PARAMETER [redacted]TORING SYSTEM FOR THE

10.4m MILLIMETREWAVE RADIO TELESCOPE

Project Report

Submitted in partial fulfillment of the requirement
for the award of degree of

Bachelor of Engineering

in

Electronics & Communication

Submitted by :
PARTHA GHOSH
VENKATARAMAN NARENDRA
MARUTHI B.R.

Under the guidance of

External Guide :
Dr. D.K. Ravindra
Head, Department of Electronics,
RRI, Bangalore.

Mr. R. Ganesan
Engineer Incharge,
Telescope Building,
RRI, Bangalore.

Internal Guide :
Prof. S. Rammurthi Rao
Associate Faculty,
Department of Electronics
& Communication.
B.M.S.C.E.
Bangalore.

Department of Electronics & Communication
B. M. S. College of Engineering
Bangalore - 560 019.
1996-97.

RAMAN RESEARCH INSTITUTE

C.V.Raman Avenue, Sadashivanagar, Bangalore - 560 080 India

CERTIFICATE

This is to certify that the project entitled **“PARAMETER MONITORING SYSTEM FOR THE 10.4 METRE MILLIMETREWAVE RADIO TELESCOPE”** has been satisfactorily completed at Raman Research Institute , Bangalore by the following students under our guidance towards the partial fulfilment for the award of the **Degree of Bachelor of Engineering in Electronics & Communication,** during the academic year **1996-97.**

PARTHA GHOSH
VENKATARAMAN NARENDRA
MARUTHI B. R.

Dr. D.K.RAVINDRA
HEAD,
RADIO ASTRONOMY LAB,
RAMAN RESEARCH INSTITUTE,
BANGALORE - 560 080.

Mr. R.GANESAN
ENGINEER INCHARGE,
MILLIMETREWAVE
OBSERVATORY, RAMAN
RESEARCH INSTITUTE,
BANGALORE - 560 080.

DEPARTMENT OF ELECTRONICS & COMMUNICATION,
B.M.S. COLLEGE OF ENGINEERING
BANGALORE 560019.

B.M. SREENIVASIAH COLLEGE OF ENGINEERING
BANGALORE UNIVERSITY

CERTIFICATE

*This is to certify that the project work entitled **"Parameter Monitoring System for the 10.4m millimetrewave Radio Telescope"** has been successfully completed by the following students in partial fulfilment of requirements for the award of the* Bachelor of Engineering in Electronics & Communication *of the Bangalore University during the academic year 1996-97.*

PARTHA GHOSH
VENKATARAMAN NARENDRA
MARUTHI . B. R

Internal Guide
Prof. S. Rammurthi Rao
Assosciate Faculty
Department of Electronics
& communication
B.M.S.C.E.

Head of the Department
Dr. P. S. Satyanaryana
Department of Electronics
& Communication
B.M.S.C.E.

ACKNOWLEDGEMENT

We express our sincere thanks to the management of RAMAN RESEARCH INSTITUTE, Bangalore for having provided us the facilities to carry out the project work.

With utmost pleasure, we hereby extend our sincere and heartfelt thanks to our guides Dr.D.K.Ravindra and Mr.R.Ganesan, for their unlimited co-operation and guidance during the course of our project work.

We gratefully acknowledge the valuable help rendered by Mr.K.Gurukiran, Mr.K.Ramesh and Mr.E.Palanichamy of RRI. We extend our gratitude to Mr.K.V.Sathyaprakash, formerly of RRI, for his guidance in the early stages of the project. We also take this oppurtunity to thank all the staff members of the Telescope Building for extending their support in carrying out this project and making this a success.

We express our profound gratitude to our internal guide, Prof. S. Rammurthi Rao, Assosciate faculty, B.M.S.College of Engineering, Bangalore for his help rendered to us during the tenure of the project.

A special note of thanks goes to our Head of the Department, Dr. P. S. Satyanaryana and the entire staff of the Electronics Department of B.M.S.C.E. for their encouragement and support throughout our project.

Partha Ghosh
Venkataraman Narendra
Maruthi B.R.

15 July,
1997

TITLE: Parameter Monitoring System for the 10.4m millimetrewave Radio telescope

Contents distribution and guidelines

Chapter 1 : INTRODUCTION:

This chapter provides an account of the objectives as well as the environment associated with the project. It is divided into the following sections:

1.1 Radio Astronomy: An Insight

It provides a brief history of radio astronomy and the role it plays in the observation of various celestial bodies.

1.2 The Radio Telescope at Raman Research Institute

A brief description of the Radio Telescope is presented in this section, complete with the specifications.

1.3 Requirement for a Monitoring System

This section lays stress on the various factors responsible for making the monitoring system an absolute neccessity.

Chapter 2 : OVERVIEW OF THE PROJECT

This chapter provides an outline of the project. The following sections are presented in conjunction with this chapter.

2.1 Primary Objectives

This section lays emphasis on the approach towards the project. The project was broken up into several modules and the completion of each module was carried out independent of the other.

2.2 Mode of Working

This section gives a sneak preview at how the different components were organised within the working system and how they interacted with one other.

Chapter 3 : HMPC

This chapter describes the control center of operations, the Health Monitoring PC and the way it communicates with the various control centers (MCMs).

Chapter 4 : DESIGN & DEVELOPMENT

4.1 The Mux Card

This section emphasises the need for an external multiplexer card in the working system as well as provides a detailed description of the steps taken towards the development of such a card.

4.2 The Voltage Limiter Card

This section provides a detailed account of the development of the conditioning card which proved to be such a crucial factor for the proper working of the system.

Chapter 5 :HARDWARE IMPLEMENTATION : Components and their Utilities

This chapter takes a closer look at the different components used in the hardware side of the project along with the wide range of utilities they offer. Key components like the **SBC 80186, SCC Z8530,** the **multiseial i/o card** and **the Digitizer Card** (ADC) are covered in detail.

Chapter 6 : SOFTWARE ORGANIZATION

This chapter has been divided into two sections namely, **the PC side** and the **SBC side**. Each section gives a detailed description of the program associated with it and also explains the logic behind it. Special emphasis is given on the software development tool, **Paradigm** and the role it plays in the execution of the SBC program. The menu driven PC program is also explained in details.

The chapter titled **Execution of Software** describes the events that take place when both the programs are executed after the system has been switched ON.

Chapter 7 : DESIGN CHOICES

This section has been divided into two segments and it vindicates the choice of hardware components as well as the software methodologies used for the project.

The chapter titled **Software Considerations** justifies the choice of ‘C’ as the programming language in the light of various considerations and standards adopted for the project.

The chapter titled **Hardware Considerations** explains the logic behind the selection of the various components used in the hardware side of the project.

Chapter 8 : TESTING AND DEBUGGING

This chapter provides a brief account of the various methods adopted for testing and validation of the system.

Chapter 9 : THE WORKING SYSTEM

This chapter takes a final look at the system set-up and throws light on the sequence of events that are generated when the system is switched ON.

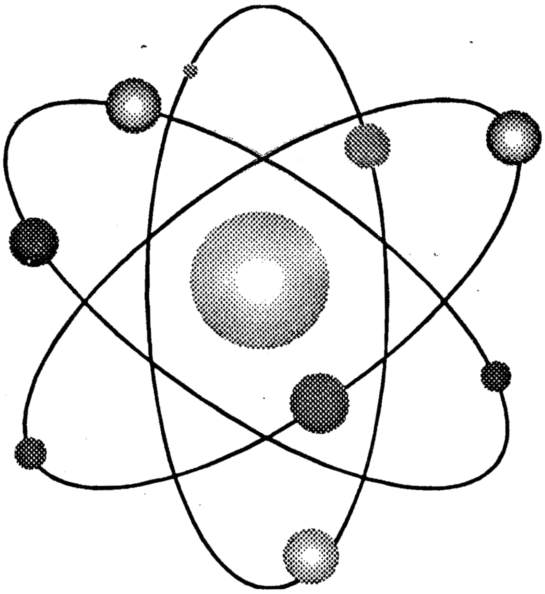
Chapter 10 : IMPORTANCE OF THE PROJECT

In summarising, it may be said that the project marks a beginning in the shape of things to come. The various important things to come out of this project are pointed out in this chapter.

Chapter 11 : LOOKING AHEAD : Provisions for the future

A brief account of the upgradations possible in future are mused over in this chapter .

Apart from this, the system features, an index containing the expanded versions of several key abbreviated terms and an appendix containning useful data are presented at the end.



Parameter Monitoring System for a 10.4m millimetrewave Radio Telescope

Chapter 1 : INTRODUCTION

1.1 Radio astronomy :An Insight

Our knowledge of the universe is based on the observations of the various celestial bodies-stars, pulsars, quasars, supernova - to name a few. With the passage of time, human resources have varied from the naked eye to the optical telescope in this heavenly pursuit of knowledge and the field called astronomy has slowly begun to take shape. As research in this field increased, it became clear that the aperture of the telescope has to be very large compared to the wavelength of light in order to make fine observations. Groundbreaking work was done in this field by a radio engineer at the Bell Telephone Laboratory called Karl Guthe Jansky. In 1932, while studying the direction of thunderstorm static, Jansky was able to detect the origin of a steady Hiss type static, which was previously unaccounted for. He observed that the hiss is due to the radio waves of extra-terrestrial origin, thus laying the foundation for Radio Astronomy. The importance of these results took a while to be recognised but since then Radio Astronomy has rapidly become a major branch of Astronomy.

Radio wavelengths are typically about a million times longer than those in the optical range are. Regions of space that are opaque to light waves because of interstellar dust, are generally transparent to radio waves.

As Radio Astronomy progressed, the need for Radio Telescopes with better resolution came up. This led to several innovations both in electronics and antenna engineering complementing each other. Many pioneering high-tech developments are due to the research done in this field. Spin-offs from this area find applications in various fields including satellite communications, space research, image processing and biomedical sciences.

1.2 The Radio Telescope at Raman Research Institute

A Radio Telescope operates in the radio band of the electromagnetic spectrum thus making it suitable for observing celestial bodies. The main components are

- An antenna with its feed that selectively collects the radio power from a narrow solid angle.

- A low noise receiver that amplifies the received signal over a restricted frequency band, detects, correlates and integrates information and stores the output in digital form.

The Raman Research Institute, Bangalore can stake its claim to being one of the finest research units in the whole country. It boasts of research facilities in several arenas including a fully equipped

Radio Astronomy laboratory complete with a millimetrewave radio telescope. Following are a few important specifications of the above-mentioned telescope:

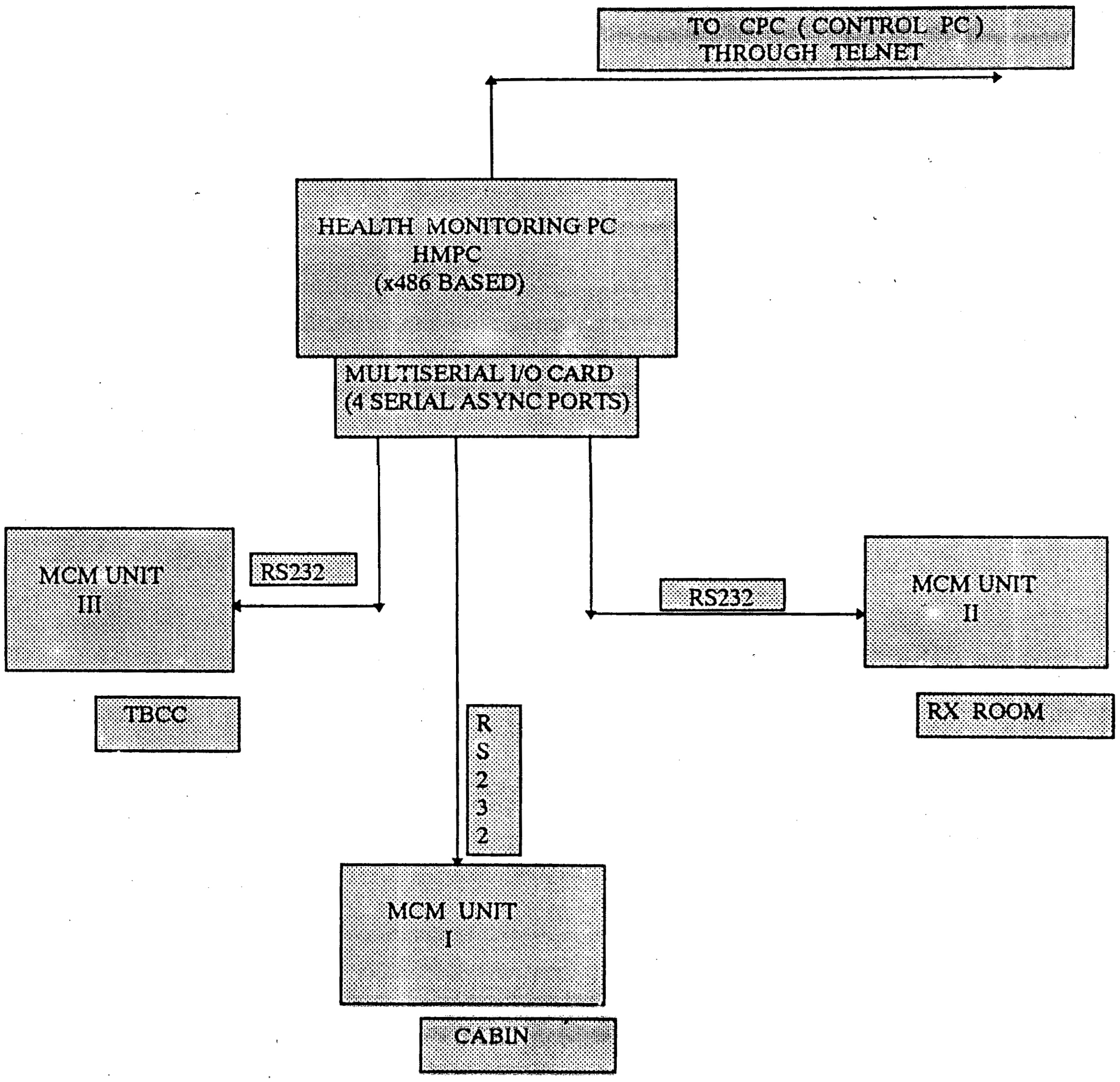
Antenna Type	:Parabolic Reflector
Diameter	:10.4 metres
Beam Width	:80 arc seconds at 80 GHz
Normal frequency range of operations	:22 GHz - 115 GHz

1.3 Requirement for a Monitoring System

The radio telescope mentioned above is quite a complicated system and has to take a lot of parameters into consideration for proper functioning. Each of these parameters is individually controlled and the working of the telescope is contingent to the parameters sticking to a pre-defined set of values. Straying from this critical range by the parameters may prove detrimental to the 'health' of the telescope and hence the need for a monitoring system arises. The monitoring system in question is also required to make provisions such that the parameters may be monitored from three separate control centers in the building, namely the cabin, the Receiver room and the TBCC. However the system monitors the parameters only after they have been converted to their appropriate voltage levels beforehand. The system should prove quite economical in terms of data storage and time.

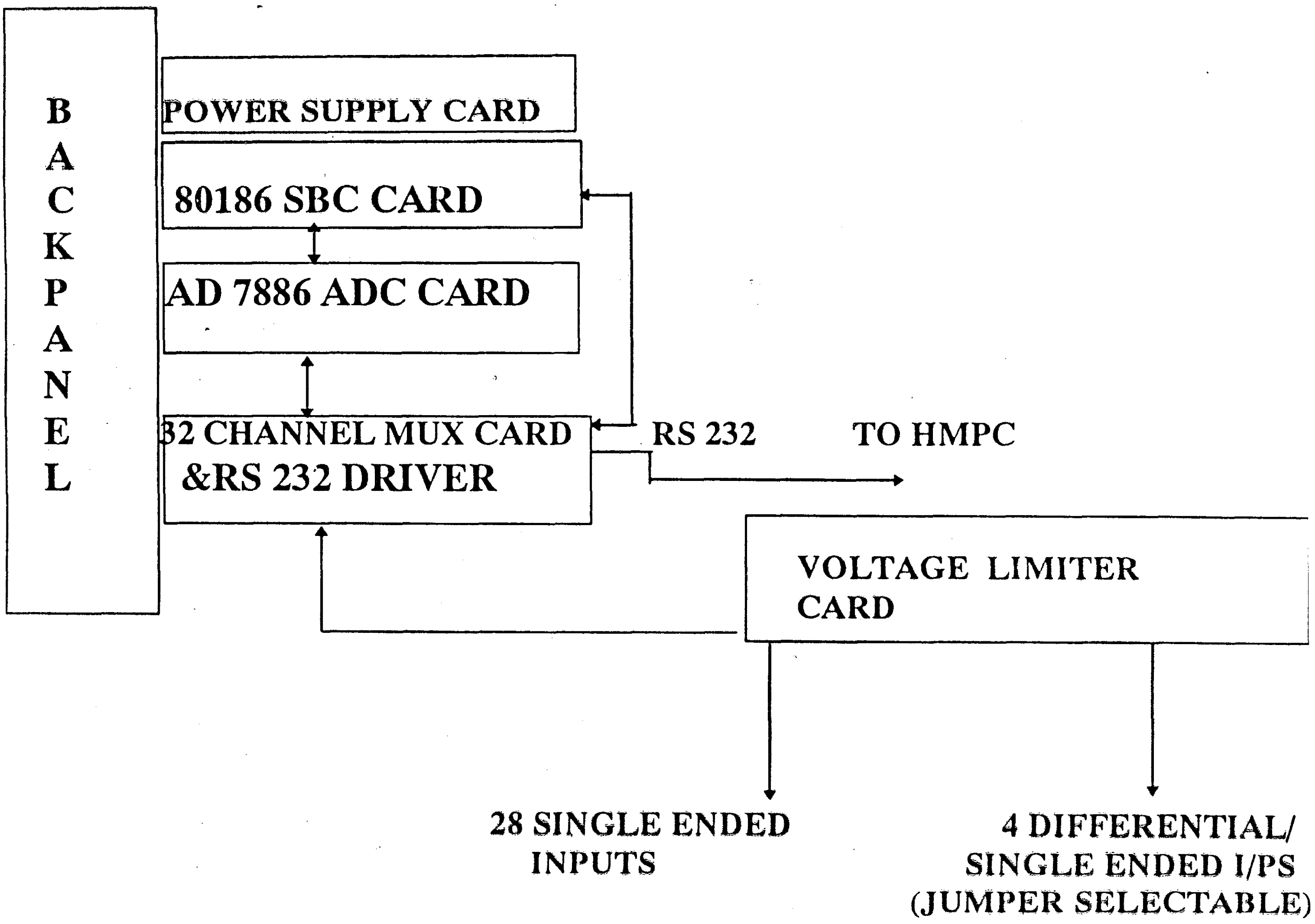
The table below shows a tentative schedule for the parameters involved in the Receiver room. The 'C' next to the parameter indicates that the parameter is critical while 'NC' refers to a non-critical parameter.

