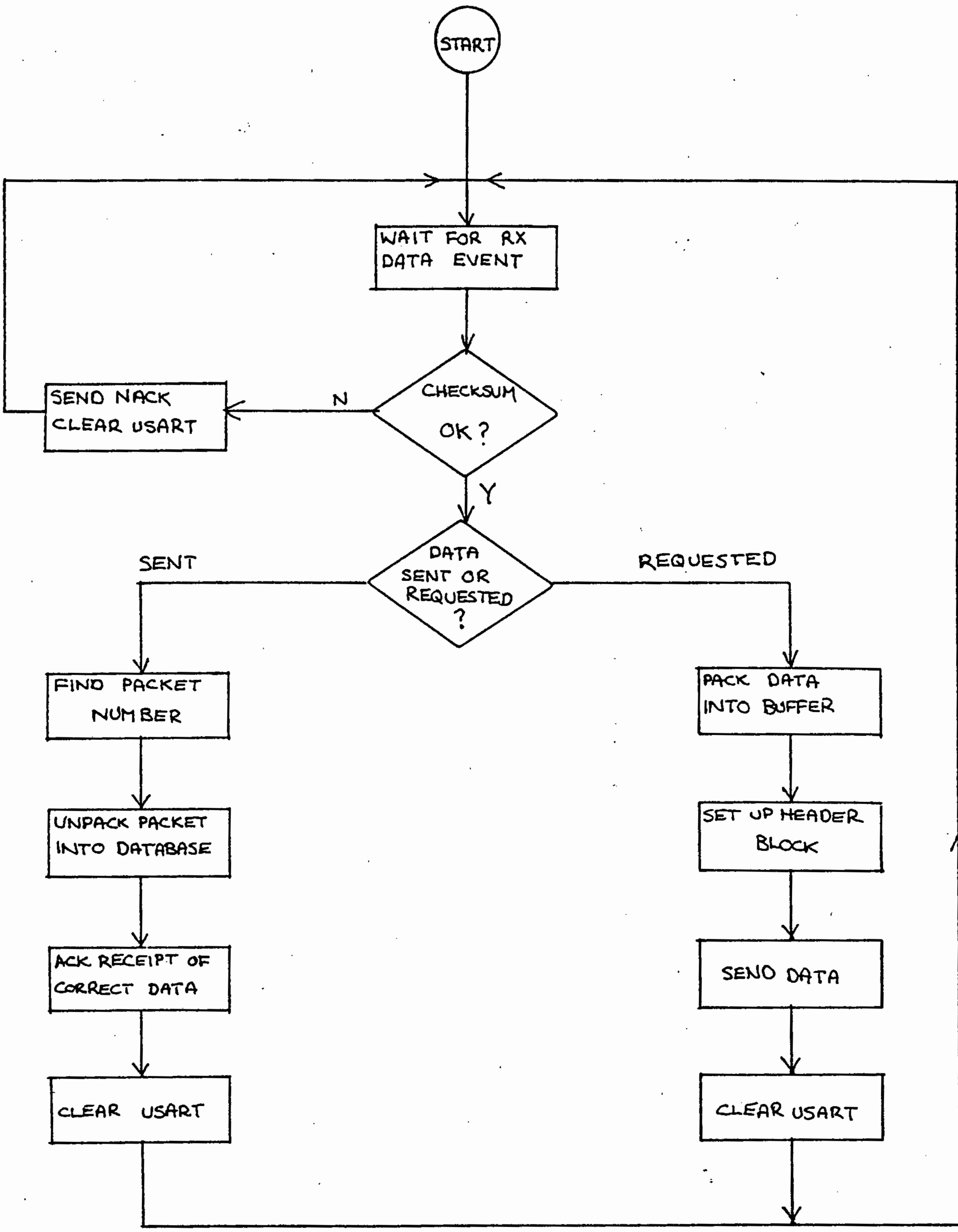


FIGURE 3.3.27: COMMUNICATION CONTROL TASK

of the message, then a system event is set which triggers the communications task to service the request from the host. Figure 3.3.28 is a flowchart of the Received data interrupt handling routine.

The Transmit channel of the USART is also configured so that the CPU is interrupted once the USART transmit buffer is empty. Because the transmission rate of the data (9600 baud) is much slower than the speed of operation of the computer, the CPU can load the transmit buffer with a character, and then continue executing S-tasks until the USART transmit buffer is empty again, which causes an interrupt and loads the next character and so on. Figure 3.3.29 is a flowchart of the Transmit data interrupt handling routine.

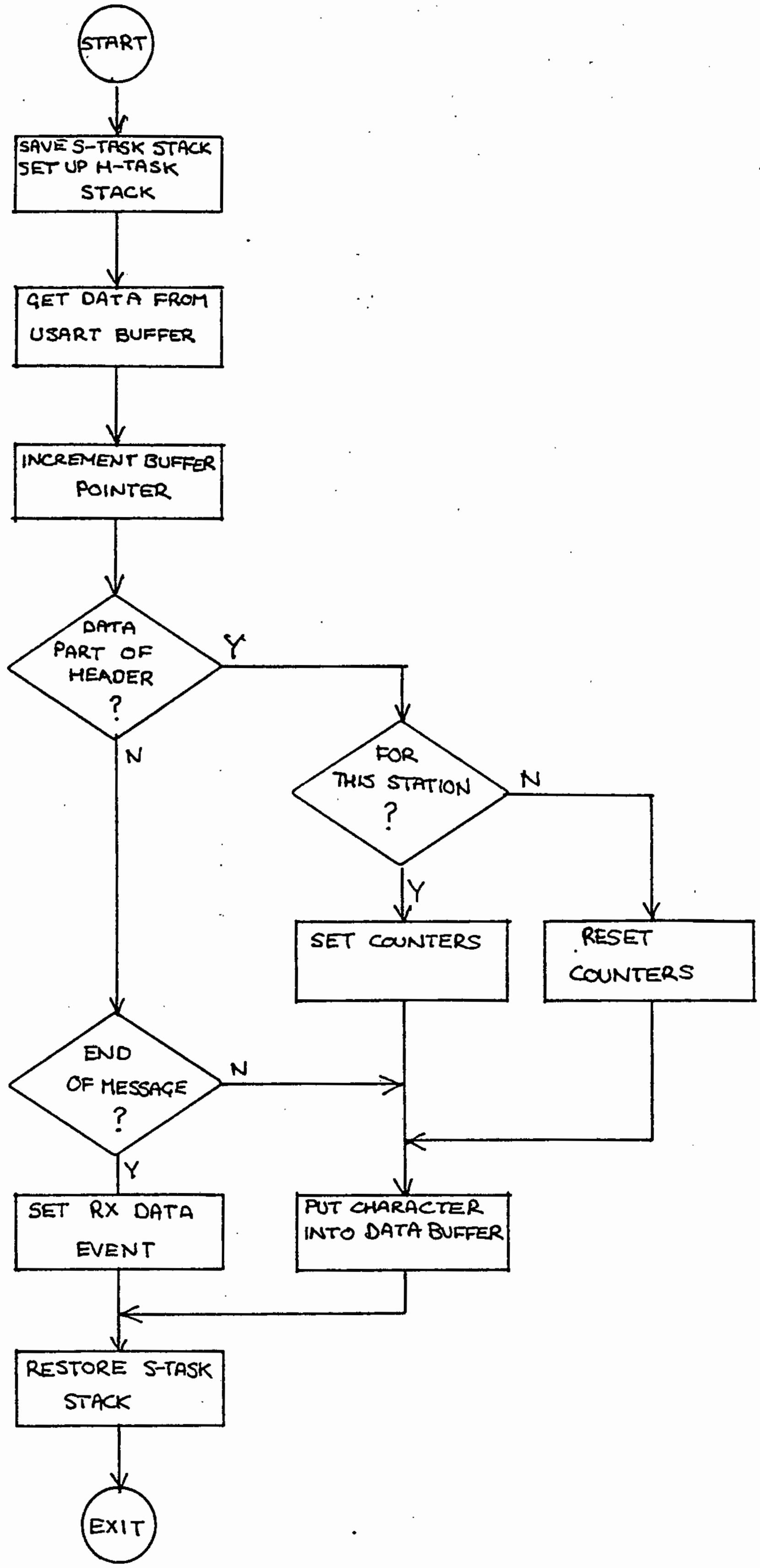


FIGURE 3.3.28: RECEIVED DATA INTERRUPT SERVICE ROUTINE

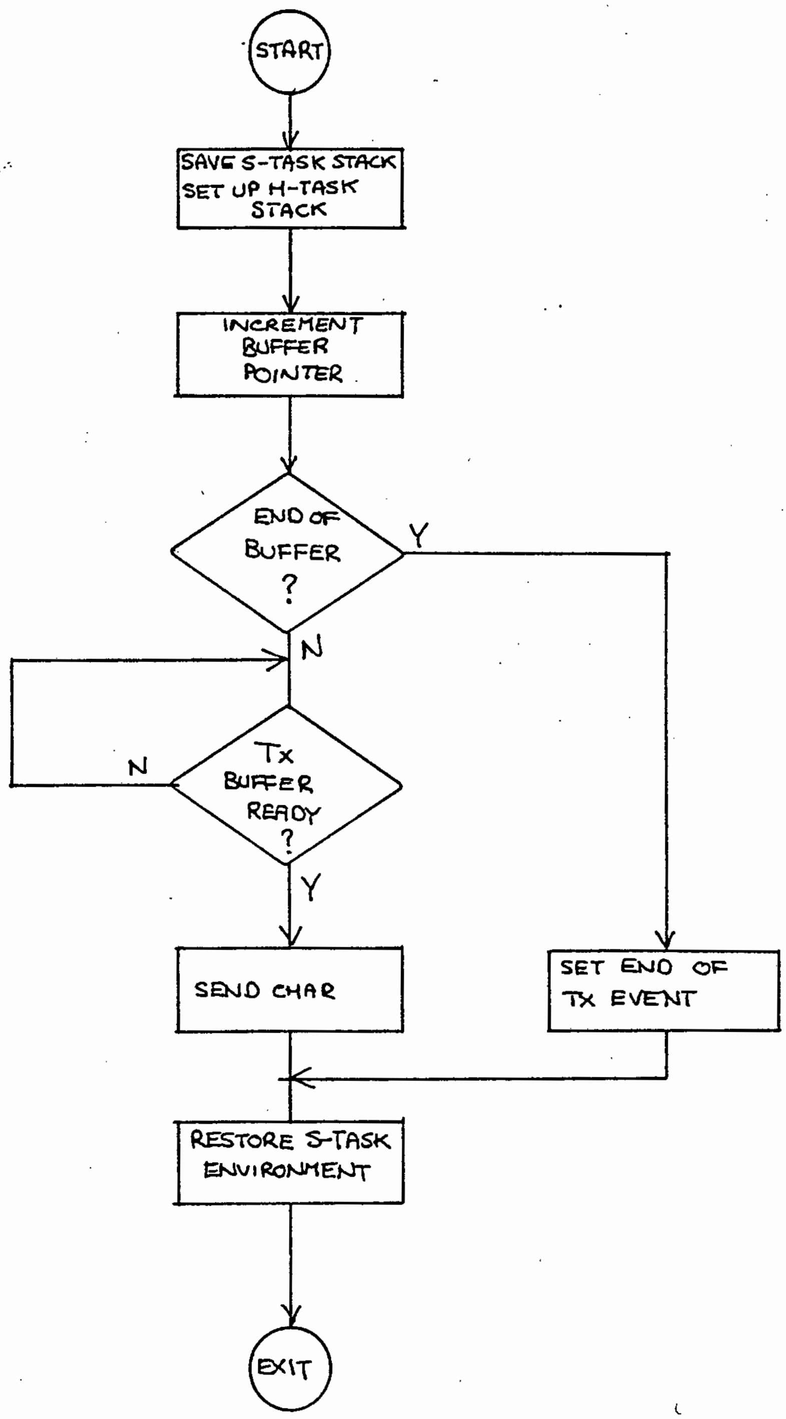


FIGURE 3.3.29: TRANSMIT DATA INTERRUPT SERVICE ROUTINE

3.4 MAKING THE SYSTEM.

The source modules for the applications software for the inverter control unit are held on RL02 10MB disc packs in the computer hardware development laboratory of SANS. The SMT source and object files are held in account DL0:[230,1]. In the RSX operating system, accounts are called User Identification Codes. They consist of the device where the file is located, and group and member numbers as follows : DLaa:[ggg,mmm]. aa is the device number, ggg is the (octal) group number, and mmm is the (octal) member number. A standard for account layouts has been established. Each project is held in UIC's with the same group number. SMT modules that have been modified for a particular project are held in member number 001 of the UIC along with the command file to make the SMT portion of the system. Project application development is done in the remaining member numbers of that UIC group number. Thus the SMT modules that had to be specially modified for this project are held in account DL1:[341,1], whilst the applications modules are found in DL1:[341,2]. This last account can be signed into with the MCR command

MCR>HEL ICUNIT/TEK

If modifications have to be made to any of the source files it will be necessary to remake the system. This can be done by signing in to account DL1:[341,2] as explained above and running the command file TOTSYS.CMD by typing @TOTSYS. This command file contains all the necessary utility calls, switches etc to rebuild the entire system. The command file will give you several prompts as follows :

Module name, DATAPREL, ALL or SMT?

If you have only altered one module, type the name of that module only (do not type the file extension .RTL). If you want to remake the entire system, type "ALL". If you want to remake the SMT portion of the system type "SMT".You will then be asked whether you want Source or Code listings, to which you must respond with a "Y"es or "N"o.

If you chose to remake SMT at the first prompt, you will be prompted :

Do you want to remake SMT?

If you answer "Y"es you will be prompted for the module name and listings as before. Once compilation and assembly of all RTL/2 source files has been completed, the SMT system will be linked, followed by a link of the SMT system with the applications modules.

This is the only interaction necessary with the command file. Appendix G contains a listing of TOTSYS.CMD, and this should be consulted for detailed information on making the system.

Notes :

1) If the database (DATA.RTL) is altered, then the data prelude file (DATAPREL.RTL) should be altered accordingly, and then all the modules and both parts of the system must be remade.

2) TOTSYS calls two linker files, one for the SMT portion of the system, and one for the applications tasks. If the linker operation has to be altered, then these two files (ROMSYS.CMD for the SMT portion in DL1:[341,1], and ICUSYS.CMD in DL1:[341,2] for the applications tasks) must be edited. See Appendix G for listings of these files. The linker map file in Appendix G is the output of ICUSYS.CMD.

When the system is ready to be blown into ROM, the SMT link command file must be edited for the correct memory addresses, both parts of the system must be re-linked, and then a ROM image must be produced using the utility FDM. The user will be prompted by FDM for the start and end addresses of the code, and for a selection mask for splitting the code. Since 8088 systems are byte orientated and not word orientated like the 8086, there is no need to split the code and the required mask is 11111111. The output from FDM is put into a file specified by the applications programmer, usually of extension type .BIN. The .BIN files can then be downloaded into an EPROM programmer, and blown into EPROM.

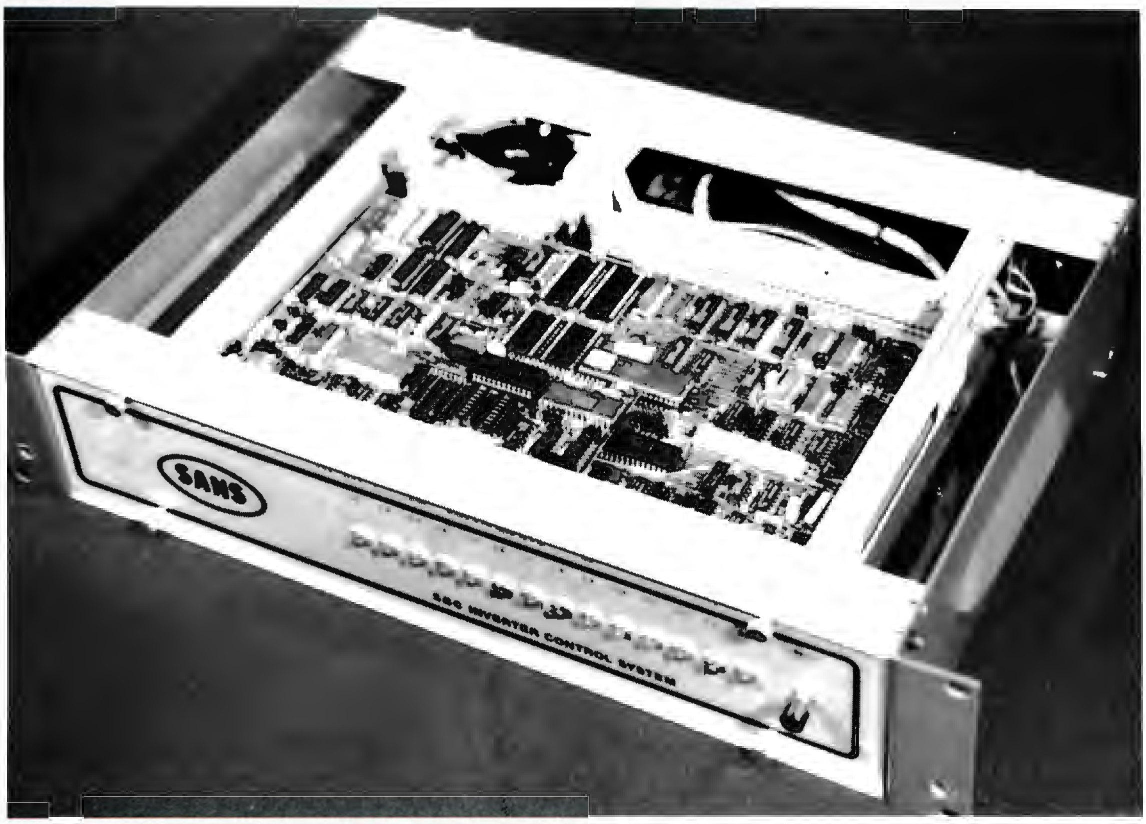


FIGURE 4.1.1

CHAPTER 4 : USING THE SYSTEM.

4.1 ASSEMBLING AND COMMISSIONING AN ICU.

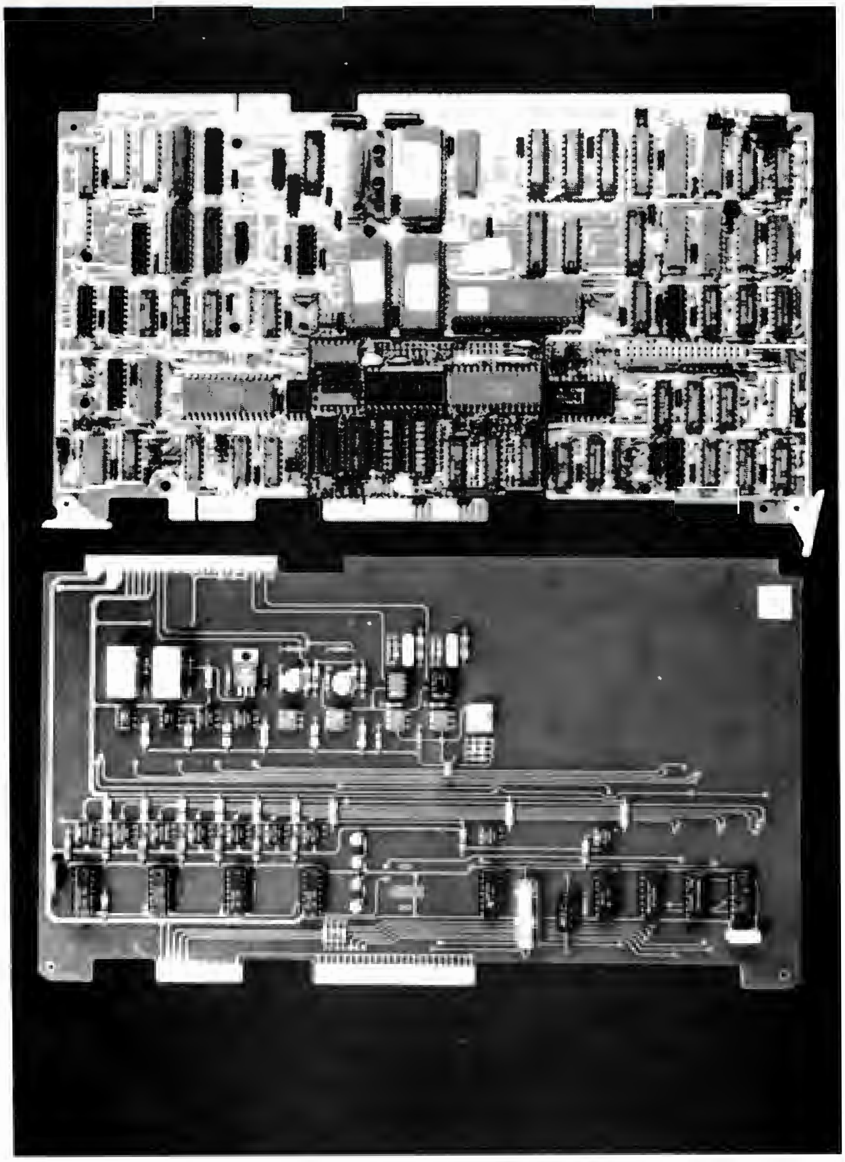
An Inverter control unit has five components :

1) A Single board computer.
2) An I/O board.
3) A drawer to house the computer and I/O board.
4) A front panel.
5) A back panel.

Figure 2.3.2 in chapter 2.3 should be consulted for a diagram showing the contents of the drawer and their orientation to each other.

4.1.1 CONFIGURING THE COMPUTER.

If a new board is received straight from the manufacturers, it will need a number of modifications in order to work as an Inverter Control unit. These changes are set out in Appendix H.9 which lists the SBC jumper allocations. Once these alterations have been made it will be necessary to install EPROM´s and extra RAM. The RAM consists of two 2168 four bit by 4k chips installed in sockets U52 and U68.


SBC

I/O

FIGURE 4.1.2

Three EPROM chips are required, labeled as follows:

ICUFA/B
VERS 1.C
DD/MM/YY

ICU identifies the EPROM as belonging to the Inverter Control Unit. FA/B is the start address of 8k blocks of the software in the EPROM´s, VERS 1.C is the version number of the software, and DD/MM/YY is the date the EPROM was blown.

At the time of writing the software had reached version 1.7 of October 1984, so before installing EPROM´s check that you have the latest version . The location of the EPROM´s is as follows :

EPROM LABEL	MEMORY ADDRESS	SOCKET NUMBER
ICUF4/6	F4000 - F7FFF	U34
ICUF8/A	F8000 - FBFFF	U64
ICUFC/E	FC000 - FFFFF	U33

For details of how to program the EPROM´s see chapter 3.4. Once all the jumpers and memory chips have been installed, the SBC board should be inserted into the top slot in the drawer.

4.1.2 CONFIGURING AN I/O BOARD.

The only action necessary before installing an I/O board is to set the station address for the host link communications on the four pole DIP switch. If the board is inserted in its proper orientation in the bottom slot of the drawer, the LSB is the rightmost bit if looked at from the front of the drawer. The address should be set according to the drawer´s position in the cabinet as follows :

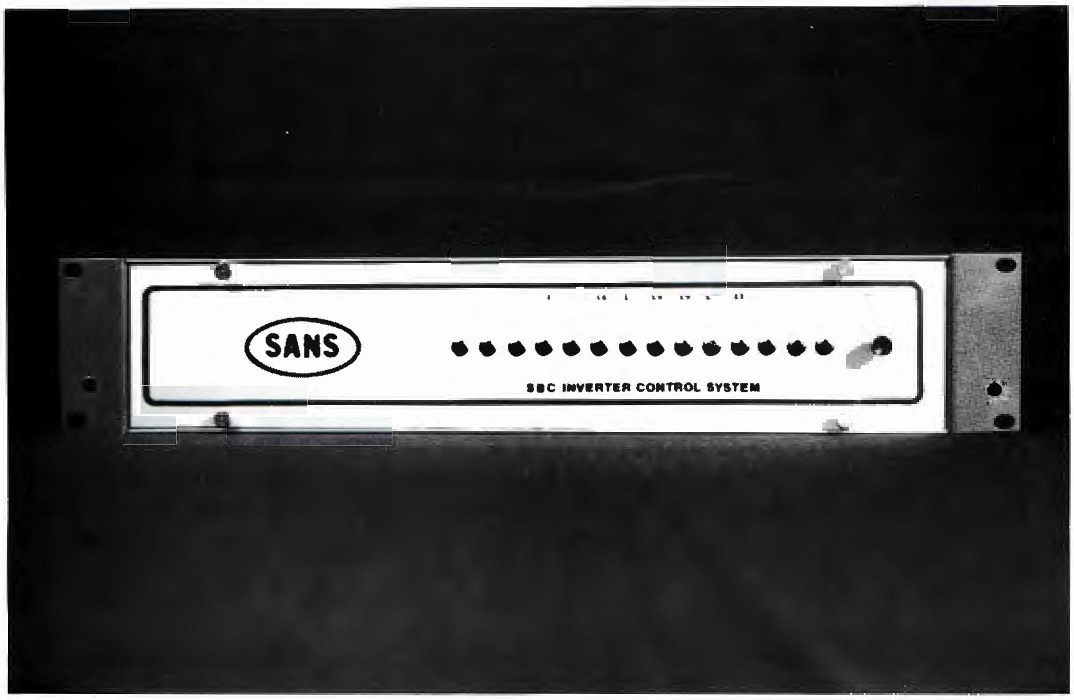


FIGURE 4.1.3

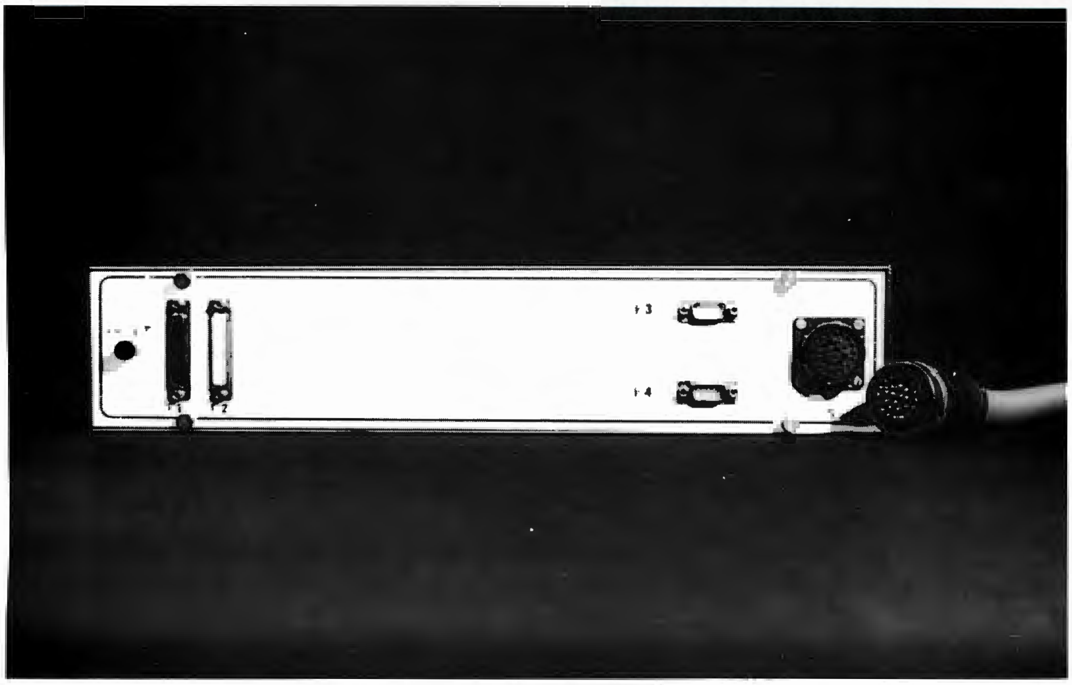


FIGURE 4.1.4

LINK NO	POSITION NO	STATION ADDRESS	LINK NO	POSITION NO	STATION ADDRESS
1	1A	1	1	1C	3
1	1B	2	1	1D	4
1	2A	5	1	2C	7
1	2B	6	1	2D	8
2	3A	1	3	3C	1
2	3B	2	3	3D	2
2	4A	3	3	4C	3
2	4B	4	3	4D	4
2	5A	5	3	5C	5
2	5B	6	3	5D	6
2	6A	7	3	6C	7
2	6B	8	3	6D	8

This table shows the location of the drawers in the cabinet as seen from the front of the cabinet. The link numbers are the serial link numbers from the host computer, and the position numbers are the wind up head position numbers.

Once the station address has been set, the I/O board should be inserted into the lowest slot in the drawer. Once the I/O board and the SBC have been inserted, the signal link cable should be inserted to connect them together. This consists of a 5 centimeter length of 50 way ribbon cable with 50 way edge connectors at either end. The edge connectors are plugged into the J1 connectors of the MULTIBUS form factor. This is the connector in the middle of the boards as looked at from the front of the drawers, and can be clearly seen in figure 4.1.5.

4.1.3 THE FRONT AND BACK PANELS.

The front panel connects to the J2 connector on the I/O board. This is the 26 pin connector on the left hand side of the drawer as seen from the front.

The back panel carries the I/O connector, the host and local VDU

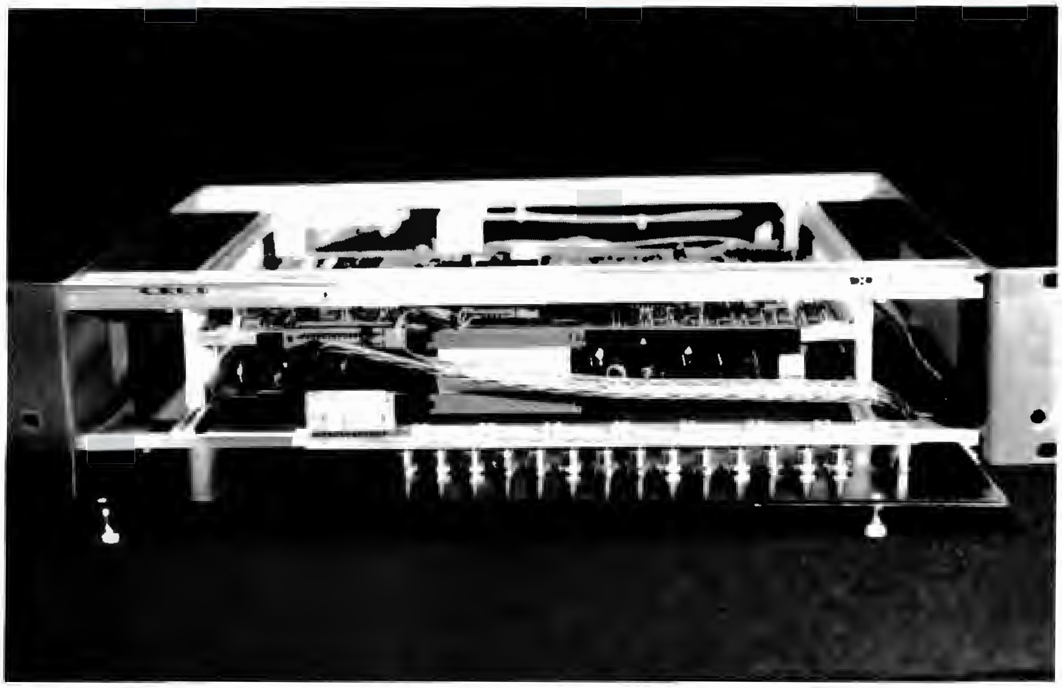


FIGURE 4.1.5

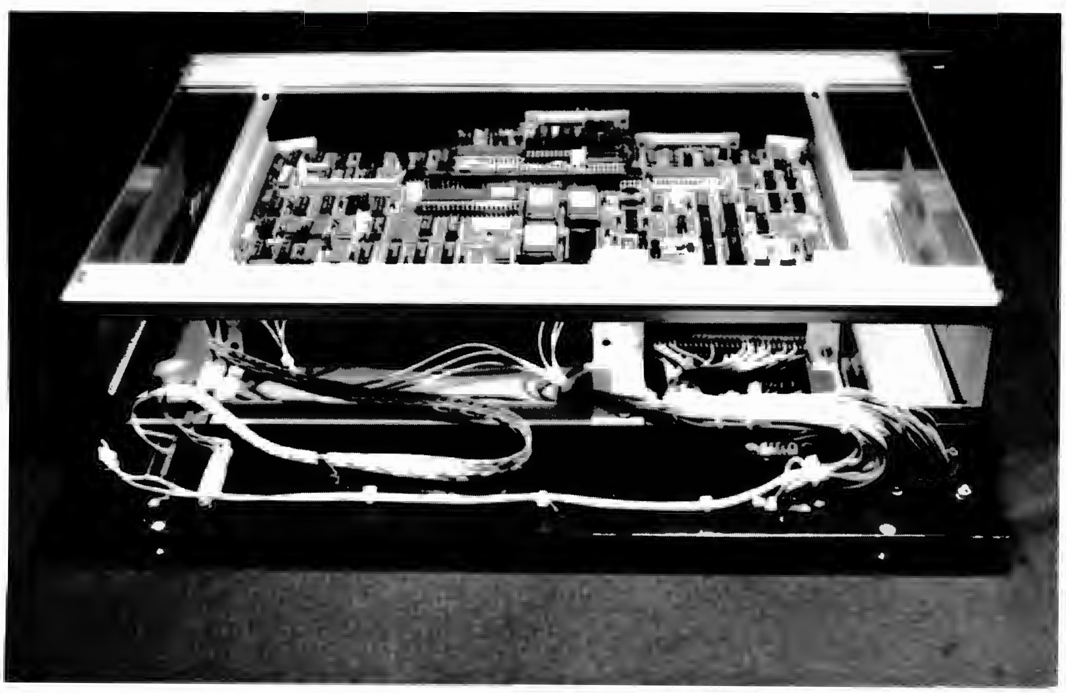


FIGURE 4.1.6

serial link sockets, the power connector socket, and the auxiliary I/O connector. For details of these connections, consult the wiring schedule in Appendix H.7. The serial links must be plugged into the SBC and piggy back boards at the front of the drawer, so the plugs and wiring must be fed down the side of the drawer. The multidrop RS-422 link connects to the piggy back 26 way edge connector, while the local VDU RS-232 link connects to the SBC J2 connector on the left hand side of the drawer as seen from the front.

Once all the connections have been made inside the drawer, it can be tested by inserting it into the ICU cabinet, or by using the test jig produced by the computer hardware department. This jig emulates a winder head, and allows the ICU to be put through all its paces. When the drawer is powered up, the amber CPU running LED (L10) should come on, and the watchdog LED (L11) should flash continuously with a 2 second period. If a local VDU is connected, it should beep and give an "OCP CLEARED" message. After this it should be possible to set up process parameters through the OCP function on the local VDU, and test the operation of the computer by running it. Software has been developed on the computer hardware development bureau to emulate the host computer, and this can be used to test the operation of the host communication facility.

4.2 THE LOCAL VDU OCP TASK.

The operator communicates with the computer either through the host link or the local VDU. Both links are menu driven, and the operator is prompted for responses. Typing "X" to any prompt gives an explanation of the options available to an operator. Out of range or incorrect responses are rejected, and the operator is reprompted.

Notes :

1) All responses or entries to the system must be terminated with a carriage return, except when the main menu is called up.

2) The system will only accept upper case characters, so the "CAPS LOCK" key on the terminal must be depressed at all times. The system will respond with a "NO!" or "INVALID CHARACTER" if lower case characters are entered.

3) The main OCP menu is called up by pressing the space bar. (Carriage return must not be pressed in this case)

4) Typing "Z" in response to any prompt will cause the system to exit back to the point where pressing the space bar will call up the main menu.

5) Typing "X" to any prompt from the system will result in a full explanation of the options available to the operator at that point.

6) If the system appears to be "hung up" and does not respond when the space bar is pressed, try pressing "Z" followed by carriage return, which will clear the system if it was busy doing something else. If the system still does not respond try RESETting the computer by pressing the RESET button on the back panel of the drawer. If there is still no response there is probably a fault in the hardware or software, and a technician should be called.

7) If no response is entered to a system prompt, then the system will time out after about two minutes and return to the default condition waiting for the space bar to be pressed.

8) If operational values are to be changed or the "Control-A" monitor is invoked, a security check will have to be passed. When the operator has finished making changes, he will be prompted for an "OPERATOR NUMBER" followed by a "PASSWORD". If both of these are entered correctly, he will be asked if he wants to "MAKE CHANGES". If he answers "Y"es, the changes will be made and the old parameters lost. If he replies "N"o or enters the password or operator number incorrectly, the system will return to the default state and the changes will be abandoned. See the T30 shift supervisor to get the passwords.

4.2.1 THE MENU OPTIONS.

Figures 4.2.1 through to 4.2.8 show printouts of a typical interaction with the ICU through a local VDU. Each of the options will be briefly explained.

Figure 4.2.1

Three basic menu options are provided. The time and date can be set or viewed, The current operating status can be viewed, and the current operating parameters can be viewed or modified. The main menu is called up by pressing the space bar when the system is in the default mode. Figure 4.2.2 shows the time and date being altered.

Figure 4.2.3

This shows menu option 2, which allows the current status of the machine to be viewed. After displaying the status, the user will be asked whether he wants the "DISPLAY AGAIN". Answering "Y"es will re-display the latest values of the status information.

Figure 4.2.4

This shows the result of typing "X" when prompted in menu option 2, and tells the operator what options are available in option 2.

Figure 4.2.5

This shows menu option 3, which allows the current operating parameters to be viewed and modified. The operator is prompted after the display. Four options are available, as can be seen by typing "X" (see Figure 4.2.6)

Figure 4.2.7

This shows how the parameters for banding avoidance point number 1 may be altered, by responding with a "B" to the prompt in menu option 3.

Figure 4.2.8

This shows how the modulation parameters can be altered by responding with an "M" to the prompt in menu option 3.

```
             *** SANS  INVERTER  CONTROL  SYSTEM  ***
OCP OPTIONS:-
1-DATE AND TIME
2-DISPLAY FIXED PARAMETERS AND SPEEDS
3-DISPLAY OR CHANGE TRAVERSE PARAMETERS
OPTION=[
```

FIGURE 4.2.1

```
                           *** SANS INVERTER CONTROL SYSTEM ***
OCP OPTIONS:-
1-DATE AND TIME
2-DISPLAY FIXED PARAMETERS AND SPEEDS
3-DISPLAY OR CHANGE TRAVERSE PARAMETERS
OPTION=[1]

00:07:33 01/01/81  CHANGE?[Y] DATE[26/6/85] TIME[15:00] OK


OCP CLEARED
```

FIGURE 4.2.2

```
                     *** SANS INVERTER CONTROL SYSTEM ***
OCP OPTIONS:-
1-DATE AND TIME
2-DISPLAY FIXED PARAMETERS AND SPEEDS
3-DISPLAY OR CHANGE TRAVERSE PARAMETERS
OPTION=[2]
         -DISPLAY FIXED PARAMETERS AND SPEEDS     15:00:33 26/06/85
 CURRENT CONTROL STATUS       = WINDER STOPPED

 WIND MAX ACCEL/DECEL  RATE   =       1.00 /   1.00 HZ/SEC
 TRAV MAX ACCEL/DECEL RATE    =       3.00 /   3.00 HZ/SEC
 WINDER MAXIMUM FREQUENCY     =     211.98 HZ
 TRAVERSE MAXIMUM FREQUENCY   =     333.33 HZ
 MINIMUM FREQUENCY CLAMP      =      13.33 HZ

 START-UP DELAY PERIOD        =          2 SECS
 WINDER SPEED SETPOINT        =      50.00 HZ
 OUTPUT FREQUENCY - WINDER    =      13.33 HZ
 CAKE  SPEED  (1MIN AVERAGE)  =       0.00 RPM

 CURRENT BANDING POINT(1-  1)=           1
 OUTPUT FREQUENCY - TRAVERSE =       13.33 HZ
 TRAVERSE SPEED  (1 MIN AVE) =        0.00 RPM
 CURRENT RIBBON RATIO         =  TRAVERSE STOPPED
 CAKE DIAMETER                =   CHUCK STOPPED

DISPLAY AGAIN?  (Y/N) ? [
```

FIGURE 4.2.3

```
OCP OPTIONS:-
1-DATE AND TIME
2-DISPLAY FIXED PARAMETERS AND SPEEDS
3-DISPLAY OR CHANGE TRAVERSE PARAMETERS
OPTION=[2]
        -DISPLAY FIXED PARAMETERS AND SPEEDS      15:01:29 26/06/85
 CURRENT CONTROL STATUS       = WINDER STOPPED

 WIND MAX ACCEL/DECEL  RATE   =         1.00 /   1.00 HZ/SEC
 TRAV MAX ACCEL/DECEL RATE    =         3.00 /   3.00 HZ/SEC
 WINDER MAXIMUM FREQUENCY     =       211.98 HZ
 TRAVERSE MAXIMUM FREQUENCY   =       333.33 HZ
 MINIMUM FREQUENCY CLAMP      =        13.33 HZ

 START-UP DELAY PERIOD        =            2 SECS
 WINDER SPEED SETPOINT        =        50.00 HZ
 OUTPUT FREQUENCY - WINDER    =        13.33 HZ
 CAKE  SPEED  (1MIN AVERAGE)  =         0.00 RPM

 CURRENT BANDING POINT(1-  1)=             1
 OUTPUT FREQUENCY - TRAVERSE =         13.33 HZ
 TRAVERSE SPEED  (1 MIN AVE) =          0.00 RPM
 CURRENT RIBBON RATIO         =  TRAVERSE STOPPED
 CAKE DIAMETER                =   CHUCK STOPPED


DISPLAY AGAIN?  (Y/N) ? [X]


    'Y' - CAUSES AN UPDATE OF THE DISPLAY.
          (WITH REFRESHED DATA AS AT DISPLAYED TIME)

    'N' - RETURNS TO 'OCP CLEARED'

    'Z' - ESCAPES TO 'OCP CLEARED'

    'X' - EXPLANATION
DISPLAY AGAIN?  (Y/N) ? [
```

FIGURE 4.2.4

```
OCP OPTIONS:-
1-DATE AND TIME
2-DISPLAY FIXED PARAMETERS AND SPEEDS
3-DISPLAY OR CHANGE TRAVERSE PARAMETERS
OPTION=[3]
    -DISPLAY OR CHANGE TRAVERSE PARAMETERS    15:02:35 26/06/85   ACTIVE PARM'S

     RIBBON POINTS          SPEEDS               TRAVERSE  MODULATION
                                             AMPLITUDE         P-JUMP        PERIOD
        R1       R2        F1       F2       F1      F2      F1      F2       SECS
  1   0.000    0.000      30.0     30.0     0.00    0.00    0.00    0.00      2.0
  2   0.000    0.000      30.0     30.0
  3   0.000    0.000      30.0     30.0
  4   0.000    0.000      30.0     30.0
  5   0.000    0.000      30.0     30.0
  6   0.000    0.000      30.0     30.0         TRAVERSE  ACCELERATION
  7   0.000    0.000      30.0     30.0          F1 TO F2     F2 TO F1
  8   0.000    0.000      30.0     30.0             0.83         0.83
  9   0.000    0.000      30.0     30.0
 10   0.000    0.000      30.0     30.0           WINDER  SPEED
 11   0.000    0.000      30.0     30.0                50.00
 12   0.000    0.000      30.0     30.0
 13   0.000    0.000      30.0     30.0
 14   0.000    0.000      30.0     30.0        MAXBAND =1
 15   0.000    0.000      30.0     30.0

REDISPLAY/BANDING CHANGE/MOD & SPEED CHANGE/EXIT (R/B/M/E) ? [
```

FIGURE 4.2.5

```
OCP OPTIONS:-
1-DATE AND TIME
2-DISPLAY FIXED PARAMETERS AND SPEEDS
3-DISPLAY OR CHANGE TRAVERSE PARAMETERS
OPTION=[3]
    -DISPLAY OR CHANGE TRAVERSE PARAMETERS    15:03:28 26/06/85   ACTIVE PARM'S

     RIBBON POINTS            SPEEDS                  TRAVERSE  MODULATION
                                             AMPLITUDE           P-JUMP          PERIOD
        R1       R2          F1       F2       F1       F2       F1       F2      SECS
  1   0.000    0.000        30.0     30.0     0.00     0.00     0.00     0.00     2.0
  2   0.000    0.000        30.0     30.0
  3   0.000    0.000        30.0     30.0
  4   0.000    0.000        30.0     30.0
  5   0.000    0.000        30.0     30.0
  6   0.000    0.000        30.0     30.0          TRAVERSE  ACCELERATION
  7   0.000    0.000        30.0     30.0           F1 TO F2     F2 TO F1
  8   0.000    0.000        30.0     30.0             0.83         0.83
  9   0.000    0.000        30.0     30.0
 10   0.000    0.000        30.0     30.0            WINDER   SPEED
 11   0.000    0.000        30.0     30.0                 50.00
 12   0.000    0.000        30.0     30.0
 13   0.000    0.000        30.0     30.0
 14   0.000    0.000        30.0     30.0         MAXBAND =1
 15   0.000    0.000        30.0     30.0

REDISPLAY/BANDING CHANGE/MOD & SPEED CHANGE/EXIT (R/B/M/E) ? [X]

 TYPE 'R' TO RE-DISPLAY THE CONTENTS OF THE TEMPORARY DB

 TYPE 'B' TO MAKE CHANGES TO THE BANDING AVOIDANCE DATA.
 TYPE 'M' TO MAKE CHANGES TO THE MODULATION PARAMETERS,
          INCLUDING WINDER SPEED.
 TYPE 'E' TO EXIT AND RETURN TO 'OCP CLEARED'.
 TYPE 'Z' TO ESCAPE TO 'OCP CLEARED'.
REDISPLAY/BANDING CHANGE/MOD & SPEED CHANGE/EXIT (R/B/M/E) ? [
```

FIGURE 4.2.6

-DISPLAY OR CHANGE TRAVERSE PARAMETERS   15:04:32 26/06/85   ACTIVE PARM'S

	RIBBON POINTS		SPEEDS		TRAVERSE MODULATION AMPLITUDE		P-JUMP		PERIOD
	R1	R2	F1	F2	F1	F2	F1	F2	SECS
1	0.000	0.000	30.0	30.0	0.00	0.00	0.00	0.00	2.0
2	0.000	0.000	30.0	30.0					
3	0.000	0.000	30.0	30.0					
4	0.000	0.000	30.0	30.0					
5	0.000	0.000	30.0	30.0					
6	0.000	0.000	30.0	30.0					
7	0.000	0.000	30.0	30.0					
8	0.000	0.000	30.0	30.0					
9	0.000	0.000	30.0	30.0					
10	0.000	0.000	30.0	30.0					
11	0.000	0.000	30.0	30.0					
12	0.000	0.000	30.0	30.0					
13	0.000	0.000	30.0	30.0					
14	0.000	0.000	30.0	30.0					
15	0.000	0.000	30.0	30.0					

TRAVERSE ACCELERATION
F1 TO F2   F2 TO F1
0.83   0.83

WINDER SPEED
50.00

MAXBAND =1

REDISPLAY/BANDING CHANGE/MOD & SPEED CHANGE/EXIT (R/B/M/E) ? [B]
MAXIMUM NUMBER OF BANDING POINTS: PRESENT = 1 CHANGE []
WHICH BANDING POINT TO CHANGE = [1]
BANDING AVOIDANCE POINT NUMBER 1

RATIO R1 PRESENT = 0.0 CHANGE [6.25]
RATIO R2 PRESENT = 0.0 CHANGE [6.20]
FREQ F1 PRESENT = 30.00 CHANGE [234.50]
FREQ F2 PRESENT = 30.00 CHANGE [230.25]
NEXT BANDING POINT = [0]

	RIBBON POINTS		SPEEDS		TRAVERSE MODULATION AMPLITUDE		P-JUMP		PERIOD
	R1	R2	F1	F2	F1	F2	F1	F2	SECS
1	6.250	6.200	234.5	230.2	0.00	0.00	0.00	0.00	2.0
2	0.000	0.000	30.0	30.0					
3	0.000	0.000	30.0	30.0					
4	0.000	0.000	30.0	30.0					
5	0.000	0.000	30.0	30.0					
6	0.000	0.000	30.0	30.0					
7	0.000	0.000	30.0	30.0					
8	0.000	0.000	30.0	30.0					
9	0.000	0.000	30.0	30.0					
10	0.000	0.000	30.0	30.0					
11	0.000	0.000	30.0	30.0					
12	0.000	0.000	30.0	30.0					
13	0.000	0.000	30.0	30.0					
14	0.000	0.000	30.0	30.0					
15	0.000	0.000	30.0	30.0					

TRAVERSE ACCELERATION
F1 TO F2   F2 TO F1
0.83   0.83

WINDER SPEED
50.00

MAXBAND =1

SECURITY EXECUTE:
OPERATOR NO == [1]
CODE NO = []
EXECUTE (Y/N) ? [Y] EXECUTED

FIGURE 4.2.7

```
-DISPLAY OR CHANGE TRAVERSE PARAMETERS    15:07:03 26/06/85   ACTIVE PARM'S

    RIBBON POINTS          SPEEDS               TRAVERSE  MODULATION
                                           AMPLITUDE          P-JUMP       PERIOD
       R1      R2        F1        F2       F1       F2       F1       F2    SECS
  1  6.250   6.200     234.4     230.2     0.00     0.00     0.00     0.00    2.0
  2  0.000   0.000      30.0      30.0
  3  0.000   0.000      30.0      30.0
  4  0.000   0.000      30.0      30.0
  5  0.000   0.000      30.0      30.0
  6  0.000   0.000      30.0      30.0         TRAVERSE  ACCELERATION
  7  0.000   0.000      30.0      30.0           F1 TO F2      F2 TO F1
  8  0.000   0.000      30.0      30.0             0.83          0.83
  9  0.000   0.000      30.0      30.0
 10  0.000   0.000      30.0      30.0           WINDER  SPEED
 11  0.000   0.000      30.0      30.0                50.00
 12  0.000   0.000      30.0      30.0
 13  0.000   0.000      30.0      30.0
 14  0.000   0.000      30.0      30.0        MAXBAND =1
 15  0.000   0.000      30.0      30.0

REDISPLAY/BANDING CHANGE/MOD & SPEED CHANGE/EXIT (R/B/M/E) ? [M]
WINDER SPEED    PRESENT =       50.00   CHANGE [150.05]
AMPLITUDE AT F1 PRESENT =          0.0   CHANGE [3.5]
AMPLITUDE AT F2 PRESENT =          0.0   CHANGE [2.5]
P-JUMP FOR F1 PRESENT =          0.0   CHANGE [2.0]
P-JUMP FOR F2 PRESENT =          0.0   CHANGE [2.15]
PERIOD PRESENT =          2.0   CHANGE [10]
ACCEL F1 TO F2 PRESENT =          0.8   CHANGE [1.5]
ACCEL F2 TO F1 PRESENT =          0.8   CHANGE [3.0]
    RIBBON POINTS          SPEEDS               TRAVERSE  MODULATION
                                           AMPLITUDE          P-JUMP       PERIOD
       R1      R2        F1        F2       F1       F2       F1       F2    SECS
  1  6.250   6.200     234.4     230.2     3.50     2.50     2.00     2.15   10.0
  2  0.000   0.000      30.0      30.0
  3  0.000   0.000      30.0      30.0
  4  0.000   0.000      30.0      30.0
  5  0.000   0.000      30.0      30.0
  6  0.000   0.000      30.0      30.0         TRAVERSE  ACCELERATION
  7  0.000   0.000      30.0      30.0           F1 TO F2      F2 TO F1
  8  0.000   0.000      30.0      30.0             1.50          3.00
  9  0.000   0.000      30.0      30.0
 10  0.000   0.000      30.0      30.0           WINDER  SPEED
 11  0.000   0.000      30.0      30.0               150.05
 12  0.000   0.000      30.0      30.0
 13  0.000   0.000      30.0      30.0
 14  0.000   0.000      30.0      30.0        MAXBAND =1
 15  0.000   0.000      30.0      30.0

SECURITY EXECUTE:
OPERATOR NO = [
```

FIGURE 4.2.8

4.4 THE HOST OCP TASK.

The host OCP task performs essentially the same function as the local VDU task. It is also menu driven, with all the same features as the local task. The main differences arise from four sources :

A) The host is the Master controller for the system.
B) System logging has to be performed by the host.
C) A "library" of operating instructions has to be held.
D) The host is controlling 24 positions.

