

PROJECT REPORT
on

**mm-WAVE MIXER CHARACTERISATION
DATA ACQUISITION AND ANALYSIS**

Carried out at
RAMAN RESEARCH INSTITUTE
Bangalore

Under the Guidance of
Mr. P.G. Ananthasubramanian
Engineer, R.R.I.

by
K. SUDHAKAR
Reg. No. 95063

Submitted to the Regional Engineering College in partial fulfilment
For the Award of Degree of
MASTER OF COMPUTER APPLICATIONS

Internal Guide
Mr. S.R. Balasundaram,
Lecturer, Dept. of Math. & Comp. Applications



**Department of Mathematics & Computer Applications,
REGIONAL ENGINEERING COLLEGE
Thiruchirappalli - 620 015**

June - 1998

DEPARTMENT OF MATHEMATICS & COMPUTER APPLICATIONS
REGIONAL ENGINEERING COLLEGE
Thiruchirappalli - 620 015

CERTIFICATE

This is to certify that the project entitled "mm-WAVE CHARACTERISATION - DATA ACQUISITION & ANALYSIS" is done by

K. SUDHAKAR

Class : M.C.A

Roll No. : AG 4174


Semester : VI

Reg No. : 95063

in partial fulfilment of requirements for the award of Master of Computer Applications degree from Regional Engineering College - Thiruchirappalli, during the academic year 1995-98. The project work was carried out at RAMAN RESEARCH INSTITUTE, Bangalore.

Project Co-ordinator

Dr. A.V. Reddy,
Asst. Professor, Dept. of Math. & Comp. Appns,
R.E.C., Thiruchirappalli.

 30.6.98

Head of the Department

Dr. A.K. Banerjee,
Dept. of Math. & Comp. Appns.
R.E.C., Thiruchirappalli.

Internal Examiner

Mr. S.R. Balasundaram,
Lecturer, Dept. of Math. & Comp. Appns,
R.E.C., Thiruchirappalli.

External Examiner

RAMAN RESEARCH INSTITUTE

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



26 June 1998

CERTIFICATE

This is to certify that the project work entitled “mm-WAVE MIXER CHARACTERISATION - DATA ACQUISITION & ANALYSIS” was carried out by K. Sudhakar in Radio Astronomy Laboratory of the Raman Research Institute - Bangalore, during January’98 to June’98 under my guidance for the partial fulfilment of the requirement for the award of Master of Computer Applications of the Regional Engineering College, Thiruchirappalli.

P. G. Ananthasubramanian

P.G. ANANTHASUBRAMANIAN
Engineer, Raman Research Institute,
Bangalore.

Golden Jubilee Year : 1998

ACKNOWLEDGEMENTS

I thank Dr. D.K. Ravindra, Head, Radio Astronomy Lab in Raman Research Institute for providing us the facilities to carry out this project work successfully.

I thank the project guide Mr. P.G. Ananthasubramanian, Engineer, Radio Astronomy Lab, R.R.I., for guiding in all phases of the project.

I thank our Head of the Dept. Dr. A.K. Banerjee, who has helped in getting the project in institutions like R.R.I. and the internal guide Mr. S.R. Balasundaram, Lecturer, Dept.of Math & Comp. Appns., RECT., who has given the correct direction in doing the project successfully from the beginning.

I thank Mr. K.B. Raghavendra Rao, R.R.I. who helped in various activities for the development of the project. I thank Mr. B.S. Girish, R.R.I. who helped in the software design. I also thank Mr. H. Nagaraj, R.R.I. who helped in the PCB development required in the project.

K. SUDHAKAR.

CONTENTS

	Page No.
Chapter 1 : INTRODUCTION	1
Chapter 2 : ANALYSIS	
2.1 Device setup & Function	8
2.2 Measurement process	8
2.3 Data acquisition system - Requirements	13
Chapter 3 : DESIGN	18
3.1 Hardware construction	18
3.2 Software	27
Chapter 4 : EVOLUTION	37
Code	56
Chapter 5 : CONCLUSION	72
Chapter 6 : BIBLIOGRAPHY	73

Chapter 1 INTRODUCTION

Radio Astronomy is the source of study of Astronomical objects and interstellar medium, in the radio frequency range of the Electromagnetic spectrum.

Raman Research Institute has a 10.4 m diameter mm-wave Telescope in the campus. The Front-end is a cryogenic receiver working in the 75-115 GHz. band. A cryogenic mixer is the first element in this receiver, used to down convert the RF (Radio freq.) to IF (Intermediate freq.) frequency of 1.4 GHz.

Mixer :

The 'mixer' is the critical component in RF systems. A mixer converts RF power at one frequency into power at another frequency to make signal processing easier and less expensive. The more fundamental reason for frequency up conversion, is to allow for the practical transmission of audio and other low-frequency information through free space. Since a mixer converts signal from one frequency to another, it is sometimes called a 'frequency converter' but the term frequency converter usually implies a mixer/ amplifier or mixer/oscillator combination. The term mixer more closely describes the mechanism through which frequency conversion occurs.

Since the mixer is usually the first or second device from the RF input, the performance of the mixer is crucial to the overall performance of the system. This Switched Mixer Test System is being used for characterising the W-band millimetre wave (mmw) mixer. The characterisation consists of the DC analysis and RF analysis which include the noise temperature and conversion loss measurements. These are performed to optimise the important mixer parameters such as dynamic range, conversion loss, bandwidth, noise figure, voltage standing wave ratio ... etc.

Data Acquisition system :

Data acquisition systems are real-time systems that are basically developed through systems programming and applied with some interfacing

hardware. It enables the computer to have access with any real world devices.

This Data acquisition system is designed for the Switched Mixer Test System which is a RF - measurement system. All the measurements have been done manually so far in the existing system. By this Automation of Data acquisition the measurements are made and controlled through a personal computer.

It will perform the following tasks :

- ◇ Control and set some parameters in SMTS
- ◇ Acquiring required analog signals from SMTS to PC
- ◇ Calculating derived measurements from data
- ◇ Storing the data in format

The block diagram of this Data acquisition system is shown in the figure. The automation is incorporated as much as possible. This project includes the construction of interfacing hardware and the software for the data acquisition.

1.1 The Switched Mixer Test System :

The SMTS is a system, contains radio-frequency devices, components and DC amplifiers inside. This is used for characterising the mixers. Characterising the mixer has two parts. One is the DC part where from the I-V data one can derive the series resistance of the diode as well as the ideality factor. Ideality factor determines the closeness of the diode to an exponential I-V profile. i.e.) how close it is to 1.

By providing two different input levels to the mixer and making power/ detected measurements at the IF output the conversion loss and the Noise figure or Noise temperature of the mixer can be determined. A SMTS is already developed to measure the output of the mixer Noise temperature.

In this project the effort put in is to acquire the data through a PC with an appropriate Interface module and analyse the data to determine these characteristics of the mixer.

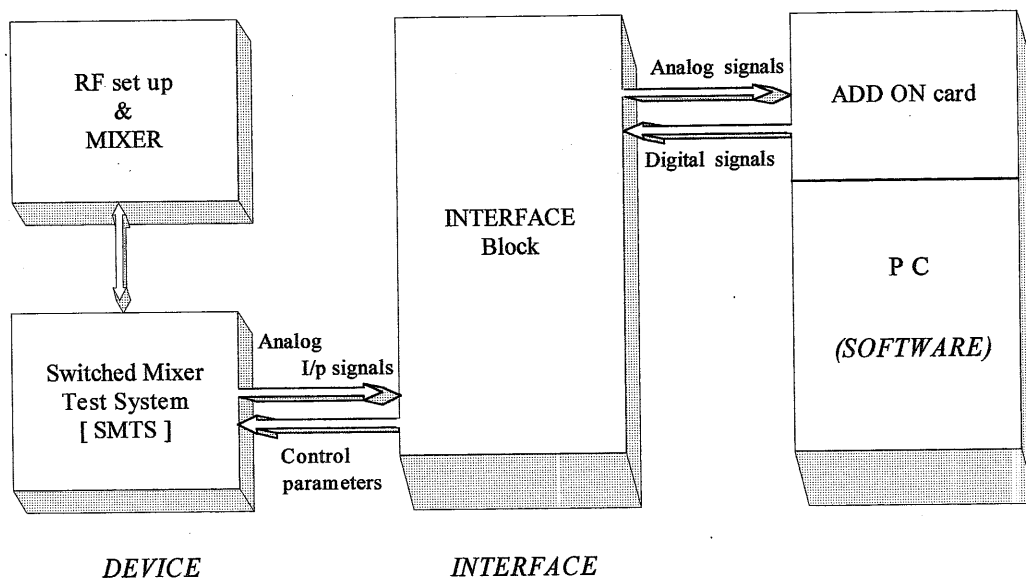


Fig. : Data acquisition system level-1 Block diagram

1.2 Interfacing hardware :

The data acquisition system primarily incorporates the three major modules Analog/Digital converter, Digital/Analog converter and Digital Input/Output buffers.

I. Analog-to-Digital converter [ADC] :

For the real world devices the analog data is acquired in digital form for the following destinations. Storage, Processing, Transmission, Display. Digital data may be stored in either raw or processed form; it may be retained for short, medium, or long periods. It may be transmitted over long distances (for example to or from outer space) or short distances (from one part to another of a micro processor-based instrument).

Processing can run from simple comparisons to complicated mathematical manipulations. One might use it for such purposes as collecting information, converting data to a useful form, using the data for controlling a physical process, performing repeated calculations to extract signals dispersed in noise, generating information for displays, simplifying many of the manually operated jobs. It all starts with getting the data in digital form, as rapidly, as frequently, as accurately, as completely and as cheaply as necessary.

The basic instrumentality for accomplishing this is the analog-to-digital converter (ADC). To accommodate the input voltage to the specified conversion relationship, some form of scaling and offsetting (signal conditioning) may be necessary, performed with an amplifier/ attenuator. To convert analog information from more than one source, either additional converters or a multiplexer may be necessary. To increase the rate at which information may be accurately converted, a sample-hold or software polled operations may be desirable. To compress an extra-wide analog dynamic range, a logarithmic amplifier or conversion relationship may be found useful.

Key factors :-

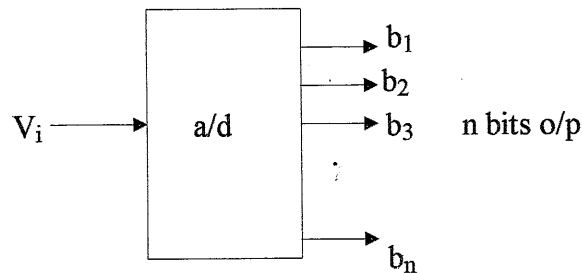
Apart from the environmental factors, the choice of configuration and circuit building blocks in data acquisition depends on a number of critical considerations, among them .

- Resolution and accuracy
- Number of analog channels to be monitored

- Sampling rate per channel
- Throughput rate
- Signal-conditioning requirements
- Intended disposition of converted data
- The cost function.

Measurement :-

The ADC will have fixed number of output lines for measuring the input voltages. The transfer function for the a/d conversion is as follows.



$$V_i = V_{fs} \sum_{i=1}^n (b_i / 2^i) + V_{off}$$

where V_{fs} is the reference voltage or the full scale range,
 V_{off} is the offset-voltage of the input voltage range
 and b_1, b_2, \dots, b_n are the corresponding output bits.

The least input voltage will produce the output as all bits 000...0 and the full scale voltage will produce the output of all high as 111...1. Each bit incrementation corresponds to the input voltage incrementation by the resolution amount.

II. Digital-to-Analog converter [DAC] :

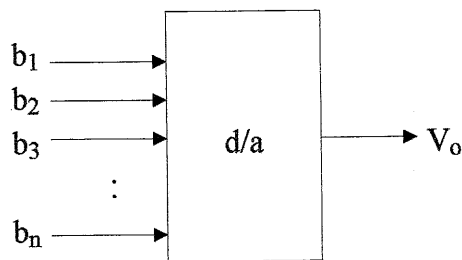
After analog data have been converted to digital form and have been duly stored, transmitted, or processed, the results to this, with some digital numbers may be required once again to intervene in the real world of phenomena. In analog or digital form, they may be used to drive meters or motors, display information, simulate devices under test, generate heat, light or sound, modulate wave forms, sound the alarm, adjust an audio gain, ... etc.

While an increasing number of real-world devices, such as numerical displays, stepping motors, printers and the like, are operated more or less directly by digital numbers. As with a/d conversion, the basic objective of d/a converter (DAC) is to get the data into the analog form, as rapidly, as accurately, as completely, as necessary.

In response to a digital code, the DAC may be used either to provide a voltage or current output (fixed-reference DAC), or to adjust the gain of an analog circuit (multiplying DAC). It can be a simple device on an IC chip, or a high-resolution, high-speed device.

To accommodate the analog output to the specified conversion relationship, some form of scaling and offsetting (signal conditioning) and energy translation (e.g., current-to-voltage) may be necessary, performed with amplifiers. To send analog information to more than one destination, either additional converters or a multiplexer and sample-holds may be necessary.

Measurements :-



The transfer function for the d/a converter is as follows.

$$V_o = V_{fs} \sum_{i=1}^n (b_i / 2^i) + V_{off}$$

V_{off} is the offset-voltage of the output voltage range and V_{fs} is the reference quantity. i.e.) a scalar multiple of the actual input reference voltage.

There are 2^n discrete voltage levels possible in the input.

All 1's output voltage is given by $v_{11} = V_{fs} (1 - 1/2^n)$

The bipolar voltage transfer function is given by,

$$V_o = -V_{fs} + 2 V_{fs} \sum (b_i / 2^i) + V_{off}$$

III. Digital Input/Output [DIO]:

In many situations, where TTL signals of HIGH and LOW are used, the Digital input/output module is required. It's applications include selecting one of the many multiplexed channels, making high/low of particular TTL line in a circuit, addressing, enabling on/off of a switch ... etc. All these kind of applications are used in large scale analog-to-digital and digital-to-analog set-ups.

Normally several ports of digital signals are required for both input/ output operations in a system. These buffers provide lot of flexibility to the users, so that any ports can be selected for input operation and/or any ports can be selected for output operations. With these, the computer can directly access, set/reset the TTL lines in any real world devices.

Chapter 2 ANALYSIS

2.1 Device set-up & Function:

The primary objective of this system is to characterise the W-band millimeter wave mixer. The architecture of the device (mixer test system) under test is depicted in the diagram. There are three local oscillators (LO_1, LO_2, LO_3) being used to generate the LO signals, at standard frequencies viz. 114 GHz, 92.4 GHz, and 83.4 GHz respectively. At a time, through a wave guide switch, one of the three local oscillators is selected. Wave guides are used here to pass the RF signals. An attenuator is used to control the power output of the oscillator. A pre-calibrated 'directional-coupler' is used to monitor the power input to the mixer.

The 'directional-filter' adjacent to it is tuned to the given oscillator and the signal is then fed to the mixer input. The IF output of the mixer is connected to SMTS, through a co-axial cable, to measure the dc characteristics and the noise performance of the specimen mixer.

2.2 Measurement process :

In characterising the mixer,

- (i) the DC characteristics and
- (ii) the RF analysis are performed.

The parameters to be measured in the system are,

1. Bias voltage (V_{mon})
2. Current monitor (I_{mon})
3. Temperatures of the mixer T, corresponding to the noise source
4. RF signal power.

There are many other parameters calculated from the measured values. They are R, VSWR, L_m , L_a , T_m , T_a , T_d .

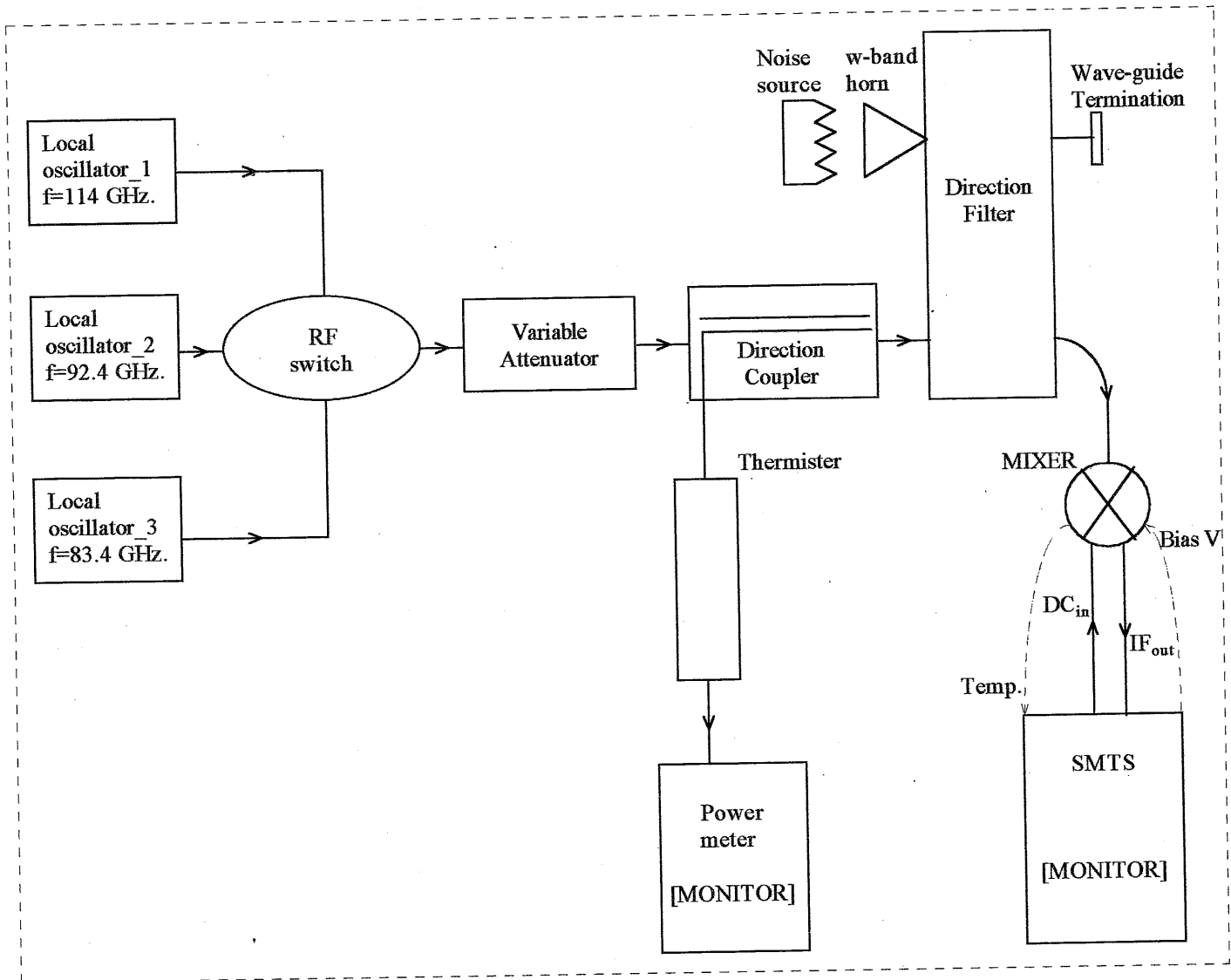


Fig. : Device Under Test - The Mixer Test System

The following mechanical, electrical, electro-mechanical control operations are made on the device during the measurement.

