DEVELOPMENT OF A REAL-TIME AUTOMATIC NETWORK ANALYZER MEASUREMENT SYSTEM

Prepared By : James John Murray
BSc. Electrical and
Electronic Engineering
Cape Town.

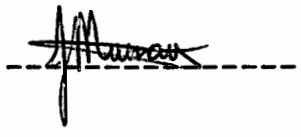
Prepared For : The Department of Electrical
and Electronic Engineering at
the University of Cape Town.

30th September, 1989.

Thesis submitted in partial fulfilment of the requirements for the degree of Master of Science in Engineering.

DECLARATION

I declare that this thesis is my own unaided work, and is being submitted to the University of Cape Town in partial fulfilment of the requirements of a Masters Degree in Engineering. It has not been submitted before to any other university for any degree or examination.



J.J.Murray 30th September 1989.

ACKNOWLEDGEMENTS

The author wishes to thank the following people for the assistance rendered during the preparation of this thesis:

Dr. R.M. Braun (Thesis Supervisor);

Mrs. M. Murray for assisting with much of the typing of this thesis.

Mr. M.V. Murray for the photography which made the thesis complete.

Mr. P. Blain for the laser printing of the thesis.

Mr. J. Da Silva for the proof reading of the thesis.

My parents and the rest of my family and friends for their support during the two years that the thesis was in progress.

ABSTRACT

This thesis concerns itself with the development of a real-time automatic Network Analyzer measurement system, based on Hewlett Packard's manual HP8410C.

The major limitation in non real-time systems is the time required to perform a measurement. Real-time systems have greater measurement speed than their non real-time counterparts, but are also generally less accurate.

The main objectives of the thesis are to survey literature on high frequency measurements, to develop hardware and software for a real-time Analyzer and to perform tests with the system.

The literature on high frequency networks related to Network Analyzers is surveyed and the relevant findings are presented as the basis for the development of a real-time system.

The hardware for the system is developed around the manual HP8410C Network Analyzer. New modules that are added to the HP8410C are an IBM compatible computer and an analogue preprocessing unit.

Software for the system is developed using the TURBO PASCAL compiler (Version 5.00). the software provides new features to the system. These features include:

(1) Instantaneous switching between various real-time display formats. The formats include a Smith chart and rectilinear displays.

(2) Vector Error corrected measurements in real-time.

(3) Hardcopies of results.

(4) Writing of measurement data to text files.

The performance of the system is evaluated by means of using the system to perform several basic measurements.

The main conclusions drawn are:

(1) Real-time measurements are only feasible for single power band sweeps.

(2) Full 12-term error correction in real-time is not feasible.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

NOMENCLATURE

*	S_{11A}	The **actual** scattering-parameter for forward reflection.
*	S_{22A}	The **actual** scattering-parameter for reverse reflection.
*	S_{21A}	The **actual** scattering-parameter for forward transmission.
*	S_{12A}	The **actual** scattering-parameter for reverse transmission.
*	S_{nnM}	The **measured** values for the above scattering-parameters.
*	Thru	A straight connection between the two test ports of a Network Analyzer. Usually a microwave coaxial cable.

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND INFORMATION

Characterizing the behaviour of linear networks that will be stimulated by arbitrary signals and interfaced with a variety of other networks is a fundamental problem in both synthesis and test processes.

Network Analysis offers a solution to these problems through a complete description of linear network behaviour in the frequency and/or time domain.
Network Analysis is thus the only effective means of completely characterizing a linear Network's behaviour at microwave frequencies.

The instruments that perform these analyses are known as **NETWORK ANALYZERS** and they have the unique capability of measuring both the magnitude and phase characteristics of a device under test (DUT).

This thesis deals with the development of a real-time automatic vector Network Analyzer measurement system based on a manual Network Analyzer, Hewlett Packard's HP8410C.

Real-time automatic network analysis is possible using advanced instruments such as Hewlett Packard's HP8510B network analyzer. However, the measurement requirements of certain institutions such as the University of Cape Town, do not justify the extremely high cost of the HP8510B.

1.1.1 Real-Time versus Non Real-Time Automatic Analyzers

Several automatic Network Analyzer systems have previously been developed based on the HP8410C [1.1, 1.2, 1.3].

The fundamental difference between these systems and the system described in this thesis, is the fact that this system operates in a **REAL-TIME** manner.
Real-time implies that the measurement system must be fast enough such that a DUT can be adjusted for optional performance in a matter of minutes over the entire frequency range of interest.

The major limitation in non real-time systems is the time required to perform a measurement. Calibrating the system for full error correction and subsequently performing a single measurement on a DUT over a specific frequency band can be extremely time consuming.
This time delay factor associated with non real-time systems poses a restriction on the type of measurement that can be performed with these measurement systems.

For example, a tuneable notch filter may require fine tuning in order to obtain a match at a specific frequency or a narrow frequency band. This measurement requires continuous adjustment of the filter over a period of time until appropriate results are obtained. A non real-time system could not effectively perform this measurement.

Unfortunately there is a price to pay for implementing a real-time system. The main trade-off for real-time measurements is the accuracy of the measurements.

Non real-time systems generally achieve greater measurement accuracy than their real-time counterparts because of two main reasons.

Firstly, non real-time systems do not have time restrictions on making measurements. These systems tune in to each measurement frequency in turn, and digitize the corresponding magnitude and phase data in their own time. In fact, some systems [1.1, 1.3] use a process of adaptive sampling whereby more samples are taken for low level measurements than for high-level measurements. These samples are then averaged to yield more accurate and meaningful measurement results.

Real-time systems on the other hand, **do** have time restrictions on making measurements since they are required to sample sweeping waveforms in real-time. An adaptive form of sampling is clearly not feasible for real-time systems.

The second reason why non real-time systems are generally more accurate than real-time systems is that the former use the 180 degrees **phase offset** capability of the HP8410C to enhance the accuracy of the phase measurement data. This phase offset capability enables the Network Analyzer to add 180 degrees of phase shift to the phase data.

Phase is displayed in a ±180 degrees format on the HP8410C. The sharp transitions from +180 degrees to -180 degrees phase shift can lead to measurement ambiguities when sampling the phase data. The 180 degrees phase offset is used around these ±180 degrees transition points, and when the phase data is interpreted later, 180 degrees phase shift is subtracted from the relevant data points.

In this way, ambiguities inherent in the sampling of phase data at the transitions between +180 degrees and -180 degrees are removed.

Note that real-time systems are generally less accurate than non real-time systems, but real-time **automatic** systems can be considerably more accurate than manual systems. The reason for this is that automatic systems are able to implement **vector accuracy enhancement** techniques, a full discussion of which is to be found in Chapter 2.

Another disadvantage in using a real-time system is the memory requirement of the computer. The graphic images of the various display formats, (eg Smith Chart) need to be stored in memory to facilitate fast retrieval for plotting and refreshing of the screen. Each one of these screens requires 64 kilobytes of memory storage. This storage is not required in non real-time systems.

Lastly a real-time system has a far more complicated data acquisition requirement than that of a non real-time system. The real-time data acquisition scheme implemented in the system described in this thesis is discussed in detail in chapter 4.

1.1.2 Rationale for Automation of the System

There are three main reasons for automating a Network Analyzer:

(1) **Allows easy interpretation of results.**
In automatic systems, plots of magnitude and phase are displayed in an absolute as opposed to a relative format. The plots available on the HP8410C manual

Analyzer require the user to remember the reference values in order to interpret any given measurement. In automatic systems, the vertical and horizontal axes are scaled in absolute values, the reference values having been automatically incorporated.

(2) **Minimizes the risk of operator errors.**
An automatic Network Analyzer will configure the system automatically, disabling manual control as far as possible. In manual systems, very often a measurement is invalidated because manual system configuration settings are incorrectly set or altered during a measurement.

(3) **Allows vector error corrected measurements.**
The characterization of the systematic errors inherent in the measurement system and the subsequent removal of these errors is possible by the use of a mathematical error model of the measurement system. Removal of these systematic errors from measurements can lead to large accuracy improvements.

Automatic Network Analyzers have the ability to provide full error correction to microwave measurements, making these systems more versatile and more accurate than their manual counterparts.

1.2 OBJECTIVES OF THE THESIS

The objectives of this thesis are:

1. To survey literature on Vector Measurements of High Frequency Networks, and Network Analyzers in particular.
2. To develop the hardware for the implementation of a real-time automatic vector Network Analyzer system. The existing HP8410C manual vector Network Analyzer system in the microwave laboratory at the University of Cape Town is to be used as the basis for the new system.
3. To develop software for the system, thereby enhancing the versatility of the Network Analyzer. In particular, the software should provide the system with the following new features:

(a) The ability to switch **instantaneously** between various real-time display formats. The display formats should include a Smith Chart display, a rectilinear display of magnitude versus frequency, a rectilinear display of phase shift versus frequency, a simultaneous rectilinear display of both magnitude and phase shift versus frequency and a list of values.

(b) Vector error corrected measurements in real-time in the form of full eight-term correction for reflection measurements and frequency-response-only correction for transmission measurements.

(c) A facility for obtaining **hardcopies** of results from the system. These **hardcopies** should include rectilinear displays of magnitude and phase shift versus frequency, Smith Charts and listings.

(d) A facility to enable the writing of measurement data to a file.

(e) The ability to program the start and stop frequencies and the RF level on the sweep oscillator from the computer.

(f) The ability to program the s-parameter and incident attenuation on the test set from the computer.

(g) Features (a) to (f) should be provided in an extensive menu driven and user friendly format. This will enable computer illiterate users to use the package easily.

4. The implementation of twelve-term error correction for two port Networks in real-time should be investigated.

5. To perform measurements with the system, thereby verifying its ability to measure automatically in real-time.

6. To draw conclusions about the system.

1.3 SCOPE AND DEVELOPMENT OF THE THESIS

The thesis deals with the theoretical background behind high frequency vector measurements, the hardware and software involved in the implementation of a real-time automatic Network Analyzer and the test results obtained using the system.

The theoretical component of the thesis, found in Chapter 2, deals with the fundamentals of microwave measurements, the configuration of a Network Analyzer and accuracy enhancement principles.

The Chapter begins with a discussion of microwave measurement fundamentals which encompasses basic transmission and reflection measurements, the reasons why phase information is important and scattering parameters (s-Parameters).

This is followed by a discussion of the configuration of a Network Analyzer which involves the various components that constitute a Network Analyzer measurement system.

Chapter 2 concludes with a discussion of accuracy enhancement principles. This section deals with vector error correction techniques for Network Analyzer measurement systems. The different types of errors inherent in these measurement systems are discussed and are combined to yield error models for the system. The eight-term and twelve-term error models for reflection and transmission measurements are dealt with in detail, and solutions to these models are presented.

Chapter 3 presents the hardware involved in the system. This incorporates a discussion of the manual HP8410C Network Analyzer and the modules which are added to the system to facilitate automation of the system.

Chapter 4 deals with the system software which compliments the hardware discussed in Chapter 3.
The Chapter begins with a detailed discussion of the real-time data acquisition routine used. This is followed by a discussion of an alternative data acquisition scheme. The conversion of relative measurements to absolute measurements is then described. This is followed by a discussion on using the HPIB bus with Turbo Pascal. The

implementation of 12-term error correction in real-time is then described and the Chapter ends with a description of the main program.

Chapter 5 presents the results obtained from test measurements performed with the real-time Network Analyzer system.

The thesis ends with conclusions which are drawn from its major findings.

REFERENCES

[1.1] Automating the HP8410B Microwave Network Analyzer, Application Note 221A, June 1980, Hewlett Packard.

[1.2] Williams, W.L.,Compton, R.C., Rutledge, D.B., "Computer Automation and Error Correction for a Microwave Network Analyzer", IEEE Trans. on Instrumentation and Measurement, Vol. 37, No 1, March 1988, pp.95-100.

[1.3] 8408B Automatic Network Analyzer, 500 MHz to 18 GHz, Operating and Service Manual, August 1982, Hewlett Packard.

CHAPTER 2

THEORETICAL BACKGROUND BEHIND VECTOR MEASUREMENTS OF HIGH FREQUENCY NETWORKS

2.1 MICROWAVE MEASUREMENT FUNDAMENTALS

2.1.1 Reflection Measurements

At microwave frequencies a standing wave pattern will be distributed along a transmission line when there is a load connected to it that is not equal to the lines's characteristic impedance. This standing wave pattern consists of the sum of the incident wave to the load, E_i and the reflected wave from the load, E_r. At the load, the ratio of E_r to E_i, and the phase difference between them are uniquely determined by the load impedance. This ratio is known as the **reflection coefficient**, $\tilde{\rho}$, where $\tilde{\rho}$ is a vector quantity containing a magnitude and phase component. $\tilde{\rho}$ represents the deviation of the load impedance from the characteristic impedance. In equation form:

$$\tilde{\rho} = \frac{E_r}{E_i} = \rho \angle \phi \text{.....eqn 2.1}$$

Other common microwave terms follow from equation 2.1:

$$Standing Wave Ratio(s) = \frac{1+\rho}{1-\rho} \text{.....eqn 2.2}$$

$$Return Loss(dB) = -20\log\rho \text{.....eqn 2.3}$$

2.1.2 Transmission Measurements

Transmission measurements are a function of the incident signal and the transmitted signal. The **linear transmission coefficient**, $\tilde{T}$, is defined as the ratio of transmitted voltage to incident voltage. In equation form:

$$\tilde{\tau} = \frac{E_t}{E_i} = \tau\angle\phi \text{.....eqn 2.4}$$

A common form of expressing transmission measurements is

$$Insertion Loss(dB) = 20\log\tau \text{.....eqn 2.5}$$

2.1.3 The Importance of Phase Information in Microwave Measurements

There are a number of important reasons why magnitude **and** phase information are measured as opposed to magnitude only.

- In the event that magnitude only information is required from a DUT, it is still preferable to measure both magnitude and phase. The reason for this is that we can obtain a much higher level of magnitude measurement accuracy by implementing vector accuracy enhancement techniques which use both magnitude and phase data.

- We can transform complex frequency domain data into the time domain, yielding more important information about the DUT.

- We can view the complex measurement data in formats such as the Smith Chart and Polar Displays.

- Phase information is fundamental to the operation of certain networks. For example, phased-array antennas require precise phasing of signal energy to elements in the array.

- Phase information is needed in order to compute **Group Delay** which is used as a measure of distortion in networks such as filters.

Figure 2.1 summarizes a few of these reasons.

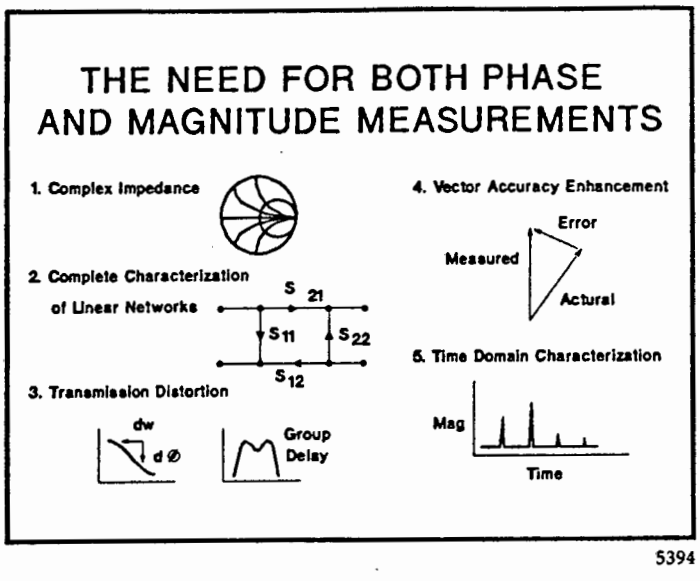


Figure 2.1. The Need for Magnitude and Phase Measurements [2.1].

Further reasons why it is preferable to measure phase information are given in [2.2].

2.1.4 s-parameters (Scattering Parameters)

At lower frequencies, parameters such as Z, H and Y parameters have been developed to characterize two-port networks. These parameters are unsuitable at RF or microwave frequencies because:

- Open and short circuits are difficult to realize.
- Devices such as transistors tend to oscillate when terminated with open or short circuits.
- It is very difficult to measure voltages and currents at these high frequencies.

At these high frequencies, scattering parameters, commonly known as **s-parameters**, are used to characterize a networks behaviour. Instead of voltages and currents, s-parameters relate incident and reflected travelling waves at the two ports of the DUT. Figure 2.2 shows a two-port device where:

a1 = incident wave at port one.
b1 = reflected wave at port one.
a2 = incident wave at port two.
b2 = reflected wave at port two.

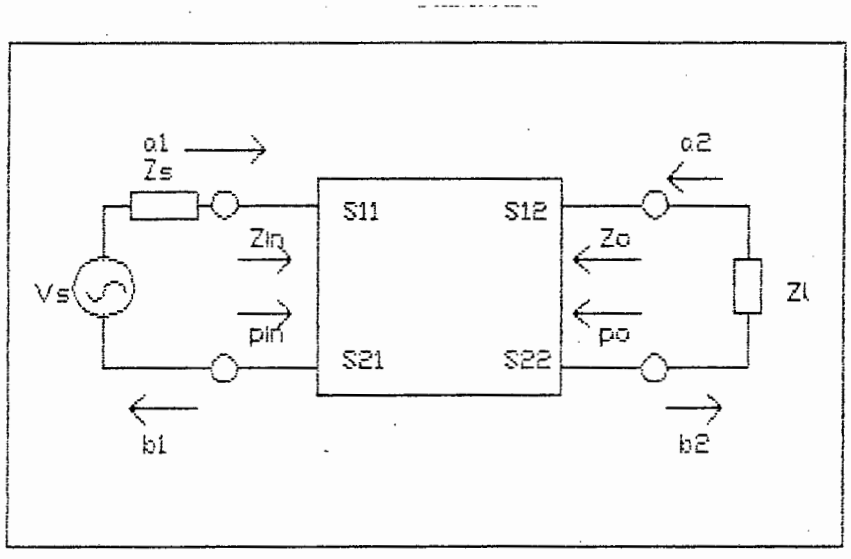


Figure 2.2. Two Port device.

We define two linear equations for a two-port device.

$$b_1 = S_{11} \cdot a_1 + S_{12} \cdot a_2 \text{.....eqn 2.6}$$

$$b_2 = S_{21} \cdot a_1 + S_{22} \cdot a_2 \text{.....eqn 2.7}$$

By using various terminations, we can isolate all four S-Parameters in terms of the a and b waves. The resultant S-Parameters are from [2.1],

$$S_{11} = \frac{b_1}{a_1}\Big|_{a_2=0} = \frac{Reflected}{Incident} = \tilde{\rho}_{forward} \quad \text{.....eqn 2.8}$$

$$S_{21} = \frac{b_2}{a_1}\Big|_{a_2=0} = \frac{Transmitted}{Incident} = \tilde{\tau}_{forward} \quad \text{.....eqn 2.9}$$

$$S_{22} = \frac{b_2}{a_2}\Big|_{a_1=0} = \frac{Reflected}{Incident} = \tilde{\rho}_{reverse} \quad \text{.....eqn 2.10}$$

$$S_{12} = \frac{b_1}{a_2}\Big|_{a_1=0} = \frac{Transmitted}{Incident} = \tilde{\tau}_{reverse} \qquad \text{.....eqn 2.11}$$

From the equations above, it is clear that S_{21} and S_{12} are equivalent to the forward and reverse transmission coefficients respectively and S_{11} and S_{22} are equivalent to the forward and reverse reflection coefficients respectively.

2.2 THE CONFIGURATION OF A NETWORK ANALYZER

Figure 2.3 shows the configuration of a Network Analyzer.

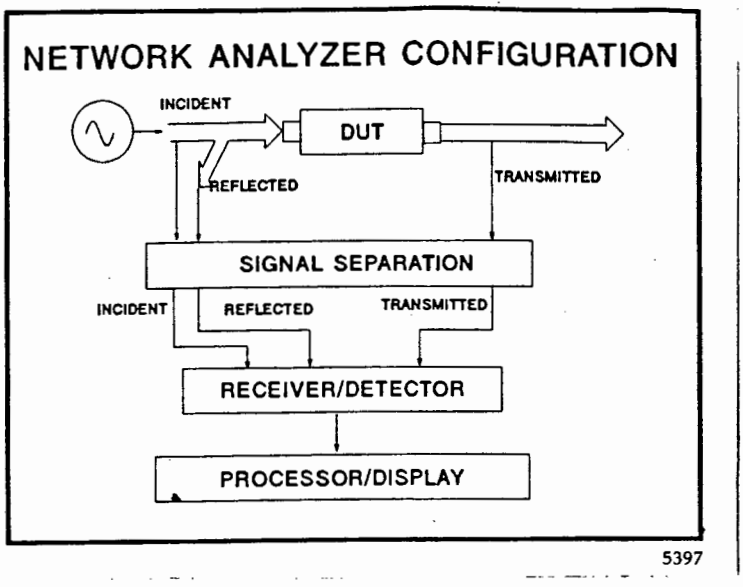


Figure 2.3. Network Analyzer Configuration [2.1].

The measurement system can be divided into four sections:

1. The Signal source which provides the incident signal to the DUT.
2. Signal Separation Devices which separate the incident, reflected and transmitted signals.

3. A Receiver to convert the microwave signals to a lower IF signal.
4. A Processor/Display to process the IF information and display it.

2.2.1 Signal Source

There are basically two types of signal sources used in Network Analyzer measurement systems, **Sweep Oscillators** and **Synthesized Sweepers.**

A sweep oscillator is in general much faster than a synthesizer and it is also much cheaper. On the other hand, a synthesized sweeper is a much more stable source. Which of the two sweepers to use for any specific measurement is determined by the nature of the measurement itself.

In general, a synthesizer is used for testing narrowband devices which display rapid magnitude variations with frequency. It is also suitable for obtaining accurate phase information especially if the DUT's phase response changes rapidly with frequency.

Conversely, measurements of most broadband devices whose responses do not change much with frequency, are ideally suited to sweep oscillators.
Figure 2.4 summarizes the applications of the two sources graphically. Note that in the case of the broadband measurement, there is virtually no difference in the magnitude responses.

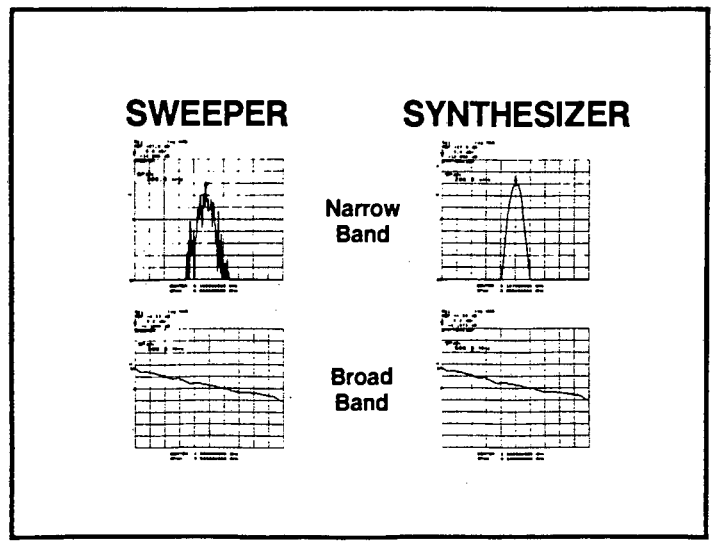


Figure 2.4. Sweeper and Synthesizer measurements [2.1].

2.2.2 Signal Separation Device

The incident, reflected and transmitted signals can be separated by Directional couplers, bridges, power splitters or high impedance probes.
Figure 2.5 shows three possible configurations for signal separation with regard to transmission measurements.

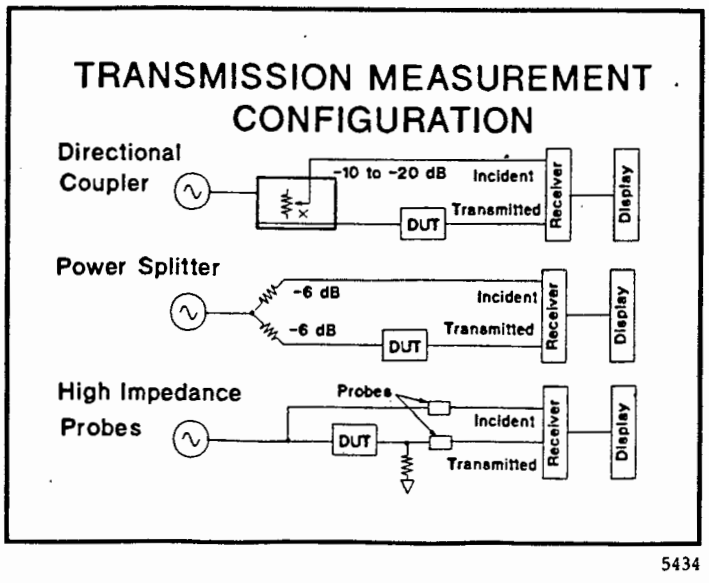


Figure 2.5. Transmission Measurement Configuration [2.1].

Of the three, directional couplers present the minimum loss (usually < 2dB) between the source and the DUT.
Power splitters present around 6dB of loss between the source and the DUT, are typically very broadband, have excellent frequency response and present a good match at the test device input.
High impedance probes are useful for making in-circuit measurements in impedances other than 50 or 75 ohms.

Reflection measurements require a directional device.
Figure 2.6 shows a reflection measurement configuration using a dual directional coupler to split the incident signal and a single directional coupler or bridge to separate the incident and reflected signals.

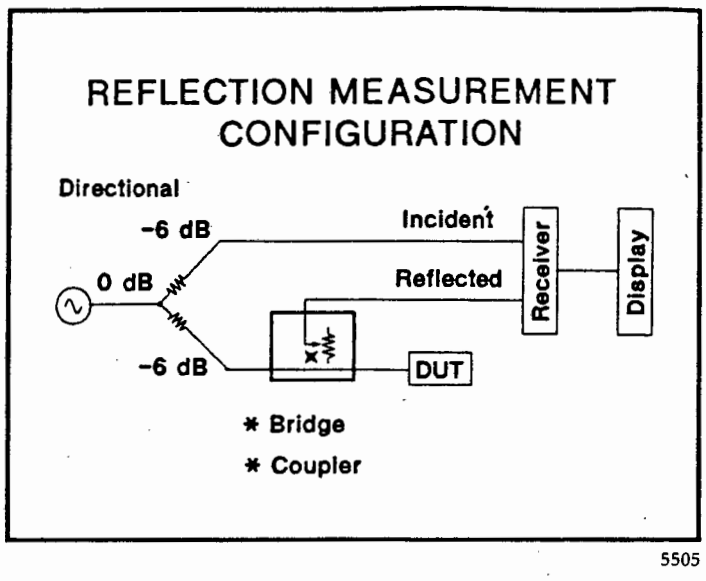


Figure 2.6. Reflection Measurement Configuration [2.1].

2.2.3 Receivers

The receiver provides the means of converting the microwave voltages to lower IF signals to allow for a more accurate measurement.

The three basic receiver techniques used in Network Analysis are shown in Figure 2.7.