The host computer is the T18 process management computer, which is in turn a foreground computer for the T30 process management computer. The host polls the ICU´s once every 5 minutes for position status information, and sends any new OPI´s that have been allocated to the machine. If the host detects that a position status is incorrect, or the ICU responds with a "Negative Acknowledge", or does not respond at all, the host prints the position number and error type on an alarm printer. Figure 4.4.1 shows the main menu options available on the host, and figure 4.4.2 shows the display produced by option 5.

Each OPI is allocated a unique number. Each time a new OPI is entered, it can be permanently stored on disc. In this way a library of OPI´s for different products can be built up, which reduces the amount of typing, and hence the scope for error when library OPI´s are recalled for use at later dates.

The T18 operators manual should be consulted for further information on the use of the host computer.

```
                    ***  ICU SUPERVISORY SYSTEM  ***
OCP OPTIONS:-
1-DISPLAY FIXED PARAMETERS AND WINDER STATUS
2-DISPLAY OR CHANGE OPI'S
3-PASS / ALLOACATE OPI'S
4-PRINT OPI
5-MAC/WINDER LINKAGE
6-UNIT OPERATION LOGGING
OPTION=[
```

FIGURE 4.4.1

*** ICU SUPERVISORY SYSTEM ***

CP OPTIONS:-

-DISPLAY FIXED PARAMETERS AND WINDER STATUS

-DISPLAY OR CHANGE OPI'S

-PASS / ALLOACATE OPI'S

-PRINT OPI

-MAC/WINDER LINKAGE

-UNIT OPERATION LOGGING

PTION=[5]

MAC/WINDER INFORMATION 15:07:39 26/08/85

NDR	LINK	ADDR	OPI-NUM	STATUS	OPI-RUN	WNDR	LINK	ADDR	OPI-NUM	STATUS	OPI-RUN
1C	1	3	10001	WORKING	10001	1A	1	1	10001	WORKING	10001
1D	1	4	10001	WORKING	10001	1B	1	2	10001	WORKING	10001
2C	1	7	10001	WORKING	10001	2A	1	5	10001	WORKING	10001
2D	1	8	10001	WORKING	10001	2B	1	6	10001	WORKING	10001
3C	2	1	10001	WORKING	10001	3A	3	1	10001	WORKING	10001
3D	2	2	10001	WORKING	10001	3B	3	2	10001	WORKING	10001
4C	2	3	10001	WORKING	10001	4A	3	3	10001	WORKING	10001
4D	2	4	10001	WORKING	10001	4B	3	4	10001	WORKING	10001
5C	2	5	10001	WORKING	10001	5A	3	5	10001	WORKING	10001
5D	2	6	10001	WORKING	10001	5B	3	6	10001	WORKING	10001
6C	2	7	10001	WORKING	10001	6A	3	7	10001	WORKING	10001
6D	2	8	10001	WORKING	10001	6B	3	8	10001	WORKING	10001

*** ICU SUPERVISORY SYSTEM ***

CP OPTIONS:-

-DISPLAY FIXED PARAMETERS AND WINDER STATUS

-DISPLAY OR CHANGE OPI'S

-PASS / ALLOACATE OPI'S

-PRINT OPI

-MAC/WINDER LINKAGE

-UNIT OPERATION LOGGING

PTION=[Z]

ERMINAL FREE

FIGURE 4.4.2

CHAPTER 5 : CONCLUSIONS.

5.1 ASSESSMENT OF COMPUTER SYSTEM.

The MAGIC development package performed all the functions required of it, although the initial release had a number of bugs in it, and some aspects of its operation were not as documented. Ways were found around all of these problems, although there is no doubt they delayed the project completion date. However it is inevitable that the first project attempted with any new system will take longer because of the learning required. Most of the problems have been fixed in later releases of the package, and considerable experience has been gained with its intricacies since then, and we feel confident of our ability to use MAGIC to develop applications software. From an efficiency point of view, RTL/2 modules compiled under MAGIC are about 20% longer than the equivalent compilation for PDP processors, so there is some overhead in code size when using MAGIC.

A question mark hangs over the future of the MAGIC package, because it does not have a large user base, and SPL are unwilling to commit themselves to future maintenance, which raises the problem of support and keeping abreast with technology advances that SANS might want to take advantage of. The runtime environment manual for RTL/2 on 8086 microprocessors indicates that the MAGIC implementation has adhered to INTEL´s iRMX conventions for register and stack usage. Contact has been made with a large company in the UK who have implemented RTL/2 on an RMX executive, and claim that it is comparatively simple to do. This offers a useful alternative to SMT if problems should ever be experienced

No problems were experienced with the SBC 88/25 computer board, apart from the usual misunderstandings and problems of development. There are now nearly forty of these boards installed in various places around the plant, and so far there has only been one genuine failure, all the rest being due to overstressing, incorrect connections or other user related problems. There have not been any problems with execution speed either. With any real time operating system it is necessary to have a good knowledge of the executive, the computer and the application to ensure that the computer performs the task required of it, and careful system design is essential. Provided due attention is paid to these factors, it seems unlikely that the computer will not be "powerful" enough to perform any of the applications foreseen at present. If this should ever happen, the application can always be directly transported to one of the more powerful INTEL computers.

The MAGIC / SBC combination has now been used in two other successful applications (a 256 thermocouple multiplexer, and a 48 stage process timer / sequencer). Two other immediate applications are in the investigation stage, and at least six other applications are being considered. The combination has become a very satisfactory standard, and the results have vindicated the original investigation.

5.2 ASSESSMENT OF PROJECT PERFORMANCE.

5.2.1 PROCESS PERFORMANCE.

There are two common measures of the success of a production process in use at SANS. One is conversion efficiency, which is the ratio of the useful product made to the amount of wasted product. The second measure is customer returns. On both of these measures the performance of Machine 5B has improved, the conversion efficiency has risen from 95.5% to 97% (on average), while customer returns have dropped from 9% to 3%.

There are three possible sources for these improvements : the first is improved inverter reliability overall because of the single inverters per position approach; the second is improvements due to the banding avoidance facility; the third can be attributed to other modifications that were made to the machine when the M/C 5B changeover was done. Unfortunately it is extremely difficult to separate the different influences to gauge their individual effects.

The inverters used to drive the winder and traverse rolls have proved to be extremely unreliable, and the computer hardware department is currently involved in a project to make an inverter system which avoids the problems of the existing units and incorporates the good features of other more reliable units used elsewhere on the plant. Unfortunately no records of inverter failure prior to the changeover are available, so it is not possible to compare the amount of lost production due to inverter failure before and after the project. However the performance of two other machines which use the same winders but the old inverter per machine approach, was compared with the performance of the new inverter per winder system. Maintenance logs were checked for winder inverter failures for each of the three machines from 1/1/85 up to 27/8/85. The results were as follows :

MACHINE NUMBER	NUMBER OF FAILURES	DURATION OF ALL FAILURES	POSITION HOURS LOST
5A	5	22 HRS	528
5B	33	66 HRS	66
6B	5	50 HRS	1200

The position hours column was derived by multiplying the number of positions affected by the time that the positions were not in operation.

Machine 5B has the new control system. The production lost on this machine due to inverter failure is between 8 and 18 times less than that on the old machines, so it is clear that a considerable improvement in lost production has been made using the inverter per position approach. Although the number of failures on M/C 5B is more than six times that of the other machines, because there are 24 inverters the reliability of the per-position inverters is 3.5 times greater than that of the per-machine inverters. It seems that greater productivity has been exchanged for increased maintenance. Using one common product as a baseline, the cost of the failures quoted in the table above are R10 890, R1 362 and R24 750 respectively. If the new approach was used on the older machines, R32 916 would have been saved. However, it is difficult to estimate maintenance costs, and the cost of converting the machines also has to be considered.

The effectiveness of the banding avoidance feature could be tested by comparing two batches of product, one made with the banding avoidance facility, and the other made without. However the production department is understanderbly reluctant to experiment in this fashion with product which is to be sold, as the feeling is that it does help, and the result of experimentation would inevitably mean lower conversion efficiencies. To be meaningful such a test would have to be done in a systematic fashion for a range of products, which cannot be justified at present. Determination of the effectiveness of banding avoidance will have to wait until a justification for the tests can be found, or until a different method for its determination can be devised. However the conversion efficiency on machine 5B is 1.5% better than that on machine 6B when both are making the same product. It is not possible to say conclusively that this is due to banding avoidance, but since the machines are very similar in most other respects it is possible that banding avoidance is the source of the improvement.

Similarly it is not possible to isolate the effects of other modifications to the machine. The only way these other factors could be accounted for is by measuring the other two factors and then subtracting their effect from the total improvement.

5.2.2 ICU PERFORMANCE.

Extensive testing has been carried out on the operation of the Inverter Control Unit. This was done using calibrated stroboscopes which can measure the speeds of the different rotating parts extremely precisely. Stroboscopes are only of use where the speed of the shafts are constant, reliable results cannot be obtained when the speeds are changing. The table below shows the results of the tests :

MOD AMP / SET SPEED		0%	2%	4%
100 Hz	C	5969.9 (0.1)	5963.0 (0.2)	5952.7 (0.5)
	S	5973 (0.05%)		
200 Hz	C	11934.2 (0.1)	11923.4 (0.3)	11905.7 (1.0)
	S	11936(0.02%)		
270 Hz	C	16099.0 (0.2)	16090.0 (1.7)	16059.3 (1.9)
	S	16104 (0.03%)		

C = results as measured by computer (brackets show maximum deviations from mean speed in RPM).
S = results measured by strobescope (brackets show percentage deviation between speed as measured by strobe and computer).
All measured values in the table are in RPM.

NOTES :

1) It was not posssible to measure speed with modulation applied to the traverse.

2) The difference in the speeds measured by the strobe and computer (with no modulation) were less than or equal to 0.05%,

which is well within the specification.

3) Motor slip is given by the difference between the set speed and the measured speeds. With no modulation the slip is as follows :

Traverse set speed	Measured speed	Slip
100 Hz (6000 RPM)	5973 RPM	0.45%
200 Hz (12000 RPM)	11936 RPM	0.53%
270 Hz (16200 RPM)	16099 RPM	0.59%

These results confirm Hudgell's findings (reference 2) that motor slip is about 0.5%, with no modulation or load. The increase in slip with speed can be attributed to windage and friction.

4) The mean motor speed measured by the computer decreases as the modulation increases, or expressed another way, motor slip increases as modulation amplitude increases. The magnitude of the slip is approximately the same as that observed by Hudgell (reference 2) using a different technique, which indicates that the speed as measured by the computer is accurate. The table below shows the slip measured by the computer for different speeds and modulation amplitudes :

	Modulation amplitude		
Traverse set speed	0%	2%	4%
100 Hz	0.45	0.62	0.79
200 Hz	0.53	0.63	0.79
270 Hz	0.59	0.68	0.87

5) A cyclical fluctuation was observed in the speed of the traverse measured by the computer, and the magnitude of this fluctuation was directly proportional to the modulation amplitude. It's source is probably the effect theoretically predicted in Chapter 3.3.3.1, which predicted a speed measurement error brought about by the slip of the motor not being accounted for. The cyclical nature of the effect is probably due to a

"beating" effect between the actual measurement interval and the modulation period, since the starting point for the measurement will be at a slightly different point on the modulation waveform each time. The worst case value occurs at 270Hz with 4% modulation. The measured value of the fluctuation was 0.012%, and the theoretically predicted value was 0.07% (the measured values were less than the theoretically predicted values in every case). These values are of the same order, and are at the resolution limit of the measurement method, so it would seem that the theoretically predicted speed errors were correct.

5.2.2.1 CONCLUSIONS ON SPEED MEASUREMENT.

The overall accuracy of both traverse and chuck speed measurements was better than the 0.1% specified, being generally of the order of 0.05%, so no further attempts were made to improve the operation of the speed measurement system.

5.2.3 DEVELOPMENT AND COMMISSIONING.

The project was commissioned in May 1984, nine months after it was started, and one month behind our own schedule. The rest of the project also ran a bit slower than expected, so there was never a point where the ICU development held back the overall project implementation. Time for contingencies had been allowed for, and in fact the project was completed slightly ahead of overall schedule.

Several bugs of varying degrees of seriousness were found in the software, and the last version to be made was version 7 of October 1984. One of the most serious bugs was in one of the SMT modules released by SPL. This was the "REAL COMPARE" function, and caused the ICU to work incorrectly. The problem was traced and the offending source module was edited. The bug has been fixed in the later releases of MAGIC.

Because of the need to get the machine into production quickly (anticipated output had already been sold to customers), the commissioning time was cut short, and this had a very bad effect on the initial reliability of the wind up system, and in fact it was not until mid-September, four months later, that all 24 positions were successfully commissioned. The main problem was with the power supplies for the ICU´s, which kept shutting down for no apparent reason (see chapter 2.3.2), and with problems in the mechanical construction of the ICU drawers. Any work performed on the ICU cabinet carried a high risk of shutting the power supplies down, which caused all the winders to stop whatever production was occurring. As a result the production department were reluctant to allow work on the machine, while at the same time putting pressure on the department to make it work properly. In the end the computer hardware department had to insist on a total shutdown for several days in order to correct the problems. This involved changing the earthing on the power supply system, modifying the power supplies themselves, and changing plugs and sockets on the ICU´s.

Longer term but less severe problems were encountered with the host communications. On one link in particular, a lot of problems were experienced in getting all eight stations working. This was mainly because there were three separate problems masking each other, and once again it was very difficult to work on the system without stopping production, which meant that it took much longer to find the problems. The situation was only corrected when the machine was shut down for five days for other reasons. The source of the problem was found by disconnecting all the stations from the line, and testing the continuity of all the plugs and sockets from one end of the link to the other. From this it was discovered that two of the links had been crossed over, so the search for the problem had been directed at the wrong link on previous attempts. When this problem had been cleared up, each station was individually reconnected to the link and tested for operation, which revealed that one position was not communicating at all. After this the stations were reconnected onto the link one by one, which led to the discovery that one station had its transmit buffer permanently enabled, so that no other station could transmit.

CONCLUSION

Two very important lessons have been learned from this phase of the project: Firstly the success of a project hinges on the physical construction of the unit, especially with regard to seemingly trivial items like plugs and sockets; Secondly, adequate commissioning time at the end of the development phase is more than compensated for by subsequent reliable operation.

The Control Unit side of the project has been operating extremely reliably since the start of 1985. There has been one computer failure since commissioning, and nearly all other problems have been due to the Inverter pulse driver transistors (on the I/O board) being burnt out, through accidental short circuiting when inverters are repaired or altered. This problem is easily remedied by swapping boards, and the Mark II I/O board has been modified to provide short circuit protection.

5.3) THE FUTURE OF THE ICU SYSTEM.

5.3.1) FURTHER DEVELOPMENT OF THE ICU.

There are two aspects of the ICU that still require more attention, namely : Cooling of the inverter control unit cabinet; and protecting the ICU's from host computer failures. Each of these will be considered in turn.

A) Cooling of the ICU drawers is done by blowing air up a duct that seals over the side of each ICU drawer in the cabinet. Each side of the drawer has a vent (see figure 4.1.1) that catches the air flow on one side and expels it on the other side. However the computers are running at a very high temperature, and it is clear that the air circulation is inadequate. This situation has serious implications for the long term reliability of the computers, and there are suspicions that the one computer failure recorded was due to overheating. Accordingly arrangements have been made to supplement the air circulation in the cabinet with the chilled air used to cool the inverters. This system has a high volume circulation, and should improve matters.

B) Every time an ICU station receives a data character from the host, the CPU is interrupted. If continuous data were received from the host, the CPU would spend all its time servicing the interrupts, and there would be no time left to service system and application tasks. This situation has arisen on two occassions, once when the host computer failed to a state where it was sending data continuously down the links, and once when the BML eight channel serial link board on the host was unplugged from its backplane, and random noise was sent down the lines. The result in both cases was to crash all the ICU's and stop production on the machines. Since it is impossible to prevent situations like this occurring, the solution will have to be implemented in software. One method being tried is to mask off the receive data interrupt for a fixed time duration once a certain number of interrupts has been exceeded. This is still under investigation.

5.3.2) NEW IMPLEMENTATIONS OF THE ICU.

There are at least nine other machines that could be converted in the same way as Machine 5B. Justification for conversion will depend on customer demand, and reliability of the old plant. At present there is a strong possibility of one other machine being converted, with two more at a later date. The ICU system produces yarn more efficiently than the old method, so it is a prime contender. A Japanese company produces a precision winder (ref 7) which has comparable performance, so the final decision will rest on cost. Using the Emersen inverters, the cost of the two systems is approximately the same, but the Japanese system is more reliable. With the prospect of reliable, cheap, locally produced inverters, there is a strong case for these units being used.

A second possible use is a modified ICU controlling existing machines. Analog controllers are difficult and expensive to obtain, and are the source of wasted production (through incorrect controller set up and inherent unreliability). The computerised ICU has shown itself to be far more reliable than the older controllers, and less prone to incorrect set up. A costing exercise has been done, and there are plans to install the computer controllers on six machines. There will be one computer per machine, it will not perform banding avoidance (so there would be no need for tacho's), and there would be no plant interlocks. In addition, it will have to control the meter pump and spin finish pumps as well as the traverse and winder inverters.

Enquiries about the system have been received from ICI in the UK and from Fibre makers in Australia. Negotiations are in progress with both companies for units to be sent for trials.

The Inverter Control Unit is still being assessed, but it is becoming increasingly clear that it offers reliable, repeatable and economical control for high speed Barmag winders, and that it has successfully satisfied all the requirements set out in the specification.

CHAPTER 6 : GLOSSARY OF TERMS AND ABBREVIATIONS.

BRICK	A basic program building block in RTL/2. There are two types of brick : data and procedure. All code and data in an RTL/2 is contained in bricks. In stand alone systems, data bricks will be copied into RAM, and procedure bricks will be contained in ROM.
CAKE	A cylinder or tube on which undrawn yarn has been wound.
CHUCK	Free running spindle which has four cardboard tubes placed on it for winding thread onto.
CONVERSION EFFICIENCY	The ratio of useful yarn made to useful yarn made plus wasted yarn.
DOFF	Remove a cake of yarn from the chuck.
H-TASK	Hardware task. See S-task.
IC	Integrated Circuit or "chip".

ICU	Inverter Control Unit. The unit which controls a Barmag Winder head, and forms the subject of this thesis.
INDIRECT COMMAND FILE	A file on an RSX system which contains MCR and other utility instructions which will be automatically executed by the computer.
MELT POOL	Part of a nylon spinning machine where polymer chip is melted prior to extrusion.
MCR	Monitor Console Routine. A command line interpreter commonly used in RSX.
OCP	Operator Command Processor. A program which allows an operator to control a computer through a terminal.
OPI	Operating Instruction. Set of machine settings which define how a particular batch of yarn will be made.
P-JUMP	An instantaneous change of frequency in the traverse modulation signal which compensates for the inertia of the traverse roll.
PIC	Programmable Interrupt Controller. An IC (8359A on the SBC 88/25) used to detect and control interrupts to the 8088 microprocessor.
PIT	Programmable Interval Timer. An IC (8253 on the SBC 88/25) used in computers for generating pulse trains and measuring time durations. Also known as a Real Time Clock or RTC.

PPI	Programmable Peripheral Interface. An IC (8255 on the SBC 88/25) used in computers to send or receive binary data.
QUENCH CHIMNEY	Part of a Nylon Spinning Machine where yarn is cooled and crystallised after extrusion from the melt pool.
RL02	A 10 Megabyte removable hard disk drive used on PDP-11 minicomputers.
RSX	A multitasking real time operating system used on DEC PDP-11 minicomputers.
RTC	Real Time Clock. See PIT.
SPINNERET	A small orifice through which molten polymer is pumped to form Nylon yarn.
S-TASK	Software task. In RTL/2 there are two types of task. Software tasks are controlled by the scheduler and come into operation according to the state of other S-tasks in the system. H-tasks directly control the computer hardware, and are usually invoked by an interrupt.
STRING UP	Thread Nylon yarn onto a winder.
THERMEX	A heated liquid used for transferring heat in Nylon Spinning machines.
UIC	User Identification Code. Defines an account in the RSX operating system. User and system files can only be accessed by suppling the correct UIC and password for the account.

UPS	Uninterruptable Power Supply. A power supply driven off a battery system that continues operation through ESCOM supply failures.
USART	Universal Synchronous Asynchronous Receiver Transmitter. An IC used in computers to convert serial data streams from peripheral devices such as terminals or printers into the parallel data required by the computer, and vice versa.
WIND UP	Draw yarn onto a cardboard tube to form a cake of yarn.
WRAP	Also known as "head wrap". Yarn which has broken off and wrapped around the windup roll instead of the spinning tube.

CHAPTER 7 : CITATIONS.

1) GIBB R.D. Status report on Barmag winder developments. Research note R375/81, ICI Fibres Research Engineering and textile department Harrogate.

2) HUDGELL A.W.D. Winder traverse modulation. Harrogate lab report HLR 135/80. ICI Fibres Research Engineering and textile department, Harrogate.

3) EVANS G.B. Optimum Modulation parameters report. ICI Fibres, Harrogate. Ref GBE/GLC.

4) LAFEBER A. Theoretical Investigation to avoid banding by improved modulation. ICI Fibres, Harrogate. Ref AL/CK.

5) HUDGELL A.W.D. Winder traverse tension. ICI Fibres, Harrogate. Ref FO/0.335/AWDH/GLC.

6) HUDGELL A.W.D. SYKES G. High speed winding. Survey of frictional and geometrical effects. ICI Fibres Research Engineering and textile department, Harrogate.

7) CAMPBELL W.E. Random or Precision Winding - choosing a system. Textile Institute and Industry, Oct 1979 p367.

8) A number of journals were consulted, see : Electronic Design, March 18th 1982 p77 and April 15, 1982 for a comprehensive survey. In addition see : Pulse Oct 1984 p6 and Dataweek Feb 24th 1984 for a survey of the South African situation.

9) INTEL Microprocessor and peripheral handbook, 1983.

10) INTEL iAPX 86,88 user's Manual.

APPENDIX A

APPENDIX A : INTRODUCTION TO NYLON SPINNING.

Nylon spinning is a mixture between a batch orientated process and a continuous process. The production of the Nylon threadline is continuous, but the winding up of the yarn is batch orientated. This thesis is concerned with the control of the wind up head, which is at the very bottom of the machine. Figure A.1 gives a general view of the wind up floor of a Nylon spinning machine, showing the 24 wind up heads. In one sense the production of the threadline is a batch process in that the market for most types of nylon is not large enough to allow for continuous production. So a batch of one particular type of yarn will be made, after which the machine will be "changed over" to manufacture a new type of yarn. The rest of the process on the higher floors will be briefly described for context.

Dried, pre-crystallised polymer is fed by an archimedian screw at a controlled rate into a "melt pool", which is heated to a carefully controlled temperature. Molten polymer is then pumped out of the melt pool through spinnerets which are small orifices of a carefully controlled shape and size, and from there through a "quench chimney" which cools the yarn and causes it to solidify. From the quench chimney it is fed to the winding head which winds the yarn onto a cardboard tube called a "bobbin". Each wind up head actually takes four threadlines, which are wound onto four cardboard tubes, which are called "cakes" or "cheeses" when they are full.

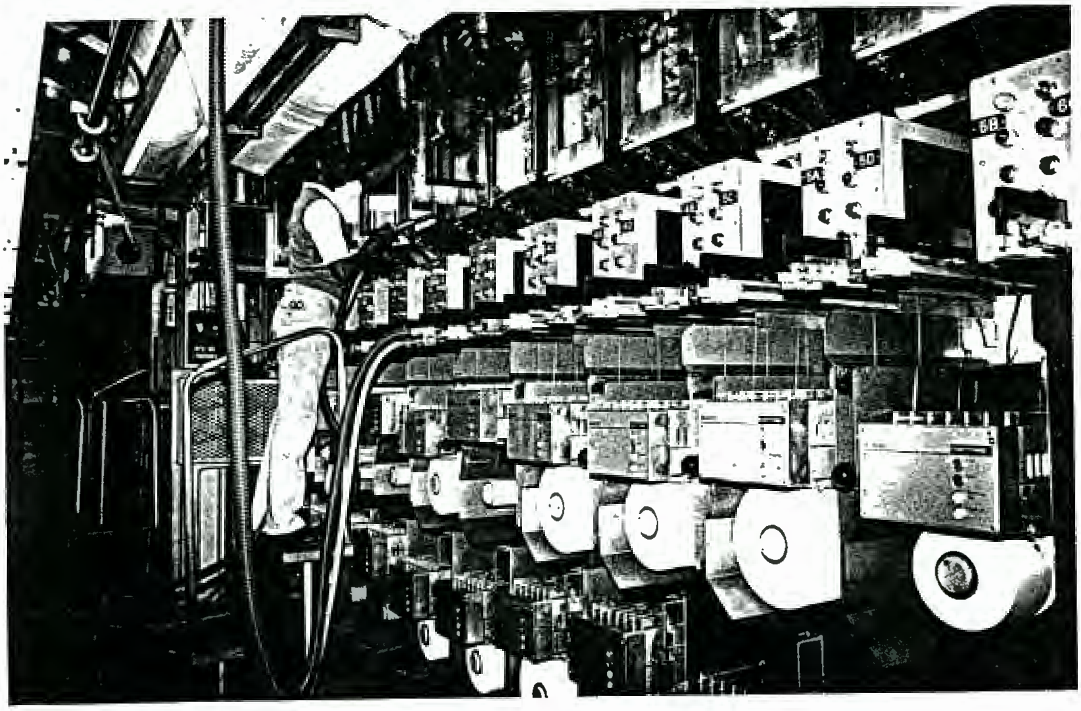


FIGURE A.1

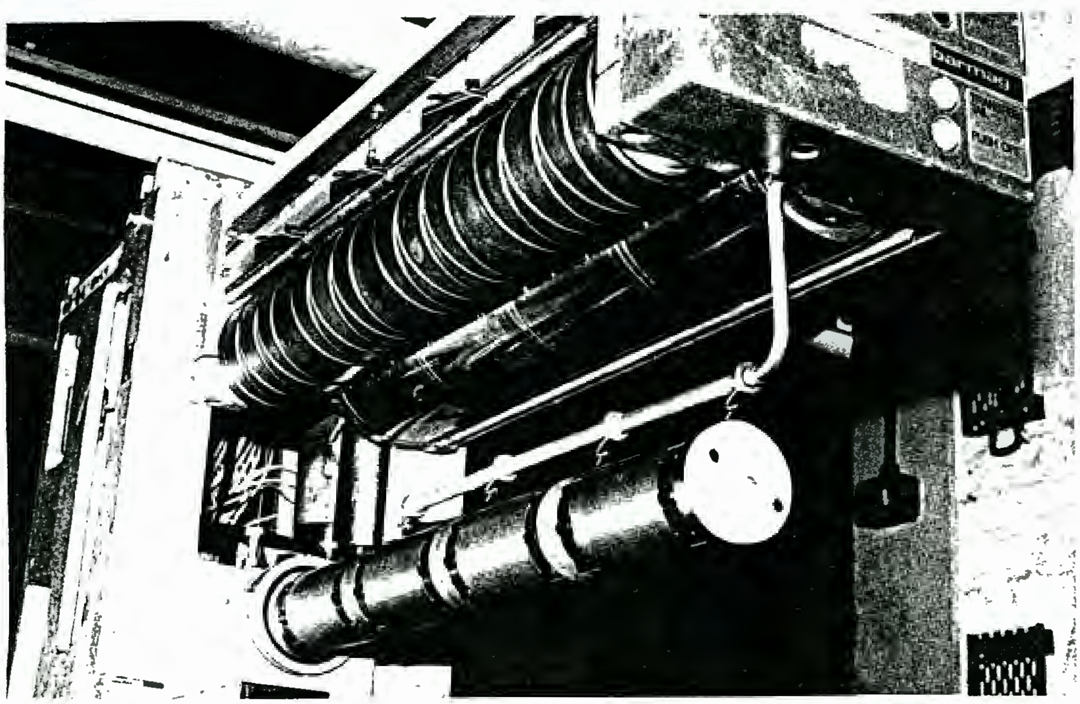


FIGURE A.2

Figure A.2 shows an unloaded, free standing wind up head. Four main components can be seen in the diagram. The first is the grooved traverse roll. The yarn runs in the groove, which guides the yarn onto the surface of the cake. The second component is the the smooth, shiny wind up roll. This is held in contact with the cake, causing it to rotate by frictional contact. The third component is the chuck which holds the cardboard tubes, and the fourth component is the transfer tail which transfers the yarn onto the cardboard tube at the start of the wind up period. Figure A.3 shows the winder in operation. If the picture is examined carefully, the threadlines can be made out as they pass through the traverse tips, into the groove roll and then onto a half full cake.

Each of the four threadlines are guided through traverse tips, which shuttle backwards and forwards in synchronism with grooves in the traverse roll. A close up view of the traverse tips is shown in figure A.4. The traverse tips guide the yarn into grooves on the traverse roll, which then feeds the yarn onto the cake in regular layers. As yarn gets wound onto the cake, its diameter grows, and so the winder head gradually lifts. Once the cake is full and the winder head has risen as far as it can go, the machine is "doffed". Firstly the threadlines are cut and a vacuum suction gun used to draw off the yarn from the quench chimney (Figure A.1 shows an operator in the process of doffing a winder, using a suction gun); secondly the winder roll is lifted off the cake surface, and the four cakes are stopped; and thirdly, the cakes are removed from the chuck, and replaced with new cardboard tubes so that the process can begin again.

The winder roll is actually the armature of a two pole permanent magnet synchronous motor, the stator being inside the armature. The function of the winder roll is to draw the yarn onto the cake at a constant speed and tension. The cake is surface driven to ensure that the yarn take up rate and tension are constant. If the chuck were driven directly, the surface velocity of the cake would increase as its diameter increased, and so the speed of the chuck would have to be proportionately decreased. This problem is

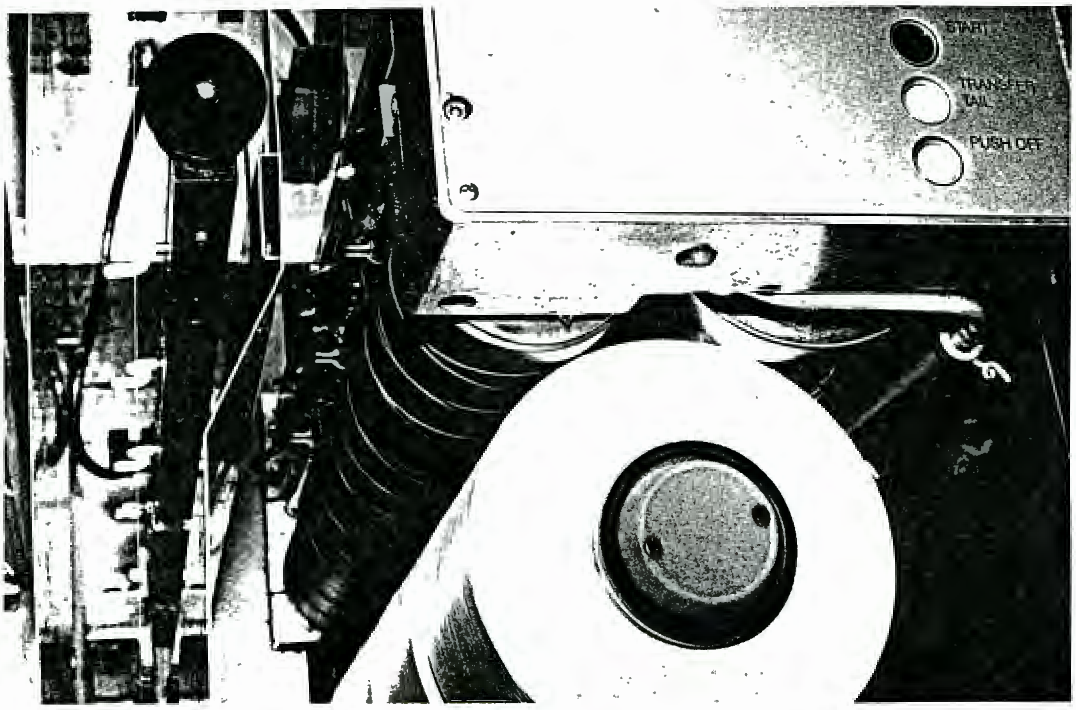


FIGURE A.3

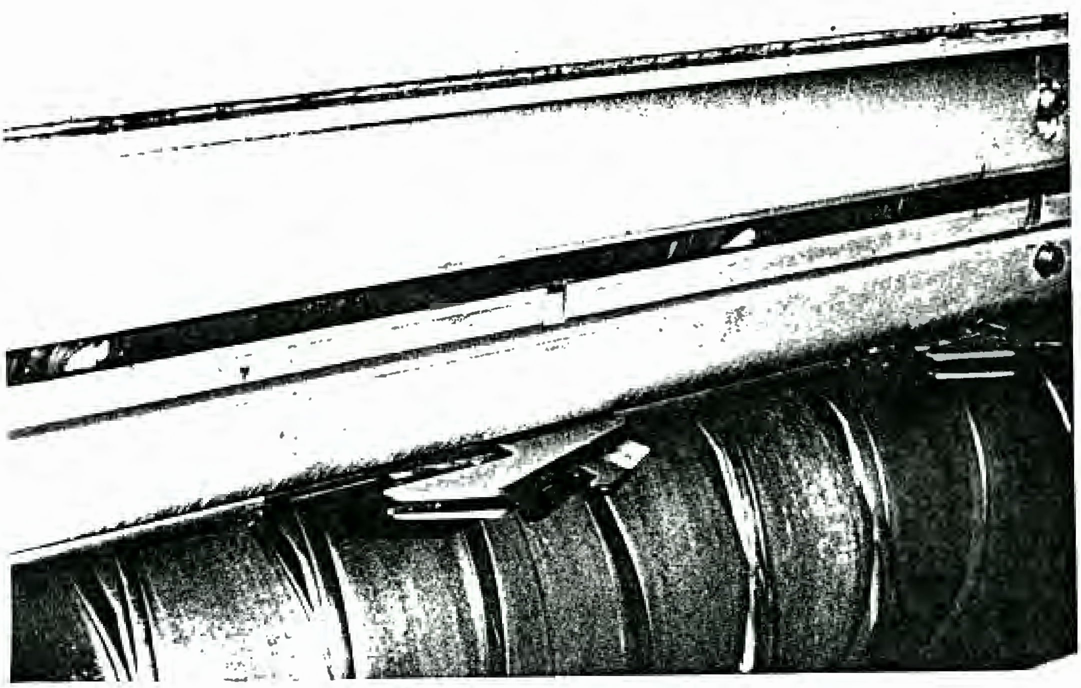


FIGURE A.4

avoided by using surface drive. A synchronous motor is used because yarn tension determines yarn quality, and tension depends on the wind up rate, so close speed control is essential. There is a small amount of slip between the winder roll and the cake which also has to be taken into account (ref 5).

The traverse roll is the armature of a two pole induction motor. The function of the traverse roll and its associated groove roll and traverse tips, is to lay the yarn in a regular fashion onto the cake, so that when the yarn is taken off by the customer (often at a high speed), there are no yarn breaks or snagging which might cause variations in yarn consistency. At first sight this may seem a trivial problem, but there are a number of problems. An examination of Appendix B, which is the production specification for the project, shows that a maximum wind up speed of 6000 metres per minute is required. A wind up period can last up to 12 hours, which means that it is possible to get up to 4000 km of yarn on a single cake. The customer must be able to draw this whole length off without any breaks or snags.

Problems arise when the cake rotates an integral number of revolutions for one double stroke of the traverse tip, in other words when the ratio of the chuck and winder speeds reaches a critical value. When this happens the crossover points of the yarn start falling in exactly the same place on successive layers of the cake, and ridges form. These cause the winder roll to vibrate against the cake surface, which compacts and traps filaments of yarn under lower filaments. This trapping causes the yarn to break when it is pulled off the cake. If the vibration becomes severe enough, upper layers can slip on lower layers towards the centre of the cake where tangles form, causing the threadline to snap and the yarn to wrap itself around one of the rolls (this condition is known as a "head wrap"). Theoretically this problem should occur instantaneously, but in practice conditions are sufficiently close to the critical ratio for the effect to occur for an appreciable time.

There are two methods of overcoming these problems. The first is to frequency modulate the traverse speed. This introduces a cyclical change in traverse frequency which ensures that the critical ratio only occurs for short periods of time. The amplitude and the rate of change of the modulation will determine how long the traverse speed dwells in the critical range. The second method of avoiding this effect, known as "banding" or "ribboning", is to rapidly alter the mean traverse speed as the critical point is approached, in order to pass it very rapidly. Once clear of the critical ratio, the mean speed is changed back to its original value.

Nylon Spinning Machines up until the present have only been able to use traverse modulation because all 24 winder heads are controlled by two inverters. The use of individually controlled motors allows a combination of both methods to be used. Traverse speed modulation removes a large proportion of the problem, whilst a change in average frequency overcomes most of the remainder. The degree of the problem varies from product to product, so different modulation and avoidance parameters are used in different situations, which accounts for the need for a programmable controller. There is also a very rigid limitation on how far the mean speed can be changed because as mentioned before the tension of the yarn is critical. There are some ribbon points on some products that it is not possible to overcome. However, a considerable improvement can be obtained by using these methods.

Figure A.5 shows the traverse modulation waveform used. The function of the P-Jump is to overcome the inertia of the roll, which makes reversal of roll acceleration unacceptibly slow if a simple triangular modulation is used. It should be noted that large modulation values shorten inverter life spans, and so are avoided as far as possible.

Figure A.6 is a graphical representation of what happens when the second method of ribbon avoidance is used. As the chuck speed decreases and a critical point is approached, the traverse speed is rapidly decreased from x to y so that the ribbon point is

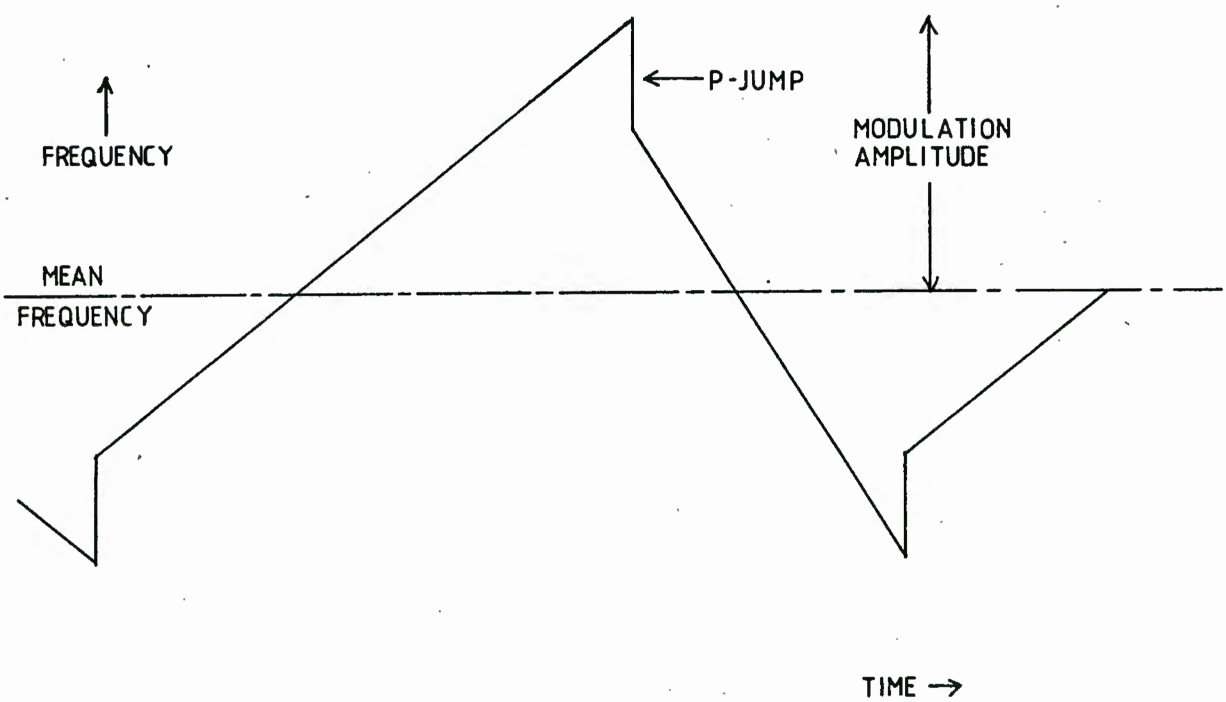


FIGURE A.5: TRAVERSE MODULATION WAVEFORM

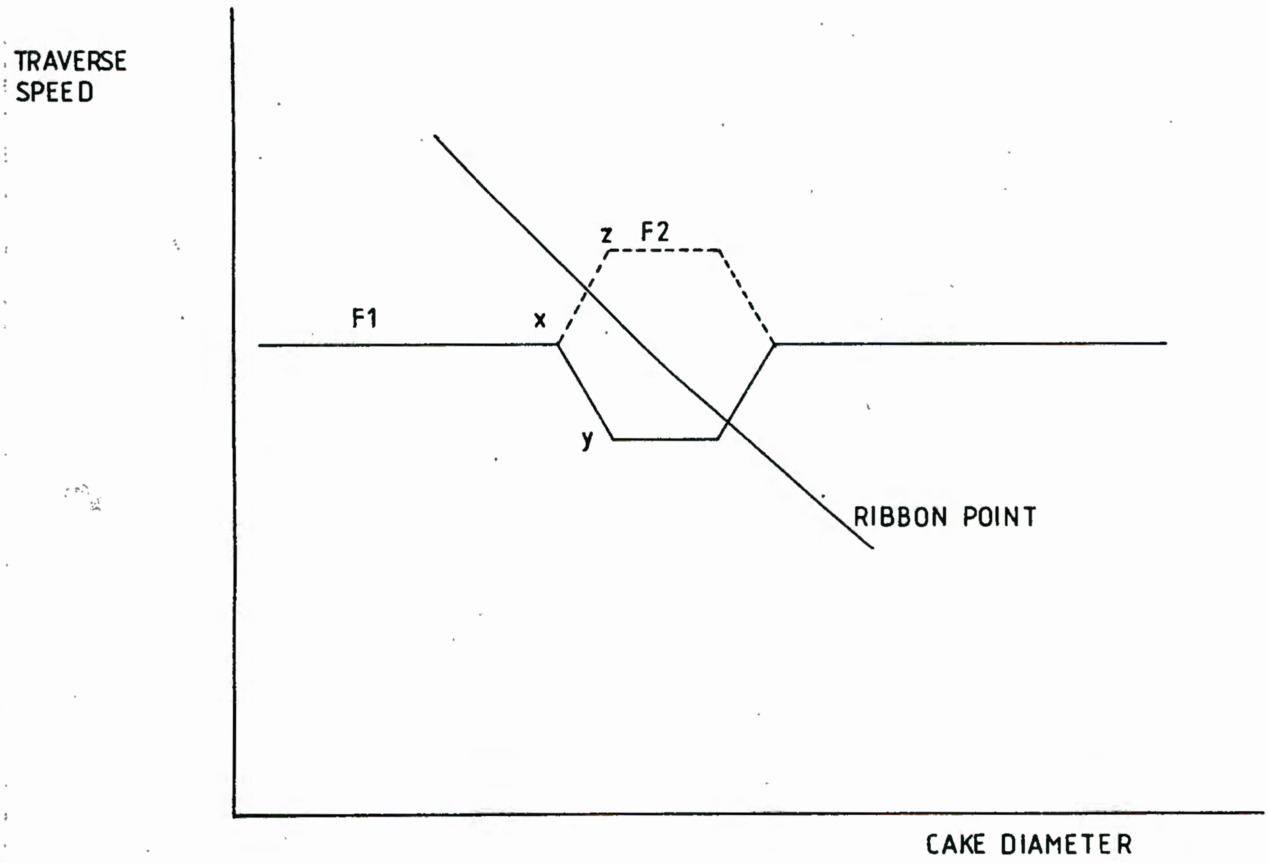


FIGURE A6 BANDING AVOIDANCE

avoided. As the ribbon point is approached again, the mean speed is altered as rapidly as possible back to its original value so that the ribbon point is traversed as rapidly as possible. Avoidance could also be taken by going from x to z.