1. L-Band IF level	-ch.1	C	1V<2V<4V
2. L-Band IF level	-ch.2	C	1V<2V<4V
3. Baseband (0-400MHz) level	-ch.1	C	1V<2V<4V
4. Baseband (0-400MHz) level	-ch.2	C	1V<2V<4V
5. Phase Lock Indication (2200 MHz)		C	1.75V<2V<2.25V
6. Phase Lock Indication (1600 MHz)		C	1.75V<2V<2.25V
7. Phase Lock Indication (400 MHz)		C	1.75V<2V<2.25V
8. Rubidium Oscillator level (5MHz)		C	1V<2V<4V
9. 120 MHz AOS Video level		NC	1V<4V<8V
10.400 MHz AOS Video level		NC	1V<4V<8V
11.500 MHz AOS Video level		NC	1V<4V<8V
12.DC Power Supply #1	-+ve unreg.	C	24V<20V<18V
13.DC Power Supply #1	-+ve unreg.	C	-24V<-20V<-18V
14.DC Power Supply #2	-+ve unreg.	C	24V<20V<18V
15.DC Power Supply #2	-+ve unreg.	C	-24V<-20V<-18V



BLOCK DIAGRAM OF THE SCHEME

BLOCK SCHEMATIC OF MCM MODULE



Chapter 2 : BRIEF OVERVIEW OF THE PROJECT

2.1 Primary Objectives

The project in question involved development of cost effective time critical monitoring system. Our primary goal was to make best use of available hardware and software resources and integrate them into the working model of a Monitor Control Module. The skeletal outline of such a scheme is shown above.

On the software side, the project involved PC based microprocessor software development. It involved development of real time software for monitoring the critical parameters of the Radio Telescope at RRI. The software was developed keeping in mind the future extensions and provisions were made to install control systems/ add control features.

On the hardware side, the design, development and testing of voltage limiter card as well as the 32-channel MUX card was carried out. The design of both the units was extensively carried out using the circuit design package ORCAD. Replication, testing and debugging of prototype SBC 186 and ADC cards also constituted a major portion of the project as both these units in conjunction form the heart of the Monitor Control Module. Hardware debugging and testing were done with the help of advanced IC testers and logic analysers.

At every developmental stage cost, efficiency and reliability were the main criteria taken into consideration and modular approach was made use of.

Software written on both monitor PC and SBC side was machine independent (written in high level language 'C'). The availability of Paradigm embedded system software and optimised cross-compiler (optimised for speed) made the software development on SBC side more flexible and easy.

The interface standard used was EIA RS232C with Multiserial I/O PCL 232. However the software was made compatible for both RS232C and RS485(multidrop).

Final developmental phase included two things:

1. Freezing the SBC program into(even and odd) Eproms.
2. Screen design in order to provide good user interface.

2.2 Mode of Working

# The SBC units have to be installed at the cabin, TBCC and the RX-room since these are the primary monitoring centres.

# HMPC will send commands to SBC units to set upper and lower limits for various parameters, to specify whether they are critical or non-critical, etc.

# SBC units will acquire from all the 32 channels, continuously, compare the

voltage levels with respective upper and lower limits and set and reset corresponding status bits.

# HMPC will get the status of 32 channels of each SBC in the form of 8 bytes. This will take approximately 4 ms making it a total of 12 ms for the 3 SBC units, and can be synchronised with start and stop pulses.

HMPC checks the status bits and if it encounters any critical parameter being set, sends 'HEALTH NOT OK' code to CPC through serial port. This generates a serial port interrupt to the CPC which will pull the DATA VALID line low. When all the critical parameters are fine, 'HEALTH OK' code will be sent and DATA VALID line becomes active.

Chapter 3 : HMPC

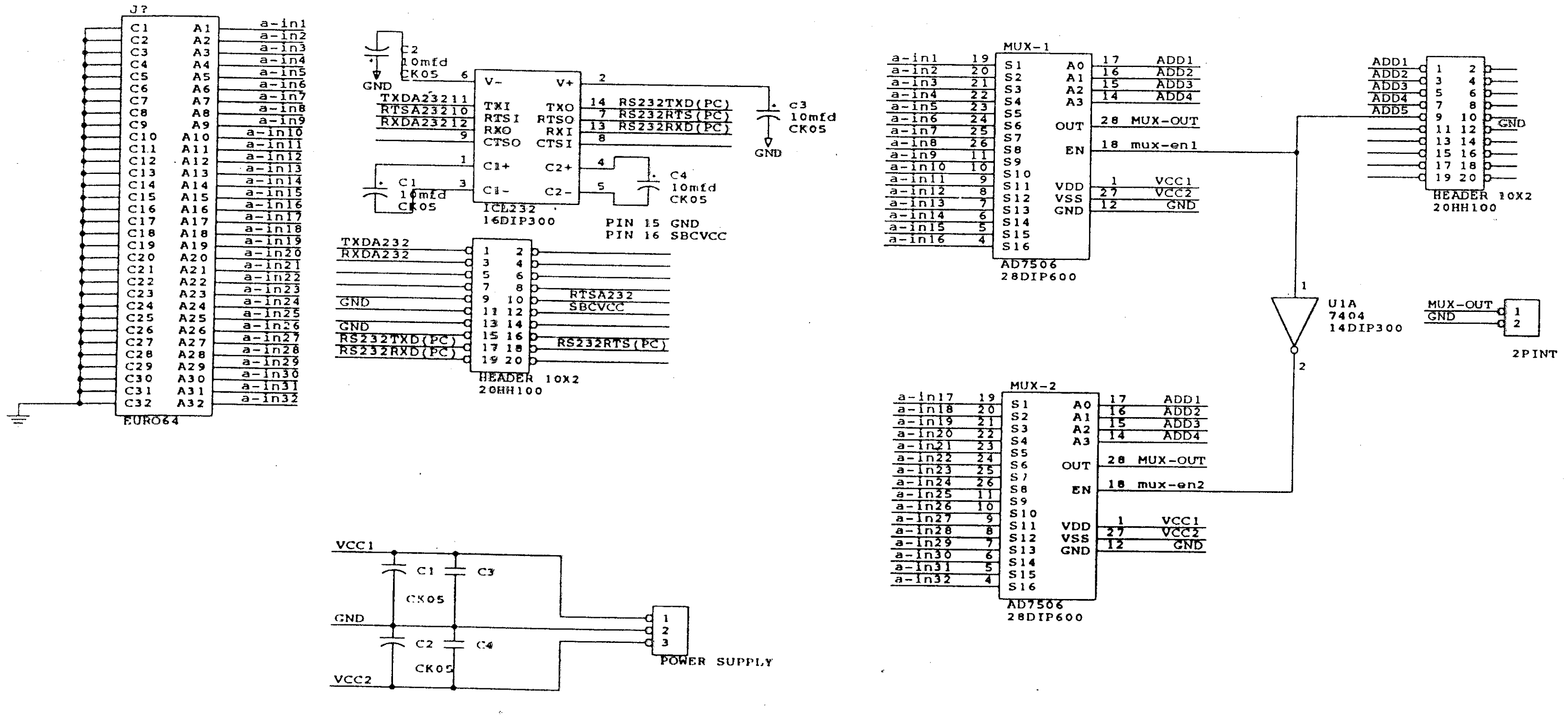
HMPC stands for the Health Monitoring PC. The HMPC is the main control center for the Monitoring System. The multiserial input-output card is the means through which the HMPC communicates with the SBC. The PC software is loaded into the HMPC and execution of this program enables the monitoring system to acquire data for its required purpose. The software was implemented in C as the project demanded a modular approach. The presence of low level features and graphical flexibility also came in handy since the program required bit manipulation and windows design in several places. All the three MCMs are in constant communication with the HMPC and it is the only means by which the user can effect changes in the

Monitoring System. The interactive menu on the HMPC monitor makes the job of implementing changes on the system a very simple affair. The user can, for example, vary the upper and lower limits for the inputs to each and every channel as well as the Scale Factor. The selection of desirable channels (by masking the undesirable ones) also takes place with considerable ease. A detailed description of the Health Monitoring PC program can be found in the Software section under the heading 'The PC Side'.

Chapter 4 : DESIGN & DEVELOPMENT

4.1 The Mux Card

For the project concerned, the design and developmental phase consisted of 32 parameters to be selected by the SBC in order that they may be monitored constantly. This necessitated the design of the card in question. The card comprises of two 16:1 multiplexers (AD7506) along with an inverter to enable only a single MUX at a time. The interface required by the PC to communicate with the SBC has been chosen to be RS232 and to this facility has been incorporated via an ICL 232 chip present on the card. The SBC provides the supply for the ICL 232. The inputs to the MUX from the Voltage Limiter card are routed via a 64-pin Euro Connector.The choice of the card was also influenced by the factor that the AD7506 has an in-built structure to handle differential inputs.



RAMAN RESEARCH INSTITUTE
BANGALORE 560 080

Title	32-CHANNEL MUX CARD	
Size B	Document Number HM/2	REV 1
Date: July 16, 1997	Sheet	of

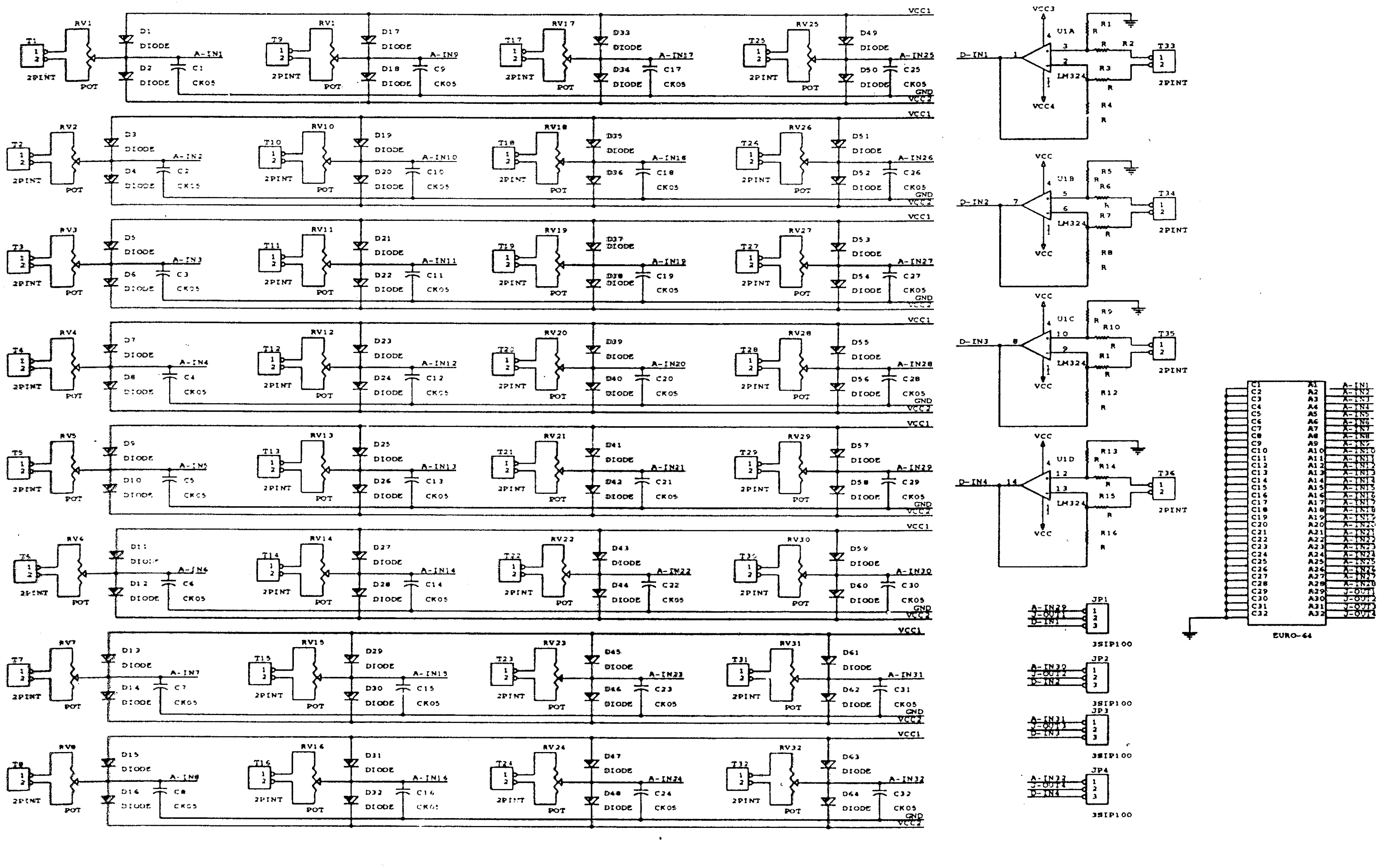
4.2 The Voltage Limiter Card

The ADC chip (AD7886) finds a very important application in this set-up as it executes the crucial job of converting the physical voltage into its digital counterpart. However, the ADC can handle voltages only in the range of +5 to -5 volts. A conditioning system for the ADC was in demand and this brought the concept of a voltage limiter card into the forefront.

The basic structure of the card consists of 32 analog voltage inputs(single-ended) being fed into 32 potentiometer and diode arrangements where the voltage is suitably clamped to remain within a range of +5 to -5 volts. However, the system has a suitable modification to enable it to work in the differential mode of operation also. The differential mode of operation is sometimes preferred to the single-ended mode in order to overcome any pickups in the mains cable and also to obtain a better CMRR. In the final design, 28 inputs were kept single-ended and 4 inputs were modified so as to have options for both single-ended and differential mode of operation.

DESIGN OF LIMITING ACTION

The following potentiometer-diode arrangement is used for carrying out the limiting action as far as the single-ended inputs are concerned. The limiter card ensures the proper working of the ADC by



RAMAN RESEARCH INSTITUTE

BANGALORE 560 080

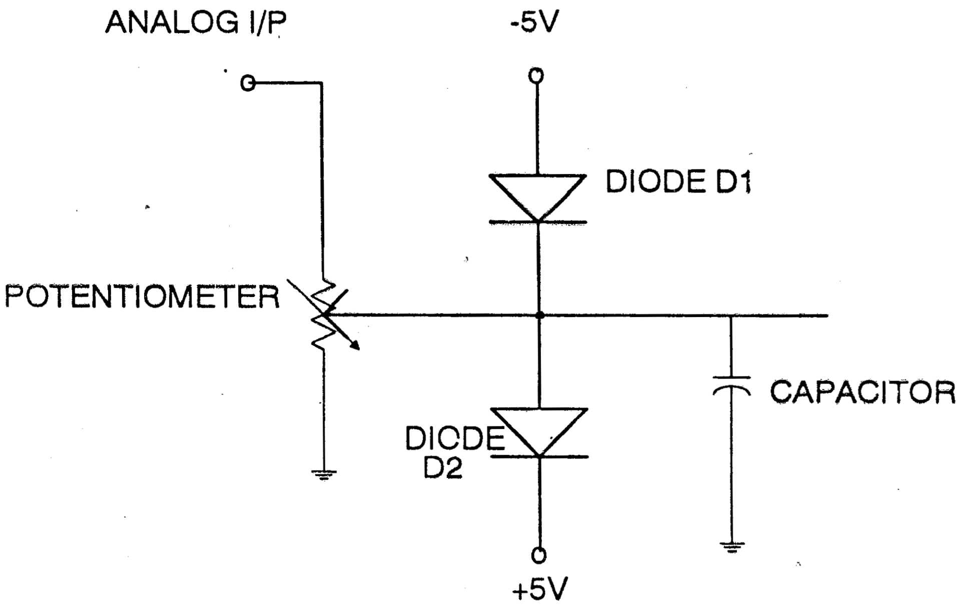
Title

VOLTAGE LIMITER CARD

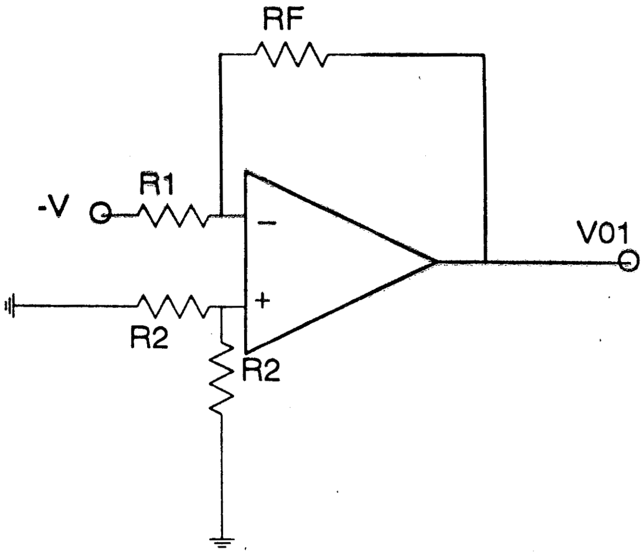
Size	Document Number	REV
C	HM/1	1

Date: July 16, 1997 Sheet of

limiting all the inputs to within +5 to –5 volts. If the analog input is more than +5 volts, the diode D2 is forward biased and the output voltage drops to zero. Similarly, if the voltage is below – 5 volts, the diode D1 is forward biased making the output voltage zero, yet again. The scale factor can be adjusted by varying the resistance on the potentiometer. The limiter card supports 32 such potentiometer-diode arrangements for 32 possible single-ended inputs. The bypass capacitor takes care of noise-related disturbances.



