COMPUTER NUMERICAL CONTROLLED DRILLING MACHINE INTERFACED TO A COMPUTER AIDED DESIGN PACKAGE

A dissertation presented to the Department of Electrical Engineering Peninsula Technikon in partial fulfilment of the requirements for Master's Diploma in Technology: Electrical

by

PIERRE FRANS MOSTERT

FEBRUARY 1996

ABSTRACT

In answer to the needs of the growing number of smaller manufacturers of Printed Circuit Boards, the "Computer Numerical Controlled Drill interfaced to a Computer Aided Design software package" was conceptualised. This development would considerably improve both productivity and efficiency and consequently would improve cost effectiveness.

The smaller manufacturer is unable to afford the more sophisticated Computer Numerical Controlled Drill equipment and the software packages which drive them. It was necessary for the Computer Numerical Controlled machine to possess all the capabilities and versatility of the more sophisticated and expensive equipment, and yet still be affordable to the smaller manufacturer. Furthermore, since such equipment would need to be operated in fairly hostile environments, it would need to be robust and reliable.

This interdisciplinary development has required the knowledge and skill from many different engineering fields, and made them readily available to industry which otherwise would not have had access to them.

The South African economy at present , has a growing need for smaller and medium business enterprises, and the needs of these must be appropriately served. It is envisaged that this development will make a significant contribution to greater economic activity.

ACKNOWLEDGMENTS

I would like to thank the Director and staff of the Electrical Engineering Department for affording me the opportunity to develop the Computer Numerical Controlled Drilling Machine, and for their support during this development period.

I would also like to thank my supervisor Nooridien Dhansay for his patience and guidance especially during the documentation phase of the development, and to my wife Janet, for helping me with the proof reading.

A special thanks to Ian Dorrat and Jan Ehlers for their enthusiasm and for encouraging me to complete the development on time.

A final belated thanks to the late Dennis Oliver for his assistance in the mechanical development of the Computer Numerical Controlled Drilling Machine.

I hereby state that the contents of the dissertation "Computer Numerical Controlled Drilling Machine Interfaced To A Computer Aided Design Package" is my own work and that the opinions contained herein are my own and not necessarily those of Peninsula Technikon.

TABLE OF CONTENTS

TABLE OF ILLUSTRATIONS

CHAPTER 1

INTRODUCTION

The smaller Printed Circuit Board (PCB) manufacturers need to increase their productivity and to operate more effectively.(The Cad/Cam Handbook,1980:61) A common problem experienced by these smaller manufacturers is the turnaround time which is impeded by the time spent drilling the Printed Circuit Boards. These small industries cannot afford the expensive and sophisticated Computer Numerical Controlled (CNC) Drilling machines that the PCB market has to offer.

The idea was conceptualised to develop a low cost reliable and robust Computer Numerical Controlled Drilling machine that possessed the capabilities and versatility of the larger, more expensive CNC Drilling machines.(CULLEY, 1985:75) This CNC Drill had to operate with existing Computer Aided Design (CAD) software packages, as the smaller industries cannot afford to purchase the more advanced CAD/CAM software systems which are normally used with CNC machines. (TEICHOLZ,1985:16.22)

The design of this CNC Drill required a knowledge which united many engineering disciplines from C and Assembler

programming to Control and Interfacing techniques. Readily available hardware components had to be sourced from various suppliers to construct the CNC Drill. To accomplish the control of the CNC Drill translation, conversion and control software programmes were developed.

The topic discussed in the various chapters are as follows:

Chapter 1 is an "Introduction ", dealing with the motivation for and the significance of developing the Computer Numerical Controlled Drill. It also deals with the historical developments of the computer numerical systems through the years.

Chapter 2 describes the topology of the system dealing with both the hardware and software aspects of the development.

Chapter 3 dissects the development into its various hardware components, with discussions on each peripheral.

In Chapter 4, the important aspects of "Serial Communication Interfacing" and its configuration are dealt with.

Chapter 5 covers the "Initialisation Procedures" which initialise the Motor Control Unit and also the all important application of the interrupt service routines.

The topic in Chapter 6 is "Computer Aided Design File Translation", and the ability of the software to extract and translate the data into coordinates for the Motor Control Unit.

The "Display Graphics" are handled in Chapter 7. The design considerations and techniques employed to develop the graphical user interface are discussed.

In any control system, an important aspect is the capability of the system to handle the error condition. This topic is reviewed in Chapter 8.

Following the development of the Computer Numerical Drill, further in depth considerations were made which would enhance the system performance. This resulted in recommendations which are described in Chapter 9.

Features incorporated in the design of the CNC Drilling machine were the following:

(1) Can be interfaced to any CAD system which supports Houston instruments DMP 51/52 plotters (DM/PL III)
(2) Machine has the capability of being controlled manually.
(3) Automatic homing facilities. (Datum)
(4) Routing facilities.

(5) Graphical representation while drilling.

(6) On-line error detection and diagnostics.

(7) Speed variation and interpolation.

This CNC Drilling machine will be beneficial to smaller PCB manufacturers and will improve productivity by increasing the speed of production of printed circuit boards.

1.1 SIGNIFICANCE.

In recent years, CAD systems have been developed by various Software Houses to improve productivity and lower the costs of manufacturing Printed Circuit Boards. Most of the large manufacturers are able to afford the enormous cost of installing a Cad/Cam system, but the smaller industries cannot afford this capital outlay. These small industries have to rely on manual labour to drill the P.C. Boards.

Developing a low cost CNC drilling machine with various options, will allow the smaller industries to operate more cost effectively. With the implementation of such a system, the turnaround time will improve by between 30 and 50 percent. A reduction of direct labour costs of 70 percent and a reduction of work costs of up to 50 percent have occurred, although labour costs have increased substantially since CAD/CAM systems were released in 1977.(BESANT,1986:380)

By using the above described interactive Numerical Control (NC) programming as proposed over the old method of entering coordinates via a paper tape or computer keyboard, a significant productivity improvement can be ensured. Research has shown that a productivity gain of 4:1 or better are not uncommon. (BESANT,1986:380)

1.2 BACKGROUND:

The early ancestor of the NC (NUMERICALLY CONTROLLED) machine was developed around the turn of the century by Joseph Jacquard. Jacquard developed a loom that could be programmed to produce the exact pattern repeatedly. This programming was accomplished by punching holes into a wide paper tape . The holes represented instructions that defined the weaving pattern. (TEICHOLZ,1985:16.5)

In the late 1940's, John T. Parsons recommended a method of automatic machine control that would guide a milling cutter for the generation of smooth profiles. The U.S. Air Force then used the Parsons concept and commissioned the Massachusetts Institute of Technology (MIT) to develop a practical implementation of the concept. MIT built a control system for a two-axis drill press using a perforated paper tape. The prototype was refined and used throughout the United States. (TEICHOLZ,1985:16.5)

In the late 1950's and 60's, programming languages such as APT were developed to assist engineers to define the proper instructions for NC machines. (CHORAFAS,1987:9)

Through the 1960's, NC manufacturers made use of hard-wired logic to accomplish control functions, due to the fact that low cost high speed computing facilities were unavailable.(CHORAS,1987:6)

During the mid-1970's computer based NC machines became available. NC machines could now be programmed via a keyboard and an intelligent terminal.(TEICHOLZ,1985:16.5)

The first CAD systems were released in the period from 1977 to 1979 and enabled the designer to develop Printed Circuit Boards on an interactive Video Display Unit. (VDU).

The early 1980's saw new concepts develop such as Computer Numerical control (CNC) and Direct Numerical Control (DNC) which were controlled via mini mainframe computers like the VAX 11/780, instead of via paper tape like the older NC machines. Other CNC machines had dedicated microprocessors with memory installed and coordinates were entered via a keypad. (CHORAFAS,1987:7).

From 1984 onward, the personal computer CAD software was developed for the 8 bit, and then later for the 16 bit,

microprocessors. The Cad systems developed incorporated high resolution colour displays, design rule checking and pattern generating. There was also a growing tendency towards the use of voice, sight and gesturing devices for operator interface.

At the end of the 1980's the CNC machines were still being controlled via CAD/CAM (COMPUTER AIDED DESIGN / COMPUTER AIDED MANUFACTURE) systems which was controlled by mini and mainframe computer or dedicated microprocessor. (CHORAFAS,1987:7). This type of system, while very efficient, is also costly. Thus in 1990 development had begun on a PC based Cad/Cam system. Efforts to integrate NC machines into computer networks like LAN (local area network) and WAN (wide area network) had already commenced.(TEICHOLZ,1985:16.20)

As certain manufacturers made the Drill Tapes and hardware components available, a true dedicated CNC Drilling machine based on a personal computer (PC) could be developed. These CNC machines could now also be interfaced to a CAD system and include various options which were previously not available.

An enquiry was made through the CSIR information services to ascertain if a CNC Drilling machine had been previously developed in South Africa. According to their databases and the National Project Register, no such machine has been developed. Refer to Appendix A.

CHAPTER 2

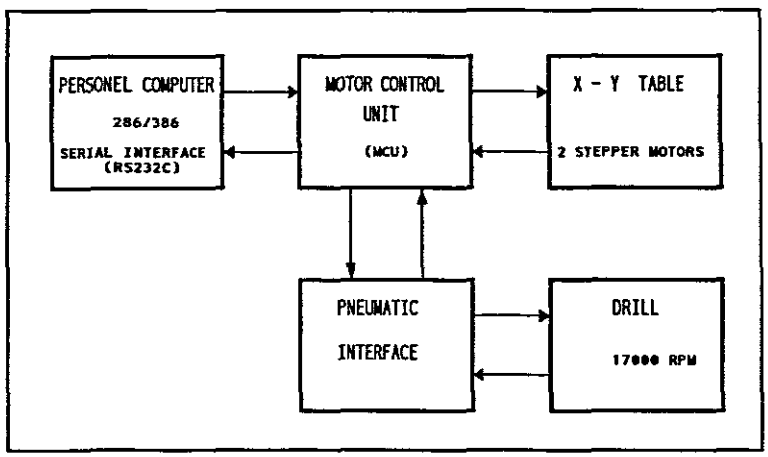
THE SYSTEMS TOPOLOGY

The system topology of the Computer Numerical Controlled (CNC) Drill embodies both hardware and software components. These hardware components need to be interfaced together to form the CNC Drill architecture. The functions of data manipulation and the controlling of the CNC drill operation, are performed by various software modules which have been developed.

2.1 HARDWARE TOPOLOGY

The CNC Drill, when regarded as a control system, consists of various elements which, when combined, form an effective automated drilling machine. The Personal Computer (PC) which requires a standard Intel 286 or 386 processor, performs the function of a Process Controller. The PC with its serial input/output (I/O) card is used to pre-process and translate data which will later be used to control the CNC Drill. Another function of the PC is to control all the CNC Drill

functions and act as a user interface between the operator and the CNC Drill. All communication between the PC and CNC Drill are achieved via a RS232c asynchronous 5 wire serial link to the Motor Control Unit(MCU).



SYSTEM TOPOLOGY

The instructions which are received from the PC are executed by the MCU 8085 based microprocessor. The MCU processes all the control signals to and from the X-Y table and Drill. The feedback signals from the Drill and X-Y table are translated and sent back to the PC for interpretation. The MCU functions are that of overall systems control and arithmetic computation of data received from the process controller. Another function which the MCU performs is the manual control function, which allows the operator to position the X-Y table manually by means of a joystick. The Motor Control Unit

controls two main components; these being the X-Y table and the Drill. The X-Y table is driven by two four phase stepper motors. The stepper motors each drive a single axis, namely the X or Y axis. Each axis is in turn driven by a lead screw.

The X-Y Table consists of a X and Y axis which are two moving beds that run on stainless steel rails. These rails are mounted to a solid base. Each bed is driven by a lead screw linear system. These lead screws are propelled by two stepper motors. Limit switches are fitted to each axis to prevent the X-Y Table from exceeding its maximum travel. An optical sensor is also mounted on each stepper motor shaft to assist the MCU in determining the position of the X-Y Table in relation to its home position.

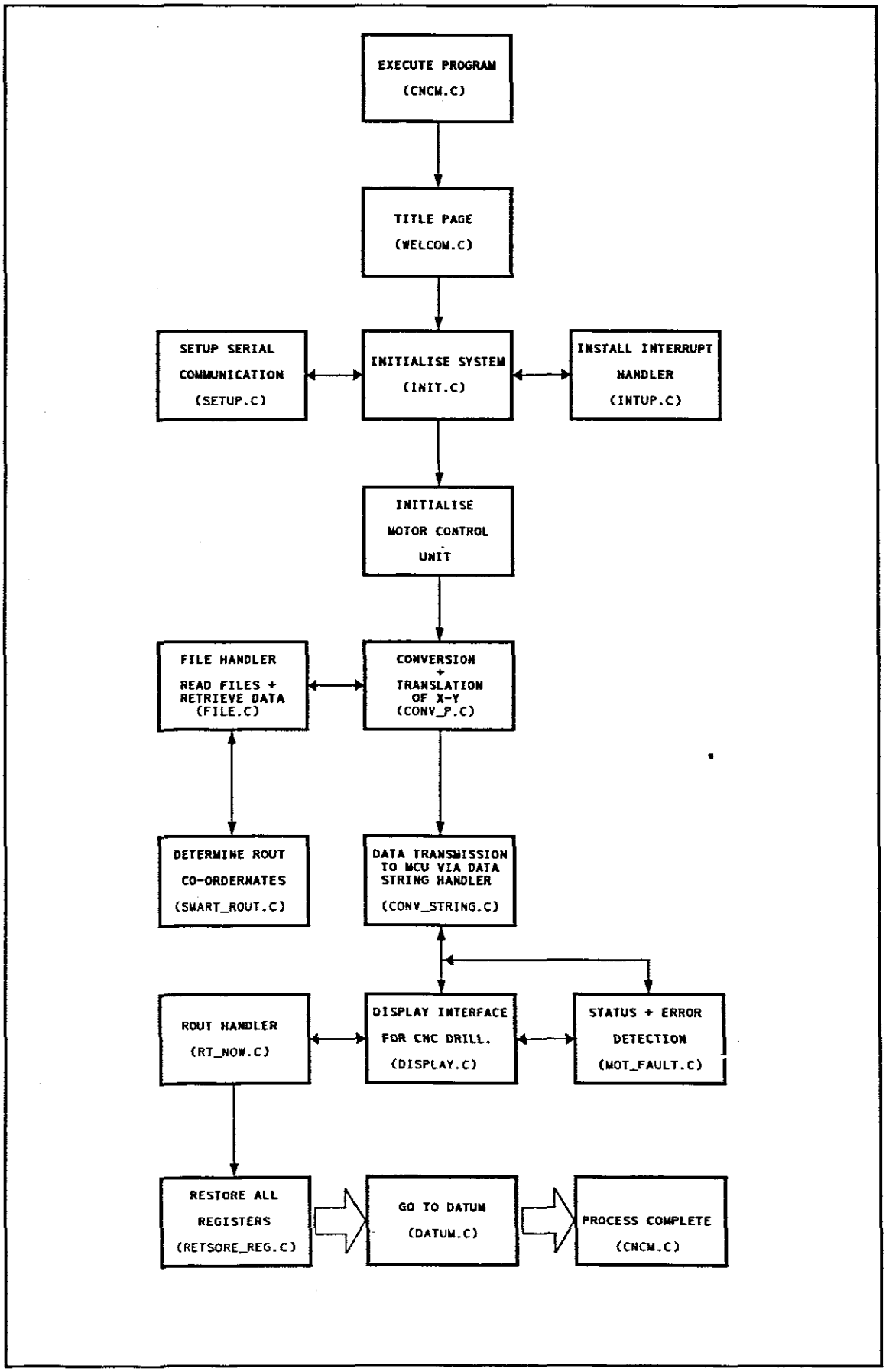
The Drill is powered by a 3 phase motor which rotates at 17000 revolutions per minute. The control of the drill is accomplished by a Pneumatic Controller which converts the electrical signals from the MCU into pneumatic signals for the drill.

2.2 SOFTWARE TOPOLOGY

The software routines and functions for translating, communicating and controlling the CNC Drill were developed in the C language. The C source code was compiled under Turbo C environment creating separate object files and these files were linked to create a single executable file called "cnc.exe". When the software is initialised, the main routine is called, being "cncm.c" and because the C language is a structured language, the main routine contains all the calls to the other routines and therefore is the controlling routine.

The main routine then calls "welcom.c" which is the title page and introduces the programme. The next routine is "init.c" which initialises the system. This routine then accesses another function "setup.c " which first enquires from the operator if any serial communication parameters need to be changed. The changing of any of the serial port parameters is done via a menu.

Once the operator has selected the correct parameters, the function then configures the Universal Asynchronous Receiver and Transmitter (UART) of the Personal Computer's serial I/O card and then returns to the initialisation routine.



SOFTWARE TOPOLOGY

All the software that was written for the CNC Drill is interrupt driven . This option was chosen for effective serial data transfer without wasting the Central Processing Unit's (CPU) machine cycles by polling for data, which reduces the processing time of the PC.

With interrupt driven software the UART is programmed to generate interrupts if data needs to be transmitted or received. The UART then requests the CPU to service the interrupt. This routine is known as the Interrupt Handler which services the interrupts and is the most efficient method of dealing with interrupts. The initialisation routine then installs the interrupt handler "intup.c". Both the UART and Programmable Interrupt Controller (PIC) are programmed at systems level. Once the routine is installed the Motor Control Unit is initialised. The interrupt service routine, PIC configuration and initialisation routine are discussed in more detail in Chapter 5.

After the communication has been established with the MCU, the main programme calls conversion and translation routines, called "conv_p.c" This routine in turn calls the file handler "file.c" which reads the data files and retrieves the data. The data is then converted and translated into coordinates, both for drilling and routing procedures of the Printed

Circuit Board. A more exact description of the translation process is dealt with in Chapter 6.

The coordinates are stored in arrays and then sent via a string handler "conv_string.c", to both the MCU and a routine which displays a graphical representation of the Printed Circuit Board (PCB) that is being drilled. The development of the graphical representation routine is discussed in Chapter 7.