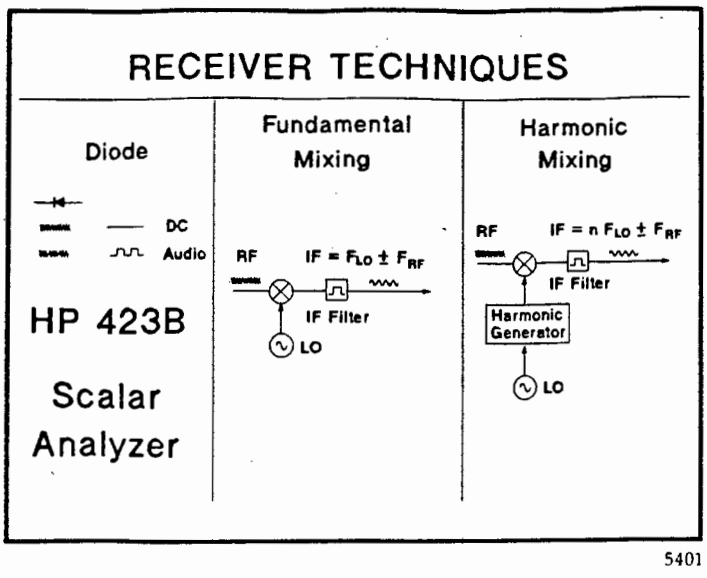


Figure 2.7. Receiver Techniques [2.1].

The diode technique of detection is most common in scalar analysis. Fundamental or Harmonic mixing is used in broadband tuned receivers and is the technique preferred for vector Network Analysis.

Figure 2.8 shows a bandpass filter magnitude response measured with a diode detector (left hand diagram) and measured with a tuned receiver. It is clear that the tuned receiver provides more dynamic range and is more immune to spurious responses.

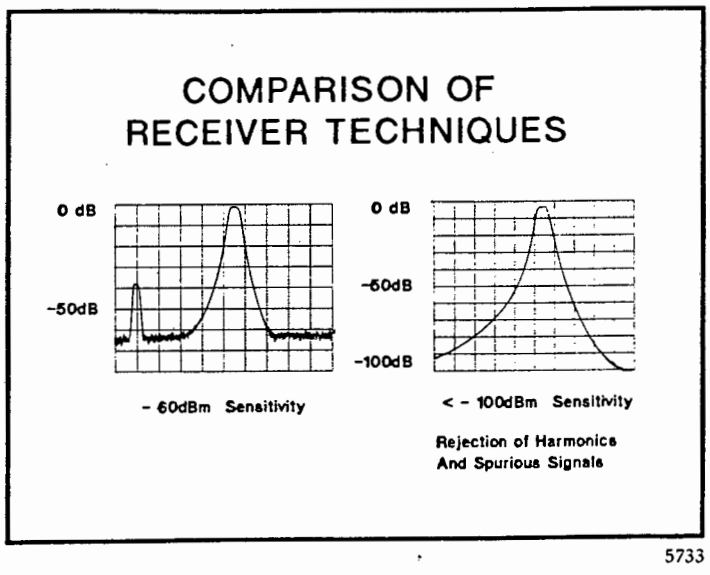


Figure 2.8: Comparison of Receiver Techniques [2.1].

2.2.4 Processor/Display

The processor "operates" on the IF signals coming from the receiver and the results are shown on a CRT display.
These "operations" refer specifically to "Ratioing" of the incident, reflected and transmitted IF signals for magnitude measurements. Note that it would have been far more difficult to perform the ratioing at RF or microwave frequencies.

Phase measurements are accomplished by comparing the phase of the reflected (or transmitted) IF signal with the incident IF signal. Figure 2.9 shows the processing operations for magnitude and phase measurements.

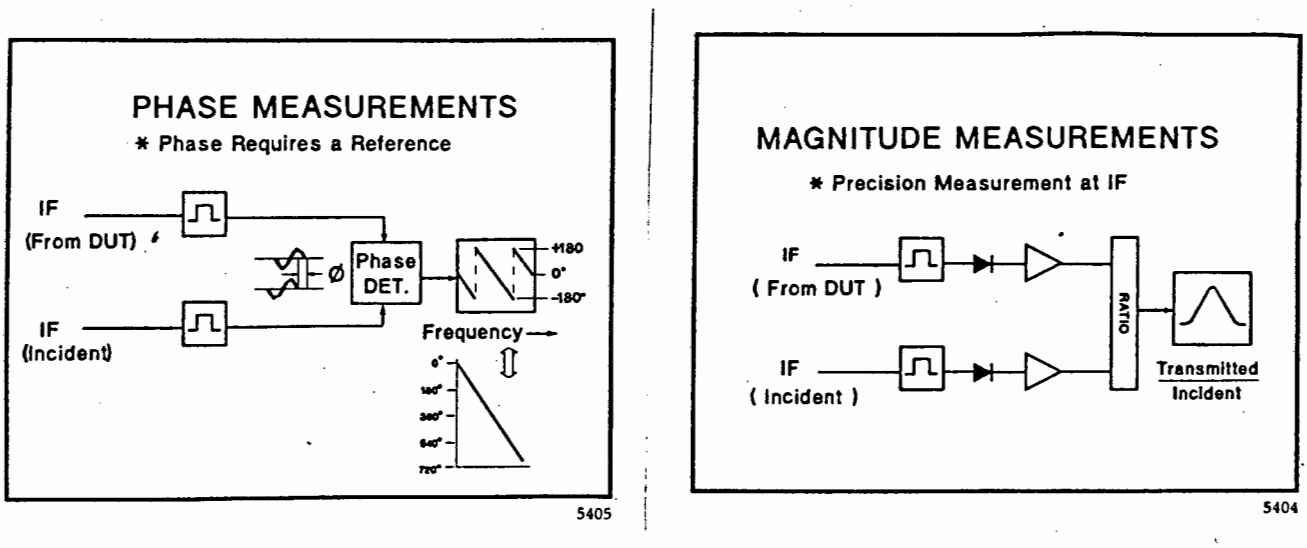


Figure 2.9. Magnitude and Phase Measurements [2.1].

Various different types of display formats exist.

2.2.4.1 **Smith Chart Format**

In general, calculations of transmission lines at high frequencies are very time consuming due to their complexity. As a result, various graphical aids have been developed to assist in the evaluation of calculations. One such graphical aid, commonly known as the **Smith Chart,** is generally accepted as being the most versatile and satisfactory for solving the most commonly encountered problems at high frequencies.

The Smith Chart is basically a plot on a reflection coefficient plane. The generally accepted form of the Chart [2.3] is derived from the relation

$$\tilde{\rho} = \frac{Z/Z_o - 1}{Z/Z_o + 1} \quad \text{.....eqn 2.12}$$

Where Z/Z_o is the normalized value of the impedance at a point on the transmission line and $\tilde{p}$ is the reflection coefficient seen previously.

Other display formats include Polar displays, Rectangular displays and Listings.

Figure 2.10 shows four possible display formats.

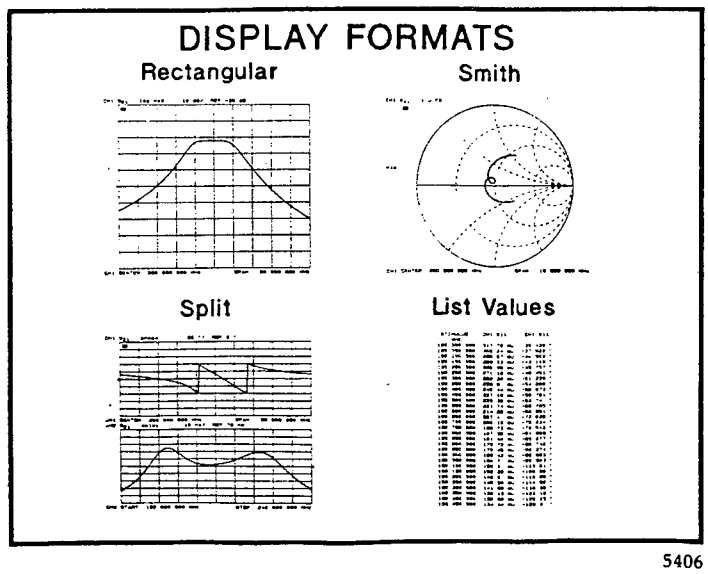


Figure 2.10. Display Formats [2.1].

2.3 ACCURACY ENHANCEMENT

A Network Analyzer measurement system has certain limitations in making accurate measurements. Any measurement performed with a Network Analyzer will have a specific error margin associated with it.

For example, consider the case where the magnitude of the reflected signal of a DUT equals the system Directivity. The measurement system is unable to distinguish between the

actual reflected signal from the DUT and that portion of the incident signal which has leaked through the Signal Separation Device.

When performing reflection measurements, the **Effective System Directivity** is often the major cause for measurement uncertainty. Similarly impedance mismatches and non-ideal frequency responses within the measurement system can lead to large measurement errors.

Ideally, one would want a measurement system free of impedance mismatches, providing infinite isolation, Directivity and dynamic range and with flat frequency responses. In practice the perfect measurement system is unrealizable.

What can we do to improve our measurement accuracy ? We could buy sophisticated hardware which would reduce these errors, but that could be very expensive. Alternatively, we could implement accuracy enhancement techniques which greatly improve the accuracy of measurements at high frequencies.

The basis of Vector Accuracy Enhancement is as follows:
The systematic errors inherent in the measurement system are modelled mathematically and various known Microwave standards are measured across the frequency band of interest. On the basis of these measurements applied to an error model, it is possible to mathematically remove/greatly reduce the systematic errors from all subsequent measurements over the same frequency band.

2.3.1 IDENTIFICATION AND CHARACTERIZATION OF ERRORS

There are two major groups of errors which exist within a Network Analyzer measurement system, namely **Random** errors and **Systematic** errors.

2.3.1.1 Random errors

These errors are random in nature and cannot be characterized and subsequently removed. They are errors due to factors like noise, humidity, temperature drift and connector repeatabilities.

We will limit our discussion to noise and connector repeatability.

(a) **Noise**

There are two different types of noise in any measurement system; low level noise and high level noise. Low level noise is the broadband noise floor of the receiver which can be varied through averaging or reducing the IFBW.
High level noise is noise or jitter of the trace data due to instability of the source. This form of noise can be substantially reduced by using a synthesizer in preference to a sweeper.

(b) **Connector Repeatability**

This is the random variation encountered when connecting a pair of RF connectors. This error limits the extent to which we can measure a very low level reflection

coefficient or a low loss transmission device. Minimization of errors due to connector repeatability can be achieved by keeping connectors clean and free of defects, and by using a torque wrench to make connections.

2.3.1.2 Systematic errors

Systematic errors are repeatable errors which can be measured by the system. Examples of systematic errors are system frequency response, impedance mismatches, leakage between signal separation devices and isolation.

Any measurement performed with a Network Analyzer will in effect be a Vector combination of the actual response of the DUT and all the errors in the system.
Fortunately, Systematic errors are the most significant in any microwave measurement. Since this group of errors is repeatable, it can be measured and subsequently removed from further measurements performed with the measuring system.
Systematic errors are quantified as Directivity, Source and Load Match, Isolation and Tracking(also known as Frequency Response error).

(a) Directivity

This error is due to that portion of the signal detected at the receiver test input which has not been reflected by the DUT. It leaks off before arriving at the DUT due to:

1. The imperfection of the Signal Separation Device.

2. Reflection effects of test cables and/or adapters used to connect the DUT to the Signal Separation Device.

The effect of Directivity on reflection measurements is shown by means of a signal flow graph in Figure 2.11.

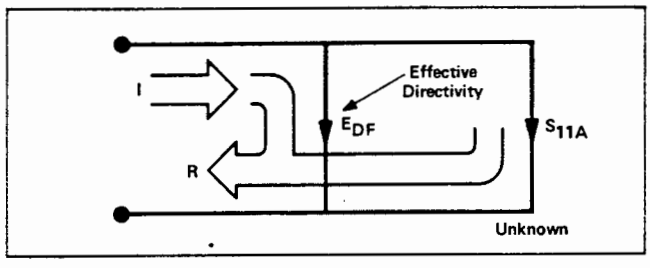


Figure 2.11. Directivity Error [2.4].

Directivity is generally the major source of uncertainty in reflection measurements.

Here is an example of how Directivity can affect a reflection measurement [2.4]:

Let's assume we desire to measure a device which has SMA connectors and an actual 24dB Return loss. Using SMA connectors in the test setup will typically degrade a test set's (eg HP8743A test set) Directivity from over 30dB to about 26dB due to adapter reflections. This corresponds to a return loss uncertainty of 20dB!
The uncorrected measured return loss value could be anywhere between 19dB and 39dB depending upon how the actual and error signals combine vectorially at the test input.

(b) <u>Source Match</u>

Looking back into a measuring system including all the

adapters and cables, we do not see a perfect match. When a signal is reflected from a DUT and returns into the test set, it is faced with a mismatch. This means that a portion of the reflected signal will be re-reflected in the opposite direction and this re-reflection effect will continue, creating an error-loop.

This effect causes the magnitude and phase of the incident wave (I) to vary as a function of S_{11A}. Levelling the source to produce constant (I) reduces this error, but since the source cannot be levelled at the test device input, levelling cannot eliminate all power variations.

Figure 2.12 below shows the addition of source match error to the signal flow graph model.

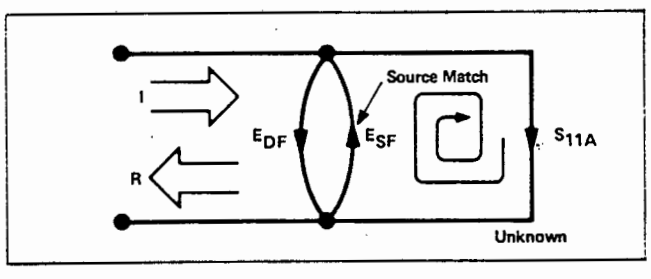


Figure 2.12. Source Match Error [2.4].

Note that source match uncertainty is a factor in both transmission and reflection measurements. This error only becomes major when dealing with devices that have high mismatches. The literature indicates [2.4] that this error can typically be reduced by a factor of ten with the use of Vector Error Correction techniques.

(c) **Load Match**

This error is defined as the vector sum of signals appearing at the receiver test input due to effects of

impedance mismatches between the test device output port and the receiver test input.
Load Match uncertainty is a factor in all transmission measurements and in reflection measurements of two-port devices.

(d) **Isolation**

Isolation is defined as the vector sum of signals appearing at the receiver detector due to Crosstalk/Leakage between the reference and test signal paths, including signal leakage across test switches and within both the RF and IF sections of the receiver.

Isolation becomes a significant factor when the transmitted signal level is reduced substantially by the test device (ie high loss devices).

The literature [2.4] indicates that the removal of Isolation errors by accuracy enhancement can extend the dynamic range of a measurement system by up to 10dB.

(e) **Tracking/Frequency Response Error**

Frequency response error represents the variation in the frequency response of the system going to and coming from the DUT. Reflection frequency response is actually the vector sum of the frequency response of the coupler, the test cables and adapters, and the input mixers or detectors.

Figure 2.13 shows the addition of frequency response error to the Signal flow graph model.

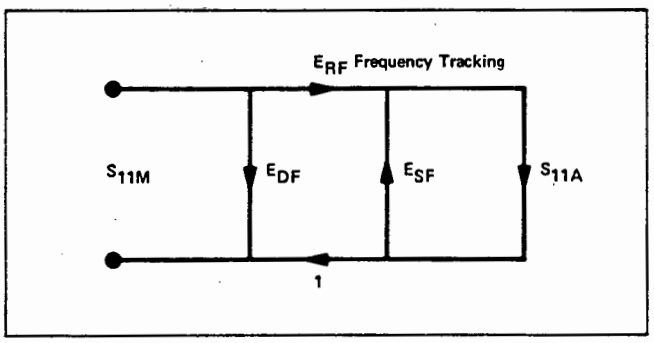


Figure 2.13. Tracking Error [2.4].

This error is a factor in both transmission and reflection measurements. The literature indicates [2.4] that frequency response error can be removed completely using vector accuracy enhancement.

2.3.2 CORRECTION OF SYSTEMATIC ERRORS

Two different error models, namely the eight-term error model and the twelve-term error model, plus the frequency-response-only calibration are covered in this section.

2.3.2.1 Frequency-Response-Only-Calibration

A frequency-response-only calibration is the most basic calibration procedure. It removes frequency-response-only errors from microwave measurements. For transmission measurements, a **thru** is used to quantify frequency response error and for reflection measurements a short-circuit is normally used.

2.3.2.2 Eight-Term Error Model

This model provides Directivity, Source Match and frequency response vector error correction for reflection measurements and frequency-response-only error correction for transmission measurements.
It is best suited to high accuracy reflection measurements of one-port devices and fast transmission measurements of two-port devices. It may also be used for reflection measurements of two-port devices providing the output port is terminated in the system characteristic impedance (usually 50 ohms).

(a) Reflection Measurements

It can be shown [2.4] that Directivity, Source Match and frequency response errors are mathematically related to the actual reflection coefficient, S_{11A} and measured reflection coefficient, S_{11M} by the following equation:

$$S_{11A} = \frac{S_{11M} - E_{DF}}{E_{SF} \cdot (S_{11M} - E_{DF}) + E_{RF}} \quad \text{.....eqn 2.13}$$

This is the eight-term error model for reflection.

If the three error terms are known, the equation can be solved for S_{11A} since we can measure S_{11M} at any given frequency. Fortunately, it is possible to quantify these three errors by means of measuring three known microwave

calibration standards.
Theoretically any three known devices/standards will suffice.

In general, a matched load, an open circuit and a short circuit termination are used.

(a.i) Measurement of a Matched Load

Let us first assume the existence of a "Perfect Load". A "Perfect Load" is a reflectionless termination. All the energy incident at the load is absorbed by the load and there is zero reflected energy.

If we were to measure such a load, we would quantify Directivity error since all the terms on the right-hand-side of equation 2.13 would be zero except Directivity. Figure 2.14 shows the measurement of a "Perfect Load".

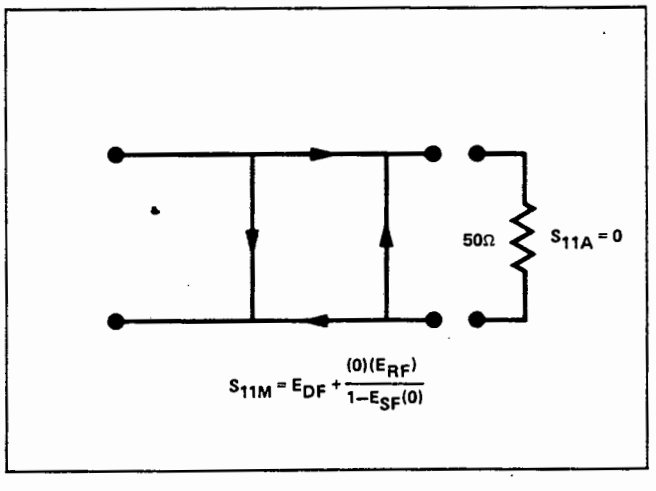


Figure 2.14. Measuring a "Perfect" Load [2.4].

In practice a "Perfect Load" is unrealizable. Figure 2.15 shows the relationship between the actual reflection coefficient of the load, the actual Directivity and the measured Directivity.

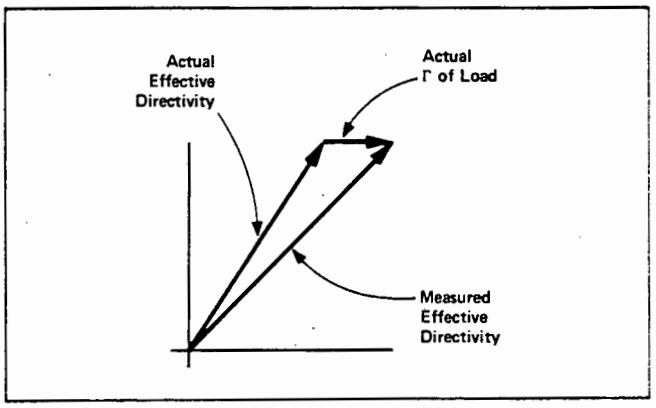


Figure 2.15. Actual and Measured Directivity [2.4].

It is clear that any reflection from the termination represents an error.

Due to the difficulty of manufacturing very high quality fixed terminations at microwave frequencies, it is recommended that a sliding load be used, especially at higher frequencies (>2 GHz). The Directivity vector at a given frequency is determined by sliding the load on the airline to create a circle of data points. The centre of this circle is the Directivity vector at that frequency. Figure 2.16 demonstrates the determination of Directivity using a sliding load.

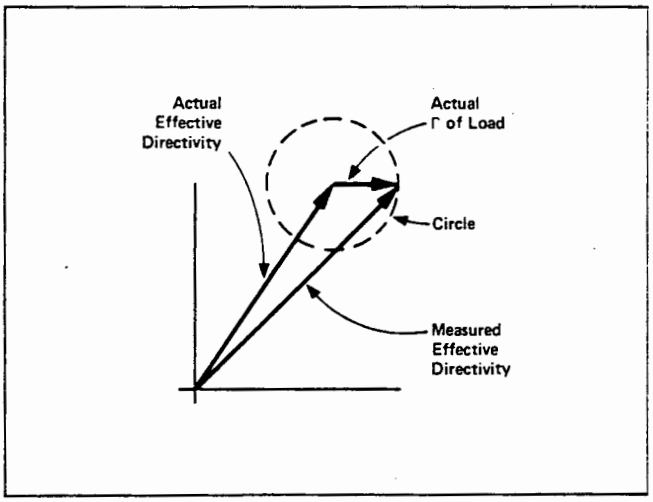


Figure 2.16. Determining Directivity with a Sliding Load [2.4].

(a.ii) **Measurement of an Open Circuit and a Short Circuit**

By measuring an open circuit and a short circuit termination and substituting the results into equation 2.9, we are left with the situation of two equations in two unknowns. These two unknowns are the error terms Source Match (E_{SF}) and Frequency response error (E_{RF}).
Figure 2.17 shows how the two simultaneous equations are obtained.

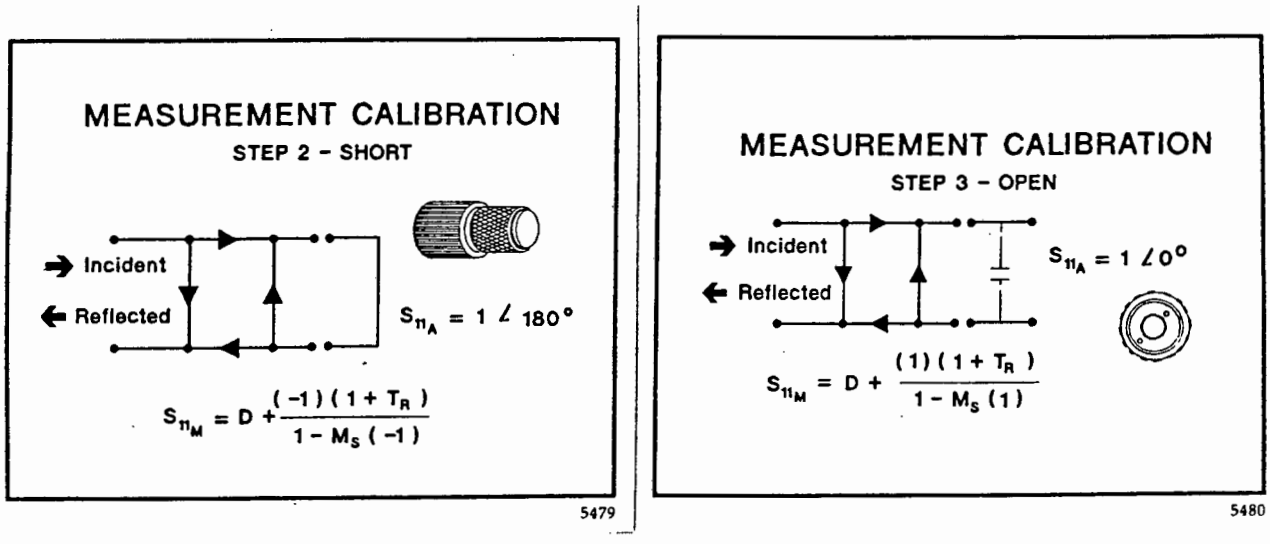


Figure 2.17. Calibration using a Short and an Open Circuit [2.1].

Once again, as in the case of the matched load termination, we are faced with an imperfect measurement standard. An open circuit has a certain amount of **fringing capacitance** which is a function of it's physical geometry and the frequency of the signal. This fringing capacitance leads to uncertainties in measurements of an open circuit.

By the use of a **Shielded open circuit** we are able to accurately model this fringing capacitance and subsequently remove it's effect from our measurements. The model used is from [2.5],

$$C(f) = C_{dc} + C_1 \cdot f + C_2 \cdot f^2 \quad \text{.....eqn 2.14}$$

where C_n is the Capacitance in Picofarads and is a function of the type of connectors being used in the test set up, and f is the frequency in Gigahertz.

Figure 2.18 shows how fringing capacitance is "controlled" by the use of a shielded open circuit.

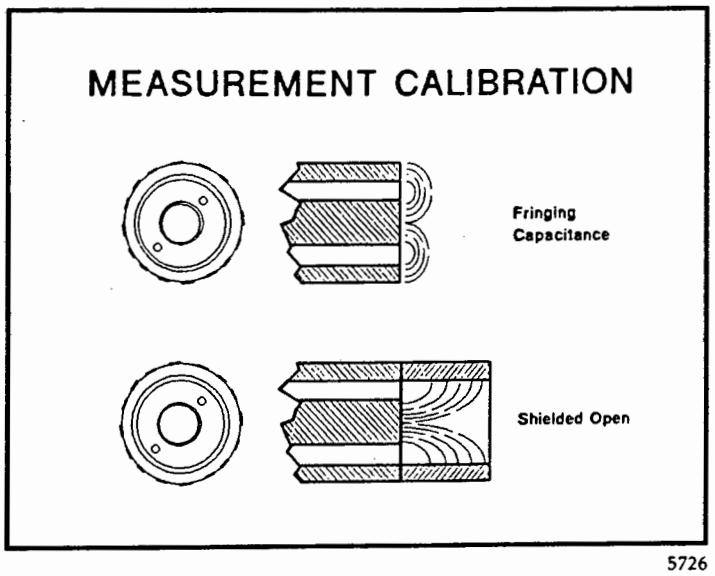


Figure 2.18. The effect of a shielded open circuit on fringing capacitance [2.1].

Now that we have quantified Directivity, Frequency response and Source Match at any specified frequency, we can extract the actual reflection coefficient ,S_{11A} from the measured value, S_{11M}, at the same frequency. Note that we cannot completely remove the systematic errors since our measurement standards are imperfect. The better the standards we can use, the smaller the residual value of error remaining.

Let's again consider the eight-term error model for reflection.

$$S_{11M} = E_{DF} + \frac{S_{11A} \cdot E_{RF}}{1 - E_{SF} \cdot S_{11A}} \quad \text{.....eqn 2.15}$$

By rearranging this equation and making S_{11A} the subject of the formula, we get the following equation.

$$S_{11A} = \frac{S_{11M} - E_{DF}}{E_{SF} \cdot (S_{11M} - E_{DF}) + E_{RF}} \quad \text{.....eqn 2.16}$$

This is the reflection component of the eight-term error model solved for S_{11A}.
Note that this error model may be applied to reflection measurements of two-port devices providing the output port of the DUT is terminated in the system characteristic impedance. The additional reflection error caused by non-ideal termination at the DUT's output port is not accounted for in the eight-tern error model.

Figure 2.19 illustrate the typical magnitude and phase uncertainties for both a response calibration measurement (frequency-response-only calibration) and a full eight-term reflection calibration.