DESIGN OF DIFFERENTIAL TO SINGLE ENDED CONVERTER



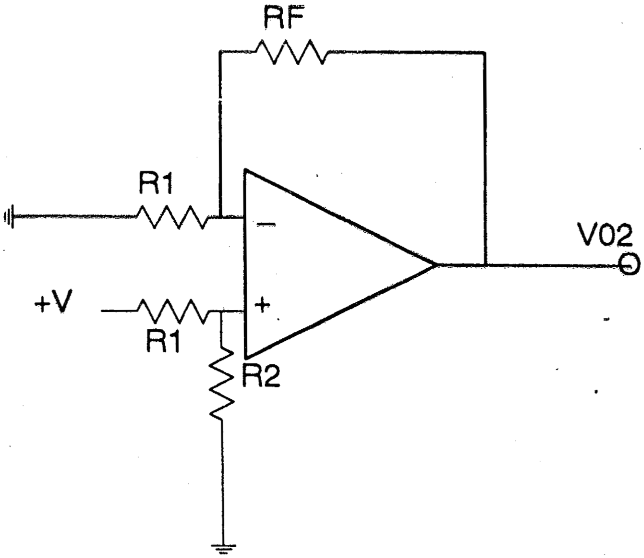
ANALYSIS

(i)

Using Superposition principle,

VO1 = -(-V)RF/R1 = **V RF/R1**

(ii)



$$VO2 = +V\ [R2/(R1+R2)]\ (1+ RF/R1)$$

According to Superposition Principle,

$$VO1 + VO2 = V0$$

$$V\ RF/R1 + V\ [R2/(R1+R2)]\ (1+ RF/R1) = +V$$

$$RF/R1 + [R2/(R1+R2)]\ (1+ RF/R1) = 1$$

$$RF\backslash R1 + [R2/(R1+R2)]\ (RF/R2) = 1 - [R2/(R1+R2)]$$

$$RF/R1 + [R2/(R1+R2)]\ (RF/R2) = R1/(R1+R2)$$

On simplification, it yields

$$RF\ (2R2 + R1) = R1^2$$

Design Choice

1. Choose **$R1 = R2$**

$$RF\ [3R1] = R1^2$$

$$RF = R1/3$$

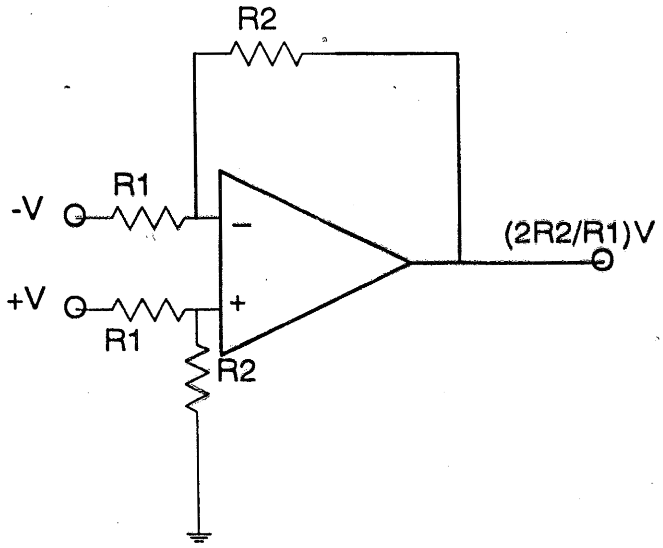
2. $$\mathbf{2\ R2RF + R1RF = R1^2}$$

Put **$R2 = RF = R1/2$**

LHS $= [\ 2\ (R1/2)\ (R1/2)\ ] + R1\ (R1/2)$

$= R1^2/2 + R1^2/2 = R1^2 =$ **RHS**

Design of differential to Single ended Converter (with Limiting Action)



If this circuit is incorporated, it can provide limiting action as well.

For instance, differential inputs -10/+10 volts have to be limited to 2 volts single ended output.

Then choose **R2 = 10K.**

R1 = 100K

Output 2.(10/100).10 v = 2v.

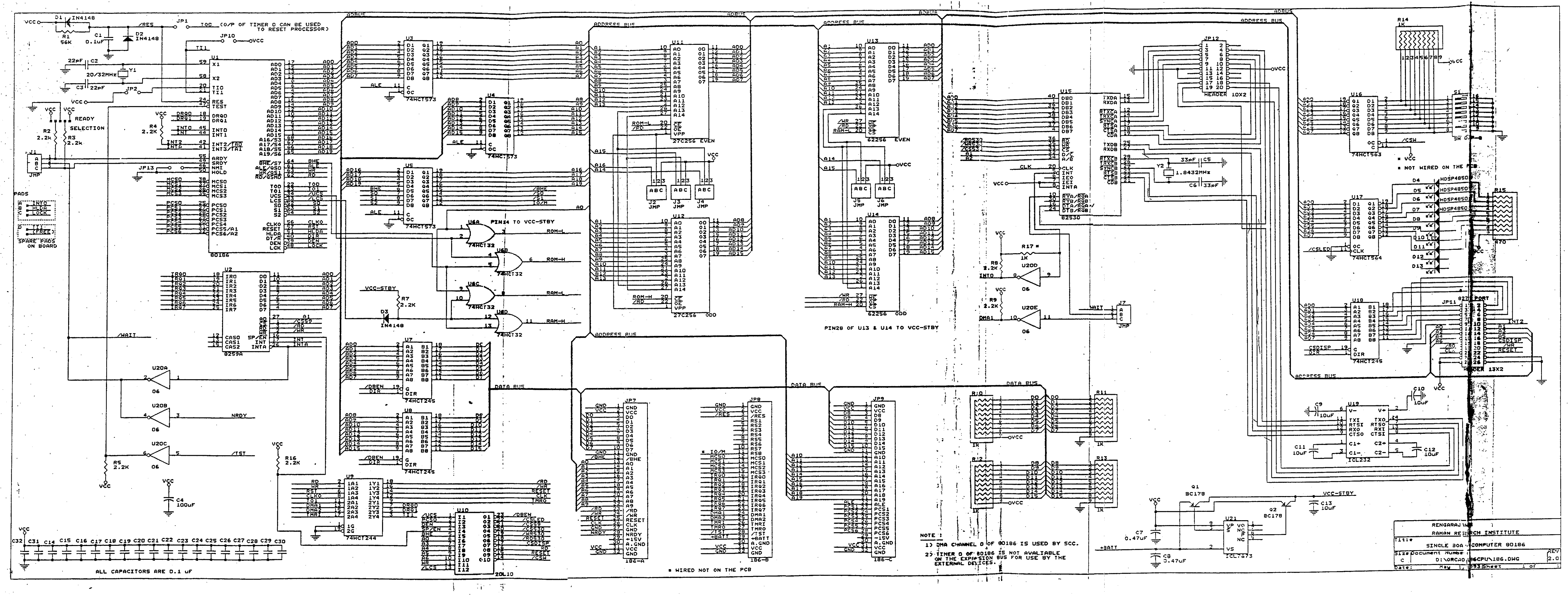
Chapter 5 : HARDWARE IMPLEMENTATION

Components And their Utilities

5.1 The Single Board Computer 80186

The SBC card is a general purpose Single Board Computer intended to be used as an embedded controller in data acquisition and/or control system applications. This card had been previously developed in RRI and had been in existence for sometime.

This single board computer design is based around INTEL's 80C186 microprocessor that runs at 16MHz. This board has two 28 pin sockets for ROM upto 128KB, and two 28 pin sockets for SRAM, upto 64KB. It has an INTEL 8259 Programmable Interrupt Controller(PIC) to increase the total interrupts handling capacity to 16 (External-9 including one NMI & Internal-7). It has a serial Communication Controller, the 8530. This controller has two independent channels that support a whole range of serial communication protocols such as Asynch, Bi-synch, SDLC/HDLC and so on. One of these channels is connected to RS 232C line drivers and it is used as the debug port which is connected to any of the COM port of a standard PC. This card also has an 8-bit DIP switch port and an 8-bit LED port useful for displaying status information. One end of the card has a 96 pin Euro



NOTE :

1) DMA CHANNEL 0 OF 80186 IS USED BY SCC.

2) TIMER 0 OF 80186 IS NOT AVALIABLE ON THE EXPAPSION BUS FOR USE BY THE EXTERNAL DELICES.

RENGARAJAN RAMAN RESEARCH INSTITUTE		
Title SINGLE BOARD COMPUTER 80186		
Size C	Document Number D:\ORCAD\86CPU\186.DWG	REV 2.0
Date: May 1, 1993	Sheet 1 of 1	

connector expansion bus, containing all the necessary signals for interfacing any user specific hardware. The other end of the card has two berg connectors, one with an 8-bit I/O bus mainly to be used with INTEL's 8279 Keyboard/Display Controller to provide the user interface, while the other connector provides the serial communication lines.

The resident firmware initialises the CPU as well as all I/O devices on the board and memory in the system. After the power on initialisation, the DIP switch settings are read and a decision is made to branch either to the Turbo Debugger Remote (TDREM) kernel or a Remote-boot routine.

5.2 The Serial Communications Controller Z8530

The Z8530 is a dual channel, multiprotocol data communications peripherals designed for use with 8- and 16-bit microprocessors. The SCC functions as a serial-to-parallel and parallel-to-serial converter/controller. The SCC can be software-configured to satisfy a wide variety of serial communication applications, including: Bus Architecture (full- and half duplex). Token passing ring (SDLC Loop mode) and Star Configurations(similar to SLAN).

The SCC contains a variety of internal functions including on-chip baud rate generators, digital phase-lock loops and crystal oscillators, which dramatically reduce the need for external logic. In addition, SDLC/HDLC enhancements have been added to the Z85C30 that allow it to be used more effectively in high speed applications.

The SCC handles asynchronous formats, synchronous character-oriented protocols such as IBM Bisync, and Synchronous bit-oriented protocols such as HDLC and IBM SDLC. This versatile device supports virtually any serial data transfer application(telecommunications, cassette, disketted, tape drivers, etc)

The device can generate and check CRC codes in any Synchronous mode. The SCC also has facilities for modem controls in both channels. In applications where these controls are not necessary, the modem controls can be used for general purpose I/O.

With access to 14 Write registers and 7 Read registers per channel, the user can configure the SCC so that it handles all asynchronous formats regardless of data size, number of stop bits or parity requirements. The SCC also accommodates all synchronous formats including characters, byte and bit-oriented protocols.

Within each operating mode, the SCC also allows for protocol variations by handling odd or even parity bits, character insertion or deletion, and many other protocol-dependent features.

Data Communication Modes

Functional Descriptions

The SCC provides two independent full-duplex channels programmable for use in any common asynchronous or synchronous data communication protocols. This includes: Asynchronous, Synchronous MONOSYNC(8-bit sync character). Synchronous BISYNC (16-bit sync character), normal SDLC and SDLC loop modes.

PROTOCOLS

A communication protocol defines a set of rules for the orderly transfer of the information between two communication devices. All communication line protocols in the industry today exchange data in either an asynchronous or synchronous manner. Asynchronous transmission is used in several protocols including the TTY protocol while synchronous transmission is used in protocols like: IBM BISYNC, Synchronous data link control (SDLC), High-level data link control (HDLC) and Advanced data communication control procedures (ADCCP).

For the project undertaken, only the Asynchronous mode of polling is concerned. Hence a brief overview of the Asynchronous Transmission Mode is presented here.

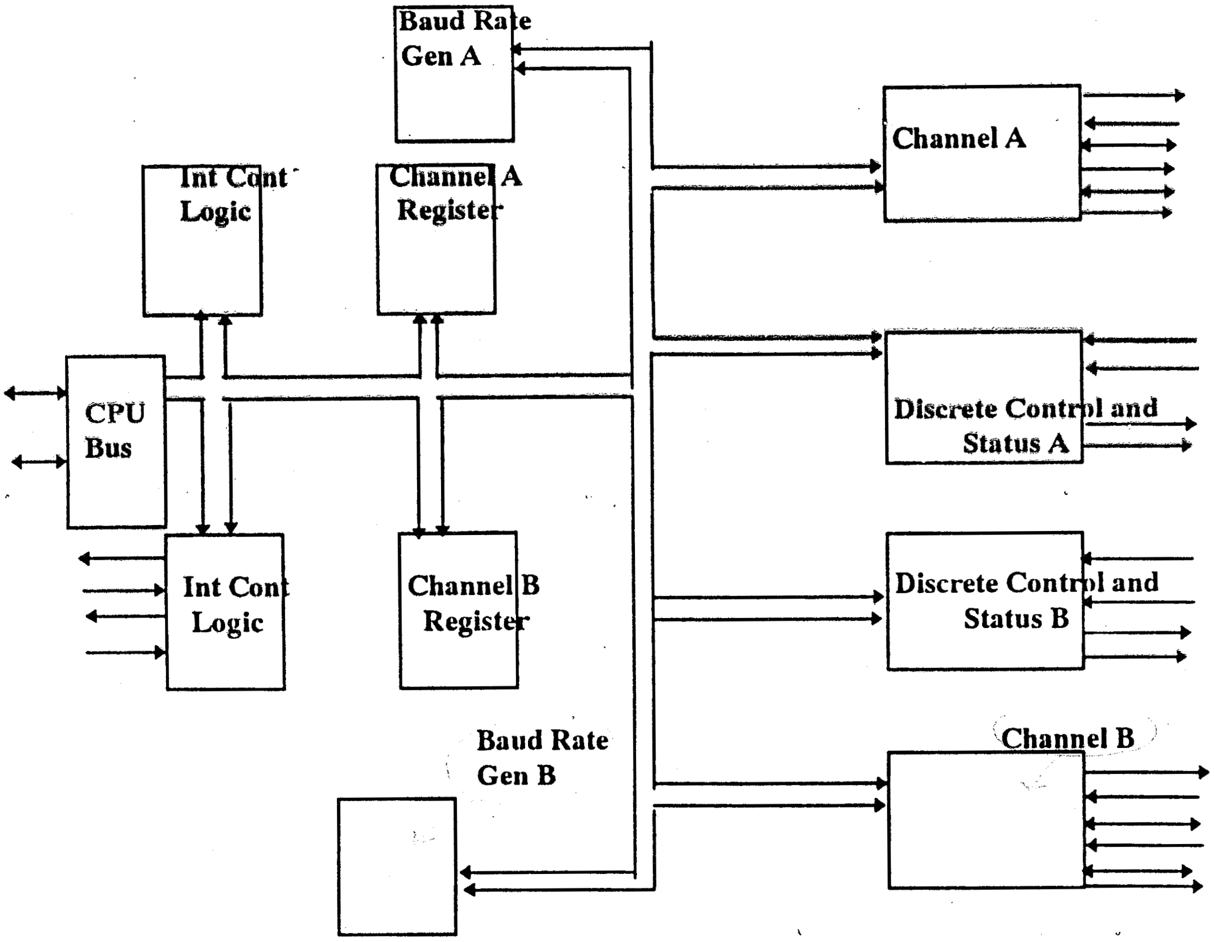
ASYNCHRONOUS TRANSMISSION

In Asynchronous transmission, as the name implies each character is transmitted as an independent entity: that is, the time between the last bit of one character and the first bit of another character can be variable.

Since the receiver must be able to detect the beginning of each character transmitted, this mode requires that at least one bit be added at the start of and end of each character for synchronisation purposes.

Synchronisation at the receiver is accomplished by sensing the transition of the start-bit for each character transmitted. The first data bit of the character is typically sampled one and one-half bit times after the high-to-low transition of the start-bit and each subsequent bit is sampled one bit time thereafter. The sampling of the bit occurs at some multiple of the data rate. Larger multiples allow a closer approximation to the sampling.

Asynchronous communication channels are found in most distributed computer systems for terminal-to-computer communications. The common "serial port" found on personal computers is an asynchronous port. It is used to attach external modems and printers, and to interface the personal computer to a minicomputer for use as a terminal.



Z8530 BLOCK DIAGRAM

5.3 Multiserial I/0 Card

The PC multiserial I/O card (PCL 232) contains 4 standard RS232C interface ports using 8250 Universal Asynchronous Communication Adaptors.The ports are fully programmable to set the start bits, stop bits, parity bit and the baud rate.

MAIN FEATURES.

1.PCL 232 with 4 asynchronous communication ports.

2.Jumper selectable interrupt levels.

3.DIP switch selection for base address.

4.IBM-PC/XT/AT compatibility, with multilink software.

APPLICATION AREAS

1. DATA ACQUISITION.

The multiserial card can be used for data acquisition from devices such as plotters, analysers, etc and even **from other remote devices**.

The monitoring system makes use of this application and Data Acquisition is carried out from the remote device (the MCM unit, in this case).

2. MULTIUSER ENVIRONMENT

3. SERIAL TO PARALLEL / PARALLEL TO SERIAL COMMUNICATION.

Specifications

- Interface system clock: 1.8432 MHz
- Method of communication : Asynchronous
- Programmable Baud Rate : 50 to 9600
- Data bit :5, 6, 7 or 8
- Stop bit : 1,1&1/2 or 2
- Parity bit : even, odd or none.
- RS232 interface chip : 1488,1489.

5.4 The Digitizer Card (ADC)

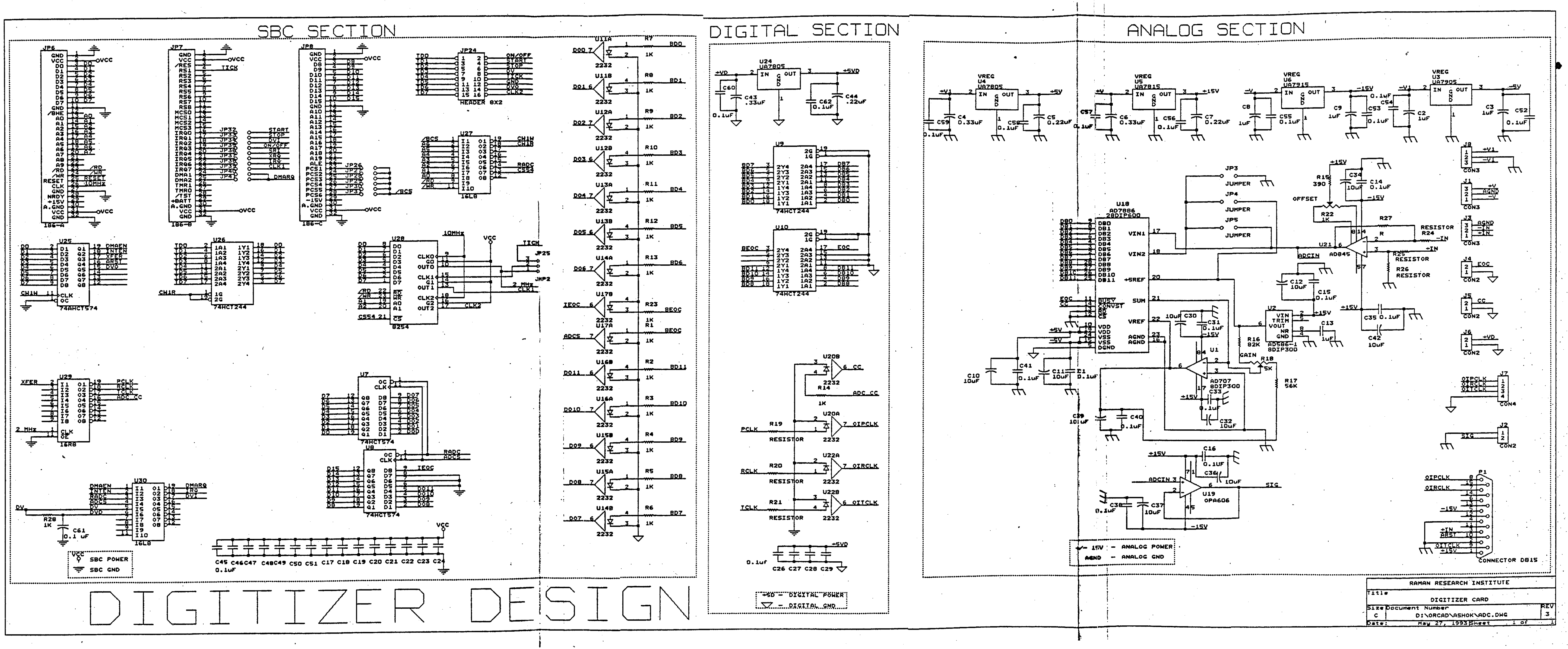
To read data from the AOS and the Filter Banks by the SBC and to send this data over the TELENET to the master PC there was a need for a card which could convert the data from the AOS and the Filter Banks. The main criterion for the DC selected was its conversion speed which had to match the fast data rate from both the AS and Filter Bank with the AOS in particular. The ADC chosen for this purpose was the Analog Devices' AD7886 which has a conversion time of 1.33 micro seconds.