While the CNC Drill is drilling the PC board, the operator can see the progress of the drill procedure displayed on a monitor. Another routine observes the status of the MCU and detects any errors that may occur. Any critical errors that do occur are reported to the operator and in certain cases the process will be terminated.

On completion of the drilling procedure, the operator has the option of routing the PC board. This process is controlled by the Rout handler "rt_now.c" which routs the edges of the PC board. When this procedure is completed, all the register values are restored to their original value and the Datum routine is called to return the X-Y table to the home position. The process is now completed and the programme is terminated.

CHAPTER 3

CNC HARDWARE AND PERIPHERALS

The CNC Drilling machine consists of a Motor Control Unit (MCU) which consists of all the electronic control modules and numerous other hardware peripherals like the Autofeed Drill, Pneumatic Controller, Hydraulic Control Unit and X-Y Table.

3.1 MOTOR CONTROL UNIT

The Motor Control Unit (MCU) consists of a number of electronic modules which, when combined, form a very effective closed loop control system. These modules are the following:

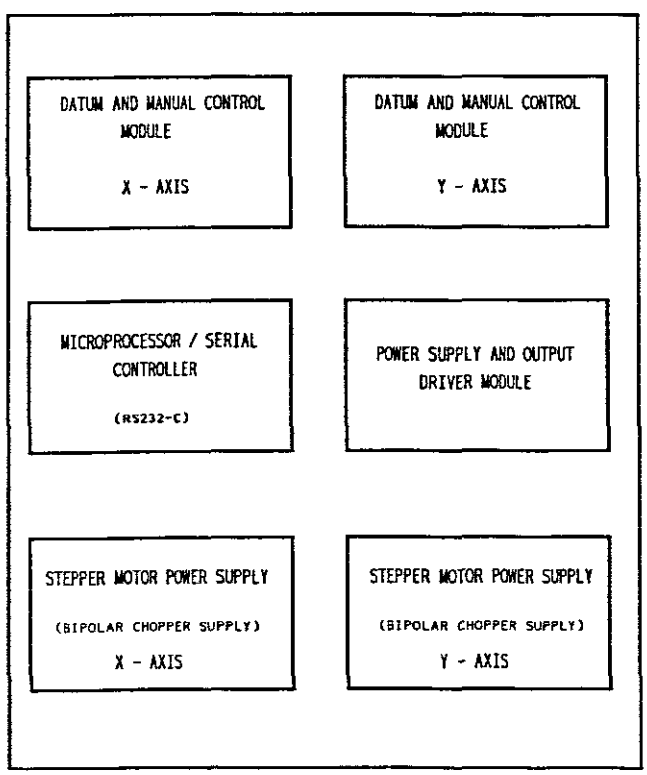
(1) Two Datum and Manual control modules

(2) Two Bipolar Drives modules

(3) Serial Microprocessor Index Controller (RS232c)

(4) Power Supply and Output Driver module

The MCU communicates with the Personal Computer (PC) via a RS232C link which employs five wire communication in an asynchronous mode.The serial link utilises the five wire communication to facilitate handshaking between the MCU and the PC. The speed of the serial link ranges from between 300 to 4800 baud. The Motor Control Units processing is accomplished by a 8085 based microprocessor which needs to compute start/stop speeds, acceleration, backlash correction and linear interpolation. It also controls all the signals both to and from the x-y table and drill.



MOTOR CONTROL UNIT HARDWARE

Its main function can be summarized as follows:

(1) Overall Systems Control. Coordination of all data transmission and timing for the CNC.

(2) Input - Output Control. Processing of all input instructions to the CNC and output signals to the microprocessor.

(3) Servo Control. Processing of CNC instructions that accomplish axis control and positioning.

(4) Arithmetic Operations. Computation of certain mathematical functions.(eg acceleration / deceleration calculations.)

3.1.1 BIPOLAR DRIVES:

The stepper motors are powered by Bipolar Drives. These two Bipolar Drives are constant current drives which maintain relatively constant currents to the motor at all speeds, and therefore, offer good stepping performance at speeds up to approximately 10 000 steps per second.

The Bipolar drives are designed to cater for a wide range of motors and deliver nominal currents of between 2 Amps and 3 Amps for a voltage range of between 24 Volts to 36 Volts which are programmed by switch settings or by means of external resistors. These Bipolar drives which employ chopper regulation achieve high overall efficiency, eliminating the need of ballast resistors and minimising power consumption.

The Drives have built in protection against short circuits across or between motor windings. The two drives are powered by an 18-0-18 volt unregulated AC supply from a mains transformer.

Each Bipolar drive supplies one axis of the x-y table and these drives can be configured in a master or slave configuration so that the drive can be synchronised. By synchronising the drives the beat frequencies are eliminated.

The Bipolar drive modules are designed to be fitted in a 19" 3u rack which plugs into a motherboard or backplane enabling all external connections to be made via screw terminals.

These chopper drives have the facilities to operate in full step mode and half step mode which allows them to step at 200 steps per revolution or 400 steps per revolution respectively. This means that during one step in full step mode the stepper motor will advance 1.8 degrees and in the half step mode 0.9 degrees.

The control signals from the microprocessor/serial controller are fed via the motherboard to the Bipolar drives. These signals are Direction and clock pulse. The Zero Phase and Motor Fault signals are sent to the microprocessor/serial controller via the backplane.

The advantage of using Bipolar chopper drives over other drive methods is that they allow features such as closed loop control, different step modes, current boost and stabilisation. In addition, improved motor performance, especially at high speed where motor torque is required.

3.1.2 SERIAL MICROPROCESSOR INDEX CONTROLLER.

The serial microprocessor index controller which is commonly known as the Indexer generates step and direction signals and allows control of up to three axes of stepper motor drives via a RS232C serial link. The indexer card can communicate via its serial port with any process controller or user-supplied machine. In the CNC Drill's specific application the process controller used is a Personal Computer. Any of the three axes which are designated X,Y and Z, may be run individually or the same move can be performed on two or three axes simultaneously.

One of the more unique capabilities of the indexer, is linear interpolation of two axes. This feature allows simultaneous moves to be programmed on X and Y axes with different distances. As a result, the operator or programmer can programme a vector at any angle on the X-Y system.

This 8085 microprocessor based indexer accepts ASCII characters via the RS232c serial interface and subsequently controls direction, distance, acceleration/deceleration and velocity for full, half and micro stepper motor drives.
The Indexer is also capable of monitoring interface status, drive status, axis status and requesting position of each drive.The indexer has its own power supply module on which the input and output transistors are also mounted. Both the Serial Microprocessor indexer card and power supply I/O board plug into a motherboard which is fitted in the rack.

In the sequence mode a preset sequence of up to 63 moves can be loaded into the indexer. The indexer can also be programmed to wait for specific input conditions, at points in the sequence allowing an external operation to be performed before the sequence continues. The asynchronous control of the indexer will be discussed in a later chapter, Chapter 5.

3.1.3 CLOCK / DATUM MODULES

The MC2 Clock /Datum card which for convenience purpose will be referred to as the Datum card is plugged into the Datum motherboard which is fitted into the rack. This motherboard accepts a maximum of two Datum cards for applications of both single and dual axis systems.

These Datum cards permit manual positioning of the X-Y table independently of the controller, as well as the incorporation of limit switches and enables a datum routine to be performed with no additional connections to the controller. Interconnections via two 8 way cables are made between the motherboards of the Datum, Serial Microprocessor Indexer and Bipolar drives forming a control bus. This control bus transfers the individual axis control signals to both the Indexer and Bipolar drive modules.

A number of manual control functions are provided for on the Datum cards. These functions allow both for X and Y motors of the X-Y table to be manually controlled in both the positive and negative direction. The functions are the following: Fast +, Fast -, Slow/Jog +, and Slow/ Jog -.

The Slow/ Jog function allows both single step or multiple slow step of the X-Y table. The speed ranges of slow and fast functions both have a upper and lower speed limit. The slow function has a range of between 50 and 1500 steps per second (or Hertz) with no acceleration / deceleration option , while the fast range is between 1 and 20 Kilohertz which also incorporates a ramped acceleration/deceleration function.

For good control of the X-Y table these functions were connected to a joystick and toggle switch. The joystick determines the direction, while the toggle switch can be switched between the slow and fast ranges and therefore determines the speed at which the X-Y table travels. Both the Fast, Slow and the Acceleration time can be preset by adjustment of the clocks on the oscillator section of the Datum cards. One factor which should be remembered is that if the acceleration is too rapid or the fast speed is set above the maximum input frequency specification of the stepper motor, the motor will stall.

Other input functions which are provided by the Datum cards are the Limit inputs and Emergency Stop. The positive and negative limit switches which are mounted on the X-Y table

are set to prevent the X-Y table from exceeding its maximum travel. When these limit switches are activated the Datum card removes the step input to the motor but still allows the Bipolar drives to apply a holding torque to the stepper motor. The Emergency Stop input permits the operator to halt the operation of the Motor control Unit in the case of emergency. When the Emergency Stop button is activated the Bipolar Drives still apply a holding torque to the stepper motor drives to prevent the X-Y tables position from being changed manually.

Another function that the Datum card possesses is the Auto Datum function which is a complex process which involves returning the X-Y table to a fixed mechanical reference point. The "Home" position is referred to as the Datum Point.

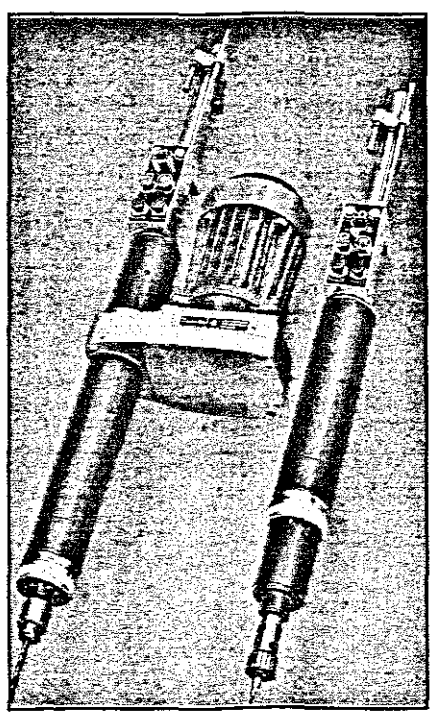
The Datum function has two modes of operation which are the Bidirectional and Unidirectional approach to the datum point. Due to the design of the X-Y table the Unidirectional mode was selected. In the Unidirectional mode, the X-Y table will always approach the datum point from the same side when the datum routine is requested, either manually or via the software. Due to the fact that the X-Y table always

approaches datum from the same side, backlash from the system is eliminated. The auto datum function will be discussed more comprehensively in the section on the X-Y table.

3.2 PERIPHERALS

Peripherals are those devices which cannot be classed as purely electronic and are not incorporated in the Motor Control Unit. These devices are the Drill, Pneumatic Controller, Stepper Motors and X-Y Table.

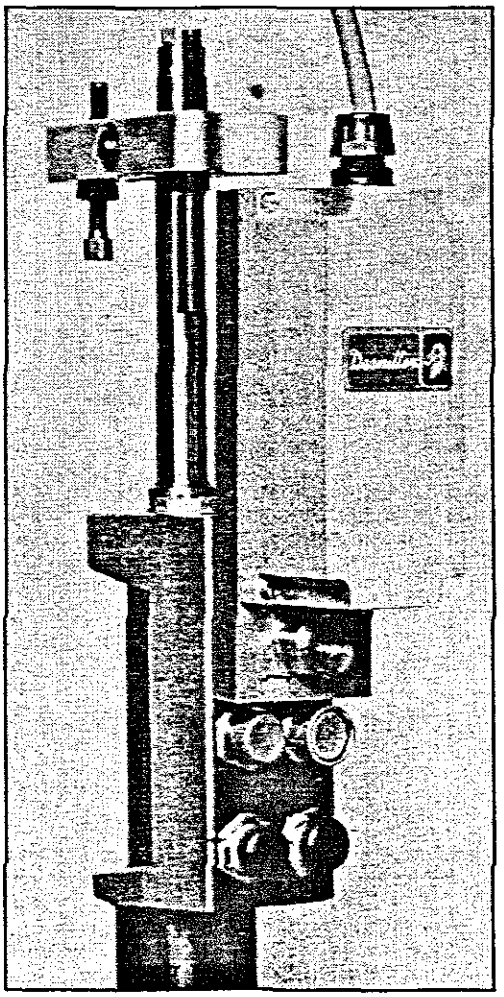
3.2.1 AUTOFEED DRILL AND PNEUMATIC CONTROLLER



AUTOFEED DRILL

The CNC Drilling Machine is fitted with a Desoutter AFDE40-17000 automatic feed drill which is mounted on an overhead gantry above the x-y table. The Drill is powered by a 250 watt three phase geared electric motor which rotates at 17000 revolutions per minute and controlled by pneumatic signalling.

The pneumatic signalling control has two options namely manual control and automatic control via a pneumatic controller. With manual control the "Start" button which is a pneumatic valve, when depressed will advance the drill toward the workpiece. The rate at which the drill advances is determined by "forward stroke speed adjuster". Similarly the "Stop" button control inhibits the advance of the drill and returns the drill to its initial home position . The return stroke is controlled by a value called the "Return Stroke Speed Adjustor."

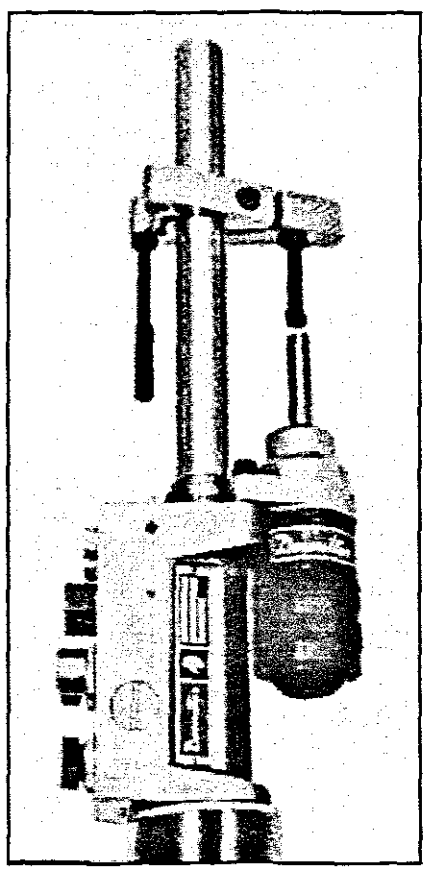


PNEUMATIC CONTROL UNIT

The pneumatic controller is the interface unit that allows the Electrical signals from the motor control unit to be converted into pneumatic signals to control the drill. The signals that the pneumatic controller provide are start, stop and sequencing of the autofeed drill. In addition, signals for pecking and dwell control are provided. This controller is powered by a 24 dc power supply which operates the

solenoids.

Pecking is the procedure when the depth of the hole to be drilled is five or more times the hole diameter. The drill advances and then retracts slightly and advances again. This procedure repeats until the drill has reached its maximum travel and is fully extended. Pecking helps to clear the drill chips and avoids excessive overheating of the drill bit which could cause possible breakage. Dwell control holds the drill at the bottom of the stroke . This may be necessary where improved depth control accuracy is required or to polish the hole.

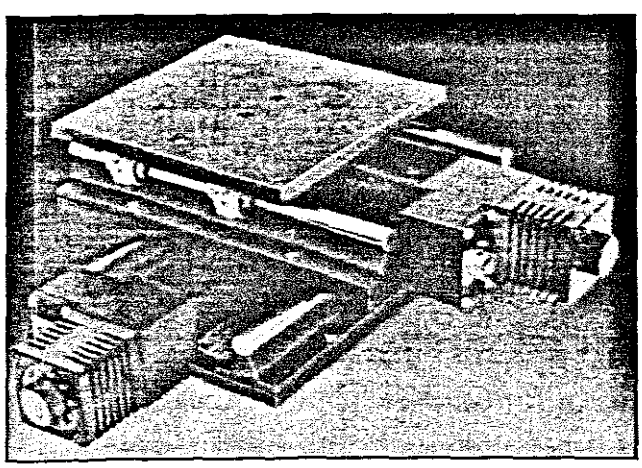


HYDRAULIC CONTROL UNIT

A hydraulic control unit (HCU) is mounted to the drill to act as a damping unit preventing the drill from advancing into the workpiece to quickly. The hydraulic control unit is a fully adjustable constant feed control allowing more controlled feed of the drill while drilling into the workpiece. Controlling the feed speed prevents the drill bit from fragmenting on initial impact on the workpiece and excessive burring is minimized by eliminating acceleration on breakthrough.

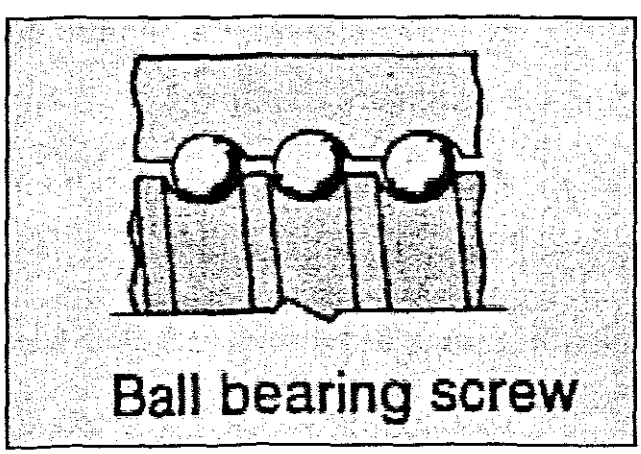
This automatic feed drill is fitted with a Jacobs key adjustable chuck which is interchangeable with a precision collet chuck. The precision collet chuck is used when special high speed Tungsten Carbide drill bits are used.