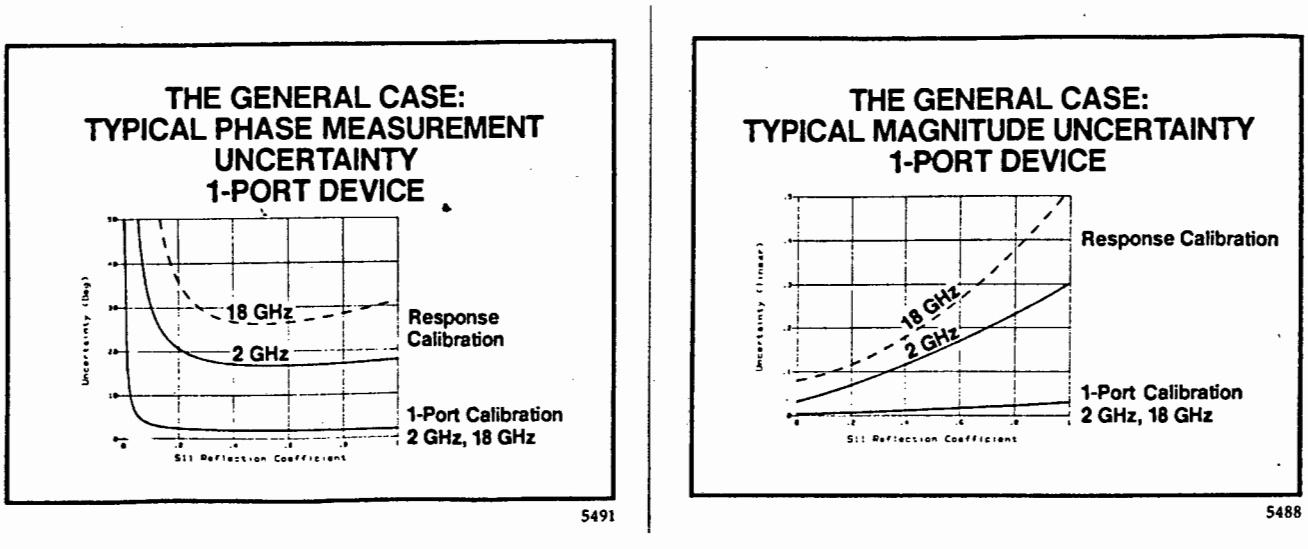


Figure 2.19. Magnitude and Phase Uncertainties [2.1].

(b) **Transmission Measurements**

Transmission measurements with an eight-term error model are limited to frequency-response-only error correction. The equations for transmission measurements are thus very simple and are as follows:

$$S_{21A} = \frac{S_{21M}}{E_{TF}} \quad \text{for forward transmission} \quad \text{.....eqn 2.17}$$

$$\text{and } S_{12A} = \frac{S_{12M}}{E_{TR}} \quad \text{for reverse transmission} \quad \text{.....eqn 2.18}$$

The frequency response error term is quantified by measuring a transmission **thru** with the test set in a transmission configuration. This means that the points at which the DUT will be connected are joined to achieve a zero loss, zero length transmission line.

2.3.2.3 **Twelve-Term Error Model**

The twelve-term error model provides full Directivity, Isolation, Source Match, Load Match and Frequency Response Vector error correction for transmission and reflection measurements of **two-port devices.**
It is best suited to high accuracy transmission and reflection measurements of two-port devices.

A drawback of this model is the fact that in order to measure **any one** of the four possible S-Parameters, the model requires the measurement of **all four** S-Parameters.

If certain conditions are met by the measurement system, a closed form solution commonly known as the twelve-term error model for S-parameter measurements is valid. The conditions to be satisfied are laid out in [2.5].

In the event that the measurement system fails to satisfy these conditions, the error model assumes a more complicated form [2.5], and can only be solved by means of a numerical method.
We will assume that our measurement system meets the conditions laid out in [2.5].

It can be shown mathematically [2.6] that the four twelve-term error model equations for a two-port device are as shown in Figure 2.20.

As was the case with the eight-term error model, through a full two-port measurement calibration, we can characterize and mathematically remove/reduce the effects of the systematic errors.

$$S_{11A} = \frac{\left[\left(\frac{S_{11M} - E_{DF}}{E_{RF}}\right)\left[1 + \left(\frac{S_{22M} - E_{DR}}{E_{RR}}\right)E_{SR}\right]\right] - \left[\left(\frac{S_{21M} - E_{XF}}{E_{TF}}\right)\left(\frac{S_{12M} - E_{XR}}{E_{TR}}\right)E_{LF}\right]}{\left[1 + \left(\frac{S_{11M} - E_{DF}}{E_{RF}}\right)E_{SF}\right]\left[1 + \left(\frac{S_{22M} - E_{DR}}{E_{RR}}\right)E_{SR}\right] - \left[\left(\frac{S_{21M} - E_{XF}}{E_{TF}}\right)\left(\frac{S_{12M} - E_{XR}}{E_{TR}}\right)E_{LF}E_{LR}\right]}$$

$$S_{21A} = \frac{\left[1 + \left(\frac{S_{22M} - E_{DR}}{E_{RR}}\right)\left(E_{SR} - E_{LF}\right)\right]\left(\frac{S_{21M} - E_{XF}}{E_{TF}}\right)}{\left[1 + \left(\frac{S_{11M} - E_{DF}}{E_{RF}}\right)E_{SF}\right]\left[1 + \left(\frac{S_{22M} - E_{DR}}{E_{RR}}\right)E_{SR}\right] - \left[\left(\frac{S_{21M} - E_{XF}}{E_{TF}}\right)\left(\frac{S_{12M} - E_{XR}}{E_{TR}}\right)E_{LF}E_{LR}\right]}$$

$$S_{12A} = \frac{\left[1 + \left(\frac{S_{11M} - E_{DF}}{E_{RF}}\right)\left(E_{SF} - E_{LR}\right)\right]\left(\frac{S_{12M} - E_{XR}}{E_{TR}}\right)}{\left[1 + \left(\frac{S_{11M} - E_{DF}}{E_{RF}}\right)E_{SF}\right]\left[1 + \left(\frac{S_{22M} - E_{DR}}{E_{RR}}\right)E_{SR}\right] - \left[\left(\frac{S_{21M} - E_{XF}}{E_{TF}}\right)\left(\frac{S_{12M} - E_{XR}}{E_{TR}}\right)E_{LF}E_{LR}\right]}$$

$$S_{22A} = \frac{\left[\left(\frac{S_{22M} - E_{DR}}{E_{RR}}\right)\left[1 + \left(\frac{S_{11M} - E_{DF}}{E_{RF}}\right)E_{SF}\right]\right] - \left[\left(\frac{S_{21M} - E_{XF}}{E_{TF}}\right)\left(\frac{S_{12M} - E_{XR}}{E_{TR}}\right)E_{LR}\right]}{\left[1 + \left(\frac{S_{11M} - E_{DF}}{E_{RF}}\right)E_{SF}\right]\left[1 + \left(\frac{S_{22M} - E_{DR}}{E_{RR}}\right)E_{SR}\right] - \left[\left(\frac{S_{21M} - E_{XF}}{E_{TF}}\right)\left(\frac{S_{12M} - E_{XR}}{E_{TR}}\right)E_{LF}E_{LR}\right]}$$

Figure 2.20. Twelve-term error model equations [2.4].

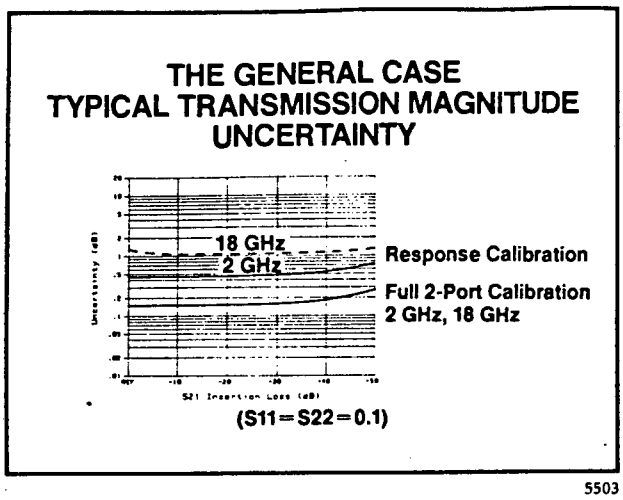
Source Match, Directivity and frequency response errors are quantified in exactly the same way as they are for the case of the eight-term error model. This gives us a total of six error terms since we measure these three errors in both directions (S_{11} and S_{22}).

A **thru** calibration is used to characterize transmission frequency response and load match errors in both directions.
Load match is quantified simply by measuring the reflection coefficient of the thru connection and transmission frequency response is quantified as in the case of the eight-term error model for transmission.

The only error remaining is Crosstalk/Isolation. This error is measured with the test set in the transmission configuration and a fixed termination installed at the points at which the test device will be connected.

Since some microwave test sets can measure both the forward and reverse characteristics of the DUT without the need to manually remove and physically reverse it, the comprehensive twelve-term transmission and reflection error model includes forward and reverse error terms, with each set consisting of six error terms. For test sets that cannot be switched between forward and reverse, the forward terms are used in both situations with the DUT being manually reversed.

Figure 2.21 illustrates the typical magnitude and phase uncertainties for both a response calibration measurement (frequency-response-only calibration) and a full twelve-term transmission calibration.



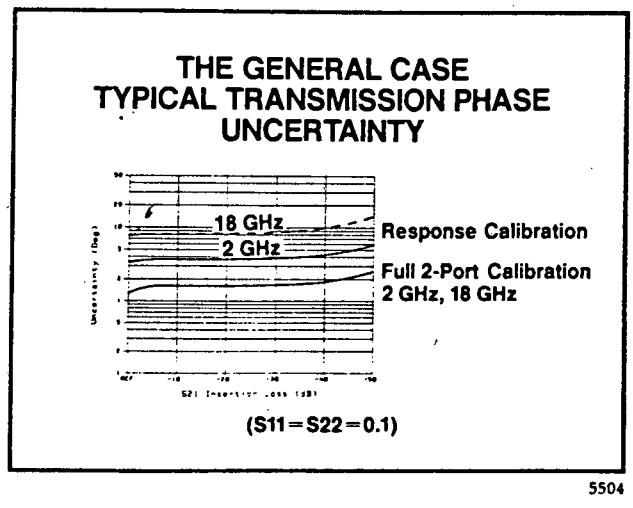
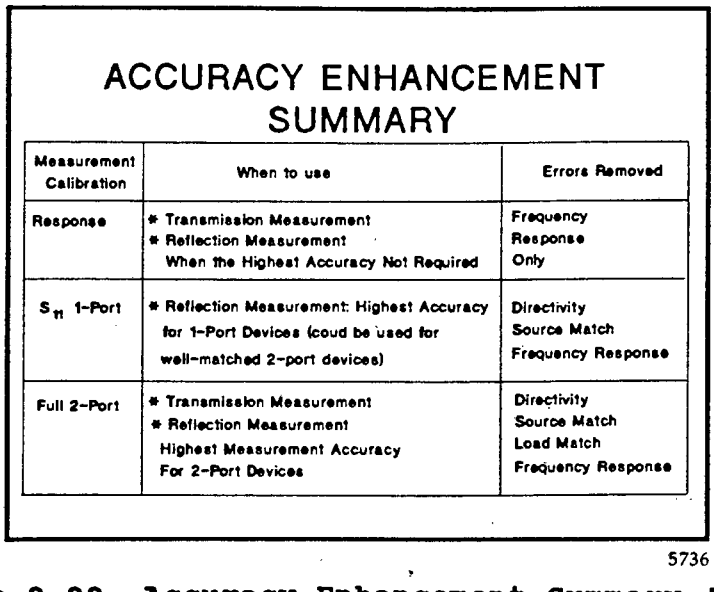


Figure 2.21. Magnitude and Phase Uncertainties [2.1].

Figure 2.22 summarizes the three measurement calibration procedures discussed previously and the circumstances under which the use of each method is preferred.



Measurement Calibration	When to use	Errors Removed
Response	* Transmission Measurement * Reflection Measurement When the Highest Accuracy Not Required	Frequency Response Only
S_{11} 1-Port	* Reflection Measurement: Highest Accuracy for 1-Port Devices (coud be used for well-matched 2-port devices)	Directivity Source Match Frequency Response
Full 2-Port	* Transmission Measurement * Reflection Measurement Highest Measurement Accuracy For 2-Port Devices	Directivity Source Match Load Match Frequency Response

Figure 2.22. Accuracy Enhancement Summary [2.1].

REFERENCES

[2.1] Vector Measurements of High Frequency Networks, March 1987, Hewlett Packard.

[2.2] Microwave Network Analyzer Applications, pp. 1.1-1.11, June 1970, Hewlett Packard

[2.3] Chipman, R.A., "Shaum's outline of theory and problems of transmission lines", pp. 184-202, McGraw-Hill, New York (1968).

[2.4] Automating the HP8410B Microwave Network Analyzer, Application Note 221A, June 1980, Hewlett Packard.

[2.5] Williams, W.L.,Compton, R.C., Rutledge, D.B., "Computer Automation and Error Correction for a Microwave Network Analyzer", IEEE Trans. on Instrumentation and Measurement, Vol. 37, No 1, March 1988, pp.95-100.

[2.6] Rehnmark, S, 'On the Calibration Process of Automatic Network Analyzer Systems", IEEE trans. Microwave theory tech., vol. MTT-22, no 4, pp. 457-458, April 1974.

CHAPTER 3

HARDWARE IMPLEMENTATION OF A REAL-TIME NETWORK ANALYZER SYSTEM

3.1 INTRODUCTION

A feature of automatic Network Analyzer systems previously developed [3.1, 3.2], is that they operate in a **non real-time** manner. The process of calibrating the Network Analyzer and subsequently performing measurements on a DUT can be very time consuming using these non real-time measurement systems.

This chapter deals with the hardware of a **real-time** Network Analyzer measurement system. In the following chapter, the software which compliments this hardware, is discussed in detail.

The basic block diagram for the hardware of the proposed system is shown in Figure 3.1.

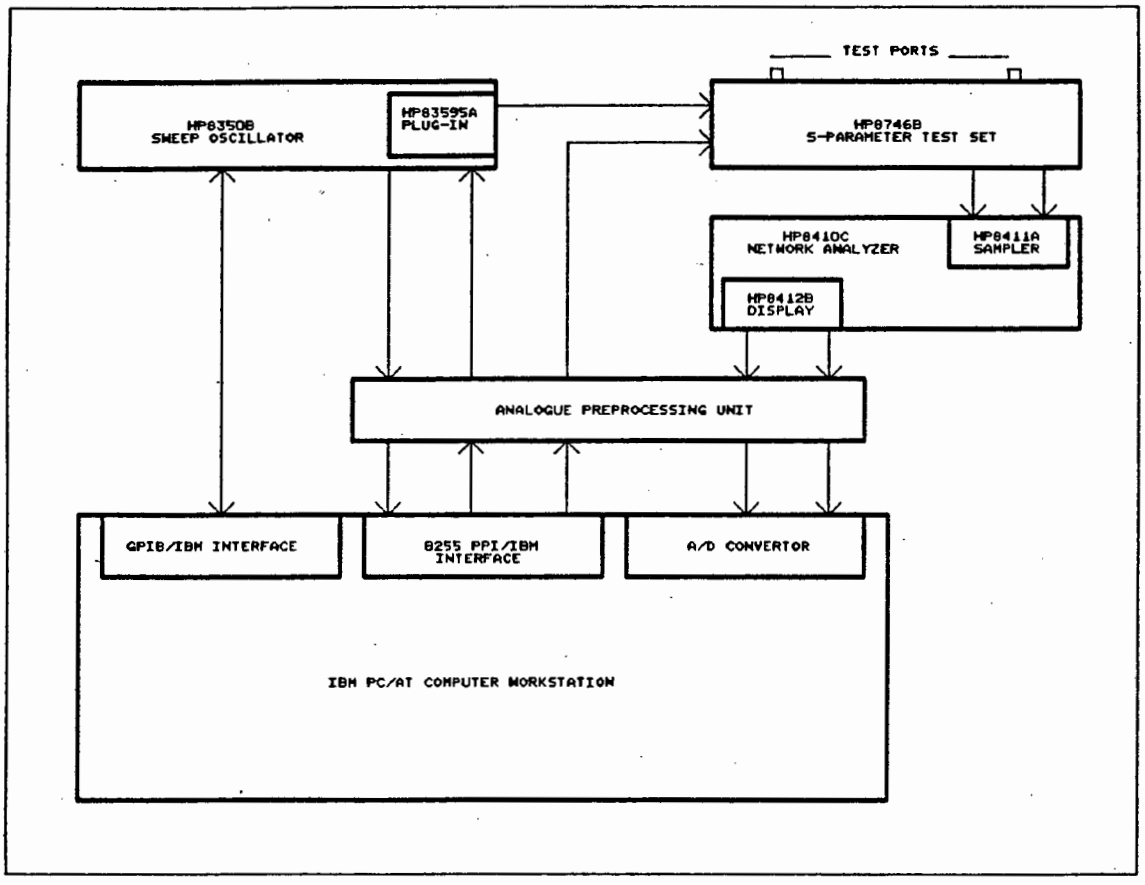


Figure 3.1. Block diagram of the Network Analyzer measurement system.

The system consists of five major modules:

(1) The HP8410C Network Analyzer Mainframe.
(2) The HP8746B S-Parameter Test Set.
(3) The HP8350B Sweep Oscillator.
(4) An IBM Pc/AT Computer Workstation.
(5) An Analogue Preprocessing Unit.

Modules (1), (2) and (3) comprise the existing manual Network Analyzer system as implemented in the postgraduate

microwave laboratory at the University of Cape Town. The addition of modules (4), the IBM Pc/AT workstation. and (5), the analogue preprocessing unit, add an automatic capability to the system. These modules allow the Network Analyzer to be controlled remotely, thereby reducing the risk of operator error which is a factor in manual measurements. Furthermore, several new features are made available by means of powerful software, a full discussion of which is to be found in chapter 4.

The basis of the real-time operation of the system is as follows;

(1) The computer triggers a sweep on the sweep oscillator.

(2) By monitoring control lines, the computer can decide when to start and when to stop digitizing the analogue phase and magnitude data coming from the display unit.

(3) Repetition of steps (1) and (2) results in a real-time display of magnitude and phase or complex impedance as a function of frequency.

3.2 DISCUSSION OF THE SYSTEM MODULES

3.2.1 HP8410C Network Analyzer Mainframe

The HP8410C Network Analyzer mainframe consists of the HP8410C Network Analyzer, the HP8411A Harmonic Frequency Convertor and the HP8412B Phase-Magnitude display.

The mainframe acts as the receiver and processor/display modules previously discussed in sections 2.2.3 and 2.2.4.

The HP8412B display provides phase and magnitude information in a rectilinear form on a CRT. Analogue voltages for both phase and magnitude are available at the rear of the display in the form of BNC output jacks.

A complete description of this module can be found in [3.3, 3.4].

3.2.2 HP8746B S-Parameter Test Set

The HP8746B s-parameter test set contains the necessary microwave circuits for measuring all four s-parameters of a two-port device from 0.5 to 12.4 GHz. This is the signal separation device discussed in section 2.2.2.

The s-parameters and incident attenuation can be set by means of front panel push-buttons (manual) or by remote-contact closures (automatic). A detailed description of this module can be found in [3.5].

3.2.3 HP8350B Sweep Oscillator

The HP8350B sweep oscillator with its HP83595A RF plug-in make up the signal source described in section 2.2.1. The source has a range of 0.01 to 26.5 GHz but the measurement system as a whole is limited to the range 0.5 to 12.4 GHz due to the s-parameter test set specifications.

All front panel controls except the power switch may be programmed remotely via the GP-IB (General Purpose Interface Bus).

The HP83595A 10 MHz to 26.5 GHz RF output is provided in five bands. When sweeping a range of frequencies larger than a single band, the switching between these bands is done automatically. There are problems associated with bandswitching [3.6] such as harmonics, sweep time, stability and switching discontinuities, but careful selection of sweep frequencies can avoid these problems.

Figure 3.2 illustrates the bandswitching points in the sequential and single band sweep modes. Note that these bandswitch points are approximate values and in practice they were found to vary by as much as 500 MHz on either side of the stated values.

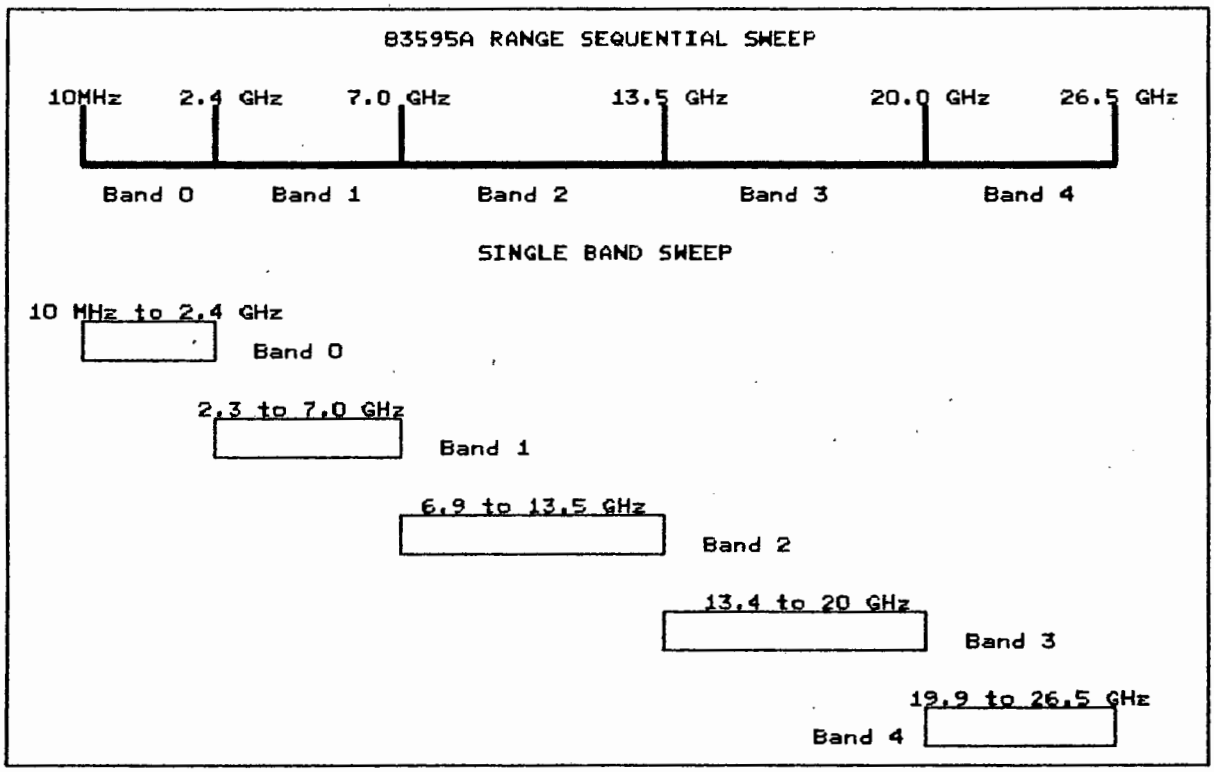


Figure 3.2. Bandswitching in Sequential and Single Band Sweep Modes.

Further details of this module can be found in [3.6, 3.7].

3.2.4 IBM Pc/AT Computer Workstation.

The IBM Pc/AT or compatible computer workstation, which includes three interface cards, the GP-IB/IBM Pc interface, the 8255 PPI/IBM Pc interface and an A/D conversion card, constitutes the brain behind the automatic Network Analyzer system.

The computer used in this system is an IBM AT compatible which is based on Intel's 16-bit 80286 microprocessor. It has a clock speed of 12 MHz with zero wait states. A 80287 math coprocessor is also incorporated to ensure speedy numerical processing of the amplitude and phase information.

3.2.4.1 GP-IB/IBM Pc Interface

The General Purpose Interface Bus (GP-IB), also known as HPIB and IEEE-488, is a system intended to link intelligent instruments together. Its major purpose is to connect instruments and test gear in laboratory environments where the distances are usually limited (less than 20 metres).

The GP-IB interface used in this measurement system is a commercially available product which can be programmed from several different high level languages. The purpose of this interface in the system is to allow the remote programming of the HP8350B sweep oscillator and the HP83595A RF plug-in unit.

3.2.4.2 8255 PPI/IBM Interface

The 8255 Programmable Peripheral Interface (PPI) is a general purpose programmable Input/Output device designed for use with microprocessors such as the 80286. It consists of three 8-bit ports which may be programmed in various different modes [3.8].

In this system, the 8255 is used in Mode zero. Port A and B are configured as outputs with port A controlling the selection of Incident Attenuation and Port B controlling the selection of s-parameters. Port C is used as an input, monitoring POS Z BLANK, **and** as an output, triggering a sweep.

The 8255 interface card used in this system was designed and built in the Digital postgraduate laboratory at the University of Cape Town. Appendix A shows the port pinouts and addresses of this interface card.

3.2.4.3 A/D Conversion Card

The A/D conversion card used in this system is a commercially available product known as the PC26 analogue to digital conversion card. It features 12-bit resolution, a maximum conversion time of around 40us and 16 multiplexed inputs.
Appendix A shows the port addresses and functions of the PC26 A/D card.
The card is used to digitize the magnitude and phase information coming from the HP8412B display, thus allowing subsequent storage and signal processing of the information.

3.2.5 Analogue Preprocessing Unit

Figure 3.3 shows the basic block diagram of the analogue preprocessing unit.

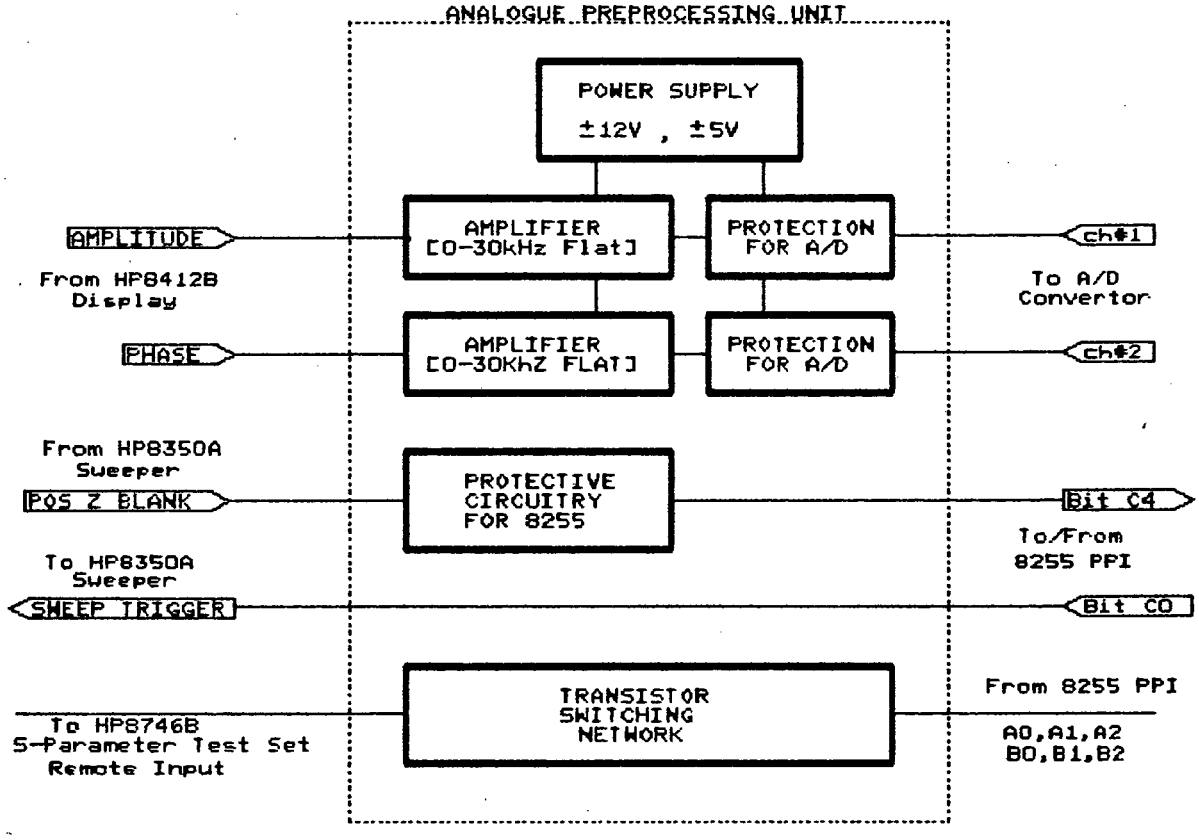


Figure 3.3. Block diagram of Analogue Preprocessing Unit.