- (i) selecting a local oscillator (electronic & mechanical)
- (ii) adjusting the attenuator (mechanical)
- (iii) To set the required power in the power meter
- (iv) setting the noise switch On/Off (electronic)
- (v) putting the noise source in hot/cold (mechanical)
- (vi) setting the Bias voltage through potentiometer (mechanical)
- (vii) setting current monitoring range in 0.1mA and 10.mA (mechanical)

Among these controls, setting the 2-state On/Off switches is simpler and can be made with the relay mechanism. Setting many analog signals through the potentiometer is an electronic operation, which in turn increases or decreases the signal connected at one end, from a maximum value to 0. This effect can be accomplished by providing equivalent voltage to the required point with the use of digital-to-analog converter (DAC). Appropriate scaling & signal conditioning mechanisms are necessary at those places, to provide safer precise values. Some of the controls, which are purely mechanical and needs variant adjustment levels are little bit difficult to automate. The attenuator control is one such kind and can be made automatic with the use of precise stepper-motors or dc motors.

I. DC characteristics :

This measurement is performed to find the DC characteristics of the diode (mixer). This is done before starting the RF analysis. A typical V-I relationship for the diode is found by first setting the I_{monitor} current to some standard levels such as, 1 μA , 10 μA , 50 μA , 0.1 mA ... etc. During this setting, the current monitoring range switch is to be selected to either of the 0.1 mA or 10 mA Ranges. When each of the required bias current has been set through the potentiometer, the I_{mon} , V_{mon} are noted down. The noise switch is made Off and the temperature is measured as Temp_{if} and the switch is made On to get the temperature Temp_{ifR} (temperature_reflected). Actually when the noise switch is made On, +28 volts is applied and the energy loss of RF signal sent is calculated from the signal reflected from the source.

I Amps	V volts	T _{if} K	T _{ifR} K
1 μA.			
10 μA.			
50 μA.			
0.1 mA.			
0.5 mA.			
1.0 mA.			
5.0 mA.			

II. RF analysis :

Here the bias voltage V is set to particular level and the required power P is set through attenuator as read on the power meter accordingly. The current I_{mon} is measured in the SMTS. Then the temperatures corresponding to noise-switch position Off, On are measured as T_{HOT} and $T_{\text{Reflected}}$. Then the noise absorber is dipped in liquid nitrogen and placed before the horn connected to the directional filter. This temperature is measured as T_{COOL} . From these measured values, the derived measures, such as R , V_{SWR} , L_a , L_m , T_a , T_m are calculated. For each power set P , different bias voltages are set and the current and temperatures are measured. This is continued for other sets of power. This process is repeated for other LO frequencies also.

Frequency = 114 GHz.		← measured values →				values →
Power P watt.	Bias voltage V volts.	Current I Amp.	Temp T _h K	Temp T _r K	Temp T _c K	R
P ₁	V ₁	I ₁	-	-	-	-
		I ₂	-	-	-	-
		I ₃	-	-	-	-
	V ₂	I ₁	-	-	-	-
		I ₂	-	-	-	-
		I ₃	-	-	-	-
P ₂	V ₁	I ₁	-	-	-	-
	V ₂	...				

This is repeated for other LO frequencies at 92.4 GHz. and 83.4 GHz.

Along with these measurements, the following details are also noted down for each measurements.

- Diode identification
- Mixer identification
- Whisker length before bending
- Whisker bent height
- Whisker post or pin protrusion
- Plate temperature
- Ambient temperature and
- Liq. N₂ temperature.

In these measurements, even though there are different parameters, they are basically exist in the form of voltages. Multi-channel analog-to-digital converters (ADC) can be used to acquire these signals. For setting the bias voltage to the mixer, the digital-to-analog converter (DAC) is to be used. The voltage range of each of the parameter to be measured is important in the designing of the circuitry for the data acquisition system. So the data to be acquired with required resolution are listed below.

Parameter	signal range	Resolution required
Voltage V : 1. DC characteristics 2. RF analysis	-100 mV to -1150 mV -0.5 V to -1.0 V	1 mV 5 mV
Current I : 1. DC characteristics 2. RF analysis	1 μ A to 5 mA 0.6 mA to 2.5 mA	1 mV 1 mV
Power P :	0.04 to 1.0 mW	better than 0.01 mW
Temperature T :	1 K to 2000 K	1 K

The measurement system has been analysed. The block diagram and the Data acquisition system's requirements are as follows :

2.3 Data Acquisition system - Requirements :

The system has to perform the following measurement :

- Bias voltage V
- Bias current I with control over current setting
- Power P
- Temperatures T_{if} , T_{ifr} , T_h , T_r , T_c with noise switch setting

The system has to control the following :

- ◆ Bias current setting
- ◆ Noise switch setting

The system also has to provide the following derived measurements :

- ◇ VSWR - voltage standing wave ratio
- ◇ L_m - mixer conversion loss
- ◇ T_m - mixer noise temperature
- ... etc.

The system has to display the signal values while setting / and measuring data. It has to make the statistical analysis every time it acquires the data. The creation of data files for the specific measurements and storing the data in the formatted way are also to be managed by the system.

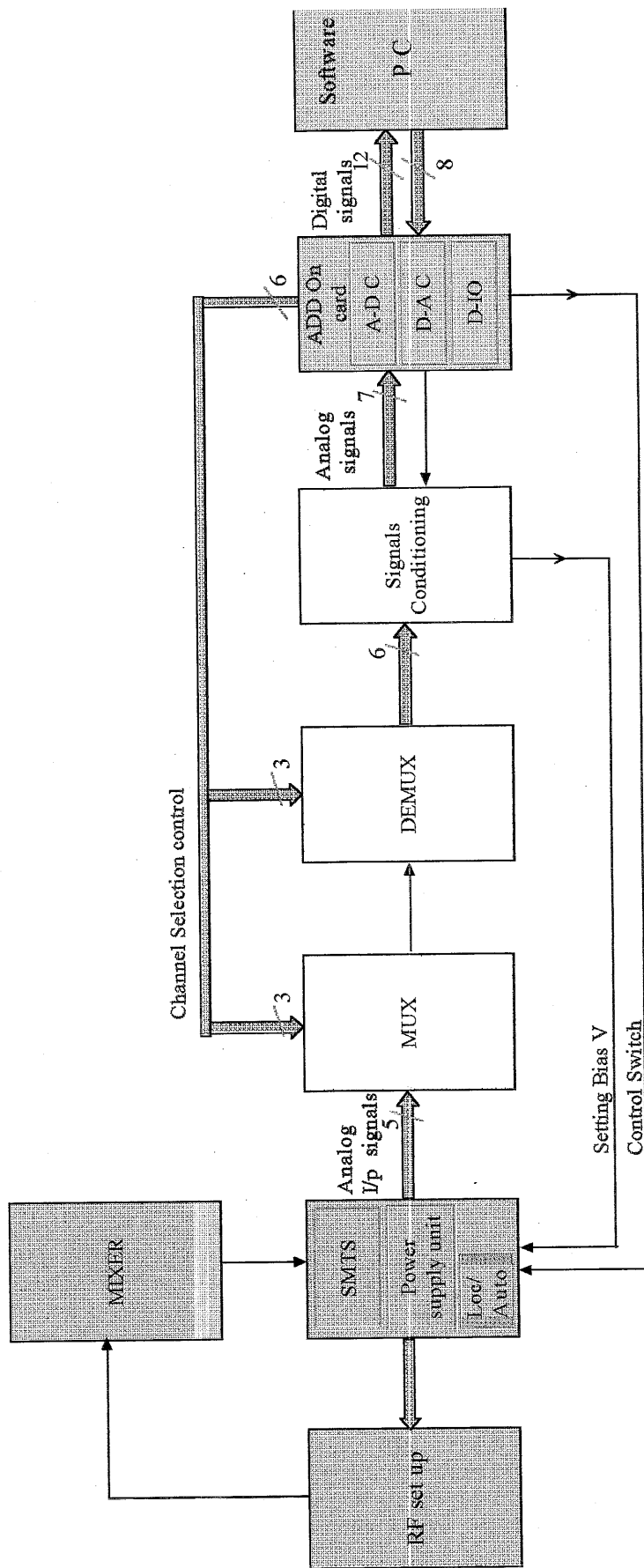


Fig : Level2 Block diagram

We will make the following strategic assumptions :

- A single-board computer (SBC) with a 386/ 486 class processor will be used.
- A set of signal converter hardware interfacing circuits will be used for measuring the analog signals with the computer which are accessible via memory mapped I/O.
- The date is supplied by and on-board clock, accessible via memory-mapped I/O.
- The signal controlling such as bias current setting will be managed by digital-to-analog modules of the interfacing hardware by the PC.
- The On/ Off switch setting of the noise switch will be controlled by relay mechanism.
- User information and interface is provided through the keyboard.
- The display is the CRT display and set of LED displays for indicating signal path propagation.

The following is the process diagram that illustrates the hardware platform on which the data acquisition software will be executed.

Software configuration :

Programming Language - Borland C++ ? C ✓

Operating system - DOS 6.2

Processor - 486

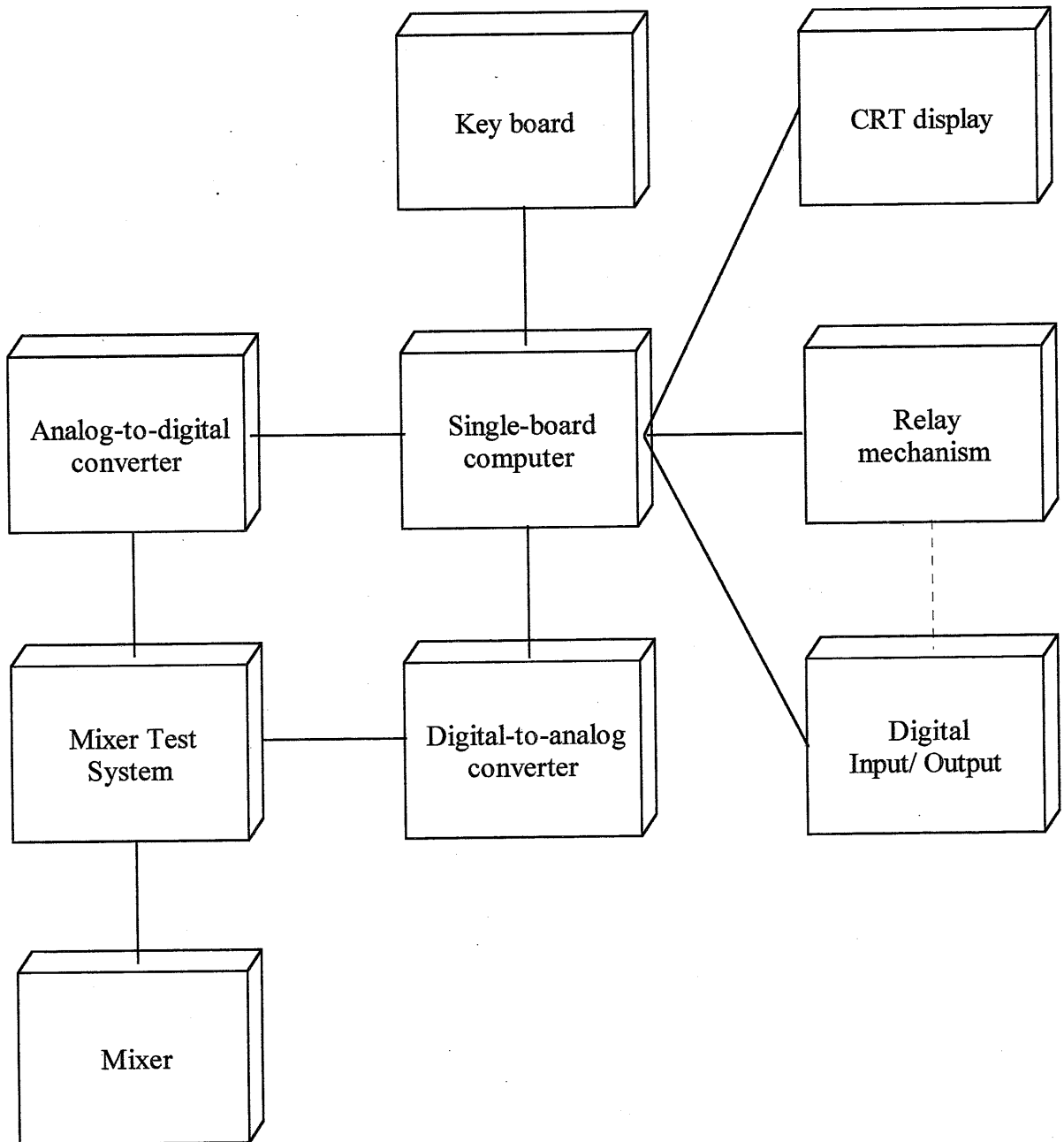


Fig. : Hardware Platform

The measurement system will be either in 'Local' mode or 'Auto' mode. The data acquisition will be performed only if the system is in the auto mode and the system will display message if the mode is in local. So the initial part of the software will check this mode connection. This module will also set the required default values to hardware devices. The specification of this class 'initialise' is as follows.

Name :

Initialise

Responsibilities :

Check the on-line connection between the measuring system and the on-board computer via the interface circuits. Set the default values into the hardware devices.

Operations :

Connectivity

Initialise

Attributes :

dac-input, adc-output, dio-input.

The class 'acquire' will serve the basic acquisition of analog signals from the mixer test system. The same is used for acquiring all signals such as voltage, current, power and temperatures.

Name :

acquire

Responsibilities :

Acquisition of all analog signals from the mixer test system and performing the analysis part.

Operations :

dc_measure

rf_measure

data_file

Attributes :

adc_output, dio_input, dac_input

The operation dc_measure is used to perform the DC characteristics portion of the measurement as explained in the topic 2.2. The operation rf_measure will perform the RF analysis portion of the measurement. The operation data_file is used to manipulate the various Data files where all the acquired and processed data are stored.

Chapter 3 DESIGN

3.1 Hardware construction :

About Interfacing Circuit :

To set the bias voltage of mixer, the noise switch and to perform the data acquisition for a mixer to be characterised, the 'ALS-PC-02 Add on card' is appropriate. This Add on card contains the A/D converter, D/A converter and Digital Input/Output buffer. The features of this ALS-PC-02 Add on card are as follows.

(i) A/D Converter (Using AD 574 A) :

- ◆ It is a 12-bit ADC.
- ◆ Jumper selectable 8 differential or 16 single ended input channels.
- ◆ Jumper selectable unipolar / bipolar analog input range (0 to +10v, $\pm 5v$, $\pm 10v$).
- ◆ Three programmable onboard timers available (using IC 8253). These timers can be used for acquisition sequence control/ timing, external synchronisation, interruption mode.
- ◆ Provision to connect TTL compatible external trigger source for acquisition sequence.
- ◆ Provision to perform data acquisition in DMA mode using the system DMA channel 1.

(ii) D/A Converter (Using DAC 1220) :

- ◆ It has two 12-bits programmable D/A converters using DAC 1220 IC.
- ◆ Jumper selectable unipolar / bipolar analog output ranges (0 to $\pm 10v$, $\pm 10v$).
- ◆ Fast settling time of less than 3 μ secs.

(iii) Digital Input / Output (Using 8255 IC's) :

- ◆ Contains 48 programmable digital I/O lines, using two 8255's.
- ◆ The I/O lines are terminated in two 26 pin headers for extending them conveniently.
- ◆ Provision to connect one bit of each 8255 to the system interrupt lines IRQ2-IRQ7.

Measurement resolution :

The measurement resolution is calculated as follows.

$$\text{Resolution} = \text{signal voltage (full-scale) range} / \text{No. Of input lines}$$

Converter	Input / Output bits	Voltage range	Measurement resolution
A/D converter	12 bits o/p	0 to 10 v	$10 \text{ v} / 2^{12} = 2.44 \text{ mV}$
		$\pm 5 \text{ v}$	$10 \text{ v} / 2^{12} = 2.44 \text{ mV}$
		$\pm 10 \text{ v}$	$20 \text{ v} / 2^{12} = 5.88 \text{ mV}$
D/A converter	12 bits i/p	0 to 10 v	$10 \text{ v} / 2^{12} = 2.44 \text{ mV}$
		$\pm 10 \text{ v}$	$20 \text{ v} / 2^{12} = 5.88 \text{ mV}$

The 'resolution' value defines the least voltage that can be measured / adjusted for the converter in the particular voltage range.

The signal equivalent voltages & resolution required for each parameter are,

Parameter	Measurement range	Equiv. voltage	Resol.n required
1. Voltage V	0 to -1.2 v DC → -600 to -1150mV RF → -5000 to -1000mV	0 to -1.2 v	1 mV <5 mV
2. Current I	1 μ A to 5 mA (In the full scales of 0.1 mA & 10 mA)	0 to -5 v	1 mV
3. Power P	0.01 mW to 1.0 mW	0 to 1 volts	better than 10 mV
4. Temperatures			
DC: T_{if}	50° to 500° K	0 to 0.5 v	1 mV
T_{ifr}	73° to 2000° K	0 to 2.0 v	
RF: T_h	150° to 450° K	0 to 0.5 v	
T_r	150° to 880° K	0 to 1.0 v	
T_c	100° to 450° K	0 to 0.5 v	

Since the signals to be measurements are exist in both +ve and -ve voltages, we select the bipolar ± 5 volts input range in the A/D converter. In the D/A we select the unipolar 0 to +10 v output range which has better resolution.

Signal conditioning :

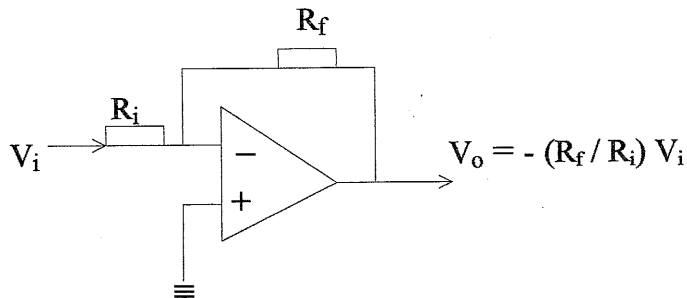
To achieve the required resolution for each of the parameters, we first select the smaller full scale voltage range ± 5 v in the ADC. The conversion techniques can be employed which expands the signal voltage range to cover

the entire A/D voltage range. This 'Signal Conditioning' process can be achieved through the application of 'operational amplifiers'.

① *Operational Amplifiers :*

(a) Inverting follower :-

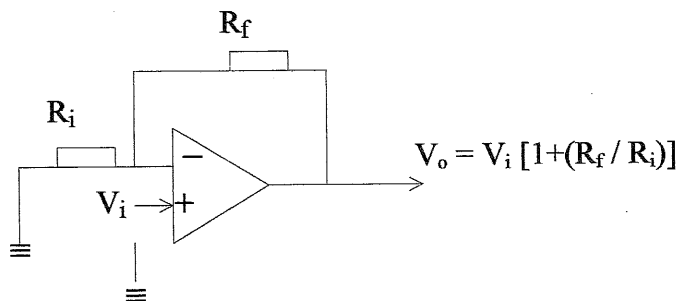
Here the input voltage is applied at the inverting-input terminal of the Op-Amp.