3.2.2 X-Y TABLE

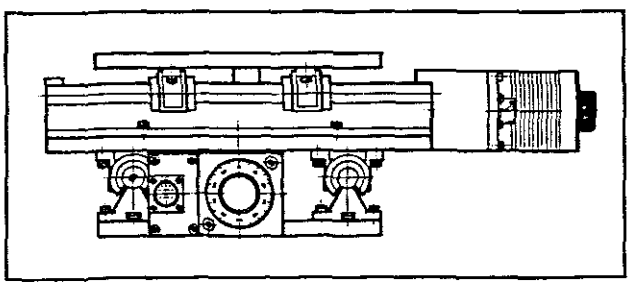


X-Y TABLE

The X-Y table which is manufactured from an aluminium alloy which has been anodised matt black and is very rugged and reliable. The X-Y table is driven by a four phase 200 step per revolution (full step) stepper motor, which gives repeatability within one motor step and an accuracy of approximately 10 microns. The table has a travel of 550 mm and can achieve a peak speed of 150 mm per second. The stepper motors of both the X and Y axes are driven by ball bearing lead screws with a pitch of 4 mm per revolution, giving a movement of 20 microns per motor step (full step) and 10 microns per half step. The double re-circulating ball nut with wiper seals runs on hardened and ground steel rails and are flash chromed for corrosion resistance. The leadscrews are protected from dust by PVC bellows.



The X-Y table has limit switches fitted on the ends of both axes to prevent the table exceeding its maximum travel. Hand wheels with graduated dials are fitted to the stepper motor shafts which enables manual positing of the X-Y table when the stepper motors are de-energised. The stepper motors are enclosed and are fitted with heat sinks for protection and accuracy requirements.All the electrical functions are enclosed and brought out to two "Jaeger 19 pin " connectors.



X-Y TABLE(side view)

The datum system consists of a dual switch system of microswitches and a optical sensor which has a open collector output.When these sensors are combined with the logic from the Datum modules, this system guarantees an absolute mechanical reference point (Datum point) to within one motor step.

The auto-datum circuit, which is operated in the Unidirectional mode returns the x-y table to a fixed mechanical reference point, is accomplished by monitoring three signals from the X-Y table. They are the following: Datum switch, Datum disc (optical sensor) and motor zero phase. In the Unidirectional mode the table will always approach Datum from the same side.

The auto-datum circuit which is a more complex process, involves returning the x-y table to a fixed mechanical reference point. This point is referred to as the Datum point. To accomplish this three signals have to be monitored. They are the following: Datum Switch, Datum Disc and Motor Zero Phase.

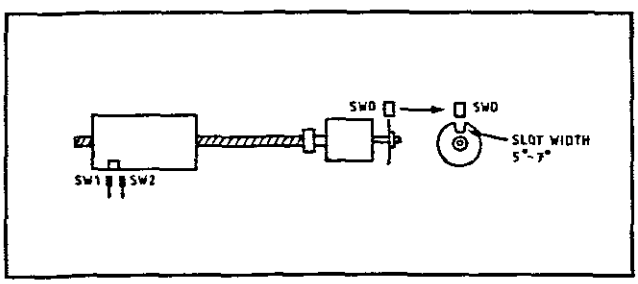
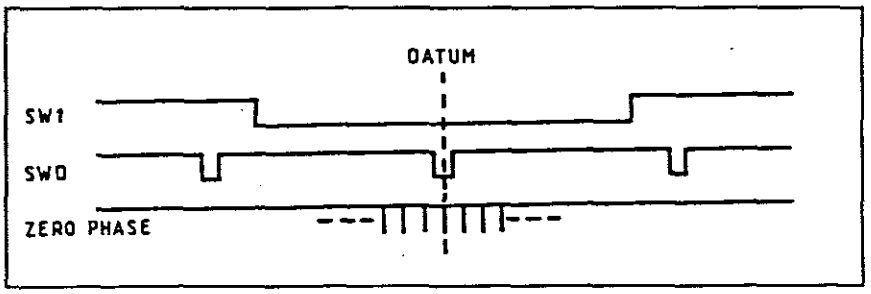


TABLE AND DISC SWITCH ARRANGEMENT

(SW1-Limit switch 1)
(SW2-Limit switch 2)
(SWD-Datum disc)

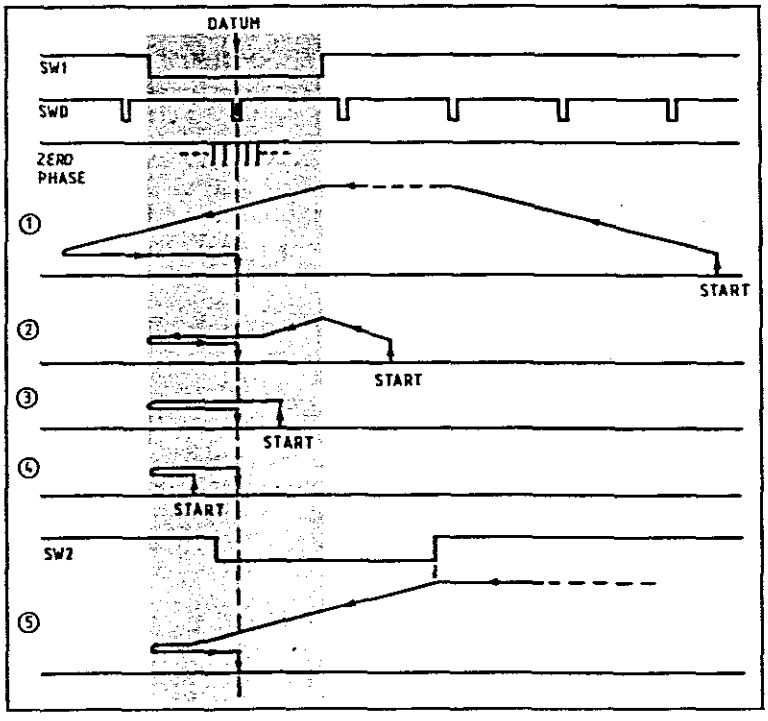
The leadscrew driven linear system is employed to drive the x-y table. On the coupling connecting the leadscrew to the stepper motor, a flat reflective surface is machined. This acts as a mirror to reflex the light source from the optical switch. The optical switch generates the "Datum Disc" signal. As the X-Y table approaches datum it activates a micro-switch which creates the "Datum switch" signal. When this signal is present the next signal to occur is the "Datum Disc" signal.

When these two signals are present, the remaining signal which must be generated is the "Zero Phase" signal which only occurs every 8 steps, when the stepper motor is running in the 400 steps per revolution mode. When this signal is present, the stepper motor will continue to step a few steps and then reverse until all three signals are present again. This process of moving past the datum point and then reversing back toward it, is to eliminate the "backlash" of the table. At the datum point, the datum card generates three signals Datum X, Datum Y and then At Datum XY.



DEFINITION OF UNIQUE DATUM POINT

When Datum X or Datum Y has been generated, the step input to the stepper motor on the particular axis is terminated. When both signals are present, the AT Datum XY signal is generated. The X-Y table has now reached the Datum Point (home).



UNIDIRECTIONAL APPROACH PATTERN

CHAPTER 4

SERIAL COMMUNICATION INTERFACING

The communication between the Personal computer which acts as the process controller and the Motor Control Unit is achieved by asynchronous RS232C serial communication as specified by the Electronic Industries Association (EIA). (CAMPBELL,1989:9) This RS232C standard is commonly used as a serial interface standard and is used by industry to avoid both signal and physical incompatibilities. The RS232c standard specifies the following areas:

(1) The mechanical characteristics for the interface, etc., connects configuration and connector types.

(2) The electrical signal characteristics and voltage levels.

(3) The specification of the function of each electrical signal across the interface.

(4) The procedures that need to be followed for certain communication applications.

The standard specifies that the maximum cable length should not exceed 15 meters with a maximum signal rate of 20 Kilo bits per second. They also specify the maximum signal voltage of +/- 12 volts with minimum voltage of +/- 3 volts. Negative voltage above 3 volts are interpreted as binary "1" while positive 3 volts are binary "0". The standard connector is a 25 pin and most commonly used is a DB25 way connector. The standard supports bidirectional communication, therefore full duplex communication is possible.

Normally Data Terminal Equipment (DTE) is connected to Data Communication Equipment (DCE) directly, but there are occasions when DTE are connected to DTE. This scenario presents a problem as the transmit pins and receive pins of the two DTE's are connected together. To solve this problem a device called a "Null Modem" is used. This null modem swaps the transmit and receive pin on the null modem side so that the transmit of the one DTE is connected to the others receive pin and vice versa. The other control or handshaking signals that are also cross connected are RTS to CTS and DSR to DTR. Instead of a null modem, cross connections can also be hard-wired on the cable to suit specific applications. (GROFTON,1986:13)

Writing the communication interface routines for the serial port in the C language is quite a complex task, especially if handshaking is required. As the C language has no facilities or fixed routines available to communicate with the UART (Universal Asynchronous Receiver and Transmitter). Therefore all asynchronous serial communication must be programmed at systems level and in order to accomplish this, a good understanding must be gained of the functions of the 8250 UART.

The main functions of the UART are:

(1) To convert parallel data from the Central Processing Unit (CPU) into serial data for transmission. And to convert the serial data which is received from an external source into parallel information.

(2) When transmitting a character, the necessary start, stop and parity bits are added to the information and when the information is received these bits are removed from the received character.

(3) The UART detects errors in parity and baud rate and reports these faults.

(4) Transmits characters at the appropriate baud rate which has been preset and calculates the parity bit on transmission.

(5) The UART sets up the relevant hardware handshaking signals and reports status of the incoming handshaking signals.

The UART has several registers which are internal memory locations. These registers hold information on the next character to be transmitted or the last character that was received, current status information of handshaking signals and any errors detected on transmission or reception of a character.

The UART registers are divided into three categories, namely control registers, status registers and buffer registers. The four control registers are: Line Control Register, Modem Control Register, Interrupt Enable Register and Baud Rate Divisor Latches.

The Line Control Register is used to set up the communication parameters. These parameters are Character Length, Stop Bit Length, Parity and Divisor Latch Access Bit.

The Modem Control Register controls the handshaking signals sent by the UART. These signals are Data Terminal Ready (DTR) and Request To Send (RTS).

The Interrupt Enable Register (IER) enables four types of interrupts which independently activate the INT pin of the CPU whenever an event occurs. The four types of interrupts which can be enabled or disabled are Receive Data Interrupt Enable (RXDA),Transmit Data Empty Enable (TXDA), Receive Interrupt Enable (RXIE) and Modem Interrupt Enable (MIE).

The Baud Rate Divisor Latch is a set of two registers which contain the number by which the clock input (1.8432 Mhz) must be divided to obtain the required Baud Rate.
(Microcommunications,1990:2-28)

The Status Registers are Line Status Register, Modem Status Register and Interrupt Identification Register. These registers report the internal status in the UART.

The Line Status Register (LSR) reports the information concerning receipt and transmission of data. Each individual bit has a different meaning. These bits are Data Ready, Parity Error, Overrun Error, Framing Error, Break Interrupt, Transmitter Holding Register Empty and Transmit Shift Register Empty.

The Modem Status Register reports the conditions present on the handshaking lines. These handshaking lines are Clear To Send (CTS), Data Set Ready (DSR), Ring Indicator (RI) and Received Line Signals Detect (RLSD).

The interrupt identification register holds the enabled active interrupt with the highest priority and the interrupt pending signal. The source of the interrupt request can be determined by evaluating bit 2 and bit 1. These two bits reflect the highest priority interrupt requested. The four interrupts are Line Status, received Data available, Transmitter holding register and Modem Status.

The third category of register are the buffer registers. These two buffer registers are namely, Transmitting and Receiving registers. The Receiver Buffer Register holds the

last character received and Transmitter Holding Register holds the next character to be transmitted by the UART.

Through the knowledge gained and a considerable amount of experimentation, a routine was developed to initialise the computer hardware for serial communication. For effective five wire communication, it was found that a strict procedure had to be followed to initialise the UART's various registers. Failing to follow this procedure caused numerous errors to occur, sometimes with disastrous effect to the entire system.

The developed procedure below is divided into a number of steps:

(1) First, all the registers need to be saved so that on completion of the program, these registers can be restored to their original values.

(2) Then the Interrupt Enable Register (IER) must be set to zero. This causes the UART to disable all interrupts and therefore inhibit all interrupt requests.

(3) The Modem Control Register (MCR) is also set to zero disabling the hardware buffer and all the handshaking signals.

(4) The UART is now ready to be re-configured according to the user's specifications. The first register to be set is the Line Control Register (LCR). By setting this register the Stop Bit Length, Character Length and Parity parameters are set.

(5) The Baud Rate is set by determining the Baud Rate Generator (BRG) Divisor. The formulae for this calculation is:

Baud Rate Divisor = BRG / Baud Rate / 16 bits.

The frequency of the BRG is equal to 1.8432 Mhz. This Baud Rate Divisor is 16 Bits which is then divided into a High Byte and a Low Byte. The LSR is then again accessed and Bit 7 is set. This is the Divisor Latch Access Bit (DLAB) which allows access to the Divisor Counter Register. The High Byte and Low Byte are loaded into their respective registers and the DLAB Bit is reset. (Microcommunications,1990:2-29).

(6) The Modem Control Register is set to the users specifications and the hardware buffer are enabled.

For effective serial data transfer without the waste of the Central Processing Unit's (CPU) machine cycles by polling for interrupts, an interrupt input / output routine was written. This routine is known as an interrupt handler and is the most productive way of servicing interrupts. This routine only services an interrupt on request from the UART. (GROFTON,1990:132)

CHAPTER 5

INITIALISATION PROCEDURES

5.1 MOTOR CONTROL UNIT INITIALISATION

The process controller or PC needs to set up a communication link with the Motor Control Unit (MCU). To establish this communication link with MCU certain procedures need to be followed. When the power is first applied to the MCU, the green light emitting diode (led) on the power supply module comes on and will stay on as long as the system is powered up. The red led on the microprocessor serial index lights up for approximately 1 second . During this time the Indexer runs through its internal initialisation routine.
(Electronic motion control and automation,1990:D5-7)

Once the indexer has initialised itself, the process controller which is a Personal Computer (PC) should transmit a parenthesis character "(", at the baud rate which the

operator has selected, while configuring the serial communication parameters. This topic is discussed in Chapter 2. When the indexer receives this character, it can determine the rate at which the character was sent, and therefore the baud rate. The indexer responds by flashing the red led twice per second and then transmitting a "U" character at the new baud rate. The PC should receive the character that was sent correctly. If the character was not received correctly, the parenthesis character must be retransmitted repeatedly until the "U" character is received. All the alpha characters are transmitted in upper case. The "U" character is followed by a carriage return which is designated <CR>. Although the parenthesis can be transmitted as often as necessary, the indexer only detects the baud rate on the first "(" received.

After the baud rate has been established, the PC controller must send information to set up the data format. There are two alternative formats available to suit the requirements of different controllers. The "U" format is terminated only by a carriage return <CR>, whilst the "V" format is terminated by a carriage return plus a line feed which is designated by <CR><LF>. If the "U" format is used, the indexer expects only a carriage return to terminate all the

instructions and it will similarly terminate all status and position data with <CR>. When the "V" format is selected, all the following instructions must be terminated by a line feed and the indexer will terminate all the returned data with a line feed plus a carriage return <CR><LF>.
(Electronic motion control and automation,1990:5-7)

For the CNC Drill software design, the "U" format option was chosen. The "U" format control instruction is in the form of "UX1X2<CR>". The "U" in the instruction string specifies the format being used. The X1 represents a table from which the programmer can select the number of stop bits, number of data bits and the significance of the eighth bit to suit the particular controller being employed. The X2 represents a table of options which can be selected to determine the parity and echo back. This table can also determine whether hardware or software flow control (handshaking) is used. Both involve signals returning from the receiving device to the transmitting device. With hardware flow control, the receiving device sends a positive voltage along a dedicated handshaking line as long as it is ready to send. When the transmitting computer receives a negative voltage the computer will stop sending data. With software flow control,

the handshaking signals consist of special characters transmitted along the data line, rather than using handshaking lines. (GROFTON,1986:8)

The character string transmitted for setting up the data configuration must be sent in the ordered sequence as discussed above. If an error occurs while transmitting any of the characters in the string, the whole string must be retransmitted. After the format instruction has been transmitted, a delay of approximately 5 mS must be allowed for the indexer to re-initialise.

When the indexer receives the correct data format instruction, the red led will cease flashing and the indexer is ready to receive move instructions. If the led comes on again it means that a communication error or critical fault condition has occurred, which must be cleared. An instruction of up to 64 characters in length can be received by the indexer and stored in its input buffer. The communication parameters that were chosen for the "U" format are: 8 data bits, 2 stop bits, no parity and no echo back.

5.2 INTERRUPT SERVICE ROUTINE

There are two methods of handling data to and from the Universal Asynchronous Receiver and Transmitter (UART), namely Polling and Interrupt Handlers. The cycle of examining status, reading, examining status and so on, is known as Polling.

Interrupt service routine is the most effective method of transferring serial data via the UART without retarding the Central Processing Unit (CPU). If the polling method is used the CPU is required to examine the UART continuously, normally in a loop in the program, until either the UART is ready to transmit or receive the expected data.

Interrupts are typically generated when a character has been received or transmitted or when the handshaking signals change. (GROFTON,1990:130) To implement interrupt- driven I/O a section of code must be installed at systems level in the interrupt vector table. This section of code is called the Interrupt Handler. Whenever a interrupt is received from the UART the computer branches to the interrupt handler. The address to which it must branch resides in the interrupt vector table.

There may be different interrupts for different events, or just one interrupt. In the latter case it is necessary for the computer to examine a special register to find out what caused the latest interrupt. Sometimes the UART can be programmed to generate interrupts on certain events and not others. (GROFTON,1986:183). The Interrupt Enable Register bits must be set to determine which events should cause an interrupt. The interrupt handler first examines the Interrupt Identification Register to determine the cause of the interrupt. The four interrupts which can be generated by the UART are Data available, Transmitter holding register empty, Line status and Modem status. The interrupt handler then stores the information in a buffer which will be examined by the interrupt service routine, which is part of the main program. The main program, which also contains the interrupt service routine and the interrupt handler, run simultaneously on the computer. They communicate with each other by leaving data in a designated buffer.

Programming, at systems level involves configuring the hardware which generates the various interrupt signals. These interrupts are prioritised by a Programmable Interrupt Controller (PIC). The IBM PC has eight IRQ (interrupt request) lines, while IBM PC-AT has sixteen IRQ lines. These IRQ line designations are shown in the table below.

IBM PC AND PC-AT IRQ LINES.