The ground loop problem in the ADC ,where three to four bits of digitised output changes constantly for a single value of the input ,arising due to improper segregation of the digital and analog supplies had to be overcome. Therefore the ADC outputs were optically isolated just before the digitised data is read by the SBC as shown in the schematic for the digitiser. The data from this digitiser card is interfaced to the SBC through the 96 pin Euro connector (JP6, JP7 & JP8). There is also provision for the TELENET signals to be input to this board through the connector JP24.

As another means of overcoming the ground loop problem the entire digitizer card has been divided into 3 sections namely:

- SBC section
- Digital section
- Analog section

The three sections are clearly seen in the schematic of the digitizer and all the three sections have individual supplies and grounds.


SBC SECTION
DIGITAL SECTION
ANALOG SECTION
DIGITIZER DESIGN
JP6
JP7
JP8
JP24
HEADER 8X2
U27
16L8
U25
74AHCT574
U26
74HCT244
U28
8254
U29
16R8
U30
16L8
U7
74HCT574
U8
74HCT574
U24
UA7805
U9
74HCT244
U10
74HCT244
U20B
U20A
U22A
U22B
2232
VREG
U4
UA7805
U5
UA7815
U6
UA7915
U3
UA7905
U18
AD7886
28DIP600
U21
AD845
U2
AD586-1
8DIP300
U1
AD707
8DIP300
U19
OPA606
JP3
JP4
JP5
JUMPER
OFFSET
GAIN
SIG
CONNECTOR DB15
SBC POWER
SBC GND
+VD - DIGITAL POWER
- DIGITAL GND
+-15V - ANALOG POWER
AGND - ANALOG GND
RAMAN RESEARCH INSTITUTE
Title
DIGITIZER CARD
Size
C
Document Number
D:\ORCAD\ASHOK\ADC.DWG
REV
3
Date:
May 27, 1993
Sheet 1 of 1

CIRCUIT DESCRIPTION

Analog section:

The heart of this section is the Analog Devices' AD7886 ADC. The features of this chip are:

- 12 bit resolution
- 1 microsecond conversion time (1.33 micro seconds including S/H time) achieved by using 15 comparators in a 4 bit flash technique.
- 750 KHz throughput rate.
- No missed codes over temperature.
- Low power consumption of 250 mW (typical).
- High speed digital interface with a bus access time of 57 nanosecs (Max).
- On chip clock oscillator provides appropriate timing for the operation ,eliminating the need for any external clocks.
- Operates from +/-5V power supplies.
- Choice of three analog ranges - (0-5) , (0-10) ,& (-5 to +5) V.

Circuit operation:

The +5V reference voltage required for the ADC is derived from a high performance +5V supply source using the Analog Devices' AD 586 (U2) which exhibits excellent stability performance.

The AD7886 also requires a -3.5V reference which must be provided at its Vref input. This voltage is generated by the Analog Devices' AD 707 (U1) with the use of 2 onchip resistors. This external amplifier serves the second function of force/sensing the Vref input , which minimises the error contributions from voltage or IR drops along the internal conductors.

The differential analog voltage to be digitized is fed through the connector J3. The differential to single ended conversion of the analog devices chip AD 845 (U21) . Any offset errors are corrected using the potentiometer R22. The single ended signal ADCIN is made available to the monitoring point J2 , after being conditioned by the opamp OPA 606 (U19). The gain adjustments of the ADC are corrected by the potentiometer R18.

The mode and range of operation of the ADC is determined by the jumpers JP3, JP4 and JP5.

JUMPER TO BE LINKED	RANGE
JP3	(0-10)V
JP4	+/-5V
JP5	(0-5)V

The $\overline{\text{RD}}$ and $\overline{\text{CS}}$ lines are permanently tied low to indicate the ready condition. The conversion is started by the CC signal connected to the $\overline{\text{CONVST}}$ pin of the ADC. The analog input voltage is sampled by the on chip S/H amplifier before being applied to the A/D converter. The transition from track to hold takes place on the falling edge of the CC. The EOC signal, which

is the output of the ADC on the $\overline{\text{BUSY}}$ goes low as soon as the conversion starts and at the same time, the data bus DB0 to DB11 is tristated. When the conversion is over the S/H amplifier goes back to the track mode and the data bus is activated to indicate that new data is ready on the output of the chip. The output from the ADC which are in the digital form is the last section in the analog part of the digitizer.

In addition to the ADC and the opamps, there are four voltage regulators U3 to U6 which provide the necessary voltages for the ADC and the opamps.

Digital section:

The digital section comprises of two buffers (U9 and U10) and the two opto isolators (U20 and U22). The power supply and ground to these IC's in the digital section are provided by a regulator(U24). The data lines and the EOC lines from the ADC are buffered in this section before they drive the LED's of the optocouplers of the SBC section. The two optocouplers are used to isolate the conversion command for the ADC and the filter bank or the AOS clock signals from the SBC section.

SBC section:

The optocoupled EOC signal from the ADC is used to latch data into the latches 74HCT54 (U7 and U8). The RADC signal derived in the chip select logic PAL (U27) is used as an output control signal for these two latches. The SBC then reads the converted data by reading the latches through the 96 pin Euro connector.

Other peripherals:

- Timing generator (8254): Device U28 on the ADC card is a programmable timer used to generate the clock reference for the AOS and the Filter Bank timing signals. Timer 0 is used to generate a 2 MHz clock signal which is used to derive the timing waveforms for the AOS and Filter Banks. Timers 1 and 2 are not used in the design. The clock input to Timer 1 can be a tick signal from the Telenet or the output of Timer 0, while the input of Timer 2 is the CPU clock. The output CLK1 can be used to interrupt the processor while the output CLK2 is made available on the Telenet connector JP24. The chip select for the device is derived from the PAL 16L8 (U27).
- Chip Select Logic: This handled by the PAL 16L8 (U27). This provides the chip selects for the Telenet buffer register(U26), digitizer command register (U25), the 8254 timer (U28) and the ADC data registers U7 and U8. The base address of these chip selects is determined by the /BCS input which is jumper selectable.

Chapter 6 : SOFTWARE ORGANIZATION

6.1 The PC Side

The software written on the PC side provides the only interface by which the user can interact with the monitoring system. This program (written in 'C') takes care of the user interaction as well as the data manipulation and the masking and unmasking of channels. The following events take place when the HMPC program is executed:

- ▷ The program loads data from the MCM.cfg file and downloads them into respective MCMs. The MCM.cfg file is a configuration file that contains the default values of the different parameters pertaining to each channel. This data includes several key parameters such as the mask patterns, upper & lower limits(voltage), scale factor etc. The settings of any channel can be accessed via the MCM.cfg file and they can be altered by suitable user inputs from the main menu. However, on initialization, the default settings would be restored.
- ▷ An interactive menu is displayed on the screen which provides the monitoring personnel with various options (HMPC commands shown below in details). The various options or HMPC commands, as they are called, can be implemented by punching the keys 0 through 9 and then feeding the suitable

text or numeric commands to the called window (if required). Menu design has been carried out with user-friendliness being the foremost criterion and pains have been taken to ensure that the acquired data can be verified through different options available on the menu. The most important command, from the project's point of view, is the one called 'CONTINUOUS ACQUISITION' which provides a continuous commentary on the status and data accumulation of each channel so that they can be reviewed and appropriate action can be taken.

▷ The acquisition data collected by the Monitoring System is regarded very important both in terms of future modifications and research utilities. So, a provision was necessary to record the acquired data so that it may be reviewed from time to time. HMPC program provides an option of writing acquired data in a data.cfg(configuration) file every n minutes. The variable 'n' can be specified by the user by inputs in the menu. However, a default setting is also provided which writes data into the data configuration file every 10 minutes.

6.2 The SBC Side

The software development for the SBC186 card is made extremely easy by exploiting the software development resources available on the PC MS-DOS operating system. System designers can develop the programs for the SBC186 in popular development environments like Borland C++, Turbo C, Microsoft C and Assembler using the PC. The relocatable code in the DOS .EXE file is then converted to absolute located code for the SBC 186 memory map by Paradigm's Locate utility. Debugging the programs for the SBC186 card is also made easy with the Turbo Debugger Remote(TDREM) kernel residing on the card. This communicates with Borland's Turbo Debugger, configured in the remote CPU mode, or the Paradigm Debug/RT running on the PC, through the RS232 serial debug port and the COM port on the PC. This provides source level debugging and the full power of Turbo Debugger to the embedded system program developer.

Software Development for the SBC186

The normal procedure for writing any application specific software for an embedded system would be to write the programs in assembly language and then burning the ROM after assembling, linking and locating the files.

This procedure is tedious, time consuming and needs an in-depth knowledge of the system architecture and assembly language constructs. All these intricacies make the programmer yearn for utilities that make the above processes transparent to him.

PARADIGM Utilities

Paradigm Systems have two utilities called Paradigm Locate and Debug/RT-186EA, which have been developed keeping in mind only the convenience of the programmer. With the above utilities at his disposal, the programmer now has the option of writing his program in high-level languages like:

- Microsoft C
- Borland C++
- Turbo C

The Paradigm utilities have support for the startup code and run-time libraries for these packages, which makes the power of these development environments available to the programmer, with a few restrictions of course! This ensures that the process of developing software for the SBC becomes as simple as writing any other software for the PC, where the code is written in a high-level language and consequently compilation, linking, debugging and execution takes place. This similarity between the development cycles of two different environments leads to greater productivity as the programmer can use his skills learnt on the PC environment for the other and the time spent on learning some other software development tool can be fruitfully utilised in another direction.

ROMming the program

Once the program is completely debugged and the programmer is satisfied with its performance, the code is ready to be fused into ROM.

The program is again compiled after suitable modifications made in the configuration file and 'make' file, with the following options:

- Not running under TDREM.
- Program should originate from the free ROM area(above 0F500H).

A .HEX file of the appropriate format, again, thorough the serial port of the PC.

6.3 Execution of software

The SBC program, at the time of documenting, was being loaded separately into the MCMs and this was eating up a little bit of time. However, this is a temporary situation and can be modified successfully once the program is ROMmed (i.e. it is loaded into the EPROM). Once the Paradigm program is loaded into the SBC and the PC program is executed, the following sequence of events is generated:

MCM SIDE: SBC outputs 0x55 to Serial Port from where it is transferred to the Receiver buffer of PC.

PC SIDE: PC sends the command string

"0X55;

MCM address;

COMMAND CODE;

Number of Bytes ;

Parameters;

0XAA"

MCM SIDE:

- Before scanning each channel, SBC checks whether a 'character' has arrived from PC.
- If YES, it checks whether the byte is **0x55**
- If NO, it promptly returns.
- If YES, it receives next byte and puts it in the 'MCM-ADDR'. It proceeds to check if the second character is its own address (MY_ADDRESS).
- If NO, it returns control.
- If YES, it outputs 'COMMAND INVALID' 0XAA. It receives a number of parameters including the f parameter and the last character. It also checks if the last

character is 'COMMAND_END (0xAA)'.

- If NO, it promptly returns.
- If YES, it interprets the command.
- If the command is successful (valid), it sends 'COMMAND_ACK' or data depending on the data.

PC SIDE:

- PC reads the SP, checks whether the character is 'COMMAND_INVALID' (0xAA).
- If NO, it returns zero (unsuccessful)
- It waits for the next character/data with timeout (depends on the command of character 'COMMAND_ACK'-exclusively for some commands).
- It returns '1' if successful, else it returns 0 (HANDSHAKE_COMMAND).

Chapter 7 : DESIGN CHOICES

7.1 Software considerations

The choice of software is affected by several factors:

- The high level language ‘C’ was the natural choice mainly because of its universal appeal and its modularity. Because a modular approach was the backbone of our project, a software language which supports modularity was the order of the day. ‘C’ came in handy in this regard as it was comparatively easier to enforce changes in the program as and when it was necessary.
- The PARADIGM embedded software system was a turning point in the software as it allowed the programmer plenty of freedom as far as the SBC program was concerned. Gone were the days of hectic assembly language programming when even the simplest of changes demanded a lot of attention and hard work. Instead the programmer keys the assembly language program in his favourite programming language and the PARADIGM takes care of the rest. Since PARADIGM supports a lot of languages like Turbo C, Borland C, Pascal, etc there was a wide variety of standards to choose from. After a lot of consideration, Borland C was given the go-ahead.
- The presence of low level features like bit manipulation also played a pivotal role in the selection of ‘C’ as the working language.

The screen design utility in the PC program required a lot of graphical flexibility and manoeuvrability within the program. To offer a suitable interactive screen design, quite a number of windows had to be called from within the program. 'C' was the perfect foil for this kind of interaction as it provides the user with a lot of graphical options.

7.2 Hardware Considerations

SBC: The choice of the Single Board Computer 186 as the means for data storage and control is governed by the capabilities and performance criteria of the project requirements. The SBC is a microcontroller with an inherent facility to accommodate a resident program in the EPROM which can be executed repeatedly .The program can be made to take into account a number of external parameters for processing. This fit in quite comfortably with the project requirements. The compatibility with a powerful software development tool like PARADIGM also influenced its selection.

ADC: AD7886 was given the nod when the question of an Analog to Digital Converter came up. It is a flash type ADC and has provisions for handling differential inputs. Although the Monitoring System is a comparatively slow speed device, the high speed nature of the AD7886 makes it ideal for improvisations in future.

ICL232: It is the serial communication driver that communicates with the HMPC via the RS232 serial interface.

MUX: An analog multiplexer plays a crucial role in the selection of channels from the address provided. A total of 32 channels has to be accommodated and so the MUX card is designed so that two 16:1 multiplexers in conjunction with an inverter are used to select the required channel. The multiplexer chosen is AD506.

VOLTAGE LIMITER: A limiter card using a diode-potentiometer arrangement keeps the analog voltage input to the ADC from shooting out of range and hence ensures the proper working of the ADC. The inputs are routed via a 64-pin Euro Connector. Due to the presence of 32 inputs with respect to a common ground, 33 pin terminals are required. The situation is resolved by shorting the grounded 32 pins and using the rest 32 for 32 input signals.

Chapter 8 : TESTING AND DEBUGGING

The testing phase began with the whole assembly being set up and the software (both PC and SBC side) running without any glitches. The system was initially tested without the Voltage Limiter card and was found to run smoothly after minor connection errors were rectified. Each channel was individually tested with a direct input voltage in the range of +5 to -5 volts to the ADC. The software took care of displaying the correct voltage but the communication between the MUX and the SBC was less than perfect and selection of channels presented a problem. However, the problem was soon eradicated with the help of Logic Analysers. Next, voltage inputs (+5 to -5 volts) were fed to more than one channel at a time and the system was found to be true again. All the above procedures were carried out for a monitoring system with a single MCM unit.

The Voltage Limiter card was included next in the testing process. Re-thinking had to be done at this stage since the design process had overlooked the provision for the second ground for each channel and 32 external jumper connections had to be implemented to negate the problem.

Chapter 9 : The Working System

The monitoring system conceived for the project has weighed several factors for it's proper utilisation, not the least of which being the user-friendliness of the system, even from a non-technical point of view. Adequate measures were taken to render the end-product free from any complexities as far as the user was concerned.

The final set-up consists of the Health Monitoring PC being connected to the three SBC units being placed at different locations of the observatory (the Cabin, the Receiver Room and the TBCC Room). The communication standard used, as mentioned before, is EIA RS232. The SBC unit along with the ADC card, the MUX card(including ICL 232 driver) and the Voltage Limiter card is housed in a compact box assembly. The power supply unit complete with a cooling fan comprises the other member of the box. Each of these boxes are placed at the three locations mentioned above and are in constant communication with the HMPC.

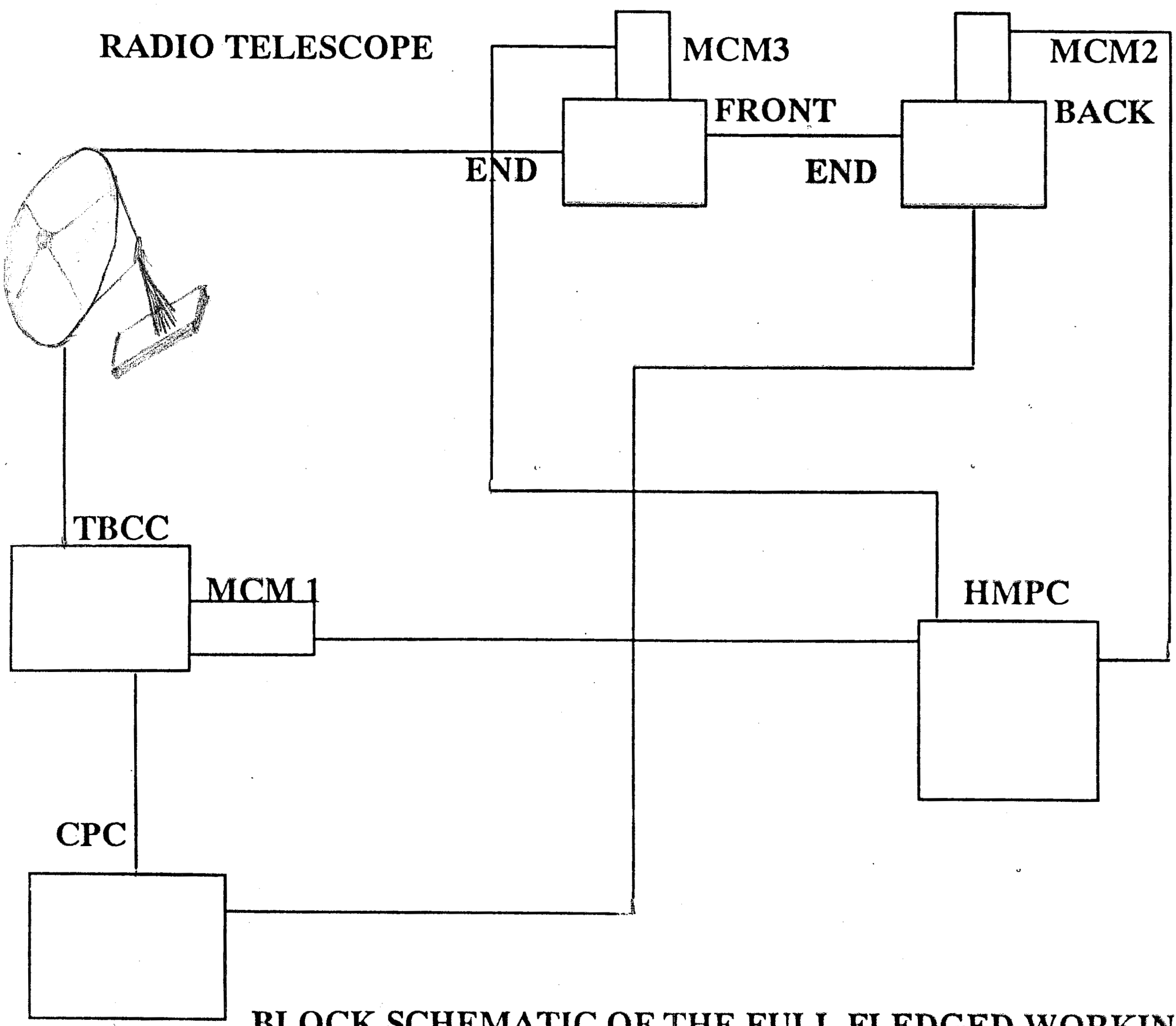
Initially, each of the SBC unit is loaded with the resident SBC program in the EPROM. The SBC program may be summarised briefly as follows:

1. As soon as the system is switched ON, initialization takes place with the MUX, the serial port and the peripherals being initialised.
2. The program takes care of acquiring from the selected channels and

compares the data (voltage) with the upper and lower limits which had been previously defined in the PC program. Accordingly the appropriate status bits are generated.