It should be noted that this method of winding is called "Random winding". There is another method used called "precision winding" where the traverse speed is kept (by mechanical means) at a fixed (and very carefully chosen) ratio of the chuck speed, which ensures that the crossover points are evenly distributed over the cake surface for all cake diameters. See reference 7 for a description and comparison of the two methods.

APPENDIX B

APPENDIX B : PRODUCTION SPECIFICATION FOR THE CONTROLLER.

This specification is the result of a series of consultations between the production department, which produced the original specification, and the computer applications group at SANS. The original production specification was studied and preliminary trials were done to test the feasibility of the goals and aims. Some of the less realistic requirements were altered after consultation with and agreement from the production department.

B.1) INTRODUCTION.

The project's aim is to produce a controller which will run the new individual position inverter system for BARMAG winders. It will allow specific control of the traverse to avoid banding and ribboning during cake build. The controller will be developed using an INTEL single board computer system running software written in RTL/2 and developed using the INTEL version of the MAGIC software development system. It will control the winder and traverse motor on each position. The standard ramp up, ramp down and traverse modulation (with P-jump) control will be provided. In addition a banding avoidance function will be provided. The strategy for this will be determined by a table of user set parameters. Advantage will be taken of the banding avoidance facility to provide a rough profile of wind on tension during the cake build.

B.2) OPERATIONAL LIMITS AND TARGETS.

PROCESS SPEEDS	MAXIMUM	MINIMUM	STEP SIZE	
Wind up speed	6000	2500		MPM
	12710	5217		RPM
	212	88	0.2	Hz
Traverse speed	6480	2700		MPM
	19155	7980		RPM
	319	133	0.2	Hz
Traverse modulation waveform				
Amplitude	+/- 4%	0%	0.2	Hz
P-jump	+/- 4%	0%	0.2	Hz
Period	30	2	1.0	sec

Modulation to be symmetrical.

B.3) BANDING AVOIDANCE.

Banding avoidance consists of ramping the traverse from one mean frequency "F1" to another temporary mean frequency "F2". This takes place as the system approaches a banding condition. The traverse would continue to run with modulation at this new frequency until the banding state had passed, when the traverse would be ramped back to normal mean speed. The avoidance action can be taken in a positive or negative direction.

During the wind up of a cake the chuck speed steadily decreases, so the ratio between the chuck and traverse speeds decreases. This can be specified as a ratio :

$$R = \frac{\text{CAKE RPM}}{\text{TRAVERSE ROLL SPEED} * (1/6)}$$

The conditions for banding can be set by specific values for R.

A typical banding avoidance operation will be specified by setting a ratio value at which to start avoidance and another corresponding one at which to end avoidance action.

All banding avoidance control parameters will be variable data set by operators and the only checks will be for consistency and safety. Allowance will be made for fifteen banding avoidance points. The acceleration and deceleration rates to and from banding avoidance are variable parameters, and are different from the general limiting rates derived from hardware limitations.

B.4) WIND ON TENSION PROFILE.

When a banding avoidance point "R1" is reached the mean traverse frequency is changed from F1 to F2. During this change modulation and P-jump will be disabled. After banding avoidance has been completed, when ratio "R2" has been reached, the mean traverse speed is ramped back to F1. The new value of F1 does not have to be the same as the original value. This enables a step profile of wind on tension to be implemented with the step changes occuring at banding avoidance points.

Modulation will be active during normal operation at the level set by the user. Modulation is disabled during the transition from F1 to F2 or F2 to F1, and the user can select different modulation parameters at F1 or F2.

B.5) TABLE OF PARAMETERS.

FIXED VARIABLES (not changeable by operators)

Winder :

Acceleration rate	1.0	Hz/sec
Deceleration rate	1.0	Hz/sec
Frequency maximum	212	Hz
Frequency minimum	13	Hz

Traverse :

Acceleration rate	3.0 (7.0)	Hz/sec
Deceleration rate	3.0 (7.0)	Hz/sec
Frequency maximum	319	Hz
Frequency minimum	13	Hz
Motor start up sync time	2	secs

Figures in brackets show maximum acceleration / deceleration rate in changing speed between F1 and F2.

CONTROL PARAMETERS (to be set by operators)

Banding avoidance start ratio	R1	x 15
Banding avoidance end ratio	R2	x 15
Normal mean traverse speed	F1	Hz x 15
Avoidance mean traverse speed	F2	Hz x 15

TRAVERSE MODULATION PARAMETERS.

Amplitude % of F1 in normal operation	0-4% of F1
P-jump % of F1 in normal operation	0-4% of F1
Amplitude of F2 in avoidance operation	0-4% of F1
P-jump % of F2 in avoidance operation	0-4% of F1

F1 to F2 rate of change	Hz/sec
F2 to F1 rate of change	Hz/sec

WINDER PARAMETERS

Winder speed	Hz

B.6) HARDWARE I/O

PULSE INPUTS

Chuck speed	RPM +/- 0.1%
Traverse speed	RPM +/- 0.1%

PULSE OUTPUTS

Winder frequency	Hz +/- 0.2Hz
Traverse frequency	Hz +/- 0.2Hz

B.7) PARAMETER MODIFICATIONS.

Direct access to the control parameter table will be possible through an interactive display and modification section of the system. It will use a standard VDU and will have security protection for the data base change functions. The responsibility of data integrity and validity will be the responsibility of the user, except where safety or consistency considerations indicate an error.

All control parameters will be displayed, and a minimal set of status data for stand alone operation will be provided. This will consist of :

Current banding point (1-15)
Status (stopped, starting, running-normal, running-avoidance)
Traverse output frequency (Hz)
Winder output frequency (Hz)
Cake speed (RPM)
Traverse speed (RPM)
Current ribbon ratio
Cake diameter.

APPENDIX C

APPENDIX C : OVERVIEW OF INVERTER CONTROL UNIT PROJECT.

Figure C.1 shows how the different parts of the system fit together. The winder and traverse inverters with their integral armatures are each driven by a variable frequency inverter. The output frequency of each inverter is controlled by a pulse train fed to the inverter from the computer.

The I/O board isolates the plant from the computer, and converts computer TTL levels to plant transducer levels.

Each of the 24 ICU's is linked to a PDP 11/23 host computer by an RS-422 multidrop serial link. The host computer monitors and logs the condition of each ICU, and sends operating instructions (control parameters for the product being made) to the ICU's. The type of yarn to be wound is chosen from a library held by the host, and sent over the serial link. This avoids the tedium of entering operating instructions (which could consist of up to 40 parameters) by hand, and also allows alterations to be made to the parameters very rapidly.

The traverse and winder inverters are housed in the inverter cabinet, which is fed traverse and winder motor control pulses, and "inverter run" interlocks from the computer. The inverters supply "inverter not tripped" interlocks back to the computer.

The traverse and winder rolls are supplied with power from the inverters, and proximity detectors mounted on the traverse roll

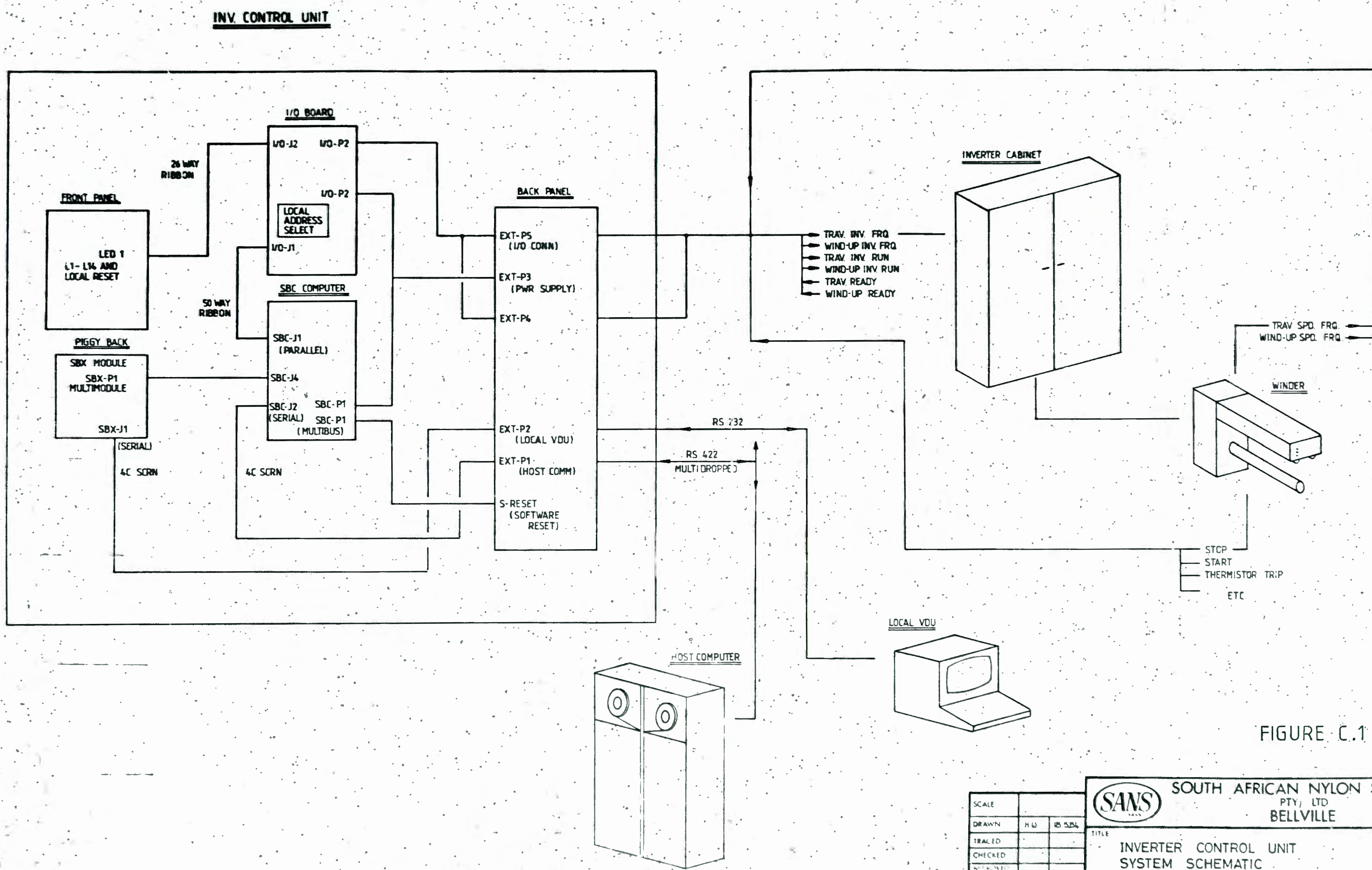


FIGURE C.1

SANS — SOUTH AFRICAN NYLON SPINNERS (PTY) LTD, BELLVILLE

TITLE: INVERTER CONTROL UNIT SYSTEM SCHEMATIC

DWG No: SAB 1627

CODE No: P-00-01-49D

SCALE	
DRAWN	H.U. 18.5.84
TRACED	
CHECKED	
APPROVED ENGINEER	
APPROVED SANS	

DESCRIPTION	DRWN DATE	CHKD DATE	APPD DATE	REV	DESCRIPTION	DATE	REFERENCE	DWG No

and chuck send speed information back to the computer. These are used to determine the ratio of their speeds, on which the decision to take banding avoidance by altering the mean speed of the traverse is based. The winder also has various interlock signals which are fully described in chapter 2.2.

TRAVERSE AND WINDER INVERTERS.

A brief description of the inverters used in this system will be given to aid understanding of the task the computer has to perform. Figure C.2 is a block diagram of the main parts of the inverter.

A three phase 460V supply is rectified and fed to a pulse width modulated chopper transistor, which produces a variable DC voltage (after the AC component has been filtered out with an inductor). The variable DC voltage is fed to a three phase transistor bridge. The transistor bases are driven by a ring counter which is clocked by the pulse train from the computer. This is why the computer output frequency has to be six times the required frequency of operation. The overall result is a six step variable frequency variable voltage three phase supply.

The computer pulse train is also fed to a frequency to voltage converter which is used to control the duty cycle of the chopper transistor, which gives a constant Volts to Hertz ratio. A digital speed signal was used rather than an analogue one because of the very tight speed tolerances required in the project.

The protection logic monitors over voltage, under voltage and over current conditions, and trips the inverter if safe limits are exceeded. Over current protection is provided for peak and average conditions.

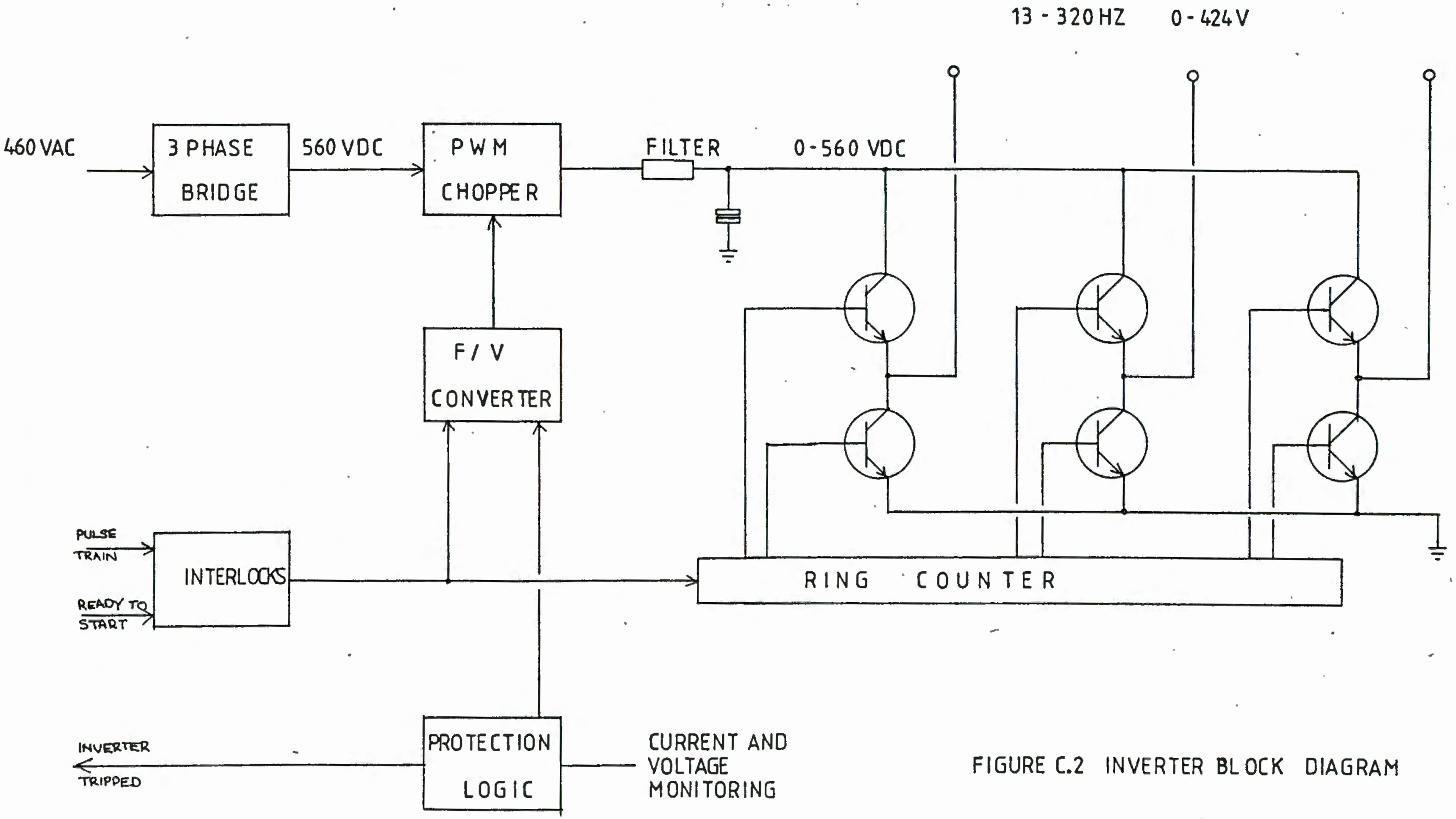


FIGURE C.2 INVERTER BLOCK DIAGRAM

APPENDIX D

APPENDIX D : OPERATIONAL REQUIREMENTS FOR COMPUTERS AT SANS.

This appendix presents the results of a survey done at the start of the project to assess the requirements for dedicated microcomputer control at SANS. The term "dedicated control" is used to distinguish it from "process control". In dedicated control a microcomputer controls part of a process, whereas in process control the entire process is controlled.

When this survey was done several previous attempts at dedicated control with microcomputers at SANS were investigated. The applications had been attempted at component level, and had shown poor results because of overrun budgets and development timescales, technical problems such as overheating and noise, or most importantly difficult maintenance of hardware and software. The purpose of this document is to try and define what is required of a dedicated microcomputer controller, both at the development stage and on the factory floor.

Several points became clear at the outset :

A) Computers installed on machines should be as flexible as possible, to allow future modifications or additions to the computer system to be done as easily as possible, with minimum disruption to the existing system.

B) Standardisation is essential for ease and speed of development, installation, commissioning and maintenance. Staff should be trained once for the basic system, with supplementary training for new applications.

C) There are a wide range of commercially available board level products which can be used as a base for building systems. In-house component level development is justified only for large volume production or specialised applications, which will seldom be required at SANS. Experience has shown that the cheapest and quickest way of getting limited quantity applications installed and running is to use board level products.

D) The company has limited hardware production facilities, so the assembly of board level products such as specilised applications or I/O boards will be contracted out to manufacturing companies.

E) The company has a very large investment in DEC computers and development facilities, using the RTL/2 language running under the RSX, MTS and SMT operating systems. Any system chosen should make as much use of existing facilities as possible.

D.1) MICROCOMPUTER REQUIREMENTS.

ENVIRONMENTAL CONDITIONS.

A) Ambient conditions.

Operating temperature range	:	0 - 45 deg C.
Humidity	:	90% non condensing.
Atmospheric corrosion	:	Mildly reducing.

B) Mains supplies.

Voltage	:	220 VAC
Frequency	:	50Hz

Mains supplies often carry large amounts of electrical noise, which the system will have to tolerate.

C) Equipment location.

Computers will often have to be mounted inside machine cabinets on the factory floor, so housings should be available which can withstand corrosion, dirt etc.

SOFTWARE.

In order to take advantage of existing software and development facilities there was a strong motivation for using RTL/2 and SMT as the applications language and multi-tasking executive respectively. Two options presented themselves here. The first was to use a small DEC computer such as the PDP 11/03 or FALCON as the target system, and use existing development facilities. The second option was to find an RTL/2 cross compiler that would produce object code for one of the more common microprocessors. A company called SPL based in the UK produces such a package, called MAGIC. Because of a technology exchange agreement between SANS and ICI in the UK, it was possible to get MAGIC at a low cost. When the prices, facilities and support for various different microprocessor systems was compared, (see appendix E) it was clear that the DEC computers were much more costly than other systems, so INTEL Multibus Single Board Computers using the MAGIC development package were eventually chosen. The MAGIC package produces object code for 6809, 8086, 68000 and LSI-11 processors, so any system with these CPU´s is suitable for operation with MAGIC.

SIGNALS FOUND IN THE PLANT.

The system had to be able to handle the wide range of signals found in the plant, and precise tailoring of applications was necessary to avoid redundancy. In other words the system had to be as modular as possible to ensure maximum flexibility.

A) ANALOG SIGNALS

Inputs :	Voltage	0-1mV, 0-10mV, 0-100mV, 0-5V, 0-50V
	Current	0-20mA, 4-20mA, 0-100mA
Outputs :	Current	4-20mA

In most applications, up to twenty analog inputs or outputs are required, although in some cases up to 240 inputs are required

B) DIGITAL SIGNALS

Inputs :	Up to 220V AC or DC
Outputs :	Up to 220V AC or DC

In most applications anywhere from twenty to three or four hundred digital inputs and outputs are required.

C) MEMORY REQUIREMENTS.

Once the decision to use MAGIC had been taken, it was necessary to choose a microcomputer capable of hosting SMT and the application tasks. SMT on its own uses up to 16K bytes of code space and 2.5K bytes of RAM space. Applications tasks were expected to use up to 64K of ROM space and up to 8K of RAM space. During the development stage systems are loaded into RAM, so the target computer had to have facilities for extra RAM during development. Battery backup will be required in some applications.

D) REAL TIME CLOCKS.

Since the computer is required for real time applications, there must be some provision for real time clocks.

E) COMMUNICATIONS.

Most applications considered required bi-directional communication either with a host computer and/or a local VDU. The data to be communicated will be operating instructions or status reports in most situations, so the data transfer rate does not need to be high, and 9600 baud operation down an RS-422 or RS-232 link should be adequate.

APPENDIX E

APPENDIX E : COMPARISON OF MICROCOMPUTER SYSTEMS.

Summarising appendix D, the requirements for the microcomputer system are as follows :

A) 16 bit processor.
B) Approximately 64K Rom space and 8K RAM space with provision for larger amounts of RAM.
C) 9600 baud serial communications.
D) Provision for battery backup.
E) Provision for frequency and event counting by Real time clock.
F) Provision for a wide range of digital and analog I/O.

A study was done of the technical literature for a comparison between different single board computers, in the context of the South African market (see ref 8). The factors considered in making the selection were as follows :

A) Processor type.
B) Address and data bus widths.
C) Form factor.
D) Power requirememts.
E) Number of pins on backplane.
F) Number of suppliers.
G) Application development facilities outside the company.
H) Facilities provided by basic SBC, and facilities for further expansion.
I) Cost, both for the basic system and for expansion.

The availability of different systems in South Africa was then considered, and the following systems were found to be readily available :

A) SABUS.
B) STD bus.
C) VME bus.
D) VERSABUS.
E) S100 bus
F) The DEC Q-bus
G) MULTIBUS

SABUS and STD were ruled out at the outset, as they do not support 16 bit processors very easily. A 16 bit processor was not available for SABUS at the time, although one was being developed. However it was felt that the backplane did not really have sufficient pins for true 16 bit operation, and that this would eventually place restrictions on the system. In addition SABUS has a rather erratic history, and there were doubts about its long term viability. The VME and VERSABUS systems were discounted because of their prohibitive cost. Both systems have large backplanes, and the cost of housing coupled with the high cost of the basic board ruled them out. In addition the supply of the boards was very uncertain.

That left the S100, Q-bus and MULTIBUS. A cost comparison between the three systems was done for several applications envisaged. The MULTIBUS system proved to offer the best price / performance ratio. This coupled with the fact that MULTIBUS is the most widely used bus in South Africa and has the largest user base in real time control applications settled the issue in its favour.

APPENDIX F

APPENDIX F : COMMUNICATIONS PROTOCOL.

The communications protocol described below was arrived at with the following aims in mind :

1) The prime requirement is data transmission between the supervisory system and a remote unit. For reasons of economy a number of units will be connected to the same link, however it is considered extremely unlikely that projects of this type will ever require the units to be able to communicate amongst themselves. Hence a simple master-slave method of controlling link usage will result in the simplest software and most efficient usage of the link.

2) As the link will be required to carry communications traffic for a number of logical channels the control information overhead for each message should be kept to a minimum. This resulted in the choice of a two pass protocol, using variable length messages containing binary data sections.

3) In an effort to keep the system's knowledge of its own status as full as possible, a receiving station will always reply to a message. The reply would be the expected response if the message was received correctly and a negative acknowledgement including the relevant error code if it had been received incorrectly. Thus the only time no response would be expected was if the line was down or the received message was distorted so that the destination

could not be determined from the message contents. In this case the receiving system would not be able to tell that the message was intended for it.

F.1) DATA LINK MESSAGE LAYOUT.

NO	NAME	NO OF BYTES	CODES
1	SYNC	1	55 HEX
2	START	1	STX
3	SOURCE ADDRESS	1	ASCII @,A-Z
4	DESTINATION ADDRESS	1	"
5	CONTROL TYPE	1	ASCII (S,R,P,A,N)
6-9	SPARE	4	-
10	LENGTH	1	BIN
11	DATA	1	BIN
	.	.	
	.	. LENGTH	
	.	. BYTES	
	.	.	
254	DATA	1	BIN
255	CHECKSUM	1	BIN

NOTES:

A) Address @ will be used for the supervisory system ie the master for that link's communications. The other addresses will be numeric starting at one but biased by 40 hex, and so will be stored as A,B,C.. etc.

B) There will usually not be more than eight to ten slave units connected on a single multi-dropped link, and their addresses will be set alphabetically from A onwards.

C) The message type codes are intended for use as listed below. Further details of their usage in connection with the protocol will be given in the next section.

(S)END DATA MESSAGE
(R)EQUEST DATA MESSAGE.
(P)OLL FOR UNSOLICITED DATA MESSAGES.
(A)CKNOWLEDGE POSITIVE RECEIPT OF A MESSAGE.
(N)EGATIVE ACKNOWLEDGE OF RECEIPT OF A MESSAGE.

D) Four spare bytes have been left in the control portion of this message layout for future developments.

E) The length byte in the control section will indicate the number of bytes of binary data included in this message in the data section. The value will be stored in binary as a single byte so there will be an upper limit of 255 bytes of data.

F) The data section layout will depend entirely on the particular application. Normally the first byte will be a data message type code. This would have a project dependant meaning and would define the contents of the remainder of the data section.

G) The last byte of the message would be a checksum. It will be generated by taking the modulus 256 value of the negative summation of all bytes in the message bounded by, but not including the STX character and the checksum itself.

H) The entire message control packet and data from sync to checksum inclusive constitute what is transmitted over the data link, however the user's useful information comprises only the data section and thus this is sometimes known as the user message or data message.

F.2) MESSAGE PROTOCOL.

```
     MASTER                                    SLAVE
     ------                                    -----

S-TYPE MESSAGE --->  <---  A - TYPE  MESSAGE (ONE DATA BYTE, AN
                                               ECHO OF SENT DATA)

                                               OR

                     <--- N - TYPE MESSAGE (ONE DATA BYTE, THE
                                             ERROR CODE)


R-TYPE MESSAGE --->  <--- A - TYPE MESSAGE (ECHO OF REQUESTED DATA
                                            MESSAGE TYPE FOLLOWED
                                            BY A VARIABLE NUMBER
                                            OF DATA BYTES)

                                               OR

                     <--- N - TYPE MESSAGE (ONE DATA BYTE, THE
                                            ERROR CODE)

P-TYPE MESSAGE --->  <--- A - TYPE MESSAGE (EITHER A CODE {255} TO
                                            SHOW NO UNSOLICITED
                                            DATA MESSAGES ARE
                                            WAITING, OR THE OLDEST
                                            DATA MESSAGE WAITING
                                            TO BE TRANSMITTED)

                                               OR

                      <--- N - TYPE MESSAGE (ONE DATA BYTE, THE
                                             THE ERROR CODE)
```

APPENDIX G

1 - LINKER OUTPUT FILE
2 - DATABASE
3 - RATIO TASK
4 - WINDER TASK
5 - WINDER CONTROL TASK
6 - OCP TASK
7 - TRAVERSE DRIVE TASK
8 - TRAVERSE CONTROL TASK
9 - CHUCK TACHO TASK
10 - DATABASE UPDATE TASK
11 - COMMUNICATIONS TASK
12 - SANS INTERACTIVE STANDARDS PROCEDURES
13 - ICU COMMON PROCEDURES
14 - MSMTU1
15 - STARTUP CODE
16 - SYSTEM BUILD COMMAND FILE

```
:* MAGIC ** LONG ADDRESSING INTEL 8086 VERSION 1.0 S.P.L. 15-OCT-84    10:59

·,ROMLNK.MAP


.INKER M SYSTEM FILE FROM ?=[341,1]SMTROM.SAV

 M>SEG RATIOAR AREAS=A2,A3
 M>SEG WDRVAR AREAS=A2,A3
 M>SEG WCONTAR AREAS=A2,A3
 M>SEG SEQ AREAS=A2,A3
 M>SEG OCP1AR AREAS=A2,A3
 M>SEG TDRVAR AREAS=A2,A3
 M>SEG TCONTAR AREAS=A2,A3
 M>SEG WTACHAR AREAS=A2,A3
 M>SEG DBUPAR AREAS=A2,A3
 M>SEG COMMS AREAS=A2,A3
 M>TASK ID=1 SEG=RATIOAR,GOSMT ENTRY=RATS PRI=58 GO
 M>TASK ID=2 SEG=WDRVAR,GOSMT ENTRY=WDRIVE PRI=C0 GO
.M>TASK ID=3 SEG=WCONTAR,GOSMT ENTRY=WALGO PRI=60 GO
 M>TASK ID=4 SEG=SEQ,GOSMT ENTRY=SEQMON PRI=80 GO
.M>TASK ID=5 SEG=OCP1AR,GOSMT ENTRY=GENOCP STACK=180 PRI=54 GO
 M>TASK ID=6 SEG=TDRVAR,GOSMT ENTRY=TDRIVE PRI=C0 GO
 M>TASK ID=7 SEG=TCONTAR,GOSMT ENTRY=TALGO PRI=60 STACK=180 GO
 M>TASK ID=8 SEG=WTACHAR,GOSMT ENTRY=WINDTA PRI=57 GO
 M>TASK ID=9 SEG=DBUPAR,GOSMT ENTRY=UPDATE PRI=55 GO
 M>TASK ID=A SEG=COMMS,GOSMT ENTRY=SENDME STACK=100 PRI=56 GO
.M>MODS SEG=RATIOAR

(BJECT MODULE FILE NAME(S) :=RATIO

.M>MODS SEG=WDRVAR

(BJECT MODULE FILE NAME(S) :=WDRV

.M>MODS SEG=WCONTAR

(BJECT MODULE FILE NAME(S) :=WCONT,COMPROC

.M>MODS SEG=TDRVAR

(BJECT MODULE FILE NAME(S) :=TDRV

.M>MODS SEG=TCONTAR

)BJECT MODULE FILE NAME(S) :=TCONT,COMPROC

.M>MODS SEG=SEQ

)BJECT MODULE FILE NAME(S) :=SQMON,INVSQ

.M>MODS SEG=OCP1AR

)BJECT MODULE FILE NAME(S) :=OCP1

.M>MODS SEG=WTACHAR

)BJECT MODULE FILE NAME(S) :=WTACHO

.M>MODS SEG=DBUPAR
```

```
BJECT MODULE FILE NAME(S) :=DBUF

M>MODS SEG=COMMS

BJECT MODULE FILE NAME(S) :=LINKDA


INK OK


M>LINK SMT=GOSMT
M>DUMP ROM=A5
OADFILE NAME? >ROMSYS

ATA TEMPLATE OCCUPIES       7B2 BYTES

RXEQ TAKEN FROM SEGMENT: GOSMT




EGMENT MAP
==========


REA A2       FROM:      100   TO:     2000   LENGTH:     1F00
     BASE     USED     STACK( LENGTH )
      100     10CA         0 (     10CA) DATA      SEG GOSMT   (SHR)
     11D0        0       100 (      100) DATA      SEG RATIOA
     12D0        0       100 (      100) DATA      SEG WDRVAR
     13D0        0       100 (      100) DATA      SEG WCONTA
     14D0       1A       100 (      11A) DATA      SEG SEQ
     15F0      1B4       180 (      334) DATA      SEG OCP1AR
     1930        0       100 (      100) DATA      SEG TDRVAR
     1A30        0       180 (      180) DATA      SEG TCONTA
     1BB0        0       100 (      100) DATA      SEG WTACHA
     1CB0        0       100 (      100) DATA      SEG DBUFAR
     1DB0        0       100 (      100) DATA      SEG COMMS

REA A3       FROM:    F4000   TO:    FB820   LENGTH:     7820
     BASE     USED     STACK( LENGTH )
    F4000      8DA         0 (      8DA) PROCS     SEG RATIOA
    F48E0      286         0 (      286) PROCS     SEG WDRVAR
    F4B70      F18         0 (      F18) PROCS     SEG WCONTA
    F5A90      A2B         0 (      A2B) PROCS     SEG SEQ
    F64C0     2A57         0 (     2A57) PROCS     SEG OCP1AR
    F8F20      2D1         0 (      2D1) PROCS     SEG TDRVAR
    F9200     1294         0 (     1294) PROCS     SEG TCONTA
    FA4A0      31C         0 (      31C) PROCS     SEG WTACHA
    FA7C0      535         0 (      535) PROCS     SEG DBUFAR
    FAD00      9B9         0 (      9B9) PROCS     SEG COMMS

REA A5       FROM:    FB820   TO:    FC000   LENGTH:      7E0
     BASE     USED     STACK( LENGTH )

REA A4       FROM:    FC000   TO:    FFFF0   LENGTH:     3FF0
     BASE     USED     STACK( LENGTH )
```

```
FC000     3F96          0 (     3F96) PROCS     SEG GOSMT  (SHR)

REA A1       FROM:   FFFF0  TO:  100000  LENGTH:      10
   BASE       USED    STACK( LENGTH )
  FFFF0        6        0 (        6) ALL       SEG STRTUP

SYSTEM TOP=    FFFFF
```