IRQ LINE	IBM PC IRQ DEVICE	IBM PC-AT IRQ DEVICE
NMI	Nonmaskable interrupt	Nonmaskable interrupt
0	Timer	Timer
1	Keyboard	Keyboard
2	Reserved	Gate from controller 1 to 2
3	Serial port 2	Serial port 2
4	Serial port 1	Serial port 1
5	Hard Disk	Parallel port 2
6	Floppy disk	Floppy disk
7	Parallel port 1	Parallel port 1
8	N/A	Clock
9	N/A	Redirection from IRQ 2
10	N/A	Reserved
11	N/A	Reserved
12	N/A	Reserved
13	N/A	Coprocessor
14	N/A	Hard Disk
15	N/A	Reserved

The IRQ lines are not directly connected to the CPU, but via the dedicated Programmable Interrupt Controller chip (PIC). The IBM XT has one PIC controller chip while the IBM AT contains two controllers. The PIC prioritises the interrupts and assigns the highest priority to IRQ 0 and lowest priority to IRQ 7. The setting up of the priorities is done when the Power On Self Test (POST) routine is executed, from the bios when the computer boots up.

The PIC has a register which enables the various interrupts. By default IRQ 3 and IRQ 4 are not enabled. When implementing Interrupt Driven I/O, an instruction must be passed to the PIC to enable the appropriate interrupt lines. First the value of the PIC register must be read at port address 21H. The IRQ3 and IRQ4 are enabled by setting bit 3 or 4 of the binary value obtained from the PIC, depending on which IRQ line is required. This new value is then sent back to the port address. PIC is now enabled and the selected UART interrupts can be set.

When the Pic receives an interrupt, assuming that the appropriate interrupt has been enabled and no other interrupt is pending, it asserts a positive voltage on the INT line to the CPU. The CPU then acknowledges the interrupt by means of a signal to the PIC, and requests the PIC to indicate which interrupt occurred. The PIC sends a number (via the data bus) to the CPU (this number is 8 plus the IRQ number). The CPU then executes the appropriate section of code by saving the current program address on the stack and executing a far call to memory location pointed to by the interrupt vector for that interrupt. The code that causes the jump through the interrupt vector table is called the interrupt handler, and points to the code that reads the interrupt and acts upon it, which is known as the interrupt service routine.

CHAPTER 6

CAD FILE TRANSLATION

One of the major design considerations was which Computer Aided Design (CAD) software packages should be used to interface to the Computer Numerically Controlled (CNC) Drill, and how to translate the CAD packages into a code that would be compatible with the hardware which was selected for the design.

Some of the software packages on the market produce a Drill Tape while others can only produce a plotter file output. From the investigation it was established that the Drill Tapes did not conform to any particular standard and that they were specific to the software manufacturer. Similarly all the CAD software packages supported three main plotters output file formats. These being DMP51, HPGL and DFX formats. The DMP51 and HPGL plotter languages were the most common and supported by the two largest plotter manufacturers namely Houston Instruments and Hewlett Packard. After careful consideration it was decided that the Houston Instruments plotter language DMP51 would be the easier plotter language to translate for the CNC Drill.

For many other CNC machines such as Lathes and Milling machines the defacto standard is the Gerber format which is used to input data into the machines, but none of the Printed Circuit Board (PCB) CAD packages supported this language format.

To extract the coordinates for the printed circuit board and the pad layouts, the CAD package had to be configured so that it would generate a DMP51 ASCII file.(Smartwork,1986:D-1). The translation software that was written would then read this file at a rate 512 bytes per file access into memory. This data would be systematically examined byte for byte and if any coordinate data was detected, this information would then be stored in an array containing either X or Y data. This process would continue until the whole plotter output file had been read.

The information contained in the X and Y arrays which was in string format, is then converted into integer format. Each coordinate is then checked to ascertain if any of the coordinate pairs appeared twice in the arrays.

A routine was developed to detect the offset coordinate of the drill relative to the edge of the Printed Circuit board which is at coordinates (0,0). To determine the distance between each pad the offset coordinate is then subtracted from the first set of values in the arrays. The resultant values are the first set of coordinates which will be sent to the Motor Control Unit, which will cause the X-Y table to move. The next value is determined by last computed value which is subtracted from the next new value read. Each time a set of X and Y coordinates is computed, the CNC drill's X-Y table will move the exact number of machine steps and the drill operation will be activated. This process will continue until the entire Printed Circuit Board has been drilled.

Once the last hole has been drilled, the CNC drill will return to its original start position and wait for the next set of instructions from the Motor Control Unit.

The operator can view the entire drilling operation on the Personal computer's monitor and can also instruct the drill to rout the printed circuit board on completion of the drilling operation. The operator also has the facility available to terminate the operation at any stage.

CHAPTER 7

DISPLAY GRAPHICS

7.1 OVERVIEW:

The Display Interface software package is required by the operator of the CNC machine, to show information pertaining to the job being executed.

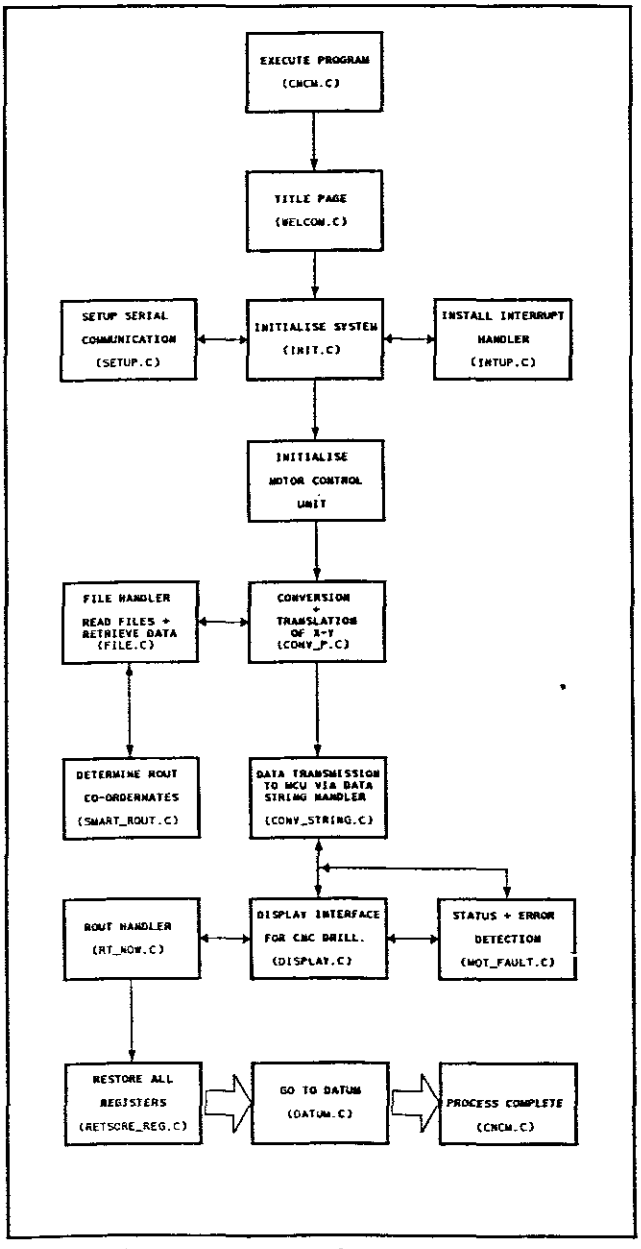
While the CNC Drilling Machine is executing a particular drill pattern, the screen will display the specific information relating to the job in progress. This makes it much easier for the operator to determine the status of a specific job. As the CNC drill moves from one location to another, the graphical representation of the Printed Circuit Board (PCB) will indicate to the user which hole is being drilled. Data on the PCB parameters will also be displayed on the screen. These parameters being board size and total number of holes.

Other information which should appear on the display is status information. Status information being X and Y coordinates and the hole number which is being drilled, and

the status of the Motor Control Unit (MCU).

If errors occur in the system while executing a specific task, these error messages should then be displayed. Provision has also been made for user intervention.

7.2 INTERFACE SPECIFICATION



SOFTWARE FLOW DIAGRAM

The Display Interface software will obtain data from various existing routines. The coordinates for the graphical representation of the PC board will be obtained from two integer buffers which are generated by a module called FILE.C. The data for these arrays are read from the CAD layout file into a file buffer. The coordinates are then extracted from this file buffer, translated into X and Y coordinates and stored in the coordinate arrays. This file read process is repeated until the complete file has been read.

A counter is employed to determine the total amount of coordinates stored in these coordinate arrays. This counter value will provide the display interface with the total number of holes detected on the PCB.

The rout coordinates which determine the size and coordinates for the border of the PCB, are stored in a separate two dimensional array which is a function of the SMART_ROUT routine.The display interface must be activated whenever the drilling or routing process commences. This is initiated when the program calls the modules CONV_P.C, CONV_STRING.C or RT_NOW. The Error Messages for the MCU are generated by a routine called MOT_FAULT.C . This routine also produces Status Messages of the state on the MCU.

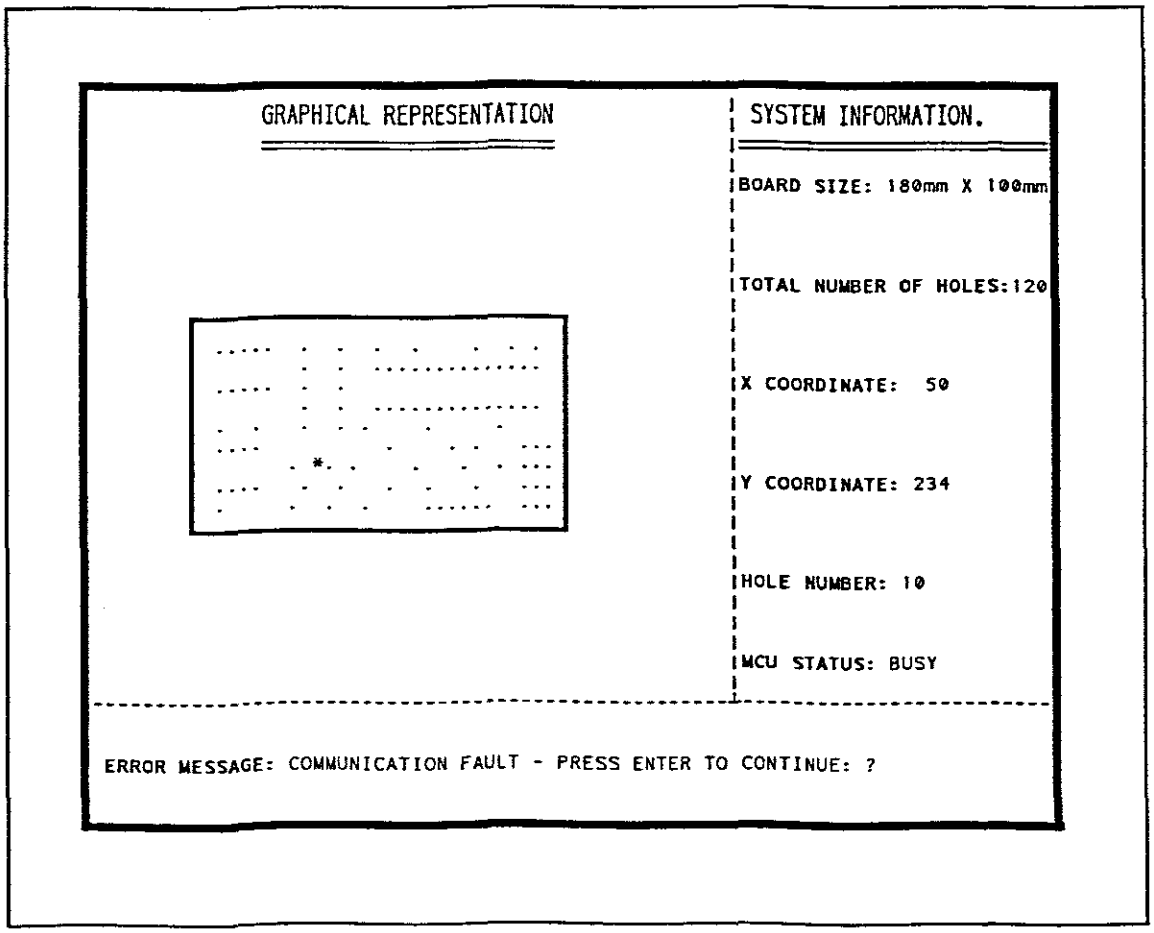
7.3 FUNCTIONAL SPECIFICATION

Specific areas need to be assigned to the screen for various functions. These areas are the following:

(a) Graphical representation field

(b) System information field

(c) Error message field



SCREEN FORMAT

The information that appears in these fields are the following:

(1) A graphical representation of the PCB being drilled.

(2) Total number of holes detected on PCB.

(3) The size of the PCB.

(4) The number of holes completed by CNC Machine.

(5) Status of MCU.

(6) Error messages when errors occur.

(7) X and Y coordinates of CNC Machine.

The method that has been employed to represent all the information on the display interface is that all the fields are displayed on the screen simultaneously.

A procedure has been developed to scale the PC boards to ensure that the entire board is displayed. Another function that is available for the operator is a facility for user intervention either to input data or abort the system.

If graphic errors are detected the program notifies the operator and then aborts. On completion of the drilling and routing process, the screen is restored to its original operation mode.

7.4 DESIGN CONSIDERATIONS:

7.4.1 OPERATION:

The Display Interface is a set of routines designed to interface with various existing routines in the CNC program. These routines are divided into three separate categories namely Display Set-up, Display Implementation and Operation routines. The Operation routines are the existing routines which control the CNC Machine and which pass various types of data to the display routines.

The Display Set-up routines create a screen image which is later called by the main program. This routine is called as soon as the PC Board to be drilled is identified. These routines are Set Image_Screen, Scale_Pcb and Display_pcb. Initially Set Image_screen is called which is configured to monitor and initialise the graphics mode. This routine then calls Scale_pcb which determines the size of the PC board from information gained from the Operation routines. The PC board is then scaled to fit into a particular area on the screen. The Display_pcb is then called, which first plots the outline of the scaled down version of PC board. The pads

(holes) are then generated. The screen cursor's position is monitored by a counter which remembers the cursor last position.

The system information headings are generated by the system_info routine and the error message headings by Error_message.
This complete screen image is saved in memory to be used later by the main routine,i.e Call Graphics.

The principle on which the Call Graphics works is to recall the screen image which was created by Set Image_Screen. This is done to increase the speed at which the screen can be updated. Once the image is displayed the cursor is moved to a new pad which is being operated upon. The systems information is displayed next to the relevant headings. An operation routine, Mot_fault is called, which monitors the status of the Motor Control Unit (MCU) and any errors that may occur during MCU operation. This routine also checks to see if there was any user intervention and if so, the program is aborted.

7.4.2 ERROR HANDLING:

There are two sources of errors and these can be generated by either the graphic mode or the MCU. The graphic mode will only generate a fault if the hardware doesn't support the graphics or the graphical functions used are not present.

The other type of errors which are generated by the MCU are detected by the MOT_FAULT routine. These errors are machine and communication errors. Most of these errors are critical errors and therefore when they are detected the program is aborted.

The errors and the ability of the control system to handle the error condition are discussed in more detail in Chapter 8.

7.4.3 DATA STRUCTURES:

There are a number of different data structures called from the Display Interface routine. The first routine is the RT_buf which is a two dimensional integer buffer containing the X and Y coordinates for the border of the PC board. These

values are also used to determine the size of the PC board. This buffer can be found in smart_rout.c routine.

The next important structure is that of coxbuf and coybuf, which are two one dimensional floating point arrays of 512 elements, which store the coordinates of the pad positions. These two data structures are found in the file handler. (file.c).

The graph_buffer which is a two dimensional integer buffer with 512 elements, is used by Scale_pcb to store the scaled down version of the PC board.The rest of the data is of the integer type.

7.5 INTERFACE SOFTWARE DEVELOPMENT:

The Display Interface Routines were developed using a software development tool called "PROGRAM DESIGN LANGUAGE" (PDL) which performs the similar role to design tools like flow charts and structure charts. This PC based development tool is simple to use, with the added advantage over the other development tools that it allows the programmer to read, review and refine their designs quickly. The detailed

design can readily be translated into code and remains as useful documentation for debugging, testing and maintenance.(Dinkelacker,1991:1.1) This language which is written in "Pseudo English" allows the routines and data to be presented in a readable format with useful references and indexes. Typical features of this tool include comments , written descriptions, procedure and data structures. With all these utilities available, error checking is simplified and the effect of modification to procedure can be easily monitored. The Display Interface Routine using PDL is shown in Appendix B.

CHAPTER 8

ERROR DETECTION

For a control system to operate efficiently, it must have good error detection and error management procedures. Errors in the CNC Drill are generated by two main sources: namely, software and hardware.

The software errors that are generated occur mainly due to problems when the Personal Computer (PC) is in the graphic mode. The errors that arise, vary from "not enough memory to load the graphics driver" to "not being able to detect the graphics hardware". Most of these errors are critical errors and therefore when they are detected the programme is aborted. These errors are the following:

Graphics Error Message Table

DECIMAL VALUE	ERROR MESSAGE
0	No Error
1	(BGI) Graphics not installed (Use initgraph)
2	Graphics hardware not detected
3	Device driver file not found
4	Invalid device driver file
5	Not enough memory to load driver
6	Out of memory in scan fill
7	Out of memory in flood fill
8	Font file not found
9	Not enough memory to load font
10	Invalid graphics mode for selected driver
11	Graphics error
12	Graphics I/O error
13	Invalid font file
14	Invalid font number
15	Invalid device number
16	Reserved for Turbo C
17	Reserved for Turbo C
18	Invalid version number of Turbo C

The hardware errors are generated by the Motor Control Unit (MCU) and PC. These errors relate more to communication or motion status problems between MCU, PC and the other hardware peripherals. The errors are detected by the "MOT_FAULT"

routine which decides on the severity of the error, and then makes a decision whether or not to terminate the process or to take corrective action.

The MOTOR_FAULT routine also monitors the motion status and reports the status of the CNC Drill back to the process controller at the time of the request.