3. The program waits for the command from the HMPC and depending on the nature of the command, the following sequence is generated:

- If NO, the program goes back and starts executing step 2 again.
- If YES, the program services the command and continues data acquisition.



BLOCK SCHEMATIC OF THE FULL FLEDGED WORKING SYSTEM

Chapter 10 : IMPORTANCE OF THE PROJECT

The project marks an important development in the implementation of additional facilities for the 10.4 m Radio Telescope. The telescope, one of its kind in Asia at the time of its inception , is still a landmark for astronomy enthusiasts all over India. However, this is the first time a step has been taken towards a full-fledged monitoring system for it.

The monitoring system , when fully functional , would take care so that the telescope continues acquiring data when the critical parameters are within limits and stops acquiring when the critical parameters are exceeded. A file which is updated at regular intervals would ensure that the data recorded from the key channels would be available for further investigation. Apart from these , the 'faulty' channels would also be indicated so that the 'reforming' action could be carried out later. In future, the monitoring system would be made to ' talk' directly with the workstation in the control room so that automatic control can be enforced on the proceedings.

Another important footnote in this project has been the utilization of the resources. Almost 90% of the components associated with this project has been in-house developments. Needless to say, there were many a happy face in the institute when the project finally came through. This is especially important in the light of the fact that many of them had made contributions in their own, unique ways, in measures that might seem small, yet might prove crucial in the long run.

The SBC 80186 card, for example , which was developed for a highly specialized purpose, had been an in-house effort and had proved to be a key factor in the success of this project. Barring a PARADIGM or an ORCAD, the entire software had been written by us and provisions have also been made so that they may be upgraded in future with the minimum of hassles.

There are , of course, some limitations which have to be contended with, for the time being. The scanning of channels takes place sequentially and hence limits the user in his quest for monitoring a desired sequence of channels. Thankfully, none of the limitations assume a permanent nature and can be taken care of quite easily in the future. The above problem, for example, can be eradicated with the help of an interrupt driven structure in the software. However the usefulness of the system far outweighs its shortcomings and the system in general marks the beginning of the shape of things to come.

From our point of view it might be said that the project has proved to be an eye opener in terms of exposure to the professional side of engineering. Full marks goes to our college as well as to RRI for providing us with the above mentioned opportunity.

Chapter 11 : LOOKING AHEAD: Provisions for the future

The system concieved for the project responds perfectly to the needs of a specific time critical Monitoring System. However, due to the modular approach taken during the hardware and software design procedures, the system can easily find application as a generic data acquisition system.

Although it may be noted that the ADC used here is of a high-speed nature as compared to the requirements of the system, yet it leaves a lot of options open as far as future implementations are concerned. The high-speed ADC has a resounding effect on the flexibility of the system and it is quite possible that in future, it may be used in the back- end of a spectral line receiver or a total power system.

Because of the processing power of the 80186 microprocessor, control features may be incorporated to the system in future.

The current data acquisition process used by the MCMs is of a sequential nature and hence inhibits the system from performing to the fullest of its capabilities. However, with suitable modifications in software, non-sequential data acquisition by the MCMs is a not-too-distant possibility in future.

SYSTEM FEATURES

- Easy to configure
- Easy to install
- Menu driven and user-friendly
- Adaptability

THROUGHPUT OF THE MCM:

Throughput rate with all the 32 channels unmasked: **4.8 milliseconds**

(Scan time for one channel is 150 microseconds)

Throughput rate with only one channel unmasked: **175 microseconds**

POWER SUPPLY REQUIREMENTS:

SBC CARD : +/- 5V REGULATED.
ADC CARD : +/- 25V UNREGULATAED.
+/- 12V UNREGULATED.
MUX CARD: +/- 15V REGULATED.
VOLTAGE LIMITER CARD: +/- 5V REGULATED
+/- 15V REGULATED

INDEX:

The list below depicts the expanded versions of several key terms whose abbreviated forms have been used frequently throughout the document:

- ADC – Analog to Digital Converter
- ADCCP - Advanced Data Communication Control Process
- ADLC - Asynchronous Data Link Control
- ASCII – American Standard Code for Information Interchange
- CMOS – Complementary Metal Oxide Semiconductor
- CMRR - Common Mode Rejection Ratio
- CPC - Control PC
- CPU – Central Processing Unit
- CRC - Cyclic Redundancy Code
- DMA – Direct Memory Access
- DRAM - Dynamic Random Access Memory
- DPLL - Digital Phase Locked Loop
- EOP - End Of Poll
- EPROM – Electrically Programmable Read Only Memory
- GHz – Giga Hertz
- HDLC- Highlevel Data Link Control
- HMPC – Health Monitoring Personal Computer
- IC – Integrated Circuit
- KB – Kilo Bytes
- LED – Light Emitting Diode

- MCM – Monitor Control Module
- MUX – Multiplexer
- PC – Personal Computer
- RAM – Random Access Memory
- RRI – Raman Research Institute
- RX-ROOM – Receiver Room
- SBC – Single Board Computer
- SCC – Serial Communications Controller
- SDLC - Synchronous Data Link Control
- SRAM - Static Random Access Memory
- TBCC - Telescope Based Control Console
- TDREM - Turbo Debugger REMote

APPENDIX

THE SINGLE BOARD COMPUTER 80186

Overview of SBC Board

The hardware features of SBC186 are as follows:

# 16 MHz, 80C186 CPU, 20 bit address bus (total 1 Mb memory space) and 16 bit wide data bus.

# Two sockets for SRAM. Jumper selectable upto 64KB (2 x 62256) with battery backup circuitry.

# Two sockets for EPROM Jumper selectable upto 128 Kb (2x 2C256) for program code.

# An external Programmable Interrupt Controller(8259) to provide more interrupts which makes the SBC186 board capable of handling a total of 9 external interrupts.

# An 8 position BAR Graph LED port.

# An 8 bit DIP switch.

# Two independent DMA channels.

# Two independent 16 bit Timers. One is reserved for on board use while the other is available on the expansion bus.

# An 8-bit bi-directional port intended for front panel user interface, for

example the 8279 keyboard & display controller and dot matrix ASCII display.

# A serial Communication Controller using 85C350 provides two serial ports, one of them is an RS232 port while the other is user configurable through specific hardware.

# Buffered expansion bus on a 96 pin Euro connector.

The 80C186 Microprocessor

♣ CMOS 8086 microprocessor with 16 bit external data bus, 1MB memory address space and 64KB I/O space. Object code compatible with Intel 8086 family.

♣ Two DMA channels with programmable priority.

♣ Three 16 bit Programmable Timers.

♣ Programmable Interrupt Controllers that can be configured for a variety of operating modes.

♣ Programmable Chip Select Unit for memory and I/O.

♣ Clock generator with external crystal 10MHz, 16MHz and 20MHz clock speeds.

♣ DRAM refresh controller.

THE SCC Z8530

CAPABILITIES

- □ Two independent full duplex channels
- □ Synchronous/Isosynchronous data rates:
  - Upto 1/4 of the PCLK (i.e. 4 Mbits/sec maximum data rate with 16 MHz PCLK Z85C30)
- □ Asynchronous capabilities:
  - Upto 250 Kbits/sec with 16Mhz(x16 mode) PCLK.
  - 5,6,7 or 8-bits per character
  - 1,1-1/2 or 2 stop bits
  - Odd or Even parity
  - x1,16,32 or 64 clock modes
  - Break generation and detection
  - Parity, Overrun and Framing Error detection
- □ Character-oriented synchronous capabilities:
  - Internal or external character synchronous capabilities:
  - or 2 sync characters in separate registers
  - Automatic CRC generation/detection

□ SDLC/HDLC Capabilities:

  - Abort sequence generation and checking
  - Automatic Zero bit insertion and deletion
  - Automatic flag insertion between messages
  - Address field recognition
  - I-Field Residue handling
  - CRC generation/detection
  - SDLC Loop mode with EOP recognition/loop entry and exit

□ Receiver data registers quadruply buffered. Transmitter data register doubly buffered.

□ NRZ, NRZ1 or FM encoding/decoding and Manchester decoding.

□ Baud-rate generator in each channel

□ A DPLL in each channel for clock recovery.

□ Crystal oscillator in each channel.

□ Local Loopbacks in Auto Echo Modes

PARADIGM UTILITIES

Paradigm Locate

Paradigm Locate is a utility that allows any DOS.EXE file to be split up and placed at user specified addresses in the target system's address space. Also, during the installation, a subset of the chosen compiler(s) run time libraries is modified by removing all the functions that are not supportable on the system(due to the functions making use of the BIOS or MS-DOS calls, which are absent on the target).It also provides a user customizable INT 21H emulation package to provide features like memory management, stream I/O and other DOS dependant run-time libraries, to be used on the SBC. The entire sequence of operations has been streamlined by using the MAKE utility. The MAKE utility starts compiling the source code, linking and then produces the final LOCATE'd file ready debugging/burning into ROM. 'MAKE' utility checks the time tags of the source files and compiles/assembles only those which have been changed since the last MAKE-ing and also aborts if any errors are reported while compiling or linking.

- **Development Cycle**

The cycle starts with the programmer keying his code in a file called, say, TEST.C. Then the MAKE utility is invoked with the following command at the DOS prompt:

make -ftest

This instructs MAKE to process the files in the sequence, specified in a special makefile called TEST.MAK. The directives in this file control the compilation, linking and invoking LOCATE. Here LOCATE looks for a special configuration file called TEST.CFG, for the translation process. Some of the important parameters specified in this file are:

I. Output file type & name (ABSFILE): This is used to select the file type and optionally supply a file name for the Absolute output file. This is useful when working with the Debugger & other development tools.

II. Assigning a Physical address to the segments (CLASS): This assigns the specified address in the directive to the first segment in a particular named 'class'.

III.Specifying the type of CPU(CPUTYPE): This informs Paradigm LOCATE of the target system microprocessor, which helps in selecting the selecting the set of permitted peripheral registers in the INITCODE directive.

IV.Directive to copy a class (DUPLICATE): This is used to copy the initialised data from the EPROM to RAM by the startup code.

V. Specifying EPROM programmer file type (HEXFILE): This tells LOCATE to produce a suitable format, .HEX file, for downloading to an EPROM programmer.

VI.Initialisation code(INITCODE): This is used to generate instructions in the correct order for the reset vectors, stack initialisation and other peripheral register initialisation like DRAM refresh, wait-state, etc.

VII.Specify access attributes for the memory regions(MAP): This directive is used to intimate LOCATE of the target system's memory structure , i.e.

regions that are read-only, read-write, reserved, etc. This information helps LOCATE in error-checking like overlapping segments, trying to write into EPROM, etc.

If the source code contains any errors, MAKE reports them and aborts the process. Then the necessary corrections are made and MAKE is invoked till it goes through successfully. If the programmer had chosen to debug it on the target, two other utilities called DEBUG/RT and TDREM are used to download it onto the target system through the serial port on the PC.

Debugging Options

The two utilities Paradigm DEBUG/RT or Turbo debugger allows the user to download the .EXE file produced by LOCATE from the PC onto the SBC, to debug it on the SBC card directly.

The Paradigm DEBUG/RT is a customised version of the Borland's Turbo Debugger for the Single Board Computers based the Intel80186 series of embedded processors. In case the DEBUG/RT is not available, the original Turbo Debugger can also be used for debugging the programs on the SBC. This makes the full power of the Turbo Debugger available to the developers of programs for the SBC186 card. The following resources are available to the programmer:

- Breakpoints - source and instruction level.
- Variable watches - for keeping track of desired variables.
- Inspect windows - for inspecting variables.

- Memory dumps.
- CPU windows - for processor status.
- Source line debugging.
- Logging, etc.

Two special features available on the Debug/RT are:

1. Display of the contents of the 80186 internal registers.
2. Single operations of loading, executing and then exiting to DOS prompt.

The DEBUG/RT interacts with the TDREM kernel residing on the SBC through the serial port to achieve this task. The TDREM (Turbo Debugger REMote) is a customised version of the actual utility, which occupies just 4 K of the 80186 space.

TDREM uses some amount of RAM space. So while debugging, the user program should originate above 0B00H.