The gain of this Op-amp is given by $A_v = - R_f / R_i$

(b) Non-inverting follower :-

Here the input voltage is applied to the non-inverting input terminal of the Op-amp.



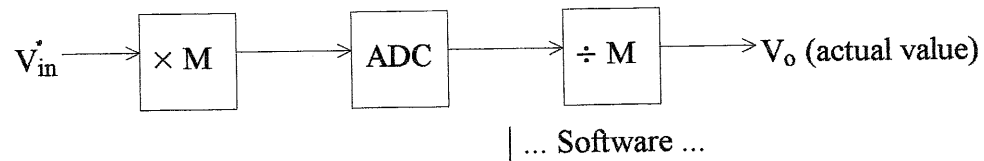
The gain of this Op-amp is given by $A_v = 1 + (R_f / R_i)$

So, when the signal is amplified by the gain, through the Op-amp and then fed to the ADC input, the resolution gets improved. The multiplier value of the Op-amp for each parameter is calculated as follows.

$$\text{Signal full-scale value} \times M = \text{ADC full-scale value}$$

$$M = (\text{ADC}_{\text{FS}} / \text{Signal}_{\text{FS}} \text{ value})$$

It is to be observed that, the output of the ADC is to be divided by the equal multiplier-amount to get actual values.



Let us consider the gain required for each measurement, for the ADC ± 5 v bipolar range mode.

1. Current I:

The input voltage full scale range is 0 to -5 volts. It requires both multiplication and addition to cover the entire ± 5 v ADC range. The multiplier value is 2 and the addition is of +5 volts. So the gain of the Op-amp is $2V_i+5$ volts.

V_i	$ADC_{in} = 2V_i + 5$
- 5 v	- 5 v
:	:
0 v	+ 5 v

The reverse operation to be done at the ADC output is $[ADC_{out}-5] / 2$

ADC_{out}	$V_o = (ADC_{out}-5) / 2$
- 5 v	- 5 v
:	:
+ 5 v	0 v

2. Voltage V:

The input voltage range is 0 to -1.25 volts. It also requires multiplication and addition operations in the Op-amp. The gain is to be set for $8V_i+5$ volts.

V_i	$ADC_{in} = 8V_i + 5$
-1.25 v	- 5 v
:	:
0	+ 5 v

The reverse operation required at the ADC output is $[ADC_{out}-5] / 8$

$\frac{\text{ADC}_{\text{out}}}{-5 \text{ v}}$	$\frac{V_o = (\text{ADC}_{\text{out}} - 5) / 8}{-1.25 \text{ v}}$
:	:
$+5 \text{ v}$	0 v

3. Power P :

The input voltage range is 0 to 1 volts. The resolution required is better than 0.01 volts. i.e.) atleast 0.005 volts. So the 2.44 mV resolution of the \pm volt range ADC is enough for power measurements. So the gain 1 (unity gain) Op-amp can be applied for this measurement.

$$\text{ADC}_{\text{in}} = V_i \times 1$$

It requires no reverse function since the gain used is unity.

4. Temperatures :

The input voltage for the current and voltage parameters are in -ve range, so when the multiplier is used, it also requires addition of some voltages. All the temperature signals T_{if} , T_{ifr} , T_{h} , T_{r} , T_{c} are exist is positive voltages only. So only appropriate multiplication is required for the conversion. All the temperature signals require 1 mV resolution.

The multiplier value for different temperatures are as follows.

I. DC characteristics :-

- (i) T_{if} : The signal input voltage range is 0 to 0.5 volts. So the converted values to be with in + 5 volts. The appropriate multiplier value is 10. $\text{ADC}_{\text{in}} = 10 \times V_i$
- (ii) T_{ifr} : The signal input voltage range is 0 to 2.0 volts. Multiplier value is 5. $\text{ADC}_{\text{in}} = 5 \times V_i$.

II. RF analysis :

- (i) T_{h} : The signal input voltage range is 0 to 0.5 volts. Multiplier value is 10. $\text{ADC}_{\text{in}} = 10 \times V_i$.
- (ii) T_{r} : Signal input voltage range is 0 to 1 volts. Multiplier value is 5. $\text{ADC}_{\text{in}} = 5 \times V_i$.
- (iii) T_{c} : Signal input voltage range is 0 to 0.5 volts. Multiplier value is 10. $\text{ADC}_{\text{in}} = 10 \times V_i$.

Gain for DAC output :

The DAC output is to be used to set the bias current. In the manual bias current setting, a potentiometer is used to give bias voltage in between 0 to -1.5 volts. Since the DAC is set in unipolar 0 to +10 v range, this output has to be converted to the range of 0 to -1.5 v. So the division of the DAC output by 6 with sign inversion [$V_o = -(V_i / 6)$] will do that.

DAC_{out}	$\text{Bias } V_{\text{in}} = -(\text{DAC}_{\text{out}} / 6)$
0 v	0 v
:	:
10 v	-1.6 v

So the final Signal-conditioning methods are listed as follows.

Parameter	Signal range	Equiv. voltage	Conversion	Re-conversion
1. Current I_{mon}	0 to -5.0 v	0 to -5.0 v	$V_1 = 2V_i + 5$	$V_2 = (V_o - 5)/2$
2. Voltage V_{mon}	0 to -1.2 v	0 to -1.2 v	$V_1 = 8V_i + 5$	$V_2 = (V_o - 5)/8$
3. Power P	0.01 to 1.0 mW	0 to 1 v	$V_1 = 1V_i$	-
4. Temperatures				
DC charact. :				
(i) T_{if}	50 to 500 K	0 to 0.5 v	$V_1 = 10V_i$	$V_2 = V_o / 10$
(ii) T_{ifr}	70 to 2000 K	0 to 2.0 v	$V_1 = 5 V_i$	$V_2 = V_o / 5$
RF analysis :				
(iii) T_h	150 to 450 K	0 to 0.5 v	$V_1 = 10V_i$	$V_2 = V_o / 10$
(iv) T_r	150 to 880 K	0 to 1.0 v	$V_1 = 5 V_i$	$V_2 = V_o / 5$
(v) T_c	100 to 450 K	0 to 0.5 v	$V_1 = 10V_i$	$V_2 = V_o / 10$

If the gain blocks are selectable, then we can use any particular gain block to any other input signals. It enables the input signals to pass through any wanted gain blocks. This method can be achieved by the analog multiplexer-demultiplexer combination.

The total gain blocks needed are,

Gain $\times 2+5$

Gain $\times 8+5$

Gain $\times 1$

Gain $\times 4$

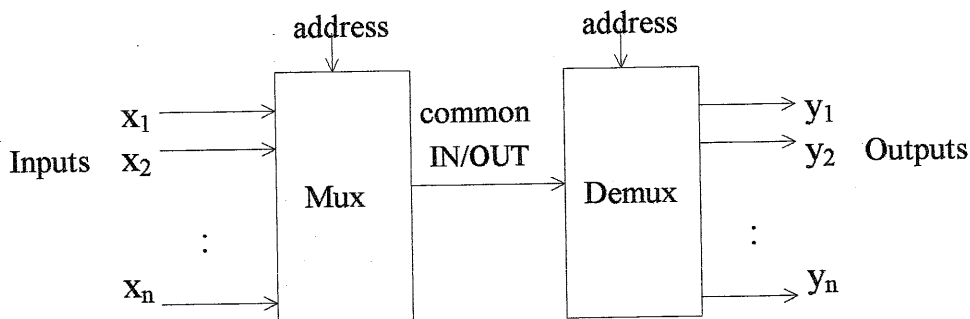
Gain $\times 5$

Gain $\times 10$ &

Gain $1/6$ for DAC output conditioning.

② Analog Multiplexer/ Demultiplexer :

To connect different gain blocks to the multi input signal channels and to make the gain blocks selectable by any other input channels, the analog multiplexer-demultiplexer combination can be designed. This set up will give many to many relation as shown in below.



Analog multiplexer/ demultiplexer IC MC 14051 B :-

This IC has 8 Input/ Output channels and three digital control inputs. The three binary signals select 1 of 8 channels to be ON and connect one of the 8 inputs to the output in the case of MUX. When these devices are used as demultiplexers, the 'channel IN/OUT' terminals are the outputs and the 'common IN/OUT' terminals are the inputs.

The block diagram of these hardware set up is shown in the figure.

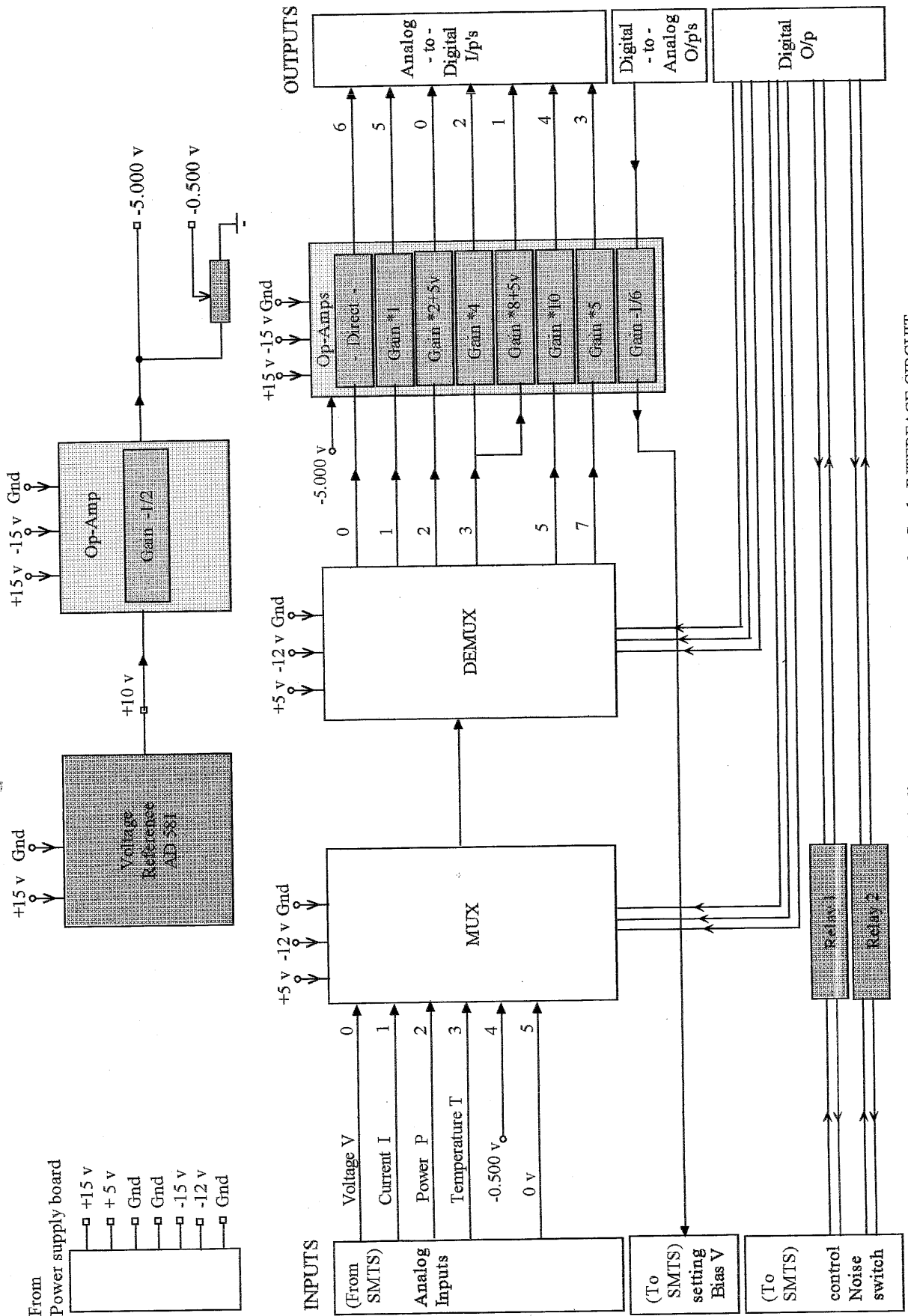


Fig : Block diagram - SMTS to PC ADD On Card INTERFACE CIRCUIT

3.2 Software :

Architectural Framework :

The architecture encompasses the pattern of well defined classes and the synchronisation of various operations inside each of the classes. The Class diagram and the State Transition diagram that express this architecture for the data acquisition system are shown in the figure.

The class 'Initialise' will be executed first prior to the start of measurements. The declaration for that will be as follows.

```

Class Initialise {
public :

    connectivity();
    Initialise();
    ...
};

Initialise::connectivity {
    output(DAC_port_id, 0);
    output(ADC_channel_mode,channel0);
    value1=inport(ADC_port_id);
    output(DAC_port_id,5000);
    output(ADC_channel_mode,channel0);
    value2=inport(ADC_port_id);
    if ((value1>10) && (abs(value2)<1000))
        strcpy(mode,'Local')
    else strcpy(mode,'Auto');
}

Initialise::Initialise {
    output(DAC_port_id,0x000);
    output(DIO_port_id,0x00);
    struct date d;
    getdate(&d);
    strcat(dcfiler,year);
    strcat(rffiler,year);
    today = d;
}

```


Class diagram

28

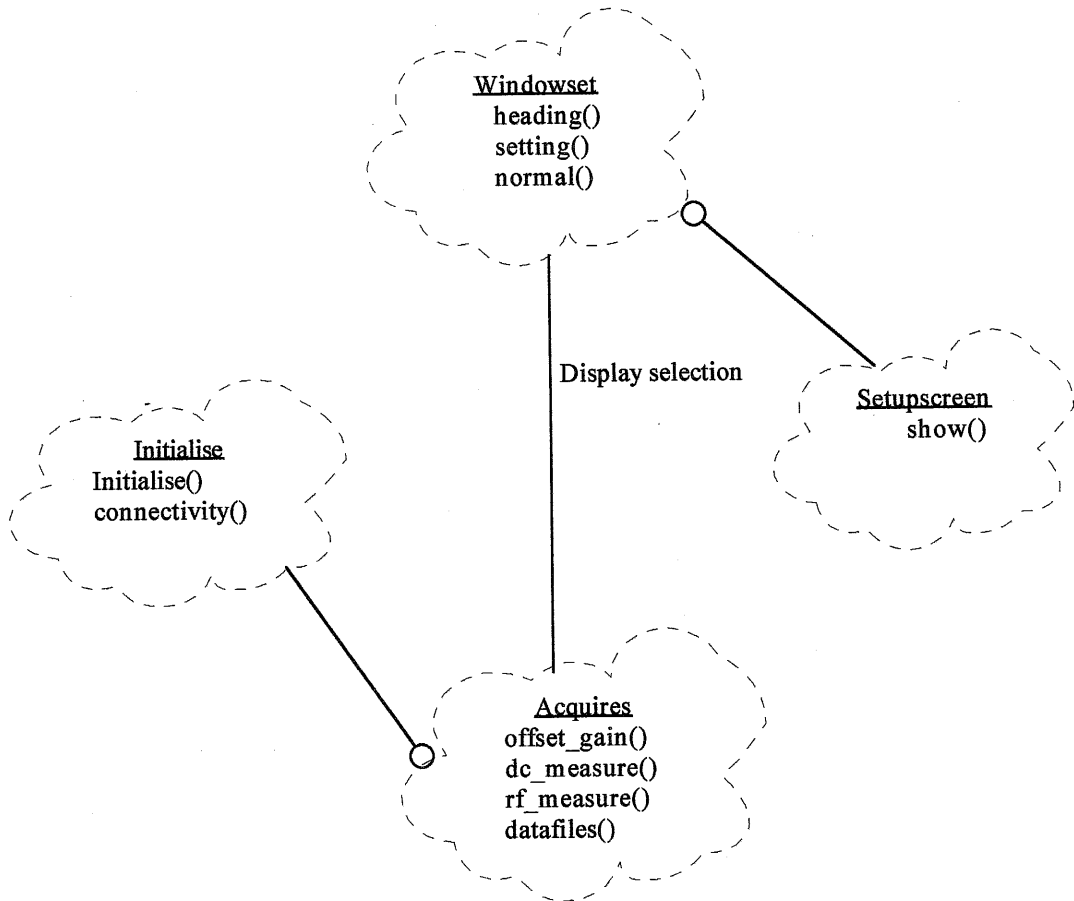


Fig : Class diagram

State Transition Diagram :

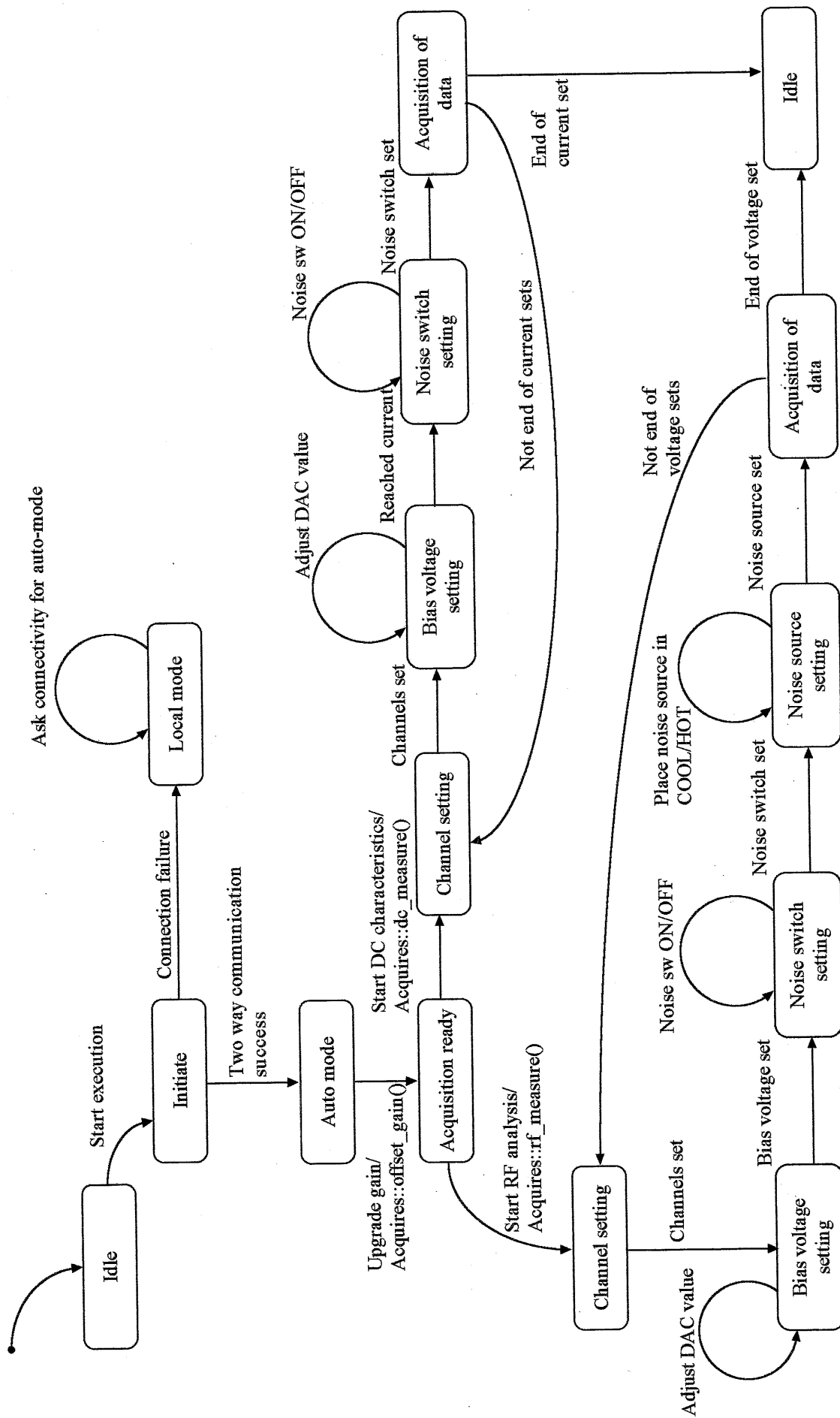


Fig : State Transition diagram

Here to access the hardware circuits and to make control over them, the basic library functions INPORT and OUTPORT are being used. The function inport() reads a word from the mentioned hardware port_id connected to the PC. Another function inportb() does the same, but reads a byte from the port. The function outport() writes a word into the hardware register indicated by the port_id. The function outportb() does the same, but writes a byte to the port.