If the MOTOR_FAULT routine detects a problem, the process controller then interrogates the MCU to clarify if the error is a communication or input related fault ie. limit switches. Typical communication faults are Parity errors, Overrun errors, Framing errors, Data faults and Out-of-range errors. The input status errors are concerned with X and Y motor faults, Emergency stop and limit switches which have been activated. The input status interrogation is also interested in input from the other peripherals which need to be serviced by the MCU.

All these faults are reported to the operator before the process controller terminates the program.

CHAPTER 9

RECOMMENDATIONS

The development of the Computer Numerical Controlled (CNC) Drill was a resounding success and many objectives set forward in the proposal were achieved. In reviewing the design, enhancements have been conceptualised which would further improve the system and increase both the versatility and capabilities of the CNC machine. It is suggested that these options should be investigated and implemented at a later stage . The conceptual improvements are discussed below:

With the present CNC Drill development the Printed Circuit Board (PCB) has been aligned manually on the table. This method of squaring each PCB to be drilled, to the edge of the table, is a tedious task. If an Optical Targeting System is used to align the PCB, this process would be much quicker and simpler. To implement an optical targeting system, a Closed Circuit Television (CCTV) camera with a close-up lens set can

be used. On the close-up lens set a cross hash and centre can be imprinted. If the CCTV is mounted perpendicular to the table, on the drill gantry which is parallel to the table, then using the target, the PCB can be aligned with ease. The PCB can be aligned either on the X or Y axis, depending which side of the PCB is square with the PCB layout. For example, if the PCB is square on the X axis, then the X-Y table could be moved to the left hand side of the X axis, and the cross hash positioned on the edge of the PCB. The X-Y table is then moved to the right hand side of the X axis. If the cross hash is not precisely on the edge of the PCB, it means the X axis is not parallel to the table. The PCB can then be adjusted accordingly and clamped down on the table before a final alignment inspection is made.

Another alignment method which can be employed is that of using two reference points on the PCB. With this method the operator will view the PCB on the graphic user interface. The first and last pad to be drilled will be highlighted. The operator will then use the CCTV targeting system to move the X-Y table, so that

the centre of the cross hash is positioned directly on top of the first pad of the PCB. The process controller will then store the coordinates of this position. The operator will then move to the last pad to be drilled, and this position will also be stored. The process controller will then, by means of Pythagoras's Theorem, mathematically adjust all the X and Y coordinates which are stored in the coordinate buffers.

The CCTV camera can also be used to digitise the PCB layout, so that the PCB's with the incorrect CAD file formats can be drilled using the digitised image of the PCB. Another alternative is using the CCTV camera to target and store the position of each pad on the PCB. This teaching of the process controller can be done by moving from one pad to the next and storing coordinates of the pad in a two dimensional array. Once the process controller has learnt the drill pattern of all the X-Y coordinates, it can write these coordinates to a file and drill the PCB.

In the present design, the drilling head of the CNC Drill is controlled by a pneumatic controller which the

Motor Control Unit (MCU) cannot control precisely. If the drilling head which is the Z axis, could be controlled by a stepper motor in the micro step mode, the drilling head could be precisely controlled. With this modification, the drill would acquire a number of new features. The CNC Drill could be used as a Milling Machine which could be interfaced to a mechanical engineering type of Computer Aided Design (CAD) package, for example, Autocad. If this feature is combined with the teach/learn option, the CNC Drill could become a versatile engraver.

A further improvement which could possibly be developed, is that of the user interface and graphics being in colour as opposed to monochrome. The error messages could then be displayed in, for example red text, while the PCB information could be in blue text, and the PCB layout in green with the white background. Although this improvement is not essential, it would make the user interface more aesthetically pleasing and user friendly.

An enhancement which could increase production speed is that of developing a chuck changing mechanism and routine to facilitate bit/ tool changes. At present this process, is being done manually by the operator.

In conclusion, the success of the CNC Drill development is highlighted by the recognition received from the South African Institute of Electrical Engineers (SAIEE) for the best student paper delivered for 1994, refer to Appendix D.

If the recommendations which are proposed above are implemented, the CNC Drill will be transformed into an extremely versatile machine, which would out class most of its competitors. This CNC Drill will be beneficial to the small PCB manufacturers who chose to incorporate this development in their production line.

CHAPTER 10

GLOSSARY OF TERMS

Acceleration

The change in velocity as a function of time. Acceleration usually refers to the increasing velocity and deceleration describes decreasing velocity.

Accuracy

A measure of the difference between expected position and actual position of a motor or mechanical system. Stepper Motor accuracy is usually specified as an angle representing the maximum deviation from the expected position.

Autofeed Drill

The type of drill used in the CNC design which is driven by a 3 phase electric motor and controlled by pneumatic signals.

ASCII

American Standard Code for information Interchange. This code assigns a number to each numeral and letter of the alphabet. In this manner, information can be transmitted between machines as a series of binary numbers.

Backlash

This refers to the mechanical play which is in X-Y table.

Baud rate

Number of bits transmitted per second.

Bidirectional Approach

The X-Y table can approach the datum point from both directions.

Closed loop

A broadly applied term relating to any system where the output is measured and compared to the input. The output is then adjusted to reach the desired condition. In motion control the term is used to describe a system wherein a velocity or position (or both) transducer is used to generate correction signals by comparison to desired parameters.

Datum (Home)

A reference position in a motion control system, usually derived form a mechanical datum. This reference point is determined by limit switches and optical discs on the x-y table. often designated as the "zero" position.

Feed Speed

The rate at which the drill advances into the workpiece.

Handshaking Signals (Flow Control Signals)

RTS: Request To Send

CTS: Clear To Send

DSR: Data Set Ready

DTR: Data Terminal Ready

Hydraulic Control Unit (HCU)

Controls the feed speed of the Autofeed drill by hydraulic damping which can be adjusted.

Limits

Properly designed motion control systems have sensors called limits that alert the control electronics that the physical end of travel is being approached and that motion should stop.

Micro Stepping

An electronic control technique that proportions the current in a stepper motor windings to provide additional intermediate positions between poles. Produces smooth

rotation over a wide speed range and high positional resolution.

Motor Control Unit (MCU)

This unit consists of a number of electronic modules which are used to control the CNC Drill.

Parity

An error detection scheme which can detect transmission errors.

Pneumatic Control Unit

A module which is attached to the Autofeed Drill and converts pneumatic signals into electrical signals for drill control by the MCU.

Process Controller

The name given to an intelligent controller that processes data and controls the CNC drilling operation.

Ramping

The acceleration and deceleration of a motor. May also refer to the change in frequency of the applied step pulse train.

Repeatability

The degree to which the positioning accuracy for a given move performed repetitively can be duplicated.

Resolution

The smallest positioning increments that can be achieved. Frequently defined as the number of steps required for the motor's shaft to rotate one complete revolution.

RS-232C

A data communication standard that encodes a string of information on a single line in a time sequential format. The standard specifies the proper voltage timing requirements so that different manufacturers devices are compatible.

Speed

Used to describe the linear and rotational velocity of a motor or other object in motion.

Start Bits

Serial Character transmissions begin with a bit which signals the receiver that the data is now being transmitted.

Step Angle

The angle the shaft rotates at upon receipt of a single step command.

Stop Bits

When transmitting serial characters, one or two bits are added to every character to signal the end of the character.

Unidirectional Approach

The X-Y table is only permitted for approaching the Datum point from one direction. The approach pattern prevents "backlash".

CHAPTER 11

BIBLIOGRAPHY

1. BESANT, C.B. & LUI, C.W.K. 1986; Computer Aided Design and Manufacture: Ellis Horwood Limited

2. CAMPBELL, J. 1986; Crafting C tools for the IBM PCS: Prentice Hall

3. CAMPBELL, J. 1988; C Programmer's guide to serial communications: Prentice Hall

4. CAMPBELL, J. 1989; The RS-232 solution: Sybex

5. CHORAFAS, D.N. 1987; Engineering productivity through CAD/CAM: Great Britain: Butherworth

6. COOMBS, C.F. 1979; Printed circuits handbook: New York: McGraw-Hill

7. CULLEY, S.J. 1985: "Justifying CAD/CAM -the steps." Effective CAD\CAM: IMechE Conference publications 1985-7

8. DINKELACKER, AM & WALKER, AJ 1991; PDLPROC - User and reference manual: Seal, Dept. Electrical Engineering - Wits University

9. Electronic motion control and automation 1990:Microdyne

10. EZZELL, B. 1989; Graphics programming in Turbo C 2.0: Addison-Wesley Publishing

11. GOFTON, P.W. 1986; Mastering serial communication: Sybex

12. LAFORE, R. 1990; C Programming using Turbo C++: The Waite Group inc: Mc Millian Computer Publishing

13. Microcommunications 1990; Volume 1: I n t e l Corporation

14. SCHILDT, H. 1987; Turbo THE Complete Reference: Orland-Osborne / McGraw-Hill

15. STARK,J. 1988; Managing CAD/CAM: McGraw-Hill

16. TEICHOLZ,E. 1985; CAD/CAM Handbook: New York: McGraw-Hill

17. The CAD/CAM Handbook 1980; C o m p u t e r v i s i o n Corporation: Bedford, Massachusetts. (eds) MACHOVER,C & BLAUTH,R.E

18 Turbo C User guide version 2.0 1988; Borland

19. Turbo C Reference guide version 2.0 1988; Borland

20. Smartwork 1986; Wintek Corporation

APPENDIX A

Division of Information Services
(Incorporating the SA Technology Information System – SATIS)
PO Box 395 Pretoria 0001 South Africa
Telefax : (012) 86-2869
International + 27 12 86-2869
Telex : 3-21287 SA
Telephone : (012) 841-2000
International + 27 12 841-2000

Direct line : (012) 841-2297

15/6/1

Information Services

CSIR

Mrs Matthys
Cape Technikon
P O Box 652
CAPE TOWN
8000

Dear Mrs Matthys

5 June 1991

NATIONAL PROJECT REGISTER (NPR)

I refer to your request of 4 June 1991, concerning computer numeric control on drilling machines combined with CAD software interface.

The following keywords (with Afrikaans equivalents) were used in the search:

Compute$ numeric$ control drilling machine
CAD software interface

($ is the truncation symbol)

No projects on the above subject are listed on our database. The database includes all information processed to date, but since the information for the National Register is supplied on a voluntary basis, our records are not necessarily complete. The database is updated till April 1991. This was the last update.

If you need any further information please contact me at (012) 841-2297.

Yours sincerely

G v. d. Walt

Mien van der Walt

PROJECT REGISTER:
TECHNOLOGY MANAGEMENT AND
BUSINESS INFORMATION PROGRAMME

APPENDIX B

PROGRAM DESIGN LANGUAGE GRAPHICS SOURCE CODE

```
%TITLE  ***** DISPLAY INTERFACE *****

%REVISION 1.02-TURBO C VER 2.0


%G MAINPROG


%D GLOBAL DATA (ALL VARIABLES ARE GLOBAL)

rt_buf                  border coordinates stored in buffer
(smart_rout)
cursor_counter     stores x-y position of screen cursor
scale_factor       scaling factor of pcb
scale_size         scale size of pcb
coxbuf_            X coordinate buffer in file_p.c for hole
coybuf_            Y coordinate buffer in file_p.c for hole
max_screen          total number of characters which can be
displayed(X-Y)
graphics_rep       max. size of the area in which PCB can be
displayed
system_info_size      size  area  designated  for  systems
information
MCU_status         condition of the motor control unit
PCB_size           actual size of PCB

graph_buffer[dot_p] stores scaled X-Y coordinates of holes
dot_p              pointer in graph_buffer


%P MAINPROC.. Call graphics which is called from conv_string

if (dot_p == 0)
set cursor_counter to offset value
endif
if user intervention then read input
```

```
restore screen to normal text mode
abort program
endif
if ( dot_p not equal  holes)
read graph_buffer[dot_p]
cursor_counter = graph_buffer[dot_p]
increment dot_p
display cursor on hole
call mot_fault
endif



%P SET IMAGE_SCREEN....Creates image screen
clear screen
initialise graphics mode
if (error in initialising)
print ERROR MESSAGE
abort program
endif
set screen resolution
determine max_screen
display system_info headings
display error_messages headings
SCALE_PCB
DISPLAY_PCB



%P SCALE_PCB...Procedure to determine scale of PCB
scale_factor = 1
dot_p = 0
graphics_rep = max_screen - system_info_size
read PCB_size from rt_buf (Largest and smallest X-Y value)
scale_size = PCB_size
if scale_size < graphics_rep then read
X-Y coordinates from coxbuf_ and coybuf_
copy to graph_buffer
elseif scale_size > graphics
while scale_size > graphics_rep
if scale_size > graphics_rep then scale_factor + 1
scale_size = PCB_size / scale_factor
save scale_factor
else
repeat
read X-Y coordinates from coxbuf_ and coybuf_
```

```
divide X-Y coordinates by scale_factor
save X-Y coordinates in graph_buffer[dot_p]
increment dot_p
until all X-Y coordinates from coxbuf_ and coybuf_
endif
endwhile
endif
PRINT_XY PCB_size



%P DISPLAY_PCB....Display the border and holes
dot_p =0
repeat
read boarder coordinates from rt_buf
plot border of pcb
until border is complete
while graph_buffer still full
read graph_buffer [dot_p]
if graph_buffer not empty
cursor_counter = graph_buffer[dot_p]
increment dot_p
PRINT_XY (symbol to indicate hole)
elseif empty
reset cursor_counter to [1,1] (return cursor to home)
endif
endwhile



%P MOT_FAULT..Existing routine developed to monitor motor
status!

while MCU_status is busy
PRINT_XY MCU_status
if (systems error)
PRINT_XY(Error Message)
abort program
endif
endwhile

%P SYSTEM _INFO...Displays system and status information
headings

PRINT_XY "BOARD SIZE:"
PRINT_XY "TOTAL NUMBER OF HOLES:"
```

```
PRINT_XY "X COORDINATE:"
PRINT_XY "Y COORDINATE:"
PRINT_XY "HOLE NUMBER:"
PRINT_XY "MCU STSATUS:"
%P ERROR_MESSAGE...Displays error message headings

PRINT_XY "ERROR MESSAGE:"


%P PRINT_XY...Prints all data / strings in graphics mode

get format (character or string)
get data / string
move screen cursor to XY position
erase data area according to string length at XY position
write data string to allocated area
```

```
**************************************

***** DISPLAY INTERFACE *****

1.02-TURBO C VER 2.0

Time  19 : 12

Date  1 3 1993

**************************************
```

TABLE OF CONTENTS

```
        ***** DISPLAY INTERFACE *****                    1 3
1993    PAGE 1 -  1


                     ****************************************


                                        MAINPROG


                     ****************************************


        ***** DISPLAY INTERFACE *****                    1 3
1993    PAGE 1 -  2
         MAINPROG


GLOBAL DATA (ALL VARIABLES ARE GLOBAL)

-  -  -  -  -  -  -  -  -  -  -  -  -  -  - D -              2  A
-D----------------------------------------

      D  1      rt_buf border coordinates stored in buffer
(smart_rout)
     D  2      cursor_counter stores x-y position of screen
cursor
     D  3     scale_factor scaling factor of pcb
     D  4     scale_size scale size of pcb
      D  5      coxbuf_ X coordinate buffer in file_p.c for
hole
     D  6      coybuf_ Y coordinate buffer in file_p.c for
hole
    D  7     max_screen total number of characters which can
be displayed(X+++
     D  8     graphics_rep max. size of the area in which PCB
can be display+++
```

```
      D  9      system_info_size size area designated for
systems information
     D 10     MCU_status condition of the motor control unit
     D 11     PCB_size actual size of PCB
       D 12       graph_buffer [dot_p] stores scaled X-Y
coordinates of holes
     D 13     dot_p pointer in graph_buffer


MAINPROC.. Call graphics which is called from conv_string

- - - - - - - - - - - - - - - P -                    2 B
-p---------------------------------------

     P  1     if (dot_p == 0)
     P  2         set cursor_counter to offset value
     P  3     endif
     P  4     if user intervention then read input
     P  5         restore screen to normal text mode
     P  6         abort program
     P  7     endif
     P  8     if ( dot_p not equal holes)
     P  9         read graph_buffer[dot_p]
     P 10         cursor_counter = graph_buffer[dot_p]
     P 11         increment dot_p
     P 12         display cursor on hole
     P 13         call mot_fault
     P 14     endif



        ***** DISPLAY INTERFACE *****                 1 3
1993   PAGE 1 -  3
       MAINPROG


SET IMAGE_SCREEN....Creates image screen

- - - - - - - - - - - - - - - P -                    3 A
-p---------------------------------------

     P  1     clear screen
     P  2     initialise graphics mode
     P  3     if (error in initialising)
```

```
     P  4          print ERROR MESSAGE
     P  5          abort program
     P  6      endif
     P  7      set screen resolution
     P  8      determine max_screen
     P  9      display system_info headings
     P 10      display error_messages headings
     P 11      SCALE_PCB
     P 12      DISPLAY_PCB


SCALE_PCB...Procedure to determine scale of PCB

- - - - - - - - - - - - - - - P -                    3  B
-P-----------------------------------------

     P  1      scale_factor = 1
     P  2      dot_p = 0
     P  3      graphics_rep = max_screen - system_info_size
     P  4     read PCB_size from rt_buf (Largest and smallest
X-Y value)
     P  5      scale_size = PCB_size
     P  6      if scale_size < graphics_rep then read
     P  7          X-Y coordinates from coxbuf_ and coybuf_
     P  8          copy to graph_buffer
     P  9      elseif scale_size > graphics
     P 10          while scale_size > graphics_rep
      P 11                if scale_size > graphics_rep then
scale_factor + 1
       P 12                      scale_size = PCB_size /
scale_factor
     P 13                save scale_factor
     P 14            else
     P 15                repeat
      P 16                      read X-Y coordinates from
coxbuf_ and coybuf_
      P 17                      divide X-Y coordinates by
scale_factor
       P 18                       save X-Y coordinates in
graph_buffer[dot_p]
     P 19                    increment dot_p
      P 20                  until all X-Y coordinates from
coxbuf_ and coybuf_
     P 21              endif
     P 22          endwhile
```

```
     P 23      endif
     P 24      PRINT_XY PCB_size
        ***** DISPLAY INTERFACE *****                    1 3
1993   PAGE 1 -  4
        MAINPROG


DISPLAY_PCB....Display the border and holes

- - - - - - - - - - - - - - - P -                        4 A
-P---------------------------------------

     P  1      dot_p =0
     P  2      repeat
     P  3          read boarder coordinates from rt_buf
     P  4          plot border of pcb
     P  5      until border is complete
     P  6      while graph_buffer still full
     P  7          read graph_buffer [dot_p]
     P  8          if graph_buffer not empty
     P  9              cursor_counter = graph_buffer[dot_p]
     P 10              increment dot_p
     P 11              PRINT_XY (symbol to indicate hole)
     P 12          elseif empty
     P 13              reset cursor_counter to [1,1] (return
cursor to home)
     P 14          endif
     P 15      endwhile


MOT_FAULT..Existing routine developed to monitor motor
status!

- - - - - - - - - - - - - - - P -                        4 B
-P---------------------------------------

     P  1      while MCU_status is busy
     P  2          PRINT_XY MCU_status
     P  3          if (systems error)
     P  4              PRINT_XY(Error Message)
     P  5              abort program
     P  6          endif
     P  7      endwhile
```

```
SYSTEM _INFO...Displays system and status information
headings

- - - - - - - - - - - - - - - P -                    4 C
-P---------------------------------------

     P  1      PRINT_XY "BOARD SIZE:"
     P  2      PRINT_XY "TOTAL NUMBER OF HOLES:"
     P  3      PRINT_XY "X COORDINATE:"
     P  4      PRINT_XY "Y COORDINATE:"
     P  5      PRINT_XY "HOLE NUMBER:"
     P  6      PRINT_XY "MCU STSATUS:"



___
        ***** DISPLAY INTERFACE *****                1 3
1993    PAGE 1 -  5
        MAINPROG


ERROR_MESSAGE...Displays error message headings

- - - - - - - - - - - - - - - P -                    5 A
-P---------------------------------------

     P  1      PRINT_XY "ERROR MESSAGE:"



PRINT_XY...Prints all data / strings in graphics mode

- - - - - - - - - - - - - - - P -                    5 B
-P---------------------------------------

     P  1      get format (character or string)
     P  2      get data / string
     P  3      move screen cursor to XY position
     P  4      erase data area according to string length at
XY position
     P  5      write data string to allocated area
```