THE SBC PROGRAM

```
#include "gendefs.h"
#include "sccdefs.h"
#include "globvars.h"
#define MCM2 1
#ifdef MCM1
#define MYADDRESS   01
#endif
#ifdef MCM2
#define MYADDRESS   02
#endif
#ifdef MCM3
#define MYADDRESS 03
#endif
unsigned int ok=0x55,notok=0xaa;

#define COMMAND_BEGIN        0x55
#define COMMAND_END          0xAA
#define COMMAND_ACK          0x55
#define COMMAND_INVALID      0xAA
#define HANDSHAKE_COMMAND    0x00
#define ANALOG_MASK_COMMAND  0x01
#define ANALOG_LIMIT_COMMAND 0x02
#define SEND_STATUS_COMMAND  0x03
#define SEND_DATA_COMMAND    0x04
#define GET_AMASK_COMMAND    0x05
#define SEND_LIMIT_COMMAND   0x07

/*******************************************************************************
THIS FUNCTION CHECKS WHETHER THE OUTPUT PORT IS FULL.IF NOT, IT WAITS FOR A
CERTAIN AMOUNT OF TIME.
*******************************************************************************/
wait_trn()
{
   int dly=10000;
   while( ! (inportb(SCC_CHNLA_CTRL) & 0x04 ) )
   {
    dly--;
    if(dly==0) break;
   }
}
/*******************************************************************************
THIS FUNCTION CHECKS WHETHER THE INPUT PORT IS FULL.IF NOT, IT WAITS FOR A
CERTAIN AMOUNT OF TIME.
*******************************************************************************/
wait_rec()
{
   int dly=32000;
   while( ! (inportb(SCC_CHNLA_CTRL) & 0x01 ) )
   {
```

```
   dly--;
   if(dly==0) break;
   }
}
/*******************************************************************************
FUNCTION TO TRANSFER A CHARACTER TO THE OUTPUT PORT (TRANSMIT BUFFER).
*******************************************************************************/
void send_char(unsigned char data)
{
 wait_trn();
 outportb(SCC_CHNLA_DATA,data);
}
/*******************************************************************************
THIS PROCEDURE RECEIVES A CHARACTER FROM SCC_CHANNEL_A .THIS PROCEDURE IS
BASIC TO ALL FUNCTIONS SUCH AS RECEIVING PC COMMAND.
*******************************************************************************/
unsigned char rec_char()
{
 wait_rec();
 return(inportb(SCC_CHNLA_DATA));
}
/*******************************************************************************
 THIS PROCEDURE SETS THE SCC REGISTER SPECIFIED BY THE PARAMETER
   'REG_NO' WITH THE VALUE SPECIFIED BY THE PARAMETER 'CONTENTS'
*******************************************************************************/
void writeto_sccreg(unsigned char reg_no, unsigned char contents)
{
 int dtly;
 outportb(CTRL_PORT,reg_no);
 dtly = 0;  /* SCC recovery time */
 outportb(CTRL_PORT,contents);
}
/*******************************************************************************
 THIS PROCEDURE INITIALIZES THE SCC FOR THE ASYNC MODE OF OPERATION
   WITH THE REQUIRED CHARACTERSTISTICS
     READS FROM CONTROL PORT TO RESET INTERNAL REGISTER POINTERS OF SCC TO 0
*******************************************************************************/
void init_scc_chnla()
{
 int i;
 unsigned int entries = sizeof(init_table)/sizeof(unsigned char);
 inportb(CTRL_PORT);
 for (i = 0 ; i < entries ; i+=2)
 writeto_sccreg(init_table[i], init_table[i+1]);
}
/*******************************************************************************/
THIS PROCEDURE CLEARS THE CONTROL PORT OF SCC 8530 CHIP
*******************************************************************************/
void clear_ctrl_port()
{
 ctrl_port_data=0x00;
 outportb(CTRL_REG1, ctrl_port_data);
}
```

```
/************************************************************************
THIS FUNCTION UNMASKS ALL THE 32 CHANNELS. USED FOR  INITIALIZATION PURPOSE
************************************************************************/
void init_limit_tables()
{
 int chnl_no;
 for(chnl_no=0;chnl_no<MAXCHANNEL;chnl_no++) analog_mask[chnl_no]=0;
}
/************************************************************************
THIS PROCEDURE PROVIDES START OF CONVERSION PULSE FOR FLASH TYPE A/D
CONVERTOR
************************************************************************/
void generate_start_conv()
{
 ctrl_port_data = ctrl_port_data | 0x04;
 outportb(CTRL_REG1, ctrl_port_data);
}
/************************************************************************
THIS PROCEDURE PROVIDES END OF CONVERSION PULSE
************************************************************************/
void stop_start_conv()
{
 ctrl_port_data = ctrl_port_data & 0xFB;
 outportb(CTRL_REG1, ctrl_port_data);
}
/************************************************************************
THIS PROCEDURE PROVIDES A DELAY OF 0.96microseconds(approx 1us)
************************************************************************/
void delay_1us()
{
 int ii;
 for(ii=0;ii<1;ii++);
}
/************************************************************************
THIS PROCEDURE PROVIDES A DELAY OF 4*0.96microseconds(approx 4us)
************************************************************************/
void delay_4us()
{
 int ii;
 for(ii=0;ii<4;ii++);
}
/************************************************************************
THIS PROCEDURE PROVIDES A DELAY OF 8*0.96microseconds(approx 8us)
************************************************************************/
void delay_8us()
{
 int ii;
 for(ii=0;ii<8;ii++);
}
```

```
/*******************************************************************************
THIS PROCEDURE OUTPUTS A BIT PATTERN TO THE CONTROL PORT FOR A ENABLING
THE DESIRED MUX AND SELECTING PARTICULAR CHANNEL OF THAT MUX
*******************************************************************************/
void select_mux_channel()
{
 unsigned char ms4bits;
 ms4bits=( (chnl_no&0x3c) <<2 );
 ctrl_port_data=( ms4bits | (chnl_no&0x03) ) ;
 outportb(CTRL_REG1, ctrl_port_data);
 delay_1us();
}
/*******************************************************************************
THIS FUNCTION IS USED FOR DATA ACQUISITION AND DATA ANALYSIS AND
INTEGRATION.
*******************************************************************************/
void scan_parameters()
{
 for(chnl_no=0;chnl_no<MAXCHANNEL;chnl_no++)
   {
   if( (inportb(SCC_CHNLA_CTRL) & 0x01 ) ) rec_pccmd();
   if(analog_mask[chnl_no]==1)
    {
    adc_data[chnl_no]=2048;
    chnl_status[chnl_no]=OK;
    continue;
    }
 select_mux_channel();
 generate_start_conv();
 delay_8us();
 stop_start_conv();
 delay_4us();
 adc_data[chnl_no]=inport(READ_ADC);
 voltage=adc_data[chnl_no];
 if( (voltage>max_limit[chnl_no]) ||(voltage<min_limit[chnl_no]) )
       {
            status=NOTOK;
            chnl_status[chnl_no]=status;
       }
       else
       {
            status=OK;
            chnl_status[chnl_no]=status;
        }
  }
  outportb(LED_PORT,++points);
  scan_count++;
  if(scan_count==1)
     {
     scan_count=0;outportb(LED_PORT,++points);
     }
}
```

```
*******************************************************************************
THIS FUNCTION UPDATES IN STATUS OF ALL THE CHANNELS. IF A CHANNEL GOES OUT
OF RANGE THE STATUS BIT MADE HIGH (SET) OTHERWISE IT IS MADE LOW(RESET)
*******************************************************************************
update_status_array()
{
   int status_data;
   unsigned char bitchk;
   int ii=0,jj=0,chnl_no=0;
   chnl_no=0;
   for(ii=0;ii<8;ii++)
      {
           mcm_status[ii]=0;
           bitchk=0x01;
           for(jj=0;jj<8;jj++)
           {
            status_data=chnl_status[chnl_no];
            if(status_data==NOTOK) mcm_status[ii]= mcm_status[ii] | bitchk;
            bitchk=bitchk<<1;
            chnl_no++;
        }
      }
}

/******************************************************************************
THIS FUNCTION IS CALLED TO UPDATE MASK PATTERN IF THE END USER SENDS A
DESIRED MASKED PATTERN.
******************************************************************************/
update_amask_array()
{
   unsigned char mask_data,bitchk;
   int ii,jj,chnl_no=0;
   for(ii=0;ii<8;ii++)
   {
    mask_data=amask_recd[ii];
    bitchk=0x01;
    for(jj=0;jj<8;jj++)
             {
              if( (mask_data&bitchk)==0x00) analog_mask[chnl_no]=0;
              else analog_mask[chnl_no]=1;
              chnl_no++;
              bitchk=bitchk<<1;
             }
   }
}

******************************************************************************
THIS FUNCTION INTERPRETS AND SERVICES THE COMMAND FROM MONITORING PC
******************************************************************************/
interpret_command()
{
 int i,jjj;
 switch(cmd_code)
```

```
{
    case HANDSHAKE_COMMAND:
           send_char(COMMAND_ACK);
           break;
    case ANALOG_MASK_COMMAND:
           send_char(COMMAND_ACK);
           memcpy(&amask_recd[0],&parameters[0],8);
           update_amask_array();
           break;
    case ANALOG_LIMIT_COMMAND:
           send_char(COMMAND_ACK);
           memcpy(&max_limit[(int)parameters[0]],&parameters[1],2);
           memcpy(&min_limit[(int)parameters[0]],&parameters[3],2);
           memcpy(&critical[(int)parameters[0]],&parameters[5],1);
           break;
    case SEND_STATUS_COMMAND:
           scan_parameters();
           update_status_array();
           send_char(COMMAND_ACK);
           for(i=0;i<32000;i++);
           for(i=0;i<8;i++)
           {
            send_char(mcm_status[i]);
               for(jjj=0;jjj<1000;jjj++);
        }
           break;
    case SEND_DATA_COMMAND:
          send_char(COMMAND_ACK);
          for(i=0;i<32000;i++);
          for(i=0;i<64;i++)
          {
          send_char(adc_data[i]&0x00ff);
          for(jjj=0;jjj<1000;jjj++);
          send_char( ((adc_data[i]&0xff00)>>8) );
          for(jjj=0;jjj<1000;jjj++);
          }
          break;
    case GET_AMASK_COMMAND:
         send_char(COMMAND_ACK);
         for(i=0;i<32000;i++);
         for(i=0;i<8;i++)
          {
             send_char(amask_recd[i]);
             for(jjj=0;jjj<1000;jjj++);
          }
         break;
    case SEND_LIMIT_COMMAND:
         send_char(COMMAND_ACK);
         memcpy(&max_limit[(int)parameters[0]],&parameters[1],2);
         memcpy(&min_limit[(int)parameters[0]],&parameters[3],2);
         memcpy(&chnl_no,(int)parameters[0],1);
         break;
    default:
         send_char(COMMAND_INVALID);
```

```
                break;
    }
 for(i=0;i<10;i++);
}
/******************************************************************************
THIS FUNCTION RECEIVES THE COMMAND FROM MONITORING PC
******************************************************************************/
rec_pccmd()
{
   int i=0,csp=0;
   pcdata=inportb(SCC_CHNLA_DATA);
   outportb(LED_PORT,pcdata);
   if(pcdata!=COMMAND_BEGIN) return;
   cmd_str[csp++]=pcdata;
   pcdata=rec_char();
   outportb(LED_PORT,pcdata);
   mcm_addr=pcdata;
   if(mcm_addr!=MYADDRESS) return;
   outportb(SCC_CHNLA_DATA,COMMAND_INVALID);
   cmd_str[csp++]=pcdata;
   pcdata=rec_char();
   outportb(LED_PORT,pcdata);
   cmd_code=pcdata;
   cmd_str[csp++]=pcdata;
   pcdata=rec_char();
   outportb(LED_PORT,pcdata);
   cmd_str[csp++]=pcdata;
   no_of_parameters=(int)pcdata;
   while(no_of_parameters)
   {
    pcdata=rec_char();
    parameters[i]=pcdata;
    cmd_str[csp++]=pcdata;
    i++;no_of_parameters--;
   }
   pcdata=rec_char();
   outportb(LED_PORT,pcdata);
   if(pcdata!=COMMAND_END) return;
   cmd_str[csp++]=pcdata;
   interpret_command();
}
void main()
{
   int ii;
   for(ii=0;ii<8;ii++) mcm_status[ii]=0x5a;
   for(ii=0;ii<8;ii++) amask_recd[ii]=0x55;
   update_amask_array();
   init_scc_chnla();
   inportb(SCC_CHNLA_DATA);
   outportb(SCC_CHNLA_DATA,COMMAND_ACK);
   clear_ctrl_port();
   init_limit_tables();
   chnl_no=0x0;
   while(1)
 {
 scan_parameters();
 }
}
```

THE PC PROGRAM

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <string.h>
#include <bios.h>
#include <ctype.h>
#include <stdlib.h>
#include <mem.h>
#include <stdlib.h>
#include<time.h>
#include<process.h>
#define TXRDYBIT  0x20
#define RXFULLBIT 0x01
#define MCM_NO 3
#define POSSTEPS 2047.0
#define NEGSTEPS 2048.0

#define COMMAND_BEGIN       0x55
#define COMMAND_END        0xAA
#define COMMAND_ACK        0x55
#define COMMAND_INVALID      0xAA
#define HANDSHAKE_COMMAND    0x00
#define ANALOG_MASK_COMMAND  0x01
#define ANALOG_LIMIT_COMMAND 0x02
#define SEND_STATUS_COMMAND  0x03
#define SEND_DATA_COMMAND    0x04
#define GET_AMASK_COMMAND    0x05
#define SEND_LIMIT_COMMAND   0x07

#define ADCMAX   +5.0
#define ADCMIN   -5.0
#define ADCBITS  12
#define ADCZERO  2048.0

#define SETTINGS 0xE0 | 0x00 | 0x00 | 0x03/* Refer 'bioscom' function of BC*/
void menu();
void display_mask();
void display_data();
void display_status();
void select_mcm_port();
void download_limits_mask();
void read_conv_limits_mask();
void get_mcm_addr();
void comm_init(int);
void display1();
void display2();
```

```
int quit_loop();
void cursor_on();
void cursor_off();
void write_to_file();
float temp;
int x1,x2,y1,y2,k,data_mask[32];
unsigned int com_data_port,com_status_port,scan_rate[32];
unsigned char mcmused[10],critical[32],buff[128],cmd_buff[15];
unsigned char mcm_char,mcm_response,mcm_status[100],mcm_mask[10],data_lsb,data_msb;
unsigned int mcm_data[100];
float
upper[32],lower[32],adcstep,disp_data[32],disp_data1[32],disp_data2[32],scale_factor[MCM_NO][32];
unsigned int ulimit,llimit,in_data;
char mcm1,mcm2,mcm3;
int mcm_addr,d,mcmnos,i,j,dumint;
unsigned char begin[10],end[5],out_cmd[15];
unsigned int ulm[32],llm[32];
unsigned char str1[20],str2[20],str3[20],str4[20],str5[20],str6[20];
unsigned char dums[20],location[20];
unsigned char
amask_final[8],mask_bit[32],mask_bit1[32],mask_bit2[32],status_bit[32],status_bit1[32],status_bit2[32];
char c;
double interval;
unsigned char key,key1;
time_t t1,t2;
FILE *fp;
/*****************************************************************************
THIS FUNCTION SELECTS MCM ADDRESS AND INITIALISES CORRESPONDING SERIAL PORT
FOR RS232 COMMUNICATION
*****************************************************************************/
void select_mcm_port()
{
    switch(mcm_addr)
    {
     case 1 :
            com_data_port= 0x2e8;
            com_status_port= 0x2ed;
            comm_init(4);
            break;
     case 2 :
            com_data_port= 0x3e8;
            com_status_port= 0x3ed;
            comm_init(3);
            break;
     case 3 :
            com_data_port= 0x2f8;
            com_status_port= 0x2fd;
            comm_init(2);
            break;
     default:
            break;
    }
}
/*****************************************************************************
```

```
THIS FUNCTION CHECKS WHETHER IT CAN TRANSMIT TO MCM 0R NOT
IT WAITS FOR A SUITABLE TIME
******************************************************************************/
void wait_tx()
{
   int dly=1000;
   while( ! (inportb(com_status_port) & TXRDYBIT) )
    {
     dly--;
     if(dly==0) break;
    }
}
/******************************************************************************
THIS FUNCTION CHECKS WHETHER IT CAN RECEIVE FROM MCM 0R NOT
******************************************************************************/
void wait_rx_stat()
{
   while( ! (inportb(com_status_port) & RXFULLBIT) );
}
/******************************************************************************
THIS FUNCTION SENDS A CHARACTER TO TRANSMIT BUFFER
******************************************************************************/
void send_char(unsigned char data)
{
 wait_tx();
 outportb(com_data_port,data);
 }
/******************************************************************************
THIS FUNCTION CHECKS WHETHER THE RECIEVER BUFFER IS FULL OR NOT (i.e. WHETHER
A CHARACTER IS RECIEVED AT THE INPUT PORT.IF NOT, ITS WAITS FOR THE SPECIFIED
TIME DURING WHICH IT EXPECTS A CHARACTER FROM ONE OF THE ADDRESSED MCMS.
******************************************************************************/
void wait_rx()
{
   long int dly=32000;
   while( ! (inportb(com_status_port) & RXFULLBIT) )
   {
    dly--;
    if(dly==0) break;
   }
}
/******************************************************************************
THIS FUNCTION RECIEVES A CHARACTER FROM ITS RECIEVE BUFFER
******************************************************************************/
unsigned char rec_char_stat()
{
   wait_rx_stat();
   return(inportb(com_data_port));
}
/******************************************************************************
THIS FUNCTION RECIEVES A CHARACTER FROM THE MCM
******************************************************************************/
unsigned char rec_char()
{
   wait_rx();
   return(inportb(com_data_port));
}
/***************************************************************************
THIS FUNCTION SENDS COMMAND TO MCM
***************************************************************************/
void send_cmd_to_mcm()
{
   int i,nbytes;
   nbytes=(int) cmd_buff[3];
   for(i=0;i<(nbytes+5);i++)
   {
    send_char(cmd_buff[i]);
   }
}
/***************************************************************************
THIS FUNCTION ACQUIRES STATUS AND 2-BYTE VOLTAGE INFORMATION OF ALL THE
CHANNELS FROM THE SELECTED MCM AND CONVERTS IT TO THE NORMAL FORM AND
DISPLAYS
THE VOLTAGES ON THE MONITOR.
***************************************************************************/
void acquire()
{
   unsigned char bitchk,i;
   int ii,jj,nbytes,k;
   if(!(send_status_cmd()))
   {
    cprintf("STATUS_OUT");
    getch();
    quit_loop();
   }
   if(!(send_data_cmd()))
   {
    cprintf("SEND_DATA_OUT");
    getch();
    quit_loop();
   }
   window(4,12,75,23);
   textbackground(CYAN);
   textcolor(BLACK);
   clrscr();
   for(i=0;i<32;i++)
   {
    if(mcm_data[i]>=2048)
    disp_data[i]=((mcm_data[i]-2048)*5.0/2048)*(scale_factor[mcm_addr][i]);
    else
    disp_data[i]=((2048-mcm_data[i])*-5.0/2048)*(scale_factor[mcm_addr][i]);
   }
   for(ii=0;ii<4;ii++)
   {
    bitchk=0x01;
    for(jj=0;jj<8;jj++)
     {
     status_bit[8*ii+jj]=mcm_status[ii]&bitchk;
```

```
  bitchk<<=1;
  }
 }
for(i=0;i<8;i++)
{
 for(j=0;j<4;j++)
  {
  switch(mcm_addr)
    {
     case 1:
            disp_data1[4*i+j]=disp_data[4*i+j];
            status_bit1[4*i+j]=status_bit[4*i+j];
                  break;
     case 2:
            disp_data2[4*i+j]=disp_data[4*i+j];
            status_bit2[4*i+j]=status_bit[4*i+j];
                  break;
     default:break;
    }
   }
  }
 gotoxy(23,2);
 cprintf("------MCM 1------");
 for(i=0;i<8;i++)
  {
  for(j=0;j<4;j++){
  if(mask_bit1[4*i+j]==0x00)
   {
    if(status_bit1[4*i+j]==0)
    {
           gotoxy(8*i+5,j+3);
           cprintf("%d:%.2f",4*i+j,disp_data1[4*i+j]);
    }
    else
    {
           textcolor(RED);
           gotoxy(8*i+5,j+3);
           cprintf("%d:%.2f",4*i+j,disp_data1[4*i+j]);
           textcolor(BLACK);
    }
   }
  else
   {
   gotoxy(8*i+5,j+3);
    cprintf("%d:MASK",4*i+j);
   }
  }
 }
gotoxy(23,7);
cprintf("------MCM 2------");
for(i=0;i<8;i++)
{
 for(j=0;j<4;j++)
  {
    if(mask_bit2[4*i+j]==0x00)
    {
     if(status_bit2[4*i+j]==0)
     {
      gotoxy(8*i+5,j+8);
      cprintf("%d:%.2f",4*i+j,disp_data2[4*i+j]);
     }
     else
     {
          textcolor(RED);
      gotoxy(8*i+5,j+8);
      cprintf("%d:%.2f",4*i+j,disp_data2[4*i+j]);
      textcolor(BLACK);
     }
    }
    else
    {
     gotoxy(8*i+5,j+8);
     cprintf("%d:MASK",4*i+j);
    }
   }
  }
}
/*******************************************************************************
THIS FUNCTION IS USED TO BREAK OUT OF THE LOOP DURING CONTINOUS ACQUISITION
WHENEVER ANY OF THE MCMS STOPS RESPONDING.
*******************************************************************************/
int quit_loop()
{
   window(1,12,80,25);
   textbackground(WHITE);
   clrscr();
   display1();
   exit(0);
   return;
}
/*******************************************************************************
THIS FUNCTION DISPLAYS A MESSAGE WHEN THE MCM IS NOT RESPONDING.
*******************************************************************************/
void display1()
{
  window(1,25,80,25);
  textbackground(WHITE);
  clrscr();
  window(15,23,40,25);
  textbackground(GREEN);
  textcolor(WHITE);
  clrscr();
  gotoxy(2,2);
  cprintf("MCM %d not responding",mcm_addr);
  getch();
}
/*******************************************************************************
THIS FUNCTION DISPLAYS A MESSAGE WHEN THE COMMAND IS SUCCESSFUL.
```

```
******************************************************************************/
void display2()
{
   window(15,23,40,25);
   textbackground(GREEN);
   textcolor(WHITE);
   clrscr();
   gotoxy(2,2);
   cprintf("COMMAND SUCCESSFUL");
   getch();
}
/******************************************************************************
THIS FUNCTION IS USED TO CHECK WHETHER THE LINK BETWEEN THE HMPC AND THE
MCM
IS ACHIEVED AND THE MCM COMMUNICATES PROPERLY.
******************************************************************************/
int send_handshake_cmd()
{
   int i,nbytes;
   cmd_buff[0]=COMMAND_BEGIN;
   cmd_buff[1]=mcm_addr;
   cmd_buff[2]=HANDSHAKE_COMMAND;
   cmd_buff[3]=0x00;
   cmd_buff[4]=COMMAND_END;
   send_cmd_to_mcm();
   for(i=0;i<10000;i++);
   mcm_response=inportb(com_data_port);
   if(mcm_response!=COMMAND_INVALID) return(0);
   mcm_response=rec_char();
   if(mcm_response!=COMMAND_ACK) return(0);
   else
   return(1);
}
/******************************************************************************
THIS FUNCTION SENDS THE MASK PATTERN TO THE SELECTED MCM.
******************************************************************************/
int send_analog_mask()
{
   int i,nbytes;
   cmd_buff[0]=COMMAND_BEGIN;
   cmd_buff[1]=mcm_addr;
   cmd_buff[2]=ANALOG_MASK_COMMAND;
   cmd_buff[3]=0x04;
   memcpy(&cmd_buff[4],&amask_final,4);
   cmd_buff[8]=COMMAND_END;
   send_cmd_to_mcm();
   for(i=0;i<10000;i++);
   mcm_response=inportb(com_data_port);
   if(mcm_response!=COMMAND_INVALID) return(0);
   mcm_response=rec_char();
   if(mcm_response!=COMMAND_ACK) return(0);
   else return(1);
}
/******************************************************************************
```