Only the basic declaration for these function are given here. The operation of these functions need several other allied communication functions and settings prior to the start of data acquisition through ADC, which are explained in detail in the next chapter.

The class 'Windowset' contains all the operations to handle the screen displays for the data acquisition. The declaration for this class is as follows.

```
Class windowset {
public :

    void heading();
    void setting(int l, int t, int r, int b, int back, int text);
    void normal();
    ...
}

windowset::heading {
    window(left,top,right,bottom);
    textbackground(color1);
    textcolor(color2);
    printf("%s",title);
}
windowset::setting(int l, int t, int r, int b, int back, int text) {
    window(l,t,r,b);
    textbackground(back);
    textcolor(text);
}
```

These objects makes the opening menu selection screen, various displays and provides the platform for user interface.

The class 'acquire' performs the data acquisition process. Its operation is largely divided into two as `dc_measure` and `rf_measure`. Each has unique hardware access, ADC input channels setting and noise switch setting in the measurement. They require careful synchronisation of several tasks.

On signal acquisition, the number of samples and the acquisition delay time are fixed to the optimum level. Whenever there is a need to change these values, it can be changed in the declaration part of the program. The Analog-to-Digital converter has totally 16-input channels and we use 7-input channels for our acquisition. The ADC will read the signal from only one channel at a time. In the software-pollled mode of the ADC, it starts the analog-to-digital conversion immediately, once the ADC channel register is set with the required channel number. If any signal-conditioning circuit is employed before the signal is applied to the ADC input channels, then it requires additional settings to be performed on these circuits from the computer. After many samples have been acquired over the data, these will be averaged and some statistical analysis is applied to these samples to find the integrity of the data.

The data file name for each set of measurements is generated in the 'Initialisation' class from the current date of the system clock. The measured data are stored along with the calculated values. All these information are stored into the created data file in the required format. The declaration of this 'acquire' class is as follows.

```
Class acquire {
public :

void offset_gain();
void dc_measure();
void rf_measure();
int datafile();
...
}
```

The multiplier values of each of the gain blocks(signal-conditioning) through which all signals are passed are significant since the same exact value is used for dividing at the output of ADC. This reverse operation of signal-conditioning on the data is performed in the software side. The 0 volts input and the standard input voltage of -0.500 volts are used for this purpose. To get the exact gain values, this operation is done.

The function of this 'offset_gain' operation is as follows.

1. Set the mux-demux channels for 0 v input through DIO_PORTB.
2. Give delay of 'time1' millisecs to settle.
3. Set the ADC input channel for this through DIO-PORTB.
4. Give delay of 'time2' millisecs for conversion.
5. Measure the bits at output of ADC.
6. Calculate voltage(off) from bits. The off value for gain-2 is volt5offset.
7. Repeat step-1 to step-5 for -0.500 v input.
8. Calculate voltage(value) from bits.
9. Gain $x = \text{abs}(\text{value} - \text{off})/0.5$
10. Repeat step-1 to step-9 for all the gains.

```

Acquire::offset_gain {
  int gain2=2,gain4=4,gain5=5,gain8=8,gain10=10,volt5=5;
  unsigned int off_mux[]={ 0x15, 0x1d, 0x1d, 0x2d, 0x3d };
  unsigned int set_adc []={ 0x80, 0x82, 0x81, 0x84, 0x83 };
  unsigned int gn_mux[]={ 0x14, 0x1c, 0x1c, 0x2c, 0x3c };
  for (int i=0;i<5;i++)
  { output(DIO_PORTB,off_mux[i]);
    delay(time1);
    output(ADC_MODE,set_adc[i]);
    delay(time2);
    int bits=inport(ADC_LOWB);
    float off=bits*resolution-5000;
    output(DIO_PORTB,gn_mux[i]);
    delay(time1);
    output(ADC_MODE,set_adc[i]);
    delay(time2);
    bits=inport(ADC_LOWB);
    float value=bits*resolution-5000;
    switch(i)
    { case 0 : gain2=fabs(value-off)/500;
      volt5=off; break;
      case 1 : gain4=fabs(value-off)/500; break;
      case 2 : gain8=fabs(value-off)/500; break;
      case 3 : gain10=fabs(value-off)/500; break;
      case 4 : gain5=fabs(value-off)/500; break;
      default : break; }
  }
}

```

The function of 'dc_measure' operation is as follows.

1. Set the mux-demux input channels for input I.
2. Delay for 'time1'.
3. Set the ADC input channel for input I.
4. Delay for 'time2'.
5. Measure bits at the ADC output.
6. If (bits < I_require) then increment dac_input, else decrement dac_input.
7. If abs(bits - I_require) < 2, then go to step-8 else apply dac_input to DAC, go to step-5.
8. Set channels for V,I,T with noise_sw Off, T with noise_sw On.
9. Measure ADC output.
10. Apply reverse conversion to the data.
11. Store the data into dcfile.
12. Repeat step-1 to step-12 for all current settings.

```

Acquire::dc_measure {
    unsigned int dc_mux[]={ 0x11, 0x18, 0x2b, 0x7b };
    unsigned int dc_adc []={ 0x81, 0x80, 0x84, 0x83 };
    unsigned int i_require[]={ 0xfae, 0xe66, 0xcc, 0x000 };
    int dac_input=0,k;
    for (int j=0;j<8;j++)
    { k=(j%4);
      do
      { output(DIO_PORTB,0x11);
        delay(time1);
        output(ADC_MODE,0x80);
        delay(time2);
        int bits=inport(ADC_LOWB);
        if (bits<i_require) dac_input++;
        if (bits>i_require) dac_input--;
        output(DAC_IN,dac_input);
      } while (abs(bits-i_require[k])>2);
    }
    for (int i=0;i<4;i++)
    { output(DIO_PORTB,dc_mux[i]);
      delay(time1);
      output(ADC_MODE,dc_adc[i]);
      delay(time2);
      bits=inport(ADC_LOWB);
      float value=bits*resolution-5000;
      data[i]=value/gain[i]; }
  
```

```

fp=fopen(dcfiler,"w+");
for (i=0;i<4;i++) fprintf(fp,"%f",data[i]); }
}

```

The function of 'rf_measure' operation is as follows.

1. Input the LO frequency value freq.
2. Set the mux-demux input channels to V.
3. Delay for 'time1'.
4. Set the ADC channel for input V.
5. Delay for 'time2'.
6. Measure the bits at the ADC output.
7. Display the output voltage.
8. Get key stroke for adjusting voltage.
9. If (key='↑') then increment dac_input
 else if (key='↓') then decrement dac_input
 else if (key='Pgup') then increment dac_input by 10 counts
 else if (key='Pgdn') then decrement dac_input by 10 counts.
10. Set dac_input into DAC.
11. If (key='enter') then go to step-6.
12. Give message to set the power in power meter.
13. Set channels of mux-demux, ADC for P, V, I, T with noise_sw Off, T
 with noise_sw On, T with noise source in liq. N₂.
14. Measure ADC output.
15. Apply reverse conversion to the measured data.
16. Calculate the derived measurements.
17. Store the data into the rfile.
18. Repeat step-1 to step-17 for other LO frequencies, Power settings.

```

Acquire::rf_measure() {
    unsigned int rf_mux[]={ 0x1a, 0x18, 0x11, 0x2b, 0x7b, 0x2b };
    unsigned int rf_adc[]={ 0x85, 0x81, 0x80, 0x84, 0x83, 0x84 };
    unsigned int key;
    for (int f=0;f<3;f++)
    { output(DIO_PORTB,0x18);
      delay(time1);
      output(ADC_MODE,0x80);
      delay(time2);
      printf("Adjust the bias voltage by ↑,↓,Pgup,Pgdn keys ");
      do {
        bits=inport(ADC_LOWB);

```

```

printf("Voltage = %f",bits*resolution-5000);
key=bioskey(0);
switch(key)
{ case 18432 : dac++; break;
  case 18688 : dac+=10; break;
  case 20480 : dac--; break;
  case 20736 : dac-=10; break;
  default   : break;}
outport(DAC_IN,dac);
} while (key!=7181);
printf("Set the Power in the power meter.");
getch();
for (int i=0;i<6;i++)
{ outport(DIO_PORTB,rf_mux[i]);
  delay(time1);
  outport(ADC_MODE,rf_adc[i]);
  delay(time2);
  bits=inport(ADC_LOWB);
  float value=bits*resolution-5000;
  data[i]=value/gain[i];
  /* calculation of derived measurements here */
  fp=fopen(rfile,"w+");
  fprintf(fp,"%f",data[i]);}
}
}

```

The class 'set-up' is a special class meant for providing the hardware set-up details to the user. It will give the information like ADC voltage range, input mode selected, the channels wired for different input signals, DAC voltage range, DIO mode, ports selected, I/O operation .. etc. These information are stored in the particular file 'setup.doc' so that it can be updated whenever there is a change made in the set-up.

The declaration of this class is as follows.

```

Class setup {
public:
void show()
{ char c;
  fp=fopen("setup.doc","r");
  while ((c=fgetc(fp))!=EOF) putchar(c);
  fclose(fp); }
}

```

The summary of these operations and their relation are depicted in the following object diagram.

Object diagram

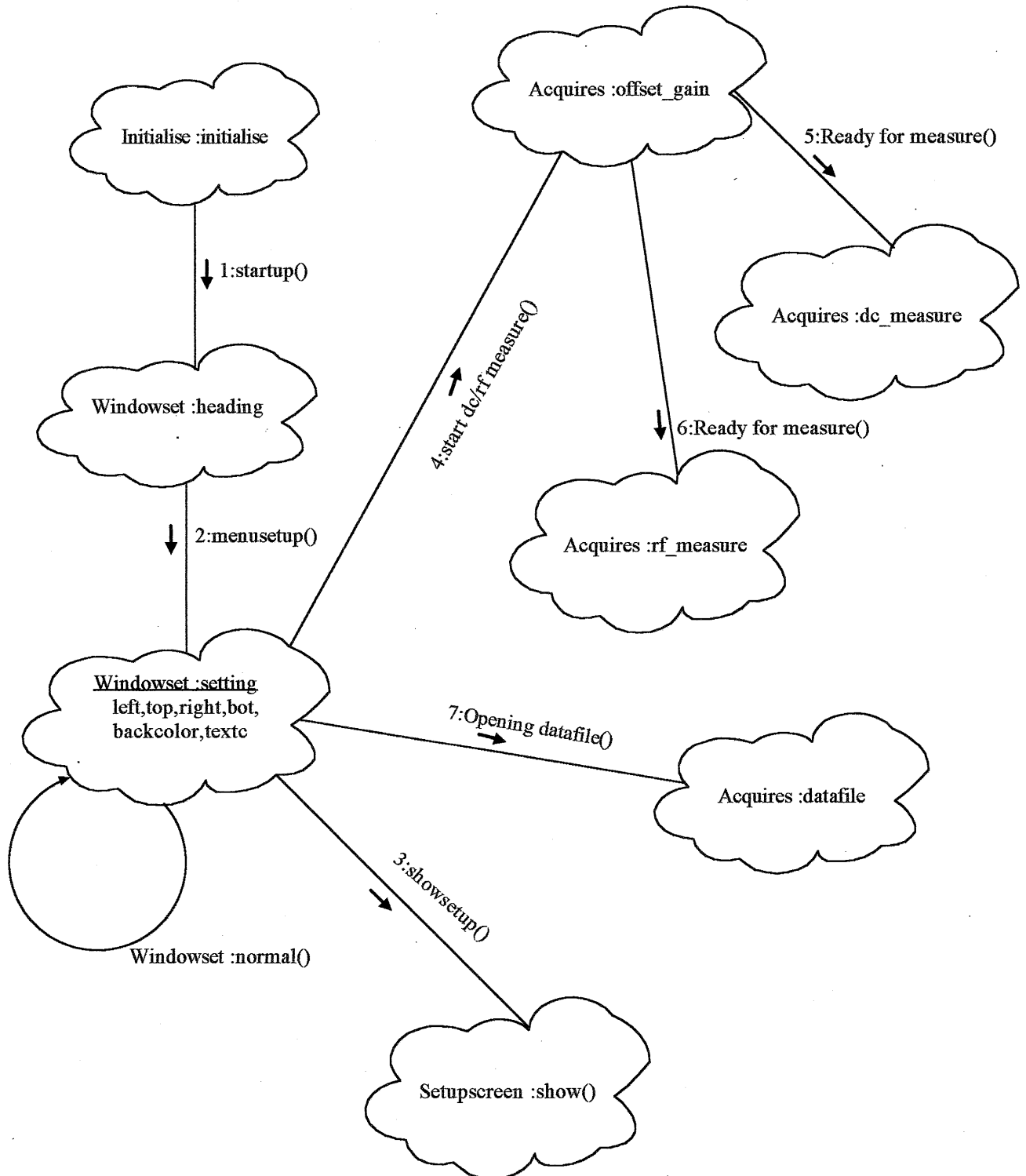


Fig : Object diagram

Chapter 4 EVOLUTION

The interfacing hardware have been constructed as designed and its functioning has been tested incrementally at various stages. Now we proceed with the incremental development of the system's function points. The process proceeds as follows.

- ◇ Developing minimal functionality acquisition, which measures one signal.
- ◇ Developing the control mechanism for current setting in the measurement system.
- ◇ Calibrating the exact conversion and reverse conversion among the data acquired.
- ◇ Completing the functions of data analysis and formatted storage of data to the file, to give expected information to the user.

Class acquire::offset_gain() :

This operation needs no external signal inputs, since the 0 v and the standard input signal -0.5 v are available within the interface board. So this module is straight away tested. The appropriate delay time required for settling the mux-demux, op-amps are obtained after several tests. This delay time plays vital role in the proper analog-to-digital conversion of signals.

Class acquire::dc_measure(),rf_measure() :

Initially for setting the bias current, the DAC output is directly used as the signal for the I_{mon} input line and the ADC is made to acquire the I_{mon} signal. For the comparison of the I and I_{require} , the actual voltage values are used first and then the bit level comparison is performed.

For the rf_measure also, the DAC output is used as the signal for the voltage setting. For both the measurements the data file format is adjusted after several tests and displays of data file information.

The development of each of the module are made iteratively and it is performed in the following way.

(i) 'Offset-Gain' operation :

For upgrading the gain values to be used in the reverse conversion process, the program module has to receive the channel setting of intermediate devices. These values are obtained and have to be set through the DIO_PORTB register. These values are to be pre-determined for the program.

The Port-B has to set the 3 control bits for Multiplexer, 3 control bits for demultiplexer and 1 for each of the two relays I, II.

The BITS \Rightarrow

Rel 2	Rel 1	Dem C	Dem B	Dem A	Mux C	Mux B	Mux A
-------	-------	-------	-------	-------	-------	-------	-------

Gain	Input	Chls M-D	BITS	in Hex	Chl in ADC
$2V_i+5$	0 v	5 2	00 010 101	0x15	0x80
	-0.5 v	4 2	00 010 100	0x14	
$4V_i$	0 v	5 3	00 011 101	0x1d	0x82
	-0.5 v	4 3	00 011 100	0x1c	
$8V_i+5$	0 v	5 3	00 011 101	0x1d	0x81
	-0.5 v	4 3	00 011 100	0x1c	
$10V_i$	0 v	5 5	00 101 101	0x2d	0x84
	-0.5 v	4 5	00 101 100	0x2c	
$5V_i$	0 v	5 7	00 111 101	0x3d	0x83
	-0.5 v	4 7	00 111 100	0x3c	
$1V_i$	0 v	5 1	00 001 101	0x0d	0x85
	-0.5 v	4 1	00 001 100	0x0c	

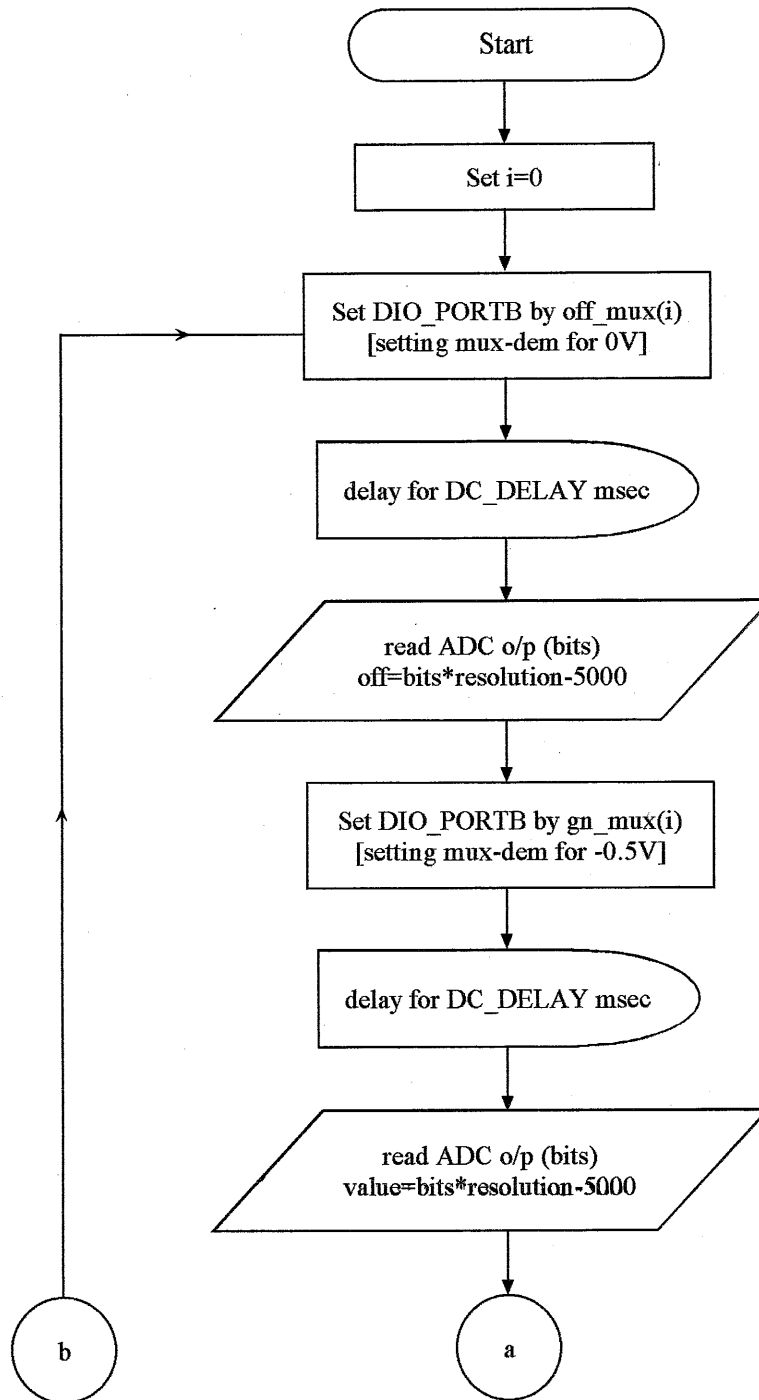
These values of channel setting are stored into arrays in the program and the DIO_PORTB is set accordingly whenever required, from the arrays. These channels selected in each part are also seen in the front panel LED's of the interfacing device.