PROCEDURES

```
  2B            1    MAINPROC.. Call graphics which is called
from conv_stri+++
  3A            2    SET IMAGE_SCREEN....Creates image screen
  3B            3    SCALE_PCB...Procedure to determine scale
of PCB
  4A             4     DISPLAY_PCB....Display the border and
holes
  4B             5     MOT_FAULT..Existing routine developed
to monitor motor +++
  4C              6      SYSTEM _INFO...Displays system and
status information h+++
  5A            7     ERROR_MESSAGE...Displays error message
headings
  5B            8     PRINT_XY...Prints all data / strings in
graphics mode
```



```
      PROCEDURE DEFINED
                REFERENCED IN PROCEDURE
                      LINES
  4A  DISPLAY_PCB....Display the border and holes
  5A  ERROR_MESSAGE...Displays error message headings
    2B    MAINPROC..  Call  graphics  which  is  called  from
conv_string
  4B  MOT_FAULT..Existing routine developed to monitor motor
```

```
status!
  5B  PRINT_XY...Prints all data / strings in graphics mode
  3B  SCALE_PCB...Procedure to determine scale of PCB
  3A  SET IMAGE_SCREEN....Creates image screen
  4C  SYSTEM _INFO...Displays system and status information
headings

___
        ***** DISPLAY INTERFACE *****                    1 3
1993   PAGE 4 -  1
        DATA INDEX


      DATA ITEM
                REFERENCED IN SEGMENT
                      LINES
      conv_string
           2B  MAINPROC.. Call graphics which is called from
conv_string
                      0
      coxbuf_
           2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                      5
           3B  SCALE_PCB...Procedure to determine scale of
PCB
                      7 16 20
      coybuf_
           2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                      6
           3B  SCALE_PCB...Procedure to determine scale of
PCB
                      7 16 20
      cursor_counter
           2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                      2
          2B  MAINPROC.. Call graphics which is called from
conv_string
                      2 10
           4A  DISPLAY_PCB....Display the border and holes
                      9 13
      DISPLAY_PCB
           3A  SET IMAGE_SCREEN....Creates image screen
```

```
                    12
          4A  DISPLAY_PCB....Display the border and holes
                    0
      dot_p
          2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                    13
          2B  MAINPROC.. Call graphics which is called from
conv_string
                    1  8  9 10 11
          3B  SCALE_PCB...Procedure to determine scale of
PCB
                    2 18 19
          4A  DISPLAY_PCB....Display the border and holes
                    1  7  9 10
      ERROR_MESSAGE
               5A   ERROR_MESSAGE...Displays error message
headings
                    0
      error_messages
          3A  SET IMAGE_SCREEN....Creates image screen
                    10
      graphics_rep
          2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                    8
          3B  SCALE_PCB...Procedure to determine scale of
PCB
                    3
```



```
      DATA ITEM
               REFERENCED IN SEGMENT
                    LINES
          3B  SCALE_PCB...Procedure to determine scale of
PCB
                    6 10 11
      graph_buffer
          2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
```

```
                    12
          2B  MAINPROC.. Call graphics which is called from
conv_string
                    9 10
          3B  SCALE_PCB...Procedure to determine scale of
PCB
                    8 18
          4A  DISPLAY_PCB....Display the border and holes
                    6  7  8  9
    IMAGE_SCREEN
          3A  SET IMAGE_SCREEN....Creates image screen
                    0
    max_screen
          2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                    7
          3A  SET IMAGE_SCREEN:...Creates image screen
                    8
          3B  SCALE_PCB...Procedure to determine scale of
PCB
                    3
    MCU_status
          2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                   10
            4B  MOT_FAULT..Existing routine developed to
monitor motor statu+++
                    1  2
    mot_fault
          2B  MAINPROC.. Call graphics which is called from
conv_string
                   13
            4B  MOT_FAULT..Existing routine developed to
monitor motor statu+++
                    0
    PCB_size
          2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                   11
          3B  SCALE_PCB...Procedure to determine scale of
PCB
                    4  5 12 24
    PRINT_XY
          3B  SCALE_PCB...Procedure to determine scale of
PCB
                   24
          4A  DISPLAY_PCB....Display the border and holes
                   11
            4B  MOT_FAULT..Existing routine developed to
```

DATA ITEM
REFERENCED IN SEGMENT
LINES
4C SYSTEM _INFO...Displays system and status
information headin+++
2 3 4 5 6
5A ERROR_MESSAGE...Displays error message
headings
1
5B PRINT_XY...Prints all data / strings in
graphics mode
0

rt_buf
2A GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
1
3B SCALE_PCB...Procedure to determine scale of
PCB
4
4A DISPLAY_PCB....Display the border and holes
3

scale_factor
2A GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
3
3B SCALE_PCB...Procedure to determine scale of
PCB
1 11 12 13 17

SCALE_PCB
3A SET IMAGE_SCREEN....Creates image screen
11
3B SCALE_PCB...Procedure to determine scale of

```
PCB
                        0
      scale_size
            2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                        4
            3B  SCALE_PCB...Procedure to determine scale of
PCB
                        5  6  9 10 11 12
      system_info
            3A  SET IMAGE_SCREEN....Creates image screen
                        9
      system_info_size
            2A  GLOBAL DATA (ALL VARIABLES ARE GLOBAL)
                        9
            3B  SCALE_PCB...Procedure to determine scale of
PCB
                        3


___
        ***** DISPLAY INTERFACE *****                    1 3
1993    PAGE 5 -  1


                        ************************************


                              END OF DESIGN DOCUMENT


                        ************************************
```

STATISTICS

PROCEDURE COUNT	:	8
PROCEDURE LINE COUNT	:	84
DATA ITEM COUNT	:	22
DATA ITEM USAGE COUNT	:	92
ERROR COUNT	:	0

APPENDIX C

CNC TURBO C SOURCE CODE

CNCM.C

```
/* Main Program call various functions */

#include <stdio.h>
#include "cnc.h"
#include<graphics.h>
#include "display.h"

main(){
welcom();
init();
conv_p();
closegraph();
rt_now();
datum_ask();
restore_registers();
printf("\n\n\n\t\t* TASK COMPLETED ! *\n");
}
```

INIT.C

```
/* Initialise MCU */

#include <stdio.h>
#include <stdlib.h>
#include <dos.h>
#include "cnc.h"
extern k,com_buf[],pt;
void init(){
clrscr();
printf("\n\n\n\n\n\n\a\t");
printf("Warning: PLEASE RESET MCU BEFORE INITIALIZING STEPPER
DRIVES !");
sleep(3);
setup();                    /* set com port parameters  */
```

```
clrscr();
inall();                /* install interrupt handler */
outportb(0x3f8,0x28);
while (com_buf[0] != 'U'){
pt=0;
}
sleep(1);
dsout("U61");
printf("\n\n\n\n\n\n\n\n\n\t\t\t");
printf("COMMUNICATION PORT IS OPEN \n\a");
sleep(2);
clrscr();
dsout(">1     <15     ^9     @4000     ");     /*     Range
speed,Start/Stop,Acceleration,Final speed */
sleep(1);
clrscr();
}
```

INTUP.C

```
#include <dos.h>
#include <stdio.h>
void interrupt inhand(void);
/*Set-up routine for interrupt handler */

int pt = 0;
void inall(void)
{
int pic;
setvect(0x0c,inhand);
pic =inportb(0x21);          /* read pic */
pic = (pic & 0xef);          /* reset IRQ4 bit */
outportb(0x21,pic);          /* new format to pic */
outportb(0x3f9,0xb);         /* set  b rx & modem int */
enable();
outportb(0x20,0x20);         /* reset pic */
sleep(1);
}

#include "cnc.h"
extern flg;
int intnr;


void interrupt  inhand(void)     /*interupt handler */
```

```
{

intnr = inportb(0x3fa);

switch(intnr){

case(0):
     mod_serv();
     break;

case(2):
     flag_r();
     break;

case(4):
     inserv();
     break;

case(6):
     call6();
     break;

     default:
     flg =1;
     break;
          }

outportb(0x20,0x20);
}

extern flg;
flag_r(){
int op;
flg = 1;
op = inportb(0x3fd);
}

extern flg;
call6(){
int s;
flg =1;
s =inportb(0x3fd);
printf("*6",s);
}
```

```
#include "cnc.h"
#include <stdlib.h>
extern flg;
int k;
char com_buf[6];

void inserv(void){              /* Interupt Service Routine */

     int d=0;
     d =inportb(0x3f8);

if(d==0x0d){
     com_buf[pt] = '\0';
     if((pt-1) !=0)
     k = atoi(com_buf);      /* convert string to integer */
     else k = com_buf[pt-1];

     pt = 0;                    /* reset pointer to zero */
     }
else {
     com_buf[pt++] = k = d;    /*load received char into

buffer */

if(pt >= 6)
     pt =0;
      }
 return(k);
}

#include <dos.h>
void mod_serv(void){
 int mstat = 0;

mstat = inportb(0x3fe);

if((mstat && 1) == 1){
  outportb(0x3fc,0x0);

  while((mstat & 48 ) != 48){
     mstat = inportb(0x3fe);
                    }
  }

 outportb(0x3fc,0xb);
  return;
```

```
}
```

FILE.C

```
/* FILE IS A PROGRAM DESIGNED TO READ BLOCKS OF 512 BYTES OF
DATA FROM A SPECIFIED FILE AND EXTRACT X-Y COORDINATES, WHICH
ARE THEN STORED IN A BUFFER. THESE COORDINATES ARE ACCESSED
BY VARIOUS OTHER ROUTINES. */

#include <stdio.h>
#include <stdlib.h>
#include <io.h>
#include <string.h>
#include "cnc.h"

FILE *fp;
unsigned long int ip,end_file;
int xcheck = 0,ycheck = 0;

void conv_f(void){
     char pathname[13],*path;
     clrscr();
     printf("\n\t\t\t\t\t\t( Example: A:\\Filename )\n");
     printf("\n\t* WARNING: Filename must be a Plot File !
*\n");
     printf("\n\n\a     PLEASE     ENTER     FILENAME:
_________\b\b\b\b\b\b\b\b\b" );

scanf("%9s",pathname);                    /* ENTER FILENAME */
     path = ".plt";
     strncat(pathname,path,(13 - strlen(path)));

 /* ADD FILE EXTENSION */

     strlwr(pathname);

if ((fp=fopen(pathname,"rb")) == NULL){

/*OPEN FILE TO READ*/

     printf("\a\n\n\t\t\t\tCANNOT OPEN FILE !\n\a");
     exit(1);
```

```
                              }
     printf("\n\n\n\n\t\tFILE WAS OPENED SUCESSFULLY\n");
     printf("\n\n\n\n\n\t\tPLEASE   WAIT   FOR   COORDINATE
TRANSLATION !\n");
     end_file = (filelength(fileno(fp)));

/* DETERMINE FILE LENGTH */

start_coord();
smart_route();
unpack_buf();
     while(ip < end_file){              /* READ UNTIL EOF */

file_read();

unpack_buf();
                    }

     fclose(fp);

     ref_hole();

     }


char fbuf[512];
int bufp;                              /* FILE READ BUFFER */
long ip_count;                         /* BYTE READ COUNTER */

file_read()
     {
     bufp=0;
     ip_count = ip;

if(fread(fbuf,sizeof(char),512,fp)!=512){

/* READ BLOCK OF DATA */

     if(feof(fp))printf("\nEnd\n");
     else printf("\nError");}

if((ip = ftell(fp)) == -1L){           /* BYTE TOTAL COUNT */
     printf("\aRead Error");}
     ip_count = ip -ip_count;
                 }
```

```
/* A ROUTINE TO EXRACT X AND Y COORDINATES FROM 512 BYTE FILE
BUFFER. */

char xbuf[6],ybuf[6];
coxbuf[500],coybuf[500];
int xp,yp,cy,cx,xb,yb;

unpack_buf()
{

int xact=0,yact=0 ;
bufp = 0;

while(bufp <= ip_count)
{
xp = yp = 0;
     while(!(( xact == 'C')&&(yact == 'A')))
 /* SEARCH FOR "CA" */
     {
          if (bufp == ip_count){
          break;}
          xact = fbuf[bufp ++];
          yact = fbuf[bufp];
     }

          ++ bufp;

if(xact == 'C'){
     while((fbuf[bufp]) != ','){        /*READ X COORDINATE*/
     if(bufp > ip_count)
     break;
     if((fbuf[bufp]) != 32){
     xbuf[xp ++] = fbuf[bufp];
                    }
     ++ bufp;
     if(fbuf[bufp] == ','){
     xbuf[xp ++] = '\0';}           /* TERMINATE X STRING */

                    }
          }



if(yact == 'A'){
          while((fbuf[bufp]) != 32){
```

```
          if(bufp > ip_count)
          break;
          if(fbuf[bufp] != ','){    /* READ  Y COORDINATE */
           ybuf[yp ++  ] = fbuf[bufp];
                          }



++ bufp;
          if(fbuf[bufp] == 32){
          ybuf[yp ++] = '\0';}      /* TERMINATE Y STRING */
                             }
          }

if (bufp <=ip_count){

     xact = atoi(xbuf);     /* CONVERT TO INTERGER */
     yact = atoi(ybuf);

     if((xact != xcheck) ||  (yact != ycheck))
     {
 /*CHECK COORDINATE HAS BEEN REPEATED */
               xcheck = xact;
               ycheck = yact;
                coxbuf[cx ++] = ( xb - xact );
                coybuf[cy ++] = (yb - yact);

               }
               }
}
}


/* A PROCEDURE TO DETERMINE OFFSET COORDINATES FOR DRILL
COORDINATES */

start_coord(){
     xb=0,yb=0;
     bufp = xp = yp =0;

file_read();

     while(!((xb == 'C')&&(yb=='A')))
     {                              /* SEARCH FOR FIRST C A */
     xb=fbuf[bufp ++];
```

```
      yb = fbuf[bufp];
      }
      ++ bufp;
if(xb=='C'){
      while((fbuf[bufp])!= ',')
      {                          /* READ X OFFSET COORDINATE */
      if(fbuf[bufp] != 32)
           {
           xbuf[xp++]=fbuf[bufp];
           }
       ++bufp;
      }
         }

if(yb=='A'){
      while((fbuf[bufp]) != 32)
      {
       if(fbuf[bufp] != ',')
       {                          /* READ Y OFFSET COORDINATE */
         ybuf[yp++] = fbuf[bufp];
         }
      ++bufp;
      }
          }

           xb = atoi(xbuf);         /* CONVERT TO INTERGER */
           yb = atoi(ybuf);
           /*printf("\n xb %d yb %d",xb,yb);*/
}
```