ODULE MAP
=========

```
EGMENT GOSMT   FROM:          0 TO:    10CA IN AREA A2
----------------------------------------------------------------

    BASE       TOP    (LENGTH)
       0        22 (        22)    ROMCBA (DATA)
      22        22 (         0)    SMTDBA (DATA)
      22        22 (         0)    SMTB1  (DATA)
      22        44 (        22)    SMTB2  (DATA)
      44       8A4 (       860)    SMTB3  (DATA)
     8A4       8A4 (         0)    MSMTCT (DATA)
     8A4       8B0 (         C)    MSMTU1 (DATA)
     8B0       90C (        5C)    MSMTPI (DATA)
     90C       9A4 (        98)    LINKDR (DATA)
     9A4       A2E (        8A)    MSMTCL (DATA)
     A2E       A3C (         E)    SMTCLK (DATA)
     A3C       A3C (         0)    SMTERR (DATA)
     A3C       A3C (         0)    MSMTEV (DATA)
     A3C       A4E (        12)    MSMTFI (DATA)
     A50       A78 (        28)    RRFILC (DATA)
     A78      10C8 (       650)    DATA   (DATA)
    10C8      10CA (         2)    INVSQ  (DATA)
SEGMENT RATIOA FROM:          0 TO:     100 IN AREA A2
----------------------------------------------------------------

    BASE       TOP    (LENGTH)
       0         0 (         0)    RATIO  (DATA)

SEGMENT WDRVAR FROM:          0 TO:     100 IN AREA A2
----------------------------------------------------------------

    BASE       TOP    (LENGTH)
       0         0 (         0)    WDRV   (DATA)

SEGMENT WCONTA FROM:          0 TO:     100 IN AREA A2
----------------------------------------------------------------

    BASE       TOP    (LENGTH)
       0         0 (         0)    WCONT  (DATA)
       0         0 (         0)    COMPRO (DATA)

SEGMENT SEQ    FROM:          0 TO:     11A IN AREA A2
----------------------------------------------------------------

    BASE       TOP    (LENGTH)
```

```
      0         14 (        14)    SQMON  (DATA)
     14         1A (         6)    INVSQ  (DATA)

EGMENT OCP1AR FROM:          0 TO:      334 IN AREA A2
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0         1B4 (       1B4)    OCP1   (DATA)

EGMENT TDRVAR FROM:          0 TO:      100 IN AREA A2
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0           0 (         0)    TDRV   (DATA)

EGMENT TCONTA FROM:          0 TO:      180 IN AREA A2
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0           0 (         0)    TCONT  (DATA)
      0           0 (         0)    COMPRO (DATA)

EGMENT WTACHA FROM:          0 TO:      100 IN AREA A2
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0           0 (         0)    WTACHO (DATA)

EGMENT DBUPAR FROM:          0 TO:      100 IN AREA A2
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0           0 (         0)    DBUP   (DATA)

EGMENT COMMS  FROM:          0 TO:      100 IN AREA A2
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0           0 (         0)    LINKDA (DATA)

EGMENT RATIOA FROM:          0 TO:      8DA IN AREA A3
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0         8DA (       8DA)    RATIO  (PROCS)

EGMENT WDRVAR FROM:          0 TO:      286 IN AREA A3
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0         286 (       286)    WDRV   (PROCS)

EGMENT WCONTA FROM:          0 TO:      F18 IN AREA A3
-----------------------------------------------------

    BASE        TOP    (LENGTH)
      0         220 (       220)    WCONT  (PROCS)
    220         F18 (       CF8)    COMPRO (PROCS)

EGMENT SEQ    FROM:          0 TO:      A2B IN AREA A3
```

```
-------------------------------------------------------

    BASE        TOP   (LENGTH)
      0         5B3 (       5B3)   SQMON  (PROCS)
     5B4        A2B (       477)   INVSQ  (PROCS)

 EGMENT OCP1AR FROM:          0 TO:     2A57 IN AREA A3
-------------------------------------------------------

    BASE        TOP   (LENGTH)
      0        2A57 (      2A57)   OCP1   (PROCS)

;EGMENT TDRVAR FROM:          0 TO:      2D1 IN AREA A3
-------------------------------------------------------

    BASE        TOP   (LENGTH)
      0         2D1 (       2D1)   TDRV   (PROCS)

;EGMENT TCONTA FROM:          0 TO:     1294 IN AREA A3
-------------------------------------------------------

    BASE        TOP   (LENGTH)
      0         59C (       59C)   TCONT  (PROCS)
     59C       1294 (       CF8)   COMPRO (PROCS)

SEGMENT WTACHA FROM:          0 TO:      31C IN AREA A3
-------------------------------------------------------

    BASE        TOP   (LENGTH)
      0         31C (       31C)   WTACHO (PROCS)

SEGMENT DBUPAR FROM:          0 TO:      535 IN AREA A3
-------------------------------------------------------

    BASE        TOP   (LENGTH)
      0         535 (       535)   DBUP   (PROCS)

SEGMENT COMMS  FROM:          0 TO:      9B9 IN AREA A3
-------------------------------------------------------

    BASE        TOP   (LENGTH)
      0         9B9 (       9B9)   LINKDA (PROCS)

SEGMENT GOSMT  FROM:          0 TO:     3F96 IN AREA A4
-------------------------------------------------------

    BASE        TOP   (LENGTH)
      0         358 (       358)   ROMCBA (PROCS)
      0           0 (         0)   SMTDBA (PROCS)
     358        3F7 (        9F)   SMTB1  (PROCS)
     3F8       10EB (       CF3)   SMTB2  (PROCS)
    10EC       10EC (         0)   SMTB3  (PROCS)
    10EC       15A1 (       4B5)   MSMTCT (PROCS)
    15A2       176A (       1C8)   MSMTU1 (PROCS)
    176A       19AB (       241)   MSMTPI (PROCS)
    19AC       1B0D (       161)   LINKDR (PROCS)
    1B0E       2827 (       D19)   MSMTCL (PROCS)
    2828       2A78 (       250)   SMTCLK (PROCS)
    2A78       2C78 (       200)   SMTERR (PROCS)
    2C78       30FF (       487)   MSMTEV (PROCS)
```

```
    3100      3A2A (      92A)   MSMTFI (PROCS)
    3A30      3AD1 (       A1)   RRFILC (PROCS)
    3AD2      3AD2 (        0)   DATA   (PROCS)
    3AD2      3F96 (      4C4)   INVSQ  (PROCS)

SEGMENT STRTUP FROM:        0 TO:        6 IN AREA A1
------------------------------------------------------------

    BASE       TOP   (LENGTH)
       0         5 (        5)   STUPCD



UNSATISFIED REFERENCE LIST
==========================



ENTRY POINTS LIST
=================

SEGMENT GOSMT  FROM:        0 TO:     10CA IN AREA A2
------------------------------------------------------------

ENTRY POINTS IN MODULE: SMTDBA

RRUSG (SD)        6, RRERR (SD)       16, RRINFO(SD)       24, RRSIO (SD)         C
RRSED (SD)       14

ENTRY POINTS IN MODULE: SMTB2

PATTER( D)       22

ENTRY POINTS IN MODULE: SMTB3

REGDAT( D)      798, TIMEDA( D)      782, SSSOFF( D)      7BA, SYSTAC(ST)       2BA
SSPRI ( D)      842, FBSTK (ST)      116, SSSTAT( D)      854, SSEOFF( D)       7FE
ERRSTK(ST)      44A, CLKSTK(ST)      382, TRAPDA( D)      7AC, HSTK  (ST)        4E
TASKDA( D)      526, INSTK (ST)      18E

ENTRY POINTS IN MODULE: MSMTU1

TRAVST( D)      8AA, WINDST( D)      8A4

ENTRY POINTS IN MODULE: LINKDR

BUFFER( D)      90C
```

```
NTRY POINTS IN MODULE: RRFILC

ILSTK( ?)      A78


NTRY POINTS IN MODULE: DATA

ODDAT( D)     10AA, SEQVAR( D)      DE8, FLAGSI( D)      E14, AWAITU( D)      EA2
ECODE( D)      E2A, MAXRAT( D)      E7E, DPARAM( D)      E7A, RPARAM( D)      AD2
PARAM( D)      A78, FLAGSO( D)      E1E, TCONTT( D)      B5C, WCONTT( D)      CE2
ATCHD( D)      E6C

EGMENT RATIOA FROM:        0 TO:     100 IN AREA A2
------------------------------------------------------

EGMENT WDRVAR FROM:        0 TO:     100 IN AREA A2
------------------------------------------------------

EGMENT WCONTA FROM:        0 TO:     100 IN AREA A2
------------------------------------------------------

EGMENT SEQ    FROM:        0 TO:     11A IN AREA A2
------------------------------------------------------

EGMENT OCP1AR FROM:        0 TO:     334 IN AREA A2
------------------------------------------------------

EGMENT TDRVAR FROM:        0 TO:     100 IN AREA A2
------------------------------------------------------

EGMENT TCONTA FROM:        0 TO:     180 IN AREA A2
------------------------------------------------------

EGMENT WTACHA FROM:        0 TO:     100 IN AREA A2
------------------------------------------------------

EGMENT DBUPAR FROM:        0 TO:     100 IN AREA A2
------------------------------------------------------

EGMENT COMMS  FROM:        0 TO:     100 IN AREA A2
------------------------------------------------------

EGMENT RATIOA FROM:        0 TO:     8DA IN AREA A3
------------------------------------------------------


ENTRY POINTS IN MODULE: RATIO

RATS  ( P)       C

SEGMENT WDRVAR FROM:        0 TO:     286 IN AREA A3
------------------------------------------------------


ENTRY POINTS IN MODULE: WDRV

WDRIVE( P)       C

SEGMENT WCONTA FROM:        0 TO:     F18 IN AREA A3
------------------------------------------------------
```

```
ENTRY POINTS IN MODULE: WCONT

WALGO ( P)        C

ENTRY POINTS IN MODULE: COMPRO

TMOD  ( P)      654, UPRAMP( P)      32D, DOWNRA( P)      220, RUN   ( P)      43B
PARAM ( P)      472, BAND  ( P)      B8F

SEGMENT SEQ    FROM:        0 TO:     A2B IN AREA A3
------------------------------------------------------------

ENTRY POINTS IN MODULE: SQMON

SEQMON( P)       98

ENTRY POINTS IN MODULE: INVSQ

INVSEQ( P)      613

SEGMENT OCP1AR FROM:        0 TO:    2A57 IN AREA A3
------------------------------------------------------------

ENTRY POINTS IN MODULE: OCP1

GENOCP( P)      BD2

SEGMENT TDRVAR FROM:        0 TO:     2D1 IN AREA A3
------------------------------------------------------------

ENTRY POINTS IN MODULE: TDRV

TDRIVE( P)       10

SEGMENT TCONTA FROM:        0 TO:    1294 IN AREA A3
------------------------------------------------------------

ENTRY POINTS IN MODULE: TCONT

TALGO ( P)        C

ENTRY POINTS IN MODULE: COMPRO

TMOD  ( P)      9D0, UPRAMP( P)      6A9, DOWNRA( P)      59C, RUN   ( P)      7B7
PARAM ( P)      7EE, BAND  ( P)      F0B

SEGMENT WTACHA FROM:        0 TO:     31C IN AREA A3
------------------------------------------------------------

ENTRY POINTS IN MODULE: WTACHO
```

```
'INDTA( P)        C

EGMENT DBUPAR FROM:        0 TO:      535 IN AREA A3
------------------------------------------------------

NTRY POINTS IN MODULE: DBUP

(PDATE( P)        0

EGMENT COMMS  FROM:        0 TO:      9B9 IN AREA A3
------------------------------------------------------

:NTRY POINTS IN MODULE: LINKDA

'ENDME( P)       26, MYADDR( P)      7CF

EGMENT GOSMT  FROM:        0 TO:     3F96 IN AREA A4
------------------------------------------------------

:NTRY POINTS IN MODULE: ROMCBA

'TORTL( P)       43, RRXEQ ( P)      1FF, CLOCKI( P)        B, STKINI( P)      267
:RGEL ( P)       F2, RETFIN( P)       88, SVDINI( P)      2DC, CHANGE( P)       B1
R03   ( R)      1C3, HUNLOC( P)       AF, HLOCK ( P)       AD

:NTRY POINTS IN MODULE: SMTB1

RETEV ( P)      3C8, QEVRRE( P)      3AA, QEV   ( P)      358

:NTRY POINTS IN MODULE: SMTB2

STOP  ( P)      416, ME    ( P)      90A, FBPROC( P)     10DA, DEFOUT( P)     10D6
.OCK  ( P)      69F, GTIM  ( P)      D8C, TSECUR( P)     1086, RESET ( P)      64C
SECURE( P)      922, SYSSTO( P)      81F, SYSRTO( P)      84F, WAITFO( P)      674
DELAY ( P)      49E, DEFIN ( P)     10C5, TIMDAT( P)      E06, SETDAT( P)      EF1
WAIT  ( P)      5C8, UNLOCK( P)      6BA, TWAIT ( P)     104A, SET   ( P)      734
CLEANU( P)      D47, DFERP ( P)     10C2, START ( P)      45A, RELEAS( P)      A17
RRNUL ( P)     10D9, TSTSCR( P)     1002

ENTRY POINTS IN MODULE: MSMTCT

R01   ( R)     10EC, R02   ( R)     111E, R11   ( R)     12AE, R13   ( R)     12A3
R30   ( R)     1539, R15   ( R)     1347, R31   ( R)     1533, R08   ( R)     113F
R32   ( R)     1176, R40   ( R)     1211, R09   ( R)     1156, R41   ( R)     122F
R17   ( R)     1400, R25   ( R)     14CE, R34   ( R)     11B5, R42   ( R)     123C
R26   ( R)     14E9, R35   ( R)     11C1, R43   ( R)     121D, R19   ( R)     14AE
R27   ( R)     150F, R36   ( R)     11CE, R28   ( R)     150A, R37   ( R)     11E1
R29   ( R)     1505, R38   ( R)     11EF, R39   ( R)     1206

ENTRY POINTS IN MODULE: MSMTU1

USERIN( P)     15A2
```

```
ENTRY POINTS IN MODULE: MSMTPI

INTTY  ( P)    185C, RRIN   ( P)    1982, OUTTTY( P)    176A, RXINTE( P)    1893
SETNEC( P)     1996, RROUT  ( P)    1842

ENTRY POINTS IN MODULE: LINKDR

LINKRX( P)     19B7, SENDBU( P)     1AE8, LINKTX( ?)    1AA2, HOSTLI( P)    19AC

ENTRY POINTS IN MODULE: MSMTCL

INSTRU( P)     1C02

ENTRY POINTS IN MODULE: SMTCLK

CLOCK  ( P)    2828

ENTRY POINTS IN MODULE: SMTERR

ERPRIN( P)     2AB6

ENTRY POINTS IN MODULE: MSMTEV

SETFMQ( P)     2C78, RRMAG  ( P)    3094

ENTRY POINTS IN MODULE: MSMTFI

RWRT   ( P)    36F0, SPS    ( P)    3130, IWRTF  ( P)   31B0, TREAD  ( P)   33B8
RREAD  ( P)    3491, ZREAD  ( P)    327F, FREAD  ( P)   339A, IWRT   ( P)   3160
HWRT   ( P)    3212, NLS    ( P)    3100, HREAD  ( P)   326B, FWRT   ( P)   3394
IREAD  ( P)    3257, FWRTF  ( P)    3397, TWRT   ( P)   3453, RWRTF  ( P)   3714

ENTRY POINTS IN MODULE: RRFILC

RRFILL( ?)     3A34, RRDITP( ?)     3A30

ENTRY POINTS IN MODULE: INVSQ

IREPTO( P)     3B8C, SETBYT( P)     3E60, EXECOD( P)    3D0F, CHOICE( P)    3C2F
DIGOUT( P)     3EDF, VALBIT( P)     3EAB, DIGIN  ( P)   3F28, SETBIT( P)    3E15
INPORT( P)     3F7C, NOXPLN( P)     3B6E

SEGMENT STRTUP FROM:        0 TO:        6 IN AREA A1
-------------------------------------------------------
LM>MAP ENTS
LM>EXIT

LINKER RUN ENDS
```

```
TITLE DATA BASE FOR CONTROL PARAMETERES.
      CREATED 12-SEP-83 BY TEK. FILE : DATA.RTL
      LAST EDITED 2-OCT-84 TEK;

LET IDLE = 30700;
% ************************************************************************ %
% *                                                                      * %
% *                  CONTROL PARAMETER DATA BASE.                        * %
% *                        RAM BASED DATA                                * %
% *                                                                      * %
% ************************************************************************ %

ENT DATA TPARAMS;                        % TRAVERSE CONTROL PARAMETERS %
   ARRAY (NUMBAND) INT                   % SPEED BEFORE/DURING BANDING %
          TF1SPEED:=(13644(NUMBAND)),    % VALUES STORED ARE COUNTS %
          TF2SPEED:=(13644(NUMBAND));    % COUNT = (2.456E6 / OUTPUT FREQUENCY)%
        % DEFAULT VALUES : 30HZ %        % CONVERSELY %
                                         % OUTFREQ = (2.456E6 / COUNT)%


   REAL TMF1AMP:= 0.0,                   % MODULATION AMPLITUDE IN PERCENT%
        TMF2AMP:= 0.0;                   % MODULATION AMPLITUDE IN PERCENT%
   REAL TMF1PJ := 0.0,                   % P-JUMP AMPLITUDE IN PERCENT%
        TMF2PJ := 0.0;                   % MODULATION P-JUMP AMPLITUDE %
   INT TMPERIOD:= 2;                     % MODULATION PERIOD IN SECONDS %
   REAL TACC    :=0.5,                   % F1 TO F2 & F2 TO F1 ACCEL & DECEL %
        TDEC    :=0.5;                   % RATES IN HZ/100 MILLISECS  %
ENDDATA;


ENT DATA RPARAMS;                        % RATIO TASK CONTROL PARAMS %
   REAL WTACHO:=0.0,TTACHO:=0.0;         % TACHO SPEEDS IN RPM %
   ARRAY (NUMBAND) REAL
            RIBS1:=(0.0(NUMBAND)),       % RIBBON RATIO START/STOP VALUES %
            RIBS2:=(0.0(NUMBAND));       % HELD AS RATIOS %
   INT  CURBAND:=1,                      % CURRENT POINTER TO BANDING PARAMS  %
        MAXBAND:=INITMAXBAND,            % MAX NUMBER OF VALID BANDING POINTS %
        NOPI:=0;                         % CURRENT OPI NUMBER 0=INIT VALUES %
ENDDATA;


ENT DATA TCONTTABLE;                     % TRAVERSE CONTROL PARAMS %
   ARRAY (NUMTAB) INT TPITVAL:=(IDLE(NUMTAB));
                % ...LOOKUP TABLE INITIALISED WITH IDLING SPEED VALUES %
   ARRAY (NUMTAB) BYTE TSTAT:=(0(NUMTAB));
   INT TPOINT:=1;                        % TABLE POINTER %
ENDDATA;


ENT DATA WCONTTABLE;                     % WINDER CONTROL PARMS %
   ARRAY (NUMTAB) INT WPITVAL:=(IDLE(NUMTAB));
                % ...LOOKUP TABLE INITIALIZED WITH IDLING SPEED VALUES  %
   INT WPOINT:=1,                        % TABLE POINTER %
       WSPEED:=8187;                     % WINDER TARGET SPEED IN COUNTS %
ENDDATA;


ENT DATA SEQVARIABLES;                   % USED BY SEQUENCE TASK%
```

```
                                         %TEMPORARY WHILE TESTING%
   INT CURSEQ;
   LABEL SEQEXIT;
   ARRAY(1)REAL MAXSTEP:=(9.9);
   ARRAY(1)INT DSEQSTAT,                 %BIT 1=HELD%
               DTIMOUT,                  %SEQ TIME-OUT CNTERS%
               DCALLSEQ,                 %SET TO 1 FOR ENTRY AT NEXT SCAN%
               DSTEP,                    %SEQ MAIN STEP%
               DSS,                      %SEQ SUB-STEP%
               HOLDSTEP,HOLDSS;          %STEP 0 IF NO HOLD%
ENDDATA;

ENT DATA FLAGSIN;                        % FLAGS FROM SEQUENCE TASK %
                                         % ASSOCIATED WITH EVENT 2 %
      INT STOPF,HOLDF,STARTF,BANDF,SYNCF;
ENDDATA;

ENT DATA FLAGSOUT;                       % FLAGS FROM CONTROL TASKS %
      INT WRUNF,TF1F,TF2F,RAMPF,RUPF,RDOWNF;
ENDDATA;

ENT DATA SECODES;
   INT NOVALID:=20;
   ARRAY(20)INT PEOPLE:=(1,2,0(6),2204,0(11));
   ARRAY(20)BYTE SECCODE:=(40,20,0(6),255,0(11));
ENDDATA;

ENT DATA WATCHDOG;               % WATCHDOG TIMER LIMIT VALUES %
   ARRAY(6)INT WATCNT   :=(2,    % 1ST: TRAV DRIVE LIMIT 2 SECS %
                           2,    % 2ND: WINDER DRIVE LIMIT 2 SECS %
                           20,   % 3RD: TRAV CONTROL LIMIT 15 SECS %
                           20,   % 4TH: WINDER CONTROL LIMIT 15 SECS %
                           45,   % 5TH: TRAV TACHO LIMIT 35 SECS %
                           15);  % 6TH: WINDER TACHO LIMIT 10 SECS %
ENDDATA;

% ******************************************************************** %
% *                                                                  * %
% *                      ROM  BASED  DATA                            * %
% *                                                                  * %
% ******************************************************************** %

ENT DATA DPARAMS;                % DRIVER CONTROL PARAMS %
    INT WMAXF:=1931,                    % WINDER MAX FREQ CLAMP IN COUNTS %
        TMAXF:=1228;                    % TRAVERSE MAX FREQ CLAMP (2000HZ) %
ENDDATA;


ENT DATA MAXRATE;
  REAL MAXACC:=0.6,                      % WINDER MAX ACCEL & DECEL RATES %
       MAXDEC:=0.6,                      % IN HZ PER 100 MS. %
       STACC:=1.8,                       % TRAVERSE RAMP UP RATE AT START %
       STDEC:=1.8,                       % TRAVERSE RAMP DOWN RATE AT STOP %
       RUNACC:=4.2,                      % TRAVERSE ACCEL RATE AT RUN %
       RUNDEC:=4.2,                      % TRAVERSE DECEL RATE AT RUN %
       TMXACC:=1.8,                      % CURRENT TRAV RATES IN HZ PER 100MS %
       TMXDEC:=1.8;
   INT IDLESP:=IDLE,                     % START UP IDLING SPEED COUNT (13.3HZ)%
```

```
     STARTSYNCTIM:=2;                        % DELAY BEFORE RAMPING UP  %
ENDDATA;

ENT DATA AWAITUPDATE;
   ARRAY (NUMBAND) REAL AWTF1SPEED,                         % DATA AWAITING UPDATE %
                        AWTF2SPEED,                         % (HENCE PREFIX OF 'AW'%
                        AWRIBS1:=(1.0(NUMBAND)),              % FROM OCP INTO  MAIN  %
                        AWRIBS2:=(1.0(NUMBAND));              % DATABASE  %
   ARRAY (NUMBAND,4) INT BANDFLAG;
                                                            % TABLE OF FLAGS OF THOSE %
                                                            % VALUES TO BE UPDATED %
   REAL   AWTMF1AMP:=0.0,AWTMF2AMP:=0.0,                    % MOD AMPLITUDE  %
          AWTMF1PJ:=0.0,AWTMF2PJ:=0.0,                      % MOD P-JUMP     %
          AWTMPERIOD:=2.0,                                  % MOD PERIOD     %
          AWTACC:=0.3,AWTDEC:=0.3,                          % F1 TO F2 RATE OF CHANGE %
          AWWSPEED:=0.0;                                    % WINDER SPEED SETPOINT %
   ARRAY (9) INT MODFLAG;                                   % TABLE OF FLAGS FOR THE%
   INT    AWMAXBAND,                                        % MODULATION VALUES  %
          UPDATEFLAG:=0,                    % DATA BASE UPDATE NECESSARY FLAG %
          OPINUM;                           % OPI NUMBER OF NEXT OPI TO BE STORED %
ENDDATA;

% DATA MOVED FROM COMPROC NECESSARY FOR NEW ROM LINKAGE FORMAT %

ENT DATA MODDATA;
   INT TFXUPSTART,TFXUPSTOP;
   REAL TFXUPINC;
   INT TFXDOWNSTART,TFXDOWNSTOP;
   REAL TFXDOWNDEC;
   INT TRAMPTIME,THOLD:=1;
   REAL UPRATE,DOWNRATE;
   INT TEMPFLAG;            % TEMP FLAG TO ENSURE TF1 OR TF2 NOT SET TOO SOON %
ENDDATA;
```

```
'ITLE DATA BASE PRELUDE FILE FOR DATA BASE.
      CREATED 22-SEP-83 BY TEK. FILE : DATAPREL.RTL
      LAST EDITED 20-MAR-84 DRT;

.ET NUMBAND = 15;                 % NUMBER OF BANDING POINTS %
.ET INITMAXBAND = 1;              % NUMBER OF POINTS IN TABLE TO BE INITIAL%
.ET NUMTAB = 128;                 % SIZE OF DRIVE & CONTROL TASK TABLE %
.ET SPBAREV = 1;                  % SPACE BAR EVENT NUMBER %
.ET SEQEV = 2;                    % SEQUENCING EVENT NUMBER %
.ET TTACHOEV = 3;                 % TRAV TACHO EVENT %
.ET WTACHOEV = 4;                 % WINDER TACHO EVENT %
.ET DBUPDATE = 5;                 % UPDATE MASTER DATABASE %
% EVENTS 6 & 7 ARE USED BY THE DL COMMS CODE %
.ET CLOCKFREQ = 2.456E6;          % FREQUENCY OF CLOCK INPUT TO TIMERS %
.ET IT50HZVAL = 24576.0;          % SYSTEM CLOCK COUNTS PER 1/50 SECS %
.ET INTLMT = 300;                 % SYSTEM COUNT LIMIT FOR INTERRUPT CLASH CHECK %

% ********************************************************************** %
% *                                                                    * %
% *                  PRELUDE FILE FOR DATA BASE.                       * %
% *          (TO BE INPUT TO SJR BEFORE MAIN .RTL FILE)                * %
% *                                                                    * %
% ********************************************************************** %

EXT DATA TPARAMS;
   ARRAY (NUMBAND) INT TF1SPEED,TF2SPEED;
   REAL TMF1AMP,TMF2AMP;
   REAL TMF1PJ ,TMF2PJ ;
   INT TMPERIOD;
   REAL TACC,TDEC;
ENDDATA;

EXT DATA RPARAMS;
   REAL WTACHO,TTACHO;
   ARRAY (NUMBAND) REAL RIBS1,RIBS2;
   INT  CURBAND,MAXBAND,NOPI;
ENDDATA;

EXT DATA TCONTTABLE;
   ARRAY (NUMTAB) INT TPITVAL;
   ARRAY (NUMTAB) BYTE TSTAT;
   INT TPOINT;
ENDDATA;

EXT DATA WCONTTABLE;
   ARRAY (NUMTAB) INT WPITVAL;
   INT WPOINT,WSPEED;
ENDDATA;

EXT DATA SEQVARIABLES;
   INT CURSEQ;
   LABEL SEQEXIT;
   ARRAY(1)REAL MAXSTEP;
   ARRAY(1)INT DSEQSTAT,
               DTIMOUT,
               DCALLSEQ,
               DSTEP,
               DSS,
```

```
                HOLDSTEP,HOLDSS;
ENDDATA;

EXT DATA FLAGSIN;
        INT STOPF,HOLDF,STARTF,BANDF,SYNCF;
ENDDATA;

EXT DATA FLAGSOUT;
        INT WRUNF,TF1F,TF2F,RAMPF,RUPF,RDOWNF;
ENDDATA;

EXT DATA SECODES;
   INT NOVALID;
   ARRAY(20)INT PEOPLE;
   ARRAY(20)BYTE SECCODE;
ENDDATA;

EXT DATA WATCHDOG;
   ARRAY(6)INT WATCNT;
ENDDATA;

% ************************************************************************%
% *                                                                    *  %
% *                          ROM  BASED  DATA                           * %
% *                                                                     * %
% *********************************************************************** %

EXT DATA DPARAMS;
    INT WMAXF,
        TMAXF;
ENDDATA;

EXT DATA MAXRATE;
  REAL MAXACC,
       MAXDEC,
       STACC,
       STDEC,
       RUNACC,
       RUNDEC,
       TMXACC,
       TMXDEC;
  INT  IDLESP,STARTSYNCTIM;
ENDDATA;

% *********************************************************************** %
% *                                                                     * %
% *  TEMPORARY DATABASE CONTAINING OCP UPDATED VALUES AWAITING THE      * %
% *      END OF A DOFF-CYCLE BEFORE UPDATING THE MAIN DATABASE.         * %
% *         (FOR DESCRIPTIVE DETAILS REFER TO 'OCPDATABASE')            * %
% *                                                                     * %
% *********************************************************************** %

EXT DATA AWAITUPDATE;

   ARRAY(NUMBAND) REAL AWTF1SPEED,AWTF2SPEED,AWRIBS1,AWRIBS2;%BANDING DATA%
   ARRAY(NUMBAND,4) INT BANDFLAG;    % WHICH VALUES HAVE BEEN UPDATED %

   REAL  AWTMF1AMP,AWTMF2AMP,AWTMF1PJ,AWTMF2PJ,AWTMPERIOD,
```

```
      AWTACC,AWTDEC,AWWSPEED;     % MODULATION DATA %
 ARRAY(9) INT MODFLAG;    %WHICH MODULATION VALUES HAVE BEEN UPDATED%
 INT    AWMAXBAND,
        UPDATEFLAG,
        OPINUM;

ENDDATA;

EXT DATA MODDATA;
   INT TFXUPSTART,TFXUPSTOP;
   REAL TFXUPINC;
   INT TFXDOWNSTART,TFXDOWNSTOP;
   REAL TFXDOWNDEC;
   INT TRAMPTIME,THOLD;
   REAL UPRATE,DOWNRATE;
   INT TEMPFLAG;
ENDDATA;
```

```
TITLE PULSE TRAIN RATIO TASK (FOR BANDING AVOIDANCE).
      CREATED 15-AUG-83 BY TEK. FILE: RATIO.RTL
      LAST EDITED 11/4/84 DRT;

LET NL = 10;
LET ENQ=5;
OPTION (1) BS;
% ************************************************************************** %
% *                                                                        * %
% *                     PULSE TRAIN RATIO TASK                             * %
% *                                                                        * %
% ************************************************************************** %

SVC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;
SVC DATA RRSIO;PROC () BYTE IN;PROC (BYTE) OUT;ENDDATA;

EXT DATA TRAVSTOP;INT TSTOP,TENDFRAC,TCNT;ENDDATA;
EXT DATA TIMEDATA;
        INT NOW;                    % CYCLIC TICK COUNT %
        INT SECSNOW,MINSNOW;        % CYCLIC CLOCK COUNTS %
        INT NTICKS;                 % OUTSTANDING TICKS TO PROCESS %
        INT TCOUNT,SECS,MINS,HOURS,DAYS,MONTHS,YEARS;
ENDDATA;

EXT PROC () BYTE RRIN;
EXT PROC (BYTE) RROUT;
EXT PROC () INT IREAD;
EXT PROC (INT,INT) IWRTF;
EXT PROC (REAL) RWRT;
EXT PROC (REAL,INT,INT) RWRTF;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT) NLS;
EXT PROC (INT) DELAY;
EXT PROC (INT,INT,LABEL) TWAIT;
EXT PROC () CLEANUP;

% ************************************************************************* %
% *                                                                       * %
% *                    RTL/2 CODE FOR RATIO TASK.                         * %
% *                                                                       * %
% ************************************************************************* %

ENT PROC RATS();

   INT TDELAY,STBFLG,TCNTLAST,TNCYCCNT;
   REAL TCONST1,TCONST2,TTIM,IT50HZ:=IT50HZVAL,RATIO;
   INT LASTTSTART,LASTTGOFRAC,NEXTTSTART,NEXTTGOFRAC,MAXFLAG;


   IN:=RRIN;
   OUT:=RROUT;
   ERL := LOCERL;
   ERP := LOCERP;
   ERN := 0;

   % INITIALISATION FOR THIS MODULE %
   TCONST2 := (CLOCKFREQ / (TF1SPEED(1) * 3.0)) * 5.0 * RIBS1(1);
        % RATIO=WTACHO*6/TTACHO => WTACHO=RATIO*TTACHO/6 %
```

```
      % TTACHO=TRAVERSE TARGET HZ*60 SECS %
      % TRAVERSE TARGET HZ=CLOCKFREQ/6*TF1SPEED %
      % THUS WINDER TARGET RPM = RATIO*(CLOCKFREQ/6*TF1SPEED)*60/6 %
  NEXTTSTART:=NOW;NEXTTGOFRAC:=0;
  TCNTLAST:=TCNT:=20;                          % INITIALISE COUNT VALUE %
  TTACHOSTART();                           % START THE INTERRUPT CYCLE %
  CURBAND := 1;
  BANDF := 0;
  RAMPF:=0;
  MAXFLAG := 0;
  STBFLG := 0;
  RATIO := 0;
  TNCYCCNT:=(15-TMPERIOD):/TMPERIOD+1;         % FACTOR TO MAKE TDELAY +/- 15 S
  TDELAY := 50 * (TMPERIOD * TNCYCCNT + 5);
  TCONST1:=2.0 * TMPERIOD * CLOCKFREQ / 9.0;

RATIO1:

  WATCNT(5):=45;        % UPDATE WATCHDOG TIMER COUNT PERIOD %

  % CALCULATE COUNT VALUE BASED ON THE CURRENT TRAVERSE OUTPUT %
  % VALUE TO GIVE APPROX MODULATION PERIOD COUNT TIME %

  IF TPITVAL(TPOINT) >= IDLESP THEN
       TCNT:=200;
  ELSEIF WRUNF#0 AND TF1F LOR TF2F#0 THEN
     TCNT:=IF BANDF=0 THEN TF1SPEED(CURBAND) ELSE TF2SPEED(CURBAND) END;
     TCNT:=INT(TCONST1/REAL TCNT * TNCYCCNT)
  ELSE TCNT:=INT(TCONST1/REAL(TPITVAL(TPOINT)));
  END;

  % WAIT FOR EVENT WHICH IS SET BY INTERRUPT SERVICE ROUTINE WHEN %
  % COUNTER HAS COUNTED TCNT PULSES FROM THE TACHO %

  TWAIT(TTACHOEV,TDELAY,RATIO2);

  % CALCULATE TTACHO IN RPM %

  LASTTSTART:=NEXTTSTART;                  % SAVE OLD START TIME FOR %
  LASTTGOFRAC:=NEXTTGOFRAC;                % THIS CALCULATION %
  NEXTTSTART:=TSTOP;                       % SAVE NEW START TIME FOR %
  NEXTTGOFRAC:=TENDFRAC;                   % NEXT CALCULATION %

% CHECK FOR POSSIBLE INTERRUPT CLASH TIMING ERROR %
  IF LASTTGOFRAC+INTLMT>IT50HZ OR NEXTTGOFRAC+INTLMT>IT50HZ THEN
     TCNTLAST:=TCNT;
     GOTO RATIO1
   END;

  TTIM:=REAL(TSTOP-LASTTSTART)+REAL(LASTTGOFRAC-TENDFRAC)/IT50HZ;
  TTACHO:=2250.0/TTIM*REAL TCNTLAST;       % IN RPM. %
                                           % TCNT/2 REVS IN TTIM 1/50THS SEC %
                                           % *60 SECS * 3/2 GEAR RATIO => 2250.0 %

  TCNTLAST:=TCNT;

% IF END OF DOFF THEN RESET POINTERS AND RECALCULATE MOD PARAMS %
```

```
IF TTACHO <= 500.0 OR WTACHO <= 500.0 THEN     % IE IF END OF DOFF OR %
     CURBAND := 1;                              % MACHINE STOPPED %
     BANDF := 0;
     MAXFLAG := 0;
     STBFLG := 0;
     TCONST2 := (CLOCKFREQ / (TF1SPEED(1) * 3.0)) * 5.0 * RIBS1(1);
     TCONST1:=2.0 * TMPERIOD * CLOCKFREQ / 9.0;
             % 2 PULSES/REV FOR TMPERIOD SECS * 2/3 %
             % DIV BY 6*(INV OUTPUT) => 2*TMPERIOD*2/3*6 %
     TDELAY := 50 * (TMPERIOD * TNCYCCNT + 5);
     RATIO := 0.0;
     TNCYCCNT:=(15-TMPERIOD):/TMPERIOD+1;
     IF RAMPF = 0 THEN
        PARAM(TMF1AMP,TMF1PJ,TF1SPEED(1));    % CALCULATE NEW MOD PARAMS %
     END;
     GOTO RATIO1;
END;

RATIO := (WTACHO * 6)/ TTACHO;          % CALCULATES RATIO OF SPEEDS %

IF TF1F = 0 AND TF2F = 0 OR             % DON'T DO RATIO CALCULATION %
   BANDF = 1 AND TF1F = 1 OR            % UNLESS AT MODULATION SPEED %
   BANDF = 0 AND TF1F = 0 THEN
   GOTO RATIO1;
END;

IF WTACHO >= TCONST2 THEN STBFLG := 1;END;

IF CURBAND = 1 AND BANDF = 0 AND STBFLG = 0 AND WTACHO < TCONST2 THEN
     GOTO RATIO1;
END;

% COMPARE WINDER AND TRAVERSE TACHO SPEEDS, IF CRITICAL SET BAND FLAG %

IF BANDF = 0 THEN                       % NO BANDING IE AT FREQ F1 %
     IF RATIO <= RIBS1(CURBAND) THEN    % IF RIBBON POINT REACHED %
        IF MAXFLAG = 0 THEN             % LAST BANDING POINT? %
             BANDF := 1;                % SET BAND FLAG %
             GOTO RATIO1;
        ELSE
             GOTO RATIO1;
        END;
     ELSE GOTO RATIO1;                  % OTHERWISE CONTINUE %
     END;
ELSE                                    % BANDING IE AT FREQ F2 %
     IF RATIO <= RIBS2(CURBAND) THEN    % IF RIBBON POINT REACHED %
        BANDF := 0;                     % RESET THE BANDING FLAG %
        CURBAND := CURBAND + 1;         % INCREMENT POINTER %
        IF CURBAND > MAXBAND THEN       % IE LAST RIBBON POINT %
             CURBAND := MAXBAND;
             MAXFLAG := 1;
             GOTO RATIO1;
        ELSE
             GOTO RATIO1;
        END;
     ELSE
        GOTO RATIO1;
     END;
```

```
    END;

    GOTO RATIO1;

 RATIO2:
    TTACHO := 0.0;
    PARAM(TMF1AMP,TMF1PJ,TF1SPEED(1));     % CALL TO PARAM PUT HERE TO OVERCOME %
                                           % PARAM UPDATE PROBS WHEN TRAVERSE %
                                           % NOT RUNNING E.G. ON MINIRIG %
    GOTO RATIO1;

LOCERL:
    CLEANUP();
    GOTO RATIO1;
ENDPROC;

PROC LOCERP(INT N);
    TWRT('#NL#ERROR NO ');IWRTF(N,5);NLS(1);
ENDPROC;

% THIS PROC CONVERTS THE MODULATION PARAMETERS FROM THE DATA BASE INTO %
% COUNT VALUES FOR THE 8254 TIMER CHIP. THE PROC IS THE SAME AS THE ONE %
% IN MODULE COMPROC.RTL. IT IS INCLUDED HERE TO AVOID PROBLEMS WITH SHARED %
% AREAS IN THE SMT LINKER %

PROC PARAM(REF REAL TMFXAMP,TMFXPJ,REF INT TMFXSPEED);
    REAL K1;

    K1:=100.0*TMFXSPEED;            % CONSTANT USED REPEATEDLY %

    TRAMPTIME := 5 * TMPERIOD - THOLD;

    IF TMFXAMP <= 0.01 THEN
         TFXUPSTART:=TFXUPSTOP:=TFXDOWNSTART:=TFXDOWNSTOP:=TMFXSPEED;
         TFXUPINC:=TFXDOWNDEC:=0.0;
    ELSE
         TFXUPSTART := INT(K1/(100.0-TMFXAMP+TMFXPJ));
         TFXUPSTOP := INT(K1/(100.0+TMFXAMP));
         TFXUPINC := (REAL TFXUPSTOP - REAL TFXUPSTART)/REAL TRAMPTIME;

         TFXDOWNSTART := INT(K1/(100.0 + TMFXAMP - TMFXPJ));
         TFXDOWNSTOP := INT(K1/(100.0 - TMFXAMP));
         TFXDOWNDEC := (REAL TFXDOWNSTOP - REAL TFXDOWNSTART)/REAL TRAMPTIME;
    END;

ENDPROC;

% ************************************************************************ %
% *                                                                      * %
% *       8088 MACHINE CODE INSERTS FOR TRAVERSE TACHO TASK.             * %
% *                                                                      * %
% ************************************************************************ %

PROC TTACHOSTART();
    CODE 10,10;
ITXCTROXPORT    EQU      0B8H             ;SBX TIMER CHANNEL 0.(FOR TRAVERSE)
    ;START TRAVERSE COUNT DOWN TO INTERRUPT.
         MOV      AX,SEG *TRAVSTOP
```

```
        MOV     ES,AX
        MOV     AX,ES:*TCNT/TRAVSTOP      ;GET TRAVERSE COUNT VALUE.
        MOV     DX,ITXCTROXPORT
        OUT     DX,AL
        MOV     AL,AH
        OUT     DX,AL
        STI
   *RTL;
:NDPROC;
```

```
'ITLE WINDER DRIVE TASK.
      CREATED 13-SEP-83 BY TEK. FILE : WDRV.RTL;

.ET NL = 10;
.ET ENQ = 5;
.ET SETF = 1;
-)PTION (1) BS;
: *********************************************************************** %
: *                                                                     * %
: *                         WINDER DRIVE TASK.                          * %
: *                                                                     * %
: *********************************************************************** %

3VC DATA RRSIO;PROC () BYTE IN;PROC (BYTE) OUT;ENDDATA;
3VC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;

:XT PROC () BYTE RRIN;
:XT PROC (BYTE) RROUT;
:XT PROC (INT) DELAY,STOP;
:XT PROC () INT ME;
:XT PROC (REF ARRAY BYTE) TWRT;
:XT PROC (INT) IWRT,NLS;
:XT PROC (INT,INT) IWRTF;
:XT PROC () INT IREAD;
:XT PROC () CLEANUP;

:NT PROC WDRIVE();
   INT WOUT,DIFF,COUNT,WLAST,WSAFEDEC,WSAFEACC,FLAG;
   INT ACCERR:=0,DECERR:=0,CLAMPERR:=0;              % ERROR COUNTERS %
   INT T100MS:=5;

   IN:=RRIN;
   OUT:=RROUT;
   ERL  := LOCERL;
   ERP  := LOCERP;
   ERN  := 0;

   STOP(ME());                        % STARTED BY CONTROL TASK %
   WPOINT:=1;
   WLAST:=WPITVAL(WPOINT);

=WDRV1:

  WATCNT(2):=2;          % UPDATE WATCHDOG TIMER COUNT PERIOD %

  % CHECK TO SEE IF MAXIMUM DECELERATION OR ACCELERATION %
  % RATES ARE BEING EXCEEDED. IF THEY ARE,LIMIT THE RATE.%

   DIFF:=WLAST-WPITVAL(WPOINT);
   IF DIFF > 0 THEN                                  % ACCELERATING %
        WSAFEACC:=WLAST-INT((CLOCKFREQ*WLAST)/(CLOCKFREQ+WLAST*MAXACC));
        IF DIFF <= WSAFEACC THEN                     % IF ACCELERATION RATE SAFE %
           COUNT:=WPITVAL(WPOINT);                   % .....OUTPUT TABLE VALUE %
        ELSE
           COUNT:=WLAST-WSAFEACC;                    % IF NOT SAFE LIMIT IT %
           ACCERR:=ACCERR + 1;                       % INC ERROR COUNTER %
        END;
```

```
  ELSEIF DIFF < 0 THEN                                      % DECELERATING %
       WSAFEDEC:=INT((CLOCKFREQ*WLAST)/(CLOCKFREQ-WLAST*MAXDEC))-WLAST;
       IF ABS DIFF <= WSAFEDEC THEN                         % IF DECEL. RATE SAFE....%
          COUNT:=WPITVAL(WPOINT);                           % ....OUTPUT TABLE VALUE %
       ELSE
          COUNT:=WLAST+WSAFEDEC;                            % IF NOT SAFE LIMIT IT %
          DECERR:=DECERR + 1;                               % INC ERROR COUNTER %
       END;
  ELSEIF DIFF = 0 THEN                                      % NO CHANGE %
       COUNT:=WPITVAL(WPOINT);                              % OUTPUT TABLE VALUE %
  END;

 CHECK TO SEE IF FREQUENCY CLAMP VALUE IS BEING EXCEEDED, IF IT IS, LIMIT %
 THE VALUE TO THE CLAMP VALUE. %

  IF COUNT < WMAXF THEN                     % NOTE THAT COUNT PROP TO 1/F %
       COUNT:=WMAXF;
       CLAMPERR:=CLAMPERR + 1;
  END;

; WRITE THE VALUE TO THE PIT (INITIALISED IN MSMTU1.RTL) %

  CODE 10,10;
;TXCTR1XPORT     EQU      0BAH               ;SBX PIT CHANNEL 1.
       MOV      AX,SS:[BP+*COUNT]
       OUT      ITXCTR1XPORT,AL               ;OUTPUT LOW COUNT.
       MOV      AL,AH
       OUT      ITXCTR1XPORT,AL               ;OUTPUT HIGH COUNT.
*RTL;

% INCREMENT THE TABLE POINTER 'WPOINT' AFTER A 100MS DELAY %

       DELAY(T100MS);                       % WAIT FOR 100MS %
       WLAST:=COUNT;                        % HOLD ACTUAL CURRENT VALUE %
       WPOINT:=WPOINT LAND HEX 7F + 1;  % DRT'S SUPER FAST METHOD %

GOTO WDRV1;

LOCERL:
  CLEANUP();
  GOTO WDRV1;
ENDPROC;

PROC LOCERP(INT N);
  TWRT('#NL#ERROR NO ');IWRTF(N,5);NLS(1);
ENDPROC;
```

```
TITLE WINDER CONTROL TASK.
      CREATED 14-SEP-83 BY TEK. FILE : WCONT.RTL ;

_ET SETF=1;
LET RESETF=0;
_ET ENQ=5;
_ET NL=10;
% ********************************************************************** %
% *                                                                    * %
% *                       WINDER CONTROL TASK.                         * %
% *                                                                    * %
% ********************************************************************** %

SVC DATA RRSIO;PROC () BYTE IN;PROC (BYTE) OUT;ENDDATA;
SVC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;

EXT PROC () BYTE RRIN;
EXT PROC (BYTE) RROUT;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT,INT) IWRTF;
EXT PROC () INT IREAD;
EXT PROC (INT) STOP,NLS;
EXT PROC () INT ME;
EXT PROC (INT,INT,LABEL) TWAIT;
EXT PROC (REF ARRAY INT,INT,INT,REF REAL) DOWNRAMP;
EXT PROC (REF ARRAY INT,INT,INT,INT) RUN;
EXT PROC (REF ARRAY INT,INT,INT,INT,REF INT,REF REAL)UPRAMP;
EXT PROC () CLEANUP;
EXT PROC (INT) START;

EXT DATA TIMEDATA;
         INT NOW,SECSNOW,MINSNOW,NTICKS;
         INT TCOUNT,SECS,MINS,HOURS,DAYS,MONTHS,YEARS;
ENDDATA;

ENT PROC WALGO();
   INT WCALCP,WSTARTP,FLAG,BEGIN,DELAY;

   % INITIALISATION. %

   IN:=RRIN;
   OUT:=RROUT;
   ERL  := LOCERL;
   ERP  := LOCERP;
   ERN  := 0;

   % DYNAMIC INIT OF WCONTTABLE %

   FOR J := 1 BY 1 TO NUMTAB DO           % DYNAMIC INITIALISATION OF %
        WPITVAL(J):=IDLESP;               % CONTROL TABLE %
   REP;

   WPOINT:=1;

   % INITIALISE FLAGS %

   WRUNF:=RESETF;
   START(2);             % START WINDER DRIVE TASK WHEN ALL INIT DONE %
```

```
 % MAIN CONTROL LOOP %

AINLOOP:
   WATCNT(4):=20;          % UPDATE WATCHDOG TIMER COUNT %
   BEGIN:=NOW;
   WCALCP:=WPOINT;                        % SET UP TABLE POINTERS TO CALC .. %
   WSTARTP:=WPOINT;                       % .. NEW VALUES. %

   IF STOPF # 0 THEN                      % STOP SEQUENCE %
        DOWNRAMP(WPITVAL,WCALCP,WSTARTP,MAXDEC);
   ELSEIF SYNCF # 0 THEN                  % START SYNC SEQUENCE %
        RUN(WPITVAL,WCALCP,WSTARTP,IDLESP);
   ELSEIF STARTF # 0 THEN                 % START SEQUENCE %
        UPRAMP(WPITVAL,WCALCP,WSTARTP,WSPEED,WRUNF,MAXACC);
   ELSEIF WRUNF # 0 THEN                  % NORMAL RUN SEQUENCE %
        RUN(WPITVAL,WCALCP,WSTARTP,WSPEED);
   END;

   DELAY:=550-NOW+BEGIN;                  % ENSURES LOOP TIME IS EXACTLY %
                                          % 11 SECONDS %
   TWAIT(SEQEV,DELAY,MAINLOOP);           % WAIT FOR 11 SECS OR EVENT 2 %

   GOTO MAINLOOP;

.OCERL:
   CLEANUP();
   GOTO MAINLOOP;
ENDPROC;

PROC LOCERP(INT N);
   TWRT('#NL#ERROR NO ');IWRTF(N,5);NLS(1);
ENDPROC;
```

```
TITLE
  --- INVSQ.RTL --- LOGIC SEQUENC MODULE FOR INVERTER CONTROL SYSTEM
  WRITTEN 8/9/83;

 LAST EDITED 29-JUN-84 DRT %

 SEQUENCE PROCEEDURE TO PERFORM LOGIC CHECKS AND SEQUENCING OF INVERTER %
 CONTROL TASKS. RUN UNDER THE CONTROL OF THE SEQUENCE MONITOR %

=*****************************************************************************%
 A SEQUENCE SHOULD NOT DO ANY DIRECT OUTPUT OR PERFORM EXCESSIVE AMOUNTS    %
 OF CODE (THIS INCLUDES DELAYS AND WAITING FOR EVENTS OR FACILITIES)        %
*****************************************************************************%

=PTION (1) BC;
ET SEQEV=2;
ET NSEQ=1;
ET NL=10;
ET SP=32;
ET ON=1;
ET OFF=0;
ET READY=6;       %POSITION START I.E READY=1 STOP=0%
.ET RUNLIGHT=1;   %POSITION READY I.E UP TO SPEED%
.ET WONRELAY=2;   %WINDER ON RELAY%
.ET TONRELAY=3;   %TRAVERSE ON RELAY%

:XT PROC(INT)SET;
:XT PROC(INT)IWRT;
:XT PROC(REF ARRAY BYTE)TWRT;
%EXT PROC(REF ARRAY BYTE,REAL,REAL,PROC(INT))REAL RREPTO;%
:XT PROC(INT,REF INT,INT)SETBIT;
:XT PROC(INT,REF BYTE,INT)SETBYT;
:XT PROC(INT,INT)INT VALBIT;
:XT PROC(BYTE,BYTE)DIGOUT;
:XT PROC(BYTE)BYTE DIGIN;

SVC DATA RRSIO;PROC()BYTE IN;PROC(BYTE)OUT;ENDDATA;

DATA LOCSEQDAT;
   INT SEQNO:=1,SST,ST:=1;
ENDDATA;

%PROC STEPOUT(INT ST,SST);%
%   OUT(SP);%
%   IWRT(ST);%
%   OUT('.');%
%   IWRT(SST);%
%   OUT(SP);%
%ENDPROC; %

PROC SETFLAG(REF INT FLAG);
   CLEARFLAGS();
   VAL FLAG:=1;
   SET(SEQEV);
ENDPROC;

PROC CLEARFLAGS();
    STARTF:=0;HOLDF:=0;STOPF:=0;BANDF:=0;SYNCF:=0;
```

```
ENDPROC;

 PROC CHECKHOLD();%
    IF DSTEP(SEQNO)=HOLDSTEP(SEQNO) AND DSS(SEQNO)=HOLDSS(SEQNO)%
       THEN IF DSEQSTAT(SEQNO)LAND 1=0 THEN  NOT HELD %
          DSEQSTAT(SEQNO):=DSEQSTAT(SEQNO) LOR 1;%
          TWRT(" HELD AT");%
          STEPOUT(DSTEP(SEQNO),DSS(SEQNO));%
          OUT(NL);%
       END;%
       GOTO SEQEXIT;%
    END;%
 ENDPROC;%

 PROC INSTEP (REF INT ST,SST) REAL;%
    REAL R;%
    R:=RREPTO("STEP",0.0,MAXSTEP(SEQNO),EXPLN);%
    VAL ST:=INT(R-0.5);   STEP%
    VAL SST:=INT((R-REAL ST)*10.0);%
    IF SST=0 THEN VAL SST:=1 END;%
    RETURN(R);%
 ENDPROC;%

 PROC EXPLN(INT X);%
    TWRT("#NL#NOT VALID");%
 ENDPROC;%

 INT PROC INVSEQ (REF INT SEQSTAT
  SEQALARM, STEP, SS, STIME, CALLTIME, TIMEOUT);
   SEQEXIT:=EXIT;
   IF SEQALARM LAND 1#0 THEN VAL STEP:=3;VAL SS:=1 END;
   SWITCH STEP OF W1,W2,W3,W4;
   RETURN;

W1:                 % SYNC AND START SEQUENCE %
BLOCK;
   IF DIGIN(READY)=0 THEN VAL STEP:=3;VAL SS:=1;RETURN;END;
   IF WTACHO<1500.0 THEN SET(DBUPDATE);END;%EVENT OCPTASK AND RATIO TASK%
   SWITCH SS OF S1, S2, S3, S4, S5, S6, S7, S8, S9;
   RETURN;
S1:
 % CHECKHOLD(); %
   CLEARFLAGS();
   DIGOUT(RUNLIGHT,OFF);DIGOUT(WONRELAY,OFF);DIGOUT(TONRELAY,OFF);
   VAL SS:=2;
S2:
S3:
 % CHECKHOLD(); %
   IF DIGIN(READY)=1 THEN
   SETFLAG(SYNCF);
   VAL SS:=4;
   END;
   RETURN;
S4:
S5:
 % CHECKHOLD(); %
   IF  WPITVAL(WPOINT) <=IDLESP AND TPITVAL(TPOINT)=IDLESP THEN
       %IS WINDER AND TRAV INV AT IDLE SPEED%
```

```
      %NOTE COUNT INVERSE FREQ HENCE < %
     DIGOUT(WONRELAY,1);DIGOUT(TONRELAY,1);VAL SS:=6;
   END;
   RETURN;
S6:
S7:
 % CHECKHOLD(); %
   VAL CALLTIME:=STARTSYNCTIM; %WAIT 15 SECS%
   VAL SS:=8;
   RETURN;
S8:
 % CHECKHOLD(); %
   SETFLAG(STARTF);
   VAL SS:=9;
S9:
 % CHECKHOLD(); %
   IF WPITVAL(WPOINT) <= WSPEED AND TPITVAL(TPOINT) <= TF1SPEED(1) THEN
        % WINDER AND TRAVERSE INVERTERS UP TO SPEED? %
      CLEARFLAGS();DIGOUT(RUNLIGHT,1);
      VAL STEP:=2;VAL SS:=1;
   END;
   RETURN;
ENDBLOCK;

W2:                     % NORMAL RUN SEQUENCE %
BLOCK;
   IF DIGIN(READY)=0 THEN VAL STEP:=3;VAL SS:=1;RETURN;END;
   IF WTACHO<1500.0 THEN SET(DBUPDATE);END;%USED IN OCPTASK AND RATIO TASK%
   SWITCH SS OF S1, S2, S3, S4, S5, S6, S7, S8, S9;
   RETURN;
S1:
   RETURN;
S2:
S3:
S4:
S5:
S6:
S7:
S8:
S9:
   RETURN;
ENDBLOCK;

W3:                     % STOP SEQUENCE %
   BLOCK;
   IF WTACHO < 1500.0 THEN SET(DBUPDATE);END;
   SWITCH SS OF S1, S2, S3, S4, S5, S6, S7, S8, S9;
   RETURN;
S1:
 % CHECKHOLD(); %
   DIGOUT(WONRELAY,OFF);DIGOUT(TONRELAY,OFF);
   VAL SS:=2;
S2:
 % CHECKHOLD(); %
   SETFLAG(STOPF);
   TF1F:=0;TF2F:=0;WRUNF:=0;              % CLEAR RUN FLAGS %
   RAMPF:=0;                              % ADDED 15/10/84 TEK %
   VAL SS:=3;
```

```
 3:
 % CHECKHOLD(); %
   DIGOUT(RUNLIGHT,OFF);
   VAL SS:=4;
 4:
  % CHECKHOLD(); %
   IF DIGIN(READY)=1
      AND WPITVAL(WPOINT)>=IDLESP AND TPITVAL(TPOINT)>=IDLESP THEN
      %NOTE COUNTS INVERSE OF SPEED HENCE > %
      VAL STEP:=1; VAL SS:=1; %GO TO START SEQ%
   END;
   RETURN;
 5:
 % CHECKHOLD(); %
 6:
 % CHECKHOLD(); %
 7:
 % CHECKHOLD(); %
 8:
 % CHECKHOLD(); %
 9:
 % CHECKHOLD(); %
   RETURN;
:NDBLOCK;

=4:
-LOCK;
   SWITCH SS OF S1, S2, S3, S4, S5, S6, S7, S8, S9;
   RETURN;
;1:
 % CHECKHOLD(); %
;2:
 % CHECKHOLD(); %
;3:
 % CHECKHOLD(); %
;4:
 % CHECKHOLD(); %
S5:
 % CHECKHOLD(); %
S6:
 % CHECKHOLD(); %
S7:
 % CHECKHOLD(); %
S8:
 % CHECKHOLD(); %
S9:
 % CHECKHOLD(); %
   RETURN;
ENDBLOCK;

EXIT:

ENDPROC;
```

```
ITLE ---  SQMON.RTL --- SEQUENCE CONTROL MONITOR
AKEN FROM SANS T30 SEQUENCE MONITOR ON CONMAC 8/9/83;

 LAST EDITED 29-JUN-84 DRT %

 ALLOWS 1) A SEQUENCE TO BE EXECUTED AT REGULAR INTERVALS ADJUSTABLE BY THE %
 SEQUENCE.2) THE ENTRY POINT FOR EACH EXECUTION TO BE SET BY THE SEQUENCE %
 ITSELF OR VIA A CONTROL OCP.3) AN EVENT'S DURATION TO BE ACCUMULATED.4) AN %
 EVENT TO  HAVE A TIMEOUT SET FOR IT.5) AND IT ALLOWS THE SEQUENCE TO BE  %
 STOPPED, HELD WHERE IT IS, HAVE A HOLD SET FOR SOME FUTURE STEP, OR AS %
 STATED IN SECTION '2' BE SET TO A DIFFERENT SEQUENCE STEP. THE SECTION %
 '5' OPTIONS  ARE MAINLY FOR DEBUGGING PURPOSES %

PTION (1) BC;

ET TIMEINC = 1;
ET SEQHOLD = 1;
ET NSEQ=1;
ET MAXINT = 32767;
ET MININT = -32767;
ET BELL=7;LET NL=10;LET SP=32;
ET OFF=0;LET CPURUN=4;

EXT PROC (INT,INT) IWRTF;
EXT PROC () BYTE RRIN;
EXT PROC (BYTE) RROUT;
EXT PROC (INT) DELAY, IWRT,NLS;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC ()CLEANUP;
EXT PROC (REF INT,REF INT,REF INT,REF INT,REF INT,REF INT,REF INT) INVSEQ;
EXT PROC (BYTE,BYTE) DIGOUT;

EXT DATA TIMEDATA;
   INT NOW, SN, MN, NTICKS,  TC, SEC, MIN, HOUR, DAY, MONTH, YEAR;
ENDDATA;

SVC DATA RRSIO;
   PROC () BYTE IN; PROC (BYTE) OUT; ENDDATA;

SVC DATA RRERR;
   LABEL ERL; INT ERN; PROC (INT) ERP;
ENDDATA;

DATA LOCALSEQVARIABLES;
   INT DELTIME;
   ARRAY(NSEQ)  PROC (REF INT,REF INT,REF INT,REF INT,REF INT
    ,REF INT,REF INT) SEQUENCE:=(INVSEQ);
   ARRAY(NSEQ) INT DSEQALARM,
                   DSTIME,
                   DCALLTIME;
ENDDATA;

ENT PROC SEQMON ();
   INT NEXTIME,FLASH;
   BYTE LED;

   ERL:=ERLAB;
   ERP:=LOCERP;
```

```
 FOR S:=1 TO NSEQ DO    % INC SEQ TIMERS %

        % INCREMENT SEQUENCE INTERVAL TIMERS (FOR LEARNING AV TIMES) %
        IF DSTIME(S)<MAXINT THEN DSTIME(S):=DSTIME(S)+TIMEINC END;

        % DECREMENT SEQUENCE RECALL TIMERS & SET FLAGS TO CALL SEQUENCES %
        IF DCALLTIME(S)>MININT THEN DCALLTIME(S):=DCALLTIME(S)-TIMEINC END;
        IF DCALLTIME(S)<TIMEINC THEN DCALLSEQ(S):=1 END;

        % DECREMENT SEQUENCE TIMEOUT COUNTERS & GIVE TIMEOUT ALARMS %
        IF DTIMOUT(S)>MININT THEN DTIMOUT(S):=DTIMOUT(S)-TIMEINC END;
        IF ABS DTIMOUT(S)<TIMEINC THEN TWRT (" DELAY IN STEP ");%
           IWRT(DSTEP(S)); OUT('.'); IWRT(DSS(S)); OUT(NL);%
        END;%
   REP;

   FOR S:=1 TO NSEQ DO
       IF DCALLSEQ(S)=1 THEN DCALLSEQ(S):=0;
          IF DSEQSTAT(S) LAND HEX 1=1 THEN GOTO SKIPSEQ END;    % SEQ "HELD" %
          CURSEQ:=S;
          SEQUENCE(S)(DSEQSTAT(S),DSEQALARM(S),DSTEP(S),DSS(S),DSTIME(S)
          ,DCALLTIME(S),DTIMOUT(S));
      END;
SKIPSEQ:
   REP;

DELAB:
   DELAY(15);     % TO ALLOW LOW PRIO TASKS IN %
   NEXTIME:=NEXTIME+TIMEINC*50;
   DELTIME:=NEXTIME-NOW;
   DELAY(DELTIME);
   GOTO ONESEC;

ERLAB:
    CLEANUP();
    TWRT(" ERROR "); IWRT(ERN);OUT(NL);
    GOTO DELAB;
ENDPROC;

PROC LOCERR(INT N);
    TWRT("#NL#ERROR NO ");IWRT(N);NLS(1);
ENDPROC;

PROC ERRPRN(INT I);                % WATCHDOG ERROR PRINTING TASK %
    NLS(2);
    TWRT("#BELL#WATCHDOG ERROR IN ");
    SWITCH I OF TDRV,WDRV,TCONT,WCONT,RATIO,WTACHO;
    GOTO NONE;
TDRV:     TWRT("TRAVERSE DRIVE ");GOTO FINISH;
WDRV:     TWRT("WINDER DRIVE ");GOTO FINISH;
TCONT:    TWRT("TRAVERSE CONTROL ");GOTO FINISH;
WCONT:    TWRT("WINDER CONTROL ");GOTO FINISH;
RATIO:    TWRT("RATIO ");GOTO FINISH;
WTACHO:   TWRT("WINDER TACHO ");GOTO FINISH;
NONE:     TWRT("NO ");
FINISH:
    TWRT("TASK");
    NLS(1);
```