The unit consists of five modules;

(1) An Amplifier module.
(2) An A/D protection module.

(3) Transistor Switching Network.
(4) Protective circuitry for the 8255 PPI.
(5) Power supply.

3.2.5.1 Design and Construction of Analog Preprocessing Unit Modules

(a) Amplifier

The amplitude and phase signals from the HP8412B display unit must be scaled up from their low levels to the A/D convertor's input range (-5V to +5V). The low level amplitude signal ranges between -2V and +2V and the low level phase signal ranges between -1.8V and +1.8V. Thus, the two amplifiers must have gains of around two and a half times.

Further amplifier requirements are;

- A flat magnitude response up to approximately 30kHz.
- A non-inverting amplifier configuration.

The magnitude response criterion is critical because we wish to reproduce the amplitude and phase data with the minimum of distortion. The amplitude information coming from the HP8412B display has a sensitivity of 50mV/dB. Thus, for every decibel (dB) of amplitude variation, the output voltage changes by 50mV. If we assume that the amplifier can introduce a maximum of 0.01 dB error in the amplitude data then we can compute the allowable ripple in the response of the amplifier as follows.

Let the allowed error, in volts, of the amplitude information at the output of the amplifier equal E_o.
Let the actual gain of the amplifier equal G_a.
Let the maximum allowable ripple in the magnitude response of the amplifier equal R_{max}.

$E_o = 0.125 \cdot 0.01 = 0.00125$ Volts, assuming a gain of 2.5 times

$$G_a = \frac{5.00125}{2} = 2.500625 \text{ times}$$

$$R_{max} = 20\log[2.00625] - 20\log[2.5] = 0.00217\,dB$$

A standard non-inverting amplifier circuit was used and its frequency response was simulated on MICROCAPS, a computer simulation package. Figure 3.4 shows the amplifier frequency response up to 40kHz. The graph shows that the ripple is less than 0.00125dB which is well within the 0.00217dB figure calculated above corresponding to an error of 0.01dB in amplitude data.

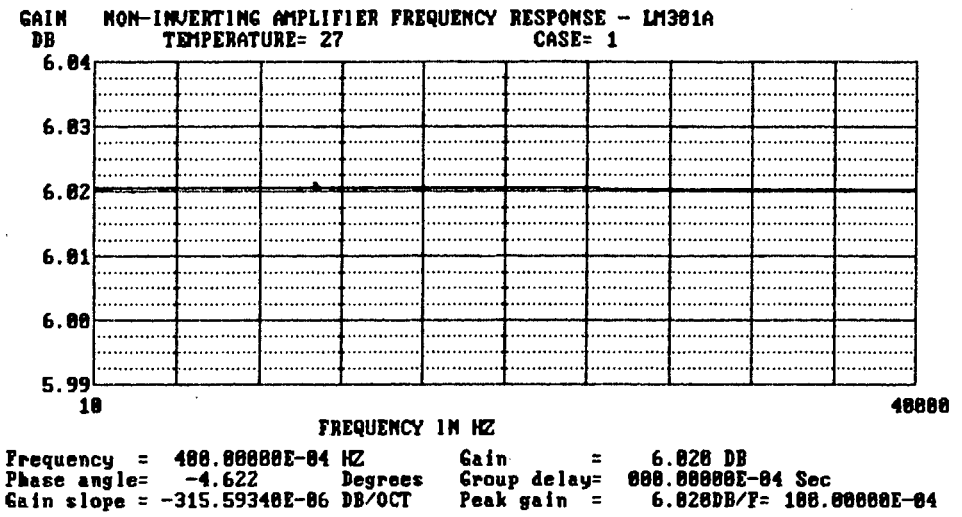


Figure 3.4. Non-inverting amplifier frequency response.

Using the same logic and knowing that the phase output has a sensitivity of 10mV/degree, we conclude that the amplifier introduces a maximum of 0.04 degrees error to the phase data.

(b) **A/D Protection Module**

The A/D convertor has an input range of -5V to +5V. To ensure that the input signal never goes far above or below this range, diode clamps are used on the output of each amplifier. These clamps limit the input signal to the range -5.6V to +5.6V.

Figure 3.5 shows the complete circuit diagram of the two amplifier channels with their protection diodes on the outputs. Note that the series output resistance used to limit the diode current must not exceed 1kΩ. This restriction is due to the fact that the A/D convertor input stage has a relatively low input impedance (22kΩ).

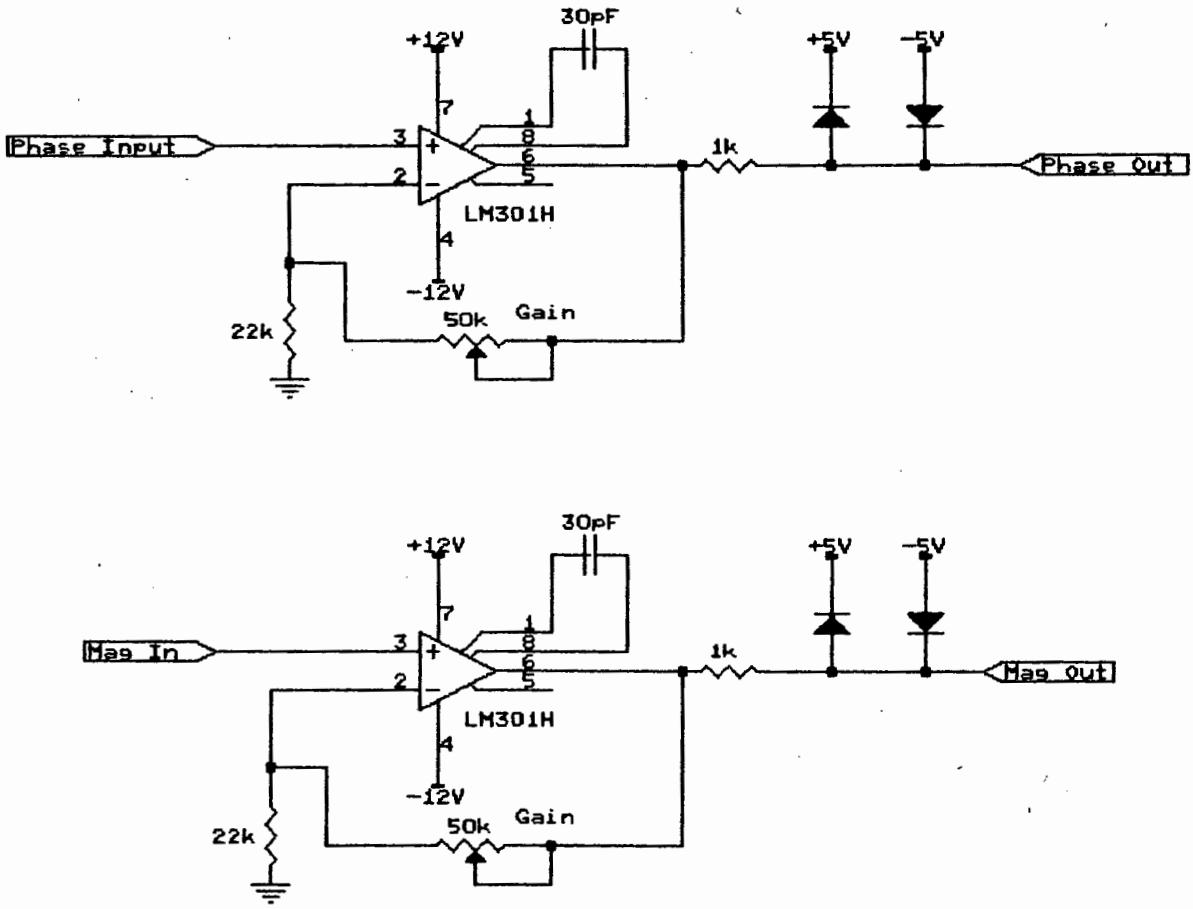


Figure 3.5. Phase and Magnitude Amplifiers.

(c) **Transistor Switching Network**

Previous automatic Network Analyzer systems [3.1, 3.2] use the HP11713A Attenuator/Switch Driver with the GP-IB interface to automate switching of the test set between reflection and transmission measurements.

In this particular system the HP11713A is inadequate because:

(1) The HP8746B has four possible s-parameters and eight possible incident attenuation settings. In other words we require twelve switch drivers and the HP11713A only has two.

(2) To communicate via the GP-IB bus from a Turbo Pascal environment (version 5.00 or earlier) requires the use of the Turbo Pascal "EXEC" command. In practice, it was found that in very large programs, using the "EXEC" command leads to memory shortage problems.

Fortunately, the HP8746B test set has a remote programming facility which makes possible the remote selection of all four s-parameters and all eight incident attenuation settings [3.5]. Remote programming of the test set is selected by closing a contact (pin 17) of a 36-pin rear panel connector to common (pin 18 or 36). The s-parameters are selected by closing two contacts (pin 24 and 6) to common and the incident attenuation settings are selected by closing three contacts (pins 15, 33 and 16) to common.

Figure 3.6 shows the truth table for selecting the s-parameters and Figure 3.7 shows the truth table for selecting incident attenuation.

Parameter to be Measured	Pin 18 or 36 to:		Pushbutton Lit
	PIN 24	PIN 6	
S_{11}	Open	Open	S_{11}
S_{12}	Open	Closed	S_{12}
S_{21}	Closed	Open	S_{21}
S_{22}	Closed	Open (CLOSED)	S_{22}
NOTE: Pin 17 must be shorted to common (18 or 36) to select remote operation.			

Figure 3.6 Remote S-Parameter Selection Truth Table [3.5].

Attenuation	Pin 18 or 36 to:			Pushbutton Lit
	"10 dB bit" Pin 15	"20 dB bit" Pin 33	"40 dB bit" Pin 16	
0 dB	Open	Open	Open	0 dB
10 dB	Closed	Open	Open	10 dB
20 dB	Open	Closed	Open	20 dB
30 dB	Closed	Closed	Open	30 dB
40 dB	Open	Open	Closed	40 dB
50 dB	Closed	Open	Closed	50 dB
60 dB	Open	Closed	Closed	60 dB
70 dB	Closed	Closed	Closed	70 dB
Note: Pin 17 must be shorted to common (18 or 36) and REMOTE-MANUAL switch in REMOTE to select INCIDENT ATTENUATION remotely.				

Figure 3.7 Remote Attenuation Selection Truth Table [3.5].

(c.i) Remote Operation Circuitry

The HP8746B test set specifications [3.5] state that a contact is at a potential of 12 volts and a short-circuit to common will draw 12mA.

In this system, the 8255 PPI is used to switch transistors which in turn switch the contacts to common. Figure 3.8 shows the circuit diagram of the remote selection of s-parameters and incident attenuation. The transistors used are 2N3904's, however any general purpose NPN transistors will suffice provided they are able to handle 12mA.

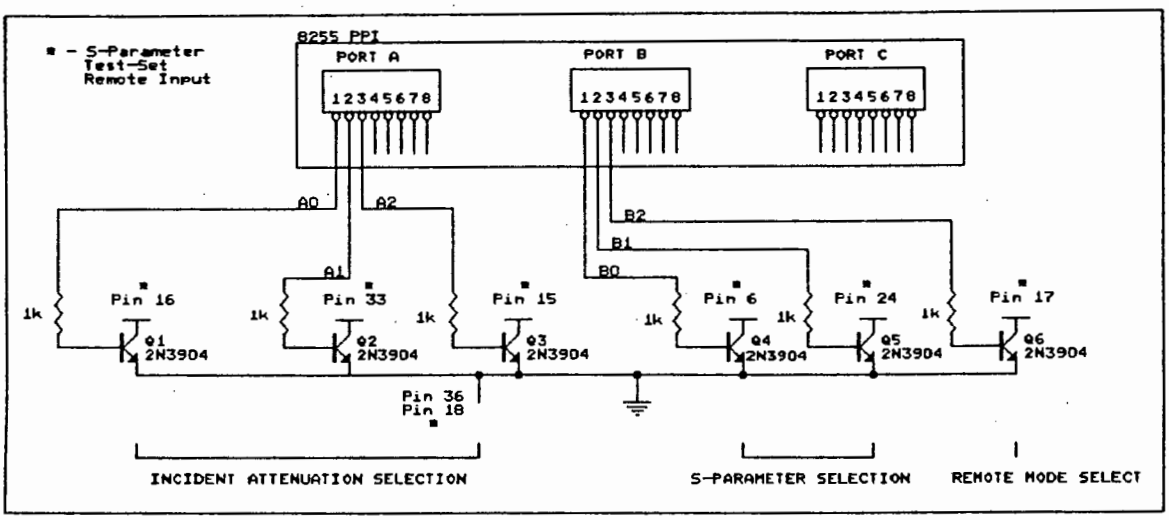


Figure 3.8. Remote selection of s-parameters and incident attenuation.

(d) **Protective Circuitry for 8255 PPI**

The display blanking waveform, POS Z BLANK, is available from the HP8350B sweep oscillator rear panel as a BNC output. A typical POS Z BLANK waveform is shown in Figure 3.9.

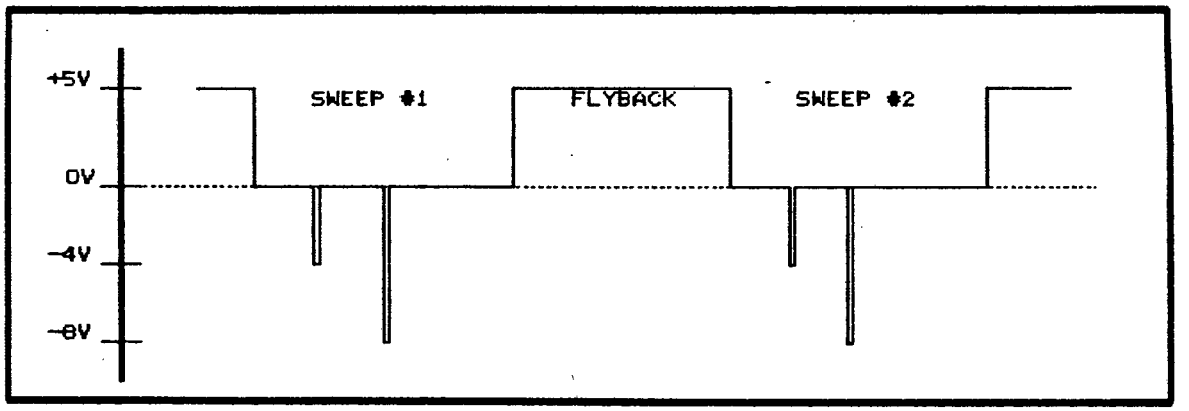


Figure 3.9. A POS Z BLANK Waveform.

During periods that the HP8350B oscillator is sweeping, 0V output is provided. During sweep retrace or flyback periods, the POS Z BLANK waveform is at +5V. Furthermore, -4V and -8V marker z-axis modulation signals are provided for an external display. A bright spot will be displayed at all points of the trace that a -4V or -8V spike is encountered. The -8V spike represents the currently active marker, the one which has last been set/adjusted.

The 8255 PPI needs protection against these negative spikes. A simple diode clamp is sufficient to limit the negative spikes to -0.6V. The diode clamp used is shown in Figure 3.10 below.

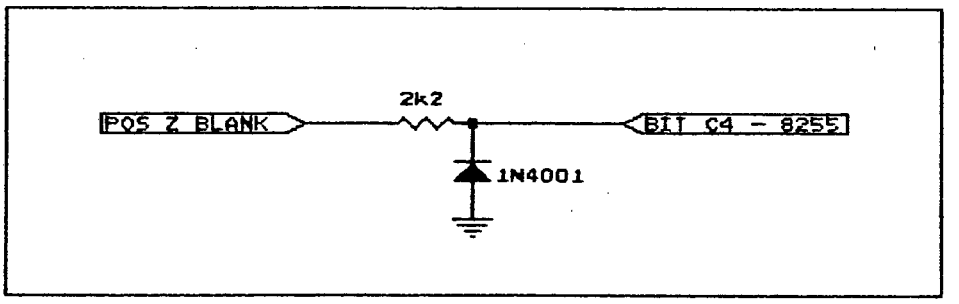


Figure 3.10 Diode Clamp.

(e) **Power Supply**

A power supply is required for the amplifiers and protection circuitry. The power supply must be capable of supplying ±12V and ±5V at 20mA. The power supply used in this system is shown in Figure 3.11.

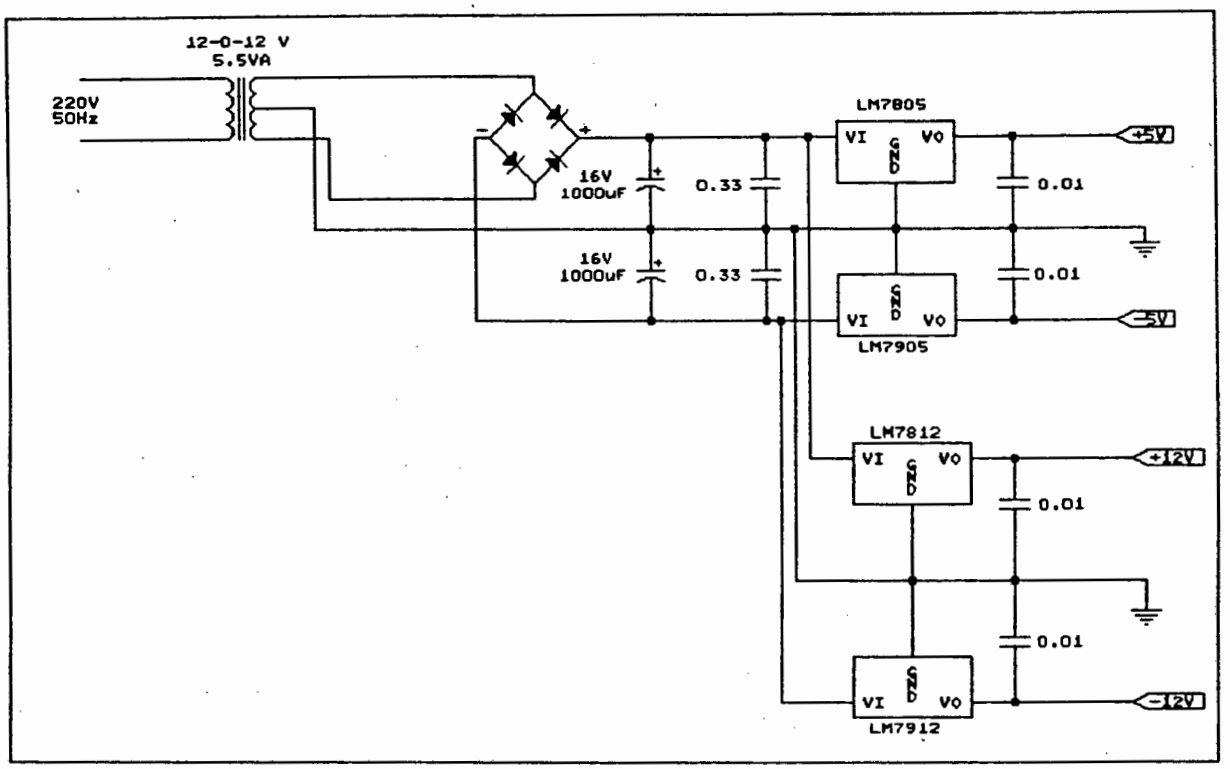


Figure 3.11 Power Supply.

Note that the use of the computer's 12V supply (usually available on the A/D convertor connector) is not recommended because this supply is unprotected against short circuits and is extremely noisy.

A photograph of the constructed analogue preprocessing unit is shown in Figure 3.12.

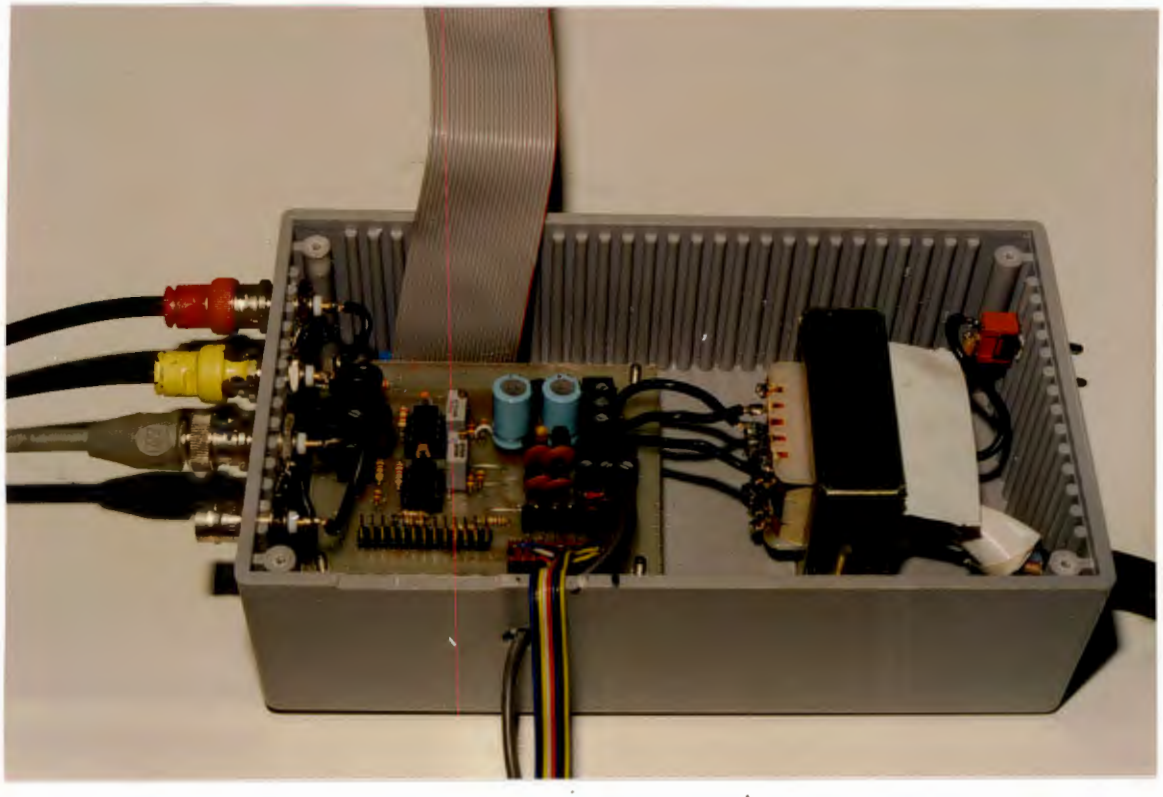


Figure 3.12 Completed Analog Preprocessing Unit.

A photograph of the completed hardware is shown in Figure 3.13.

Figure 3.13. Real-Time Network Analyzer.

Figure 3.14 shows a detailed wiring diagram of the complete measurement system.

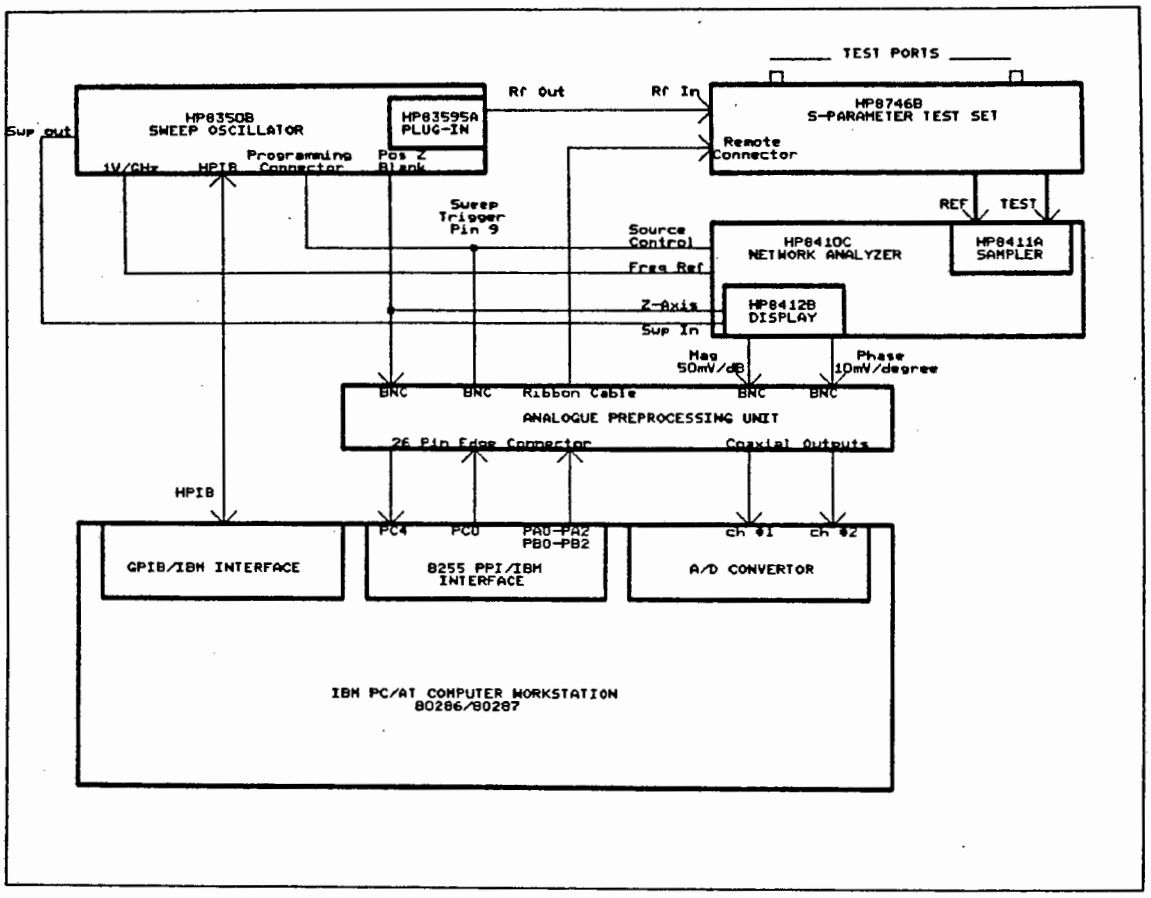


Figure 3.14. System Wiring Diagram.

REFERENCES

[3.1] 8408B Automatic Network Analyzer, 500MHz to 18 GHz, Operating and Service Manual, Hewlett Packard.

[3.2] Automating the HP8410B Microwave Network Analyzer, Application Note 221A, June 1980, Hewlett Packard.

[3.3] Network Analyzer 8410A, Harmonic Frequency Convertor 8411A, Operating and Service Manual, December 1971, Hewlett Packard.

[3.4] 8412B Phase-Magnitude Display, Operating and Service Manual, December 1981, Hewlett Packard.

[3.5] S-Parameter Test Set, 0.5-12.4 GHz, 8746B, Operating and Service Manual, August 1971, Hewlett Packard.

[3.6] 83595A RF PLUG-IN, 0.01 to 26.5 GHz, Operating Information, October 1982, Hewlett Packard.

[3.7] 8350B Sweep Oscillator, Operating Manual, January 1983, Hewlett Packard.

[3.8] PC39 Analogue to Digital and Digital to Analogue Conversion Board with DMA, Operating Manual, Eagle Electric.