These bits values are stored in the arrays with the following declaration.

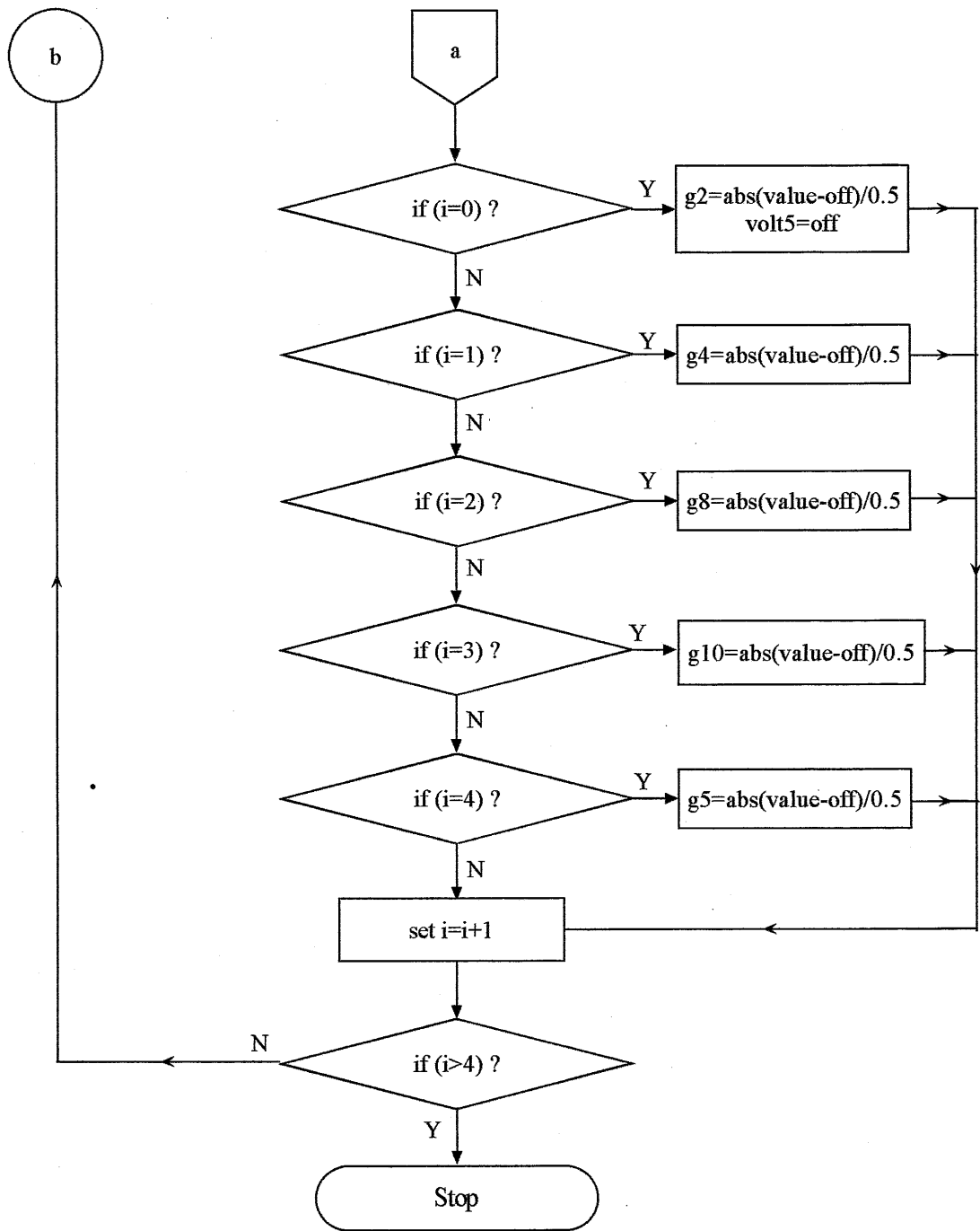
```
{ unsigned int off_mux[]={ 0x15, 0x1d, 0x1d, 0x2d, 0x3d };
  unsigned int set_adc []={ 0x80, 0x82, 0x81, 0x84, 0x83 };
  unsigned int gn_mux[]={ 0x14, 0x1c, 0x1c, 0x2c, 0x3c };
  ... ; }
```

The flow chart of this module is as follows.

Flow chart : Offset_Gain calculation



Chapter 4 DEVELOPMENT



(ii) 'dc-measure' operation :

This module has to be checked for providing automatic bias current-setting which includes comparison of ADC measured value and the required I value. This also has to acquire the I, V, T_{if} , T_{ifr} after every time, the bias current had been reached automatically to required level. Since the comparison is made at the bit level, the bit values for the $I_{require}$, which passes through a conversion phase are calculated and used in the program as follows.

V_i	$V_o = \times 2 + 5$	bit in hex	array()
0 v	+5000 mV	fff	-
1 μ A = -100mV	+4800 mV	fae	$i_{require}(0)$
5 μ A = -500mV	+4000 mV	e66	$i_{require}(1)$
10 μ A = -1000mV	+3000 mV	ccc	$i_{require}(2)$
50 μ A = -5000mV	-5000 mV	000	$i_{require}(3)$

These bits are stored in the array 'i_require' and used for the comparison. When acquisition of I, V, T_{if} , T_{ifr} are to be started the program has to obtain the channel setting bits corresponding to these for giving into the DIO port. It is calculated as before as follows.

Rel 2	Rel 1	Dem C	Dem B	Dem A	Mux C	Mux B	Mux A
-------	-------	-------	-------	-------	-------	-------	-------

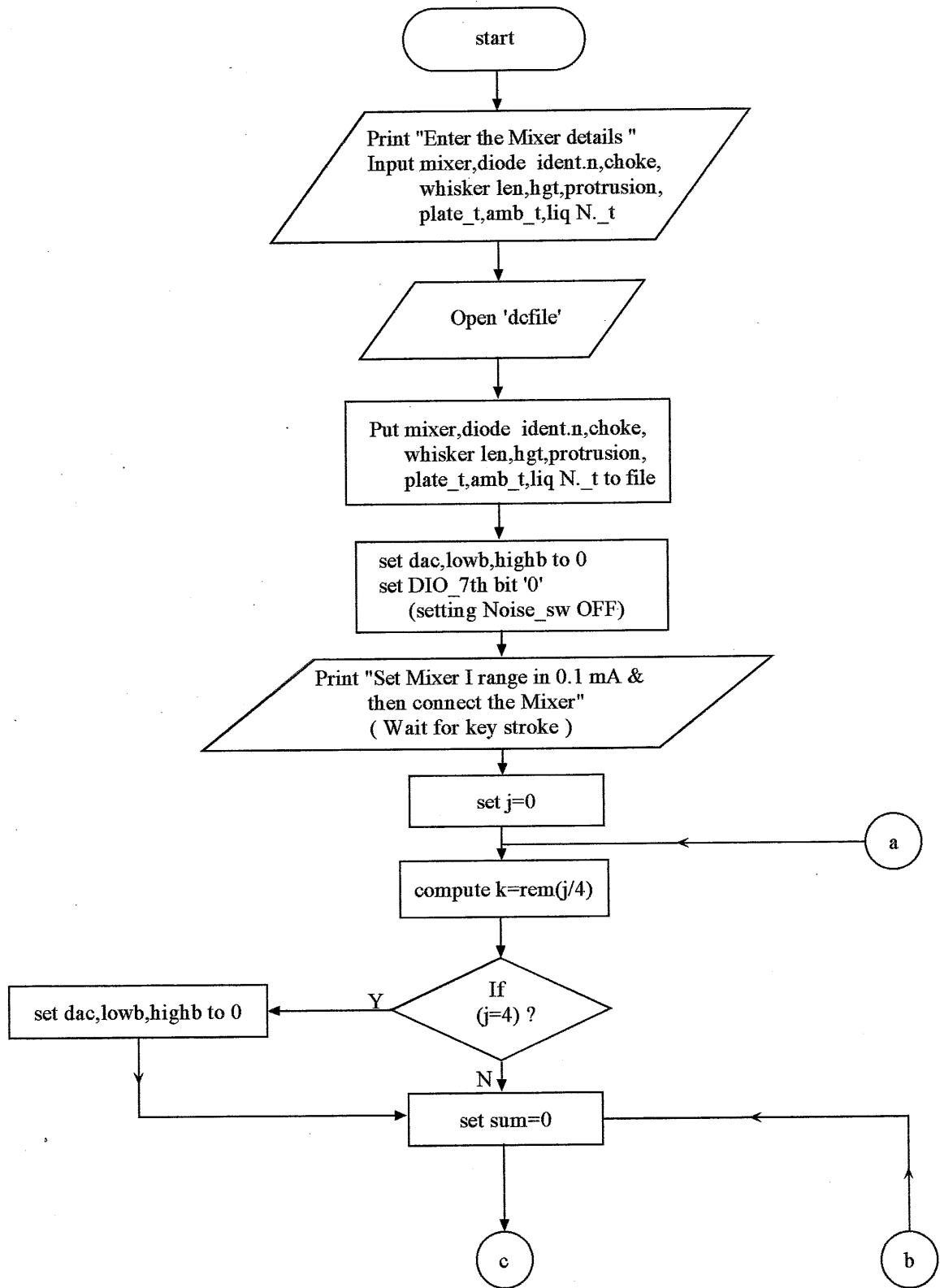
Signal	set Rl_2	Rl_1	M	D	BITS	in Hex	Chl in ADC
I	0	0	1	2	00 010 001	0x11	0x80
V	0	0	0	3	00 011 000	0x18	0x81
T_{if}	0	0	3	5	00 101 011	0x2b	0x84
T_{ifr}	0	1	3	7	01 111 011	0x7b	0x83

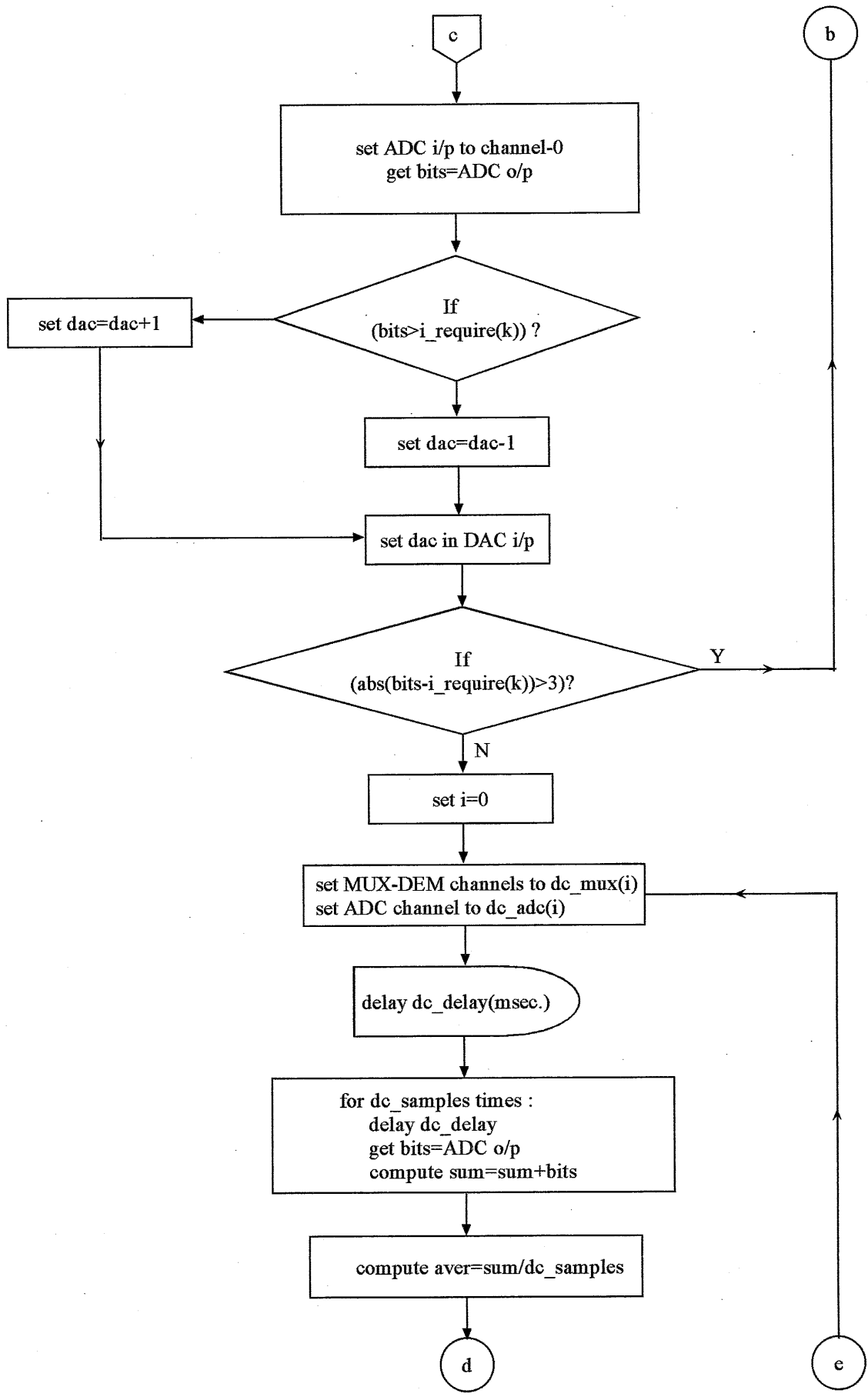
These are stored in the arrays with the following declaration.

```
{ unsigned int dc_mux[] = { 0x11, 0x18, 0x2b, 0x7b };
  unsigned int dc_adc[] = { 0x80, 0x81, 0x84, 0x83 };
  unsigned int i_require[] = { 0xfae, 0xe66, 0xccc, 0x000 };
  ...
}
```

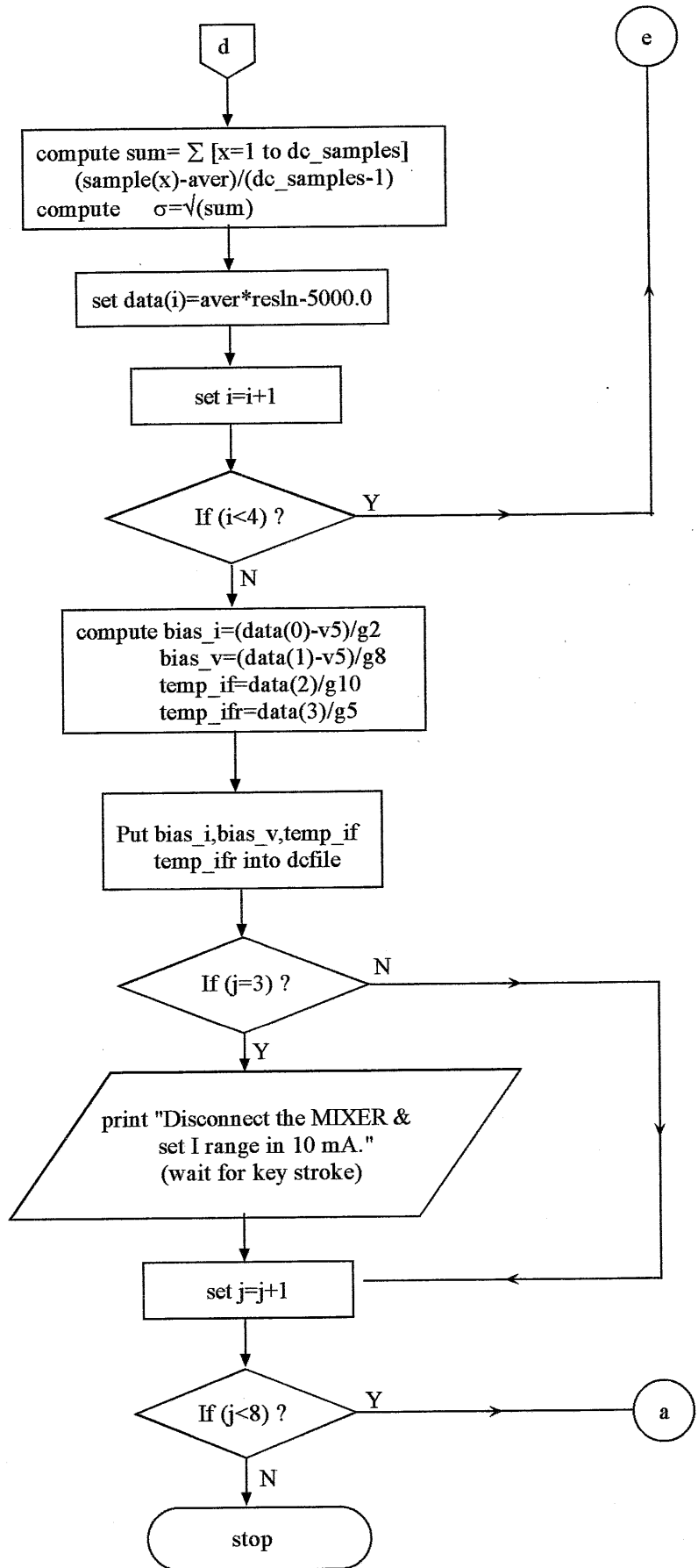
The flow chart of this module is given below.

DC Characteristics





Chapter 4 EVOLUTION



(iii) 'rf-measure' operation :

In the RF measurement part there is automatic voltage/ bias setting. The parameter power is set manually in the intermediate time by the user. The voltage is adjustable through the keypad by the user. After every set of power and bias voltage, the program has to set the channels for starting acquisition of P, V, I, T_h , T_r , and T_c .

These channel settings are constructed as follows.