NDPROC;

```
▬ITLE OCP MODULE FOR T18 INVERTER CONTROL PROJECT
ΞG 03/11/83
ᴧST EDITED 2-OCT-84 TEK;

 ***************************************************************************** %
 *                                                                           * %
 *      THIS MODULE CONSISTS OF FIVE SECTIONS                                * %
 *      1.   PRELUDE AND LOCAL DATA - DATAPREL                               * %
 *                                  - LET,EXT PROCS                          * %
 *                                  - DATA OCPDATABASE                       * %
 *      2.   ENTRY AND BASIC MENU DECISIONS - PROC GENOCP                    * %
 *      3.   OPERATOR INPUT AND UPDATE - PROC COLLECTCHANGE                  * %
 *                                     - PROC CHANGEDATABASE                 * %
 *      4.   DATA TRANSLATION AND SCREEN DISPLAY - PROC FIXEDPARAMDISPLAY* %
 *                                               - PROC READDATA             * %
 *      5.   SERVICE PROCEDURES - PROCS MYIN,MYRREPTO,LERP,OPTIONS,          * %
 *                              - LOCERP,SCREENHEAD,CLEARSCREEN,             * %
 *                              - CLEARFLAGS,UNIT,XPLN1,                     * %
 *                              - XPLN2,XPLN4,XPLN5                          * %
 *                                                                           * %
 *****************************************************************************V%

.ET NBTIMES4 = 60; % NUMBAND * 4 %
.ET INPUT = 1;            %INPUT STREAM NUMBER  %
.ET OUTPUT = 1;           %OUTPUT STREAM NUMBER %
.ET NL = 10;
.ET SPACE = 18;
.ET ENQ=5;
.ET NOPTS=3;
.ET OPENBRACKET=91;
.ET ESC=27;
.ET FIELD=8;                   % FIELD SIZE OF FORMATTED OUTPUT%
.ET EOM=3;
.ET SECLEVEL=40;

SVC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;
SVC DATA RRSIO;PROC ()BYTE IN;PROC (BYTE) OUT;ENDDATA;
SVC DATA RRSED;BYTE TERMCH,IOFLAG;ENDDATA;

EXT PROC (INT) DELAY;
EXT PROC (REF ARRAY BYTE,REF ARRAY BYTE,PROC(INT)) INT CHOICE;
EXT PROC (REF ARRAY BYTE,INT,INT,PROC(INT)) INT IREPTO;
EXT PROC () BYTE RRIN;
EXT PROC (BYTE) RROUT;
EXT PROC ()CLEANUP;
EXT PROC () INT IREAD;
EXT PROC (INT) IWRT;
EXT PROC (INT) INT EXECODE;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT) SPS;
EXT PROC (INT) NLS;
EXT PROC (INT) STOP;
EXT PROC ()INT ME;
EXT PROC(INT)TIMDAT;
EXT PROC()SETDAT;
EXT PROC(INT)WAIT,RESET,SET;
EXT PROC () REAL RREAD;
EXT PROC (REAL) RWRT;
```

```
EXT PROC (REAL,INT,INT) RWRTF;
EXT PROC (INT,INT) IWRTF;

OPTION (1) BC;
% ****************************************************************  %
% *                                                              *  %
% *                 "DATABASE"  USED BY OCP TASK                 *  %
% *  COMPARISON BETWEEN "REAL" DATABASE AND THIS ONE IS SHOWN BELOW  *  %
% *                                                              *  %
% ****************************************************************  %

DATA OCPDATABASE;
                                       % DATA BASE CONTENT   -    DISPLAY      %
                                       % -----------------        -------      %
   ARRAY(NUMBAND) REAL DISPTF1SPEED,   %       COUNTS            M/C  HZ       %
                       DISPTF2SPEED,   %       COUNTS            M/C  HZ       %
                       DISPRIBS1,      %       RATIO             RATIO         %
                       DISPRIBS2;      %       RATIO             RATIO         %
   ARRAY(NBTIMES4) BYTE DISPBANDFLAG;
   ARRAY(9) INT DISPMODFLAG;
   INT I,FLAGO,DISPMAXBAND;
   REAL
        DISPTMF1AMP,DISPTMF2AMP,       %       PERCENT            PERCENT      %
        DISPTMF1PJ,DISPTMF2PJ,         %       PERCENT            PERCENT      %
        DISPTMPERIOD,                  %       SEC                SECONDS      %
        DISPTACC,DISPTDEC,             %       HZ/100MSEC         *10.0/6.0    %
        DISPWFREQ,DISPTFREQ,           %       COUNTS             M/C HZ       %
%       WTACHO,TTACHO                          RPM                RPM          %
%       MAXACC,MAXDEC                          HZ/100MSEC         *10.0/6.0    %
        DISPWSPEED,                     %      COUNTS             M/C HZ       %
        DISPWMAXF,                      %      COUNTS             M/C HZ       %
        DISPTMAXF,                      %      COUNTS             M/C HZ       %
        DISPIDLESP,                     %      COUNTS             M/C HZ       %
        DISPRRATIO,                     %  RATIO & SIZE CALCULATED FROM CAKE %
        DISPCAKE;                       %  AND TRAVERSE SPEEDS.              %
ENDDATA;

DATA OCPDATA;
   ARRAY(NOPTS)REF ARRAY BYTE OPTAGS:=(
   "-DATE AND TIME",
   "-DISPLAY FIXED PARAMETERS AND SPEEDS",
   "-DISPLAY OR CHANGE TRAVERSE PARAMETERS");
   ARRAY(5) REF ARRAY BYTE UNITLIST:=(
   " HZ",
   " HZ/SEC",
   " SECS",
   " RPM",
   " MM");
ENDDATA;

% ****************************************************************** %
% *                                                                * %
% *   SECTION 2:             OCP1 TASK                             * %
% *                                                                * %
% ****************************************************************** %

ENT PROC GENOCP();              %ENTRY POINT FOR OCP TASK%
   INT K,S,A,F:=0;
```

```
 BYTE B;
 REAL WINDSP,ZWSPEED;
 IN:=MYIN;
 OUT:=RROUT;
 ERL:=LOCERL;
 ERP:=LOCERP;
 UPDATEFLAG:=0;
 ERN:=0;
'ARTLAB:
 TWRT("#NL#OCP CLEARED#NL#");
 WAIT(SPBAREV);
 SCREENHEAD();
 CLEARFLAGS();
 TWRT('OCP OPTIONS:-');
 FOR K:=1 TO NOPTS DO
     OUT(NL);
     IWRT(K);
     TWRT(OPTAGS(K));
 REP;
 K:=OPTIONS(NOPTS);
 ERL:=STARTLAB;
 SWITCH K OF DATETIME,DISPFIXPARAM,DISPTPARAM;
 GOTO STARTLAB;
ATETIME:
 SCREENHEAD();
 OUT(NL);TIMDAT(-1);SPS(2);TWRT('CHANGE?#ENQ#');
 B:=IN();
 IF B='Y' THEN SETDAT() END;
 OUT(NL);
 GOTO STARTLAB;
ISPFIXPARAM:
    SCREENHEAD();
    SPS(9);TWRT(OPTAGS(K));SPS(5);TIMDAT(-1);OUT(NL);
      FIXEDPARAMDISPLAY();
    OUT(NL);
    SWITCH CHOICE('DISPLAY AGAIN? ','YN',XPLN1) OF DISPFIXPARAM,STARTLAB;
  GOTO STARTLAB;
ISPTPARAM:
 IF UPDATEFLAG = 1 THEN
     READDATA(1);GOTO REDISP;
  ELSEIF UPDATEFLAG = 0 THEN
     READDATA(0);GOTO REDISP;
  ELSE UPDATEFLAG:=0;GOTO ERL;
 END;
REDISP:
    SCREENHEAD();
    SPS(4);TWRT(OPTAGS(K));SPS(3);TIMDAT(-1);
    TWRT(IF UPDATEFLAG=0 THEN '  ACTIVE' ELSE '  FUTURE' END);
    TWRT(' PARM'S#NL#');
    READDATA(2);
    F:=0;
    SWITCH CHOICE('REDISPLAY/BANDING CHANGE/MOD & SPEED CHANGE/EXIT',
                 'RBME',XPLN4)
              OF REDISP,CHANGEBAND,CHANGEMOD,STARTLAB;
    GOTO STARTLAB;

CHANGEBAND:
 F:= COLLECTCHANGE(3);
```

```
   IF F # 1 THEN GOTO REDISP; END;
   READDATA(2);
   IF EXECODE(SECLEVEL) # 0 THEN
      CHANGEDATABASE(3);
      CLEARFLAGS();
   ELSE
      DELAY(25);
   END;
   GOTO DISPTPARAM;
 HANGEMOD:
   F:= COLLECTCHANGE(2);
   IF F # 1 THEN  GOTO REDISP; END;
   READDATA(2);
   IF EXECODE(SECLEVEL) # 0 THEN
      CHANGEDATABASE(2);
      CLEARFLAGS();
   ELSE
      DELAY(25);
   END;
   GOTO DISPTPARAM;

 .OCERL:
   CLEANUP();
   IF ERN#0 THEN
      TWRT("ERROR NO ");
      IWRT(ERN);
      ERN:=0;
      OUT(NL);
   END;
   GOTO STARTLAB;
ENDPROC;

% ********************************************************************** %
% *                                                                    * %
% *                    SECTION 3:                                      * %
% *                                                                    * %
% ********************************************************************** %

PROC COLLECTCHANGE(INT X) INT;
%*********************************************************************** %
%*                                                                     * %
%*        THIS PROC CALLS 'MYRREPTO' TO COLLECT THE NECCESARY          * %
%*         CHANGES REQUESTED BY OCP.  X=0 NO COLLECTION                * %
%*                                    X=1 NO COLLECTION                * %
%*                                    X=2 MODULATION & WINDER PARAMS   * %
%*                                    X=3 BANDING AVOIDANCE            * %
%*                                                                     * %
%*********************************************************************** %
 INT XFLAG:=0,J,LINE;

 IF X=2 THEN
   DISPWSPEED:=MYRREPTO("WINDER SPEED  ",DISPWSPEED,80.0,220.0,2,XPLN2);
     IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(9):=1;END;
     IF FLAGO = 2 THEN GOTO FINISH END;
   DISPTMF1AMP:=MYRREPTO("AMPLITUDE AT F1",DISPTMF1AMP,0.0,8.0,1,XPLN2);
     IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(2):=1;END;
     IF FLAGO = 2 THEN GOTO FINISH END;
   DISPTMF2AMP:=MYRREPTO("AMPLITUDE AT F2",DISPTMF2AMP,0.0,8.0,1,XPLN2);
```

```
    IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(3):=1;END;
    IF FLAGO = 2 THEN GOTO FINISH END;
  DISPTMF1PJ:=MYRREPTO('P-JUMP FOR F1',DISPTMF1PJ,0.0,8.0,1,XPLN2);
    IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(4):=1;END;
    IF FLAGO = 2 THEN GOTO FINISH END;
  DISPTMF2PJ:=MYRREPTO('P-JUMP FOR F2',DISPTMF2PJ,0.0,8.0,1,XPLN2);
    IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(5):=1;END;
    IF FLAGO = 2 THEN GOTO FINISH END;
  DISPTMPERIOD:=MYRREPTO('PERIOD',DISPTMPERIOD,2.0,30.0,1,XPLN2);
    IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(6):=1;END;
    IF FLAGO = 2 THEN GOTO FINISH END;
  DISPTACC:=MYRREPTO('ACCEL F1 TO F2',DISPTACC,0.5,7.0,1,XPLN2);
    IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(7):=1;END;
    IF FLAGO = 2 THEN GOTO FINISH END;
  DISPTDEC:=MYRREPTO('ACCEL F2 TO F1',DISPTDEC,0.5,7.0,1,XPLN2);
    IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(8):=1;END;
    GOTO FINISH;
.SEIF X = 3 THEN
   DISPMAXBAND:=INT(MYRREPTO('MAXIMUM NUMBER OF BANDING POINTS: ',
                  REAL(DISPMAXBAND),1.0,REAL(NUMBAND),0,XPLN6));
  IF FLAGO = 1 THEN XFLAG:=1;DISPMODFLAG(1):=1;END;
  LINE:=IREPTO('WHICH BANDING POINT TO CHANGE',0,NUMBAND,XPLN5);
  IF LINE = 0 THEN GOTO FINISH END;
  SCREENHEAD();
_INEAGAIN:
  TWRT('BANDING AVOIDANCE POINT NUMBER  ');IWRT(LINE);OUT(NL);
     DISPRIBS1(LINE):=MYRREPTO('RATIO R1',DISPRIBS1(LINE),0.0,8.0,1,
                    XPLN2);
        IF FLAGO = 1 THEN XFLAG:=1;DISPBANDFLAG(LINE*4-1):=1;END;
        IF FLAGO = 2 THEN GOTO FINISH END;
     DISPRIBS2(LINE):=MYRREPTO('RATIO R2',DISPRIBS2(LINE),0.0,8.0,1,
                    XPLN2);
        IF FLAGO = 1 THEN XFLAG:=1;DISPBANDFLAG(LINE*4):=1;END;
        IF FLAGO = 2 THEN GOTO FINISH END;
     DISPTF1SPEED(LINE):=MYRREPTO('FREQ  F1 ',DISPTF1SPEED(LINE),
                       30.0,320.0,2,XPLN2);
        IF FLAGO = 1 THEN XFLAG:=1;DISPBANDFLAG(LINE*4-3):=1;END;
        IF FLAGO = 2 THEN GOTO FINISH END;
     DISPTF2SPEED(LINE):=MYRREPTO('FREQ  F2 ',DISPTF2SPEED(LINE),
                       30.0,320.0,2,XPLN2);
        IF FLAGO = 1 THEN XFLAG:=1;DISPBANDFLAG(LINE*4-2):=1;END;
        IF FLAGO = 2 THEN GOTO FINISH END;
  LINE:=IREPTO('NEXT BANDING POINT ',0,DISPMAXBAND,XPLN5);
  IF LINE # 0 THEN  GOTO LINEAGAIN END;
:LSE
  GOTO ERL;
:ND;

-INISH:
  RETURN(XFLAG);
:NDPROC;

PROC CHANGEDATABASE( INT B);
%*****************************************************************************%
%*                                                                           *%
%* AFTER RECEIVING CLEARANCE TO UPDATE THE MASTER DATABASE SUBSEQUENT TO     *%
%* THE OCPDATABASE BEING UPDATED AND CHECKED,THIS PROC RE-CONVERTS THE       *%
%* VARIABLES FROM OCP FORM TO DATABASE VALUES.                               *%
```

```
:* DIVIDED INTO THREE SECTIONS - B=0 ERROR NO CHANGE TAKES PLACE          *%
:*                              B=1 NO ACTION                             *%
:*                              B=2 CHANGE MODULATION & WINDER PARAMS     *%
:*                              B=3 CHANGE BANDING PARAMETERS             *%
:*                                                                        *%
:*************************************************************************%

    FOR I:= 1 TO NUMBAND DO
        AWTF1SPEED(I):=DISPTF1SPEED(I);
        AWTF2SPEED(I):=DISPTF2SPEED(I);
        AWRIBS1(I):=DISPRIBS1(I);
        AWRIBS2(I):=DISPRIBS2(I);
    REP;
    AWTMF1AMP:=DISPTMF1AMP;
    AWTMF2AMP:=DISPTMF2AMP;
    AWTMF1PJ:=DISPTMF1PJ;
    AWTMF2PJ:=DISPTMF2PJ;
    AWTMPERIOD:=DISPTMPERIOD;
    AWTACC:=DISPTACC;
    AWTDEC:=DISPTDEC;
    AWWSPEED:=DISPWSPEED;
    AWMAXBAND:=DISPMAXBAND;

 IF B=2 THEN
   FOR I:= 2 TO 9 DO
      IF DISPMODFLAG(I)=1 THEN
        MODFLAG(I):=DISPMODFLAG(I);
      END;
   REP;
   UPDATEFLAG:=1;
 END;

 IF B=3 THEN
   FOR I:= 1 TO NUMBAND*4 DO
      IF DISPBANDFLAG(I)=1 THEN
        BANDFLAG(1+((I-1):/4),1+((I-1) MOD 4)):=DISPBANDFLAG(I);
      END;
   REP;
   IF DISPMODFLAG(1)=1 THEN
     MODFLAG(1):=DISPMODFLAG(1);
   END;
   UPDATEFLAG:=1;
 END;

 IF B=0 OR B<2 OR B>3 THEN
   TWRT('#NL# NO UPDATE OF DATABASE TOOK PLACE');
   GOTO ERL;
 END;

ENDPROC;

PROC FIXEDPARAMDISPLAY();
%   **********************************************************************   %
%   *                                                                    *   %
%   *   THIS SECTION CALCULATES AND DISPLAYS A 'DATABASE' WHICH          *   %
%   *           CONTAINS THE DATABASE VALUES  IN A FORM                  *   %
%   *           SUITABLE FOR PRESENTING  TO THE SCREEN.                  *   %
%   *       DISPLAYING FIXED PARAMETERS AND SPEED INDICATION.            *   %
```

```
%   *                                                                   *  %
%   *********************************************************************  %

DISPWSPEED  := CLOCKFREQ / (REAL(WSPEED) * 6.0);
DISPWMAXF   := CLOCKFREQ / (WMAXF * 6.0);
DISPTMAXF   := CLOCKFREQ / (TMAXF * 6.0);
DISPIDLESP  := CLOCKFREQ / (IDLESP * 6.0);
DISPWFREQ   := CLOCKFREQ / (WPITVAL(WPOINT) * 6.0);
DISPTFREQ   := CLOCKFREQ / (TPITVAL(TPOINT) * 6.0);

TWRT(' CURRENT CONTROL STATUS       =');

%        DETERMINE THE PRESENT OPERATING PHASE WITHIN THE SEQUENCE    %
IF TF2F=1 OR BANDF=1 THEN
     TWRT(' BANDING AVOIDANCE');
  ELSEIF WRUNF=1 AND TF1F=1 THEN
     IF WTACHO < 500.0 THEN
        TWRT('   CHUCK STOPPED')
     ELSE
        TWRT('  RUNNING')
     END;
  ELSEIF STARTF=1 OR SYNCF=1 THEN
     TWRT('  STARTING');
  ELSEIF STOPF=1 AND TTACHO < 500.0 THEN
     TWRT(' WINDER STOPPED');
  ELSEIF STOPF=1 THEN
     TWRT('  STOPPING');
  ELSE
     TWRT(' BUSY CHANGING STATUS');
END;
 OUT(NL);
TWRT('#NL# WIND MAX ACCEL/DECEL  RATE  =');RWRTF((MAXACC*1.667),FIELD,2);
     TWRT(' /');RWRTF((MAXDEC*1.667),3,2);UNIT(2);
TWRT('#NL# TRAV MAX ACCEL/DECEL RATE   =');RWRTF((TMXACC*1.667),FIELD,2);
     TWRT(' /');RWRTF((TMXDEC*1.667),3,2);UNIT(2);
TWRT('#NL# WINDER MAXIMUM FREQUENCY    =');RWRTF(DISPWMAXF,FIELD,2);UNIT(1);
TWRT('#NL# TRAVERSE MAXIMUM FREQUENCY  =');RWRTF(DISPTMAXF,FIELD,2);UNIT(1);
TWRT('#NL# MINIMUM FREQUENCY CLAMP     =');RWRTF(DISPIDLESP,FIELD,2);UNIT(1);
 OUT(NL);
TWRT('#NL# START-UP DELAY PERIOD       =');IWRTF(STARTSYNCTIM,11);UNIT(3);
TWRT('#NL# WINDER SPEED SETPOINT       =');RWRTF(DISPWSPEED,FIELD,2);UNIT(1);
TWRT('#NL# OUTPUT FREQUENCY - WINDER   =');RWRTF(DISPWFREQ,FIELD,2);UNIT(1);
TWRT('#NL# CAKE  SPEED  (1MIN AVERAGE) =');RWRTF(WTACHO,FIELD,2);UNIT(4);
 OUT(NL);
TWRT('#NL# CURRENT BANDING POINT(1-');IWRTF(MAXBAND,2);TWRT(')=');
           IWRTF(CURBAND,11);
TWRT('#NL# OUTPUT FREQUENCY - TRAVERSE =');RWRTF(DISPTFREQ,FIELD,2);UNIT(1);
TWRT('#NL# TRAVERSE SPEED  (1 MIN AVE) =');RWRTF(TTACHO,FIELD,2);UNIT(4);
TWRT('#NL# CURRENT RIBBON RATIO        =');
    IF TTACHO < (DISPIDLESP * 60.0) THEN
       TWRT('  TRAVERSE STOPPED');
    ELSEIF WTACHO < 500.0 THEN
       TWRT('   CHUCK STOPPED');
    ELSE
       DISPRRATIO:= (WTACHO * 6.0) / TTACHO;
       RWRTF(DISPRRATIO,7,3);
    END;
```

```
  TWRT('#NL# CAKE DIAMETER                     =');
      IF WTACHO < 500.0   THEN
         TWRT('   CHUCK STOPPED');
      ELSE
         DISPCAKE:= (DISPWSPEED / WTACHO) * 9014.723;
         RWRTF(DISPCAKE,FIELD,2);UNIT(5);
      END;
  OUT(NL);
ENDPROC;

PROC READDATA(INT Z);

%   ********************************************************************  ;
%   *                                                                  *  ;
%   *          THIS SECTION CALCULATES A 'DATABASE' WHICH              *  ;
%   *            CONTAINS THE DATABASE VALUES  IN A FORM               *  ;
%   *            SUITABLE FOR DISPLAYING  AND  AMMENDING               *  ;
%   *                   FOR  USE IN THE OCP TASK                       *  ;
%   *                                                                  *  ;
%   ********************************************************************  ;


 IF Z=0 THEN
  OPINUM:= - NOPI; % MANUAL MODS TO OPI NEGATE DATA LINK STORED OPI NUMBER ;
  FOR I:=1 TO NUMBAND DO
     IF TF1SPEED(I)<10 THEN TF1SPEED(I):=10;END;
     IF TF2SPEED(I)<10 THEN TF2SPEED(I):=10;END;
     DISPTF1SPEED(I):= REAL (CLOCKFREQ/(TF1SPEED(I) * 6.0));
     DISPTF2SPEED(I):= REAL (CLOCKFREQ/(TF2SPEED(I) * 6.0));
     DISPRIBS1(I)    := RIBS1(I);
     DISPRIBS2(I)    := RIBS2(I);
  REP;

  DISPTMF1AMP  := TMF1AMP;
  DISPTMF2AMP  := TMF2AMP;
  DISPTMF1PJ   := TMF1PJ;
  DISPTMF2PJ   := TMF2PJ;
  DISPTMPERIOD:= REAL TMPERIOD;
  DISPTACC     := TACC * 1.6667;
  DISPTDEC     := TDEC * 1.6667;
  DISPWSPEED   := CLOCKFREQ / (REAL(WSPEED) * 6.0);
  DISPWMAXF    := CLOCKFREQ / (WMAXF * 6.0);
  DISPTMAXF    := CLOCKFREQ / (TMAXF * 6.0);
  DISPIDLESP   := CLOCKFREQ / (IDLESP * 6.0);
  DISPMAXBAND  := MAXBAND;
 GOTO RET;
 END;

 IF Z=1 THEN
   OPINUM:=0; % MANUAL MODS TO OPI ZERO DATA LINK STORED OPI NUMBER %
   FOR I:=1 TO NUMBAND DO
     DISPTF1SPEED(I):=AWTF1SPEED(I);
     DISPTF2SPEED(I):=AWTF2SPEED(I);
     DISPRIBS1(I):=AWRIBS1(I);
     DISPRIBS2(I):=AWRIBS2(I);
   REP;
    DISPTMF1AMP:=AWTMF1AMP;
    DISPTMF2AMP:=AWTMF2AMP;
```

```
      DISPTMF1PJ:=AWTMF1PJ;
      DISPTMF2PJ:=AWTMF2PJ;
      DISPTMPERIOD:=AWTMPERIOD;
      DISPTACC:=AWTACC;
      DISPTDEC:=AWTDEC;
      DISPWSPEED:=AWWSPEED;
      DISPMAXBAND:=AWMAXBAND;
    GOTO RET;
  END;

  IF Z=2 THEN
    TWRT('#NL#      RIBBON POINTS          SPEEDS');SPS(15);
        TWRT('TRAVERSE  MODULATION#NL#');
    SPS(41);TWRT('AMPLITUDE            P-JUMP      PERIOD');
    TWRT('#NL#          R1      R2       F1        F2        F1        F2');
        SPS(6);TWRT('F1       F2      SECS');
    OUT(NL);
    IWRTF(1,2);
    RWRTF(DISPRIBS1(1),3,3);RWRTF(DISPRIBS2(1),2,3);
    RWRTF(DISPTF1SPEED(1),7,1);RWRTF(DISPTF2SPEED(1),5,1);
    RWRTF(DISPTMF1AMP,4,2);RWRTF(DISPTMF2AMP,4,2);
    RWRTF(DISPTMF1PJ,4,2);RWRTF(DISPTMF2PJ,4,2);RWRTF(DISPTMPERIOD,4,1);
    OUT(NL);
    FOR I:= 2 TO NUMBAND DO
      IWRTF(I,2);
      RWRTF(DISPRIBS1(I),3,3);RWRTF(DISPRIBS2(I),2,3);
      RWRTF(DISPTF1SPEED(I),7,1);RWRTF(DISPTF2SPEED(I),5,1);
      IF I = 6  THEN
         SPS(9);TWRT('TRAVERSE  ACCELERATION');
      END;
      IF I = 7  THEN
         SPS(10);TWRT('F1 TO F2    F2 TO F1');
      END;
      IF I = 8  THEN
         RWRTF((DISPTACC),13,2);RWRTF((DISPTDEC),7,2);
      END;
      IF I = 10 THEN
         SPS(11);TWRT('WINDER  SPEED');
      END;
      IF I = 11 THEN
         RWRTF(DISPWSPEED,17,2);
      END;
      IF I = 14 THEN SPS(8);
         TWRT('MAXBAND =');IWRT(DISPMAXBAND);
      END;
      OUT(NL);
    REP;
  END;
RET:  % RETURN %
ENDPROC;

% *********************************************************************** %
% *                                                                     * %
% *                 SECTION 5:                                          * %
% *                                                                     * %
% *********************************************************************** %

PROC MYIN()BYTE;
```

```
   BYTE B;
   IOFLAG:=0;
   B:=RRIN();
   IF B='Z' OR IOFLAG#0 THEN GOTO ERL END;
   RETURN(B);
ENDPROC;

PROC MYRREPTO (REF ARRAY BYTE CUE,REAL VALUE,MIN,MAX,INT DEC,
              PROC(INT) XPLNTN) REAL;
   %GIVES A CURRENT REAL VALUE AND PROMPTS FOR A NEW VALUE%
   %CHECKS VALIDITY.CARRAIGE RETURN LEAVES VALUE AS BEFORE%
   %BASED IN INSTD 'RREPTO' BY DEG 18-10-83%

   INT SIZE:=0;
   REAL R:=0.0;
   ERN:=0;
   ERP:=LERP;

   FLAGO:=0;
   SIZE:=(FIELD - DEC);
RPT:
   OUT(NL);TWRT(CUE);TWRT(" PRESENT =");RWRTF(VALUE,SIZE,DEC);
           TWRT("  CHANGE #ENQ#");
    R:=RREAD();

   IF TERMCH = 'X' THEN XPLNTN(0); GOTO RPT; END;

   IF TERMCH = 'F' THEN FLAGO:=2;GOTO XIT END;

   IF R<MIN OR R>MAX OR IOFLAG#0 OR ERN#0 THEN
      IF ERN#0 OR IOFLAG#0 THEN % MAYBE NUL %
XIT:
         ERN:=0;
         IOFLAG:=0;
         IF TERMCH = EOM THEN  % YES IT WAS %
            ERP:=LOCERP;
            RETURN(VALUE);     % LEAVE AS BEFORE %
         END;
      END;
      TWRT(" INVALID NO.");
      GOTO RPT;
   END;
   ERP:=LOCERP;
   FLAGO:=1;
   RETURN(R);
ENDPROC;

PROC LERP(INT A);
   ERN:=A;
ENDPROC;

PROC OPTIONS(INT LIMIT)INT;
    INT K:=0;
   OUT(NL);TWRT("OPTION=#ENQ#");
   K:=IREAD();
   IF K>LIMIT OR K<1 THEN ERP(5001);END;
   RETURN(K);
ENDPROC;
```

```
PROC LOCERP(INT N);
   TWRT("#NL#OUT OF RANGE  ERROR:");IWRT(N);NLS(1);
   GOTO ERL
ENDPROC;

PROC SCREENHEAD();
    CLEARSCREEN();
    SPS(SPACE);
    TWRT("***  SANS  INVERTER  CONTROL  SYSTEM  ***#NL(1)#");
ENDPROC;

PROC CLEARSCREEN();
    TWRT("#ESC,OPENBRACKET#2J#ESC#8");
ENDPROC;

PROC CLEARFLAGS();
    FOR I:=1 TO NUMBAND*4 DO
       DISPBANDFLAG(I):=0;
    REP;
    FOR I:=1 TO 9 DO
       DISPMODFLAG(I):=0;
    REP;
ENDPROC;

PROC UNIT (INT A);
   TWRT(UNITLIST(A));
ENDPROC;

PROC XPLN1(INT J);
  SCREENHEAD();
  OUT(NL);
  TWRT("#NL#    'Y' - CAUSES AN UPDATE OF THE DISPLAY.");
  TWRT("#NL#          (WITH REFRESHED DATA AS AT DISPLAYED TIME)");
  OUT(NL);
  TWRT("#NL#    'N' - RETURNS TO 'OCP CLEARED'");
  OUT(NL);
  TWRT("#NL#    'Z' - ESCAPES TO 'OCP CLEARED'");
  OUT(NL);
  TWRT("#NL#    'X' - EXPLANATION");
ENDPROC;

PROC XPLN2 (INT J);
      SCREENHEAD();
      TWRT("#NL# TYPE 'Z' TO QUIT - ANY CHANGES ARE DISCARDED");
      TWRT("#NL#      'F' TO FINISH MAKING CHANGES");
      TWRT("#NL# 'RETURN' TO LEAVE VALUE UNCHANGED");
      TWRT("#NL# 'INVALID NO' OCCURS FOR ALL OTHER CASES WHERE THE VALUES");
      TWRT("#NL#             ARE OUTSIDE THE FOLLOWING RANGES:");
      TWRT("#NL#      RATIO'S               1.0    TO  8.0");
      TWRT("#NL#      TRAVERSE SPEED     130.0HZ   TO  320.0HZ");
      TWRT("#NL#      WINDER SPEED        80.0HZ   TO  220.0HZ");
      TWRT("#NL#      MODULATION AMP       0.0 %   TO    8.0 % OF MEAN FREQ");
      TWRT("#NL#      MOD P-JUMP AMP       0.0 %   TO   70.0 % OF MOD AMP");
      TWRT("#NL#      MODULATION PERIOD    2.0SEC  TO   30.0SEC");
      TWRT("#NL#      RATE OF CHANGE       0.5HZ/S TO    7.0HZ/S");
ENDPROC;
```

```
PROC XPLN4(INT J);
   % SWITCH ON CHANGE OF BANDING,WINDER SPEED,MODULATION DATA.%
   SCREENHEAD();
   TWRT("#NL# TYPE 'R' TO RE-DISPLAY THE CONTENTS OF THE TEMPORARY DB");
   OUT(NL);
   TWRT("#NL# TYPE 'B' TO MAKE CHANGES TO THE BANDING AVOIDANCE DATA.");
   TWRT("#NL# TYPE 'M' TO MAKE CHANGES TO THE MODULATION PARAMETERS,");
   TWRT("#NL#            INCLUDING WINDER SPEED.");
   TWRT("#NL# TYPE 'E' TO EXIT AND RETURN TO 'OCP CLEARED'.");
   TWRT("#NL# TYPE 'Z' TO ESCAPE TO 'OCP CLEARED'.");
ENDPROC;

PROC XPLN5(INT J);
   % WHICH BANDING POINT TO CHANGE %
   SCREENHEAD();
   TWRT("#NL# ENTER THE NUMBER OF THE LINE TO BE CHANGED IE 1 TO 15");
   TWRT("#NL# ENTER A '0' TO EXIT PROPERLY.");
ENDPROC;

PROC XPLN6(INT J);
   % UPDATE OF MAXBAND %
   SCREENHEAD();
   TWRT("#NL# THERE ARE A MAXIMUM OF 15 ALLOWED BANDING POINTS.");
   TWRT("#NL# THE FIGURE SHOWN INDICATES THE CURRENT NUMBER IN USE.");
   TWRT("#NL# TO LEAVE UNCHANGED TYPE 'RETURN' .");
   TWRT("#NL#          ELSE ENTER THE REQUIRED NEW VALUE");
ENDPROC;
```

```
"ITLE TRAVERSE DRIVE TASK.
      CREATED 12-OCT-83 BY TEK. FILE : TDRV.RTL;

.ET NL = 10;
.ET ENQ = 5;
)PTION (1) BS;
% ************************************************************************ %
% *                                                                      * %
% *                       TRAVERSE DRIVE TASK.                           * %
% *                                                                      * %
% ************************************************************************ %

3VC DATA RRSIO;PROC () BYTE IN;PROC (BYTE) OUT;ENDDATA;
3VC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;

EXT PROC () BYTE RRIN;
EXT PROC (BYTE) RROUT;
EXT PROC (INT) DELAY,STOP;
EXT PROC () INT ME;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT) IWRT,NLS;
EXT PROC (INT,INT) IWRTF;
EXT PROC () INT IREAD;
EXT PROC () CLEANUP;

ENT PROC TDRIVE();
   INT DIFF,COUNT,TLAST,TSAFEDEC,TSAFEACC,T100MS:=5;
   REAL TACCEL,TDECEL,MAXJUMP:=54.0;

   IN:=RRIN;
   OUT:=RROUT;
   ERL := LOCERL;
   ERP := LOCERP;
   ERN := 0;

   STOP(ME());
   TPOINT := 1;
   TLAST:=TPITVAL(TPOINT);

TDRV1:

  WATCNT(1):=2;          % UPDATE WATCHDOG TIMER COUNT %

   % CHECK TO SEE IF MAXIMUM DECELERATION OR ACCELERATION %
   % RATES ARE BEING  EXCEEDED. IF THEY ARE, LIMIT THEM.  %

   DIFF:=TLAST-TPITVAL(TPOINT);
   IF DIFF > 0 THEN                                 % ACCELERATING %

        IF TSTAT(TPOINT) = 4 THEN                   % IF POS P-JUMP %
           TACCEL:=MAXJUMP;
        ELSE TACCEL:=TMXACC;END;

        TSAFEACC:=TLAST-INT((CLOCKFREQ*TLAST)/(CLOCKFREQ+TLAST*TACCEL));

        IF DIFF <= TSAFEACC THEN                    % IF ACCELERATION RATE SAFE %
           COUNT:=TPITVAL(TPOINT);                  % .....OUTPUT TABLE VALUE %
        ELSE
```

```
           COUNT:=TLAST-TSAFEACC;                          % IF NOT SAFE LIMIT IT %
        END;

    ELSEIF DIFF < 0 THEN                                   % DECELERATING %

        IF TSTAT(TPOINT) = 2 THEN                          % IF NEG P-JUMP %
           TDECEL:=MAXJUMP;
        ELSE TDECEL:=TMXDEC;END;

        TSAFEDEC:=INT((CLOCKFREQ*TLAST)/(CLOCKFREQ-TLAST*TDECEL))-TLAST;

        IF ABS DIFF <= TSAFEDEC THEN                       % IF DECEL. RATE SAFE....%
           COUNT:=TPITVAL(TPOINT);                         % ....OUTPUT TABLE VALUE %
        ELSE
           COUNT:=TLAST+TSAFEDEC;                          % IF NOT SAFE LIMIT IT %
        END;
    ELSEIF DIFF = 0 THEN                                   % NO CHANGE %
        COUNT:=TPITVAL(TPOINT);                            % OUTPUT TABLE VALUE %
    END;

% CHECK TO SEE IF FREQUENCY CLAMP VALUE IS BEING EXCEEDED, IF IT IS, LIMIT %
% THE VALUE TO THE CLAMP VALUE. %

    IF COUNT < TMAXF THEN                          % NOTE THAT COUNT PROP TO 1/F %
        COUNT:=TMAXF;
    END;

% WRITE THE VALUE TO THE PIT (INITIALISED IN MSMTU1.RTL) %

    CODE 10,10;
ITXCTR2XPORT    EQU     OBCH                   ;SBX PIT CHANNEL 0.
        MOV     AX,SS:[BP+*COUNT]
        OUT     ITXCTR2XPORT,AL                ;OUTPUT LOW COUNT.
        MOV     AL,AH
        OUT     ITXCTR2XPORT,AL                ;OUTPUT HIGH COUNT.
*RTL;

% INCREMENT THE TABLE POINTER 'TPOINT' AFTER A 100MS DELAY %

        DELAY(T100MS);                         % WAIT FOR 100MS %
        TLAST:=COUNT;                          % HOLD ACTUAL CURRENT VALUE %
        TPOINT:=TPOINT LAND HEX 7F + 1;        % INCREMENT TCONTTABLE POINTER %

GOTO TDRV1;

_OCERL:
    CLEANUP();
    GOTO TDRV1;
ENDPROC;

PROC LOCERP(INT N);
    TWRT('#NL#ERROR NUMBER ');IWRTF(N,5);NLS(1);
ENDPROC;
```

```
TITLE TRAVERSE CONTROL TASK.
      CREATED 12-OCT-83 BY TEK. FILE : TCONT.RTL
      LAST EDITED 2-OCT-84 BY TEK ;

LET SETF=1;
LET RESETF=0;
LET ENQ=5;
LET NL=10;
LET RAI=REF ARRAY INT;
LET RI=REF INT;
OPTION(1) BC;
% ************************************************************************ %
% *                                                                      * %
% *                      TRAVERSE CONTROL TASK.                          * %
% *                                                                      * %
% ************************************************************************ %

SVC DATA RRSIO;PROC () BYTE IN;PROC (BYTE) OUT;ENDDATA;
SVC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;

EXT PROC () BYTE RRIN;
EXT PROC (BYTE) RROUT;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT,INT) IWRTF;
EXT PROC () INT IREAD;
EXT PROC (INT) NLS,START;
EXT PROC (INT,INT,LABEL) TWAIT;
EXT PROC (RAI,INT,INT,REF REAL) DOWNRAMP;
EXT PROC (RAI,INT,INT,INT) RUN;
EXT PROC (RAI,INT,INT,INT,RI,REF REAL)UPRAMP;
EXT PROC (INT,INT,RI) TMOD;
EXT PROC (RI,RI,RI,REF REAL,INT,INT) BAND;
EXT PROC (REAL,INT,INT) RWRTF;
EXT PROC (REF REAL,REF REAL,RI) PARAM;
EXT PROC () CLEANUP;

EXT DATA TIMEDATA;
        INT NOW,SECSNOW,MINSNOW,NTICKS;
        INT TCOUNT,SECS,MINS,HOURS,DAYS,MONTHS,YEARS;
ENDDATA;

ENT PROC TALGO();
   INT TCALCP,TSTARTP,DELAY,BEGIN,PCALCFLAG,TEMP1,DUMMY,FLAG:=0;

   % INITIALISATION. %

   IN:=RRIN;
   OUT:=RROUT;
   ERL := LOCERL;
   ERP := LOCERP;
   ERN := 0;

   % DYNAMIC INIT OF WCONTTABLE %

   FOR J := 1 BY 1 TO NUMTAB DO
        TPITVAL(J):=IDLESP;
        TSTAT(J):=0;
   REP;
```

```
   TPOINT:=1;

   % INITIALISE FLAGS %

   TF1F:=TF2F:=PCALCFLAG:=TEMPFLAG:=BANDF:=RESETF;
   RUPF:=RDOWNF:=RAMPF:=RESETF;

   % CALCULATE PARAMETERS FOR INITIAL F1 MODULATION %

   PARAM(TMF1AMP,TMF1PJ,TF1SPEED(1));

   START(6);             % DRIVE TASK STARTED ONCE INITIALISATION COMPLETE %

MAINLOOP:
   WATCNT(3):=20;        % UPDATE WATCHDOG TIMER COUNT PERIOD %
   BEGIN:=NOW;
   TCALCP:=TPOINT;                       % SET UP TABLE POINTERS TO CALC .. %
   TSTARTP:=TPOINT;                      % .. NEW VALUES. %

   IF STOPF # 0 THEN                               % STOP SEQUENCE %
        TMXDEC:=STDEC;TMXACC:=STACC;
        DOWNRAMP(TPITVAL,TCALCP,TSTARTP,TMXDEC);
        FOR I := 1 TO NUMTAB DO
           TSTAT(TCALCP) := 0;           % FILL COND CODE TABLE WITH 0'S %
           TCALCP := TCALCP LAND HEX 7F + 1;
        REP;
   ELSEIF SYNCF # 0 THEN                           % START SYNC SEQUENCE %
        RUN(TPITVAL,TCALCP,TSTARTP,IDLESP);
   ELSEIF STARTF # 0 THEN                          % START SEQUENCE %
        UPRAMP(TPITVAL,TCALCP,TSTARTP,TF1SPEED(1),TF1F,TMXACC);
   ELSEIF RAMPF # 0 THEN                           % NEW OPI %
        PARAM(TMF1AMP,TMF1PJ,TF1SPEED(1));
        BAND(TF1SPEED(1),RAMPF,DUMMY,STDEC,TCALCP,TSTARTP);
   ELSEIF BANDF = 0 AND TF1F # 0 AND TF2F = 0 THEN       % F1 MODULATION %
        TMXDEC:=RUNDEC;TMXACC:=RUNACC;
        TMOD(TCALCP,TSTARTP,TF1SPEED(CURBAND));
   ELSEIF BANDF # 0 AND TF1F # 0 AND TF2F = 0 THEN       % F1 TO F2 RAMP %
        IF BANDF # PCALCFLAG THEN
           PCALCFLAG := BANDF;
           PARAM(TMF2AMP,TMF2PJ,TF2SPEED(CURBAND));
        END;
        BAND(TF2SPEED(CURBAND),TF1F,TF2F,TACC,TCALCP,TSTARTP);
   ELSEIF BANDF # 0 AND TF1F = 0 AND TF2F # 0 THEN       % F2 MODULATION %
        TMOD(TCALCP,TSTARTP,TF2SPEED(CURBAND));
   ELSEIF BANDF = 0 AND TF1F = 0 AND TF2F # 0 THEN       % F2 TO F1 RAMP %
        IF BANDF # PCALCFLAG THEN
           PCALCFLAG := BANDF;
           PARAM(TMF1AMP,TMF1PJ,TF1SPEED(CURBAND));
        END;
        BAND(TF1SPEED(CURBAND),TF2F,TF1F,TDEC,TCALCP,TSTARTP);
   END;

   DELAY:=550-NOW+BEGIN;                 % 1/50 THS TO GET HERE %
   TWAIT(SEQEV,DELAY,MAINLOOP);          % WAIT FOR 11 SECS OR EVENT 2 %
   GOTO MAINLOOP;

LOCERL:
   CLEANUP();
```

```
   GOTO MAINLOOP;
ENDPROC;

PROC LOCERR(INT N);
   TWRT("#NL#ERROR NO ");IWRTF(N,5);NLS(1);
ENDPROC;
```

```
TITLE WINDER TACHO TASK.
      CREATED 20-JAN-84 TEK. FILE: WTACHO.RTL
      LAST EDITED 8-MAR-84 DRT;

LET NL = 10;
LET ENQ=5;
OPTION (1) BS;
% ************************************************************************ %
% *                                                                      * %
% *                      WINDER TACHO TASK                               * %
% *                                                                      * %
% ************************************************************************ %

SVC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;
SVC DATA RRSIO;PROC () BYTE IN;PROC (BYTE) OUT;ENDDATA;

EXT DATA WINDSTOP;INT WSTOP,WENDFRAC,WCNT;ENDDATA;
EXT DATA TIMEDATA;
        INT NOW;                 % CYCLIC TICK COUNT %
        INT SECSNOW,MINSNOW;     % CYCLIC CLOCK COUNTS %
        INT NTICKS;              % OUTSTANDING TICKS TO PROCESS %
        INT TCOUNT,SECS,MINS,HOURS,DAYS,MONTHS,YEARS;
ENDDATA;

EXT PROC () BYTE RRIN;
EXT PROC (BYTE) RROUT;
EXT PROC () INT IREAD;
EXT PROC (INT,INT) IWRTF;
EXT PROC (REAL) RWRT;
EXT PROC (REAL,INT,INT) RWRTF;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT) NLS;
EXT PROC (INT) STOP,DELAY;
EXT PROC () INT ME;
EXT PROC (INT,INT,LABEL) TWAIT;
EXT PROC () CLEANUP;

ENT PROC WINDTACH();

  INT T8SEC:=400;                               % DELAY T*20 mS %
  REAL WCONST1,WTIM,IT50HZ:=IT50HZVAL;
  INT WCNTLAST:=100,LASTWSTART,LASTWGOFRAC,NEXTWSTART,NEXTWGOFRAC;


  IN:=RRIN;
  OUT:=RROUT;
  ERL := LOCERL;
  ERP := LOCERP;
  ERN := 0;

  % INITIALISATION FOR THIS MODULE %

  WCONST1 := 4.0 * CLOCKFREQ/3.0;        % 2 PULSES PER REV FOR 4 SECS %
                                         % OVER 6*(INV OUTPUT) => 4.0/3.0 %

  NEXTWSTART := NOW;NEXTWGOFRAC := 0;
  WTACHO:=0.0;
  WCNT:=20;
```

```
   WTACHOSTART();                                  % START INTERRUPT ROUTINES %

WIND1:

   WATCNT(6):=15;        % UPDATE WATCHDOG TIMER COUNT PERIOD %

   % CALCULATE COUNT VALUE ON THE CURRENT WINDER OUTPUT %
   % VALUE TO GIVE APPROX 8 SEC COUNT PERIOD %

   IF WPITVAL(WPOINT) >= IDLESP THEN
       WCNT:=100;
   ELSE WCNT:=IF WTACHO>500.0 AND TTACHO>500.0 THEN
                 INT(WCONST1/REAL(WPITVAL(WPOINT))*WTACHO/TTACHO*1.28571)
              ELSE                                       % 3/2*6/7=(INITIAL R)  %
                 100
              END;
   END;

   % WAIT FOR EVENT WHICH IS SET BY INTERRUPT SERVICE ROUTINE WHEN %
   % COUNTER HAS COUNTED WCNT PULSES FROM THE TACHO %

   TWAIT(WTACHOEV,T8SEC,WIND2);

   % CALCULATE WTACHO IN RPM %

   LASTWSTART:=NEXTWSTART;                     % SAVE OLD START TIME FOR %
   LASTWGOFRAC:=NEXTWGOFRAC;                   % CURRENT CALCULATION %
   NEXTWSTART:=WSTOP;                          % SAVE NEW START TIME FOR %
   NEXTWGOFRAC:=WENDFRAC;                      % NEXT CALCULATION %

% CHECK FOR POSSIBLE INTERRUPT CLASH AND IGNORE RESULT IF SO %
   IF LASTWGOFRAC+INTLMT>IT50HZ OR NEXTWGOFRAC+INTLMT>IT50HZ THEN
      WCNTLAST:=WCNT;
      GOTO WIND1
   END;

   WTIM:=REAL(WSTOP-LASTWSTART)+REAL(LASTWGOFRAC-WENDFRAC)/IT50HZ;
   WTACHO:=1500.0/WTIM*REAL WCNTLAST;     % IN RPM. %
                                          % WCNT/2 REVS IN WTIM 1/50THS SEC %
                                          % *60 SECS IN 1 MIN => 1500.0 %
   WCNTLAST:=WCNT;

   GOTO WIND1;

WIND2:
   WTACHO:=0.0;
   GOTO WIND1;

LOCERL:
   CLEANUP();
   GOTO WIND1;
ENDPROC;

PROC LOCERP(INT N);
   TWRT("#NL#ERROR NO ");IWRTF(N,5);NLS(1);
ENDPROC;

% ************************************************************************ %
```

```
% *                                                                        * %
% *        8088 MACHINE CODE INSERTS FOR RATIO TASK.                       * %
% *                                                                        * %
% ************************************************************************* %

PROC WTACHOSTART();
   CODE 10,10;

IT@CTR1@PORT     EQU      0D2H              ;SBC TIMER CHANNEL 1.(FOR WINDER)
   ;START WINDER COUNT DOWN TO INTERRUPT.CALLED ONCE AT START UP TO
   ;INITIATE INTERRUPT SEQUENCE.
        MOV      AX,SEG *WINDSTOP
        MOV      ES,AX
        MOV      AX,ES:*WCNT/WINDSTOP      ;GET WINDER COUNT VALUE.
        MOV      DX,IT@CTR1@PORT
        OUT      DX,AL                     ;LOAD LOW COUNT VALUE.
        MOV      AL,AH                     ;LOAD HI COUNT VALUE.
        OUT      DX,AL
   *RTL;
ENDPROC;
```

```
TITLE T18 INVERTER CONTROL SYSTEM
UPDATE MODULE TO INPUT THE TEMPORARY DATABASE 'AWAITUPDATE'
INTO THE MASTER DATABASE IF ANY OCP INPUTS HAVE OCCURED.
DEG 02/11/83 (REFERENDUM DAY) MODULE : DBUP.RTL
LAST EDITED 2-OCT-84 TEK;

SVC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;

EXT PROC (INT) WAIT;

ENT PROC UPDATEDB();
% ********************************************************************** %
% *                                                                      %
% *      THIS MODULE/PROC UPDATES THE MASTER DATABASE IF ANY CHANGES   * %
% *      HAVE BEEN MADE TO THE TEMPORARY DATABASE VIA THE  OCP TASK.   * %
% *   THE PROC WAITS FOR 'DBUPDATE' EVENT - SET BY A LOW CHUCK SPEED   * %
% *                                                                  * %
% ********************************************************************** %

ERL:=STARTLB;
ERN:=0;
STARTLB:
  WAIT(DBUPDATE);  % EVENT SET BY WTACHO(CHUCK SPEED) LOW %
  IF UPDATEFLAG = 0 THEN GOTO STARTLB; END;
  IF UPDATEFLAG = 1 THEN GOTO CHANGEDB; END;
  UPDATEFLAG:=0;
  GOTO STARTLB;

CHANGEDB:
  ERL:=CHANGEDB;
  UPDATEFLAG:=0;

        % ADDED TO VERSION 1.7 BY TEK 11-10-84 %

  IF ABS(TF1SPEED(1)-INT(CLOCKFREQ/(AWTF1SPEED(1)*6.0))) <= 5 THEN
        RAMPF:=0;                  % DON'T RAMP FOR SMALL CHANGE %
  ELSE
        RAMPF:=1;                  % INITIATES SMOOTH RAMP TO NEW SPEED %
  END;

  IF STOPF = 0 THEN                % DON'T SET FLAGS IF IN STOP STATE %
        TF1F:=1;TF2F:=0;           % RESET FLAGS FOR NEW CYCLE %
        BANDF:=0;
  ELSE
        RAMPF:=0;
  END;

        % END OF VERSION 1.7 PATCH %

% ********************************************************************** %
% *                                                                      %
% *   IF ANY OF THE BANDING POINTS HAVE BEEN CHANGED THEN RE-CONVERT   * %
% *           THEM FROM OCP DATA FORM TO DATABASE VALUES.               * %
% *                                                                  * %
% *                                                                  * %
% ********************************************************************** %

  FOR I:= 1 TO NUMBAND DO
    IF BANDFLAG(I,1)=1 THEN    % FREQUENCY BEFORE BANDING AVOIDANCE %
```

```
      BANDFLAG(I,1):= 0;
      TF1SPEED(I):= INT(CLOCKFREQ/(AWTF1SPEED(I) * 6.0));
     END;
   IF BANDFLAG(I,2)=1 THEN   % FREQUENCY DURING BANDING AVOIDANCE %
      BANDFLAG(I,2):= 0;
      TF2SPEED(I):= INT(CLOCKFREQ/(AWTF2SPEED(I) * 6.0));
     END;
   IF BANDFLAG(I,3)=1 THEN   % BANDING AVOIDANCE START RATIO %
      BANDFLAG(I,3):= 0;
      RIBS1(I):= AWRIBS1(I);
     END;
   IF BANDFLAG(I,4)=1 THEN   % BANDING AVOIDANCE STOP RATIO %
      BANDFLAG(I,4):= 0;
      RIBS2(I):= AWRIBS2(I);
     END;
 REP;

 IF MODFLAG(1)=1 THEN
   MODFLAG(1):=0;
   MAXBAND:=AWMAXBAND;   % NUMBER OF BANDING POINTS IN USE %
  END;

% ************************************************************************ %
% *                                                                      * %
% *          UPDATE OF MODULATION DATA CHANGES (IF ANY)                  * %
% *                                                                      * %
% ************************************************************************ %

 IF MODFLAG(2)=1 THEN
   MODFLAG(2):=0;
   TMF1AMP:=AWTMF1AMP;   % MODULATION AMPLITUDE AT FREQ 1 %
 END;

 IF MODFLAG(3)=1 THEN
   MODFLAG(3):=0;
   TMF2AMP:=AWTMF2AMP;   % MODULATION AMPLITUDE AT FREQ 2 %
  END;

 IF MODFLAG(4)=1 THEN
   MODFLAG(4):=0;
   TMF1PJ:=AWTMF1PJ;     % P-JUMP AMPLITUDE AT FREQ 1 %
  END;

 IF MODFLAG(5)=1 THEN
   MODFLAG(5):=0;
   TMF2PJ:=AWTMF2PJ;     % P-JUMP AMPLITUDE AT FREQ 2 %
  END;

 IF MODFLAG(6)=1 THEN    % PERIOD OF MODULATION %
   MODFLAG(6):=0;
   TMPERIOD:=INT AWTMPERIOD;
  END;

 IF MODFLAG(7)=1 THEN    % ACCELERATION RATE FROM F1 TO F2 %
   MODFLAG(7):=0;
   TACC:=AWTACC / 1.6667;
  END;
```

```
  IF MODFLAG(8)=1 THEN     % ACCELERATION RATE FROM F2 TO F1 %
    MODFLAG(8):=0;
    TDEC:=AWTDEC / 1.6667;
   END;

  IF MODFLAG(9)=1 THEN     % CHANGE WINDER SPEED SETPOINT %
     MODFLAG(9):=0;
     WSPEED:=INT (CLOCKFREQ / (AWWSPEED * 6.0));
  END;

  NOPI:=OPINUM;
  OPINUM:=0;

  ERL:=STARTLB;
  GOTO STARTLB;
ENDPROC;
```

```
TITLE DATALINK TO HOST
      M.MALENGRET 20-DEC-83
      LAST EDITED 16-JUL-84 DRT

MODULE********LINKDA.RTL************

;
LET SYNC='U';
LET STX =2;
LET LINKRXEV=6;
LET CR=13;
LET BUFSIZE=138;
LET NL=10;
LET SP=32;
LET DLMSGSNDCMP=7; % DL MSG SEND COMPLETE EVENT FLAG %

SVC DATA RRSIO;PROC ()BYTE IN;PROC (BYTE) OUT;ENDDATA;
SVC DATA RRERR;LABEL ERL;INT ERN;PROC (INT) ERP;ENDDATA;

EXT PROC (INT,INT,LABEL) TWAIT;
EXT PROC ()BYTE INPORTA;
EXT PROC (BYTE)RROUT;
EXT PROC (REF ARRAY BYTE)TWRT;
EXT PROC (INT)IWRT;
EXT PROC ()RETEV;
EXT PROC (INT)QEV;
EXT PROC (INT)WAIT;
EXT PROC ()PTORTL;
EXT PROC (INT) SENDBUFF;
EXT PROC (INT) RESET;

EXT DATA BUFFERS;
    BYTE CHAR;
    ARRAY(BUFSIZE)BYTE RXBUFF;
    INT LRBP,
        LTBP,
        LRCOUNT,
        LTMAX,
        SEND;
ENDDATA;

ENT PROC SENDME();
   BYTE TYPE,MESSAGE,ERRNO;
   REAL RTEMP;
   INT I,CRS,STBIT,M;
   REF INT RI:=I;
   ERL:=LOCERL;
   OUT:=RROUT;

START:
   RESET(DLMSGSNDCMP);
   CRS:=12;
   LRBP:=0;
   WAIT(LINKRXEV);
   IF CHECKSUM()#0 THEN
      ERRNO:=1;
      GOTO ERROR
   END;
```

```
   MESSAGE:=RXBUFF(5);
   IF MESSAGE='R' THEN GOTO RXMESS;
   ELSEIF MESSAGE='S' THEN GOTO SXMESS;
   ELSEIF MESSAGE='Z' THEN
      DABLTX();
      GOTO START;
   ELSE ERRNO:=2; GOTO ERROR;
   END;
RXMESS:
   PAKR(TTACHO,RXBUFF(12));
   PAKR(WTACHO,RXBUFF(16));
   RTEMP:=CLOCKFREQ/(TPITVAL(TPOINT)*6.0);
   PAKR(RTEMP,RXBUFF(20));
   RTEMP:=CLOCKFREQ/(WPITVAL(WPOINT)*6.0);
   PAKR(RTEMP,RXBUFF(24));
   RXBUFF(28):=BYTE CURBAND;
   PACKI(NOPI,RXBUFF(29));
   RXBUFF(31):=IF TF2F=1 OR BANDF=1 THEN 1
               ELSEIF TF1F=1 AND WRUNF=1 THEN IF WTACHO<500.0 THEN 4 ELSE 2 END
               ELSEIF STARTF=1 OR SYNCF=1 THEN 3
               ELSEIF STOPF=1 AND TTACHO<500.0 THEN 6
               ELSEIF STOPF=1 THEN 5
               ELSE 7
               END;
   CRS:=32;
   GOTO OKEY;
SXMESS:
   TYPE:=BYTE(RXBUFF(11));
   SWITCH TYPE OF BADT,TYPE2,TYPE3,TYPE4,TYPE5;
BADT:
   ERRNO:=3; GOTO ERROR;%NO TYPE 1 FOR S MESSAGE%
TYPE2:
   UNPACKI(RXBUFF(12),OPINUM);
   FOR K:=1 TO 15 DO
      UNPAKR(RXBUFF(10+K*4),AWTF1SPEED(K));
      UNPAKR(RXBUFF(70+K*4),AWTF2SPEED(K));