CONV_P.C

```
/* A TRANSLATION ROUTINE TO CONVERT DISTANCE IN INCHES
   INTO MACHINE STEPS. */

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <graphics.h>
#include "cnc.h"
#include "display.h"
extern cx,cy,coxbuf[],coybuf[];
extern system_info_X,MaxY;
float xstep,ystep,ycount,xcount;
```

```
conv_p(){
      int ceb,xact,yact;
      float xlast=0,ylast = 0;
      ycount=0;

/*conv_f();*/
initialise_graph();
      ceb = cx;
      cx=cy=xstep=ystep =0;
while(cx != ceb )
{
      dsout("R1 $");
      gprintf((system_info_X + 109),(MaxY * .68),"%d",cx);
      xact = coxbuf[cx ++];
      yact = coybuf[cy ++];
xstep = (( (xact - xlast) * 25.4)/10);

 /*Conversion to machine steps */

ystep = (( (yact - ylast) * 25.4)/10);

      xlast = xact;                /* Last x step executed */
      ylast = yact;               /* Last y step executed */;
      xcount = xcount + xstep;
      ycount = ycount + ystep;

dsout("@5000");

conv_string();
gprintf((system_info_X + 103),(MaxY * .34),"%6.0f",xstep);
gprintf((system_info_X + 103),(MaxY * .51),"%6.0f",ystep);
drill();
}
xlast =xact;
ylast = yact;
xstep = (( (0-xlast) * 25.4)/10);
/*Conversion to machine steps */
ystep = (( (0-ylast) * 25.4)/10); /* Move to HOME position*/
conv_string();
}


#include <string.h>

conv_string(){
      char xstring[10],ystring[10],gh;
```

```
      itoa(xstep,xstring,10);
      itoa(ystep,ystring,10);

if((xstep != 0) && (ystep != 0)){

      if(xstep < 0)
           xstep = xstep * -1;

      if(ystep < 0)
           ystep = ystep * -1;
if(xstep >= ystep){       /*LINEAR INTERPOLATION X/Y MODE */
      datout("X ");
      datout(xstring);
      datout(" / Y ");
      datout(ystring);
      dsout("@ 2500 $");
}

else if(ystep >= xstep){ /* LINEAR INTERPOLATION Y/X MODE */
      datout("Y ");
      datout(ystring);
      datout(" / X ");
      datout(xstring);
      dsout(" @2500 $");
      }

      /*mot_fault();*/Call_Graphics();
      }

else {if(xstep != 0){                /* LINEAR X MODE */
      datout("X ");
      datout(xstring);
      dsout(" $");
   Call_Graphics();
      }

 if(ystep != 0){                     /* LINEAR Y MODE */
      datout("Y ");
      datout(ystring);
      dsout(" $");

           /*mot_fault();*/Call_Graphics();
           }
           /*gh=getche();*/
           }
           }
```

```
/* MOTION AND FAULT ROUTINE */

extern k,MaxY;
extern system_info_X;

void mot_fault(void)
{
char *m_s;
k = 'E';
      dsout("E");                  /* Check Busy Status */

while(k != 'C'){                  /* Check for Clear Status */
dsout("E");

if(k == 'E')
m_s ="BUSY ";
     else if(k== 'C')
     m_s = "CLEAR";
          else if(k== 'F')
          m_s = "FAULT";

gprintf((system_info_X + 110),(MaxY * .85),"%s",m_s);
if(k == 'F'){                        /* Check Fault Status */
dsout("Q");
gprintf(125,(MaxY-10)," Interface Fault ! %d",k);

if(k == 32)
gprintf(125,(MaxY-10)," - Emergency Stop Activated");
     else if(k == 16)
     gprintf(125,(MaxY-10)," - Drive Fault ");
          else if(k == 128)
          gprintf(125,(MaxY-10)," - Communication Fault");
               else if(k == 64)
               gprintf(125,(MaxY-10),"    -    Limit    Switch
Activated ");

          else {gprintf(125,(MaxY-10)," - Error Code: Q:
%d",k);
                    dsout("K");
                    sleep(1);
                    gprintf(125,(MaxY-10),"Motion: %d",k);}

     sleep(5);
     closegraph();
     exit(0);
```

```
                }

                    }
}

#include "cnc.h"

void drill(void)
{
dsout("P1,10 $");
dsout("R1 $");


}




if(k != 72){
     dsout("P2,55 $");          initiate datum routine
                 }

while((k & 72) != 72){
     dsout("I");                check status
     printf("\n\t\tRX CODES: %d",k);
     if((k & 1) == 1){
     printf("\n\n\t\tEMERGENCY STOP ACTIVATED !\a\n");
     exit(1);}
          }
     printf("\n\n\n\n\n\n\n\t\tX AND Y AXIS ARE AT DATUM.\a
\n");
```

DISPLAY.C

```
/* INITIALISE: Initialises the graphics system and reports
   any errors which occur. */

#include  <graphics.h>
#include <stdio.h>
#include <stdlib.h>
#include "display.h"
#include <dos.h>
#include <conio.h>
```

```
unsigned int MaxX,MaxY;
int ErrorCode,GraphMode,GraphDriver;
double AspectRatio;

void initialise_graph(void)
{
int xasp,yasp;
conv_f();

GraphDriver = DETECT;
initgraph(&GraphDriver, &GraphMode, "" );
 /*Set graphics mode */

ErrorCode = graphresult();       /* Detect graphic errors */
if(ErrorCode != grOk ){
               printf("\nGraphics     System
Error:%s\n",grapherrormsg(ErrorCode));
   exit(1);
   }
/* Set screen resolution /*

if(GraphDriver == 9)
     setgraphmode(2);
     else if(GraphDriver == 1)
     setgraphmode(4);
          else if(GraphDriver == 3)
          setgraphmode(1);
               else if(GraphDriver == 2)
               setgraphmode(5);


MaxX = getmaxx();              /* Maximum number of pixels */
MaxY = getmaxy();

getaspectratio(&xasp,&yasp);       /* Get Aspect ratio */
AspectRatio = (double)xasp / (double)yasp;

     setactivepage(0);
     setvisualpage(0);
     settextjustify(CENTER_TEXT,CENTER_TEXT);
     settextstyle(TRIPLEX_FONT,HORIZ_DIR,4);
     gprintf((MaxX/2),(MaxY/2),"PLEASE WAIT !");
     settextstyle(DEFAULT_FONT,HORIZ_DIR,0);
        settextjustify(LEFT_TEXT,TOP_TEXT);
     sleep(2);
     cleardevice();
```

```
frame();          /* Generate border */

system_info();          /* Write system headings to screen */

error_message();          /* Write error heading to screen */

user_int();               /* Check for user intervention */

scale_pcb();            /* Scale PCB to fit allocated area */

display_PCB();            /* Display select pcb on screen */

}


/* FRAME - Generates the screen border and determines the
graphics area used to display the PC Board */

#include <graphics.h>
#include  "display.h"
int graphic_rep_X,graphic_rep_Y;

void frame(void){
setlinestyle(0,0,3);
line(0,0,MaxX,0);
line(MaxX,0,MaxX,MaxY);
line(MaxX,MaxY,0,MaxY);
line(0,MaxY,0,0);
setlinestyle(0,0,1);
graphic_rep_X = MaxX - 189;        /*Calculate graphics area*/
graphic_rep_Y = MaxY - 20;
line(graphic_rep_X ,0,graphic_rep_X,graphic_rep_Y);
line(MaxX,graphic_rep_Y,0,graphic_rep_Y);
}


/* SYSTEM_INfO - writes all information headings to the
active page */

#include "display.h"
extern cx,ErrorCode;
extern unsigned int MaxX,MaxY;
int system_info_X ;

void system_info(void)
{
```

```
system_info_X = MaxX - 187;
gprintf(system_info_X,(abs((MaxY * 2.6)/100)),"Board Size:
0 X 0 mm");
gprintf(system_info_X,(abs((MaxY  *  17)/100)),"Number  of
Holes: 0");
gprintf(system_info_X,(abs((MaxY * 34)/100)),"X Coordinate:
0");
gprintf(system_info_X,(abs((MaxY * 51)/100)),"Y Coordinate:
0");
gprintf(system_info_X,(abs((MaxY * 68)/100)),"Hole Number:
0");
gprintf(system_info_X,(abs((MaxY * 85)/100)),"MCU Status  :
BUSY");

}


/*Error_message - Writes the Error Message heading to the
screen */

#include "display.h"
void error_message(void)
     {
     gprintf(4,(MaxY - 10),"ERROR MESSAGE : ");
     }



/* Scale_pcb - Determines the correct scale_factor for PCB
and scales all the X-Y coordinates according to the
calculated scale_Factor */

#include <stdio.h>
#include <ctype.h>
#include "display.h"

float scale_factor;
extern signed int rt_buf[2][4];
extern coxbuf[];
extern coybuf[];
extern cx;
extern double AspectRatio;
extern system_info_X;
int graph_buffer[2][500];
int temp_X,temp_Y,small_X,small_Y;
```

```
void scale_pcb(void)
{
int rt_p,rt_x,rt_y,large_X,large_Y,sx,sy;
scale_factor = 1;
temp_X = temp_Y = large_X = large_Y = rt_p = rt_x = rt_y =
sx =sy = 0;
small_X = small_Y = 10000;

while(rt_p < 4){

rt_x = rt_buf[0][rt_p];            /*Read rout buffer */
rt_y = rt_buf[1][rt_p];

if (rt_x < 0)
     rt_x = rt_x * -1;
                                  /* Convert to positive value */
if (rt_y < 0)
     rt_y = rt_y * -1;

if(rt_x < small_X)
     small_X = rt_x;        /* Determine smallest X value */


if(rt_x > temp_X){             /*Determine largest X value */
          large_X = temp_X;
          temp_X = rt_x;
            }
if((rt_x < temp_X) & (rt_x > large_X))
     large_X = rt_x;        /* Find second largest X value */


if(rt_y < small_Y)           /* Determine smallest Y value */
     small_Y = rt_y;

if(rt_y > temp_Y){             /* Determine largest Y value */
          large_Y = temp_Y;
          temp_Y = rt_y;
          }

if((rt_y < temp_Y) & (rt_y > large_Y))
     large_Y = rt_y;        /* Find second largest Y value */

            rt_p++;
            }
small_Y = small_Y * AspectRatio;
```

```
 /* Correct aspectratio for small_Y*/

large_Y = large_Y *AspectRatio;
temp_X = large_X + small_X ;
temp_Y = large_Y + small_Y ;

sx = temp_X;
sy = temp_Y;


/*Determine correct scaling factor for particular PC board
size */

while(!((sx < (graphic_rep_X - 7))&&(graphic_rep_Y > sy)))
     {
     scale_factor = scale_factor + 0.1;
     sx =temp_X / scale_factor;
     sy =temp_Y / scale_factor;
     }

     temp_X = (temp_X / scale_factor);
     temp_Y = (temp_Y / scale_factor) ;


/* Scale all PC Board X-Y coordinates and store in graphics
buffer */

rt_p = 0;
while(rt_p < cx){
graph_buffer[0][rt_p] =  (coxbuf[rt_p]) / scale_factor;
graph_buffer[1][rt_p] = ((coybuf[rt_p]) / scale_factor) *
AspectRatio;
if(graph_buffer[0][rt_p] < 0)
     graph_buffer[0][rt_p] = (graph_buffer[0][rt_p]) * -1;
if(graph_buffer[1][rt_p] < 0)
     graph_buffer[1][rt_p] = (graph_buffer[1][rt_p]) * -1;
     rt_p ++;


}
 /* Approximate board size calculation in mm */
 sx =sy =0;
 sx=(large_X * 25.4)/1000;
 sy = (large_Y * 25.4)/1000;
  gprintf((system_info_X  +  86),(MaxY  *  .026),"%d  x  %d
mm",sx,sy);
```

```
}

/*Display_pcb - Generates the graphical representation of the
PC Board on the active screen. This routine creates both PCB
border as well as the pads.*/

#include <graphics.h>
#include "display.h"

extern temp_X,temp_Y,cx,small_X,small_Y;
extern float scale_factor;
extern graph_buffer[][];
int last_x,last_y,dot_p,count_x,count_y ;

void display_PCB(void)
                {
count_x = count_y = last_x = last_y = 0;

gprintf((MaxX - 56),(MaxY * 0.17),"%d",cx);

/* Print max. holes */

small_X = small_X / scale_factor;
small_Y = small_Y / scale_factor;
setlinestyle(3,0,1);
rectangle(7,7,temp_X,temp_Y);      /* Draw PCB outline */
setlinestyle(0,0,1);

count_x = small_X + 7;

/* Set cursor counter to screen offset */

count_y = small_Y + 7;
circle(count_x,count_y,3); /* Indicate reference hole/pad */

/*Draw all the pads in their respective positions in solid
white */

while( dot_p != cx){
     count_x = count_x + (graph_buffer[0][dot_p] - last_x );
     count_y = count_y + (graph_buffer[1][dot_p] - last_y );
     last_x = graph_buffer[0][dot_p];
     last_y = graph_buffer[1][dot_p++];

     setfillstyle(SOLID_FILL,WHITE);
```

```
     circle(count_x,count_y,2);
     floodfill(count_x,count_y,15);
     }
 last_x = last_y = dot_p = 0;      /* Reset values */
 setfillstyle(EMPTY_FILL,WHITE);
 }


/* A function called to  indicate the drill movement and
position */

#include "display.h"
#include <graphics.h>
extern count_x,count_y,last_x,last_y;

void Call_Graphics(void)
          {
     if(dot_p == 0){
     count_x = small_X + 7;
     count_y = small_Y + 7;
                    }

     user_int();           /* Check for user intervention */

 if (dot_p != cx){
     count_x = count_x + (graph_buffer[0][dot_p] - last_x);
     count_y = count_y + (graph_buffer[1][dot_p] - last_y);
     last_x = graph_buffer[0][dot_p];
     last_y = graph_buffer[1][dot_p++];
     floodfill(count_x,count_y,0);
       circle(count_x,count_y,2);
           }

     mot_fault();

     }



 /* A routine to check for user intervention in a program */

 #include <graphics.h>
 #include "display.h"

 void user_int(void)
 {
```

```
 int u_int =0;
 if(kbhit())              /*Check if keyboard ahs been hit */

     {
     u_int = getch();          /* Retrieve character */
     if (u_int == 27)          /* If Escape key */
          {
          gprintf(125,(MaxY  -10),"USER  TERMINATED  PROGRAM
          !");
          sleep(5);            /* Time delay */
          closegraph();        /* Terminate graphics mode */
          exit(0);            /*Abort program */
          }
     }
 }
```

DSOUT.C

```
/*STRING HANDLER A ROUTINE TO TX A STRING VIA SERIAL PORT */

#include<stdio.h>
#include<string.h>
#include <dos.h>
#include "cnc.h"
extern flg,k;

unsigned char f[20];

datout(d)
char *d;
     {                                /* STRING TO BE PROCESSED*/
     int yt = 0;
     strcpy(f,d);                     /* COPY STRING TO BUFFER */
     /*printf("\n\t\tTX CODES: ");*/

while(f[yt]!='\0')          /* READ BUFFER AND CHECK FOR NULL*/
          {
          for(;;){
          if(flg == 1)
          break;
          }

while((f[yt]!='\0')&&(flg == 1))

/* CHECK BUFFER AND TX FLAG  */
```

```
            {
            flg = 0;
            outportb(0x3f8,f[yt]);
            /*printf("%c",f[yt]);*/
            ++yt;
                  }
                  }
      return;
}

dsout(d){
datout(d);
      for(;;){
      if(flg == 1)
            break;}

            flg = 0;
      outportb(0x3f8,0x0d);              /* SEND <CR>*/

                                        /* 1 SECOND DELAY*/
                  if(flg !=1)
                     sleep(1);


      return;
}
```

GPRINTF.C

```
/* GPRINTF: Used like printf except the output is sent to the
screen in graphics mode at specified co-ordinates.*/

#include <stdio.h>
#include <stdio.h>
#include <graphics.h>
#include <stdarg.h>
#include "display.h"

void gprintf(long int xloc,long int yloc,char *fmt,...)
{
va_list argptr;                  /* argument list pointer */
char str[140];              /* buffer to build string into */
struct textsettingstype textinfo;
```

```
va_start(argptr,format); /* initialise va_functions */
vsprintf(str,fmt,argptr );/* add element to str buffer */
gettextsettings(&textinfo);
erasestr(xloc,yloc,str); /* erase the area first */
outtextxy(xloc,yloc,str );

/* write string in graphics mode */

yloc += textheight("H") + 2;     /* Advance to next line */
va_end(argptr );                 /* close va_functions */
}



/*ERASESTR: Used to erase a portion of the screen before a
new string is written to the screen or simply to remove a
string from the screen.*/

#include <stdlib.h>
#include <graphics.h>
#include <stdio.h>
#include "display.h"

void erasestr(long int xloc,long int yloc,char *str)
{
 struct textsettingstype textinfo;

/*text setting information*/

 int xdim,ydim;
 unsigned long int *hi;                    /*image pointer*/
 void *textimage;                /*check graphic text setting*/
 char *p;
 gettextsettings(&textinfo);  /*get dimension of area */
 switch(textinfo.direction)
 {
     case HORIZ_DIR:  xdim = textwidth(str) + 4;
                      ydim = textheight(str);
                      xloc --;
                      break;
     case VERT_DIR:   ydim = textwidth(str);
                     xdim = textheight(str);
                     yloc++;
                     break;
          }
switch(textinfo.horiz)   /*adjust horizontal position */
```

```
      {
      case LEFT_TEXT:
      break;
      case CENTER_TEXT:xloc-= xdim/2;
      break;
      case RIGHT_TEXT: xloc-= xdim;
      break;

      }
switch(textinfo.vert)          /*adjust vertical position */
      {
      case BOTTOM_TEXT:yloc -= ydim;
      break;
      case CENTER_TEXT:yloc-= ydim/2;
      break;
      case TOP_TEXT:
      break;
}
while(xloc < 0){xloc++;xdim--;}

/*if position is off screen */

while(yloc < 0){yloc++;ydim--;}  /*move back to valid area*/
hi = imagesize(xloc,yloc,xdim,ydim);
outtextxy(100,yloc,*hi);
textimage =  (char *) malloc(*hi);
if(textimage == '\0'){
closegraph();
}
textimage = malloc(imagesize(xloc,yloc,xdim,ydim));
getimage(xloc,yloc,(xloc+xdim),(yloc+ydim),textimage);
putimage(xloc,yloc,textimage,XOR_PUT);
free(textimage);          /*release the memory allocated */

}
```

WELCOM.C

```
#include "graphics.h"
#include "conio.h"
#include "stdlib.h"
void box(int,int,int,int,int);

void welcom(void){
```

```
int driver,mode,monitor,s,maxx,maxy,ErrorCode;
unsigned size;
driver = DETECT;
mode = 0;
initgraph(&driver,&mode,"");
ErrorCode = graphresult();
if(ErrorCode != grOk)
     {
printf("\nGraphics      System
Error:%s\n",grapherrormsg(ErrorCode));
     }

maxx = getmaxx();
maxy = getmaxy();

box((maxx *.01),(maxy * .03),(maxx * .97),(maxy * .89),15);
box((maxx * .02),(maxy * .04),(maxx * .96),(maxy * .87),15);
settextstyle(TRIPLEX_FONT,HORIZ_DIR,5);
outtextxy((maxx * .13),(maxy * .29),"* C.N.C Drill Software
*");
settextstyle(SMALL_FONT,HORIZ_DIR,4);
outtextxy((maxx * .65),(maxy * .76),"Copyright P.Mostert.");
settextstyle(SMALL_FONT,HORIZ_DIR,2);
outtextxy((maxx * .72),(maxy * .80),"1994");
settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
outtextxy((maxx  *  .03),(maxy  *  .72),"Press  ENTER  to
continue");
s=getch();

cleardevice();
restorecrtmode();
if(s != 0X0D){
exit(1);}
}


void box(int startx,int starty,int endx,int endy,int color){
setcolor(color);
line(startx,starty,startx,endy);
line(startx,starty,endx,starty);
line(endx,starty,endx,endy);
line(endx,endy,startx,endy);
}
```