Signal	set Rl_2 Rl_1 M D	BITS	in Hex	Chl in ADC
P	0 0 2 1	00 001 010	0x0a	0x85
V	0 0 0 3	00 011 000	0x18	0x81
I	0 0 1 2	00 010 001	0x11	0x80
T_h	0 0 3 5	00 101 011	0x2b	0x84
T_r	0 1 3 7	01 111 011	0x7b	0x83
T_c	0 0 3 5	00 101 011	0x2b	0x84

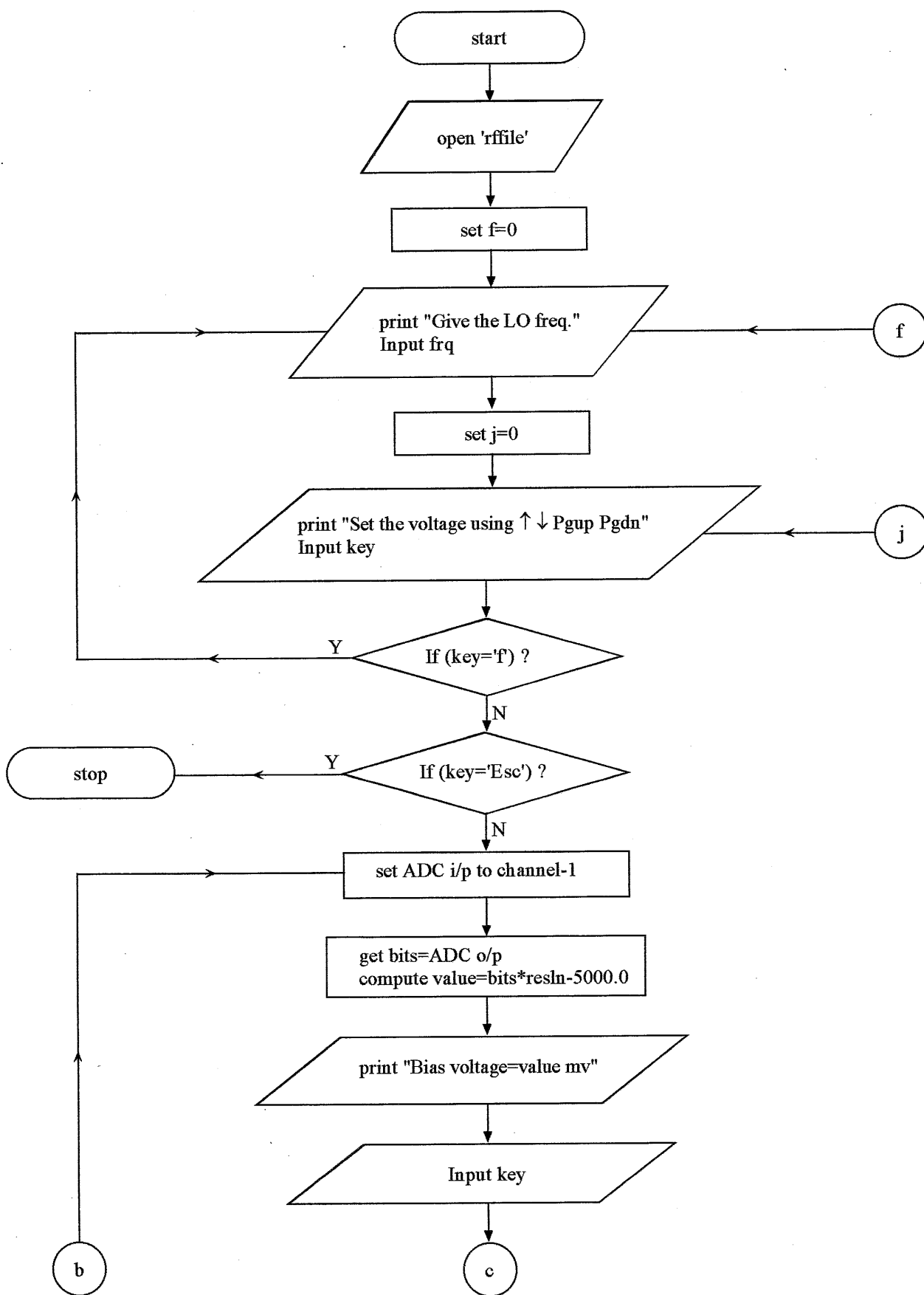
These bit values are stored into the arrays with the following declaration.

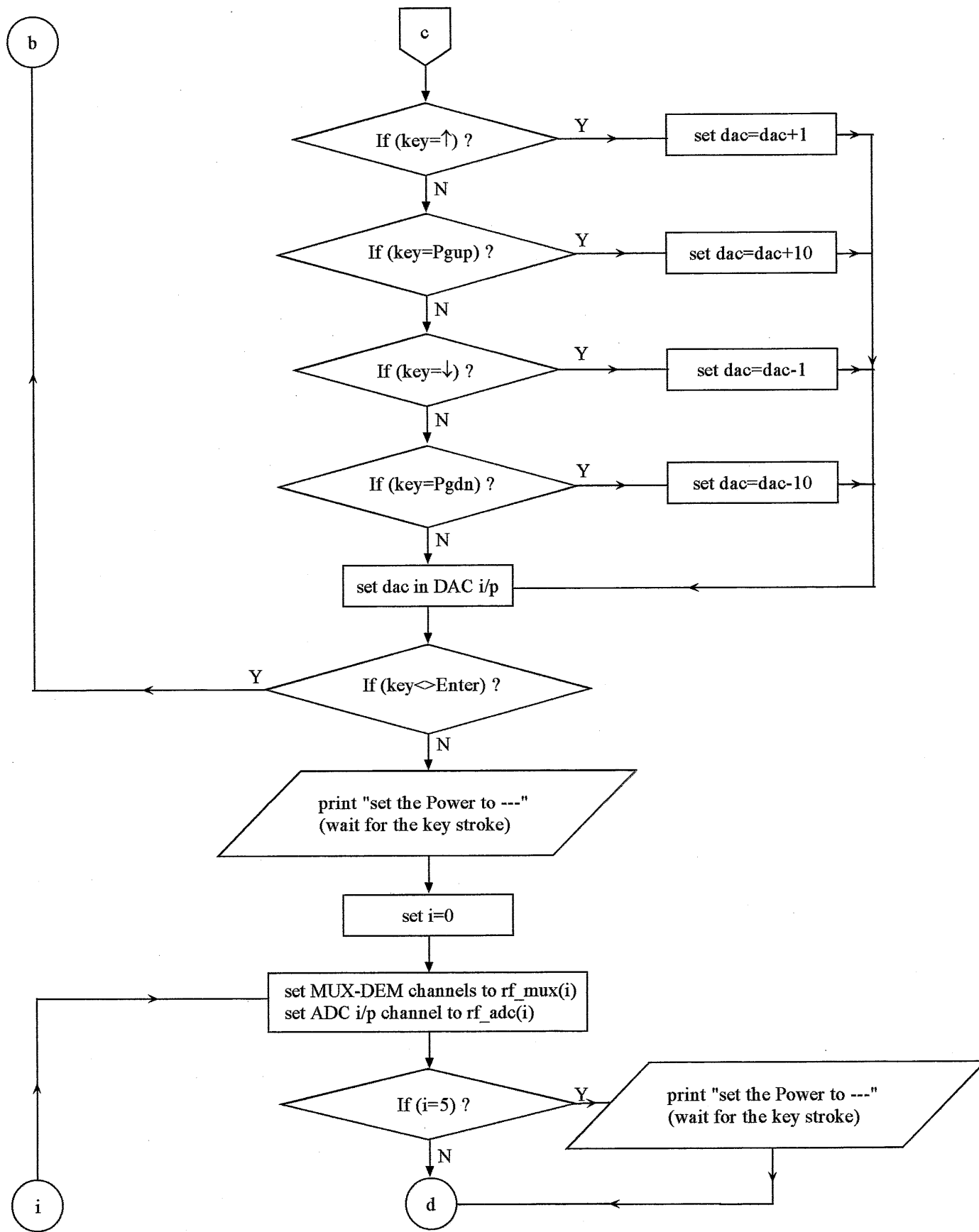
```
{ unsigned int rf_mux[]={ 0x0a, 0x18, 0x11, 0x2b, 0x7b, 0x2b };
  unsigned int rf_adc[]={ 0x85, 0x81, 0x80, 0x84, 0x83, 0x84 };
  ...;
}
```

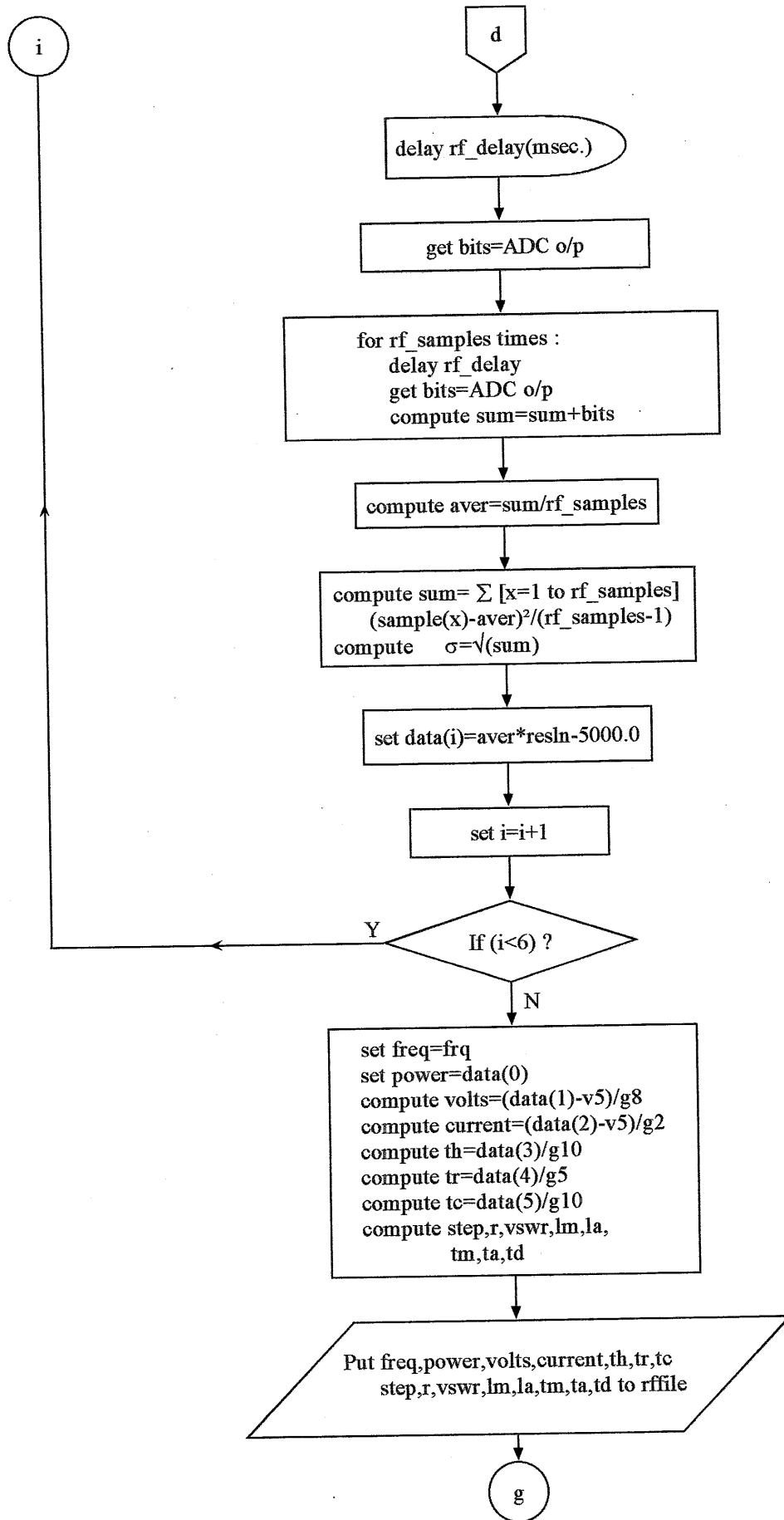
The construction of this module is made with the following flow chart.

First the DAC output has been fed as the signal for the V_{mon} input. The arrow keys and the pgup, pgdn keys are used to set the V_{mon} at the required values such as 0.60 v, 0.65, 0.70 ... etc. For other input lines, the DAC 0 v has been connected as the signal. The acquisition is made for these inputs. The data file storage is displayed on the screen and the format of the data file information is adjusted incrementally.

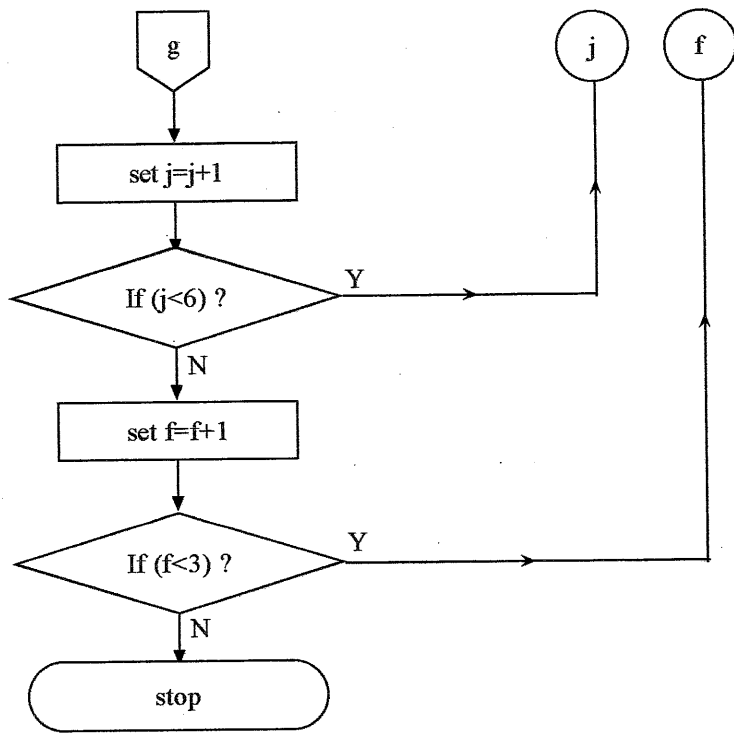
Later the actual signals are fed to the input channels and the V_{mon} is set when the DAC signal conditioned output (gain -1/6) is applied to the bias voltage supply pin. This time all other information such as diode identification, mixer details, amb. temperature, whisker detail ... etc. are added to the data file and these information are to be displayed along with the data tables.







Chapter 4 EVOLUTION



(iv) Date file Report and printing :

At every mode of measurements, the acquired data are stored into a unique data file. The data filename for this is created by using the system clock. To distinguish and to make it unique, the dc_measure acquisition are stored into the files with name 'dcymm.dd' and the rf_measure acquisition are stored with file names 'rfymm.dd'. The yy,mm,dd stands for current year's last two digits, month and day. If more acquisition is made on the same day, the extension can be added up with new numbers.

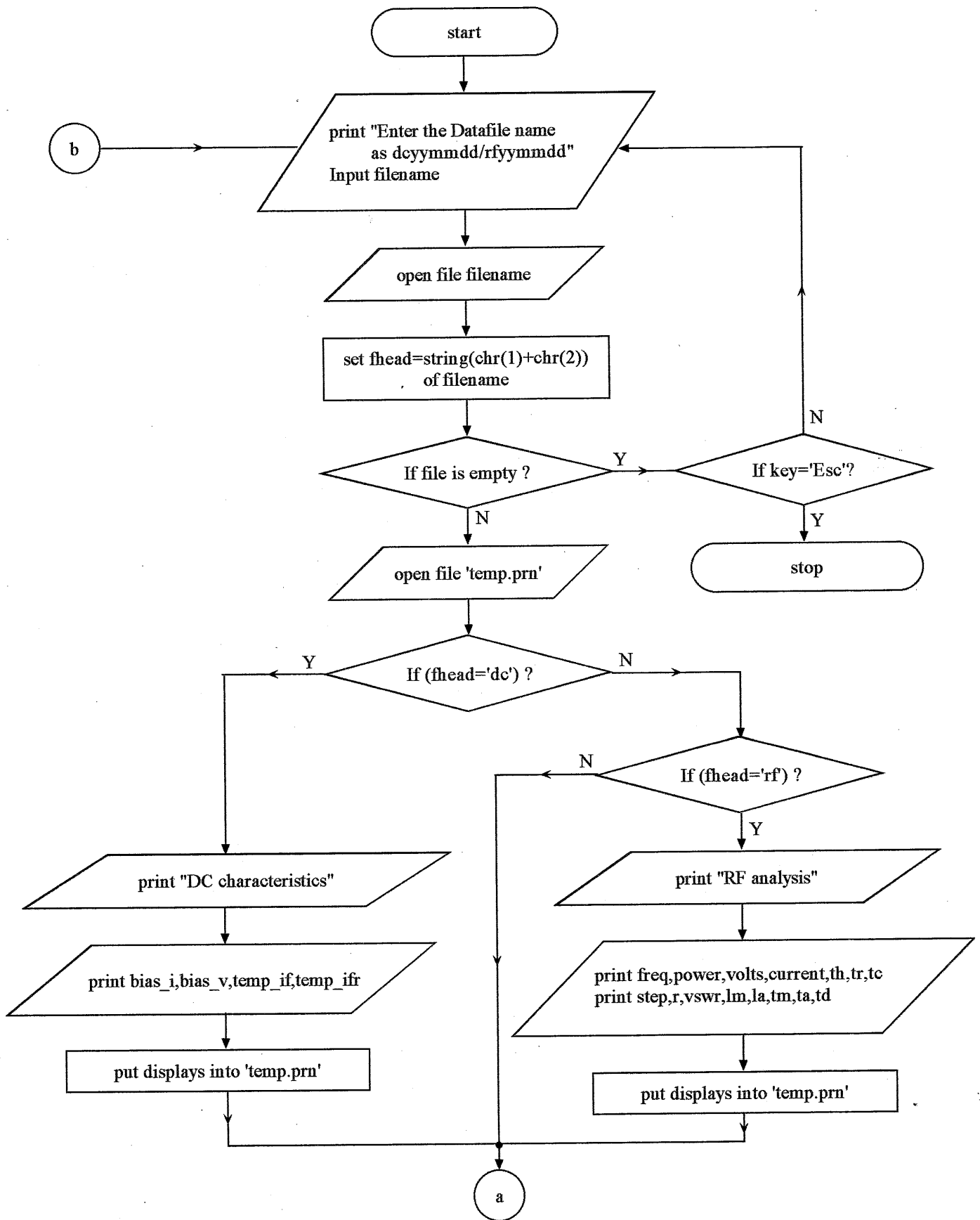
When the stored data files are needed to open for viewing and printing, the files are checked by their file-heads 'dc' and 'rf'. This file-heads are used for generating different formats for the dc-characteristics and rf-analysis parts separately. They have different format and displays for viewing and printing.