```
THIS FUNCTION SENDS THE ANALOG LIMITS TO THE SELECTED MCM.
*******************************************************************************/
int send_analog_limits()
{
   int i,nbytes,chnl_no;
   cmd_buff[0]=COMMAND_BEGIN;
   cmd_buff[1]=mcm_addr;
   cmd_buff[2]=ANALOG_LIMIT_COMMAND;
   cmd_buff[3]=0x08;
   cmd_buff[12]=COMMAND_END;
   for(chnl_no=0;chnl_no<32;chnl_no++)
   {
    cmd_buff[4]=(unsigned char) chnl_no;
    memcpy(&cmd_buff[5],&ulm[chnl_no],2);
    memcpy(&cmd_buff[7],&llm[chnl_no],2);
    memcpy(&cmd_buff[9],&critical[chnl_no],1);
    memcpy(&cmd_buff[10],&scan_rate[chnl_no],2);
    send_cmd_to_mcm();
    mcm_response=inportb(com_data_port);
    if(mcm_response!=COMMAND_INVALID) return(0);
    mcm_response=rec_char();
    if(mcm_response!=COMMAND_ACK) return(0);
    else {delay(2);continue;}
   }
   return(1);
}
/*******************************************************************************
THIS FUNCTION REQUESTS THE MCM TO SEND THE STATUS INFORMATION OF THEIR
PARAMETERS.
*******************************************************************************/
int send_status_cmd()
{
   unsigned char bitchk,i;
   int ii,jj,nbytes,k;
   cmd_buff[0]=COMMAND_BEGIN;
   cmd_buff[1]=mcm_addr;
   cmd_buff[2]=SEND_STATUS_COMMAND;
   cmd_buff[3]=0x00;
   cmd_buff[4]=COMMAND_END;
   send_cmd_to_mcm();
   delay(1);
   mcm_response=inportb(com_data_port);
   if(mcm_response!=COMMAND_INVALID) return(0);
   mcm_response=rec_char();
   if(mcm_response!=COMMAND_ACK) { return(0);}
   else
   {
    for(i=0;i<4;i++) mcm_status[i]=rec_char_stat();
   }
   return(1);
}
/*******************************************************************************
THIS FUNCTON IS USED TO SEND THE UPPER LIMIT,LOWER LIMIT AND SCALE FACTOR
FOR A PARTICULAR CHANNEL OF A MCM ONLINE.IT CHECKS THE VALIDITY OF THESE
```

```
PARAMETERS AND CONVERTS INTO A 12-BIT DATA(ACCORDING TO THE FORMULA).
****************************************************************************/
int send_limit_cmd()
{
   int i,nbytes,number;
   float up_lim,low_lim;
   cmd_buff[0]=COMMAND_BEGIN;
   cmd_buff[1]=mcm_addr;
   cmd_buff[2]=SEND_LIMIT_COMMAND;
   cmd_buff[3]=0x05;
   cmd_buff[9]=COMMAND_END;
   window(20,13,65,22);
   textbackground(BLUE);
   textcolor(YELLOW);
   clrscr();
   disable();
   enter_chno:   gotoxy(3,2);
   cprintf("ENTER CHANNEL NUMBER:");
   if(! scanf("%d",&number))
   goto enter_chno;
   enable();
   memcpy(&cmd_buff[4],&number,1);
   gotoxy(3,4);
   cprintf(" ENTER THE UPPER LIMIT:");
   scanf("%f",&up_lim);
   gotoxy(3,6);
   cprintf(" ENTER THE LOWER LIMIT:");
   scanf("%f",&low_lim);
   if(up_lim<low_lim)
   {
    temp=up_lim;
    up_lim=low_lim;
    low_lim=temp;
   }
   enter:  gotoxy(3,8);
   cprintf(" ENTER SCALE FACTOR:");
   scanf("%f",&scale_factor[mcm_addr][number]);
   if(scale_factor[mcm_addr][number]<=0.0)
   goto enter;
   up_lim=up_lim/scale_factor[mcm_addr][number];
   low_lim=low_lim/scale_factor[mcm_addr][number];
   if (up_lim>ADCMAX) up_lim=ADCMAX;
   if (low_lim<ADCMIN) low_lim=ADCMIN;
   if(up_lim >= 0.0) adcstep=POSSTEPS; else adcstep=NEGSTEPS;
   ulimit=(unsigned int) (ADCZERO + ( (adcstep/ADCMAX)*up_lim) );
   if (low_lim >= 0.0) adcstep=POSSTEPS; else adcstep=NEGSTEPS;
   llimit=(unsigned int) ( ADCZERO + ( (adcstep/ADCMAX)*low_lim) );
   memcpy(&cmd_buff[5],&ulimit,2);
   memcpy(&cmd_buff[7],&llimit,2);
   send_cmd_to_mcm();
   delay(1);
   mcm_response=inportb(com_data_port);
   if(mcm_response!=COMMAND_INVALID) return(0);
   mcm_response=rec_char();
   if(mcm_response!=COMMAND_ACK) return(0);
   else return(1);
}
/*****************************************************************************
THIS FUNCTION REQUESTS THE MCM TO SEND THE VOLTAGES OF ALL THE UNMASKED
CHANNELS.IF THE MCM RESPONDS, THE DATA IS RECIEVED AND STORED IN DIFFERENT
MEMORY VARIABLES.
*****************************************************************************/
int send_data_cmd()
{
   int i,nbytes;
   cmd_buff[0]=COMMAND_BEGIN;
   cmd_buff[1]=mcm_addr;
   cmd_buff[2]=SEND_DATA_COMMAND;
   cmd_buff[3]=0x00;
   cmd_buff[4]=COMMAND_END;
   delay(2);
   send_cmd_to_mcm();
   delay(1);
   mcm_response=inportb(com_data_port);
   if(mcm_response!=COMMAND_INVALID) return(0);
   mcm_response=rec_char();
   if(mcm_response!=COMMAND_ACK){ return(0);}
   else
   {
    for(i=0;i<32;i++)
    {
     data_lsb=rec_char_stat();
     data_msb=rec_char_stat();
     mcm_data[i]=(data_msb<<8) | data_lsb;
    }
    return(1);
   }
}
/*****************************************************************************
THIS FUNCTION RECIEVES THE MASK DATA OF THE SELECTED MCM AND DISPLAYS IT ON
THE MONITOR.
*****************************************************************************/
int get_analog_mask()
{
   int i,nbytes;
   cmd_buff[0]=COMMAND_BEGIN;
   cmd_buff[1]=mcm_addr;
   cmd_buff[2]=GET_AMASK_COMMAND;
   cmd_buff[3]=0x0;
   cmd_buff[4]=COMMAND_END;
   send_cmd_to_mcm();
   delay(1);
   mcm_response=inportb(com_data_port);
   if(mcm_response!=COMMAND_INVALID) return(0);
   mcm_response=rec_char();
   if(mcm_response!=COMMAND_ACK) return(0);
   else
   {
```

```
   for(i=0;i<4;i++) mcm_mask[i]=rec_char_stat();
   return(1);
   }
}
/*******************************************************************************
THIS FUNCTION IS USED TO TRANSFER THE ANALOG LIMITS TO THE SELECTED MCM.IF
THE
MCM IS NOT RESPONDING IT FLASHES A MESSAGE ON THE SCREEN.
*******************************************************************************/
void download_limits_mask()
{
  select_mcm_port();
  if(!(send_handshake_cmd())) {
  getch();
  window(10,25,36,25);
  textbackground(WHITE);
  textcolor(RED);
  clrscr();
  cprintf("  MCM %d NOT RESPONDING ",mcm_addr);
  getch();
  window(10,25,36,25);
  textbackground(MAGENTA);
  textcolor(WHITE);
  clrscr();
  return;};
  if(!(send_analog_mask()))
  {
   cprintf("\nMCM %d not responding",mcm_addr);
   return;
  };
  if(!(send_analog_limits()))
  {
  cprintf("\nMCM %d not responding",mcm_addr);
  return;
  };
}
/*******************************************************************************
THIS FUNCTION IS USED TO PROCESS THE DATA OF SELECTED MCMS.THE UPPER LIMIT,
LOWER LIMIT AND SCALE FACTORS ARE STORED IN MEMORY VARIABLES AND ARE
CONVERTED INTO CORRESPONDING 12-BIT DIGITAL VALUE.
*******************************************************************************/
void read_conv_limits_mask()
{
  for(i=0;i<32;i++)
  {
   fgets(buff,81,fp);
   printf("\n%s",buff);/*******/
   sscanf(buff,"%s %d %f %f %s %f
%d",str3,&mask_bit[i],&upper[i],&lower[i],&critical[i],&scale_factor[mcm_addr][i],&scan_rate[i]);
   scale_factor[mcm_addr][i]=(scale_factor[mcm_addr][i]<=0.0)?1.0:scale_factor[mcm_addr][i];
   if (upper[i]>ADCMAX) upper[i]=ADCMAX;
   if (lower[i]<ADCMIN) lower[i]=ADCMIN;
   if(upper[i] >= 0.0) adcstep=POSSTEPS; else adcstep=NEGSTEPS;
   ulimit=(unsigned int) ((ADCZERO + ( (adcstep/ADCMAX)*upper[i])/scale_factor[mcm_addr][i]));
   if (lower[i] >= 0.0) adcstep=POSSTEPS; else adcstep=NEGSTEPS;
   llimit=(unsigned int) (( ADCZERO + ( (adcstep/ADCMAX)*lower[i])/scale_factor[mcm_addr][i]));
   llm[i]=(unsigned int)llimit;
   ulm[i]=(unsigned int)ulimit;
  }
  for(i=0;i<4;i++)
  {
   amask_final[i]=0x00;
   for(j=0;j<8;j++)
   {
    mask_bit[8*i+j]<<=j;
    amask_final[i]|=mask_bit[8*i+j]&0xff;
   }
  }
}
/********************************************************************************
THIS FUNCTION IS USED TO INITIALISE THE RS232 COMMUNICATION PORT.
********************************************************************************/
void comm_init(int comm)
{
   bioscom(0,SETTINGS,comm-1); /** Init RS-232 comm. **/
   /*(comm==1 COM1, comm==2 COM2*/
   inportb(com_data_port);
}
/********************************************************************************
THIS FUNCTION READS THE MCM.CFG DATA FILE AND PROCESSES ON IT.IT CHECKS
WHICH OF THE MCM'S ARE IN USE AND PROCESSES DATA OF THESE MCMS ONLY.
********************************************************************************/
void read_cfg_file()
{
   highvideo();
   window(1,1,80,25);
   textbackground(MAGENTA);
   textcolor(WHITE);
   clrscr();
   lowvideo();
   if ((fp=fopen("mcm.cfg","r"))==NULL)
   {
    cprintf("Error opening mcm.cfg file\n");
    exit(0);
   }
   fgets(buff,81,fp); /*First line*/
   fgets(buff,81,fp); /*First line*/
   fgets(buff,81,fp); /*Second line*/
   while((fgets(buff,81,fp)!=NULL))/*(for 1st MCM)Third line onwards*/
   {
    sscanf(buff,"%s %d %s %s %s",str1,&mcm_addr,dums,dums,location);
    fgets(buff,81,fp); /*Fourth line*/
    sscanf(buff,"%s %s",str2,mcmused);
    fgets(buff,81,fp);  /*Fifth line*/
    fgets(buff,81,fp);  /*Sixth line*/
    sscanf(buff,"%s",begin);
    getch();
    if (strcmp(begin,"BEGIN")!=0)
    {
     cprintf("\nMismatch in configuration file format\n");
     fclose(fp);
     exit(0);
    }
   if(strcmp(mcmused,"NOT_USED")==0)
    {
     for(i=0;i<32;i++) fgets(buff,81,fp);
     fgets(buff,81,fp); /*Sixty fifth line*/
     sscanf(buff,"%s",end);
     if (strcmp(end,"END")!=0)
      {
       cprintf("\nMismatch in configuration file format\n");
       fclose(fp);
       exit(0);
      }
      continue;
    }
   read_conv_limits_mask();
   download_limits_mask();
   fgets(buff,81,fp); /*Sixty fifth line*/
   sscanf(buff,"%s",end);
   if (strcmp(end,"END")!=0)
    {
     cprintf("\nMismatch in configuration file format\n");
     fclose(fp);
     exit(0);
    }
   }
  fclose(fp);
  cprintf("\nFinished reading configuration file");
}
/*************************************************************************************
THIS FUNCTION IS USED TO DISPLAY THE MASK INFORMATION SENT TO THE MCM.
*************************************************************************************/
void display_mask()
{
  window(15,13,65,22);
  textbackground(BLUE);
  textcolor(YELLOW);
  clrscr();
  gotoxy(5,1);
  cprintf(" MCM_MASK ");
  for(i=0;i<4;i++)
   {
    for(j=0;j<8;j++)
     {
      k=0x01;
      k<<=j;
      if(mcm_mask[i]&k)
       {
        mask_bit[8*i+j]=0x01;
        data_mask[8*i+j]=1; }
      else
     {
      mask_bit[8*i+j]=0x00;
      data_mask[8*i+j]=0;
     }
     gotoxy(12*i+2,j+2);
     cprintf("ch_no%d :%d",8*i+j,data_mask[8*i+j]);
     }
    }
}
/*****************************************************************************
THIS FUNCTION IS USED TO DISPLAY THE VOLTAGES OF ALL THE CHANNELS OF THE
SELECTED MCM.
*****************************************************************************/

void display_data()
{
   if(!(get_analog_mask())) quit_loop();
   for(i=0;i<4;i++)
   {
    for(j=0;j<8;j++)
    {
     k=0x01;
     k<<=j;
     if(mcm_mask[i]&k)
     {
      mask_bit[8*i+j]=0x01;
      data_mask[8*i+j]=1;
     }
     else
     {
      mask_bit[8*i+j]=0x00;
      data_mask[8*i+j]=0;
     }
    }
   }
   window(15,14,65,22);
   textbackground(GREEN);
   textcolor(YELLOW);
   clrscr();
   gotoxy(15,12);
   cprintf("MCM DATA: ");
   j=0;
   gotoxy(5,j+17);
   for(i=0;i<32;i++)
   {
    if(mcm_data[i]>=2048)
    disp_data[i]=((mcm_data[i]-2048)*5.0/2048)*(scale_factor[mcm_addr][i]);
    else
    disp_data[i]=((2048-mcm_data[i])*-5.0/2048)*(scale_factor[mcm_addr][i]);
   }
   for(i=0;i<4;i++)
   {
    for(j=0;j<8;j++) {
    gotoxy(11*i+2,j+2);
    if(mask_bit[8*i+j]==0x0)
    cprintf("%d: %f",8*i+j,disp_data[8*i+j]);
    else
    cprintf("%d: MASKED ",8*i+j);
    }
   }
}
/*********************************************************************************
THIS FUNCTION DISPLAYS THE STATUS INFORMATION OF THE SELECTED MCM ON THE
SCREEN.
*********************************************************************************/
void display_status()
{
   unsigned char bitchk,i;
   int ii,jj,nbytes,k;
   window(1,13,80,23);
   textbackground(CYAN);
   textcolor(YELLOW);
   clrscr();
   for(i=0;i<8;i++)
   {
    for(j=0;j<4;j++)
    {
    gotoxy(10*i+1,2*j+3);
    cprintf("CH_NO%d:",4*i+j);
    }
   }
   for(ii=0;ii<4;ii++)
   {
    bitchk=0x01;
    for(jj=0;jj<8;jj++)
    {
     status_bit[8*ii+jj]=mcm_status[ii]&bitchk;
     bitchk<<=1;
    }
   }
   for(i=0;i<8;i++)
   {
    for(j=0;j<4;j++)
    {
     if(status_bit[4*i+j]!=0x00)
     {
      window(10*i+8,15+j*2,10*i+8,15+j*2);
      textbackground(RED);
      clrscr();
     }
    }
   }
}
/*********************************************************************************
THIS FUNCTION IS USED TO GENERATE A MENU SCREEN.
*********************************************************************************/
void menu()
{
```

```
window(1,1,80,25);
clrscr();
textbackground(WHITE);
window(1,1,80,3);
clrscr();
textbackground(BLUE);
textcolor(WHITE);
gotoxy(1,1);
cprintf("                    10.4M mmw Radio Telescope                    ");
textbackground(LIGHTMAGENTA);
textcolor(WHITE);
gotoxy(1,2);
cprintf("              Health Monitoring Initialisation/Setup              ");
window(1,3,80,10);
clrscr();
textbackground(CYAN+BLINK);
textcolor(YELLOW);
clrscr();
gotoxy(5,2);
cprintf("    MCM COMMANDS      ");
gotoxy(5,4);
cprintf("0 -> HANDSHAKE COMMAND  ");
textcolor(YELLOW+WHITE);
gotoxy(5,4);
cprintf("0 ->");
textcolor(YELLOW);
textbackground(CYAN+BLINK);
gotoxy(5,5);
cprintf("1 -> SEND ANALOG MASK   ");
textcolor(YELLOW+WHITE);
gotoxy(5,5);
cprintf("1 ->");
textcolor(YELLOW);
textbackground(CYAN+BLINK);
gotoxy(5,6);
cprintf("2 -> SEND ANALOG LIMITS ");
textcolor(YELLOW+WHITE);
gotoxy(5,6);
cprintf("2 ->");
textcolor(YELLOW);
textbackground(CYAN+BLINK);
gotoxy(5,7);
cprintf("3 -> GET STATUS        ");
textcolor(YELLOW+WHITE);
gotoxy(5,7);
cprintf("3 ->");
textcolor(YELLOW);
textbackground(CYAN+BLINK);
gotoxy(5,8);
cprintf("4 -> GET DATA          ");
textcolor(YELLOW+WHITE);
gotoxy(5,8);
cprintf("4 ->");
textcolor(YELLOW);
```