CHAPTER 4

SYSTEM SOFTWARE

This chapter deals with the software component of the real-time automatic Network Analyzer measurement system. The software consists of over 4000 lines of TURBO PASCAL source code and performs the task necessary for real-time magnitude and phase data acquisition, control and calibration of the system, and the display of the magnitude and phase data.

A listing of the program source code can be found in Appendix B.

4.1 REAL-TIME DATA ACQUISITION SCHEME

The real-time data acquisition scheme used in this project represents a novel technique for capturing magnitude and phase data as a function of frequency in real-time. Previous automatic Network Analyzer systems [4.1, 4.2, 4.3] implemented non real-time data acquisition schemes.

The scheme used in this thesis consists of eleven major modules;

(1) Interrupt Control.
(2) Control of the sweep trigger.
(3) Monitoring of POS Z BLANK before a sweep begins.

(4) Delay to synchronize sampling and start of sweep.
(5) Starting an A/D Conversion.
(6) A/D conversion delay.
(7) Input from the A/D Convertor.
(8) Delay to ensure 256 samples.
(9) Multi-band sweeps.
(10) Masking of data.
(11) Delay between sweeps.

The data acquisition flowchart is shown in Figure 4.1.

4.1.1 Discussion of Data Acquisition Scheme Modules

4.1.1.1 Interrupt Control

Interrupts zero to seven of the 80286 microprocessor [4.4] are disabled before real-time sampling commences. This step is crucial to the success of any real-time data acquisition scheme. To understand why disabling the interrupts is so important, consider Figure 4.2 which is a representation of a typical real-time data acquisition routine.

The first sample period during which 256 data points are sampled lasts T seconds. An 80286 interrupt occurs during the second sample period and this interrupt takes t seconds to be serviced by the microprocessor. The result of this interrupt is that the second sample period takes (T+t) seconds to complete as compared to the uninterrupted sample period duration of T seconds.

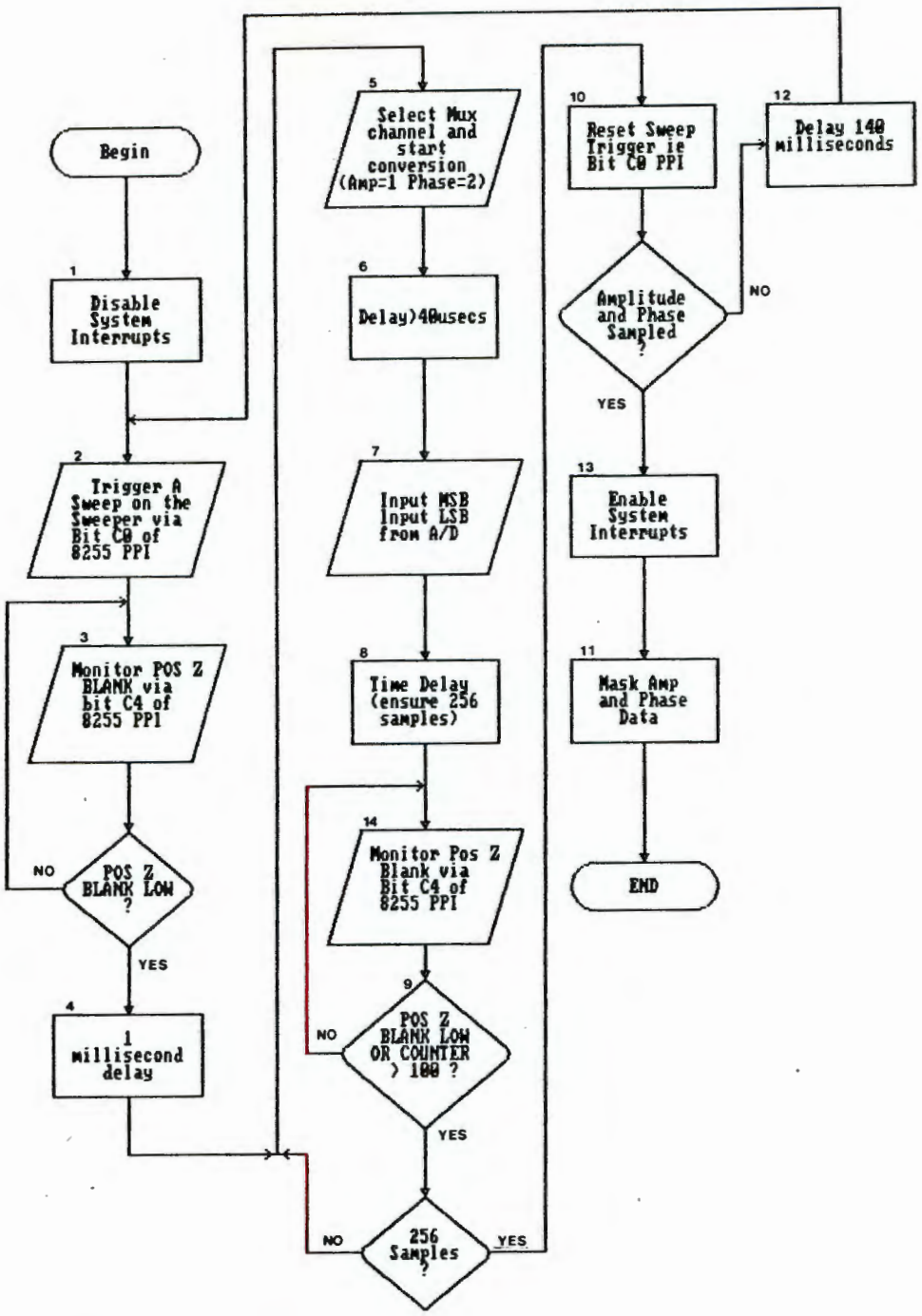


Figure 4.1. Real-Time Data Acquisition Flowchart.

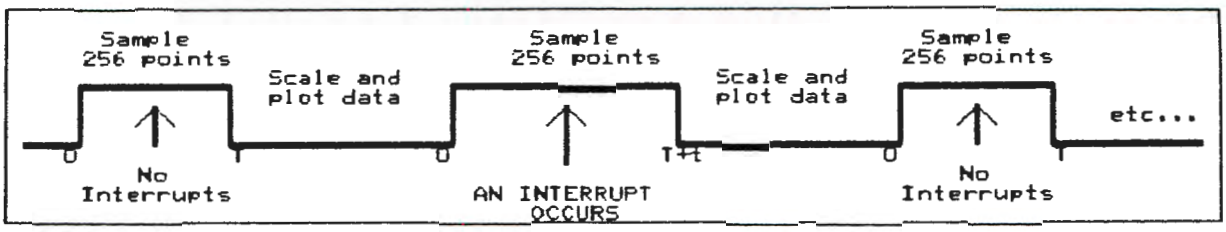


Figure 4.2. Real-time Data Acquisition Timing Waveform.

The crucial point about the interrupts is that they occur randomly and therefore result in random shifts in time of the sampled real-time data. The effects of these random shifts in time are twofold. Firstly, plotting the data in a real-time manner on a computer screen results in random trace jitter.

Figure 4.3 illustrates this effect clearly.

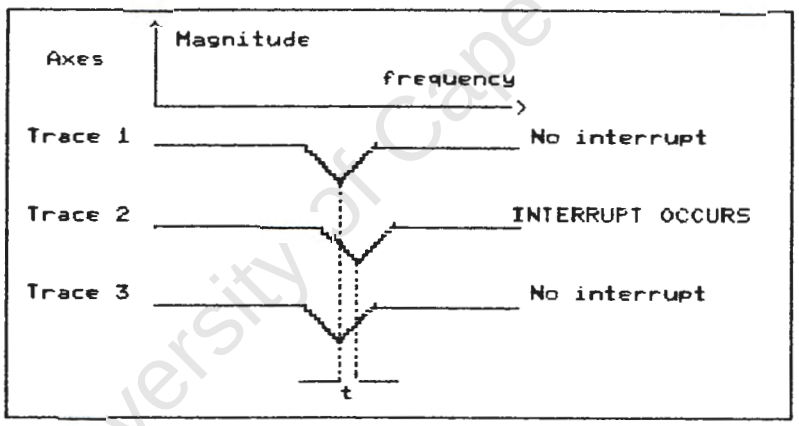


Figure 4.3. Real-Time Trace Jitter due to system interrupts.

Secondly, accuracy enhancement techniques become infeasible since we can no longer exactly relate every measurement data point with every calibration data point, due to the random shifts in data.

The solution is to disable the system interrupts before real-time sampling begins and to re-enable them on completion of the sampling.

4.1.1.2 Control Of The Sweep Trigger

A single sweep is triggered on the HP8350B Sweep Oscillator by raising bit C0 of the 8255 PPI high. After sampling has been completed, the trigger is reset by lowering bit C0.

4.1.1.3 Monitoring of POS Z BLANK before a Sweep begins

In practice it was found that there was a **non-constant** delay between the triggering of a sweep and the actual start of the sweep. It is therefore not practical to commence sampling immediately after triggering a sweep.

After a sweep is triggered, the POS Z BLANK waveform is monitored via bit C4 of the 8255 PPI until it goes low (zero volts). This action solves the problem of non-constant delay discussed above since POS Z BLANK is only low for periods where a sweep condition is true.

4.1.1.4 Delay to Synchronize Sampling and Start of Sweep

In practice it was found that after POS Z BLANK goes low, indicating the start of a sweep, there is a fixed time delay of 1 millisecond before the actual sweep begins.

Figure 4.4 summarizes the timing considerations discussed in this section, section 4.12 and section 4.13.

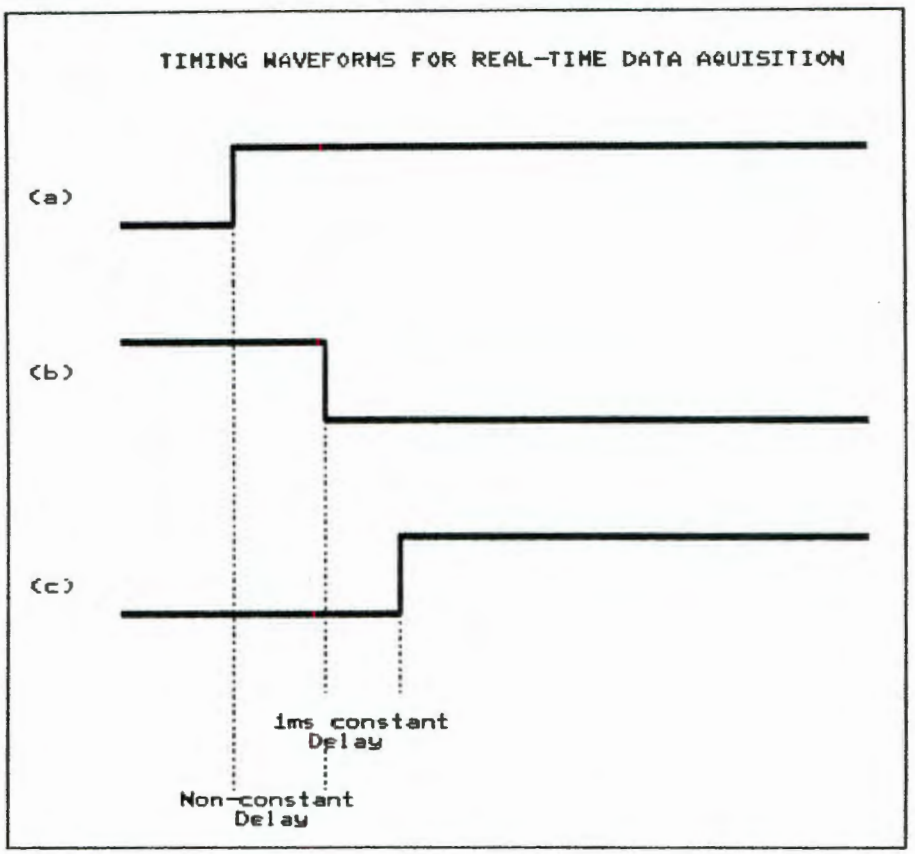


Figure 4.4. Timing waveforms for Data Acquisition. Waveform (a) is Trigger sweep, (b) is POS Z BLANK and (C) is ACTUAL SWEEP START.

4.1.1.5 **Starting An A/D Conversion**

The sequence for an analogue to digital conversion is as follows:

1. Select a channel and clear the software clock (bit C0). Channel 1 is used for the amplitude information and channel 2 is used for the phase information.

2. Select a channel and raise the software clock bit high. This starts an A/D conversion.

4.1.1.6 A/D Conversion Delay

The A/D convertor requires a minimum of 40 microseconds in order to complete a conversion.

A software loop is used to generate this delay and the loop increment is determined on a trial and error basis. A square wave is generated through bit A0 of the 8255 PPI by the following TURBO PASCAL source code.

```
REPEAT
 PORT[$260] := 00000000;                {output a low on A0}
 for i := 1 to INC do inline($90);      {Delay}
 PORT[$260] : = 00000001;               {output a high on A0}
 for i := 1 to INC do inline($90);      {Delay}
UNTIL KEYPRESSED;
```

This square wave is sent into a frequency counter and by adjusting the INC variable in the software loop, the frequency of the square wave can be adjusted. The INC variable is varied until the square wave has a period of greater than or equal to 40 microseconds.

Note that if a computer with a clock speed other than 12MHz with zero wait states is used in the system, this calibration procedure will have to be performed again.

4.1.1.7 Input from the A/D Convertor

The A/D convertor has 12 bits of resolution but only 8 bits can be read in to the computer at any one time via the

PC26's on-board 8255 PPI. Thus, the most significant bit (MSB) is read in separately to the least significant bit (LSB) through the 8255 ports.

4.1.1.8 Delay to ensure 256 Samples

A time delay which is a function of the clock speed of the computer is used to ensure that 256 data points are collected during each sweep time period of 99 milliseconds. In this system the computer clock speed is 12MHz with zero wait states which corresponds to the following software delay.

```
for i = 1  to TIME_DELAY to inline($90);
```

where TIME_DELAY is a constant equal to 110.

The constant, TIME_DELAY, is determined on a trial and error basis in the following manner: The sweep trigger and POS Z BLANK lines are monitored on a logic Analyzer which is set to trigger on the sweep trigger line. TIME_DELAY is adjusted until the sweep trigger waveform goes low immediately after the POS Z BLANK line has finished sweeping as shown in Figure 4.5.

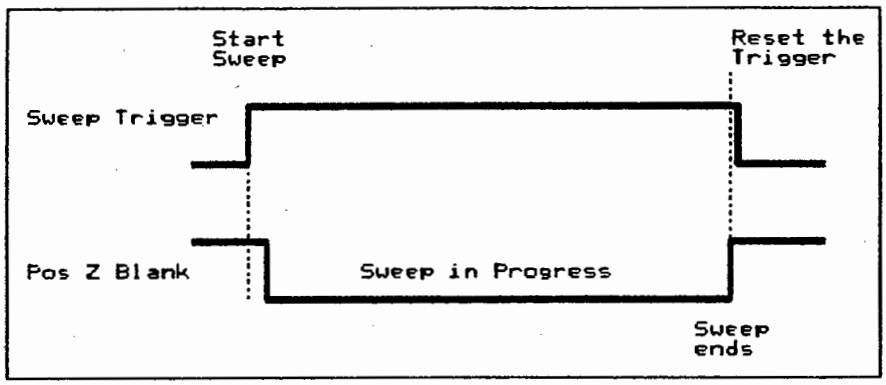


Figure 4.5. Timing waveforms to ensure 256 samples.

This ensures that 256 samples are taken during the 99 millisecond sweep time of the sweep oscillator.

Note that if a different computer is used which has a clock speed other than 12MHz with zero wait states, or the sweep time is changed, the TIME_DELAY constant will have to be redetermined.

4.1.1.9 Multi-band Sweeps

The waveform representing POS Z BLANK shown in Figure 4.5 is a simplification of an actual POS Z BLANK waveform in that it only applies to sweeps within any one of the five power bands described in section 3.2.3.

Figure 4.6 shows the effect of sweeping over multi power bands on an actual POS Z BLANK waveform.

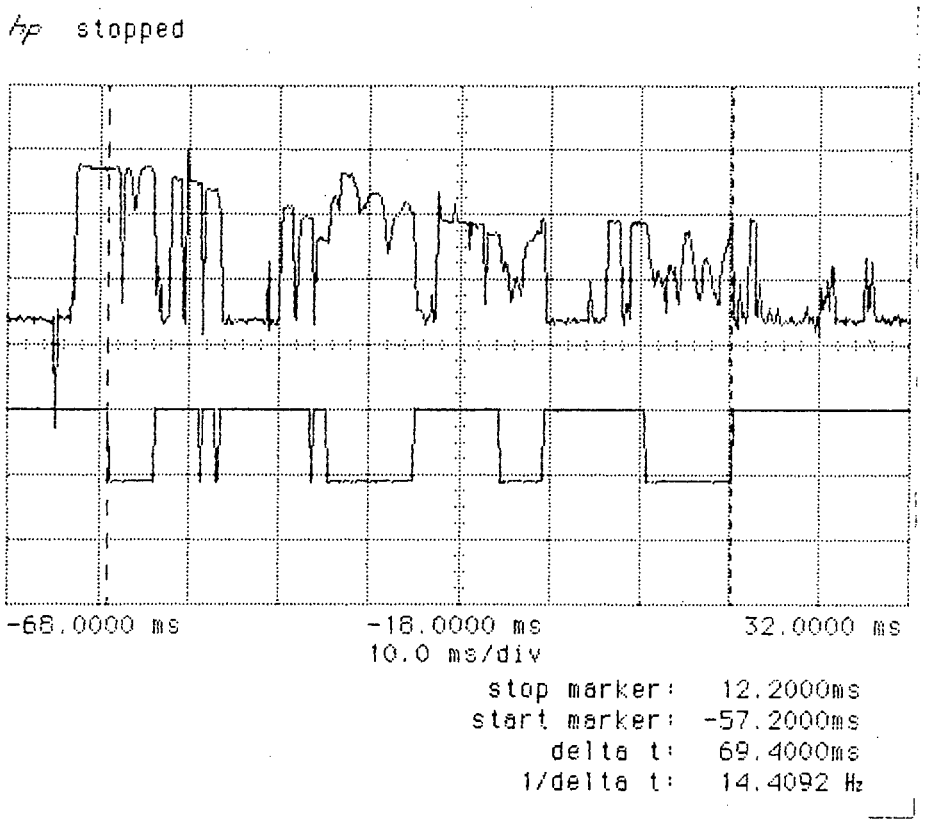


Figure 4.6. The effect of multi power band sweeps on POS Z BLANK.

The top waveform represents the actual magnitude versus frequency data and the bottom waveform is the POS Z BLANK waveform. The two dotted lines represent the start and stop of the sweep which is set for thirty milliseconds. The diagram shows that the POS Z BLANK waveform changes from a high to a low condition several times during the sweep. This variation in the status of the POS Z BLANK line is due to the discontinuities associated with power band switching as described in section 3.2.3.

At all periods of the sweep that the POS Z BLANK line goes high, the amplitude and phase data is invalid and sampling

must be disabled. Since the variations in the POS Z BLANK line are not repeatable, the sampling has to be **intelligent** in order for it to follow these variations.

Figure 4.7 represents the non-repeatability of the status of the POS Z BLANK line during a multi band sweep.

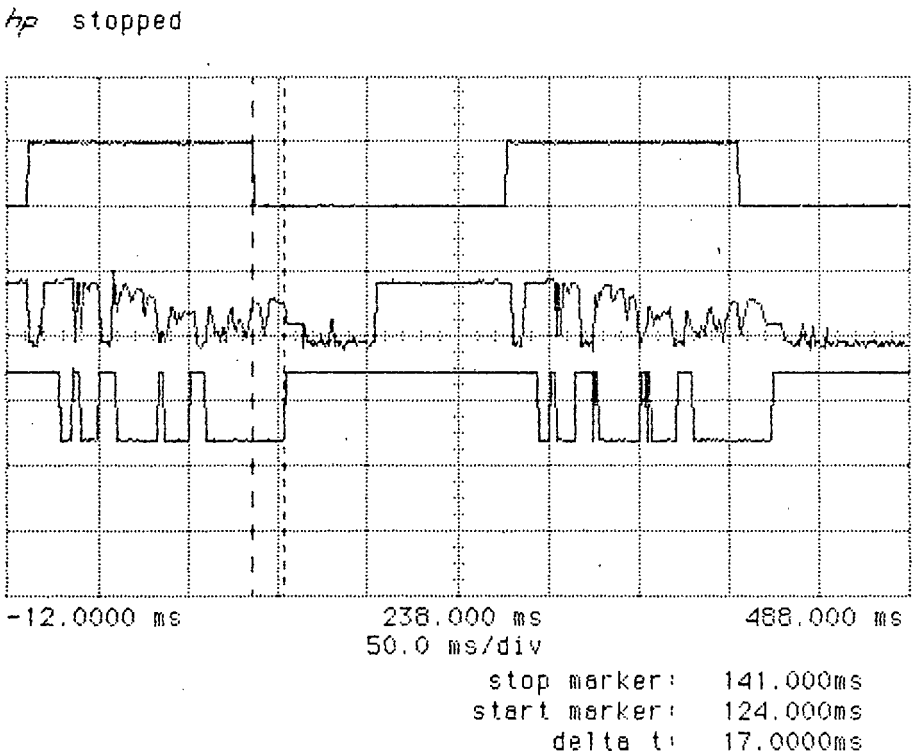


Figure 4.7. The effect of multi band sweeps on POS Z BLANK. The top waveform is Trigger sweep, the middle waveform is Amplitude and the bottom waveform is POS Z BLANK.

Two sequential sweeps are illustrated in the diagram and the POS Z BLANK waveform (bottom waveform) is clearly different for the two sweeps.

The program compensates for the fluctuations in the POS Z BLANK line by monitoring the POS Z BLANK line between every sample. Samples are only taken when POS Z BLANK is low. This compensation is not ideal since the software is unable to "follow" these changes perfectly accurately and this results in timing jitter when performing multi band sweeps.

Careful selection of sweep frequencies, ensuring that measurements lie within any one power band overcomes the problem of timing jitter.

4.1.1.10 Masking Of Data

The PC 26 A/D converter has 12 bits of resolution but only 8 bits can be read into the computer at any one time via the **on-card** 8255. The 4 MSB's of the sample are therefore masked to yield a 12 bit value.

4.1.1.11 Delay between Sweeps

After the amplitude data has been sampled, the phase data must be sampled. The Network Analyzer requires a pause prior to each sweep in order to allow it to lock initially. The duration of the pause is specified in [4.5] as 30 milliseconds. In practice it was found that a longer pause of 140 milliseconds was required to ensure repeatable measurements.

4.2 ALTERNATIVE DATA ACQUISITION SCHEME

The real-time data acquisition scheme discussed in section

4.1 performs well for single power band sweeps but suffers from timing jitter when sweeping over more than one power band.

One alternative data acquisition scheme which was investigated uses the SWEEP waveform in preference to the POS Z BLANK waveform to overcome the problem of switching discontinuities associated with multi power band sweeps. The relationship between a sweep waveform and a POS Z BLANK waveform is shown in Figure 4.8. During periods of the sweep that POS Z BLANK goes high, the sweep waveform deviates from it's characteristic ramp shape and maintains a constant voltage level.

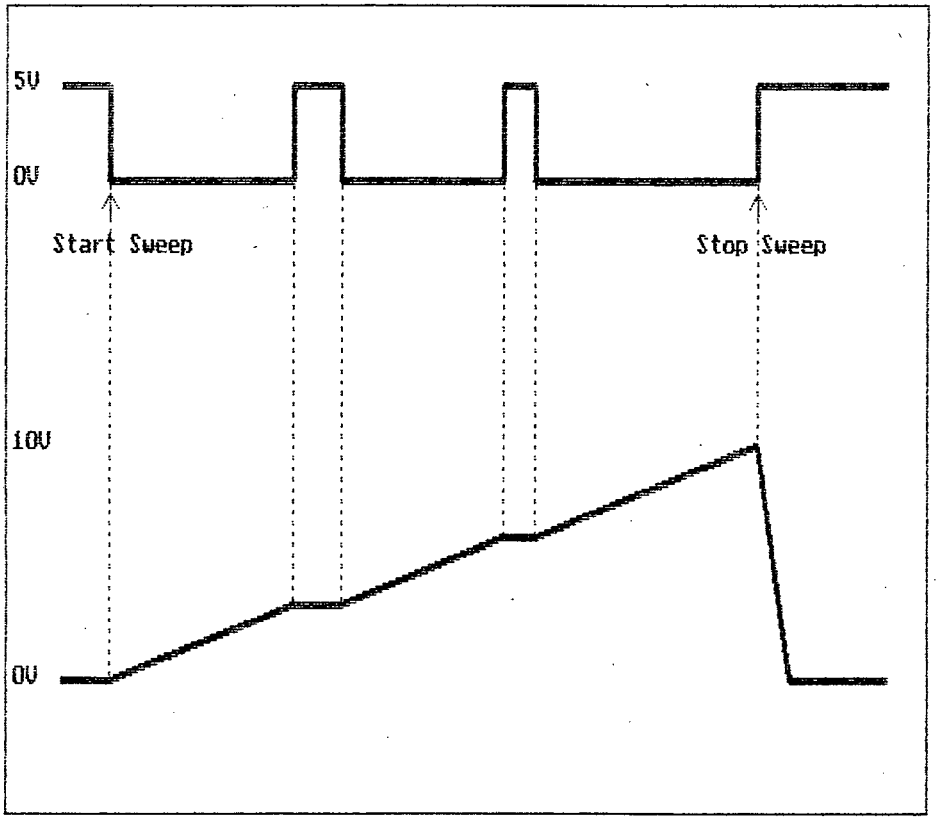


Fig 4.8. Relationship between POS Z BLANK and Sweep waveforms.

The idea behind this data acquisition scheme is to sample the SWEEP waveform through the A/D Convertor and to use the actual voltage level of these samples to decide on whether to sample the magnitude or phase data. Assuming 256 samples are to be taken during a sweep, a sample must be taken at 39.0625 millivolt increments of the sweep waveform. This will ensure that 256 samples of magnitude or phase are taken by the time the sweep waveform reaches it's final value of 10 volts.

This data acquisition scheme was implemented but was proved to be unsatisfactory due to the fact that the sweep waveform did not consistently vary between zero and ten volts. In practice the sweep waveform was found to deviate by up to ±1 volt of it's initial and final values.

4.3 REPRESENTATION OF PHASE DATA

Phase information is represented on the HP8412B display in a ±180 degree format. This format as well as a continuous phase representation format is shown in Figure 4.9.

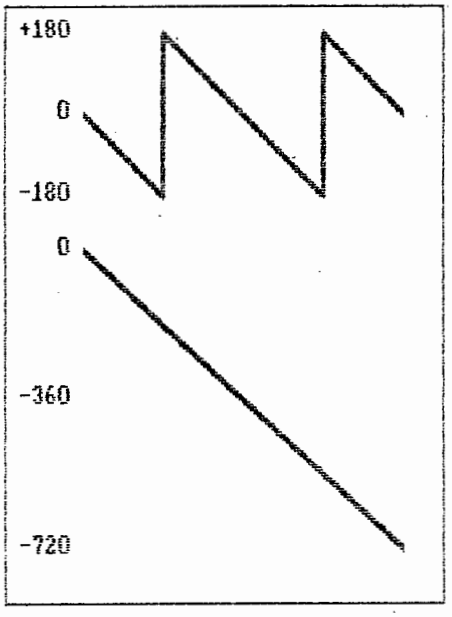


Figure 4.9. Phase formats. The top and bottom representations are equivalent.

The latter representation is a more suitable means for displaying, storing and processing phase information.