      BANDFLAG(K,1):=1;
      BANDFLAG(K,2):=1;
   REP;
   GOTO OKEY;
TYPE3:
   UNPACKI(RXBUFF(12),I);
   IF I#OPINUM THEN ERRNO:=4;OPINUM:=0;GOTO ERROR END;
   FOR K:=1 TO 15 DO
      UNPAKR(RXBUFF(10+K*4),AWRIBS1(K));
      UNPAKR(RXBUFF(70+K*4),AWRIBS2(K));
      BANDFLAG(K,3):=1;
      BANDFLAG(K,4):=1;
   REP;
   GOTO OKEY;
TYPE4:
   UNPACKI(RXBUFF(12),I);
   IF I#OPINUM THEN ERRNO:=4;OPINUM:=0;GOTO ERROR END;
   UNPAKR(RXBUFF(14),AWTMF1AMP);
   UNPAKR(RXBUFF(18),AWTMF2AMP);
   UNPAKR(RXBUFF(22),AWTMF1PJ);
   UNPAKR(RXBUFF(26),AWTMF2PJ);
   UNPAKR(RXBUFF(30),AWTACC);
```

```
    UNPAKR(RXBUFF(34),AWTDEC);
    UNPAKR(RXBUFF(38),AWTMPERIOD);
    UNPAKR(RXBUFF(42),AWWSPEED);
    AWMAXBAND:=RXBUFF(46);
    FOR K:=1 TO 9 DO MODFLAG(K):=1 REP;
    UPDATEFLAG:=1;
    GOTO OKEY;
TYPE5:
    FOR K:=1 TO 20 DO
       UNPACKI(RXBUFF(12+K*2),PEOPLE(K));
       SECCODE(K):=RXBUFF(K+53);
    REP;
    NOVALID:=20;
    GOTO OKEY;
ERROR:
    SETSTART();RXBUFF(5):='N';RXBUFF(10):=1;
    RXBUFF(11):=ERRNO;
    RXBUFF(12):=CALCCSUM(12);
    SENDBUFF(12);
    GOTO TSTCMP;
 OKEY:
    SETSTART();
    RXBUFF(5):='A';
    RXBUFF(10):=BYTE(CRS-11);
    RXBUFF(11):=TYPE;
    RXBUFF(CRS):=CALCCSUM(CRS);
    SENDBUFF(CRS);
TSTCMP:
    TWAIT(DLMSGSNDCMP,10,CLR);
    M:=0;
    WHILE M<50 DO;% WAIT FOR SERIAL BUFFER TO BE EMPTY, BUT NOT TOO LONG %
       FOR K:=1 TO 5 DO
       CODE 0,0;
          MOV DX,0A2H ; PIGGY BACK STATUS ADDRESS
          IN  AL,DX   ; READ STATUS
          AND AL,04H  ; CHECK TXE BIT
          MOV SS:[BP+*STBIT],AL
       *RTL;
       M:=M+1;
       IF STBIT=0 THEN GOTO NOTYET END;
       REP;
       GOTO CLR;
NOTYET:
    REP;
CLR:
    DABLTX();
    GOTO START;
LOCERL:
    TWRT("#NL# DATA LINK COMMS TASK ERROR ERN = ");
    IWRT(ERN);
    OUT(NL);
    GOTO START;
ENDPROC;

PROC SETSTART();
    RXBUFF(1):=SYNC;
    RXBUFF(2):=STX;
    RXBUFF(3):=MYADDRESS();
```

```
   RXBUFF(4):='@';%DEST ADDR%
ENDPROC;

PROC DABLTX();
   CODE 0,0;
      MOV DX, 0A2H ; PIGGY BACK SERIAL PORT
      MOV AL, 037H ; ENABLE RX & TX AND RESET ERROR FLAG AND DTR & RTS = 1
      OUT DX,AL    ; WRITE ABOVE CMD
   *RTL;
ENDPROC;

ENT PROC MYADDRESS()BYTE;
   RETURN(BYTE(INPORTA()LAND HEX OF + 64));% IE SET SITE 1 TO ADDRESS A %
ENDPROC;

PROC CALCCSUM(INT A)BYTE;
   INT TMP:=0;
   FOR I:= 3 TO A-1 DO
   TMP:=TMP-RXBUFF(I);
   REP;
   RETURN(BYTE TMP);
ENDPROC;

PROC CHECKSUM()BYTE;
INT TMP:=0;
   FOR I:=3 TO LRBP DO
      TMP:=TMP+RXBUFF(I);
   REP;
   RETURN(BYTE TMP);
ENDPROC;

PROC PACKI(REF INT RI,REF BYTE RB);
   BLOCK REF BYTE RBTMP:=RB;ENDBLOCK;
   BLOCK REF INT RI1;
      VAL RI1:=RI;
   ENDBLOCK;
ENDPROC;

PROC UNPACKI(REF BYTE RB,REF INT RI);
   BLOCK REF BYTE RBTMP:=RB;ENDBLOCK;
   BLOCK REF INT RI1;
      VAL RI:=RI1;
   ENDBLOCK;
ENDPROC;

PROC UNPAKR(REF BYTE RB,REF REAL RR);
   BLOCK REF REAL RRTMP:=RR;REF BYTE RBTMP:=RB;ENDBLOCK;
   BLOCK INT I1,I2,I3,I4;
      I2:=I2+2;           %  ORDER OF WORDS IN REAL IS REVERSED AS WELL %
   ENDBLOCK;
   BLOCK REF INT RI1,RI2;
      % REDUCE BIAS ON EXPONENT BY 2 PDP TO INTEL REAL FORMAT %
      VAL RI1:=IF RI2=0 THEN 0
               ELSE (RI2 LAND HEX 807F)LOR (RI2 LAND HEX 7F80 - HEX 100)
               END;
   ENDBLOCK;
   BLOCK INT I1,I2,I3,I4;          % DOUBLE WORD REF'S SEG AND POINTER %
      I2:=I2-2;
```

```
      I4:=I4+2;
   ENDBLOCK;
   BLOCK REF INT RI1,RI2;
      VAL RI1:=RI2;
   ENDBLOCK;
ENDPROC;

PROC PAKR(REF REAL RR,REF BYTE RB);
   BLOCK REF REAL RRTMP:=RR;REF BYTE RBTMP:=RB;ENDBLOCK;
   BLOCK INT I1,I2,I3,I4;
      I2:=I2+2;
   ENDBLOCK;
   BLOCK REF INT RI1,RI2;
     % INCREASE BIAS OF EXPONENT BY 2 INTEL TO PDP REAL FORMAT CONVERSION %
      VAL RI2:=IF RI1=0 THEN 0
               ELSE (RI1 LAND HEX 807F)LOR (RI1 LAND HEX 7F80 + HEX 100)
               END;
   ENDBLOCK;
   BLOCK INT I1,I2,I3,I4;          % DOUBLE WORD REF'S SEG AND POINTER %
      I2:=I2-2;
      I4:=I4+2;
   ENDBLOCK;
   BLOCK REF INT RI1,RI2;
      VAL RI2:=RI1;
   ENDBLOCK;
ENDPROC;
```

```
TITLE DRIVER MODULE FOR COMMS BETWEEN iSBC AND HOST
      M.MALENGRET 20-DEC-83
      LAST EDITED 2-MAR-84 DRT
      MODULE ** LINKDR.RTL ***;

LET SYNC='U';
LET STX=2;
LET LINKRXEV=6;
LET CR=13;
LET BUFSIZE=138;
LET SOURCEADDRESS='@';
LET DLMSGSNDCMP=7;  % DATA LINK MSG SEND COMPLETED EVENT %

EXT PROC ()RETEV;
EXT PROC (INT)QEV;
EXT PROC ()PTORTL;
EXT PROC () RETFIN;
EXT PROC ()BYTE INPORTA;

ENT DATA BUFFERS;
    BYTE CHAR:=CR;
    ARRAY(BUFSIZE)BYTE RXBUFF;
    INT LRBP:=0,
        LTBP:=0,
        LRCOUNT:=0,
        LTMAX:=0,
        SEND;
ENDDATA;

ENT PROC HOSTLINK();
      CODE 24,0;
      SIO@DATA@PORT EQU 0A0H
      SIO@STAT@PORT EQU 0A2H
      PUBLIC LINKRXINT
&CRN  P,LINKRXINT,QOZ
LINKRXINT:
   CALL RP@PTORTL
   MOV DX,SIO@DATA@PORT
   IN AL,DX
   MOV *CHAR/BUFFERS,AL
   *RTL;
   LRBP:=LRBP+1;
   SWITCH LRBP OF SYNCHAR,STXCHAR,SRADDR,DTADDR,RET,RET,RET,
   RET,RET,LNGTH;
   GOTO CONT;
SYNCHAR:
   IF CHAR#SYNC THEN LRBP:=0;GOTO CON1 END;GOTO RET;
STXCHAR:
   IF CHAR=STX THEN GOTO RET END;
RETST:
   LRBP:=1;GOTO SYNCHAR;
SRADDR:
   IF CHAR=SOURCEADDRESS THEN GOTO RET ELSE GOTO RETST END;
DTADDR:
   IF CHAR=DESTADDR() THEN GOTO RET ELSE GOTO RETST END;
LNGTH:
   LRCOUNT:=CHAR;
   IF LRCOUNT+11>LENGTH RXBUFF THEN LRCOUNT:=0 END;
```

```
TITLE
STANDARD INTERACTIVE PROCS
MODIFIED FROM CONMAC FOR SANS P17 PACKERMAC  19 9 80
   10/12/81 ADDED EXECODE,REMOVED EXEQUES
   20/4/84 LAST EDITED DRT
MODULE *** INTSTD.RTL *** ;

%EDITED 16-9-83 DRT FOR USE IN MAGIC MICRO SYSTEM %

% CONVERSATIONAL PROGRAMS ARE MORE EFFECTIVE WHEN THEY MAKE USE OF A GOOD %
% SET OF STANDARD INTERACTIVE PROCS.  THE OBJECTIVES ARE... %
% (1) TO MAKE PROGRAMS MORE COMPACT %
% (2) TO SIMPLIFY TRAINING BY CONVERSING IN A CONSISTENT METHOD & STYLE %
% (3) TO ENCOURAGE OPTIONAL EXPLANATIONS.  EXPERIENCED OPERATORS WILL HOLD %
%     RAPID CONVERSATIONS IN THE COMPACT "CUE & REPLY" FORM, WHILE THE %
%     LESS CONFIDENT MAY RESPOND WITH "X".  THIS SHOULD RESULT IN MORE %
%     EXPLANATIONS BEING PROGRAMMED WHILE AVOIDING UNSOLICITED CLUTTER. %
% (4) TO INSIST THAT CHANGES ARE PROPERLY COMPLETED ON "EXECUTE" %
%     AND A HARD-COPY JOTTING IS MADE. %
% PROGRAMMERS MAY HAVE SOMETIMES TO CREATE THEIR OWN SPECIAL PROCS: %
% THESE SHOULD CONFORM A.F.A.P. TO THE STYLE OF THE STANDARDS BELOW. %

LET REJECT = 'Z';
LET BACKSPACE=OCT 10;
LET NL = 10;
LET YEAH = 1;
LET SP = 32;
LET EOM = 3;
LET ENQ = 5;
LET ON=1;
LET OFF=0;



EXT PROC (REAL,INT,INT) RWRTF;
EXT PROC () REAL RREAD;
EXT PROC () INT IREAD,ME;
EXT PROC (INT) TIMDAT, IWRT, DELAY;
EXT PROC (INT) SPS;
EXT PROC (INT,INT) IWRTF;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC () SETNECHO;

SVC DATA RRSIO;
   PROC () BYTE IN; PROC (BYTE) OUT;
ENDDATA;

SVC DATA RRERR;
   LABEL ERL; INT ERN; PROC (INT) ERP;
ENDDATA;

SVC DATA RRSED;
   BYTE TERMCH, IOFLAG;
ENDDATA;

EXT DATA TIMEDATA;
   INT NOW,SECSNOW,MN,NTICKS,TCOUNT,SECS,MINS,HRS,DAYS,MONTHS,YEARS;
ENDDATA;
```

```
ENT PROC NOXPLNTN (INT J);
   TWRT("#NL#    SORRY - NO EXPLANATION AVAILABLE");
ENDPROC;

ENT PROC IREPTO (REF ARRAY BYTE CUE, INT MIN,MAX, PROC(INT) XPLNTN) INT;
   PROC (INT) REMERP:=ERP;
   INT J;
   ERP:=XPLNTN;
RPT:
   OUT(NL); TWRT(CUE); TWRT(" = #ENQ#");
   J:=IREAD();
   IF J<MIN OR J>MAX OR IOFLAG#0 THEN
      IF IOFLAG = 0 THEN
         TWRT(" INVALID NO ");
      ELSE
         IOFLAG := 0;
      END;
      GOTO RPT;
   END;
   ERP:=REMERP;
   RETURN(J);
ENDPROC;

%ENT PROC RREPTO (REF ARRAY BYTE CUE, REAL MIN,MAX, PROC(INT) XPLNTN) REAL;%
%   REAL R;%
%   PROC (INT) REMERP:=ERP;   ERP:=XPLNTN;%
%RPT:%
%   OUT(NL); TWRT(CUE); TWRT(" = #ENQ#");%
%   R:=RREAD();%
%   IF R<MIN OR R>MAX OR IOFLAG#0 THEN%
%      IF IOFLAG = 0 THEN%
%         TWRT(" INVALID NO ");%
%      ELSE%
%         IOFLAG := 0;%
%      END;%
%      GOTO RPT;%
%   END;%
%   ERP:=REMERP;%
%   RETURN(R);%
%ENDPROC;%

ENT PROC CHOICE (REF ARRAY BYTE CUE, REPLIST, PROC(INT) XPLNTN) INT;
   INT L:=LENGTH REPLIST;   BYTE C;
RPT:OUT(NL); TWRT(CUE); TWRT(" (");
   FOR J:=1 TO L DO
      OUT(REPLIST(J)); IF J<L THEN OUT('/') END;
   REP;
   TWRT(") ? #ENQ#");
   C:=IN(); IF C='X' THEN XPLNTN(0); GOTO RPT; END;
   FOR J:=1 TO L DO
      IF C=REPLIST(J) THEN RETURN(J) END;
   REP;
   IF C = EOM THEN
      RETURN(0);
   END;
   TWRT("    NOT IN LIST");   GOTO RPT;
```

```
ENDPROC;

ENT PROC EXECODE(INT SECURITY)INT;              %SECURITY=0-NO SECURITY%
                                %RETURNS 0-INVALID%
                                %           -1 NO SECURITY%
                                %           1-30  OPS NO%
   INT PERSON:=-1,PCODE:=0,NOCHAR:=2;
   IF SECURITY#0 THEN
      TWRT("#NL#SECURITY EXECUTE:");
      PERSON:=IREPTO("OPERATOR NO",1,NOVALID,NOXPLNTN);
      SETNECHO();
      PCODE:=IREPTO("CODE NO",1,32767,NOXPLNTN);
      IF PEOPLE(PERSON)#PCODE OR SECCODE(PERSON)<SECURITY THEN
         TWRT(" INVALID NO ");OUT(NL);RETURN(0);
      END;
   END;
   IF CHOICE("EXECUTE","YN",NOXPLNTN)=1 THEN
      TWRT(" EXECUTED");RETURN(PERSON);
   END;
   RETURN(0);
ENDPROC;

EXT DATA PATTERNS;   %DATA 'PATTERNS' IN SMT%
   ARRAY(16)INT MASKS;
ENDDATA;

ENT PROC SETBIT(INT BITNO,REF INT WD,INT V);
   IF V#0 THEN
      VAL WD:=WD LOR MASKS(BITNO);
   ELSE
      VAL WD:=WD LAND NOT MASKS(BITNO);
   END;
ENDPROC;

ENT PROC SETBYT(INT BITNO,REF BYTE WD,INT V);
   IF V=1 THEN
      VAL WD:=WD LOR BYTE(MASKS(BITNO));
   ELSE
      VAL WD:=WD LAND BYTE(NOT MASKS(BITNO));
   END;
ENDPROC;

ENT PROC VALBIT(INT BITNO,WD)INT;
   IF WD LAND MASKS(BITNO)#0 THEN RETURN(1) END;
   RETURN(0);
ENDPROC;

DATA DIG;
   BYTE OUTSTORE:=0;
ENDDATA;

ENT PROC DIGOUT(BYTE SELECT,STATE);
   REF BYTE STORE:=OUTSTORE;SETBYT(SELECT,STORE,IF STATE=1 THEN 0 ELSE 1 END);
   OUTPORTB(OUTSTORE);
ENDPROC;

ENT PROC DIGIN(BYTE SELECT)BYTE;
   INT INPUT:=INPORTA();
```

```
   IF VALBIT(SELECT,INPUT)=ON THEN RETURN(0)END;
   RETURN(1);
ENDPROC;

PROC OUTPORTB(BYTE STATE);
   CODE 20,0;
PORTBADDR EQU OCAH
OUTP:
   MOV AL,SS:[BP+*STATE]   ;LOAD STATE VALUE TO REG AX
   OUT PORTBADDR,AL        ;OUTPUT BINARY VALUE OF STATE
*RTL
ENDPROC;

ENT PROC INPORTA()BYTE;
   BYTE A;
   CODE 14,0;
   PORTAADDR EQU OC8H
   IN AL,PORTAADDR
   MOV SS:[BP+*A],AL
   *RTL;
   RETURN(A);
ENDPROC;
```

```
TITLE PROCS USED MORE THAN ONCE BY WINDER & TRAVERSE TASKS.
      CREATED 22-SEP-83 BY TEK. FILE : COMPROC.RTL
      LAST EDITED 2-OCT-84 BY TEK;

LET SETF=1;
LET RESETF=0;
LET NL=10;
LET DELOFFSET = 3;
OPTION (1) BC;
% ************************************************************************ %
% *                                                                      * %
% *          PROCS COMMON TO SEVERAL PARTS OF THE WINDER                 * %
% *                 AND TRAVERSE CONTROL TASKS.                          * %
% *                                                                      * %
% ************************************************************************ %


SVC DATA RRSIO;PROC () BYTE IN;PROC (BYTE) OUT;ENDDATA;

EXT PROC () BYTE RRIN;
EXT PROC (BYTE) RROUT;
EXT PROC (REF ARRAY BYTE) TWRT;
EXT PROC (INT,INT) IWRTF;
EXT PROC (INT) NLS;
EXT PROC (REAL,INT,INT) RWRTF;

% ************************************ %
% RAMP DOWN TO STOP ****************** %
% ************************************ %

ENT PROC DOWNRAMP(REF ARRAY INT PITVAL,INT CALCP,STARTP,REF REAL DEC);
   INT LASTDR,CURRENT;

   % CALCULATES CONTROL TABLE VALUES FOR STOPPING MACHINE %


    FOR I:=0 BY 1 TO 1 DO                  % HOLD SPEED CONST FOR 2 TICS %
        PITVAL((STARTP + I)LAND HEX 7F + 1):=PITVAL(STARTP);
    REP;

    CALCP:=(CALCP+2)LAND HEX 7F + 1;
    LASTDR:=PITVAL(CALCP);

DRLOOP:
   CURRENT:=INT((CLOCKFREQ*LASTDR)/(CLOCKFREQ-LASTDR*DEC));

   IF CURRENT > IDLESP THEN
        CURRENT:=IDLESP;                   % CAN'T GO LESS THAN 240 HZ %
        RUN(PITVAL,CALCP,STARTP,CURRENT); % FILL REST OF TABLE WITH STOP VALS %
        GOTO DREXIT;
   END;

   PITVAL(CALCP):=CURRENT;                 % WRITE NEW VALUE INTO TABLE %
   LASTDR:=CURRENT;                        % REMEMBER CURRENT VALUE %
   CALCP:=CALCP LAND HEX 7F + 1;           % DO NEXT TABLE VALUE %

   IF CALCP = STARTP THEN                  % FILLED THE TABLE YET? %
        GOTO DREXIT;
```

```
    ELSE GOTO DRLOOP;
    END;

DREXIT:
ENDPROC;

% ************************************** %
% RAMP UP TO RUN *********************** %
% ************************************** %

ENT PROC UPRAMP(REF ARRAY INT PITVAL,INT CALCP,STARTP,SPEED,REF INT RUNF,
    REF REAL ACC);
    INT LASTUP,CURRENT;

    % CALCULATES CONTROL TABLE VALUES FOR RAMPING MACHINE UP %

    FOR I:=0 BY 1 TO 1 DO                    % HOLD SPEED CONSTANT FOR 2 TICS %
         PITVAL((STARTP + I)LAND HEX 7F + 1):=PITVAL(STARTP);
    REP;

    CALCP:=(CALCP+2)LAND HEX 7F + 1;
    LASTUP:=PITVAL(CALCP);

UPLOOP:
    CURRENT:=INT((CLOCKFREQ*LASTUP)/(CLOCKFREQ+LASTUP*ACC));

    IF CURRENT <= SPEED THEN
         CURRENT:=SPEED;                     % CURRENT SPEED = SET SPEED %
         VAL RUNF:=SETF;                     % SET THE RUN FLAG %
         RUN(PITVAL,CALCP,STARTP,CURRENT);   % FILL REST OF TABLE WITH RUN VALS %
         GOTO UPEXIT;
    END;

    LASTUP:=CURRENT;                         % REMEMBER CURRENT VALUE %
    PITVAL(CALCP):=CURRENT;                  % WRITE NEW VALUE INTO TABLE %
    CALCP:=CALCP LAND HEX 7F + 1;            % INCREMENT THE POINTER %
    IF CALCP = STARTP THEN
         GOTO UPEXIT;
    ELSE GOTO UPLOOP;
    END;

UPEXIT:
ENDPROC;

% ************************************** %
% RUN AT CONSTANT SPEED **************** %
% ************************************** %

ENT PROC RUN(REF ARRAY INT PITVAL,INT CALCP,STARTP,FREQ);

    % CALCULATES CONTROL TABLE VALUES FOR STEADY OUTPUT FREQUENCY %

RUNLOOP:
    PITVAL(CALCP):=FREQ;                     % OUTPUT A VALUE TO THE TABLE %
    CALCP:=CALCP LAND HEX 7F + 1;            % INCREMENT POINTER %

    IF CALCP = STARTP THEN                   % END OF TABLE YET? %
         GOTO RUNEXIT;
```

```
   ELSE GOTO RUNLOOP;
   END;

RUNEXIT:
ENDPROC;

% ************************************** %
% CALCULATE MODULATION PARAMETERS ******* %
% ************************************** %

ENT PROC PARAM(REF REAL TMFXAMP,TMFXPJ,REF INT TMFXSPEED);
   REAL K2;

   K2:=100.0*TMFXSPEED;                                % PARAM CALC CONSTANT %

   TRAMPTIME:=5*TMPERIOD-THOLD;

   IF TMFXAMP <= 0.01 THEN
        TFXUPSTART:=TFXUPSTOP:=TFXDOWNSTART:=TFXDOWNSTOP:=TMFXSPEED;
        TFXUPINC:=TFXDOWNDEC:=0.0;
   ELSE
        TFXUPSTART:=INT(K2/(100.0-TMFXAMP+TMFXPJ));
        TFXUPSTOP:=INT(K2/(100.0+TMFXAMP));
        TFXUPINC:=(REAL TFXUPSTOP-REAL TFXUPSTART)/REAL TRAMPTIME;

        TFXDOWNSTART:=INT(K2/(100.0+TMFXAMP-TMFXPJ));
        TFXDOWNSTOP:=INT(K2/(100.0-TMFXAMP));
        TFXDOWNDEC:=(REAL TFXDOWNSTOP-REAL TFXDOWNSTART)/REAL TRAMPTIME;
   END;
ENDPROC;

% ************************************** %
% MODULATION RUN VALUES **************** %
% ************************************** %


ENT PROC TMOD(INT TCALCP,TSTARTP,REF INT TFXSPEED);

% FILLS TCONTTABLE WITH MODULATION COUNT VALUES %

   INT CCNOW,CCCOUNT,TARGET,I,VALUE,A1,A2,FLAG;
   REAL RATE,TEMP;

   TCALCP:=(TCALCP+THOLD+DELOFFSET)LAND HEX 7F + 1;
        % DELOFFSET LEAVES SEVERAL VALUES UNTOUCHED TO LEAVE TIME FOR %
        % CALCULATION. ALSO USED IN PROC BAND AND MUST HAVE SAME VALUE %
   CCNOW:=TSTAT(TCALCP);                               % GET CURRENT COND CODE %
   FLAG := 0;

   SWITCH CCNOW OF ONE,TWO,THREE,FOUR;

   ZERO: TARGET:=TFXUPSTOP;                  % SELECT CALCULATION PARAMETERS %
         CCNOW:=1;                           % APPROPRIATE FOR CONDITION CODE %
         RATE:=TFXUPINC;                     % CORRESPONDING TO STARTING POINT %
         TEMP:=TFXSPEED;
         GOTO MODO;
   ONE:  TARGET:=TFXUPSTOP;
         RATE:=TFXUPINC;
```

```
        TEMP := TFXUPSTART;
        CCCOUNT := TRAMPTIME;
        GOTO MOD1;
   TWO: WHILE TSTAT(TCALCP) = 2 DO        % MOVE POINTER TO START OF P-JUMP %
           TCALCP := (TCALCP - 2) LAND HEX 7F + 1;
        REP;
        CCNOW:=1;
        CCCOUNT := 0;
        GOTO MOD3;
   THREE:TARGET:=TFXDOWNSTOP;
        RATE:=TFXDOWNDEC;
        TEMP := TFXDOWNSTART;
        CCCOUNT := TRAMPTIME;
        GOTO MOD1;
   FOUR: WHILE TSTAT(TCALCP) = 4 DO       % MOVE POINTER TO START OF P-JUMP %
           TCALCP:=(TCALCP - 2) LAND HEX 7F + 1;
        REP;
        CCNOW:=3;
        CCCOUNT := 0;
        GOTO MOD3;

MOD0:
   TEMP:=TEMP + RATE;                     % RAMP UP TO TFXUPSTOP IF %
   TPITVAL(TCALCP):=INT(TEMP);            % THIS IS FIRST ENTRY TO %
   TSTAT(TCALCP):=BYTE(CCNOW);            % THIS PROC AFTER START UP %
   IF INT(TEMP) <= TARGET THEN
       TPITVAL(TCALCP):=TARGET;
       FOR I := 1 TO THOLD DO
          TCALCP := TCALCP LAND HEX 7F + 1;
          IF TCALCP = TSTARTP THEN RETURN;END;
          TPITVAL(TCALCP) := TFXDOWNSTART;
          TSTAT(TCALCP) := 2;
       REP;
       CCNOW := 3;
       RATE := TFXDOWNDEC;
       TARGET := TFXDOWNSTOP;
       CCCOUNT := TRAMPTIME;
       TEMP := REAL TFXDOWNSTART;
       GOTO MOD2;
   ELSE TCALCP:=TCALCP LAND HEX 7F + 1;
       IF TCALCP = TSTARTP THEN
           RETURN;                        % EXIT IF TABLE HAS BEEN FILLED %
       END;
       GOTO MOD0;
   END;

MOD1:                                     % OBTAIN VALUES FOR CCCOUNT AND %
                                          % TEMP FOR CURRENT %
   A1 := TPITVAL(TCALCP);                 % STARTING POINT IN TABLE %
   A2 := 0;
   I := (TCALCP - 2) LAND HEX 7F + 1;

   WHILE TPITVAL(I) = A1 AND CCCOUNT # 0 DO
       CCCOUNT := CCCOUNT - 1;            % OF CURRENT LEVEL OF TPITVAL %
       I := (I - 2) LAND HEX 7F + 1;
       A2 := A2 + 1;
   REP;
```

```
   IF CCNOW = 1 THEN
        WHILE INT(TEMP) > A1 AND CCCOUNT # 0 DO
                                              % FIND NUMBER OF 100 MS STEPS FROM %
           TEMP := TEMP + RATE;               % STARTING POINT TO CURRENT LEVEL OF %
           CCCOUNT := CCCOUNT - 1;            % TPITVAL AND CORRESPONDING CCCOUNT %

        REP;
   ELSE
        WHILE INT(TEMP) < A1 AND CCCOUNT # 0 DO
           TEMP := TEMP + RATE;
           CCCOUNT := CCCOUNT - 1;
        REP;
   END;

   TEMP := TEMP + (A2 * RATE);
   GOTO MOD3;

MOD2:

   TCALCP:=TCALCP LAND HEX 7F + 1;

   IF TCALCP = TSTARTP THEN
        RETURN;
   END;

   TEMP:= TEMP + RATE;                        % UPDATE CONTROL TABLE VALUE %
   TSTAT(TCALCP):=BYTE(CCNOW);                % UPDATE COND CODE TABLE %
   TPITVAL(TCALCP):=INT TEMP;

   CCCOUNT:=CCCOUNT-1;

MOD3:
   IF CCCOUNT > 0 THEN GOTO MOD2;END;

   IF CCNOW = 1 THEN

      FOR I := 1 TO THOLD DO
        TCALCP:=TCALCP LAND HEX 7F + 1;
        IF TCALCP = TSTARTP THEN RETURN;END;
        TPITVAL(TCALCP) := TFXDOWNSTART;
        TSTAT(TCALCP) := 2;
      REP;
      CCNOW := 3;
      RATE := TFXDOWNDEC;
      TARGET := TFXDOWNSTOP;
      TEMP := REAL TFXDOWNSTART;
      CCCOUNT := TRAMPTIME;

   ELSE                        % IF CCNOW := 3 %

      FOR I := 1 TO THOLD DO
        TCALCP := TCALCP LAND HEX 7F +1;
        IF TCALCP = TSTARTP THEN RETURN;END;
        TPITVAL(TCALCP) := TFXUPSTART;
        TSTAT(TCALCP) := 4;
      REP;
      CCNOW := 1;
      RATE := TFXUPINC;
```

```
        TARGET := TFXUPSTOP;
        TEMP := REAL TFXUPSTART;
        CCCOUNT := TRAMPTIME;

    END;
    FLAG := 0;
    GOTO MOD2;

ENDPROC;

% ************************************* %
% GO TO/FROM BANDING ******************* %
% ************************************* %

ENT PROC BAND(REF INT TFXSPEED,FLAG1,FLAG2,REF REAL RATE,INT TCALCP,TSTARTP);

% FILLS TCONTTABLE WITH RAMP VALUES ON ENTRY TO OR EXIT FROM A BANDING POINT %


    INT J,K,DIFF;
    REAL TEMP,SLOPE;

    IF TEMPFLAG # 0 AND TSTAT(TCALCP) # 0 THEN    % IE AT NEW MODULATION POINT %
         VAL FLAG2 := FLAG2 NEV 1;          % FOR THIS CONDITION TCALCP MUST %
         VAL FLAG1 := FLAG1 NEV 1;          % NOT BE MOVED BACK 3 PLACES, SO %
         TEMPFLAG := 0;                     % GO TO TMOD DIRECTLY %
         GOTO BAND4;
    END;

    SLOPE := REAL TPITVAL(TCALCP)*(1-(CLOCKFREQ/
                        (RATE*REAL TPITVAL(TCALCP)+CLOCKFREQ)));

    TEMP := TPITVAL(TCALCP);
    DIFF := TPITVAL(TCALCP) - TFXSPEED;           % RAMP UP OR DOWN? %
    J := 2;

    IF DIFF < 0 THEN GOTO BAND0;                  % RAMP DOWN %
    ELSEIF DIFF > 0 THEN                          % RAMP UP %
         SLOPE := -SLOPE;
         GOTO BAND1;
    ELSEIF DIFF = 0 THEN GOTO BAND3;              % END OF RAMP %
    END;

BAND0:                                            % RAMPING DOWNWARDS %
    WHILE TPITVAL(TCALCP) <= TFXSPEED AND J >= 0 DO
         TSTAT(TCALCP) := 0;
         TPITVAL(TCALCP) := TPITVAL(TSTARTP);     % OUTPUT CURRENT VALUE FOR %
         J := J - 1;                              % J * 100 MS TO LIMIT MAX %
         TCALCP := TCALCP LAND HEX 7F + 1;        % ACCELERATION %
    REP;

LOOP1:
    TEMP := TEMP + SLOPE;
    TPITVAL(TCALCP) := INT TEMP;
    TSTAT(TCALCP) := 0;
    TCALCP := TCALCP LAND HEX 7F + 1;
    IF TCALCP = TSTARTP THEN RETURN;END;
    IF INT TEMP >= TFXSPEED THEN
```

```
        TEMP := TFXSPEED;
        GOTO BAND3;
    END;
    GOTO LOOP1;

BAND1:                                                    % RAMPING UPWARDS %
    WHILE TPITVAL(TCALCP) >= TFXSPEED AND J >= 0 DO
        TSTAT(TCALCP) := 0;
        TPITVAL(TCALCP) := TPITVAL(TSTARTP);
        J := J - 1;
        TCALCP := TCALCP LAND HEX 7F + 1;
    REP;

LOOP2:
    TEMP := TEMP + SLOPE;
    TPITVAL(TCALCP) := INT TEMP;
    TSTAT(TCALCP) := 0;
    TCALCP := TCALCP LAND HEX 7F + 1;
    IF TCALCP = TSTARTP THEN RETURN;END;
    IF INT TEMP <= TFXSPEED THEN
        TEMP := TFXSPEED;
        GOTO BAND3;
    END;
    GOTO LOOP2;

BAND3:
    IF TEMPFLAG = 1 THEN          % TEMPFLAG IS SET THE FIRST TIME TPITVAL %
        VAL FLAG2 := FLAG2 NEV 1;         % REACHES OPERATIONAL SPEED. RESET ON SE
        VAL FLAG1 := FLAG1 NEV 1;         % PASS THROUGH BAND PROC %
        TEMPFLAG := 0;
    ELSE
        TEMPFLAG := 1;
    END;

      TSTAT(TCALCP) := 0;         % ENSURES MOD CALC STARTS FROM MEAN SPEED %
      TCALCP := (TCALCP - (2 + DELOFFSET) - THOLD) LAND HEX 7F + 1;
                          % MOVE TCALCP BACK 3 + THOLD PLACES %
                          % SINCE TMOD MOVES IT FORWARD %
                          % 3 + THOLD PLACES %
BAND4:
    TMOD(TCALCP,TSTARTP,TFXSPEED);
    RETURN;

ENDPROC;
```

```
TITLE   SMT COMMON LET ITEMS, MODES AND DATA-BRICKS
                                    15-AUG-80
        UPDATED FOR LONG 8086 BY BRIAN DOBBING  2-11-81
        MODIFIED FOR INTEL 88/25 BY TERENCE KIRK 7-9-83 ;
% THESE DECLARATIONS ARE PARAMETERISED ON AMESG,SYSIO0                 %
% SYSIO1,SYSIO2,REALSO                                                  %

LET REFAB=      REF ARRAY BYTE;
LET NL=10;  LET SP=32;  LET TAB=9;
LET EOM=3;  LET ENQ=5;  LET BELL=7;
LET VT=11;  LET CR=13;  LET FF=12;
LET EOS=        128;
LET TRUE=1;
LET FALSE=0;
LET GO= 0;
LET NOGO=       64;
LET PWFEV=      -10;
LET CTLAEV=-15;                         % CTLA TASK EVENT               %
LET CLKTASKNO = -2;                     % CLOCK TASK NUMBER             %
LET SETTASKNO = -3;                     % SET TASK NUMBER               %
LET CTLATASKNO = -4;                    % CTLA TASK NUMBER              %
LET NSFAC =16;                  % NUMBER OF SYSTEM FACILITIES   %
LET NSTSK = 6;                  % NUMBER OF SYSTEM TASKS        %
LET NSEV =16;                   % NUMBER OF SYSTEM EVENTS       %
LET FACEV=-11;                  % EVENT WAKES UP SECURED TASKS  %
LET STOPP=OCT 100;
LET SUSP=OCT 200;
LET WTNG=2;
LET NOTSTOP=OCT 277;
LET EVQLEN=16;          % MUST BE AN INTEGRAL POWER OF TWO      %
LET NEVQS=15;           % EVQLEN MINUS ONE                      %
LET TTACHOEV=3;         % TRAV TACHO EVENT.(TEK) %
LET WTACHOEV=4;         % WINDER TACHO EVENT.(TEK) %
MODE DELREC ( INT TIMUP, TASK, REF DELREC NXT );


MODE GETTIME (INT DD,MM,YY,H,M,S);

SVC DATA RRSIO; PROC () BYTE IN; PROC (BYTE) OUT; ENDDATA;
SVC DATA RRERR; LABEL ERL; INT ERN; PROC (INT) ERP; ENDDATA;
SVC DATA RRSED; BYTE TERMCH, IOFLAG; ENDDATA;
SVC DATA RRUSG; INT LINE, USAGE; ENDDATA;



EXT DATA TASKDATA;
INT CURTASK,CURTEX,TASKLOCK,NXTCUR,EVIN,EVOUT;
DELREC ADEL;
REF DELREC FRPTR;
BYTE HIPRI, LOPRI;
ARRAY (16) INT  TIMEOUT;
ARRAY (16) DELREC DEL;
ARRAY (16) STACK CELL;
ARRAY (16) BYTE EVFAC,STAT,XSTAT,WTCHN;
ARRAY (1,32) INT EVBITS;
ARRAY (16) REF BYTE STATADS;
ARRAY (EVQLEN) INT EVQ;
ARRAY (32) BYTE FACILITY,FACTOTSK;
```

```
EXT PROC (BYTE) OUTTTY;
EXT PROC () BYTE INTTY;



% I/O FORMATTING PROCS %

EXT PROC (INT) HWRT, IWRT, SPS, NLS;
EXT PROC (INT,INT) IWRTF;
EXT PROC (FRAC) FWRT;
EXT PROC () INT IREAD, HREAD, ZREAD;
EXT PROC () FRAC FREAD;
EXT PROC (REFAB) TWRT;


EXT PROC (REAL,INT,INT) RWRTF;
EXT PROC () REAL RREAD;


TITLE SECOND PRELUDE FILE FOR LONG ADDRESSING SMT+ ON INTEL 8086.
EXISTS FOR THE BENEFIT OF SMTBAS.RMP, WHICH DOES NOT WANT THESE EXT
        DEFINITIONS IN ITS PRELUDE;

EXT PROC () CHANGE, HLOCK, HUNLOCK, PTORTL, RETFIN ;
EXT PROC (INT) RRGEL;
TITLE
                SMT-PLUS OPERATING SYSTEM FOR LONG-ADDRESSING ON 8086
      USER TASK INITIALISATION AND DEVICE-SPECIFIC CODE
      **** MODULE SMTU1 ****;
OPTION (1);

% THIS MODULE IS CONFIGURED ON SYSIOO, MAGIC, AMESG               %

%USER INIT PROCS CALLED BY USERINITS%



ENT PROC USERINITS();
   % USER EDITED INITIALISATION PROCEDURE                                  %
   % CALLED BY STARTUP TASK BEFORE INTERRUPTS ARE ENABLED                  %

   % THIS CODE SECTION CONTAINS DEVICE-SPECIFIC INITIALISATIONS.            %
   % IT IS INCLUDED HERE TO ALLOW THE USER TO AMEND THE VALUES IF           %
   % NECESSARY.                                                             %

    CODE 200,0;
        EXTRN   RP@CLOCKINT:NEAR
        EXTRN   RXINTERRUPT:NEAR
        EXTRN   LINKRXINT:NEAR
        EXTRN   LINKTXINT:NEAR

        IC@PORTA         EQU      0C0H
        IC@PORTB         EQU      0C2H
```

```
IC@ICW1          EQU    13H
IC@ICW2          EQU    20H
IC@ICW4          EQU    0FH
IT@CONTROL@PORT  EQU    0D6H
ITXCONTROLXPORT  EQU    0BFH
IT@CTR0@PORT     EQU    0D0H
IT@CTR1@PORT     EQU    0D2H      ;Winder tacho.
IT@CTR2@PORT     EQU    0D4H
ITXCTR0XPORT     EQU    0B8H
ITXCTR1XPORT     EQU    0BAH
ITXCTR2XPORT     EQU    0BCH
IT@COM2          EQU    034H      ;System clock.
IT@C1M0          EQU    070H      ;Winder tacho.
IT@C2M3          EQU    0B6H      ;USART clock.
ITXCOM0          EQU    030H      ;Traverse tacho.
ITXC1M3          EQU    076H      ;Winder drive.
ITXC2M3          EQU    0B6H      ;Traverse drive.
IT@50HZ          EQU    24576D    ;Crystal divider to give 50Hz.
IT@4800          EQU    010H      ;For 4800 Baud.
IT@9600          EQU    008H      ;For 9600 Baud.
IT@RDCLK         EQU    000H      ;Counter latch mode.
SIOXDATAXPORT    EQU    0A0H
SIOXSTATXPORT    EQU    0A2H
SIOXCRTXCMD      EQU    037H      ;Enable Rx & Tx and reset error flags.
SIOXCRTXMODE     EQU    04EH      ;8 bits+ 1 stop,no parity,Baud=9600.
SIOXRESET        EQU    040H
SIO@DATA@PORT    EQU    0D8H
SIO@STAT@PORT    EQU    0DAH
SIO@CRT@CMD      EQU    037H
SIO@CRT@MODE     EQU    04EH      ;8 bits + 1 stop,no parity,baud=9600.
SIO@RESET        EQU    040H
PIO@CONTROL@REG  EQU    0CEH      ;Address of 8255 cntrl reg
PIO@MODE@CONF    EQU    090H      ;Mode 0 A=input ,B & C=output
PC3SET           EQU    007H      ;Set port C bit 3.
PC3RES           EQU    006H      ;Reset port C bit 3.

INTSEG           EQU    000H      ;Data Segment of interrupt table.
INT21            EQU    084H      ;INT21 pointer to off board address.
INT22            EQU    088H      ;INT22 pointer to clock service.
INT23            EQU    08CH      ;INT23 pointer to trav tacho routine.
INT24            EQU    090H      ;INT24 pointer to wind tacho routine.
INT25            EQU    094H      ;INT25 pointer to SBX Tx routine.
INT26            EQU    098H      ;INT26 pointer to SBC Rx routine.
INT27            EQU    09CH      ;INT27 pointer to SBX Rx routine.

                 ;initialise PIT channel 0 for 50HZ clock interrupts.
MOV     DX,IT@CONTROL@PORT
MOV     AL,IT@COM2
OUT     DX,AL
MOV     DX,IT@CTR0@PORT
MOV     AX,IT@50HZ
OUT     DX,AL
MOV     AL,AH
OUT     DX,AL

                 ;initialise iSBX PIT channels 1 and 2 into mode 3
                 ;for winder and traverse drivers respectively.
MOV     DX,ITXCONTROLXPORT
```

```
MOV     AL,ITXC1M3
OUT     DX,AL
MOV     AL,ITXC2M3
OUT     DX,AL
                ;initialise iSBC  PIT channel 1 for winder tacho
                ;pulse counting, and iSBX PIT channel 0 for traverse
                ;tacho pulse counting.(Count value loaded by service
                ; routines in WDRV.RTL and TDRV.RTL)
                ;(TEK)
MOV     AX,IT@C1M0
OUT     IT@CONTROL@PORT,AL
MOV     AX,ITXCOMO
OUT     ITXCONTROLXPORT,AL

                ;set up PIC interrupt vector 7 for iSBX USART Rx.
                ;(RS422 Multidrop link to host).
MOV     BX,INTSEG
MOV     ES,BX
MOV     ES:INT27,OFFSET LINKRXINT
MOV     ES:INT27+2,CS

                ;set up PIC interrupt vector 6 for iSBC USART Rx.
                ;(RS232 serial link to local VDU);
MOV     ES:INT26,OFFSET RXINTERRUPT
MOV     ES:INT26+2,CS

                ;set up PIC interrupt vector 5 for iSBX USART Tx.
MOV     ES:INT25,OFFSET LINKTXINT
MOV     ES:INT25+2,CS

                ;set up PIC interrupt vector 4 for winder tacho.(TEK)
MOV     ES:INT24,OFFSET RP@24SERVICE
MOV     ES:INT24+2,CS

                ;set up PIC interrupt vector 3 for traverse tacho.
MOV     ES:INT23,OFFSET RP@23SERVICE
MOV     ES:INT23+2,CS

                ;set up PIC interrupt vector 2 for system clock.
MOV     ES:INT22,OFFSET RP@CLOCKINT
MOV     ES:INT22+2,CS

                ;set up PIC interrupt vector 1 for address trap.
MOV     ES:INT21,OFFSET RP@BADMEM
MOV     ES:INT21+2,CS

                ;Initialise Interrupts.
MOV     DX,IC@PORTA
MOV     AL,IC@ICW1
OUT     DX,AL
MOV     DX,IC@PORTB
MOV     AL,IC@ICW2
OUT     DX,AL
MOV     AL,IC@ICW4
OUT     DX,AL
MOV     AL,00000001B     ;unmask all used interrupts.
OUT     DX,AL
```

```
                        ;initialise USART's:

                        ;First :initialise SBC PIT channel 2 for USART clock.
       MOV     DX,IT@CONTROL@PORT
       MOV     AX,IT@C2M3
       OUT     DX,AL
       MOV     DX,IT@CTR2@PORT
       MOV     AX,IT@9600          ;TO GIVE 9600 BAUD CLOCK.
       OUT     DX,AL
       MOV     AL,AH
       OUT     DX,AL

                        ;Second :Initialise SBX USART.

                        ;RESET USART:-
       CLI
       MOV     DX,SIOXSTATXPORT
       MOV     AL,0                          ;TO CLEAR USART.
       OUT     DX,AL
       MOV     CX,2                          ;DELAY
LP2:   LOOP    LP2
       OUT     DX,AL
       MOV     CX,2                          ;DELAY
LP3:   LOOP    LP3
       OUT     DX,AL
       MOV     CX,2                          ;DELAY
LP4:   LOOP    LP4
       OUT     DX,AL
       MOV     CX,2                          ;DELAY
LP5:   LOOP    LP5
       MOV     AL,SIOXRESET
       OUT     DX,AL

       MOV     CX,200
DELAYABIT:
       NOP
       LOOP    DELAYABIT           ;USART SETTLING TIME


                        ;Configure SBX USART.
       MOV     AL,SIOXCRTXMODE
       OUT     DX,AL
       MOV     CX,2                ;DELAY
LP6:   LOOP    LP6
       MOV     AL,SIOXCRTXCMD
       OUT     DX,AL

                        ;Third :Initialise SBC USART.

                        ;RESET USART:-
       CLI
       MOV     DX,SIO@STAT@PORT
       MOV     AL,0
       OUT     DX,AL
       MOV     CX,2
LP7:   LOOP    LP7
       OUT     DX,AL
       MOV     CX,2
```

```
LP8:    LOOP     LP8
        OUT      DX,AL
        MOV      CX,2
LP9:    LOOP     LP9
        OUT      DX,AL
        MOV      CX,2
LP10:   LOOP     LP10
        MOV      AL,SIO@RESET
        OUT      DX,AL

        MOV      CX,200
WAITAWHILE:
        NOP
        LOOP     WAITAWHILE


                           ;Configure SBC USART.
        MOV      AL,SIO@CRT@MODE
        OUT      DX,AL
        MOV      CX,2
LP11:   LOOP     LP11
        MOV      AL,SIO@CRT@CMD
        OUT      DX,AL


                           ;INITIALLISE PIO 8255

        MOV      AL,PIO@MODE@CONF          ;MODE 0 PORT A=INPUT B,C=OUTPUT
        OUT      PIO@CONTROL@REG,AL        ;ADDRESS OF 8255 CONTROL REG

   ;This section clears the off board memory latch.
        MOV      AL,PC3RES
        OUT      PIO@CONTROL@REG,AL
        MOV      AL,PC3SET
        OUT      PIO@CONTROL@REG,AL

        STI
   *RTL
ENDPROC;


% USER HARDWARE TASKS :-                          %

% ******************************************** %
% OFF BOARD MEMORY ADDRESS TRAP ************** %
% ******************************************** %

PROC INTSERV21();
   CODE 20,0;
RP@BADMEM:
   ;This section clears the off board memory latch.
        MOV      AL,PC3RES
        OUT      PIO@CONTROL@REG,AL
        MOV      AL,PC3SET
        OUT      PIO@CONTROL@REG,AL
   ; set off board memory error number thirty.
        POP      CX        ; CLEAR RETURN ADDRESS...
        POP      CX        ; ... NOT GOING BACK
```

```
        POPF                ; RESTORE FLAGS
        MOV     CX,1EH      ; RRGEL(30.)
        PUSH    CX
        CALL    RP@RRGEL
                            ; DOSEN'T RETURN
    *RTL;
ENDPROC;


% ****************************************** %
% TRAVERSE TACHO SERVICING ROUTINE ************ %
% ****************************************** %


ENT DATA TRAVSTOP;                % DATA BRICK FOR INTSERV23 %
    INT TSTOP,TENDFRAC,TCNT;
ENDDATA;

PROC INTSERV23();          % INT SERVICE ROUTINE FOR TRAVERSE TACHO (TEK) %
   CODE 0,0;
RP@23SERVICE:
        CALL    FAR PTR RP@PTORTL
   ;Read system clock counter on the fly to get fraction of 'NOW'.
        MOV     AL,IT@RDCLK
        OUT     IT@CONTROL@PORT,AL
        IN      AL,IT@CTR0@PORT           ;GET LSB.
        MOV     CL,AL
        IN      AL,IT@CTR0@PORT           ;GET MSB.
        MOV     CH,AL
        MOV     *TENDFRAC/TRAVSTOP,CX
*RTL;
   TSTOP:=NOW;
   CODE 0,0;
   ;Reload interrupt counter.
        MOV     AX,*TCNT/TRAVSTOP
        MOV     DX,ITXCTR0XPORT
        OUT     DX,AL
        MOV     AL,AH
        OUT     DX,AL
   *RTL;
   QEVRRET(TTACHOEV);
ENDPROC;


% ****************************************** %
% WINDER TACHO SERVICING ROUTINE ************* %
% ****************************************** %


ENT DATA WINDSTOP;                %  DATA BRICK FOR INTSERV24    %
        INT WSTOP,WENDFRAC,WCNT;
ENDDATA;

PROC INTSERV24();          % INT SERVICE ROUTINE FOR WINDER TACHO (TEK) %
        CODE 20,0;
RP@24SERVICE:
        CALL    FAR PTR RP@PTORTL
   ;Read system clock counter on the fly to get fraction of 'NOW'
```

```
        MOV       AL,IT@RDCLK
        OUT       IT@CONTROL@PORT,AL
        IN        AL,IT@CTR0@PORT               ;GET LSB.
        MOV       CL,AL
        IN        AL,IT@CTR0@PORT               ;GET MSB.
        MOV       CH,AL
        MOV       *WENDFRAC/WINDSTOP,CX
    *RTL;
    WSTOP:=NOW;
    CODE 0,0;
    ;Reload interrupt counter.
        MOV       AX,*WCNT/WINDSTOP
        MOV       DX,IT@CTR1@PORT
        OUT       DX,AL
        MOV       AL,AH
        OUT       DX,AL
    *RTL;
    QEVRRET(WTACHOEV);
ENDPROC;

%ENT PROC UPWFAIL();%    % NOT IMPLEMENTED FOR 8086       %
    % USER EDITED PROCEDURE WHICH IS CALLED ON POWER FAIL RESTART.       %
    % THERE ARE SEVERAL POWER FAIL RESTART MECHANISMS WHICH CAN BE       %
    % IMPLEMENTED DEPENDING ON THE CODE OF THIS PROCEDURE:-              %

    %  (1) IF THIS PROCEDURE IS NULL, A POWER FAIL RESTART WILL          %
    %      SIMPLY START ALL USER TASKS AS FOR A NORMAL SYSTEM            %
    %      STARTUP (COLD START FROM ZERO).                               %

    %  (2) IF THIS PROCEDURE ENDS BY CALLING 'RETEV' , THIS WILL         %
    %      ALLOW ALL TASKS TO CONTINUE FROM THE POINT THEY HAD           %
    %      REACHED WHEN POWER FAIL OCCURRED.                             %

    %  (3) THIS PROCEDURE RUNS IN A LIMITED HTASK ENVIROMENT, AND        %
    %      PROCEDURE QEV MAY BE USED TO SET EVENTS. IT IS NOT PERMITTED %
    %      TO USE START, AND STOP, BUT EXT DATA TASKDATA MAY BE          %
    %      MANIPULATED DIRECTLY, AS IN CLOCKINT - SMTB1. IT IS THE       %
    %      RESPONSIBILITY OF THE USER TO CHECK WHETHER THIS IS           %
    %      PERMISSIBLE IN HIS SYSTEM. IT WILL USUALLY BE PREFERABLE      %
    %      TO USE AN STASK WAITING ON PWFEV, THE POWER FAIL EVENT TO     %
    %      TIDY UP.                                                      %

    %  (4) THERE ARE TWO SYSTEM FEATURES WHICH MAY BE USEFUL ON          %
    %      POWER FAIL/RESTART:-                                          %
    %      (A) PWFLAG IN PWFDATA IS SET NON ZERO AFTER A POWER FAILURE. %
    %          (NOTE: THE SYSTEM NEVER ZEROS THIS FLAG.)                 %
    %      (B) EVENT NUMBER-10(NEGATIVE BECAUSE IT IS A RESERVED SYSTEM %
    %          EVENT), WILL BE SET AFTER A POWER FAIL/RESTART.           %
%ENDPROC;%
```

```
        ;M       STUPCD
        ;X       R01
        ;O       ;
        ;T        START UP CODE FOR ROM VERSION OF ICU
        ;T          WRITTEN 22-2-84 DRT
        ;T          *** STARTUP.RTL ***;
        ;O       ;
        ;L=      0
        ;B       STARTUP
        ;L=      17
        ;W

RTL@PROCS        SEGMENT WORD PUBLIC 'PCLASS'

RTL@SPOOL        LABEL   WORD
;

; START UP CODE FOR ROM VERSION OF ICU
;   WRITTEN 22-2-84 DRT
;   *** STARTUP.RTL ***

;
; THIS MODULE HAD TO BE EDITED FROM AN RTL COMPILATION ASM FILE
; HENCE USE THIS VERSION NOT THE RTL. IT INCLUDES THE XRF INFO

;

RTL@PROCS        ENDS


RTL@DATA         SEGMENT WORD PUBLIC 'DCLASS'
RTL@DATA         ENDS


RTL@PROCS        SEGMENT WORD PUBLIC 'PCLASS'
        ASSUME   DS:RTL@DATA, CS:RTL@PROCS
RP@STARTUP       PROC    FAR
; CODE INSERT--PARAMETERS 00000H AND 00000H
      EXTRN RP@RRXEQ:FAR
      JMP FAR PTR RP@RRXEQ

;
RTL@PROCS        ENDS


        END
```

```
.;COMMAND FILE TO COMPILE ETC ICUNIT SYSTEM FORMAT IS -
.;@TOTSYS MODULE,SOURCE LISTING,CODE LISTING OR INPUT CAN
.; BE DONE INTERACTIVLY.
.; FILENAME IS TOTSYS.CMD 29-NOV-83 DRT
.; MODIFIED 21-DEC-83 TEK
.;
.;
.ENABLE SUBSTITUTION
.ENABLE GLOBAL
.GOTO QRY1
.QRY:
; No! please re-enter module name
.QRY1:
.ASKS NAM "Module name, DATAPREL, ALL or SMT"
.IF NAM = "" .GOTO QRY
.IF NAM = "SMT" .GOTO 15
.GOSUB 102
.IF NAM <> "DATA" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 1
.SETS MOD "DATA"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.1:
.IF NAM <> "WCONT" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 2
.SETS MOD "WCONT"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.2:
.IF NAM <> "TCONT" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 3
.SETS MOD "TCONT"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.3:
.IF NAM <> "COMPROC" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 4
.SETS MOD "COMPROC"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.4:
.IF NAM <> "WDRV" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 5
.SETS MOD "WDRV"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.5:
.IF NAM <> "TDRV" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 6
.SETS MOD "TDRV"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
```

```
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.6:
.IF NAM <> "RATIO" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 7
.SETS MOD "RATIO"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.7:
.IF NAM <> "WTACHO" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 8
.SETS MOD "WTACHO"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.GOSUB 13
.IF <EXSTAT> = 2 .GOSUB 101
.8:
.IF NAM <> "INVSQ" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 9
.SETS MOD "INVSQ"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.9:
.IF NAM <> "SQMON" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 10
.SETS MOD "SQMON"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.10:
.IF NAM <> "INTSTD" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 11
.SETS MOD "INTSTD"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/SS/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.11:
.IF NAM <> "DBUF" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 12
.SETS MOD "DBUF"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.12:
.IF NAM <> "OCP1" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 14
.SETS MOD "OCP1"
SJR 'MOD','SLST',SY:'MOD'=DATAPREL,'MOD'/TA:S3100/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.14:
.IF NAM <> "LINKDA" .IF NAM <> "DATAPREL" .IF NAM <> "ALL" .GOTO 15
.SETS MOD "LINKDA"
```

```
SJR 'MOD',SLST,SY:'MOD'=DATAPREL,'MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.15:
.ASK SYS "Do you want to remake SMT "
.IFF SYS .GOTO 20
SET /UIC=[341,1]
.GOTO QRY3
.QRY2:
; No! Please re-enter module name
.QRY3:
.ASKS NAM "Module name or ALL"
.IF NAM = "" .GOTO QRY2
.GOSUB 102
.IF NAM <> "MSMTU1" .IF NAM <> "ALL" .GOTO 16
.SETS MOD "MSMTU1"
SJR 'MOD','SLST',SY:'MOD'='MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.16:
.IF NAM <> "MSMTCLA" .IF NAM <> "ALL" .GOTO 17
.SETS MOD "MSMTCLA"
SJR 'MOD','SLST',SY:'MOD'='MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.17:
.IF NAM <> "MSMTPIO" .IF NAM <> "ALL" .GOTO 18
.SETS MOD "MSMTPIO"
SJR 'MOD','SLST',SY:'MOD'='MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.18:
.IF NAM <> "LINKDR" .IF NAM <> "ALL" .GOTO 19
.SETS MOD "LINKDR"
SJR 'MOD','SLST',SY:'MOD'='MOD'/CN:F/NA:'MOD'
.IF <EXSTAT> = 2 .GOSUB 100
ASM 'MOD','CLST'='MOD'.XRF,'MOD'
.IF <EXSTAT> = 2 .GOSUB 101
.GOSUB 13
.19:
LLM @SMTROM
.GOSUB 13
SET /UIC=[341,2]
.IF <EXSTAT> <> 2 .GOTO 20
.EXIT
.20:
LLM @ROMLNK
.IF <EXSTAT> = 2 .GOTO 21
.GOSUB 13
.EXIT
.21:
; Link failed.
.RETURN
```

```
.GOSUB 13
.EXIT
.13:
PIP *.ASM;*/DE
PIP *.XRF;*/DE
PIP *.*/PU/LD
.RETURN
.100:
;'<EXSTAT>' 'MOD' Compilation failure
.RETURN
.101:
;'<EXSTAT>' 'MOD' Assembly failure
.RETURN
.102:
.ASKS SLST " Source listing Y/N "
.ASKS CLST " Code listing Y/N "
.IF SLST <> "Y" .SETS SLST ""
.IF CLST <> "Y" .SETS CLST ""
.IF SLST = "Y" .SETS SLST "LP.SRC/SP"
.IF CLST = "Y" .SETS CLST "LP.LST/SP"
.RETURN
```