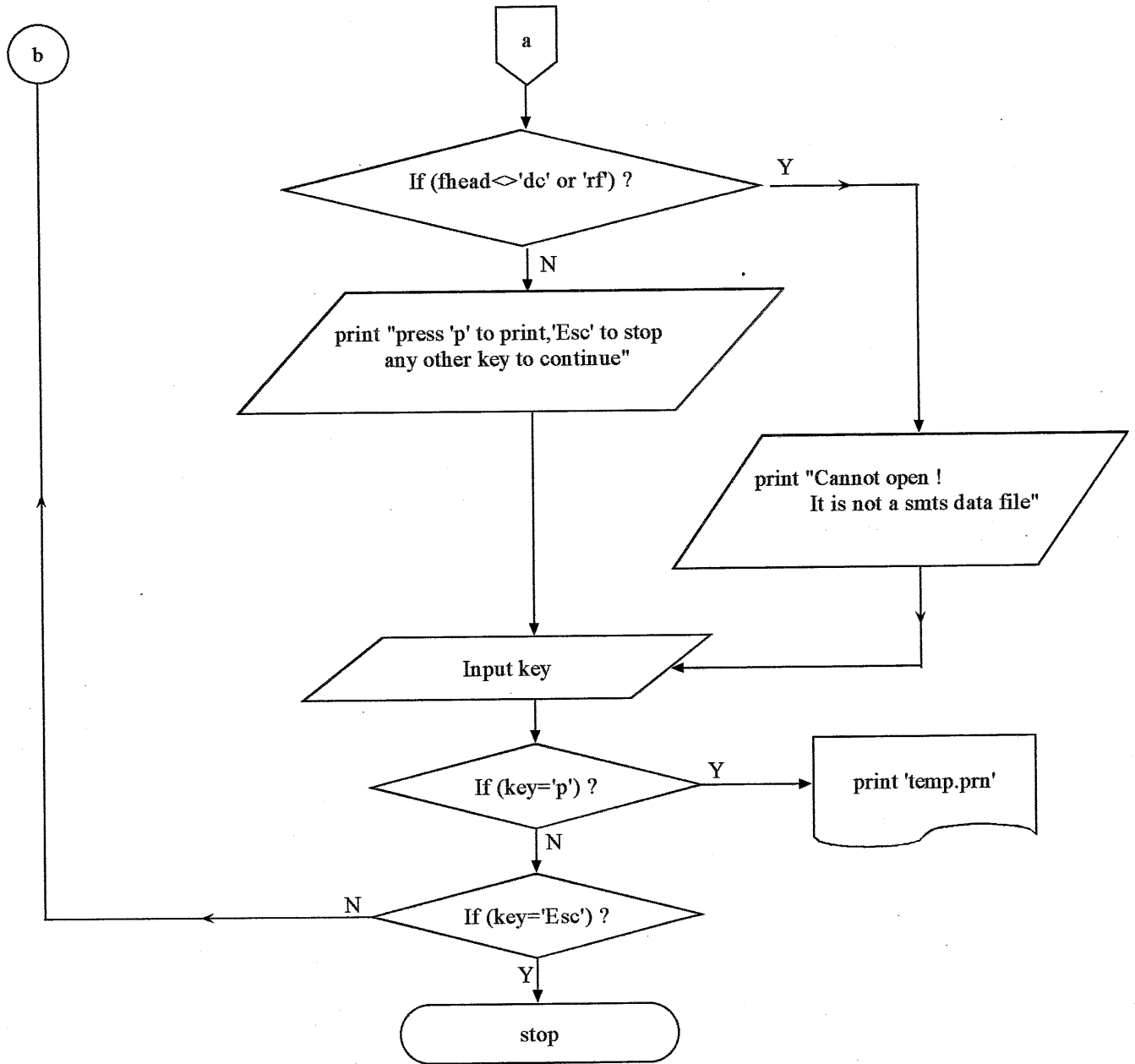
The process of making the hard copy of the data file is performed with the constructs of 'ifstream' and 'ofstream' functions as follows.

```
{ ifstream infile;
  infile.open("temp.prn");
  ofstream outfile;
  outfile.open("PRN");
  ... ;
  while (infile.get(c)!=0) outfile.put(c);
  ... ;
}
```

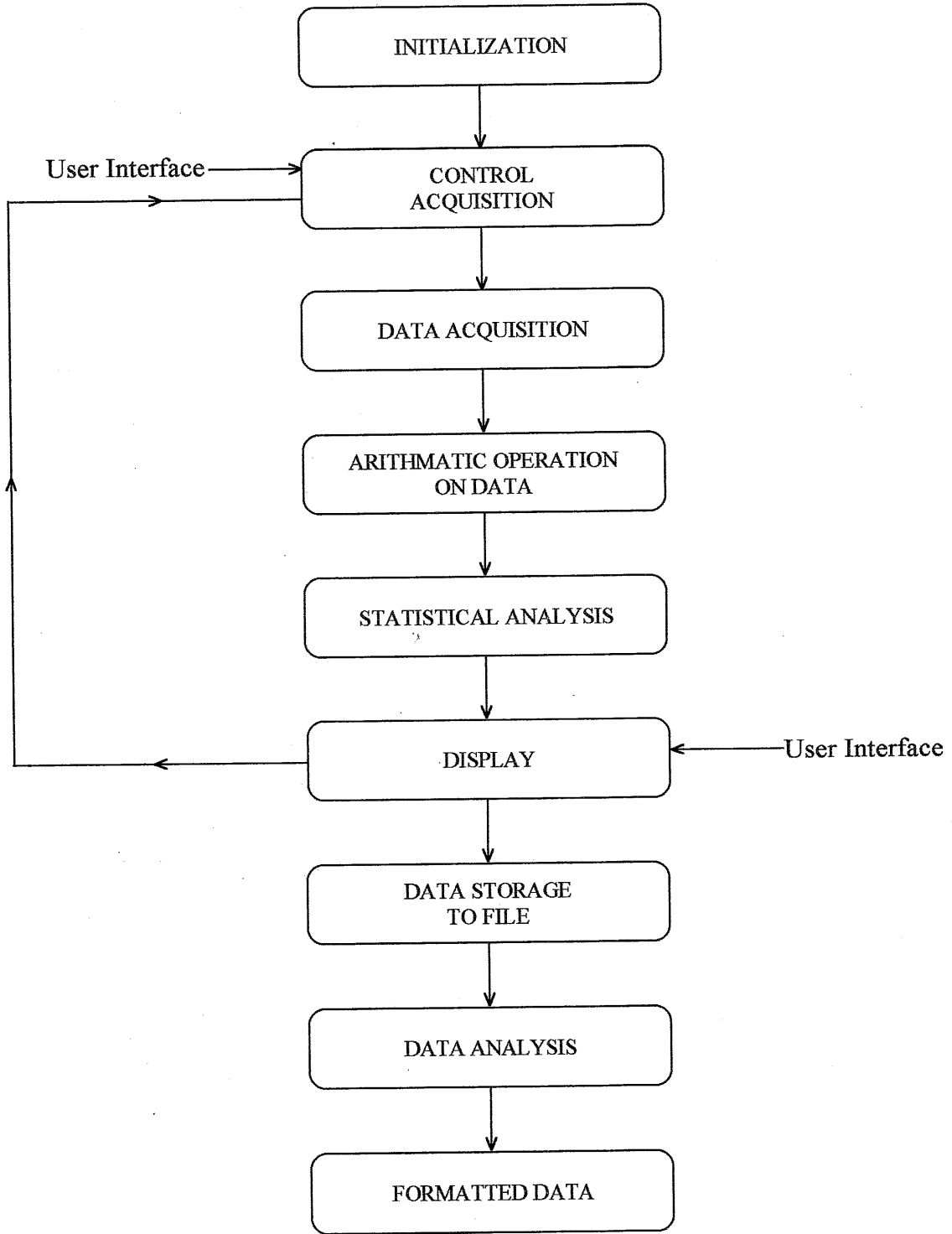
where 'temp.prn' is the file in which the formatted information of the data-file are stored in.

The flow chart of this 'data-file' report generation is as follows.





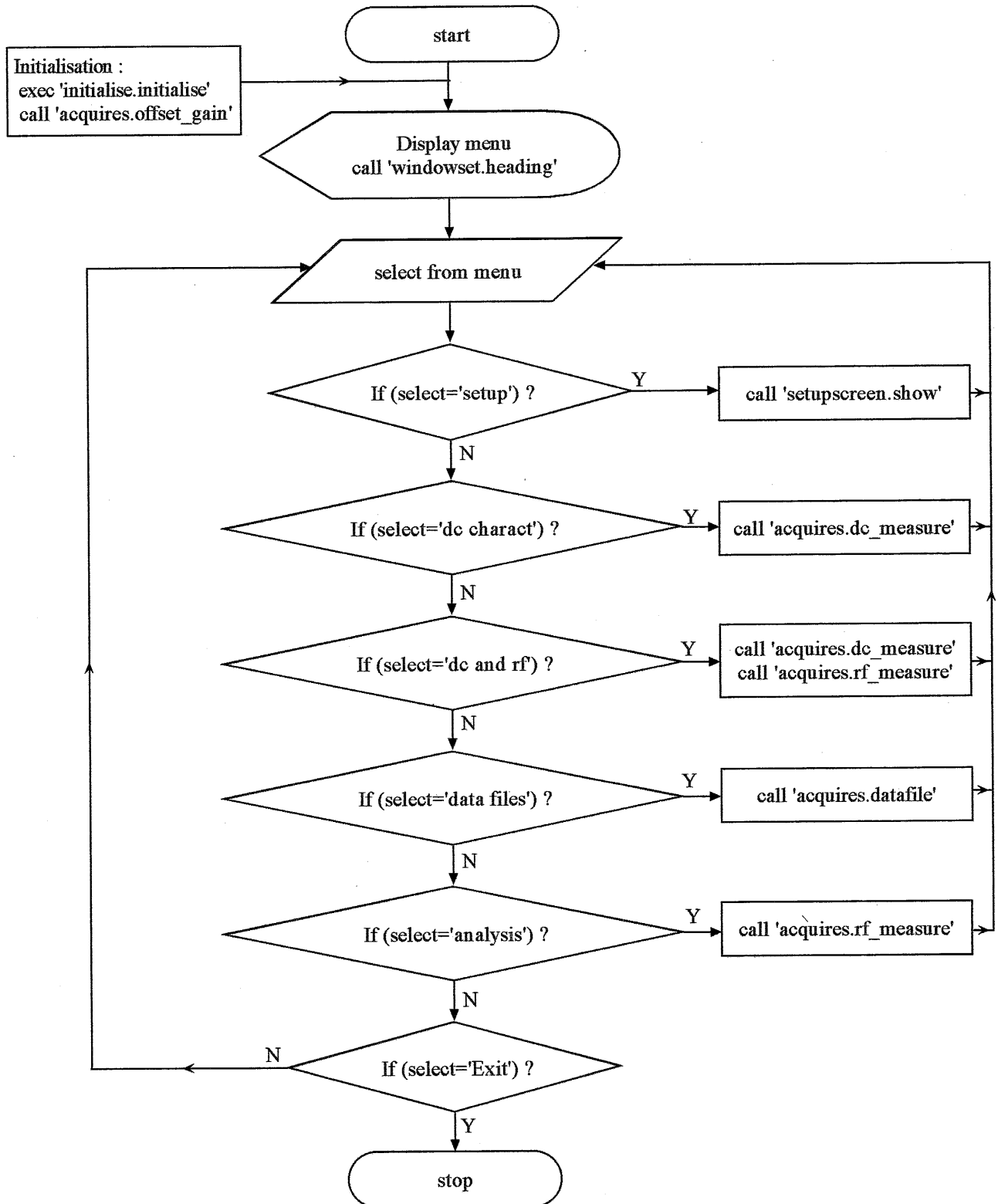
Software :Functional block diagram



SMPS DATA ACQUISITION			20.06.98		
Set up	DC charact	DC and RF	Data files	Analysis	Exit
[Main menu content area]					

Fig. Program main menu

Chapter 4 EVOLUTION Flow chart - main program



CODE

```
/* Program SMTS Data Acquisition - For DC characteristics, RF measurements*/  
/* Sets Bios Current, acquires Current, Voltage, Power, Temperatures  
and controls Noise switch - Performs the Analysis*/
```

```
#include <iostream.h>  
#include <fstream.h>  
#include <stdio.h>  
#include <conio.h>  
#include <string.h>  
#include <ctype.h>  
#include <stdlib.h>  
#include <math.h>  
#include <dos.h>  
#include <bios.h>
```

```
#define GAIN_SAM 10  
#define DC_CMP_SAM 5  
#define DC_SAMPLES 10  
#define DC_DELAY 200 /* Delay in milliseconds */  
#define RF_CMP_SAM 5  
#define RF_SAMPLES 10  
#define RF_DELAY 2 /* Delay in milliseconds */
```

```
#define BA 0x320  
#define ADC_LOWB BA+0x014  
#define ADC_HIGHB BA+0x015  
#define ADC_STATUS BA+0x016  
#define ADC_MODE BA+0x018
```

```
#define DAC_LOWB BA+0x01c  
#define DAC_HIGHB BA+0x01d
```

```
#define DIO_PORTA BA+0x008  
#define DIO_PORTB BA+0x009  
#define DIO_PORTC BA+0x00a  
#define DIO_CONTROL BA+0x00b
```

```
struct dc_data  
{ float bias_i;  
int bias_v;  
int temp_if;  
int temp_ifr;  
};
```

```
struct rf_data
{
    float freq;
    float power;
    float volts;
    float currnt;
    float th;
    float tr;
    float tc;
    int step;
    double r;
    double vswr;
    double lm;
    double la;
    double tm;
    double ta;
    double td;
};

struct rf_analyse
{
    char mixer[7];
    char diode[12];
    char choke[7];
    float wis_len;
    float wis_ht;
    float wis_pro;
    float plate_t;
    float ambient_t;
    float liqn2_t;
};

int off_mux[]={ 0x15,0x1d,0x1d,0x2d,0x3d };
int set_adc[]={ 0x80,0x82,0x81,0x84,0x83 };
int gn_mux[]={ 0x14,0x1c,0x1c,0x2c,0x3c };

unsigned int dc_mux[]={ 0x11,0x18,0x2b,0x7b };
unsigned int dc_adc[]={ 0x80,0x81,0x84,0x83 };
unsigned int i_require[]={ 0xfae,0xe66,0xcc,0x000 };
unsigned int rf_mux[]={ 0x0a,0x18,0x11,0x2b,0x7b,0x2b };
unsigned int rf_adc[]={ 0x85,0x81,0x80,0x84,0x83,0x84 };
float resln =2.441406;
float offsets = 0.0;
float g2=2.0,g4=4.0,g5=5.0,g8=8.0,g10=10.0;
char dcfile[12]="dc";
```

```
char rffile[12]="rf";
date today;
rf_analyse index;
FILE *fp;
```

class initialise

```
{ public : initialise()
    { char year[3],mon[3],day[3];
      unsigned int low,high,count=0x030; /* zero set 0x030 */
      int i,j;
      outport(DIO_CONTROL,0x80);
      low=(count & 0x0ff);
      high=(count & 0xf00);
      high=(high>>8);
      outportb(DAC_HIGHB,high);
      outportb(DAC_LOWB,low);
      struct date d;
      getdate(&d);
      itoa((d.da_year-1900),year,10);
      itoa(d.da_mon,mon,10);
      itoa(d.da_day,day,10);
      strcat(dcfiler,year); strcat(dcfiler,mon);
      strcat(dcfiler,"."); strcat(dcfiler,day);
      strcat(rffiler,year); strcat(rffiler,mon);
      strcat(rffiler,"."); strcat(rffiler,day);
      today = d;
    }
void connectivity()
    { unsigned int lbyte,hbyte,count[]={ 0x000, 0xb34},bits[2];
      for (int i=0;i<2;i++)
      { lbyte=(count[i] & 0x0ff);
        hbyte=(count[i] & 0xf00);
        hbyte=(hbyte>>8);
        outportb(DAC_HIGHB,hbyte);
        outportb(DAC_LOWB,lbyte);
        outport(DIO_PORTB,0x11);
        delay(DC_DELAY);
        outportb(ADC_MODE,0x80);
        delay(DC_DELAY);
        bits[i]=(inpw(ADC_LOWB)&0xffff);
      }
      if ((bits[0]<0xeb8)||bits[1]>0xeff)
      { printf("\a\n Two way communication failure...");
        printf("\n\nThe SMTS may not be in the `AUTO mode");
        printf(" or cable connection is not proper.....");
      }
    }
}
```

```

        getch());};
    }
};

```

class acquires

```

{ private : unsigned int lowb;
        unsigned int highb;
public : void off_gain()
        { float off,value,v5_ref;
          int i,j,k,bits,sample,sum;
          for (i=0;i<0;i++)
            {outport(DIO_PORTB,off_mux[i]);
             delay(DC_DELAY);
             outportb(ADC_MODE,set_adc[i]);
             for (j=0;j<1;j++) { k=0;
              while (((inportb(ADC_STATUS)&0x02)==1) && (k<220))
                { k++;
                 if ((inportb(ADC_STATUS)&0x02)==0) break;}
              if (k==220) printf("<ADC st wrong>");
              else printf("<ADC st OK>");
              delay(DC_DELAY);
              bits=(inport(ADC_LOWB)&0xfff);
              off=bits*resoln-5000;
              bits=0; }
             outport(DIO_PORTB,gn_mux[i]);
             delay(DC_DELAY);
             outportb(ADC_MODE,set_adc[i]); k=0;
             while (((inportb(ADC_STATUS)&0x02)==1) && (k<220))
               { k++;
                if ((inportb(ADC_STATUS)&0x02)==0) break;}
             if (k==220) printf("<ADC st wrong>");
             else printf("<ADC st OK>");
             delay(DC_DELAY);
             bits=(inpw(ADC_LOWB)&0xfff);
             value=bits*resoln-5000;
             switch(i)
             { case 0: g2 =fabs(value-off)/500;
               v5_ref=off; break;
               case 1: g4 =fabs(value-off)/500; break;
               case 2: g8 =fabs(value-off)/500; break;
               case 3: g10=fabs(value-off)/500; break;
               case 4: g5 =fabs(value-off)/500; break;
               default:break;
             }
          }
}

```



```

    }

    void dc_measure()
    { int i,j,k,l,tnr=2100;
      unsigned int bits,dac;
      float sample[200],value,sum,aver,data[4],ff=3;
      float r3[8],ciif[8],r=0.08557,vo,eta,rs,rd,avi=0,bi2=0,ev,evo;
      double dca[8][4],di;
      char c;
      dc_data dc;
      rf_analyse a;

      printf("\nEnter the Mixer details #");
      b=bioskey(0);
      if (b!=283) {
        printf("\n Mixer identification  : "); scanf("%s",a.mixer);
        printf("\n Diode identification  : "); scanf("%s",a.diode);
        printf("\n Plate temperature ( K): "); scanf("%f",&a.plate_t);
        index=a; };
      fp=fopen(dcfiler,"w+");
      fprintf(fp,"%7s %12s %7s %4.3f %4.3f %4.3f %4.3f %4.3f
      %4.3f\n",a.mixer,a.diode,a.choke,a.wis_len,a.wis_ht,a.wis_pro,a.plate_t,a.ambie
      nt_t,a.liqn2_t);
      outport(DIO_PORTB,0x00);
      dac=2; bits=0; lowb=0; highb=0; gotoxy(2,8);
      printf("\n ## Set Mixer I RANGE in 0.1 mA & then Connect the
      MIXER ##");
      getch();
      for (j=0;j<8;j++)
      { k=(j%4);
        if (k==3) continue;
        if (j==4)
          { dac=2; lowb=0;
            highb=0; bits=0;};
        do
        { sum=0.0;
          outport(DIO_PORTB,0x09);
          delay(DC_DELAY);
          outportb(ADC_MODE,0x85);
          delay(DC_DELAY);
          bits=(inport(ADC_LOWB)&0xffff);
          gotoxy(2,10);
          printf("\n Bias current (V) | %.5f Volts count_ADC>
          %x",bits*2.4414-5000,bits);
          printf("\n Difference-----<%4d>",abs(bits-i_require[k]));