For example, consider the removal of the phase response of the test cables from a measurement. If the phase responses are in continuous format, a simple subtraction of phase responses will yield measurement data without the effect of the test cables. If however, the phase responses were in a ±180 degree format, they would have to be converted to a continuous phase format before the test cables phase response could be removed.

A flowchart of the procedure used to convert phase from a ±180 format to a continuous format is given in Figure 4.10.

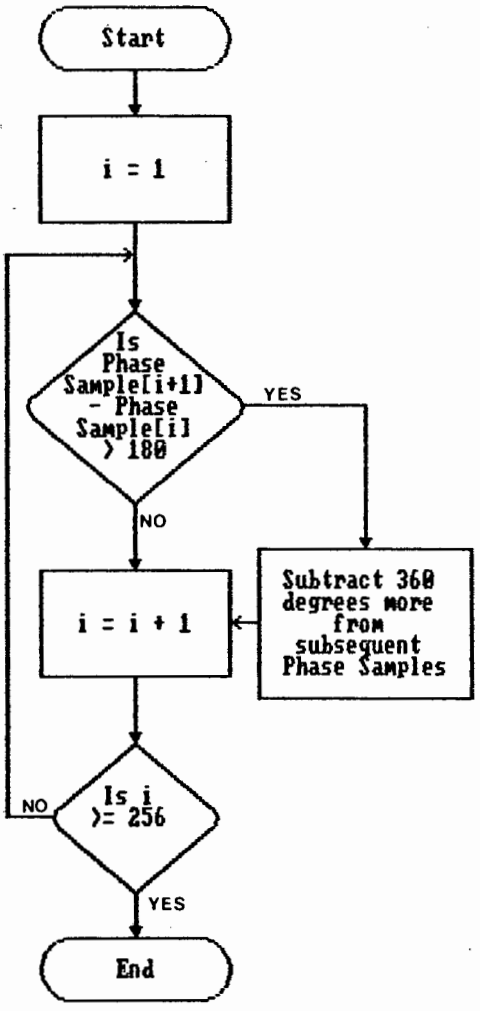


Figure 4.10. Phase Conversion Flowchart.

4.4 CONVERSION OF RELATIVE MEASUREMENTS TO ABSOLUTE MEASUREMENTS

The displays available on the HP8410C manual Network Analyzer require the user to remember the reference values in order to interpret any given measurement in terms of absolute values.

The system software of the measurement system described in this thesis automatically incorporates the reference values into subsequent measurements, providing real-time displays with vertical and horizontal axes scaled in absolute values.

It achieves this conversion by prompting the user to set the HP8410C Network Analyzer test channel gain to 55dB during system initialization, and uses this setting as it's reference level for all subsequent measurements.

4.5 USING THE HPIB BUS WITH TURBO PASCAL

The HPIB card used in this system is supplied with the HP82990A Command Library Software. This software provides the routines necessary for communication between several different high-level languages [4.6] and the HPIB card. Unfortunately TURBO PASCAL is not one of the languages that is supported.

A lower level of communication than that which is available using the HP82990A software is possible using the HPIB Peripheral Driver [4.6].

The Peripheral driver is designed specifically for communication between computers and HP printers and plotters. The HP8350B sweep oscillator is configured as a printer in the measurement system enabling the programming of all of it's front panel settings via the computer.

There are two steps in the installation of the peripheral driver. These steps are:

1. The files HPIBMODE.COM and HPIB.SYS must be copied onto the root directory from which the program is being executed.

2. The system's CONFIG.SYS file must be edited or created if one does not already exist. The following code must be added to the CONFIG.SYS file;

   HPIBMODE HPIB = 719 2

   The purpose of this code is to configure a default HPIB system device. The number 719 refers to the address of the HP8350B sweep oscillator which is therefore the default HPIB system device.

   The number 2 which follows the address is a timeout variable specified in seconds. This timeout variable stands for the length of the delay before **peripheral-not-ready** is assumed and it's inclusion prevents system "lockup".

The TURBO PASCAL code required to program any of the HP8350B sweep oscillators front panel settings is as follows:

```
ASSIGN(GPIO,'HPIBDEV');          {open a disc file, HPIBDEV}
REWRITE(GPIO);                   {initialize the file}
WRITE(GPIO,STRING);              {output a command to HPIB}
CLOSE(GPIO);                     {close the file}
```

The variable, **STRING**, contains the command that is to be programmed via the HPIB. A full list of the possible HPIB commands can be found in [4.7].

4.6 12 TERM ERROR CORRECTION IN REAL-TIME

In practice, the implementation of full 12-term error correction in real-time was found to be infeasible. There are two reasons for this.
Firstly, since 12-term correction requires the measurement of all four s-parameters in order to correct for any one s-parameter, the HP8746B s-parameter test set must be remotely switched four times before each sweep.
The continuous switching of the s-parameter test set coaxial switches represents excessive wear on the switches.

Secondly there are switching transients which occur each time the s-parameter test set is switched. The total settling time required to allow the decay of the transients was found in practice to be around five seconds. This time plus the computational time required for full 12-term correction represents a total time in excess of eleven seconds per sweep - far too long to be considered real-time.

4.7 PROGRAM DESCRIPTION

The program is written in an extensive **pull down** menu format characteristic of modern commercial software packages. The different options in any menu can be scrolled through by means of the up and down arrow keys on the computer keyboard and the highlighted option can be

selected by pressing the ENTER key. This format makes the software easy to use and reduces the risk of operator error.

The basic flowchart of the program logic is shown in Figure 4.11.

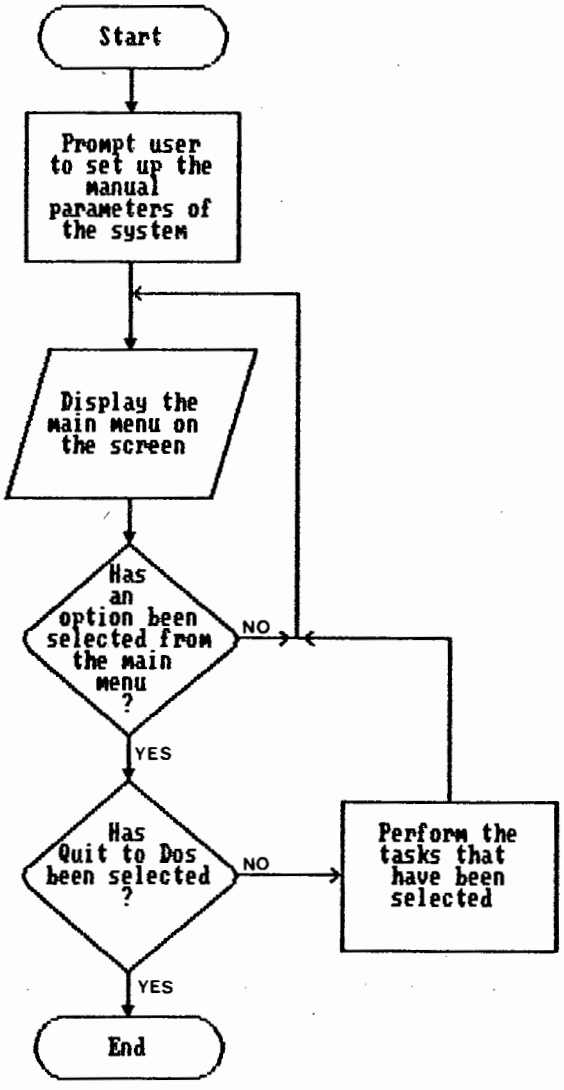


Figure 4.11. Flowchart of the Program Logic.

The main menu which is displayed on the screen after the manual system parameters have been set up provides seven options.

These options are:

(i) Frequency Range/RF level.
(ii) S-Parameter/Attenuation.
(iii) Frequency domain display.
(iv) Full error correction.
(v) Text files.
(vi) Amplifier Calibration.
(vii) Exit to DOS.

A photograph of the main menu is shown in Figure 4.12.

The block located at the top right hand corner of the main menu shows the current start and stop frequencies, RF level and s-parameter. When the program is run for the first time the current s-parameter defaults to S22, the RF level defaults to 5.0dBm and the program interrogates the sweeper to determine the default start and stop frequencies.

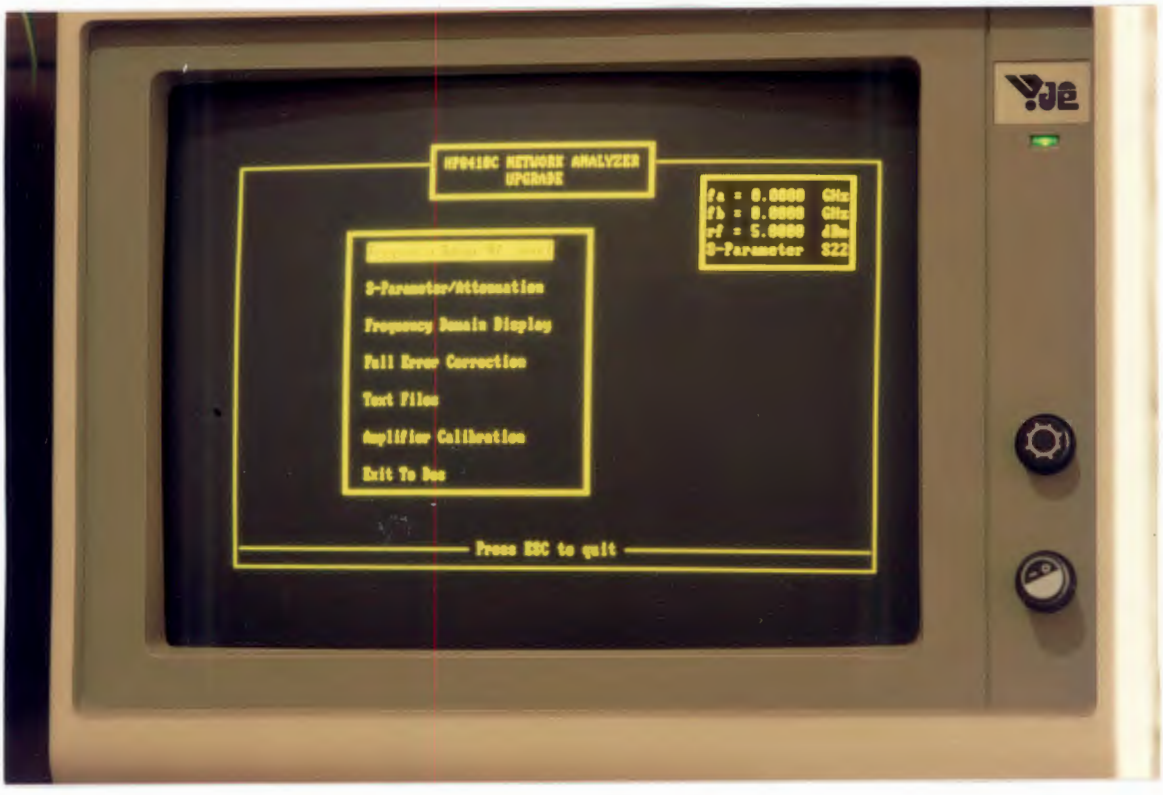


Figure 4.12. Program Main Menu.

4.7.1 Manual System Parameters

The Network Analyzer measurement system described in this thesis is not entirely automated. Several system parameters need to be set manually by the user before the system can be used. When the system is initialized, the user is prompted to set up the manual system parameters according to Figure 4.13.

System Component	Parameter Setting
HP8410C Network Analyzer	Test Channel Gain = 55 dB Sweep Stability = CW Source = Normal
HP8412B Phase/Mag Display	BW = 10 kHz
HP8746B Test Set	Mode = Remote(Rear of set)

Figure 4.13. Manual System Parameters.

Note that if any of these parameters are changed during program execution, the measurements will be invalidated.

4.7.2 Description Of Main Menu Options

4.7.2.1 Frequency Range/RF Level

This option enables the selection of the start and stop frequencies of the sweep oscillator and the RF level of the HP83595A plug-in respectively.
When this option is selected, a second menu is pulled down over the main menu as shown in Figure 4.14.

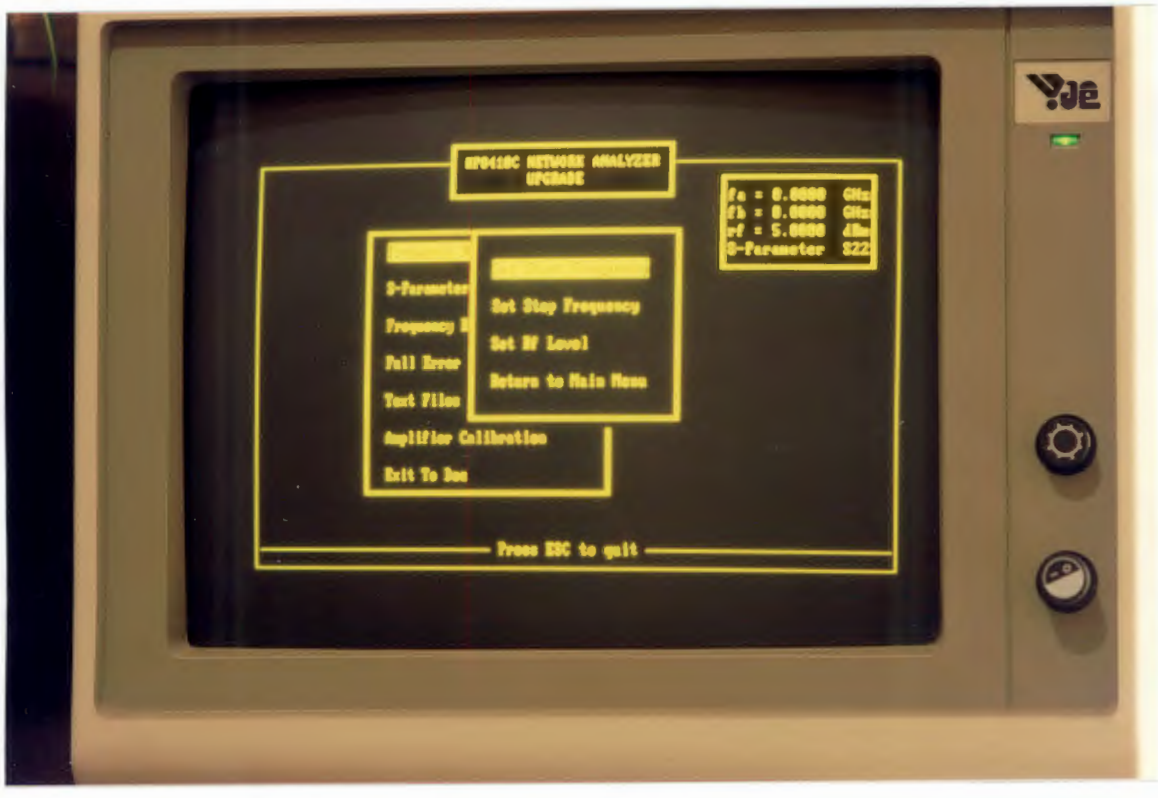


Figure 4.14. Program Sub-menu level one.

Pressing the ESC key or selecting the "RETURN TO MAIN MENU" option returns the program to the main menu. Selecting any of the other three options will bring up a third menu as shown in Figure 4.15.

Figure 4.15. Program sub-menu level two.

New values for start and stop frequencies and RF level can now be entered in via the computer keyboard and these values will be automatically programmed on the HP8350B sweep oscillator and HP83595A plug-in respectively.

4.7.2.2 S-Parameter/Attenuation

The required s-parameter and incident attenuation can be programmed on the HP8746B S-parameter test set by selecting

this option. Figure 4.16 shows the format of this menu option.

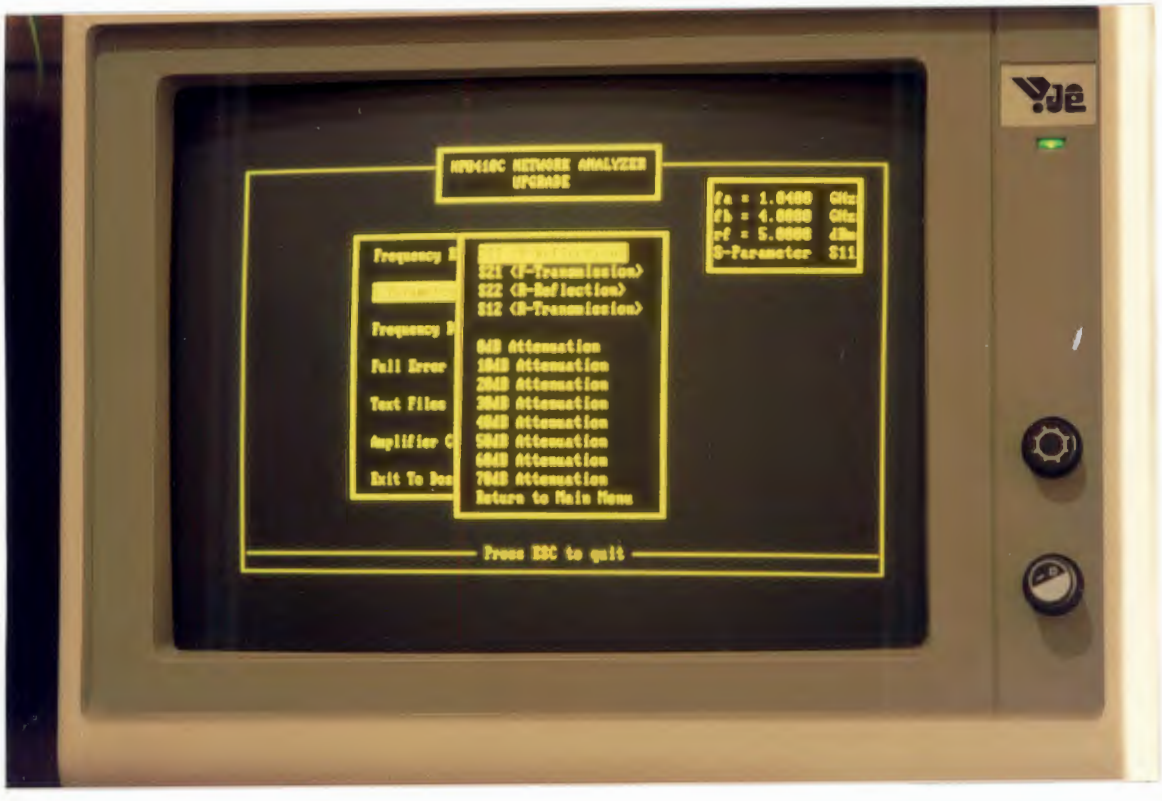


Figure 4.16. S-Parameter and Incident Attenuation menu.

4.7.2.3 <u>Frequency Domain Display</u>

Selecting this option brings down a second menu which gives the user the option of selecting a Smith Chart or a rectangular display. Both options provide a real-time swept frequency plot on the computer screen.

The Smith Chart option plots a Smith Chart on the computer screen as shown in Figure 4.17.

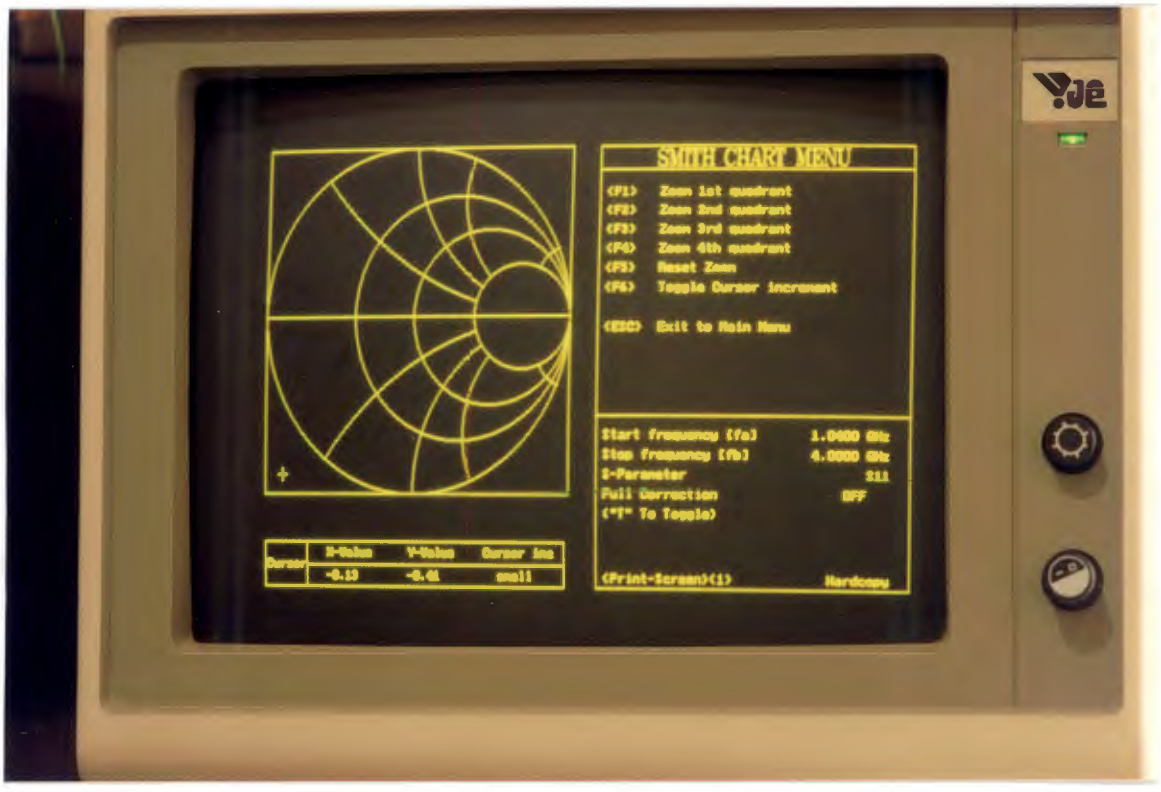


Figure 4.17. Real-Time Smith Chart Display.

The cursor at the bottom left hand corner of the Smith Chart can be moved around the chart using the four arrow keys on the computer keyboard. The cursor's complex impedance co-ordinates are displayed in the table below the Smith Chart.

The menu which appears to the right of the chart as shown in Figure 4.17 further enhances the versatility of the software by making available several options.

These options include a **zooming** facility which allows the user to **zoom** into any one of the four quadrants of the

chart and an adjustable cursor increment which facilitates fast cursor movement around the Smith Chart.

The **zooming** facility of the Smith Chart is illustrated in Figure 4.18 which represents a **zoom** into the second quadrant of the chart.

Figure 4.18. Real-Time "Zoomed" Smith Chart Display.

Selection of the rectangular display option "pulls down" a third menu providing the option of magnitude versus frequency, phase versus frequency or magnitude and phase simultaneously versus frequency displays.

Figure 4.19 shows a magnitude versus frequency rectangular real-time plot and Figure 4.20 shows a dual magnitude and phase display.

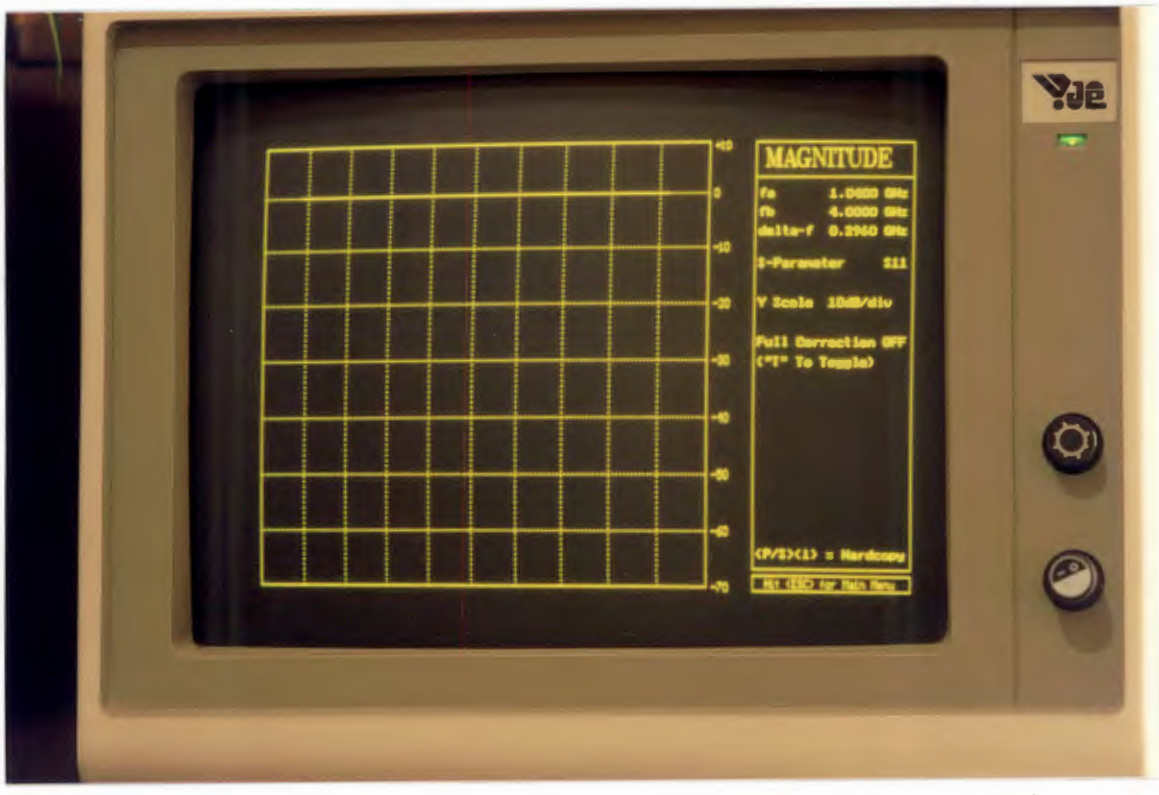


Figure 4.19. Real-Time Magnitude versus Frequency Display.

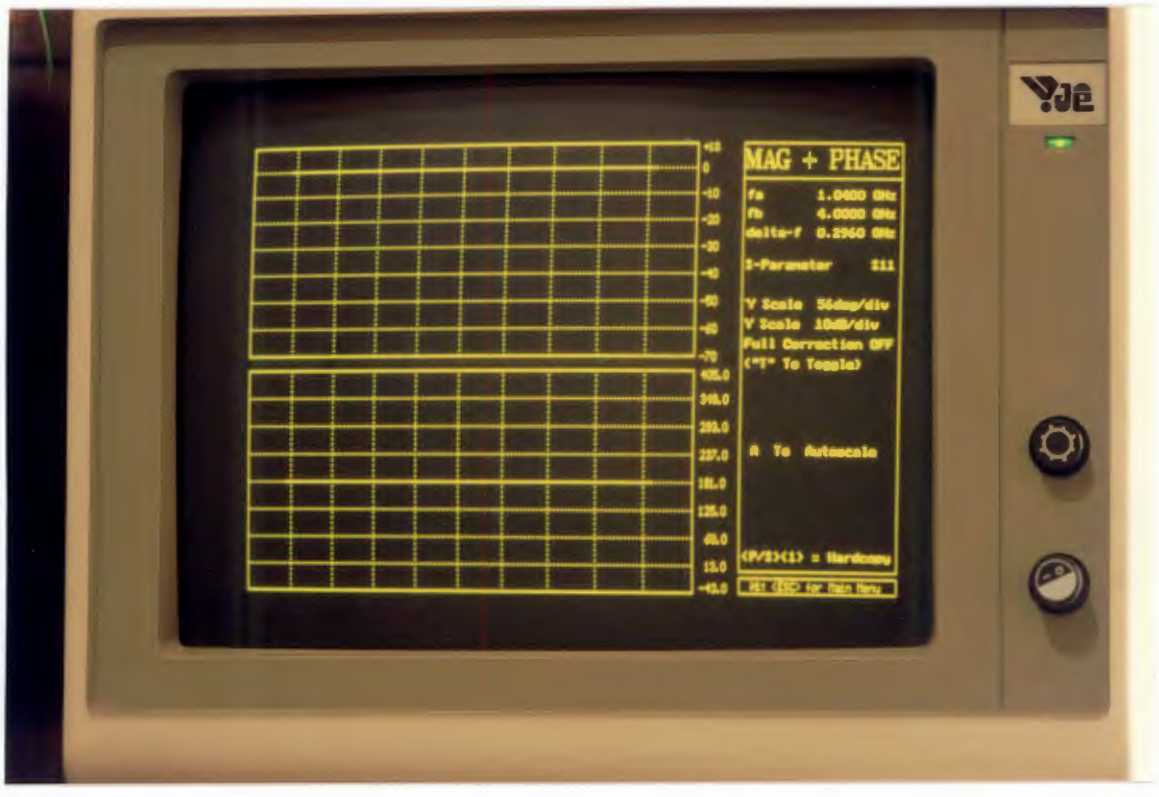


Figure 4.20. Dual Real-Time Display.