LOCBOD.C

```
#include "cnc.h"
#include <stdio.h>
#include <conio.h>
extern cx,cy,coxbuf[],coybuf[];
ref_hole(){
     char j,gh;

     clrscr();

if((cx==cy) & ((cy==cy) != 0))
printf("\n\n\n\n\n\t\tTOTAL NUMBER OF HOLES: %d\n",cx);
printf("\n\n\n\n\n\t\aTHE REFERENCE HOLE IS LOCATED: ");
     if(coybuf[0] < coybuf[1])
printf(" In the LEFT HAND BOTTOM CORNER !\n");
     else if(coybuf[0] > coybuf[1])
printf("In the  RIGHT HAND BOTTOM CORNER !\n");

printf("\n\n\n\t\tPLEASE MOVE THE DRILL TO THE REFERENCE HOLE
AND ALIGN !\n");
printf("\n\n\t\t\t\tWHEN READY\n");
printf("\n\n\t\tEnter: _\bR to CONTINUE or _\bA to ABORT");
 j = (tolower(getch()));
if(j == 'a'){
     restore_registers();
     exit(0);}
          }



 /* A PROCEDURE TO DETERMINE COORDINATES FOR ROUTING. */

/*The coordinates for the border of the layout are located
in the first 512 bytes of data. The coordinates are located
between two pointers namely L6 and L0. To route only the X-Y
DA coordinates are required. These are stored in rt_buf
buffer for later use by rt_now routine */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
extern xb,yb,x_large,y_large,bufp,xp,yp;
extern char xbuf[6];
extern char ybuf[6];
extern char fbuf[512];
signed int rt_buf[2][4];
```

```
smart_route()
{
      int rt_p =0;
      unsigned char xr,yr;
      bufp = xp = yp = xr = yr = 0;


      while(!((xr == 'L')&&(yr=='6')))
      {                                /* SEARCH FOR FIRST L 6 */
      xr=fbuf[bufp];

      yr = fbuf[++bufp];
      if(bufp == 512)
      continue;
      }
      ++bufp;

while(!((xr == 'L')&&(yr=='0'))){

if((xr=='D')&&(yr == 'A')){
            xp = yp = 0;
           while((fbuf[bufp])!= ','){

/* READ X  COORDINATE */

      while(!(isdigit(fbuf[bufp])))
           ++bufp;

if(fbuf[bufp] != 32)
           xbuf[xp++] = fbuf[bufp];

            ++bufp;
if(fbuf[bufp] == ',')
        xbuf[xp++] = '\0';
                              }
      ++bufp;
      while((fbuf[bufp]) != 32){
if(fbuf[bufp] != ',')          /* READ Y OFFSET COORDINATE */
     ybuf[yp++] = fbuf[bufp];

           ++bufp;

if(fbuf[bufp] == 32)
           ybuf[yp++] = '\0';

                     }
```

```
if(rt_p >4) rt_p =0;
             rt_buf[0][rt_p] = xb - atoi(xbuf);

/* CONVERT TO INTERGER */
             rt_buf[1][rt_p++] = yb - atoi(ybuf);

            }
     xr=fbuf[bufp++];

     yr = fbuf[bufp];
     if(bufp == 512)
     continue;
          if(xr == 'D')
          yr =fbuf[++bufp];

          }

rt_now();
}



#include <dos.h>
extern float xstep;
extern float ystep;

rt_now(){

int xact,yact,rt_p,xrout,yrout;
char gh;
float xlast=0,ylast=0;


rt_p = 0;
xact = rt_buf[0][rt_p];
yact = rt_buf[1][rt_p++];
xstep = (((xact - xlast)*25.4)/10);      /* MOVE TO START */
ystep = (((yact - ylast)*25.4)/10);
xlast = xact;
ylast = yact;
conv_string();
/*Reduce speed for routing.*/
dsout(">1 <2 ^4 @600");

/*Range speed,Start/Stop,Acceleration,Final speed*/
```

```
sleep(1);

while(rt_p < 5){
     if(rt_p == 4){
          xact =rt_buf[0][0];
          yact = rt_buf[1][0];
               }
     else {xact =rt_buf[0][rt_p];
          yact = rt_buf[1][rt_p];
          }
     ++ rt_p;
xstep = (((xact - xlast)*25.4)/10);
ystep = (((yact - ylast)*25.4)/10);
     xlast = xact;
     ylast = yact;
     xrout = xstep;
     yrout = ystep;

if(xstep < 0)
     xstep = xstep * -1;

/*Convert X-Y step positive integer*/

if(ystep < 0)
     ystep = ystep * -1;

if (xstep > ystep){              /*Move Y first*/
        xstep = 0;
     ystep = yrout;
     printf("\n\tUP\a\n");
conv_string();
     xstep = xrout/2;            /*Route half X */
     ystep = 0;
     printf("\n\a\tDOWN\a\n");
conv_string();
     printf("\n\tUP\a");
if(xrout > 0)
     xstep = 100;              /* Leave nipple */
     else xstep = -100;
conv_string();
     if(xrout > 0)
     xstep = (xrout/2) - 100;
     else xstep = (xrout/2) + 100;
     printf("\n\t\aDOWN\a\n");  /* Route remainig X */
conv_string();
               }
```

```
else    {
     xstep = xrout;
     ystep =0;                    /* Move X */
     printf("\n\tUP\a\n");
conv_string();
     xstep = 0;                   /* Route half Y */
     ystep = yrout / 2;
     printf("\n\a\tDOWN\a\n");
conv_string();
     printf("\n\tUP\a");
     if(yrout < 0 )
     ystep = -100;                 /* Leave nipple */
     else ystep = 100;
conv_string();
     if(yrout > 0)
     ystep = (yrout/2) - 100;    /*Route remaining Y*/
     else ystep = (yrout/2) + 100;
     printf("\n\t\aDOWN\a\n");
conv_string();
          }
 }
/*Reset speed to original speed range*/
dsout(">1 <15 ^8 @4000");

/*Range speed,Start/Stop,Acceleration,Final speed*/

sleep(1);
printf("\n\tUP\a");
xstep = (((0-xlast)*25.4)/10);

/*Return to 0,0 coordernate */

ystep = (((0-ylast)*25.4)/10);
conv_string();

     }
```

SETUP.C

```
/* UART STANDARD INITIALISATION PROCEDURE FOR 5 WIRE
COMMUNICATION */

#include <dos.h>
#include <stdio.h>
#include <stdlib.h>
```

```
#include <conio.h>
#include <graphics.h>
#include <ctype.h>
#include "cnc.h"
#define com1 0x3f8
#define com2 0x2f8
unsigned int flg;
void interrupt (*old_vector)();

setup(){
int     baud_rate,data_bits,stop_bits,port,comparm,
BaudRate_Divisor,lp;
char parity,*com_port,menu_select;

baud_rate = 2400;
data_bits = 8;
parity = 'n';
stop_bits = 2;
menu_select = '\0';
com_port = "com1";
port = com1;
comparm = 7;
flg = 1;

save_registers();              /* save some uart registers */
disable();
outportb(0x3fe,0x0);
outportb(0x3f9,0);                  /* disable interrupts */
outportb(0x3fc,0);             /* disable hardware buffer */

while(menu_select != 'x'){
clrscr();
printf("\n\t\t\t Serial Port Parameter");
printf("\n\t\t*---------------------------------------*\n");
printf("\t\t|     A          Baud     Rate              :
%d\t\t\t|\n\t\t:\t\t\t\t\t|\n",baud_rate);
printf("\t\t|     B          Data     Bits              :
%d\t\t\t|\n\t\t|\t\t\t\t\t|\n",data_bits);
printf("\t\t|     C         Parity                      :
%c\t\t\t|\n\t\t|\t\t\t\t\t|\n",parity);
printf("\t\t|     D          Stop     Bits              :
%d\t\t\t|\n\t\t|\t\t\t\t\t|\n",stop_bits);
printf("\t\t|     E           Serial     Port          :
%s\t\t\t|\n\t\t|\t\t\t\t\t|\n",com_port);
printf("\t\t|     X           Exit     Setup           :
Run\t\t\t|\n\t\t|\t\t\t\t\t|\n");
```

```
printf("\t\t| Q  Quit          : Abort\t\t|\n");
printf("\t\t*-----------------------------------------*\n\n");
printf("\t\tPlease Enter Selection to change item value\n\n
");
printf("\t\t\t  Selection : \a");

menu_select = (tolower(getche()));

switch(menu_select){

     case 'a':
     printf("\n\n\t\t\t  [ Options : 4800 2400 1200 600 300
     ]");
          printf("\n\n\t\t\t  Baud Rate : \a");
          scanf("%4d",&baud_rate);

          switch(baud_rate){

               case 4800:
               break;

               case 2400:
               break;

               case 1200:
               break;

               case 600:
               break;

               case 300:
               break;

               default:
               baud_rate = 2400;
               printf("\n\t\t\t\a    An    Illegal    Option
!\a\a\a");
               break;
                    }
           break;

     case 'b':
          printf("\n\n\t\t\t  [ Options : 7 8 ]");
          printf("\n\n\t\t\t  Data Bits : \a");
          scanf("%1d",&data_bits);
```

```
        switch(data_bits){

             case 7:
             comparm = ((comparm & 252) ^ 2);
             break;

case 8:
        comparm = ((comparm & 252) ^ 3);
             break;

        default:
        data_bits = 8;
        comparm = ((comparm & 252) ^ 3);
        printf("\n\t\t\t\a An Illegal Option ! \a\a\a");
             break;
                  }
        break;

case 'c':
        printf("\n\n\t\t\t  [ Options : E N O ]");
        printf("\n\n\t\t\t     Parity : \a");
        scanf("%c",&parity);

        switch(parity){

             case 'e':
             comparm = ((comparm & 199) ^ 40);
             break;

             case 'n':
             comparm = (comparm & 199);
             break;

        case 'o':
        comparm = ((comparm & 199) ^ 32);
        break;

        default:
        parity = 'n';
        comparm = (comparm & 199);
        printf("\n\t\t\t\a An Illegal Option !\a\a\a");
        break;
                  }
        break;
```

```
case 'd':
     printf("\n\n\t\t\t  [ Options : 1 2 ]");
     printf("\n\n\t\t\t  Stop Bits : \a");
     scanf("%1d",&stop_bits);

     switch(stop_bits){

     case 1:
     comparm = (comparm & 251) ;
     break;

     case 2:
     comparm = ((comparm & 251) ^ 4);
     break;

     default:
     stop_bits = 2;
     comparm = ((comparm & 251) ^ 4);
     printf("\n\t\t\t\a An Illegal Option !\a\a\a");
          break;
               }
     break;


case 'e':
     printf("\n\n\t\t\t  [ Options : Com1 Com2 ]");
     printf("\n\n\t\t\tSerial Port : Com\a");
     scanf("%1d",&port);
          if(port ==2){
          com_port = "com2";
          port =com2; }
               else { com_port ="com1";
               port=com1; }
               break;


case 'x':
     printf("\n\n\t\t\t  EXIT\n");
     break;

case 'q':
     exit(1);

     default:
          printf("\n\t\t\t\aAn Illegal option !\a\a");
        }
```

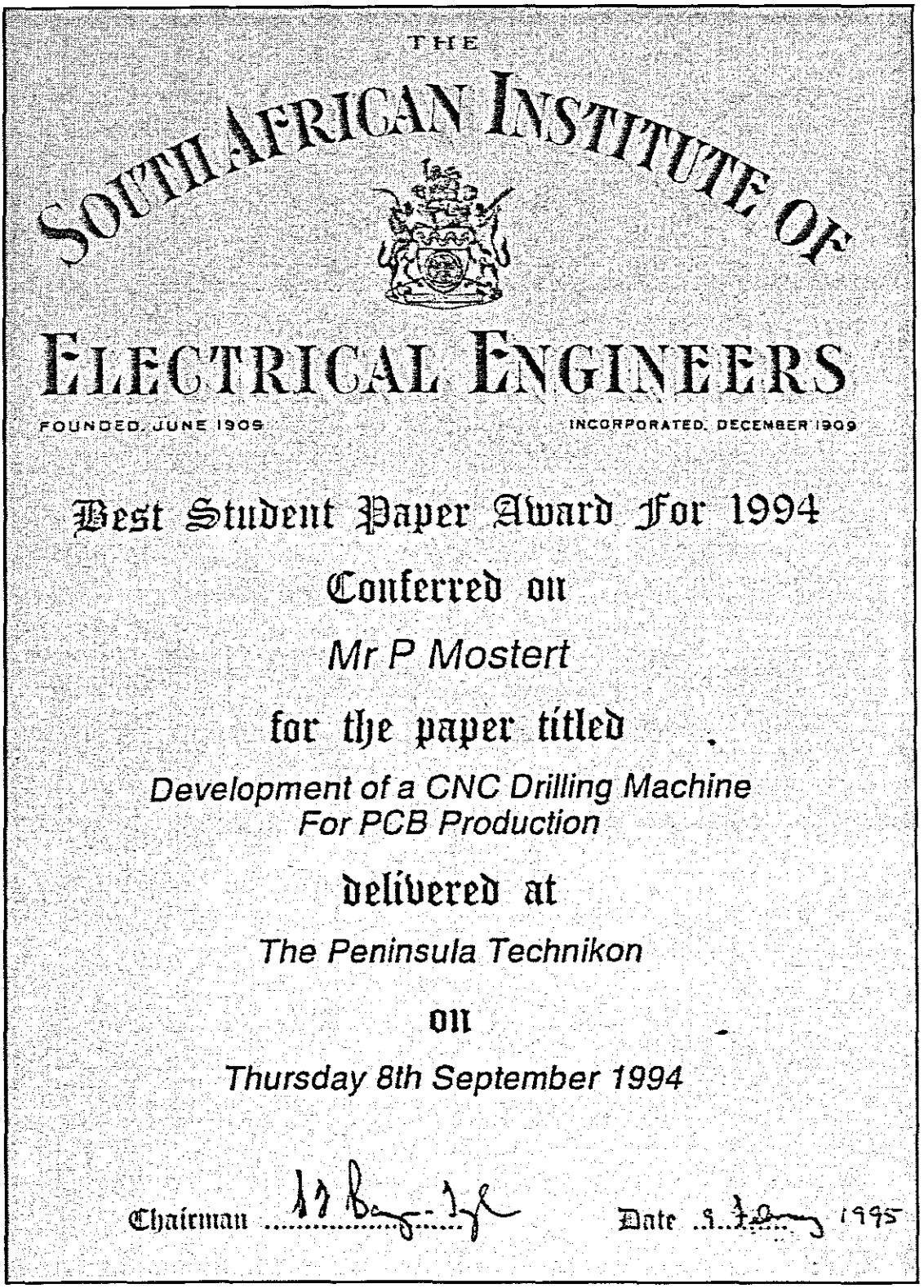
```
}
outportb(0x3fb,comparm);          /* update uart hardware */
comparm |= 0x80;     /* set dlab bit for baud rate access */
outportb(0x3fb,comparm);

     BaudRate_Divisor = 1.8432E06 /baud_rate/16;
     outportb(0x3f8,(BaudRate_Divisor & 255 ));
     outportb(0x3f9,((BaudRate_Divisor >>8) & 1));
     comparm = (comparm & ~0x80);        /* reset dlab bit */
     outportb(0x3fb,comparm);
     old_vector = getvect (0x0c);       /* save old vector */

outportb(0x3fc,0XB);                    /* DTR & RTS & OUT2 */
  }
```

SERVREG.C

```
/* uart_restore().Restore COM1 interrupt vectors and internal
registers prior to exit.*/

#include <dos.h>
void uart_restore(void)

  {
     outportb (0x3fc, 0) ; /*inhibit 8250 interrupts*/
setvect (0x0c, old_vector) ;  /* restore entry vector */
     restore_registers() ; /*restore entry register set */

  }  /*  END: uart_restore()  */

/*save_registers(). This function save the 8250 registers
prior to setting up an interrupt driven environment. These
registers should be restored using the restore_registers()
function before the program terminates.*/

int reg_save[6];
void save_registers (void)

   {
     int x ;          /* work space */
     int port_base ;

for (x = 0, port_base = 0x3f9 ; x < 4 ; x++, port_base ++)
       reg_save [x] = inportb (port_base) ;
```

```
outportb (0x3fb, (reg_save[2] | 0x80)) ;/* set DLAB bit */
     reg_save[4] = inportb (0x3f8) ;
/* get low divisor latch */
     reg_save[5] = inportb (0x3f9) ;/*then high divisor */
     outportb (0x3fb, 0) ;          /* remove DLAB bit */

   }  /* END: save_registers(). */

/* restore_registers(). restore 8250 registers before exiting
from program.*/
#include <dos.h>

restore_registers (void)

    {
     int x, port_base ;        /* workspace */

     outportb (0x3fb, 0x80) ;       /* set DLAB bit */
     outportb (0x3f8, reg_save[4]); /*restore low divisor */
     outportb (0x3f9, reg_save[5]) ;/*then high divisor */
     outportb (0x3fb, 0) ;          /* remove DLAB bit */

for (x = 0, port_base = 0x3f9 ; x < 4 ; x++, port_base++)
outportb (port_base, reg_save[x]) ; /*restore 4 registers */

    }  /* END: restore_registers() */
```

APPENDIX D



THE
SOUTH AFRICAN INSTITUTE OF
ELECTRICAL ENGINEERS
FOUNDED JUNE 1909 INCORPORATED DECEMBER 1909

Best Student Paper Award For 1994

Conferred on

Mr P Mostert

for the paper titled

Development of a CNC Drilling Machine For PCB Production

delivered at

The Peninsula Technikon

on

Thursday 8th September 1994

Chairman ..... Date 1995