```

```

    if (bits>i_require[k]) dac--;
    if (bits<i_require[k]) dac++;
    printf("      count_DAC> %x ",dac);
    lowb = (dac & 0x0ff);
    highb = (dac & 0xf00);
    highb = (highb>>8);
    outportb(DAC_HIGHB,highb);
    outportb(DAC_LOWB,lowb);
    c=getch();
    if (j==4)
        { printf("\n dac=%x, l=%x,h=%x
:bits=%x,%x",dac,lowb,highb,bits,i_require[k]);
          getch(); return; }*/
        } while ((abs(bits-i_require[k])>2)&&(c!='*'));
        printf("**** REACHED %x count ****",i_require[k]);
        c=getch();
        if (c=='*') return;
    for (i=0;i<4;i++)
    { outport(DIO_PORTB,dc_mux[i]);
      outport(ADC_MODE,dc_adc[i]);
      printf("\n channels set for data %d ",i);
      getch();
      delay(DC_DELAY);
      lowb=(inpw(ADC_LOWB)&0xfff);
      for (l=0;l<DC_SAMPLES;l++)
      { delay(2);
        bits =(inpw(ADC_LOWB) & 0xfff);
        sample[l]=(bits*resoln)-5000.0; gotoxy(2,12);
        printf("\n chl%x > %x sample(%2d)= %2f
",dc_adc[i],bits,l+1,sample[l]);
          sum =sum+sample[l];
        };
      aver=sum/DC_SAMPLES; sum=0;
      for (i=0;i<DC_SAMPLES;i++) sum=sum+pow((sample[i]-aver),2);
      sum =sqrt(sum/(DC_SAMPLES-1));
      data[i]=aver;*/
    };
    dc.bias_i =(data[0]-v5)/g2;   dc.bias_v =int(((data[1]-v5)/g8);
    dc.temp_if=int(data[2]/g10);  dc.temp_ifr=int(data[3]/g5);
    for (i=0;i<4;i++) dca[j][i]=data[i];
    if (j!=1)
        { r3[j]=(data[3]-data[2])/tnr;
          ciif[j]=(data[2]-(r3[j]*a.plate_t))/(1.0-r3[j]);};
    fprintf(fp,"%4.3f %4d %4d
%4d\n",dc.bias_i,dc.bias_v,dc.temp_if,dc.temp_ifr);

```



```

rf_analyse a;
fp=fopen(rffile,"w+");
printf("\nEnter the Mixer details #");
b=bioskey(0);
if (b!=283) {
printf("\n Mixer identification  :"); scanf("%s",a.mixer);
printf("\n Diode identification  :"); scanf("%s",a.diode);
printf("\n Choke                    :"); scanf("%s",a.choke);
printf("\n Whisker length   (mm): "); scanf("%f",&a.wis_len);
printf("\n Whisker height   (mm): "); scanf("%f",&a.wis_ht);
printf("\n Whisker protrusion (mm): "); scanf("%f",&a.wis_pro);
printf("\n Plate temperature (øK): "); scanf("%f",&a.plate_t);
printf("\n Ambient temperature(øK): "); scanf("%f",&a.ambient_t);
printf("\n Liquid N2 temp.   (øK): "); scanf("%f",&a.liqn2_t);
index=a; }

for (f=0;f<3;f++)
{ gotoxy(2,2);
printf("Select the LO frequency # ");
printf("\n1-78.1, 2-83.4, 3-92.4, 4-108.5, 5-110.3, 6-112, 7-114
GHz. : ");

scanf("%f",&frq);
data[0]=rfloss[frq-1][0];
outport(DIO_PORTB,0x36);
for (j=0;j<6;j++)
{ gotoxy(2,4);
printf("#### Set the VOLTAGE using %c %c Pgup & Pgdn
Keys ",24,25);
printf("\n\n [ Press `Enter' after setting,");
printf("\n Press `f' to skip to next LO freq., or");
printf("\n Press `Esc' to stop this measurement ]");
b=bioskey(0);
if ((b==8550)||b==8518) break;
if (b==283) return;
do
{ lowb = (dac & 0x0ff);
highb = (dac & 0xf00);
highb = (highb>>8);
outportb(DAC_HIGHB,highb);
outportb(DAC_LOWB,lowb);
outport(DIO_PORTB,0x18);
outport(ADC_MODE,0x81);
delay(5);
bits=(inpw(ADC_LOWB)&0xffff);
value=bits*resoln-5000.0;

```

```

gotoxy(2,10);
printf("\n Bias Voltage - V> %6.4f mV   bits_ADC=%x", (value-
5000)/8, bits);
printf("\n                               bits_DAC=%x", dac);
adjust=bioskey(0);
switch(adjust)
{ case 18432 : dac++;   break;
  case 18688 : dac=dac+10; break;
  case 20480 : dac--;   break;
  case 20736 : dac=dac-10; break;
  default   : break; };
} while (adjust!=7181);
gotoxy(2,9);
printf(" ** Set the required POWER value manually ** ");
getch();
for (i=0; i<6; i++)
{ outport(DIO_PORTB, rf_mux[i]);
  outport(ADC_MODE, rf_adc[i]);
  if (i==5)
  { gotoxy(2,9);
    printf("~~ Show COLD Load and `press space bar' ~~");
    getch();
  };
  delay(2); value=0;
  bits=(inpw(ADC_LOWB)&0xffff);
  for (l=0; l<RF_SAMPLES; l++)
  { delay(RF_DELAY);
    bits  =(inpw(ADC_LOWB) & 0xffff);
    sample[l]=bits*2.441406-5000.0; gotoxy(2,12);
    printf("\n chl%x > %x sample(%2d)= %2f
", rf_adc[i], bits, l+1, sample[l]);
    value  =value+sample[l];
  };
  value=value/RF_SAMPLES;
  for (i=0; i<RF_SAMPLES; i++) sum=sum+pow((sample[i]-
value),2);
  sum =sqrt(sum/(RF_SAMPLES-1));*/
  printf("\n chl set for data %d", i); getch();
  data[i+1]=value;
}
rf.freq =data[0];
rf.power =data[1];   rf.volts=(data[2]-5)/8;
rf.currnt=(data[3]-5)/2;   rf.th=data[4]/10;
rf.tr  =data[5]/5;   rf.tc=data[6]/10;

```

```

for (i=0;i<6;i++) printf("%6.2f ",data[i]);
/* analysis portion */
b3=pow(10,(-a3/10));
b4=pow(10,(-a4/10));
bif=b3*b4;
dt=rf.th-rf.tc;
teq=(a.ambient_t+a.plate_t)/2;
r3=(rf.tr-rf.th)/tnr;
r2=(r3/pow(bif,2.0));
a1=rfloss[frq-1][1];
a2=rfloss[frq-1][2];
b1=pow(10,(-a1/10));
b2=pow(10,(-a2/10));
brf=b1*b2;
/* dtcal is step at the mixer input (RF loss accounted for) */
dtcal=brf*(a.ambient_t-a.liqn2_t);
/* rlm is the mixer conv loss corrected for IF loss & not for IF mismatch */
rlm=2*dtcal*bif/dt;
dlm=10*log10(rlm);
/* rla is available mix conv loss corrected for IF mismatch also */
rla=rlm*(1-r2);
dla=10*log10(rla);
/* to is DSB Tmix corrected for RF, IF and IF mismatch */
to=(rf.th-((r3*a.ambient_t)+((1-bif)*r2*bif*teq)+((1-bif)*teq)))*dtcal/dt-
((a.ambient_t*brf)+(b2*(1-b1)*a.ambient_t)+((1-b2)*teq));
/* tm is same but SSB and new flexible formula */
tm=(rf.th-((r3*(a.ambient_t)+(b3*bif*r2*(1-b4)*teq)+(bif*r2*(1-
b3)*a.plate_t)+(b4*(1-b3)*a.plate_t)+((1-b4)*teq)))*2*dtcal/dt-
2*((a.ambient_t*brf)+(b2*(1-b1)*a.ambient_t)+((1-b2)*teq));
/* tm1 is to be corrected for mistakes */
tm1=(rf.th-bif*((1-b4)*(1+(1/bif))*teq+((1-
b3)*(r3+b4/bif)*a.plate_t)+(r2*bif*a.ambient_t)))*rla/(bif*(1-r2))-
(2*(a.ambient_t*b2+(1-b2)*teq));
/* tmo is to be corrected for mistakes */
tmo=(rf.th-bif*((1-b4)*(1+1/bif)*teq+a.plate_t*(1-b3)/b3))*(rlm/bif)-
(2*(a.ambient_t*b2+(1-b2)*teq));
/* tmr is calculated assuming perfect IF match and rlm */
tmr=(rf.th-((b4*(1-b3)*a.plate_t)+((1-b4)*teq)))*2*dtcal/(dt*(1-r2))-
2*((a.ambient_t*brf)+(b2*(1-b1)*a.ambient_t)+((1-b2)*teq));
td =tm/(rla-2);
av =(1+sqrt(r3))/(1-sqrt(r3));
rf.step=dt; rf.r =r3;
rf.vswr=av; rf.lm=dla;
rf.la =dlm; rf.tm=tm;
rf.ta =tmr; rf.td=td;

```

```

printf("Data
:step=%4.1f,r=%5.3f,vswr=%5.3,lm=%5.3f,la=%5.3f,Tm=%4d,Ta=%4d,Td=%4d",
rf.step,rf.r,rf.vswr,rf.lm,rf.la,int(rf.tm),int(rf.ta),int(rf.td));
getch();
/*fprintf(fp,"%5.2f %4.2f %4.2f %4.2f %4d %4d %4d %3d %5.3f %5.3f %4.2f
%4.2f %4d %4d
%4d\n",rf.freq,rf.power,rf.volts,rf.currnt,rf.th,rf.tr,rf.tc,rf.step,rf.r,rf.vswr,rf.lm,rf.la,in
t(rf.tm),int(rf.ta),int(rf.td));
    output(DIO_PORTB,0x00);
    getch();*/
    };
};
fclose(fp);
}

int datafile()
{ dc_data dc;
  rf_data rf;
  char filename[12],fhead[3],c;
  unsigned int key;
  FILE *tm;
  do {
    *filename='\0'; *fhead='\0';
    printf("\n\n Enter the DataFile name (dcyymm.dd/rfyymm.dd) :
");
    scanf("%s",filename);
    if ((fp=fopen(filename,"r"))==NULL)
        { printf(" No such file exist !          Press 'Esc' to stop,any
other key to continue ");
          key=bioskey(0); }
    else
        { strcat(fhead,filename,2);
          tm=fopen("temp.prn","w+");
          fhead[0]=tolower(fhead[0]); fhead[1]=tolower(fhead[1]);
          if (strcmp(fhead,"dc")==0)
              {printf("\n%c DC CHARACTERISTICS %c      %c %s
%c\n",178,178,17,filename,16);
                fprintf(tm,"\n  DC CHARACTERISTICS File : %s
\n\n",filename);

printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n");
          fprintf(tm,"-----\n");
          printf(" Bias I  V(mV)  T_if K  T_ifR K\n");
          fprintf(tm," Bias I  V(mV)  T_if K  T_ifR K\n");

```

```

printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n");
        fprintf(tm,"-----\n");
        while (!feof(fp))
        { fscanf(fp,"%f %d %d
%d\n",&dc.bias_i,&dc.bias_v,&dc.temp_if,&dc.temp_ifr);
          printf(" %8.3f %4d %4d
%4d\n",dc.bias_i,dc.bias_v,dc.temp_if,dc.temp_ifr);
          fprintf(tm," %8.3f %4d %4d
%4d\n",dc.bias_i,dc.bias_v,dc.temp_if,dc.temp_ifr);
        }

printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
        fprintf(tm,"-----\n");
        fcloseall(); };
        if (strcmp(fhead,"rf")==0)
{fp=fopen(filename,"r");
  printf("\n%c RF ANALYSIS %c %c %s %c\n",176,176,17,filename,16);
  fprintf(tm,"\n\n RF ANALYSIS File : %s \n\n",filename);

printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n");
  fprintf(tm,"-----\n");
  printf(" P V(mV) I Th Tr Tc/Tn step R VSWR LM LA TM TA
TD\n");
  fprintf(tm," P V(mV) I Th Tr Tc/Tn step R VSWR LM LA TM
TA TD\n");

printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA\n");
  fprintf(tm,"-----\n");
  while (!feof(fp))
  {fscanf(fp,"%f %f %f %f %d %d %d %d %f %f %f %f %d %d
%d\n",&rf.freq,&rf.power,&rf.volts,&rf.currnt,&rf.th,&rf.tr,&rf.tc,&rf.step,&rf.r,&rf.vswr,&rf.lm,&rf.la,&rf.ta,&rf.td);
    printf("%5.2f %4.2f %4.2f %4.2f %4d %4d %4d %3d %5.3f %5.3f %4.2f %4.2f
%4d %4d
%4d\n",rf.freq,rf.power,rf.volts,rf.currnt,rf.th,rf.tr,rf.tc,rf.step,rf.r,rf.vswr,rf.lm,rf.la,rf
.tm,rf.ta,rf.td);
    fprintf(tm,"\n%5.2f %4.2f %4.2f %4.2f %4d %4d %4d %3d %5.3f %5.3f %4.2f
%4.2f %4d %4d
%4d\n",rf.freq,rf.power,rf.volts,rf.currnt,rf.th,rf.tr,rf.tc,rf.step,rf.r,rf.vswr,rf.lm,rf.la,rf
.tm,rf.ta,rf.td);
  }
}

```



```

printf("AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA
AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
fprintf(tm, "\n-----\n");
fcloseall(); };
if ((strcmp(fhead, "dc")!=0)&&(strcmp(fhead, "rf")!=0))
    { printf("\n Cannot Open. It is not a SMTS DATA ACQUISITION file ! Press
'Esc'to stop");
      key=bioskey(0);}
else
    {printf("\n ppp Press 'P' for printer, 'Esc' to stop, any other keys to continue ");
      key=bioskey(0);
      if ((key==6512)||((key==6480))
          {ifstream infile;
            infile.open("temp.prn");
            ofstream outfile;
            outfile.open("PRN");
            printf("..... PRINTING");
            while (infile.get(c) != 0) outfile.put(c); };
          };
        } while (key!=283);
      return (1);
    }
};

```

class setupscreen

```

{ public : void show()
    { char c;
      fp=fopen("setup.doc", "r");
      while ((c=fgetc(fp))!=EOF) putchar(c);
      fclose(fp);
    }
};

```

class windowset

```

{ public : void heading()
    { window(1, 1, 80, 2);
      textbackground(LIGHTCYAN); textcolor(WHITE); clrscr();
      printf("          SMTS DATA ACQUISITION");
      printf("
%d.%d.%d\n", today.da_day, today.da_mon, (today.da_year-1900));
    }
    void setting(int l, int t, int r, int b, int back, int text)
    { window(l, t, r, b);

```

```

        textbackground(back); textcolor(text);
        clrscr();
    }
    void normal()
    { window(1,1,80,25);
      textbackground(BLACK); textcolor(WHITE);
      clrscr();
    }
};

```

main()

```

{ char opt,key,buffer1[480],buffer2[2516];
  int x=2,y=2;
  unsigned int b;
  initialise init;
  acquires acquire;
  setupscreen setup;
  clrscr();
  acquire.off_gain();
  windowset windows;

  windows.heading();
  windows.setting(1,2,80,2,7,0);
  printf(" Set Up    DC charact  DC and RF  Data Files  Analysis  Exit");
  windows.setting(1,3,80,25,11,7);
  for(int i=1;i<=80;i++) printf("%c",205);
  do { y=x;
      windows.setting(y,2,y+12,2,11,0);
      switch(y)
      { case 2 :printf(" Set Up  "); break;
        case 15:printf(" DC charact "); break;
        case 28:printf(" DC and RF "); break;
        case 41:printf(" Data Files "); break;
        case 54:printf(" Analysis "); break;
        case 67:printf(" Exit  "); break;
        default:break;}
      b=bioskey(0);
      switch(b)
      { case 19200 : if (x==2) x=67;
                    else x-=13; break;
        case 19712 : if (x==67) x=2;
                    else x+=13; break;
        default : break;}
      if (b==7181) windows.setting(1,4,80,25,11,7);
      else windows.setting(y,2,y+12,2,7,0);

```

```
switch(y)
{ case 2 : if (b==7181) { gettext(4,6,77,22,buffer2);
                    windows.setting(4,6,77,22,7,0);
                    setup.show(); getch();
                    puttext(4,6,77,22,buffer2);
                    windows.setting(1,4,80,25,11,7);
                    break;}
      else printf(" Set Up "); break;
  case 15: if (b==7181) { init.connectivity();
                    acquire.dc_measure(); break;}
      else printf(" DC charact "); break;
  case 28: if (b==7181) { init.connectivity();
                    acquire.rf_measure(); break;}
      else printf(" DC and RF "); break;
  case 41: if (b==7181) { gettext(1,1,80,3,buffer1);
                    windows.normal();
                    acquire.datafile(); clrscr();
                    puttext(1,1,80,3,buffer1);
                    windows.setting(1,4,80,25,11,7);
                    break; }
      else printf(" Data Files "); break;
  case 54: if (b==7181) { printf("\n\n\n\t analysis ***** "); break;}
      else printf(" Analysis "); break;
  case 67: if (b==7181) { windows.normal();
                    exit(2); }
      else printf(" Exit "); break;
  default: break;}
} while (1);
return 0;
}
```

SMTS DATA ACQUISITION						20.06.08
Set up	DC charact	DC and RF	Data files	Analysis	Exit	

Fig. Program main menu display

Chapter 5 CONCLUSION

The software for the mixer characterisation - Data acquisition and Analysis has been developed with the construction of interfacing hardware. The software makes the required access to the hardware devices. It acquires the different signals from the Mixer Test System and performs all analysis part which is required for the mixer characterisation.

The interfacing hardware are tested and tuned to the required performance. The on-line testing has been performed with a sample mixer. The interfacing hardware has improved the resolution factor for each measurements on the Mixer Test System in using the ALS-PC Add On card.

The overall system is integrated and the tuning of the system is going on. This requires several steps to reach the required performance.

With the application of some electronic instruments, the system can be easily upgraded for complete automation. After this the entire measurement and analysis process would be completely automatic.

Chapter 6 BIBLIOGRAPHY

1. "Object-Oriented Programming in Turbo C++"
by Robert Lafore - Galgotia publications Pvt. Ltd.
2. "Object-Oriented Analysis and Design with Applications"
by Grady Booch - The Benjamin/Cummings Publishing Company, Inc.
3. "Micro computer Interfacing Handbook A/D & D/A"
by Joseph J. Carr - TAB Books Inc.
4. "Analog-Digital and Digital-Analog Conversion"
by Bernard Loriferne - Heyden & Son Ltd., London.
5. "Analog-to-Digital / Digital-to-Analog Conversion Techniques"
by David F. Hoeschele, Jr. - John Wiley & Sons, Inc., New York.
6. "Analog-Digital Conversion Handbook"
by The Engg. Staff of Analog Devices, Inc. - Prentice Hall.
7. "LINEAR Data Book"
by National Semiconductor Corporation.
8. "INTEL Peripheral Design Handbook - 1980"
by Intel corporation.
9. "MOTOROLA Master Selection Guide and Catalog"
by Motorola Semiconductor Products Inc.
10. "INTERFACE IC - Integrated Circuit D.A.T.A. Book"
by Electronic Information Series, D.A.T.A. Inc., New Jersey.
11. "TTL Logic Data Manual 1982"
by Signetics.