The magnitude versus frequency display has a fixed vertical scale of 10 dB/division irrespective of whether a single or dual display is selected.

The phase versus frequency display computes it's own vertical scale on the basis of the initial maximum and minimum phase values. Thereafter, the vertical scale can be forced to be recomputed by pressing the "A" key on the computer keyboard.

This autoscaling feature of phase plots is demonstrated in Figures 4.21 and 4.22.

Figure 4.21 consists of two diagrams. The diagram on the left represents the phase response with the system configured for no-connection. A frequency response-only-calibration has been performed and the phase response is therefore zero degrees at all frequencies.

Now a DUT is connected between the test ports and the diagram on the right shows the resultant phase response. Note that the plot does not fit on the screen.

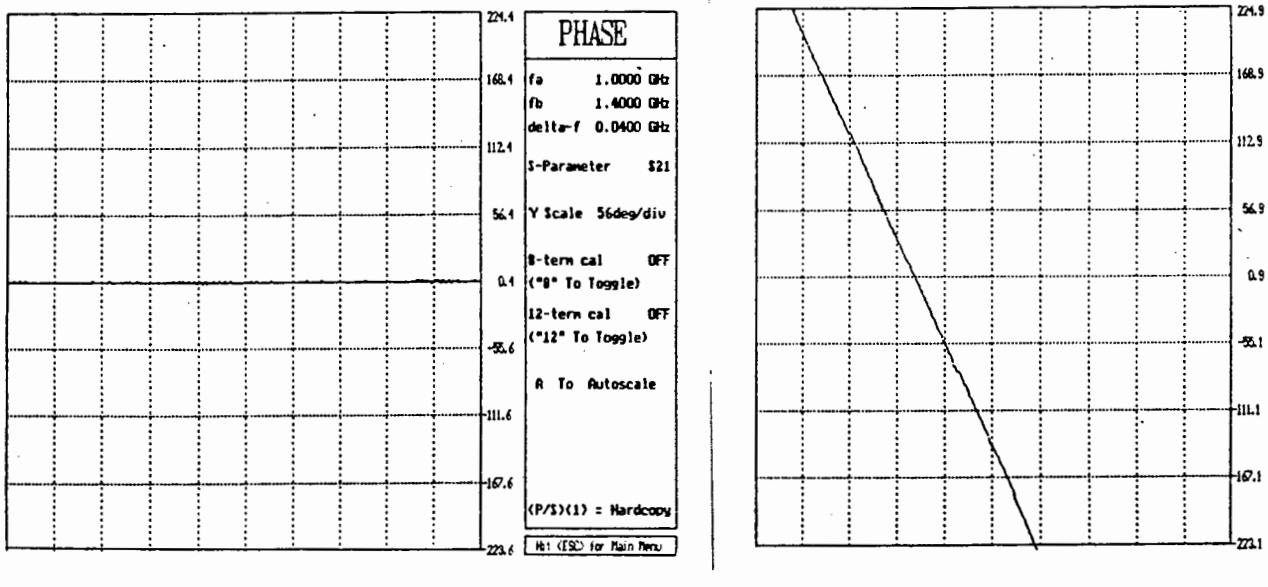


Figure 4.21. Autoscaling feature of Phase plots - Stage one.

The phase response shown in Figure 4.21 is now **autoscaled** and the resultant phase response is shown in Figure 4.22.

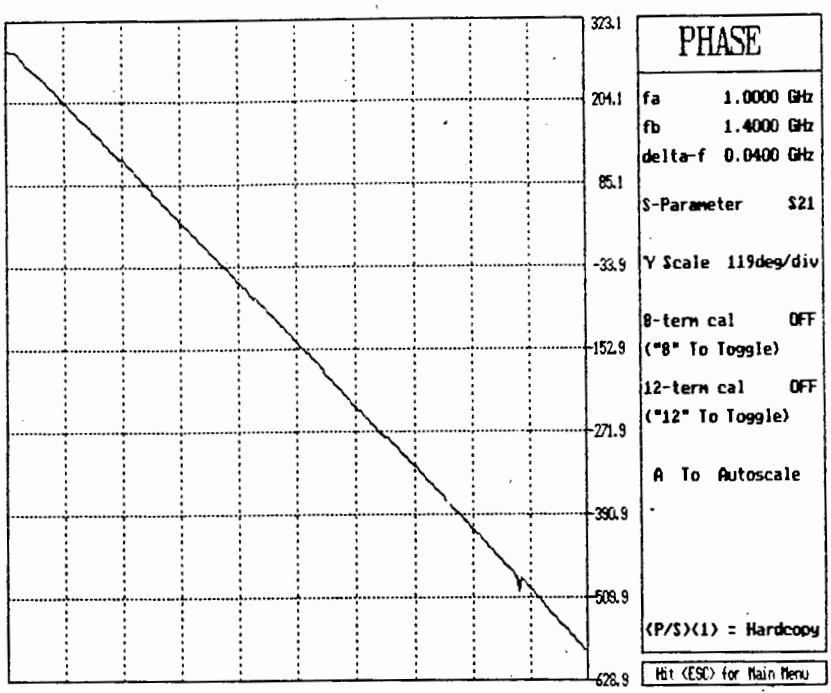


Figure 4.22. Autoscaling feature of Phase plots - Stage two.

An improvement to the software would be a facility which enables a user to enter in his/her own value for the magnitude or phase vertical scale.

Hardcopies of all rectangular and Smith Chart displays are possible by pressing the Print Screen key followed by the number 1 key.

4.7.2.4 Full Error Correction

(a) Frequency-Response-Only-Correction

A frequency-response-only or normalization calibration is

automatically performed irrespective of whether the full error correction option is selected or not. This calibration is useful for removing the effects of test cables and adapters from measurements. Measurement speed with frequency-response-only calibration is approximately one trace per 400 milliseconds.

(b) **8-Term Correction**

The full error correction option in the main menu provides full 8-term error correction, as discussed in section 2.3.2.1., for reflection measurements in real-time. The microwave standards used to calibrate out the systematic errors are an open circuit, a short circuit and a fixed load.

Measurement speed with full error correction on, is approximately one trace per 1.8 seconds.

The status of full error correction can be toggled by pressing the "T" key, providing a calibration has already been performed.

Note that in the frequency-response-only and 8-term calibration schemes discussed above, the data acquisition for the various different microwave standards used to quantify the systematic errors is performed in real-time. This represents a huge saving in time when compared to non real-time systems [4.1, 4.2, 4.3], since we can now perform a full error correction calibration in under 30 seconds.

4.7.2.5 **Text Files**

This option allows the currently active s-parameter to be written to a text file on disc.

The currently active s-parameter refers to the s-parameter that was last displayed on either a Smith Chart or a rectangular plot. The data written to the text file will be fully error corrected only if a full correction calibration has previously been performed **and** the corrected s-parameter has subsequently been displayed on a Smith Chart or rectangular plot.

This facility allows the storage of data on disc and permits possible comparison between measured data and theoretical data. For example, CAD packages such as the microwave simulation package, TOUCHSTONE, enable a user to design and simulate a microwave network on a computer. The typical procedure is to design and simulate a network on the computer and then to build and test the device on a Network Analyzer. If the test results are written to a file, the theoretical and measured values can be compared simultaneously on the computer and conclusions can more easily be drawn.

The format in which the data is written to disc is shown in Figure 4.23.

S-Parameter = S11

freq [GHz]	Amp [dB]	Phase [deg]
1.00000	-0.700000	388.220
1.00156	-0.700000	388.080
1.00312	-0.700000	387.940
1.00469	-0.700000	387.520
1.00625	-0.700000	386.680
1.00781	-0.700000	386.400
1.00937	-0.700000	386.540
1.01094	-0.700000	385.980
1.01250	-0.700000	385.840
1.01406	-0.700000	385.440
1.01562	-0.700000	385.840
"	"	"
"	"	"
"	"	"
"	"	"
1.39219	-0.700000	288.780
1.39375	-0.700000	287.800
1.39531	-0.700000	286.540
1.39687	-0.700000	286.260
1.39844	-0.460000	286.680
1.40000	-0.700000	286.820

Figure 4.23. S-Parameter Text file output format.

4.7.2.6 Amplifier Calibration

Figure 4.24 shows the signal chain from the HP8412B display to the A/D plug-in card in the computer.

The A/D converter quantizes it's input into 1 of 4096 possible levels and the software interprets each level in terms of dB's or degrees.

Anywhere along the signal chain, variances or drift may, and do, occur with time, leading to inaccurate measurements.

The amplifier calibration option in the main menu overcomes this problem by computing a new amplification factor for the signal chain shown in Figure 4.24 each time this option is selected.

The basic procedure is as follows:

1. The system is configured for transmission measurements and the user is prompted to connect a thru between the test ports and to set the test channel gain equal to 55dB.

2. The thru magnitude response is sampled and the data is stored.

3. The user is prompted to set the test channel gain equal to 45dB.

4. The thru magnitude response is sampled again and the data is again stored.

5. The difference between the data obtained in steps 2 and 4 corresponds to a change of 10dB at the input to the signal chain. An amplification factor can now easily be determined.

6. The process is now repeated for the phase channel and a phase amplification factor is determined.

Both amplification factors are written to a file on disc and are read by the program during initialization. Each

time this option is chosen, new amplification factors are written over the old ones.

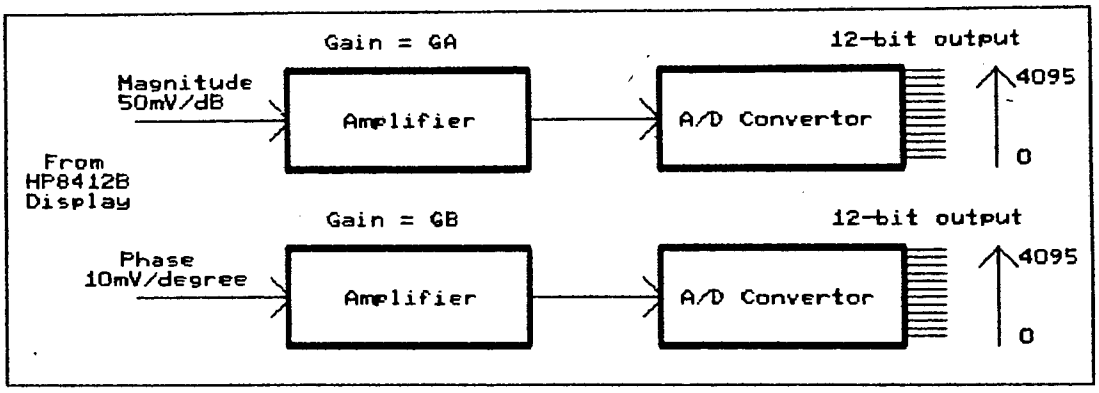


Figure 4.24. Magnitude and Phase Data Signal Chain.

4.7.2.7 **Exit To Dos**

Selecting this option returns the user to the DOS environment and re-configures the Network Analyzer into a manual mode.

4.2.3 **Running The Program**

The following files are required to run the program:

(i) MENUFIN.EXE.
(ii) MENUFIN.OVR.
(iii) HPIBMODE.COM.
(iv) HBIB.SYS.
(v) HARDCOPY.COM.
(vi) PRINTER.DEF.
(vii) CONFIG.SYS.

The file, HARDCOPY.COM, must be run before the main program, MENUFIN.EXE, to enable the graphics printout facilities within the program.

The memory requirement of the computer is 640 kilobytes.

REFERENCES

[4.1] Automating the HP8410B Microwave Network Analyzer, Application Note 221A, June 1980, Hewlett Packard.

[4.1] 8408B Automatic Network Analyzer, 500 MHz to 18 GHz, Operating and Service Manual, August 1982, Hewlett Packard.

[4.3] Williams, W.L., Compton, R.C., Rutledge, D.B., "Computer Automation and Error Correction for a Microwave Network Analyzer", IEEE Trans. on Instrumentation and Measurement, Vol 37, No 1, March 1988, pp. 95-100.

[4.4] IBM Pc-AT Technical Reference Manual, March 1984, pp. 1.10-1.11.

[4.5] Network Analyzer 8410A, Harmonic Frequency Convertor 8411A, Operating and Service Manual, December 1971, Hewlett Packard.

[4.6] HP82990A Command Library Software, Reference Manual.

[4.7] Quick Reference Guide for the HP8350A Sweep Oscillator, September 1980, Hewlett Packard.

CHAPTER 5

SYSTEM PERFORMANCE

A number of basic measurements were performed using the real-time automatic Network Analyzer system discussed in this thesis. This chapter presents the results which were obtained from these measurements.

5.1 MEASUREMENT OF AN ATTENUATOR

The s-parameter test set test port was terminated with a 6dB attenuator and the measurement system was configured for reflection measurements. The theoretical return loss for this configuration over the specified attenuator frequency range is 12dB. Figures 5.1 and 5.2 show the measured return loss for the 6dB attenuator from 3 GHz to 5 GHz without and with full error correction respectively.

Note that the error corrected measurement is almost constant at 12dB over the entire frequency range. The non error corrected measurement displays a ripple component of around 2dB on either side of the expected 12dB return loss.

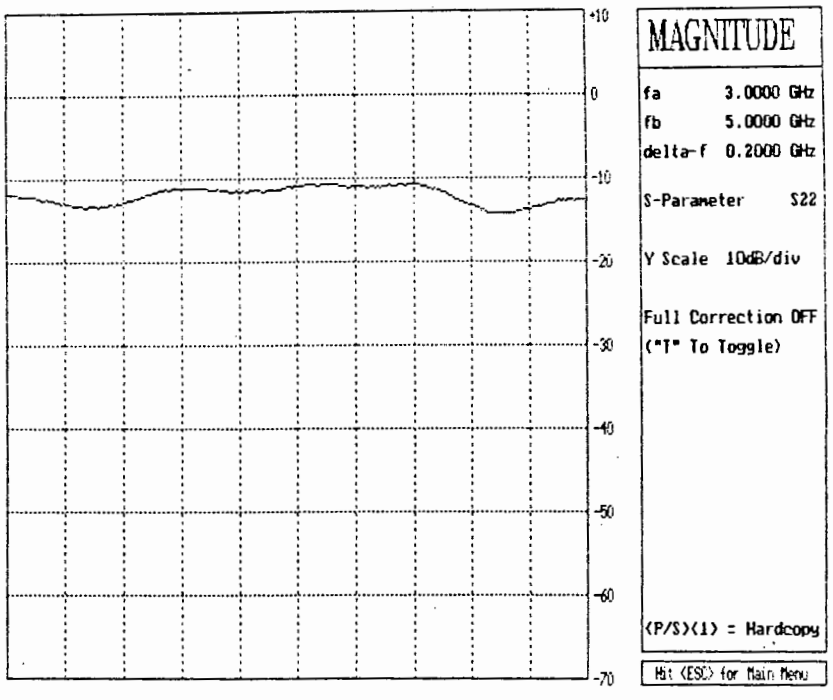


Figure 5.1. 6dB attenuator Return Loss without error correction.

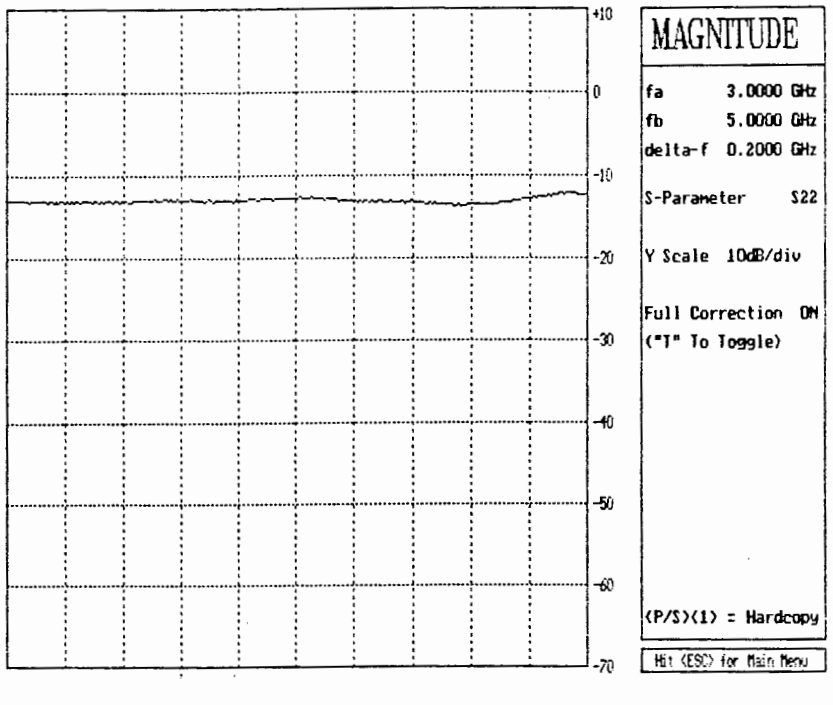


Figure 5.2. 6dB attenuator Return Loss with error correction.

5.2 MEASUREMENT OF A COAXIAL CABLE

The transmission characteristics of a 1.3 metre coaxial cable terminated with a 10dB attenuator were measured with the real-time system.

Figure 5.3 shows the results obtained. The insertion loss is approximately 11dB which corresponds to the 10dB attenuator insertion loss plus the insertion loss of the coaxial cable.

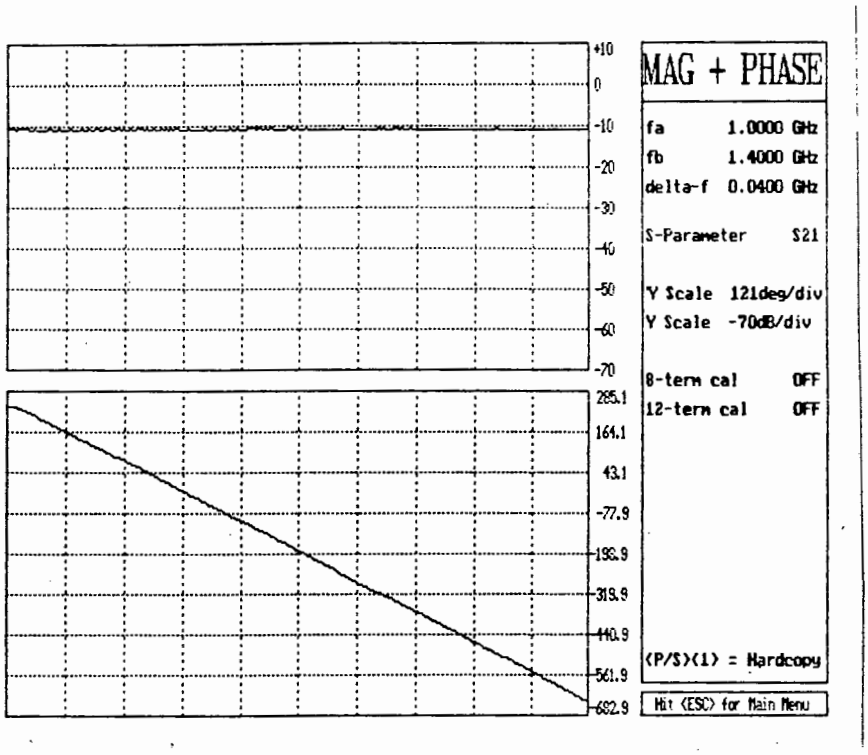


Figure 5.3. Coaxial Cable Magnitude and Phase Insertion Loss.

Note that, as expected, the phase response of the coaxial cable changes linearly as a function of frequency. Note too, that the phase is represented in a continuous format rather than a ±180 degree format.

5.3 MEASUREMENT OF AN OPEN CIRCUIT

The reflection coefficient of an **ideal** open circuit has a magnitude of 0dB and a phase shift of 0 degrees.
Figure 5.4 shows the measured return loss of an open circuit with full error correction off.

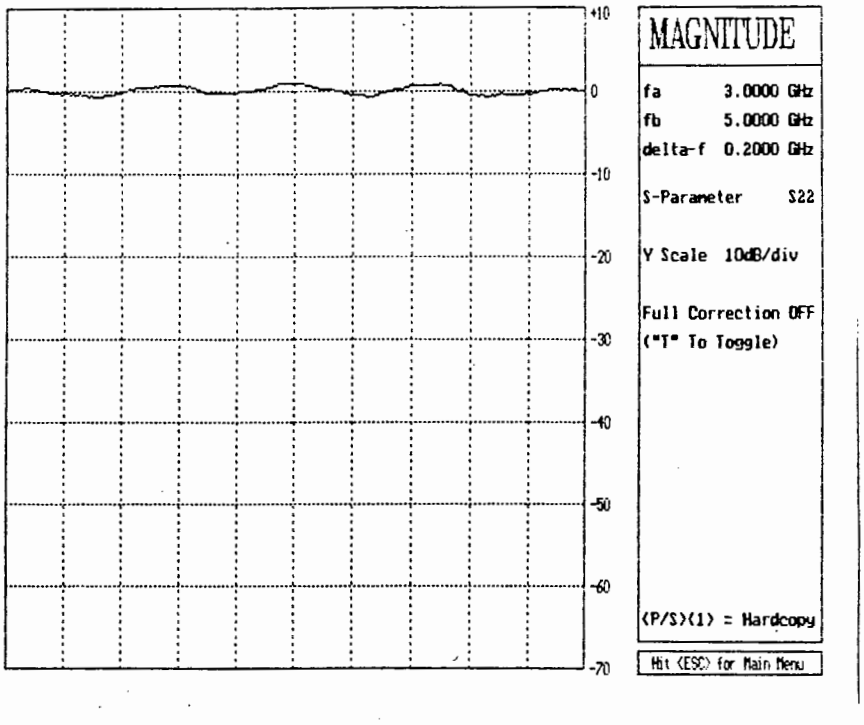


Figure 5.4. Open Circuit Return Loss without Error Correction.

The systematic errors inherent in the measurement system, result in approximately 2dB of ripple in both directions of the expected 0dB return loss.

Figure 5.5 shows the effect of full error correction on the measurement.

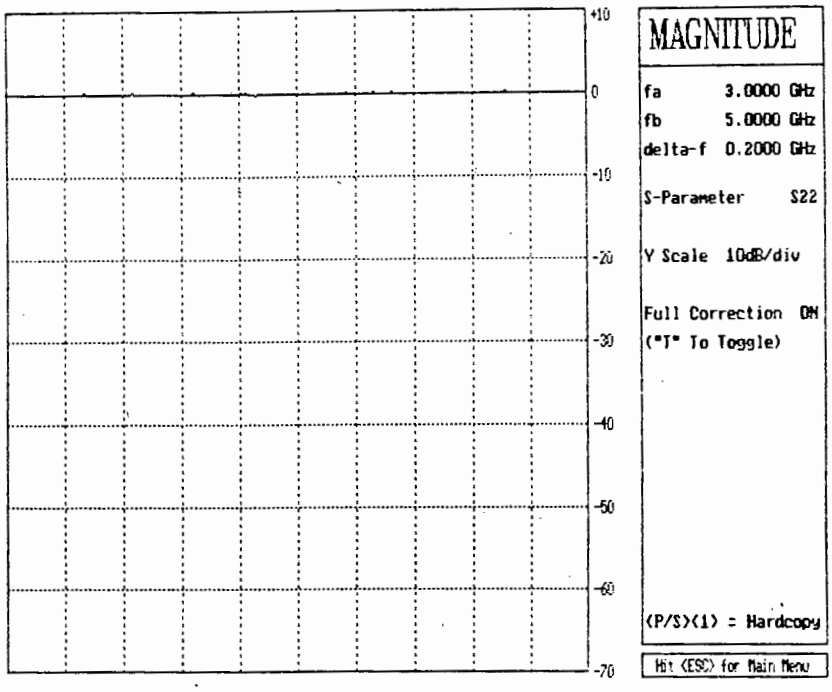


Figure 5.5. Open Circuit Return Loss with Error Correction.

The measured return loss is now very close to the expected return loss since the systematic errors have been characterized and removed.

Similarly, the difference in the phase response of an open circuit, without and with full error correction, are shown in Figures 5.6 and 5.7 respectively.

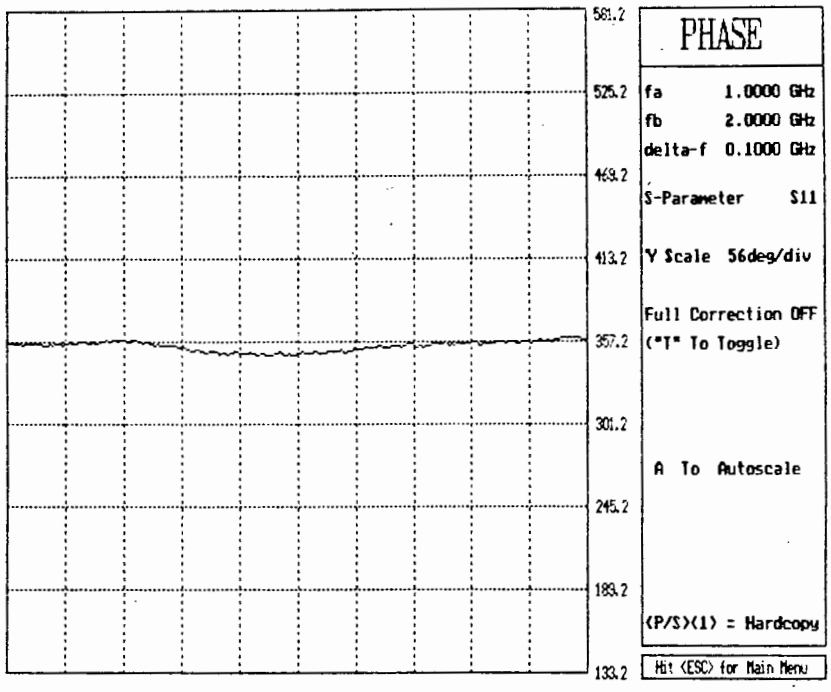


Figure 5.6. Open Circuit Phase Response without Error Correction.