APPENDIX H

1- SBC SCHEMATIC
2- SBX SCHEMATIC
3- I/O BOARD SCHEMATIC
4- I/O BOARD COMPONENT LAYOUT
5- POWER DISTRIBUTION SCHEMATIC
6- COMMUNICATIONS SCHEMATIC
7- WIRING SCHEDULES
8- ICU DRAWER FIELD WIRING CONNECTIONS
9- SBC JUMPER CONNECTIONS

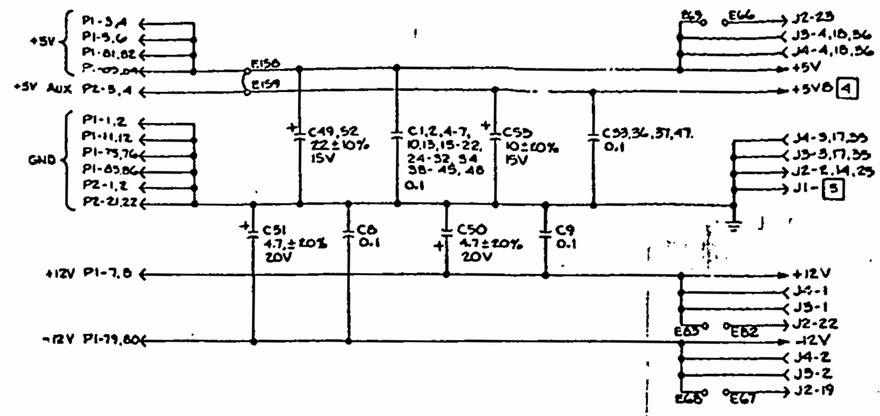


TABLE A		
EPROM SIZE SELECT JUMPERS		
TYPE	DECODE	J6
2716	NONE	1-14, 2-13, 3-12
2732	E90-E91	1-14, 5-12, 6-9
2764	E92-E93	1-14, 6-9
2754	E90-E91 E92-E93	1-14, 6-9, 7-8

TABLE B		
DEVICE TYPE	PORT NO.	FUNCTION
8251A	00D8	DATA
	00DA	CONTROL
8253-5	00D0	COUNTER#0
	00D2	COUNTER#1
	00D4	COUNTER#2
	00D6	CONTROL
8255A-5	00C8	PORT A
	00CA	PORT B
	00CC	PORT C
	00CE	CONTROL
8259A	00C0	REG#1
	00C2	REG#2

POWER, GROUND AND SPARE GATE LOCATOR CHART							
REFERENCE DESIGNATION	DEVICE TYPE	POWER PINS					SPARE GATES OUTPUT PINS
		GND	+5V	+12V	-12V	+5VB	
U51, U67, 52, 68	2168R	10				20	
U69, 70	3625A-1	9	18				
U30	74LS00	7	14				U30-8
U1, 18, 37	74S00	7	14				U18-3,6,8, U37-8
U27	74S02	7	14				U27-4,10
U6	74LS04	7	14				
U29, 38	74S04	7	14				U29-8, U38-4,8,10,12
U42	7406	7	14				U42-4,8
U2, 28	74LS08	7	14				U2-3,8, U28-6,11
U47, 54, 71	74S08	7	14				[illegible]
U19, 39	74S10	7	14				U39-6,12
U3	74S32	7	14				U3-3,11
U55	74S37	7	14				U55-11
U20	74LS74	7	14				
U43	74S112	8	16				
U5	74LS123	8	16				U5-13
U26, 40, 41, 53	74S138	8	16				
U13	74LS163	8	16				
U12	74S163	8	16				
U4	74S175	8	16				
U58, 59, 60, 61	74LS240	10	20				
U36	74S260	7	14				
U46, 66	74S373	10	20				
U14	75188	7		14	1		
U15	75189	7	14				
U32	8088	1,20	40				
U31	8097	8	16				U31-14
U16, 17	8224	8	16				
U24	8251A	4	26				
U23	8253-5	12	24				
U22	8255A-5	7	26				
U21	8259A	14	28				
U45	8284A	9	18				
U25	8286	10	20				
U7, 63	8287	10	20				
U44, 57	8288	10	20				
U56	8289	10	20				
U49	74LS00	7				14	U49-8
XU8, 9, 10, 11	SOCKET	7	14				
U30, 62	SPARE	7,8	16				
U48	74LS373	10	20				

NOTES: UNLESS OTHERWISE SPECIFIED
1. CAPACITANCE VALUES ARE IN MICROFARADS, +80% -20%, 50V
2. RESISTANCE VALUES ARE IN OHMS, 1/4W, 5%.
3. ALL RESISTOR PAKS ARE 10K.
4 "+5VB" INDICATES BATTERY +5V.
5 ALL ODD NUMBER PINS ON J1 ARE GROUND.
6 EPROM SIZE SELECT CONNECTIONS ARE AS SHOWN IN TABLE A.
7 PORT FUNCTIONS ARE AS SHOWN IN TABLE B
8 REMOVED
9 U33, U34, U52, U64, U65 AND U68 ARE CUSTOMER INSTALLED.

REF. DESIG.	
LAST USED	NOT USED
U71	U35
Y3	E185, 186
C55	C11, 14, 23
RP3	
R34	
DS1	
CR2	
J8	
E187	

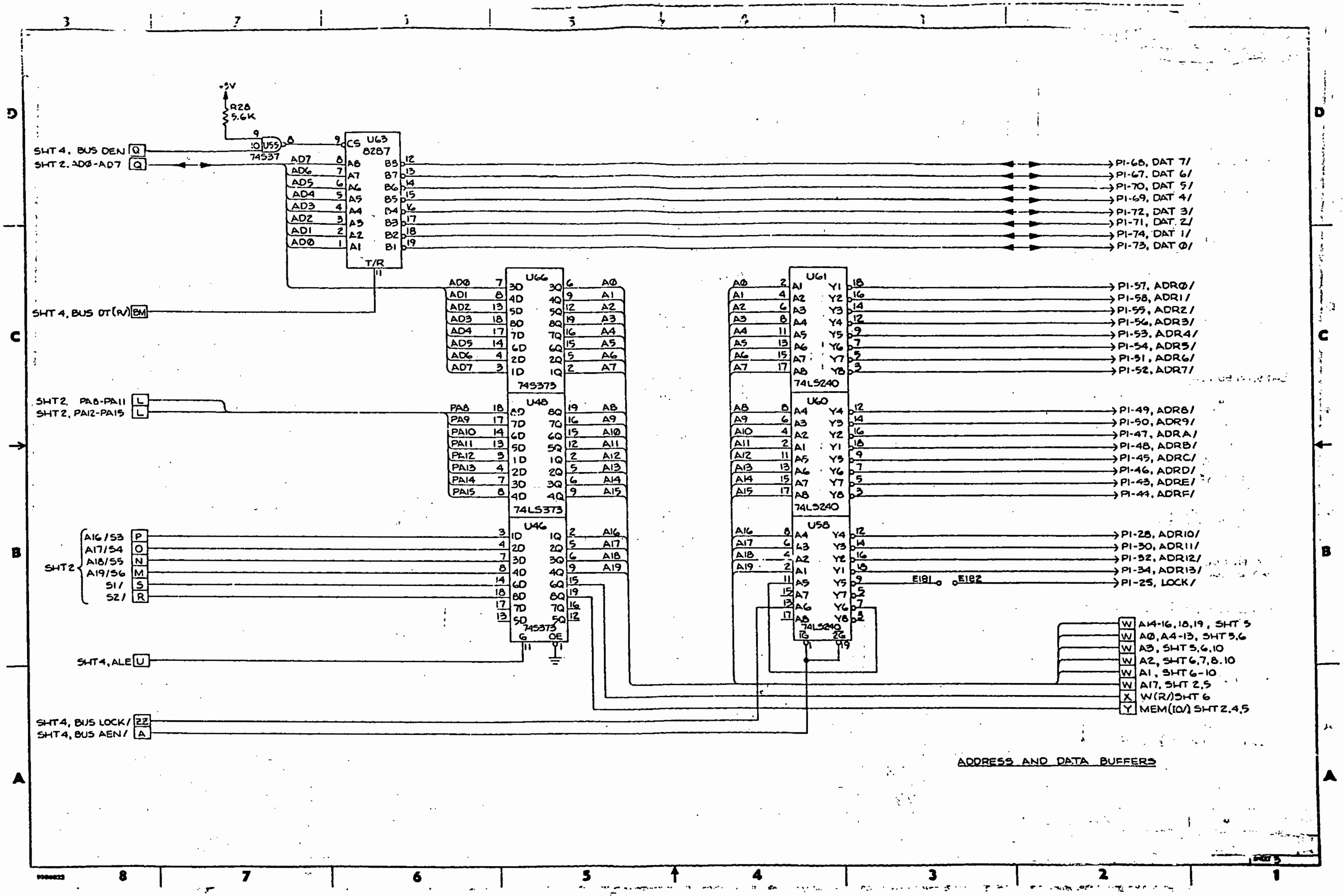
ITEM NO. | DESCRIPTION
QUANTITY PER DASH NO. | PARTS LIST

UNLESS OTHERWISE SPECIFIED:
1. DIMENSIONS ARE IN INCHES.
2. BREAK ALL SHARP EDGES.
3. DO NOT SCALE DRAWING.
4. TOLERANCES:
ANGLES ±
.XX ±
.XXX ±