```
   textbackground(CYAN+BLINK);
   gotoxy(40,2);
   cprintf("    MISC COMMANDS      ");
   gotoxy(40,4);
   cprintf("5 -> INPUT MCM ADDRESS  ");
   textcolor(YELLOW+WHITE);
   gotoxy(40,4);
   cprintf("5 ->");
   textcolor(YELLOW);
   textbackground(CYAN+BLINK);
   gotoxy(40,5);
   cprintf("6 -> GET ANALOG MASK  ");
   textcolor(YELLOW+WHITE);
   gotoxy(40,5);
   cprintf("6 ->");
   textcolor(YELLOW);
   textbackground(CYAN+BLINK);
   gotoxy(40,6);
   cprintf("7 -> INPUT ANALOG LIMITS");
   textcolor(YELLOW+WHITE);
   gotoxy(40,6);
   cprintf("7 ->");
   textcolor(YELLOW);
   textbackground(CYAN+BLINK);
   gotoxy(40,7);
   cprintf("8 -> CONTINOUS ACQUISITION");
   textcolor(YELLOW+WHITE);
   gotoxy(40,7);
   cprintf("8 ->");
   textcolor(YELLOW);
   textbackground(CYAN+BLINK);
   gotoxy(40,8);
   cprintf("9 -> QUIT FROM PROGRAM  ");
   textcolor(YELLOW+WHITE);
   gotoxy(40,8);
   cprintf("9 ->");
   textcolor(YELLOW);
   textbackground(CYAN+BLINK);
   window(1,11,80,25);
   textbackground(WHITE);
   clrscr();
   textbackground(MAGENTA);
   textcolor(WHITE);
   gotoxy(5,15);
   cprintf(" PRESS KEY TO  -->  ENTER COMMAND CODE       ");
}
/*****************************************************************************
THIS FUNCTION IS USED TO CONFIGURE THE PORT FOR THE MCM SELECTED.
*****************************************************************************/
void get_mcm_addr()
{
   window(1,25,80,25);
   textbackground(WHITE);
   clrscr();
```

```
  window(25,18,60,23);
  textbackground(RED);
  textcolor(WHITE);
  clrscr();
  gotoxy(2,2);
  cprintf("Enter MCM address : ");
  scanf("%d",&mcm_addr);
  gotoxy(2,4);
  cprintf("New MCM address is %d",mcm_addr);
  select_mcm_port();
}
/*************************************************************************
THIS FUNCTION CHOOSES APPROPRIATE FUNCTIONS TO BE EXECUTED ACCORDING TO
THE
CHOICE ENTERED.
*************************************************************************/
void exec_command()
{
   int i;
   while(1)
   {
    menu();
    gotoxy(5,16);
    key=getch();
    switch(key)
    {
     case '0':
               if(!(send_handshake_cmd()))
               {
                display1();
               }
               else
               {
                window(17,15,42,19);
                textbackground(GREEN);
                textcolor(WHITE);
                clrscr();
                gotoxy(5,3);
                cprintf("MCM %d is alive!",mcm_addr);
                    getch();
               }
               break;
     case '1':
            window(17,12,65,22);
               textbackground(GREEN);
               textcolor(WHITE);
               clrscr();
               gotoxy(4,1);
               cursor_on();
               cprintf("Enter mask pattern :  0-->unmask 1-->mask ");
            for(i=0;i<4;i++)
               {
                amask_final[i]=0x00;
                    for(j=0;j<8;j++)
               {
               exit_loop: gotoxy(10*i+6,j+3);
                   cprintf("chno%d:",8*i+j);
                   scanf("%c",&data_mask[j]);

while(toascii(((int)data_mask[j]))!=toascii(((int)'1'))&&toascii(((int)data_mask[j]))!=toascii(((int)'0')))
                   goto exit_loop;
                   mask_bit[8*i+j]=data_mask[j]&=0x0001;
          data_mask[j]<<=j;
               amask_finai[i]|=data_mask[j]&0x00ff;
               }
                  gotoxy(11*i+5,11);
              cprintf("CHK:%x",amask_final[i]);
              }
              if(!(send_analog_mask()))
              {
               display1();
              }
              else
              {
               display2();
              }
              cursor_off();
              break;
    case '2':
          if(!(send_analog_limits()))
          {
               display1();
              }
              else
              {
               display2();
              }
              break;
    case '3':
          if(!(send_status_cmd()))
              {
               display1();
              }
              else
              {
                  display_status();
                  display2();
              }
              break;
    case '4':
             if(!(send_data_cmd()))
             {
                  display1();
             }
             else
             {
              display_data();
              display2();
```

```
            }
            break;
case '5':
            get_mcm_addr();
            break;
case '6':
        if(!(get_analog_mask()))
            {
             display1();
            }
            else
            {
             display_mask();
             display2();
            }
            break;
case '7':
            if(!(send_limit_cmd()))
            {
             display1();
            }
            else
            {
             display2();
            }
            break;
case '8':
           window(5,18,45,20);
           textbackground(MAGENTA);
           textcolor(WHITE);
           clrscr();
           gotoxy(2,1);
           cprintf(" SAVE EVERY HOW MANY MINUTES?");
           scanf("%d",&interval);
           interval*=60;
           if(interval<600)
           interval=600;
           gotoxy(2,2);
           cprintf("DEFAULT 10 MINUTES ");
           mcm_addr=1;
           select_mcm_port();
           if(!(get_analog_mask())) quit_loop();
       for(i=0;i<4;i++)
           {
           for(j=0;j<8;j++)
           {
            k=0x01;
            k<<=j;
            if(mcm_mask[i]&k)
               {
                mask_bit1[8*i+j]=0x01;
                data_mask[8*i+j]=1;
            }
                else
```

```
              {
                   mask_bit1[8*i+j]=0x00;
                   data_mask[8*i+j]=0; }
          }
                 }
                 mcm_addr=2;
                 select_mcm_port();
                 if(!(get_analog_mask())) quit_loop();
       for(i=0;i<4;i++)
         {
         for(j=0;j<8;j++)
                  {
          k=0x01;
          k<<=j;
          if(mcm_mask[i]&k)
          {
                   mask_bit2[8*i+j]=0x01;
          data_mask[8*i+j]=1;
                  }
          else
          {
                   mask_bit2[8*i+j]=0x00;
          data_mask[8*i+j]=0;
                  }
         }
                 }
            t1=time(NULL);
            while(1)
            {
                 mcm_addr=1;
                 select_mcm_port();
                 acquire();
                 mcm_addr=2;
                 select_mcm_port();
                 acquire();
                 t2=time(NULL);
                 if(difftime(t2,t1)==interval)
                 {
                  t1=t2;
                  write_to_file();
                 }
                 if(kbhit()) key1=toascii(((int)getch()));
                 if ( key1==toascii(((int)'9')))
                 break;
                }
            break;
case '9':
       window(1,1,80,25);
            textbackground(BLACK);
            textcolor(WHITE);
            clrscr();
            return;
default:
        break;
```

```
    }
   }
}
void main()
{
  clrscr();
  read_cfg_file();
  cursor_off();
  menu();
  exec_command();
  cursor_on();
}
/*******************************************************************************
THIS FUNCTION TURNS THE CURSOR OFF
*******************************************************************************/
void cursor_off()
{
   union REGS rg;
   rg.h.ah =1;
   rg.x.cx = 0x2000;
   int86(0x10,&rg,&rg);
}
/*******************************************************************************
THIS FUNCTION TURNS THE CURSOR ON
*******************************************************************************/
void cursor_on()
{
   union REGS rg;
   rg.h.ah = 1;
   rg.x.cx = 0x0507; // the "underline cursor"
   int86(0x10,&rg,&rg);
}
/*******************************************************************************
THIS FUNCTION WRITES THE MCMS DATA INTO A FILE
*******************************************************************************/
void write_to_file()
{
   int k;
   disable();
   if((fp=fopen("data.cfg","a+"))!=NULL)
   {
    fprintf(fp,"\n%s",ctime(&t1));
    fputs("MCM1:\n",fp);
    for(i=0;i<32;i++)
    {
     k=i/8;
     if(i==k*8)
     fputs("\n",fp);
     fprintf(fp," %d: %.3f  ",i,disp_data1[i]);
    }
    fputs("\nMCM2:\n",fp);
    for(i=0;i<32;i++)
    {
     k=i/8;
     if(i==k*8)
     fputs("\n",fp);
     fprintf(fp,"%d: %.3f  ",i,disp_data2[i]);}
     fclose(fp);
    }
    else
    cprintf("FILERR");
    enable();
}
•
```

HEADER FILES

SCCDEFS.H

```
/* ------------------------ SCC REGISTERS ---------------------------  */
#define REG0   0x00
#define REG1   0x01
#define REG2   0x02
#define REG3   0x03
#define REG4   0x04
#define REG5   0x05
#define REG6   0x06
#define REG7   0x07
#define REG8   0x08
#define REG9   0x09
#define REG10  0x0A
#define REG11  0x0B
#define REG12  0x0C
#define REG13  0x0D
#define REG14  0x0E
#define REG15  0x0F

#define STN_ADR 0x02

#define SCC_CHNLB_CTRL  0x0408
#define SCC_CHNLB_DATA  0x040A
#define SCC_CHNLA_CTRL  0x040C
#define SCC_CHNLA_DATA  0x040E

#define CTRL_PORT     SCC_CHNLA_CTRL
#define DATA_PORT     SCC_CHNLA_DATA

#define CLK_CONTROL    0x05    /* No External Crystal ,
                           Rx Clock  = RTxC pin,
                           Tx Clock  = RTxC pin,
                           TRxC pin is configured as output,
                           TRxC pin  = Tx clock       */

    #define RESET         0x80    /* Channel A Reset           */

    #define BRG_ENABLE     0x00    /* Disable The BR generator
                           (BRG not  used on SBC card) */

    #define STATION_ADDR   STN_ADR /* The Slave address          */

    #define ADDR_SRCH      0x04    /* Address search mode enabled */
```

```
/* -------------------- OTHER CONSTANTS ------------------------------ */

#define VALUE_R9            0x07
                /* No Reset,
                   Vector includes status in bits D3,D2 & D1,
                   MIE disabled ,
                   Disable Lower Chain,
                   No Vector mode                    */

#define SET_R9              VALUE_R9 | RESET
                /* Channel reset                     */

#define VALUE_R1            0x60
                /* Select Receive request on W/REQ pin,
                   Select DMA function,
                   DMA disabled,
           Rx Interrupts disabled
                 · Parity is not Special Condition,
                   Tx & Ext Interrupts Disabled      */

#define SET_R1              VALUE_R1 | RX_INTR
                /* Select Rx interrupt mode          */

#define DMA_ENABLE          SET_R1 | 0x80
                /* Enable the DMA funtion            */

#define VALUE_R14           0x02
                /* No DPLL command,
                   No Local Loopback,
                   No Auto Echo,
                   DTR request function,
                   BR generator source = PCLK,
                   BR generator disabled             */

#define SET_R14             VALUE_R14 | BRG_ENABLE
                /* Enable the BRG (as defined )      */

#define DISABLE_RX          0xD8 | ADDR_SRCH
                 /* Rx 8 bits/char,
                   Auto Enables Off,
                   Enter Hunt Mode,
                   Rx CRC Enable,
                   Rx Disabled                       */

#define ENABLE_RX           DISABLE_RX | 0x01
                /* Enable Rx                         */

#define DISABLE_TX          0x61
                /* DTR pin = high (inactive),
                   Tx 8 bits/char,
                   Do not send  Break ,
                   Tx Disabled,
                   SDLC CRC polynomial used,
    REG5,  /* Point to Register 5              */
    0x60,  /* tx 8bits/char               */

    REG6,  /* Point to register 6              */
    0x00,

    REG7,  /* Point to register 7              */
    0x00,

    REG9,  /* Point to Register 9              */
    0x01,  /*vector include status             */

    REG10, /* Point to Register 10              */
    0x00,

    REG11, /* Point to Register 11              */
    0x56,  /*rxc=txc=brg(baud generator output)  */

    REG12, /* Point to register 12              */
    0x18,  /*to generate 1200 baud @ 4 MHz       */

    REG13, /* Point to register 13              */
    0x00,

    REG14, /* Point to register 14              */
    0x03,  /* BRG source=sys clock,enable BRG     */

    REG15, /* Point to register 15              */
    0x00,  /* All ext status interrupts off      */
};
unsigned int
        ii ;
```

GENDEFS.H

```
/*------------------------------------------------------------------------*/
#define    DIGIT_BASE 0x480
#define    READ_ADC   DIGIT_BASE + 0x06
#define    CTRL_REG1  DIGIT_BASE + 0x00
#define    LED_PORT   0x410
/*------------------------------------------------------------------------*/
#define MAXCHANNEL      32
#define MAXLIMIT        5.0
#define MINLIMIT       -5.0
#define OK           0x55
#define NOTOK           0xAA
```

```
   RTS pin = high (inactive),
                 Tx CRC Enabled                    */

#define ENABLE_TX            DISABLE_TX | 0x08
               /* Enable Tx                        */

#define ASSERT_RTS           ENABLE_TX | 0x02
               /* RTS pin = low (active)                */

#define DEASSERT_RTS           ENABLE_TX
               /* RTS pin = high (inactive)              */

#define ASSERT_DTR            ENABLE_TX | 0x00
               /* DTR = low ( enable the RS-485 line-drivers)  */

#define DEASSERT_DTR           ENABLE_TX | 0x80
               /* DTR = high ( disable the RS-485 line-drivers)*/
```

GLOBVARS.H

```
unsigned char
        chnl_no;
unsigned int
        ctrl_port_data=0x0;
unsigned int
        adc_data[64],
        max_limit[64],
        min_limit[64],
        analog_mask[64],
        chnl_status[64];
unsigned int
        scan_count=0x0,
        points=0x0;
unsigned char init_table[] =
{
/* ---------- Section 1 : Modes & Constants -----------------------   */

   REG9,  /* Point to register 9              */
   0x40,  /* channel reset                  */

   REG4,  /* Point to Register 4              */
   0x04,  /* 1 stop bit, no parity, brg = x1 mode */

   REG2,  /* Point to resister 2              */
   0x20,  /* vector = 20h                   */

   REG3,  /* Point to Register 3              */
   0xC0,  /* rx 8bits/char,no auto enable        */
```

MCM.CFG

```
MCM CONFIGURATION FILE
CH NO=Channel number UL=Upper limit LL=lower limit C?=Critical Parameter?
SR=scanrate SF=scale factor
MCM_ADDRESS: 01 MCM LOCATION: CABIN
USED/NOT_USED: NOTUSED
CH NO. MASK  UL  LL  C?  SF   SR
BEGIN
 00    0  3.0 -4.5 Y  0.0  100
 01    0  2.5 -2.5 N  1.0  10
 02    0  5  0  N  1.0  10
 03    0  2  0  N  1.0  10
 04    0  1  0  N  1.0  10
 05    0  0  0  N  1.0  10
 06    0  5  3  N  1.0  10
 07    0  4  0  N  1.0  10
 08    0  2.3  -3.13  N  1.0  10
 09    0  2  -1  N  1.0  10
 10    0  1  -2  N  1.0  100
 11    0   0  -4  N  1.0  10
 12    0  5  -5  N  1.0  10
 13    0  4  0  N  1.0  10
 14    0  3  0  N  1.0  10
 15    0  2  0  N  1.0  10
 16    0  1  0  Y  1.0  10
 17    0  0  0  N  1.0  10
 18    0  5  0  N  1.0  10
 19    0  4  0  N  1.0  10
 20    0  3  0  N  1.0  100
 21    0  2  0  N  1.0  10
 22    0  1  0  N  1.0  10
 23    0  0  0  N  1.0  10
 24    0  5  1  N  1.0  10
 25    0  6  -6  N  1.0  10
 26    0  7  -7  N  1.0  10
 27    0  8  -8  N  1.0  10
 28    0  0  0  N  1.0  10
 29    0  1.0 -1.0  N  1.0  10
 30    0  2.5 -2.5  N  1.0  10
 31    0  5.0 -5.0  N  1.0  10
 END
 MCM_ADDRESS: 02 MCM LOCATION: RX-ROOM
 USED/NOT_USED: USED
 CH NO. MASK  UL  LL  C?  SF
 BEGIN
  00    0  15.0 -15.0 N  1.0  10
  01    1  2.5 -2.5 N  1.0  10
  02    0  5  0  N  1.0  100
  03    1  2  0  N  1.0  10
  04    0  1  0  N  1.0  10
  05    1  0  0  N  1.0  10
  06    0  5  3  N  1.0  10
```

```
07    1  4  0  N   1.0  10
08    0  3 -3  N   1.0  10
09    1  2 -1  N   1.0  10
10    0  1 -2  N   1.0  10
11    1  0 -4  N   1.0  10
12    0  5 -5  N   1.0  10
13    1  4  0  N   1.0  10
14    0  3  0  N   1.0  10
15    1  2  0  N   1.0  10
16    0  1  0  Y   1.0  10
17    1  0  0  N   1.0  10
18    0  5  0  N   1.0  10
19    1  4  0  N   1.0  10
20    0  3  0  N   1.0  10
21    1  2  0  N   1.0  10
22    0  1  0  N   1.0  10
23    1  0  0  N   1.0  10
24    0  5  1  N   1.0  10
25    1  6  -6 N   1.0  10
26    0  7  -7 N   1.0  10
27    1  8  -8  N  1.0  10
28    0  0  0  N   1.0  10
29    1  1.0 -1.0  N 1.0  10
30    0  2.5 -2.5  N 1.0  10
31    1  5.0 -5.0  N 1.0  10
END
MCM_ADDRESS: 03 MCM LOCATION: TBCC
USED/NOT_USED:NOT USED
CH NO. MASK  UL  LL  C?  SF
BEGIN
00    0  5.0 -5.0 N  1.0   10
01    1  2.5 -2.5 N  1.0   10
02    1  5  0  N  1.0   10
03    1  2  0  N  1.0   10
04    1  1  0  N  1.0   10
05    1  0  0  N  1.0   10
06    1  5  3  N  1.0   10
07    1  4  0  N  1.0   10
08    1  3 -3  N  1.0   10
09    1  2 -1  N  1.0   10
10    1  1 -2  N  1.0   10
11    1  0 -4  N  1.0   10
12    1  5 -5  N  1.0   10
13    1  4  0  N  1.0   10
14    1  3  0  N  1.0   10
15    1  2  0  N  1.0   10
16    1  1  0  Y  1.0   10
17    1  0  0  N  1.0   10
18    1  5  0  N  1.0   10
19    1  4  0  N  1.0   10
20    1  3  0  N  1.0   10
21    1  2  0  N  1.0   10
22    1  1  0  N  1.0   10
23    1  0  0  N  1.0   10
```

```
24    1  5    1  N  1.0   10
25    1  6   -6  N  1.0   10
26    1  7   -7  N  1.0   10
27    1  8   -8   N 1.0   10
28    1  0    0  N  1.0   10
29    1  1.0 -1.0 N 1.0   10
30    1  2.5 -2.5 N 1.0   10
31    1  5.0 -5.0 N 0.0   10
END
```