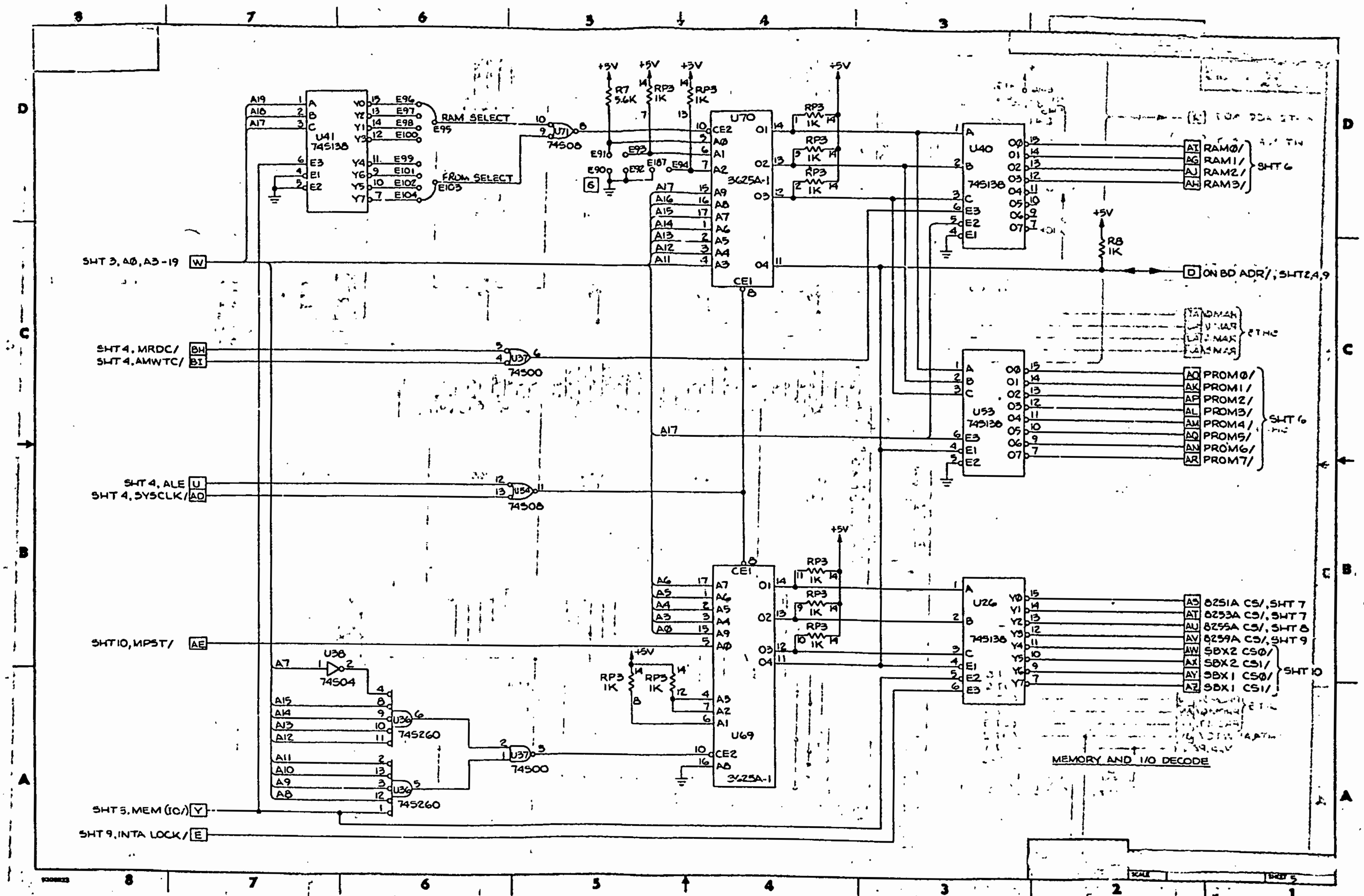
TITLE SCHEMATIC, iSBC 88/25

SHEET OF 10

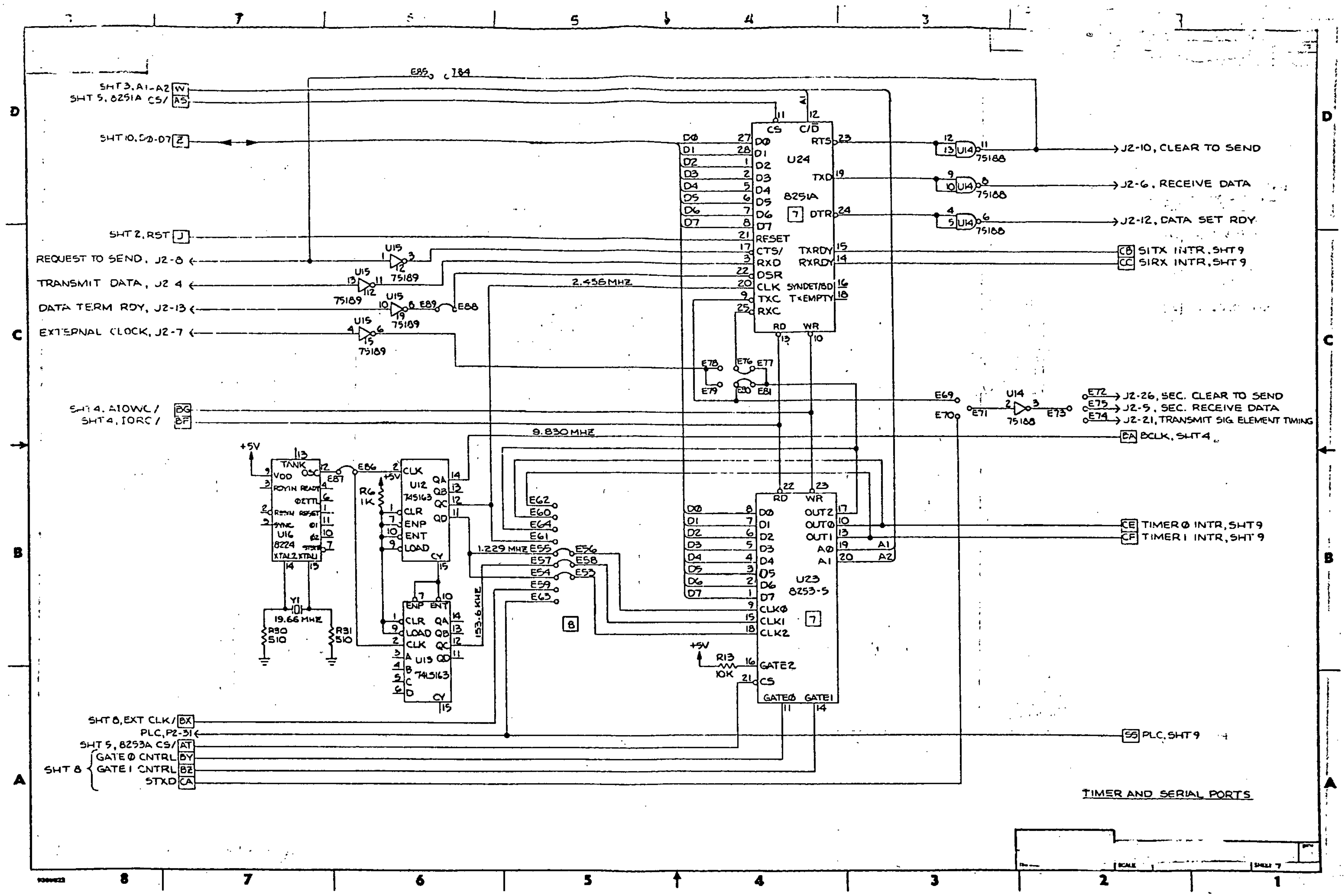
FIGURE H.1

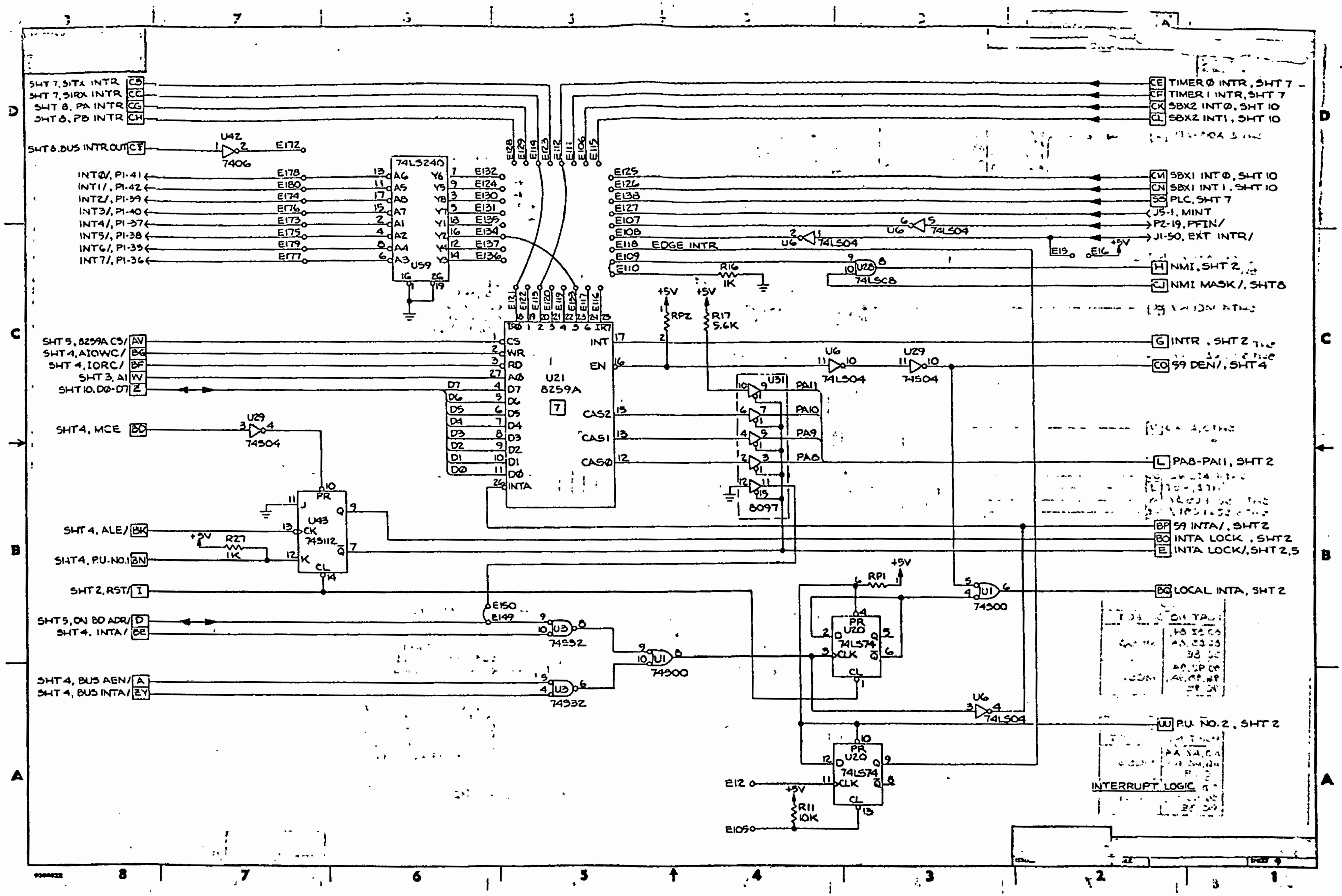


ADDRESS AND DATA BUFFERS



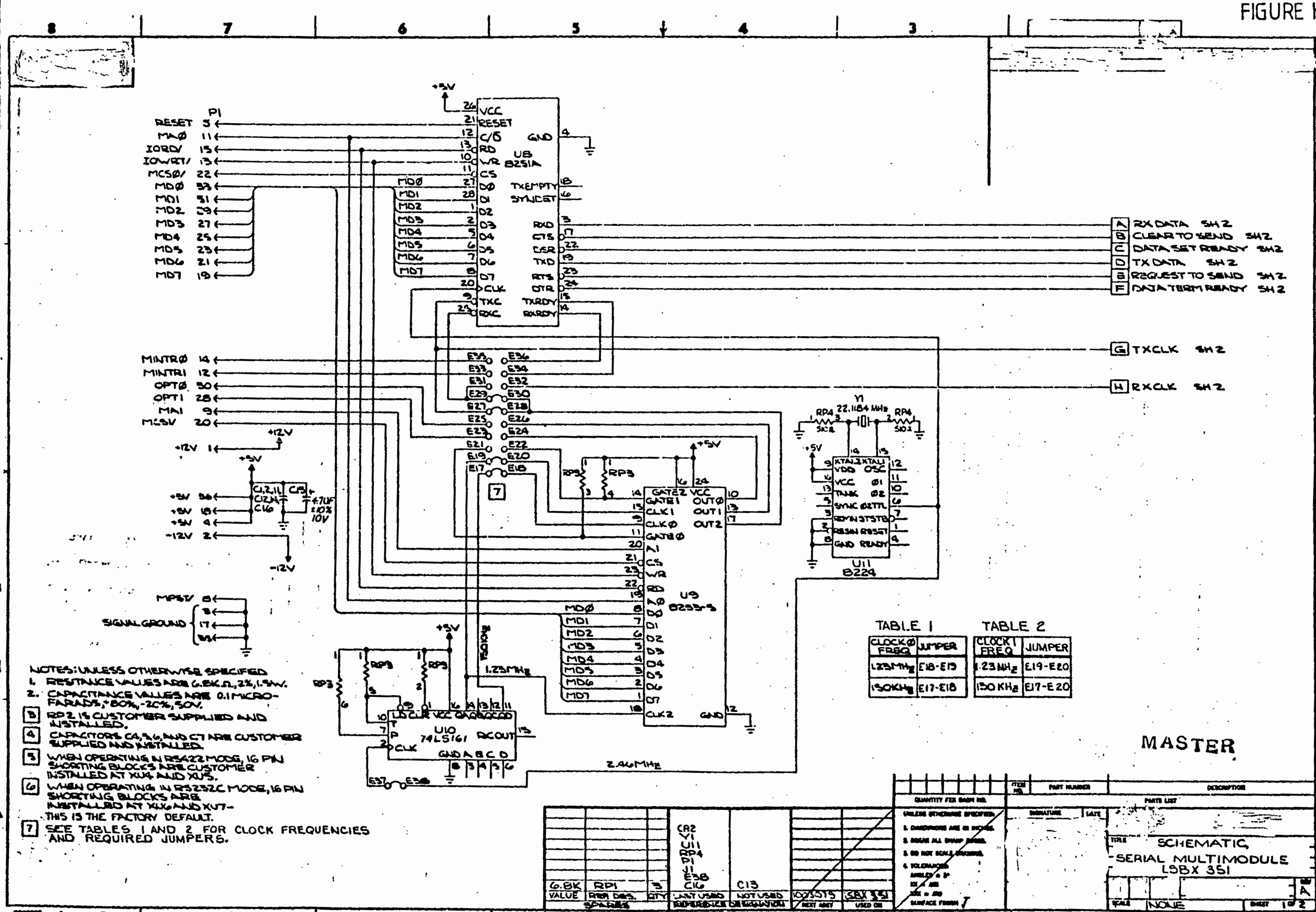
MEMORY AND I/O DECODE


SHT 3, A1-A2 W
SHT 5, 8251A CS/ AS
SHT 10, D0-D7 Z
SHT 2, RST J
REQUEST TO SEND, J2-8
TRANSMIT DATA, J2 4
DATA TERM RDY, J2-13
EXTERNAL CLOCK, J2-7
SHT 4, AIOWC/ BG
SHT 4, IORC/ BF
U24
8251A
CS C/D RTS TXD DTR RESET CTS/ RXD DSR CLK TXC RXC TXRDY RXRDY SYNDET/BD TxEMPTY RD WR
U15 75189
U14 75188
2.456 MHZ
9.830 MHZ
1.229 MHZ
153.6 KHZ
J2-10, CLEAR TO SEND
J2-6, RECEIVE DATA
J2-12, DATA SET RDY
CB SITX INTR, SHT 9
CC SIRX INTR, SHT 9
J2-26, SEC. CLEAR TO SEND
J2-5, SEC. RECEIVE DATA
J2-21, TRANSMIT SIG ELEMENT TIMING
CA BCLK, SHT 4
U16 8224
TANK OSC VDD RDYIN READY ØTTL RESIN RESET SYNC Ø1 Ø2 STSTB XTAL2 XTAL1
Y1 19.66 MHZ
R30 510
R31 510
R6 1K
U12 74S163
U13 74LS163
CLK CLR ENP ENT LOAD QA QB QC QD CY
U23 8253-5
CLK0 CLK1 CLK2 GATE2 CS GATE0 GATE1 OUT2 OUT0 OUT1 A0 A1
R13 10K
+5V
CE TIMER 0 INTR, SHT 9
CF TIMER 1 INTR, SHT 9
SHT 8, EXT CLK/ BX
PLC, P2-31
SHT 5, 8253A CS/ AT
SHT 8 GATE 0 CNTRL BY
GATE 1 CNTRL BZ
STXD CA
55 PLC, SHT 9
TIMER AND SERIAL PORTS



SHT 7, SITX INTR
SHT 7, SIRX INTR
SHT 8, PA INTR
SHT 8, PB INTR
SHT 8, BUS INTR OUT
U42
7406
E172
74LS240
U59
INT0/, P1-41
INT1/, P1-42
INT2/, P1-39
INT3/, P1-40
INT4/, P1-37
INT5/, P1-38
INT6/, P1-35
INT7/, P1-36
TIMER 0 INTR, SHT 7
TIMER 1 INTR, SHT 7
SBX2 INT0, SHT 10
SBX2 INT1, SHT 10
SBX1 INT0, SHT 10
SBX1 INT1, SHT 10
PLC, SHT 7
J5-1, MINT
P2-19, PFIN/
J1-50, EXT INTR/
EDGE INTR
74LS04
R16
1K
NMI, SHT 2
NMI MASK/, SHT 8
74LS08
+5V
RP2
R17
5.6K
U21
8259A
INT
EN
CAS2
CAS1
CAS0
INTA
INTR, SHT 2
59 DEN/, SHT 4
U31
8097
PA11
PA10
PA9
PA8
PA8-PA11, SHT 2
SHT 5, 8259A CS/
SHT 4, AIOWC/
SHT 4, IORC/
SHT 3, A1
SHT 10, D0-D7
SHT 4, MCE
U29
74S04
U43
74S112
SHT 4, ALE/
SHT 4, P.U. NO. 1
R27
1K
SHT 2, RST/
59 INTA/, SHT 2
INTA LOCK, SHT 2
INTA LOCK/, SHT 2,5
LOCAL INTA, SHT 2
SHT 5, ON BD ADR/
SHT 4, INTA/
SHT 4, BUS AEN/
SHT 4, BUS INTA/
U3
74S32
U1
74S00
RP1
U20
74LS74
U6
P.U. NO. 2, SHT 2
R11
10K
E12
E105
INTERRUPT LOGIC

FIGURE H.2


P1
RESET
MA0
IORD/
IOWRT/
MCS0/
MD0
MD1
MD2
MD3
MD4
MD5
MD6
MD7
+5V
VCC
RESET
C/D
RD
WR
CS
GND
U8
8251A
TXEMPTY
SYNDET
D0
D1
D2
D3
D4
D5
D6
D7
CLK
TXC
RXC
RXD
CTS
DSR
TXD
RTS
DTR
TXRDY
RXRDY
RX DATA SH 2
CLEAR TO SEND SH 2
DATA SET READY SH 2
TX DATA SH 2
REQUEST TO SEND SH 2
DATA TERM READY SH 2
TXCLK SH 2
RXCLK SH 2
MINTR0
MINTR1
OPT0
OPT1
MA1
MCS1/
+12V
-12V
SIGNAL GROUND
MPST/
Y1
22.1184 MHz
RP4
XTAL1 XTAL2
VDD OSC
VCC
TANK
SYNC
RESIN RESET
GND READY
U11
8224
RP3
GATE2 VCC
GATE1 OUT0
CLK1 OUT1
CLK0 OUT2
GATE0
U9
8253-5
CLK2
GND
U10
74LS161
RCOUT
1.23 MHz
2.46 MHz
E37
E38
TABLE 1
TABLE 2
CLOCK0 FREQ
JUMPER
1.23 MHz
E18-E19
150 KHz
E17-E18
CLOCK1 FREQ
JUMPER
1.23 MHz
E19-E20
150 KHz
E17-E20
NOTES: UNLESS OTHERWISE SPECIFIED
1. RESISTANCE VALUES ARE 6.8K Ω, 2%, 1/8W.
2. CAPACITANCE VALUES ARE 0.1 MICRO-FARADS, +80%, -20%, 50V.
3 RP2 IS CUSTOMER SUPPLIED AND INSTALLED.
4 CAPACITORS C4, 5, 6, AND C7 ARE CUSTOMER SUPPLIED AND INSTALLED.
5 WHEN OPERATING IN RS422 MODE, 16 PIN SHORTING BLOCKS ARE CUSTOMER INSTALLED AT XU4 AND XU5.
6 WHEN OPERATING IN RS232C MODE, 16 PIN SHORTING BLOCKS ARE INSTALLED AT XU6 AND XU7 - THIS IS THE FACTORY DEFAULT.
7 SEE TABLES 1 AND 2 FOR CLOCK FREQUENCIES AND REQUIRED JUMPERS.
MASTER
SCHEMATIC,
SERIAL MULTIMODULE
iSBX 351
NONE
A

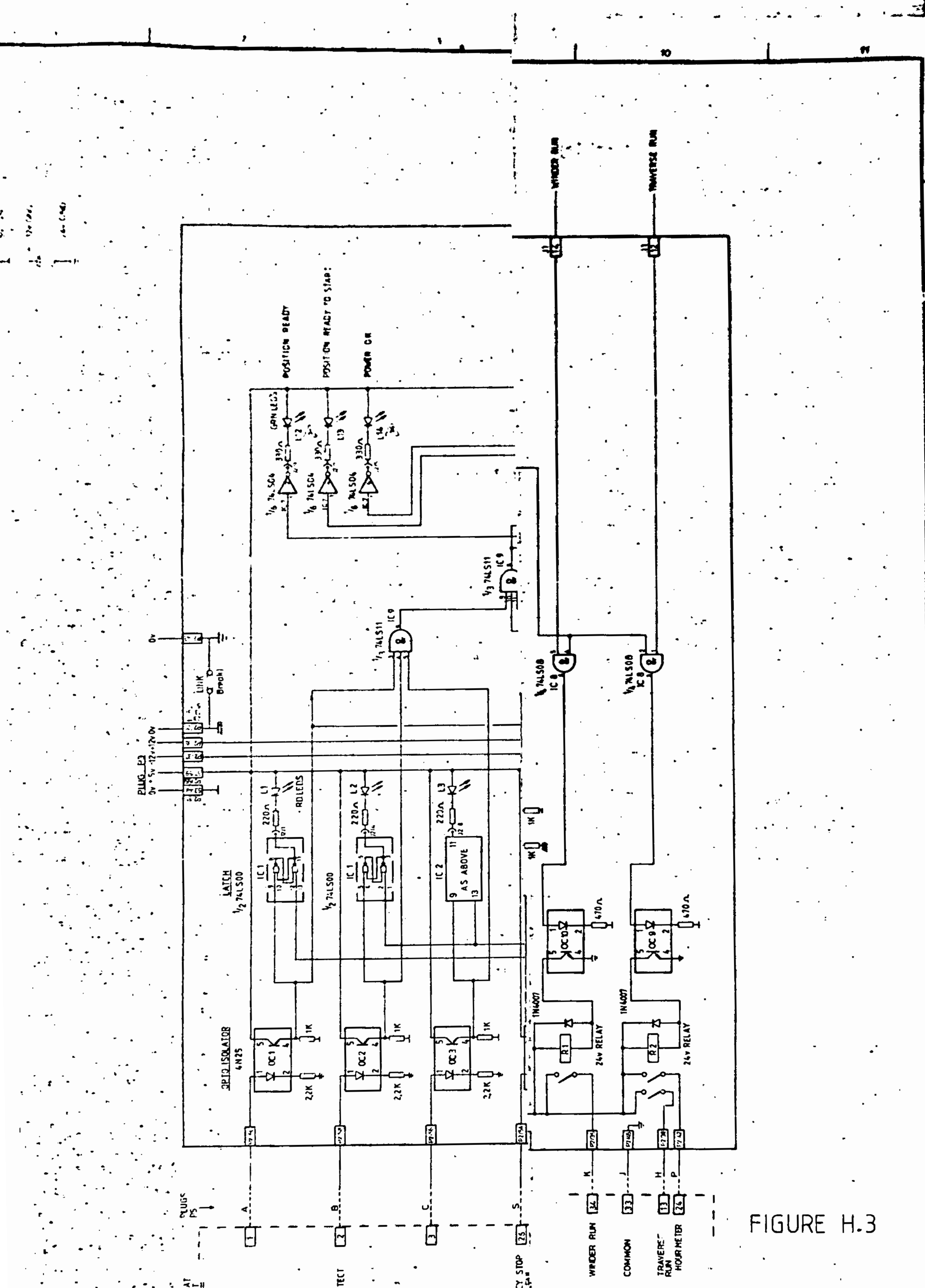


FIGURE H.3

SANS — SOUTH AFRICAN NYLON SPINNERS (PTY) LTD BELLVILLE

TITLE: ICU INVERTER CONTROL UNIT INTERFACE BOARD - CIRCUIT DIAGRAM MARK II

DWG Nº SA/B 1688

CODE Nº P_00_01_49D

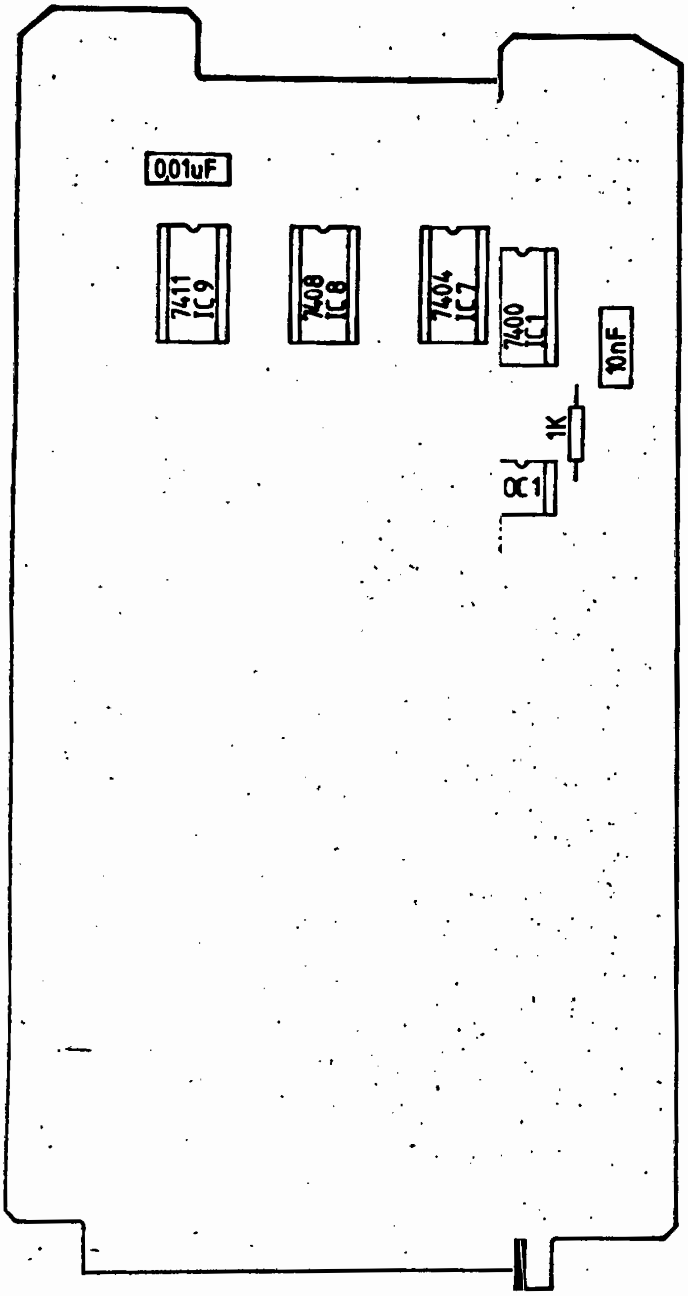


FIGURE H.4

NOTE: OC 1-13 ARE 4N33
OC 14-17 ARE 4N25

CAN NYLON SPINNERS
(PTY) LTD
BELLVILLE

T

DESCRIPTION	DRW'N	CHK'D	APP'D
	DATE	DATE	DATE

TIAL COMMUNICATION AND THE
IERS (PTY) LTD IT MUST NOT BE
CONSENT OF SOUTH AFRICAN
TURNED IMMEDIATELY ON THE

REV

METRIC REF SCALE 1 - 100 mm

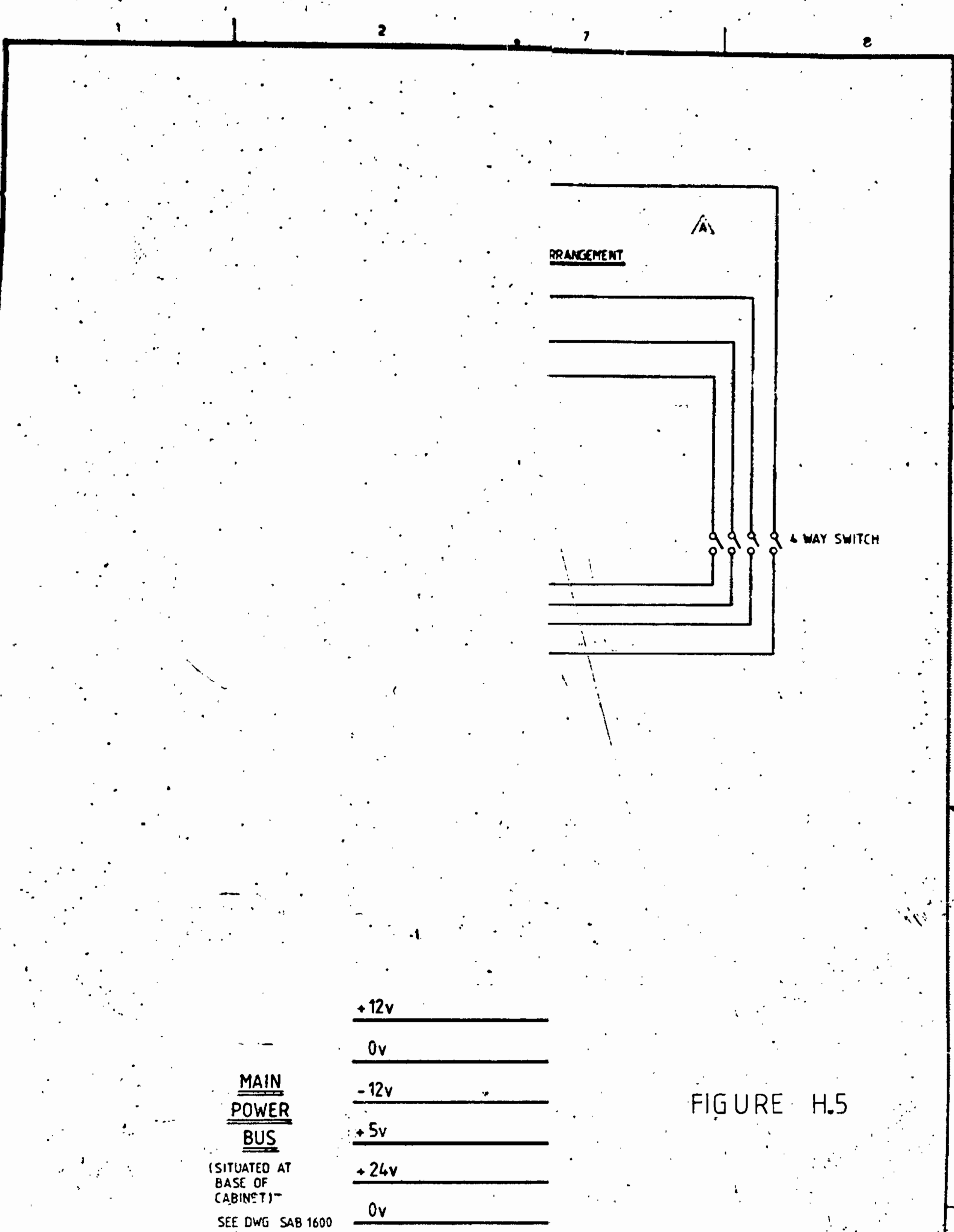


FIGURE H.5

SOUTH AFRICAN NYLON SPINNERS (PTY) LTD
BELLVILLE

B
TER CONTROL CABINET
POWER DIST.

1633

1-01-49D

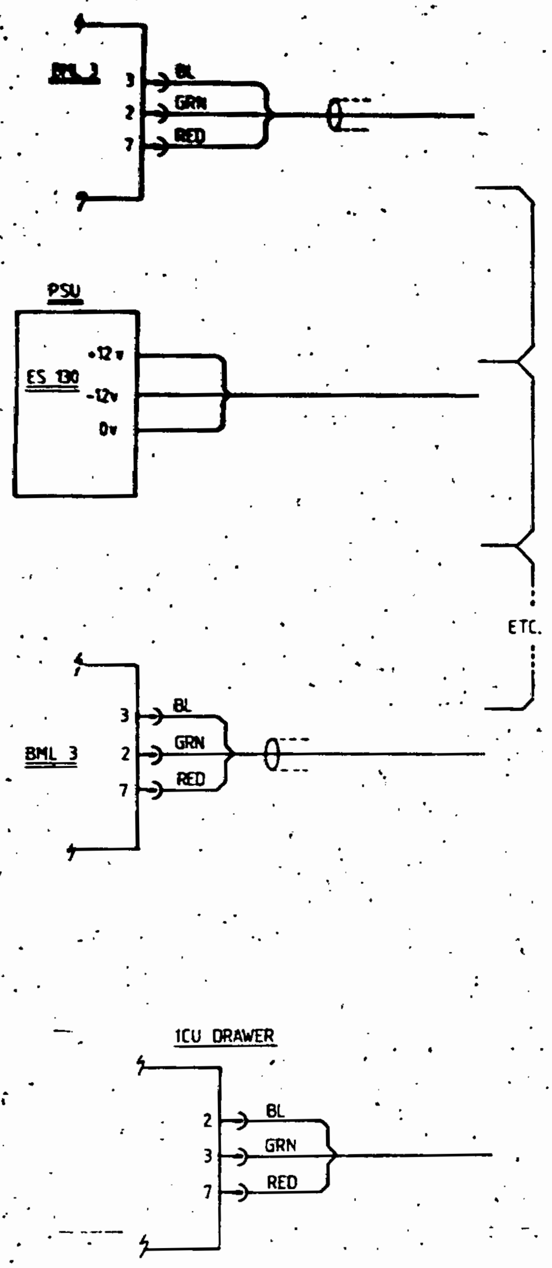


FIGURE H.6

SOUTH AFRICAN NYLON SPINNERS
(PTY) LTD
BELLVILLE

5B INVERTER CONTROL
1UNICATION & SIGNAL MONITORING
MATIC

B 1626

11-01-49D

REV	DESCRIPTION	DRWN DATE	CHK'D DATE	APP'D DATE	REV

2 2

SANS BELLVILLE
ENGINEERING DEPARTMENT
COMPUTER HARDWARE SECTION

ICUSCH.DOC
[300,20]

Issued
24th July, 1984

Revision 05 COMPUTER CONNECTIONS FOR ICU PROJECT

I N T E R C O N N E C T I O N S C H E D U L E S

F O R

I N V E R T E R C O N T R O L C O M P U T E R

Prepared by T.E. Kirk.

Rev 00 released 23/2/84
Rev 01 corrected and altered 12/3/84
Rev 02 serial link details altered 16/3/84
Rev 03 D9 wiring included. 20/3/84
Rev 04 serial link details altered.
Rev 05 power switch details included.

FIGURE H.7

C O N T E N T S.

A) Key to conventions used in this document.

In this document the following conventions will be used for naming sockets and plugs (See Fig 1 for an illustration of the physical appearance and location of the boards).

XXX-YYY-ZZZ

Where XXX is the unit where the socket or plug is located.

SBC = SBC 88/25 Single Board Computer.
SBX = SBX 351 "Piggy Back" serial Comms card.
I/O = I/O Board.
EXT = Back panel of computer drawer.
LED = Board carrying LED's mounted on front panel.
WX = Terminal strip at other end of I/O plug.

YYY is the particular plug or socket on a board or panel.

ZZZ is the pin number on the plug or socket.

e.g. SBC-P1-14 would be pin 14 on the P1 connector of the SBC 88/25 single board computer.

B.1) Computer Multibus connector.

Multibus Pin source.	Multibus Pin destination.	Connection Function.
SBC-P1-1,2,85,86	EXT-P3-6,7	GROUND for 5V and +/-12V.
SBC-P1-3,4,83,84	EXT-P3-1,2	5V Power.
SBC-P1-7,8	EXT-P3-3	+12V Power
SBC-P3-79,80	EXT-P3-4	-12V Power
SBC-P1-1 SBC-P1-14	EXT-RESET EXT-RESET	Software Reset.

B.2) I/O Board to External I/O connector.

I/O Plug Pin source.	I/O Board P2 destination.	Terminal Strip number.	Connection Function.
EXT-P5-A	I/O-P2-60	1	Stop Button.
EXT-P5-B	I/O-P2-58	2	Wrap Detect sense.
EXT-P5-C	I/O-P2-56	3	Oil Mist Fail.
EXT-P5-D	I/O-P2-44	4	Tailing Button.
EXT-P5-E	I/O-P2-46	5	Start Button.
EXT-P5-F	I/O-P2-32	6	Position Ready.
EXT-P5-G	I/O-P2-43	31	24V GND.
EXT-P5-H	I/O-P2-38	32	Traverse Run 24V.
EXT-P5-J	I/O-P2-45	33	Winder/Trav Common.
EXT-P5-K	I/O-P2-36	34	Winder Run 24V.
EXT-P5-L	I/O-P2-50	16	Traverse Inverter Ready
EXT-P5-M	I/O-P2-48	17	Winder Inverter Ready.
EXT-P5-N	I/O-P2-30	23	Head Lift Solenoid.
EXT-P5-P	I/O-P2-42	24	Hour Meter 24V.
EXT-P5-R	I/O-P2-52	25	Thermistor Trip.
EXT-P5-S	I/O-P2-54	26	Emergency Stop.
EXT-P5-T		27	Spare.
EXT-P5-U	EXT-P3-9 I/O-P2-40	11	+24V DC.
EXT-P5-V	I/O-P2-7	28	Chuck Speed TP.
EXT-P5-W	I/O-P2-1	29	Chuck/trav TP GND.
EXT-P5-X	I/O-P2-3	30	Traverse Speed TP.
EXT-P5-Y		N/C	Spare.
EXT-P5-Z		N/C	Spare.
EXT-P5-a	I/O-P2-6 I/O-P2-8	8 7	Chuck Tacho Screen. Chuck Tacho Core.

EXT-P5-b	I/O-P2-2	10	Trav Tacho Screen.
	I/O-P2-4	9	Trav Tacho Core.
EXT-P5-c	I/O-P2-14	19	Trav Freq out Screen.
	I/O-P2-16	18	Trav Freq out Core.
EXT-P5-d	I/O-P2-10	21	Wind Freq out Screen.
	I/O-P2-12	20	Wind Freq out Core.
EXT-P5-e	I/O-P2-41	22	Screens for Co-ax.

(The external I/O terminal pin numbers are included for ease of reference.)

These connections are made by links which are crimped into snap-in plug pins at the plug end, and soldered to connector lugs at the other end.

The MIL spec plug used for P5 has an unreliable record where the coax connectors are concerned, and so these connections were duplicated with twisted pair wires on P4, which is connected as follows:

I/O board source.	Back panel P4 destination.	connection function
I/O-P2-6	EXT-P4-1	Chuck tacho screen
I/O-P2-8	EXT-P4-6	core.
I/O-P2-2	EXT-P4-2	Trav tacho screen.
I/O-P2-4	EXT-P4-7	core.
I/O-P2-14	EXT-P4-3	Trav freq out screen.
I/O-P2-16	EXT-P4-8	core.
I/O-P2-10	EXT-P4-4	Winder freq out screen.
I/O-P2-12	EXT-P4-9	core.

B.3) I/O board to Computer Board.

SBC J1 connector	I/O J1 connector.	Connection Function
SBC-J1-4	I/O-J1-4	Host Communication Rx.
-6	-6	Host Communication Tx.
-8	-8	CPU Running.
-10	-10	Software Error.

-12	-12	Traverse Run.
-14	-14	Winder Run.
-16	-16	Position Ready.
-18	-18	Traverse Freq Out.
-24	-24	Winder Freq Out.
-26	-26	Winder Tacho Pulses In.
-28	-28	Traverse Tacho Pulses In.
-38	-38	Position Ready to start.
-40	-40	Tailing Button.
-42	-42	Address select Bit 0
-44	-44	Address select Bit 1
-46	-46	Address select Bit 2
-48	-48	Address select Bit 3

The I/O and Computer boards are linked together by a 40 way one-to-or ribbon cable terminated at both ends by 3M plugs. The cable is fitted betwee the SBC-J1 connector and the I/O-J1 connector.

8.4) Local VDU serial link.

Computer board link source.	Back panel P1 destination.	Connection function
SBC-J2-4	EXT-P1-2	Transmit Data. (Blue)
SBC-J2-6	EXT-P1-3	Receive Data. (Green)
SBC-J2-14	EXT-P1-7	Signal Ground. (Red)
SBC-J2-8	EXT-P1-8	RTS (Not used). (Yellow)

8.5) Host to ICU link.

Piggy back board source	Back panel P2 destination	Connection function.
SBX-J1-8	EXT-P2-3	Receive Data.(-ve)
SBX-J1-12	EXT-P2-1	Transmit Data.(-ve)
SBX-J1-9	EXT-P2-14	Transmit Data.(+ve)
SBX-J1-5	EXT-P2-16	Receive Data.(+ve)

The local VDU link is a standard 3-wire RS-232 Serial link, and the host link is a stndard RS-422 multidrop link. In both cases the connection is made using 4-core screened cable.

3.6) Power supplies.

Supply Source pin.	4PDT switch connection. incoming/outgoing	Supply destination.	Connector function.
EXT-P3-1,2	SW-2/3	SBC-P1-3,4,5,6, 81,82,83,84 I/O-P2-55,57,59	5V Power
EXT-P3-6,7	N.C.	SBC-P1-1,2,11, 12,75,76,85 86 I/O-P2-49,51,53	5V GND.
EXT-P3-3	SW-5/6	SBC-P1-7,8 I/O-P2-35	+12V Power
EXT-P3-4	SW-8/9	SBC-P1-79,80 I/O-P2-37	-12V Power
EXT-P3-8	N.C.	I/O-P2-1,5,9,13 17,23,27,10, 14,6,2	+/-12V GND.(Also used as screen GND.)
EXT-P3-9	SW-11/12	I/O-P2-40 EXT-P5-U	+24V Power
EXT-P3-5	N.C.	I/O-P2-43,45, 41,47	24V GND.

These connections are made by individually soldered leads.

C.1) I/O to terminal strip and interposing Relay.

Terminal Strip	I/O plug Pin.	Connection function.
1	A	Stop Button.
2	B	Wrap Detect.
3	C	Oil Mist Fail.
4	D	Tailing Button.
5	E	Start Button.
6	F	Position ready.
7	a	Winder Tacho core.
8	a	Winder Tacho Screen.
9	b	Trav Tacho core.
10	b	Trav Tacho Screen.
11	U	24V
12	REL-1	Interposing relay contacts.
13	REL-1	"
14	REL-2	"
15	REL-2	"
16	L	Traverse Inverter ready.
17	M	Winder Inverter ready.
18	c	Trav Freq out core.
19	c	Trav freq out screen.
20	d	Winder freq out core.
21	d	Winder freq out screen.
22	e	Co-axial cable screens.
23	N	Head Lift solenoid.
24	P	Hour meter 24V relay contact.
25	R	Thermistor Trip.
26	S	Emergency stop.
27	T	Spare.
28	V	Chuck Speed test point.
29	W	Test point Common.
30	X	Traverse speed test point.
31	G	24V GND.
32	H	Traverse run relay (24V to coil)
33	J	Trav/Wind run common.
34	K	Winder run relay (24V to coil)
N/C	Y	Spare.
N/C	Z	Spare.

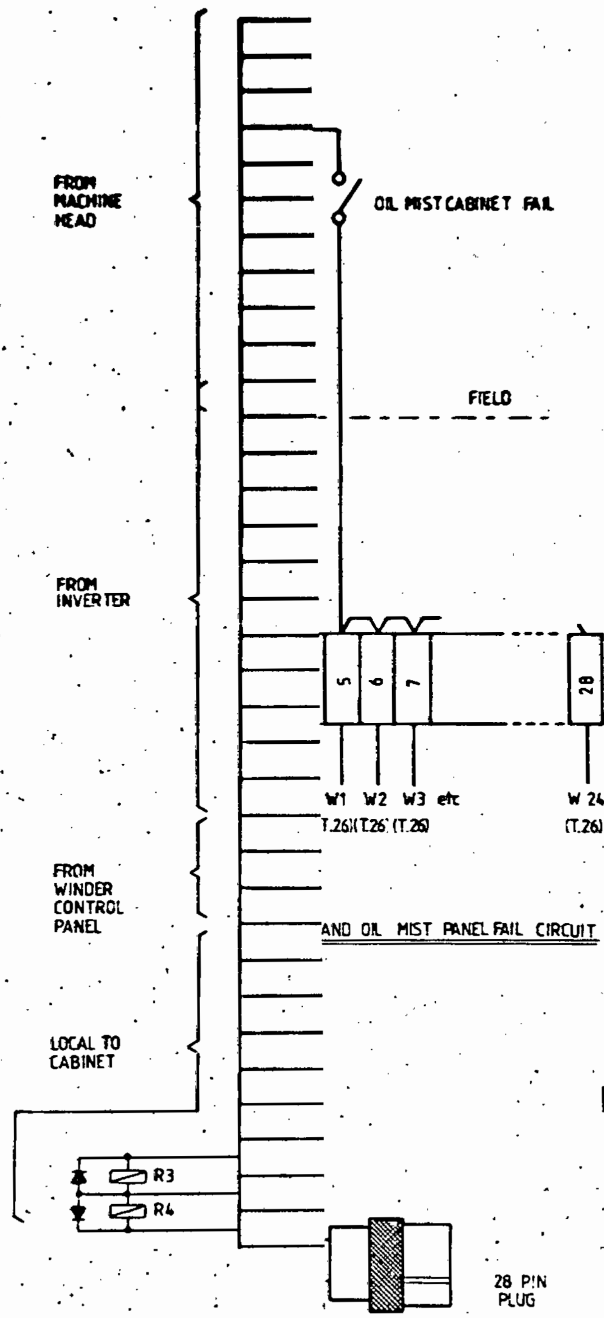


FIGURE H.8

SOUTH AFRICAN NYLON SPINNERS
(PTY) LTD
BELLVILLE

ACHINE 5B
IVERTER CONTROL FIELD WIRING

SAB 1422

P_01_01_49D

REV	DESCRIPTION	DRWN DATE	CHKD DATE	APPD DATE	REV

REV 2

LIST OF SIGNALS AND ASSOCIATED JUMPERS ON THE INVERTER CONTROL UNIT SBC 88/25 BOARD.

SIGNAL TYPE	SIGNAL ORIGIN/DESTINATION	JUMPER REMOVE/INSTALL	SBC PLUG & PIN NUMBERS	IC REF.
DIGITAL INPUTS	ADDRESS SELECTION - BIT 0	BUFFER U7 ON BOARD	J1-48	U7 & U22
(ON BOARD BUFFER)	(4-INPUTS) BIT 1	NO JUMPERS	J1-46	
PORT A	BIT 2		J1-44	
	BIT 3		J1-42	
	TAILING BUTTON - BIT 4		J1-40	
	BIT 5		J1-38	
DIGITAL OUTPUTS	POSITION READY - BIT 0	INSTALL BUFFERS	J1-16	XU-10 &
(INSTALL BUFFERS)	WINDER RUN - BIT 1	XU-10 (74LS00)	J1-14	XU-11 &
PORT B	TRAVERSE RUN - BIT 2	XU-11 (74LS08)	J1-12	U22
	WATCHDOG TIMER - BIT 3	NO JUMPERS	J1-10	
	CPU RUNNING - BIT 4	CUT TRACK IC22-22	J1-8	
		SOLDER IC48-11 TO XU11-1,2		
	COMMUNICATION-TX - BIT 5	CUT TRACK IC22-23	J1-6	
		SOLDER IC30-8 TO XU11-4,5		
		SOLDER IC21-23 TO IC30-9,10		
	COMMUNICATION-RX - BIT 6	CUT TRACK IC22-24	J1-4	
		SOLDER IC21-25 TO XU11-9,10		
DIGITAL OUTPUTS	OFF BOARD MEMORY INTERRUPT	REMOVE E50-E46 INSTALL E50-E105 (RESET LATCH)		
		INSTALL E12-E13-E14	10mS timeout	
		INSTALL E122-E118	IR1 interrupt	
PARALLEL I/O	JUMPER REMOVAL REQUIRED	REMOVE LINK E36-E40		
		REMOVE LINK E37-E41		
		REMOVE LINK E44-E48		
		REMOVE LINK E45-E49		
LOCAL VDU COMMS		INSTALL LINK E114-E117	SBC Rx int to IR6	
		INSTALL LINK E84-E85	CTS-RTS	
FREQUENCY OUTPUT		INSTALL BUFFER XU-9		XU-9 &
(INSTALL BUFFER)		(USE 74LS00 IC'S)		SBX-U9
FROM TIMER ON	WINDER FREQUENCY -SBX TIMER CLOCK-1	REMOVE LINK E47-E43	J1-24	
SBX BOARD	(OPTO ON SBC BOARD)	INSTALL LINK E43-E139		
		ON SBX-BOARD:		
		INSTALL LINK E26-E25		
		ON SBC-BOARD:		
	TRAVERSE FREQUENCY -SBX TIMER CLOCK-2	REMOVE LINK E46-E50	J1-18	
	(OPT1 ON SBC BOARD)	INSTALL LINK E46-E140		
		ON SBX-BOARD:		
		INSTALL LINK E28-E23		
		ON SBC-BOARD:		
FREQUENCY INPUT		INSTALL HEADER XU-8		XU-8 &
(INSTALL HEADER)		(LINK PINS 11&12,8&9)		SBX-U9 &
FROM TIMERS ON SBX	WINDER FREQUENCY -SBC TIMER CLOCK-1	REMOVE LINK E38-E42	J1-26	U-23
		INSTALL BUFFER	SHMITT BUFFER FOR	
		U-50 (74LS13)	CLEAN TACHO PULSES.	
		SOLDER U50-8 TO E58 &		
		U50-9,10,12,13 TO E38		
	TRAVERSE FREQUENCY -SBX TIMER CLOCK-0	REMOVE LINK E35-E39	J1-28	

FIGURE H.9

	(MINTR1 ON SBC BOARD) VIA J4-24 ON SBC BOARD	INSTALL LINK E120-E126 SOLDER U50-6 TO J4:24 & U50-1,2,4,5 TO E35 ON SBX-BOARD: REMOVE LINK E17-E18 REMOVE LINK E27-E28 REMOVE LINK E29-E30 INSTALL LINK E27-E29 INSTALL LINK E24-E33 INSTALL P1:24-E18	IR3 -TO GO TO SBC-UART CLOCK (SOLDER WIRE ONTO P1:24)	
UART CLOCKS	TO COMBINE CLOCK FOR SBC AND SBX UART'S -SBC TIMER CLOCK-2	ON SBC-BOARD: SOLDER J4:10 TO U24-9 ON SBX-BOARD: INSTALL P1:10-E31	(SOLDER WIRE ONTO J4:10 AND PIN 9 OF THE 8251) (SOLDER WIRE ONTO J1:10)	
8254 MODIFICATION	INCREASE CLOCK RATE TO 2.456 MHZ	ON SBX-BOARD: CUT TRACK TO U9-18 SOLDER U9-18 TO E20 REMOVE LINK E19-E20 INSTALL LINK E20-E38 (LEAVE E37-E38 INSTALLED)	 U-9=[PIT]=8254 (SOLDER WIRE ONTO PIN 18 OF THE 8254)	U-9
RAM MODIFICATION	TO GET SBC TO ACCEPT 2168 RAM CHIPS	ON SBC BOARD: CUT TRACK TO U40-14 SOLDER U40-13 TO U52-9	 (SOLDER WIRE ONTO PIN 13 OF U-40 & PIN 9 OF U-52)	U-40,U-52
ROM MODIFICATION	TO GET SBC TO ACCEPT 27128 ROM CHIPS	ON SBC BOARD SOCKET J6: INSTALL JUMPER PIN 1-14 INSTALL JUMPER PIN 6-9 INSTALL JUMPER PIN 7-8 REMOVE JUMPER PIN 2-13 REMOVE JUMPER PIN 3-12 REMOVE JUMPER PIN 4-11	INSTALL JUMPERS ON J6	J6

23 DEC 1985

SBC BOARD STANDARD JUMPER CONFIGURATION

JUMPER PAIR	IN/OUT	JUMPER PAIR	IN/OUT	JUMPER PAIR	IN/OUT	PINS NOT CONNECTED
1-2	IN	80-81	IN	156-157	IN	3,5,7,8,11,15,16,19
4-6	IN	82-83	OUT	158-159	IN	28,32,42,47,57
9-10	IN	84-85	IN	160-161	IN	69,118,123
12-13	IN					
13-14	IN	86-87	IN	163-164	IN	124,132,135,153
15-16	OUT	88-89	IN	165-166	IN	162,167,180
17-18	IN	90-91	IN	168-169	IN	
20-21	OUT	92-93	IN	170-171	IN	
22-24	OUT	94-187	IN	172-173	OUT	
23-25	IN	95-96	IN	174-175	OUT	
26-30	IN	97-98	OUT	176-177	OUT	
27-29	OUT	99-100	OUT	178-179	OUT	
30-31	IN	101-102	OUT	181-182	OUT	
33-34	OUT	103-104	IN	183-184	IN	
35-39	OUT					
36-40	OUT					
37-41	OUT	109-110	IN			
38-58	IN	111-119	IN			
43-139	IN	112-113	IN			
44-48	OUT	114-117	IN			
45-49	OUT	116-125	IN			
46-140	IN	120-126	IN			
51-52	OUT	122-118	IN			
53-54	IN					
55-56	IN					
59-60	OUT					
61-62	OUT					
63-64	OUT	133-J4:26	IN			
65-66	OUT	141-142	OUT			
67-68	OUT	143-145	IN			
70-71	OUT	144-146	OUT			
72-73	OUT	147-148	OUT			
74-75	OUT	149-150	IN			
76-77	IN	151-152	IN			
78-79	OUT	154-155	IN			