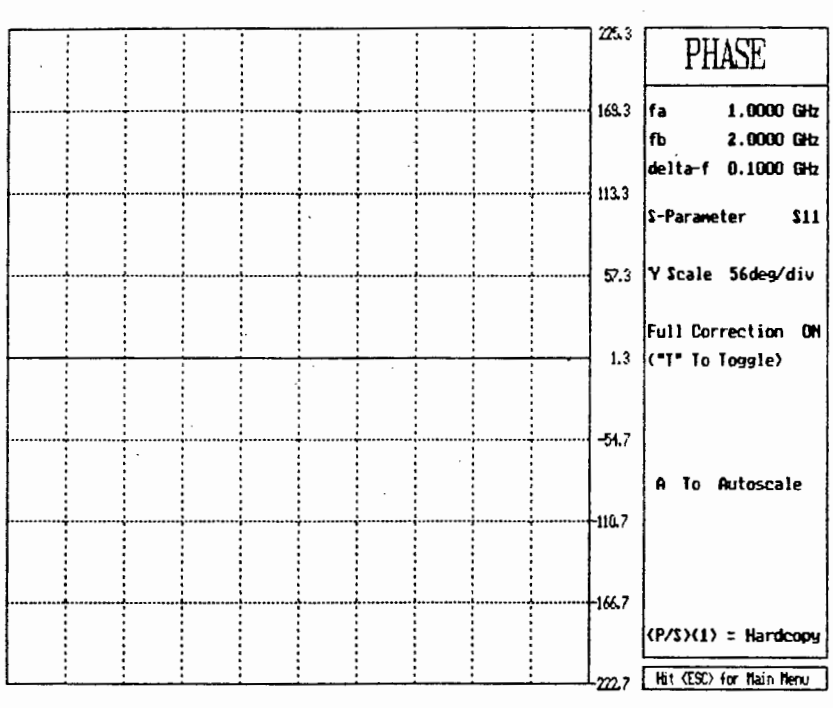


Figure 5.7. Open Circuit Phase Response with Error Correction.

5.4 MEASUREMENT OF A 50 OHM TERMINATION

On a Smith Chart display, a 50 ohm termination corresponds to a small spot at the centre of the chart.
The systematic errors in the system result in this spot being spread out in all directions as shown in Figure 5.8.

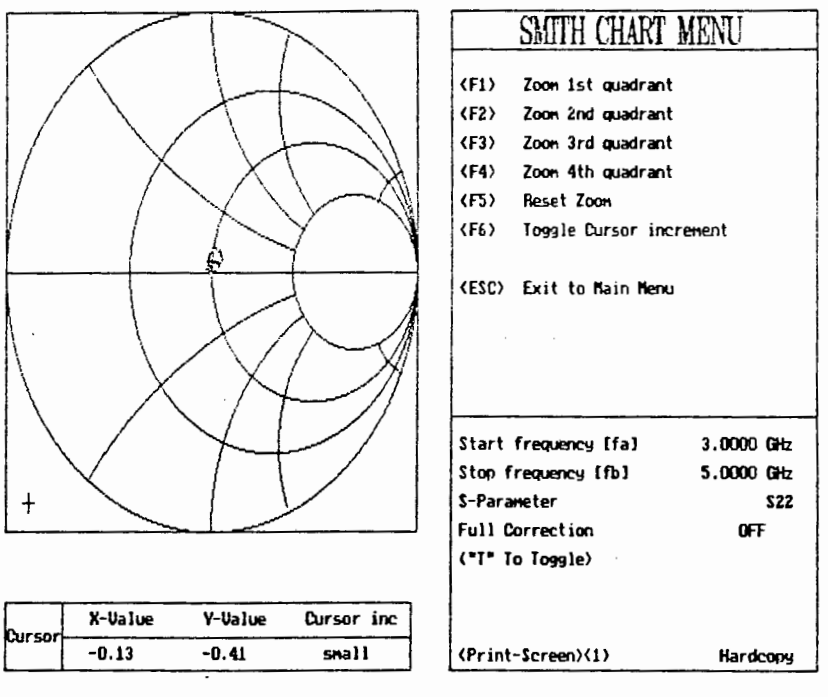


Figure 5.8. Smith Chart Display of 50ohm load without Error Correction.

Figure 5.9 shows the effect of full error correction on a Smith Chart measurement. The spot at the centre of the chart in now hardly visible and the systematic errors have effectively been removed from the measurement.

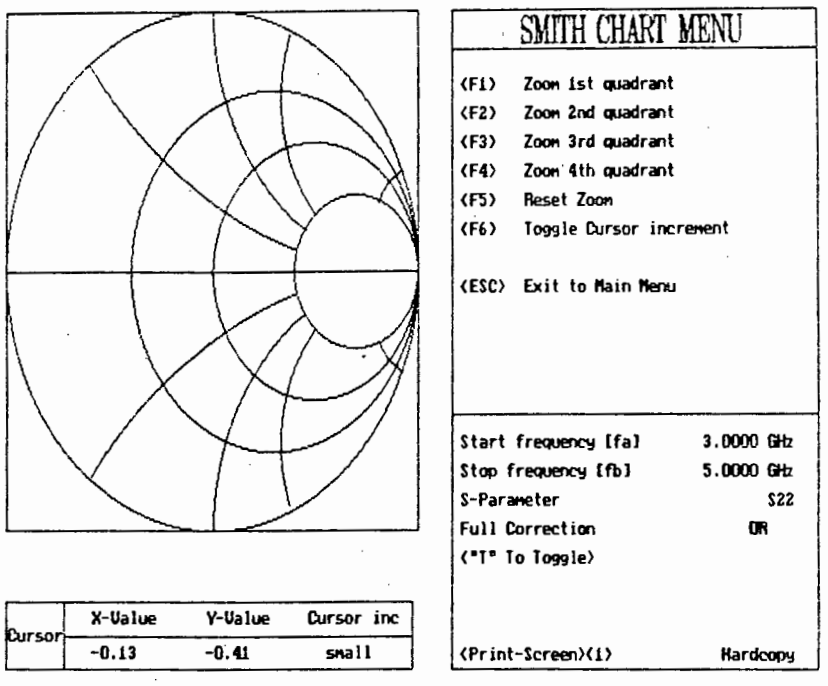


Figure 5.9. Smith Chart Display with Error Correction.

CHAPTER 6

CONCLUSIONS

In view of the findings of this thesis, the following conclusions may be drawn;

1. The hardware for a real-time automatic Network Analyzer measurement system, based on the HP8410C Network Analyzer, has been developed.

2. Over 4000 lines of TURBO PASCAL software has been developed to provide the measurement system with the following features:

   (a) The ability to switch instantaneously between real-time Smith Chart and rectangular displays, and lists of values.

   (b) The ability to display frequency-response-only and eight term error corrected measurements in real-time.

   (c) The ability to obtain hardcopies of all measurements.

   (d) The ability to write measurement data to a text file for subsequent storage or processing.

3. Automatic real-time measurements are only feasible for frequency ranges that lie within any one of the five power bands of the HP83595A RF plug-in.

4. Full 12-term vector error correction in real-time is not feasible.

APPENDIX A

TECHNICAL DATA ON THE 8255 AND PC26 INTERFACE CARDS

8255 Interface Board

Port Pinouts

Port 1 - Component Side

PA_0	1	PA_1	26
PA_2	2	PA_3	25
PA_4	3	PA_5	24
PA_6	4	PA_7	23
PC_0	5	PA_1	22
PC_2	6	PA_3	21
PC_4	7	PA_5	20
PC_6	8	PA_7	19
PB_0	9	PB_1	18
PB_2	10	PB_3	17
PB_4	11	PB_5	16
PB_6	12	PB_7	15
GND	13	GND	14

Port 0 - Pinouts as above

Addresses

Port 0	260H - 263H	Port A	260H
	(368H - 36BH)	Port B	261H
		Port C	262H
		Control	263H

Port 1	264H - 267H	Port A	264H
	(36CH - 36FH)	Port B	265H
		Port C	266H
		Control	267H

PC26 A/D Interface Card

Port Addresses and Functions of the A/D.

	ADDRESS	FUNCTION
PORT A	700H or 780H	bits A0 to A7 are the 8 least significant bits of the digital output (A0 is the LSB).
PORT B	701H or 781H	bits B0 to B3 are the 4 most significant bits of the digital output (B3 is the MSB). bit B4 is used as a TTL trigger input (pin 18 of the 25-D connector)
PORT C	702H or 782H	bits C4 to C7 are the channel address for the multiplexer (C7 is the MSB). bit C0 softwareclock bit C1 softswitch softwareclock (C1 = 1) bit C3 interrupt enable (C1 = 1)

```
{ +++ ------------------------------------------------------------------ +++ }

{Procedures to display pull down menus for the display-format option}
{and to service the relevant procedures}

procedure process_response_5;  {Phase or mag or both display formats selected}
 begin
  Case flag_5 of
   1: begin
  { Magnitude only }
       g:=1;                                  {This selects mag only in unit PLOT}
       option:=1;
       menu_frame_7(g);
       rect_plot(s_par_calibration_flag);      {The rectangular-plot unit PLOT}
       reset_old;                             {Restore menus to previous format}
      end;
   2: begin
  { Phase only }
       option:=2;
       g:=2;
       menu_frame_7(g);                  {This selects phase only in unit PLOT}
       rect_plot(s_par_calibration_flag);      {The rectangular-plot unit PLOT}
       reset_old;                             {Restore menus to previous format}
      end;
   3: begin                                              {Magnitude and Phase}
       option := 3;
       g:=3;
       menu_frame_7(g);                       {Selects mag + phase in unit PLOT}
       rect_plot(s_par_calibration_flag);      {The rectangular-plot unit PLOT}
       reset_old;                             {Restore menus to previous format}
      end;
   4: begin                                          {Return to Previous Menu}
       Exit := true;                                                {Exit Flag}
      end;
   end;{Case}
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Display_format;
{*************************************************************************}
{A procedure which decides on which option are selected from the display type}
{menus. Display type implies mag and/or phase rectilinear displays. The      }
{possible keypresses are up arrow, down arrow, Return and Esc                }
{*************************************************************************}
 begin
  box;                                  {generate the different menus in layers}
  g5:=1;
  menu_frame_4(g5,msg_mag_phase,flag_5);
  repeat
    read_keyboard(Ans2,Exit);                          {monitor the keyboard}
    if (Ans2 = #72) or (Ans2 = #80) then        {if up or down arrows pressed}
     begin
       Case Ans2 of
        #72: begin
              if flag_5 = 1 then g5 := 4                              {up arrow}
              else g5 := flag_5-1;                                  {down arrow}
              menu_frame_4(g5,msg_mag_phase,flag_5);       {draw the new menu}
             end;
        #80: begin
```

```
              if flag_5 = 4 then g5 := 1                          {down arrow}
              else g5 := flag_5+1;                                  {up arrow}
              menu_frame_4(g5,msg_mag_phase,flag_5);          {draw the new menu}
             end;
       end;{Case Ans2}
     end;
    if Ans2 = #13 then process_response_5;    {if return selected then respond}
  until Exit = True;
  flag_5 := 1;                                      {Reset this flag to default}
  block_size := 7;                                          {redraw the menus}
  reset_menu(block_size);
  Exit := False;
 end;

{*****************************************************************************}

procedure process_response_2;                          {Freq display-type option}
 begin
  Case flag_2 of
   1: begin                                                  {Rectangular Display}
       Display_format;                          {Mag and/or phase display option}
       g6:=3;
       menu_frame_3(g6,msg_smith_rect,flag_2);
      end;
   2: begin                                                  {Smith Chart Display}
       main(start_str,stop_str,s_par_flag);
       flag_5 := 1;                                 {Reset this flag to default}
       block_size := 7;
       reset_menu(block_size);
       g6:=3;
       menu_frame_3(g6,msg_smith_rect,flag_2);
      end;
   3: begin                                                {Return to Main Menu}
       Exit_to_main_menu := true;
      end;
   end;{Case}
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure freq_display;
{*****************************************************************************}
{ The format of the display type is selected in this procedure. Formats       }
{ available are rectangular and Smith Chart displays.                         }
{*****************************************************************************}
 begin
  block_size:=7;       {draw the menu structures for the display format option}
  block(block_size);
  g6:=1;
  menu_frame_3(g6,msg_smith_rect,flag_2);
  repeat
    read_keyboard(Ans,Exit_to_main_menu);                {monitor the keyboard}
    if (Ans = #72) or (Ans = #80) then     {if an up or down arrow key pressed}
     begin
       Case Ans of
        #72: begin
              if flag_2 = 1 then g6 := 3                              {up arrow}
              else g6 := flag_2-1;
              menu_frame_3(g6,msg_smith_rect,flag_2);       {generate new menu}
             end;
```

```
        #80: begin
              if flag_2 = 3 then g6 := 1                              {down arrow}
              else g6 := flag_2+1;
              menu_frame_3(g6,msg_smith_rect,flag_2);      {generate new menu}
             end;
       end;{Case Ans}
     end;
    if Ans = #13 then process_response_2;  {If return pressed, service option}
  until Exit_to_main_menu = True;
  flag_2 := 1;                                        {Reset this flag to default}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure text_file;
{*****************************************************************************}
{ This procedure enables the presently active s-parameter to be written to a }
{ text-file on disc. The currently active s-parameter means the s-parameter  }
{ that has last been displayed on either the Smith chart or the rectanglar   }
{ displays. The s-parameter will be error corrected ONLY if a calibration was}
{ performed AND the calibration was active while the sparameter was displayed}
{*****************************************************************************}
 var
  freq_inc:real;
  frequency:array[0..max_array_size] of real;
  outF:text;
  filename:string;
  i:integer;
  ch:char;
 begin
  Window(1,1,80,25);                          {Draw all the fancy headings etc}
  clrscr;
write('=============================================================================');
write(':            The Presently Active S-Parameter be written to
Disc          :');
write(':                                    Hit any
Key                                :');
write('=============================================================================');
  GotoXY(36,20);
  writeln('<ESC Exits>');
  ch := readkey;                                                  {Read keyboard}
  if ch = #0 then ch := readkey;
  if ch <> #27 then                                                 {ESC = #27}
   begin                                    {If Esc not selected then carry on}
    clrscr;
    GotoXY(18,10);
    cursor_off(10,0);       {Enter the filename that you require for the data}
    writeln('Enter Filename including extension <TEST.DAT>');
    GotoXY(18,11);
    Readln(filename);   {if return is selected, defalut fileneame is TEST.DAT}
    if (filename = ' ')OR(filename = '') then filename := 'TEST.DAT';
    Window(30,6,78,25);cursor_off(0,0);
    assign(outf,filename);                                {Initialize the file}
    rewrite(outF);
    Case s_par_flag of
     S11: writeln(outF,'S-Parameter = S11');        {Write the header to file}
     S12: writeln(outF,'S-Parameter = S12');
     S21: writeln(outF,'S-Parameter = S21');
     S22: writeln(outF,'S-Parameter = S22');
    end;
```

```
{Reconvert the phase and mag to degrees and dBs - After they have been scaled}
    Case option of
     1: for i := 0 to size do Amp[i] :=
round(centre_mag[4]-((Amp[i]-centre_mag[2])/scale_mag[2]));
     2: for i := 0 to size do Phase[i] :=
round(centre_phase[4]-((Phase[i]-centre_phase[2])/scale_phase[2]));
     3: for i := 0 to size do
         begin
         Amp[i] := round(centre_mag[4]-((Amp[i]-centre_mag[2])/scale_mag[2]));
         Phase[i] :=
round(centre_phase[4]-((Phase[i]-centre_phase[2])/scale_phase[2]));
         end;
    end;{Case}
    freq_inc := (f_stop-f_start)/size;
    frequency[0] := f_start;
    for i := 1 to size do frequency[i] := frequency[i-1] + freq_inc;
    writeln(outF,'freq [GHz]','       ','Amp [dB]','           ','Phase
[deg]');
    writeln(outF,' ');
    for i := 0 to size do
    writeln(outF,frequency[i]:8:5,'       ',Amp[i]/scale:10:6,'
',Phase[i]/scale:10:3);
    close(outF);
   end;
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Full_Correction;
{*************************************************************************}
{ This procedure deals with the option of Full error correction. Full error  }
{ implies eight term correction for reflection measurements. The prompts for }
{ the various microwave standards are presented and the sampling and          }
{ subsequent storage of the calibration data is controlled from within this  }
{ procedure. Also the error terms are calculated in this procedure and while }
{ they are being computed, a maessage to that effect is displayed.           }
{*************************************************************************}
 var
  one_complex,two_complex:complex;      {1 and 2 expressed as Complex numbers}
  temp:complex;                                {temporary complex variable}
  ch:char;
  i:integer;
 begin
  Window(1,1,80,25);
  Full_Error_Correction_active := FALSE;      {default is full correction off}
  Case s_par_calibration_flag of
   S11,S22:begin                                {Full correction for reflection}
           standard := F_Load;              {display prompts for cal standards}
           if s_par_calibration_flag = S11 then
            display_headings(25,10,'Connect A Fixed Load to port 1',ch)
           else
            display_headings(25,10,'Connect A Fixed Load to port 2',ch);
           if ch <> #27 then
            begin
             data_aquisition(Edf_Amp,Edf_Phase);        {sample the standard}
             standard := Open;                        {standard = open circuit}
             if s_par_calibration_flag = S11 then
              display_headings(20,10,'Connect A Shielded Open Circuit to port
1',ch)
```

```
              else
               display_headings(20,10,'Connect A Shielded Open Circuit to port
2',ch);
              if ch <> #27 then                              ESC exits at any time)
               begin
                data_aquisition(Amp_Cal_Open,Phase_Cal_Open);{Sample standard)
                 standard := short;                                {New standard)
                 if s_par_calibration_flag = S11 then
                  display_headings(24,10,'Connect A Short Circuit to port
1',ch)
                 else
                  display_headings(24,10,'Connect A Short Circuit to port
2',ch);
                 if ch <> #27 then
                 data_aquisition(Amp_Cal_Short,Phase_Cal_Short);
               end;
             end;
{*************************************************************************)
{ Put numbers into complex and LINEAR form.                                 }
{ Edf is stored in Load_Cal,Esf in Short_Cal and Erf in Open_Cal.            }
{ At this point Amp is in levels and phase is continuous levels*scale.      }
{*************************************************************************)
           if ch <> #27 then                  {ESC Exits from the calibration)
            begin
             clrscr;
             gotoXY(17,12);   {Display message while error terms are computed)
             writeln('Computation of Error Terms in Progress ....');
             for i := 0 to size do                          {Compute Error Terms)
              begin
               Load_Cal[i].CmplxType := Polar;
               Load_Cal[i].Re := -(0.2*Edf_Amp[i])/temp_a;
               Load_Cal[i].Re := Power(10,Load_Cal[i].Re/20);
               Load_Cal[i].Im := (Edf_Phase[i]/scale)/temp_p;

               Short_Cal[i].CmplxType := Polar;
               Short_cal[i].Re := -(0.2*Amp_Cal_Short[i])/temp_a;
               Short_cal[i].Re := Power(10,Short_cal[i].Re/20);
               Short_cal[i].Im := (Phase_Cal_Short[i]/scale)/temp_p;

               Open_Cal[i].CmplxType := Polar;
               Open_Cal[i].Re :=  -(0.2*Amp_Cal_Open[i])/temp_a;
               Open_Cal[i].Re :=  Power(10,Open_Cal[i].Re/20);
               Open_Cal[i].Im :=  (Phase_Cal_Open[i]/scale)/temp_p;
               Esf[i].CmplxType := Polar;
               Erf[i].CmplxType := Polar;
               Complex_value[i].CmplxType := Polar;
              end;
             {Compute the Error Terms Erf and Esf)
              one_complex.CmplxType := Polar;
              one_complex.Re := 1;one_complex.Im := 0;
              two_complex.CmplxType := Polar;
              two_complex.Re := 2;two_complex.Im := 0;
              Temp.CmplxType := Polar;

              for i := 0 to size do
               begin
                CmplxMult(two_complex,Load_Cal[i],Temp);
                CmplxSub(Temp,Short_Cal[i],Temp);
                CmplxSub(Temp,Open_Cal[i],Temp);
                CmplxSub(Short_Cal[i],Open_Cal[i],Esf[i]);
```

```
                CmplxDiv(Temp,Esf[i],Esf[i]);
                CmplxSub(Load_Cal[i],Short_Cal[i],Temp);
                CmplxAdd(one_complex,Esf[i],Erf[i]);
                CmplxMult(Temp,Erf[i],Erf[i]);
               end;
              freq_response_cal := False;
{freq response-only cal is now redundant since a full cal has been performed}
             end;
            end;
   S12,S21: inline($90);
  end;{Case}
  Window(40,9,80,25);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Calibrate;
{*****************************************************************************}
{ This procedure decides on whether S11 or S22 is selected for a full       }
{ calibration.                                                              }
{*****************************************************************************}
 begin
  klein_box;                                          {the menus are drawn}
  g3:=1;
  menu_frame_3(g3,msg_s_par_cal,flag_9);
  repeat
    read_keyboard(Ans9,Exit);                        {monitor the keyboard}
    if (Ans9 = #72) or (Ans9 = #80) then          {Up or down arrows pressed}
     begin
       Case Ans9 of
        #72: begin                                                {up arrow}
              if flag_9 = 1 then g3 := 3
              else g3 := flag_9-1;
              menu_frame_3(g3,msg_s_par_cal,flag_9);          {draw new menus}
             end;
        #80: begin
              if flag_9 = 3 then g3 := 1                          {down arrow}
              else g3 := flag_9+1;
              menu_frame_3(g3,msg_s_par_cal,flag_9);          {draw new menus}
             end;
       end;{Case Ans2}
     end;
  until (Ans9 = #13) or (Exit = True);         {ESC Exits, return is serviced}
  if (Ans9 = #13) then                             {A return has been pressed}
   begin
    Case flag_9 of
     1: begin
         s_par_calibration_flag := S11;
         Full_Correction;            {The full correction procedure is called}
        end;
     2: begin
         s_par_calibration_flag := S22;
         Full_Correction;            {The full correction procedure is called}
        end;
     3: inline($90);
    end;
   end;
  flag_9 := 1;                                    {Reset this flag to default}
  block_size := 7;
  reset_menu(block_size);                                  {Redraw the menus}
  Exit := False;
```

```
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure select_an_option;
{*************************************************************************}
{ If Return is pressed from the main-menu then the option selected is     }
{ serviced in this procedure                                              }
{*************************************************************************}
 begin
   Window(30,6,78,25);
   Case main_menu_flag of
      1: begin
       freq_set;                                        {Frequency range option}
       g:=2;
          menu_frame_7(g);          {After returning the 2nd option is chosen}
      end;
      2: begin
          s_parameter;                {S-parameter and attenuation selection}
          g:=3;                     {After returning the 3rd option is chosen}
          menu_frame_7(g);
         end;
      3: begin
          freq_display;                        {Frequency Display format option}
          flag_1 := 1;                         {Start freq(not stop) flag reset}
         end
      4: begin
          Calibrate;                                  {Full Calibration option}
          menu_frame_7(g);
         end;
      5: text_file;                             {Write data to text file option}
      6: begin
          Window(1,1,80,25);
          call_amp_cal;                         {Amplifier Calibration option}
          Window(30,6,78,25);
         end;
      7: begin                                      {This option exits to Dos}
          Exit_to_dos := True;
         end;
     end; {Case}
    end;

{ +++ ------------------------------------------------------------------ +++ }

procedure service_key;
{*************************************************************************}
{Service an arrow/return keypressed to decide on which option has been chosen}
{options include display format,s-parameter and incident atten selection,     }
{ start and stoop freqs and RF level, full correction and amplifier cal.      }
{*************************************************************************}
label 1;
begin
1:repeat
    Exit_to_main_menu := False;                 {Default setting of exit flag}
    read_keyboard(Ch,Exit_to_dos);                      {monitor the keyboard}
    if (Ch = #72) or (Ch = #80) then           {up or down arrow keys pressed}
     begin
       Case Ch of
        #72: begin
              if main_menu_flag = 1 then g := 7                          {up arrow}
```

```
            else g := main_menu_flag-1;
            menu_frame_7(g);                                    {draw new menus}
           end;
       #80: begin
            if main_menu_flag = 7 then g := 1                    {down arrow}
            else g := main_menu_flag+1;
            menu_frame_7(g);                                    {draw new menus}
           end;
     end;{Case Ch}
     Goto 1;
   end;
   if Ch = #13 then select_an_option;                  {Return was selected}
   Window(1,1,80,25);
   outline;                                             {Sketch menu outline}
   heading;                                          {Main heading in outline}
   g := main_menu_flag;                                        {Default Menu}
   menu_frame_7(g);
 until (Exit_to_dos = True) or (Ch = 'e') or (Ch = 'E');      {Possible EXITS}
 clrscr;


{ +++ ------------------------------------------------------------------ +++ }
{Main program}

BEGIN
 Initializer;                                        {Initialization procedure}
 read_gain;                                       {Read the phase and amp gains}
 temp_p:= 4.096*gain_phase_channel;            {Scaling factor for phase channel}
 temp_a:= 4.096*gain_amp_channel;           {Scaling factor for magnitude channel}
 service_key;                                          {Check for keys pressed}
 port[$261] := $00;                                   {Unselect Remote Control}
 cursor_off(20,12);                          {Reset the cursor to the Dos standard}
END.

{ +++ ------------------------------------------------------------------ +++ }

unit initial;
{*************************************************************************}
{ A unit to initialize variables for the main program                     }
{*************************************************************************}
{$O+,F+}                                        {Compiler option for Overlaid units}
{*************************************************************************}
interface
{*************************************************************************}
uses crt,graph,mainmenu,menu_utility,plot_unt;       {units used in this unit}

var
 Exit_to_dos:boolean;                {These 4 vars are used in the main program}
 flag_3:integer;
 Exit:boolean;

procedure Initializer;     {The procedure that is called from the main program}
{*************************************************************************}
implementation
{*************************************************************************}

procedure call_menu_unit;                    {Call the unit MAINMENU which draws}
 var                                                  {an 8410C on the titile page}
  grdriver,grmode:integer;
 begin
```

```
   grdriver := detect;                                  {Go into the graphics mode}
   InitGraph(grdriver,grmode,'');
   Draw_Analyzer;                                        {Call the unit-mainmenu}
   RestoreCrtMode;                                          {Return to textmode}
   Delay(400);                                {Delay to prevent "Jitter" when going}
 end;                                                      {from graph to text mode}

{ +++ ------------------------------------------------------------------ +++ }

procedure Initializer;
{*****************************************************************************}
{ Procedure to initialize system. The sweeper front panel controls are        }
{ programmed via the HPIB bus                                                 }
{*****************************************************************************}
 begin
  cursor_off(0,0);                                                {Switch off cursor}
  Assign(GPIO,'HPIBDEV');                          {Initialize the HBIB Device file}
  ReWrite(GPIO);                                                      {Rewrite it}
  port[$263] := $88;                                          {initialize 8255 card}
  port[$703] := $92;                                           {initialize A/D PPI}
  port[$261] := $07;                          {Remote Mode Selected on s-par test set}
  port[$260] := $03;                                            {Default S22+60dB}
  clrscr;
  gotoxy(26,10);                                    {Display message while reading}
  writeln('Reading via HPIB.....');                   {initial conditions via HPIB}
  Read_via_HPIB('opfa',f_start);             {Read THe exsisting f_start and f_stop}
  Read_via_HPIB('opfb',f_stop);                {via HPIB and make them the defaults}
  SelectHPIB(719);                                      {Address the 8350B Sweeper}
  Write(GPIO,'st99ms');                                 {Default Sweep time = 99 ms}
  Write(GPIO,'t3');                                {Default = Internal sweep trigger}
  Write(GPIO,'ts');                                            {Default Time Sweep}
  Write(GPIO,'ak0');                                   {Default Ampt Marker = off}
  Write(GPIO,'dp1');                                    {Default Displ blank = on}
  Write(GPIO,'rp0');                                      {Default Rf blank = off}
  Write(GPIO,'md0');                                {Default square wave mod = off}
{Plug in Unit}
  Write(GPIO,'pl5dm');                                 {Default Power Level = 5dBm}
  Write(GPIO,'ps0');                                    {Default Power Sweep = off}
  Write(GPIO,'sl0');                                          {Default = Slope off}
  Write(GPIO,'a1');                                {Default ALC Mode = internal}
  Write(GPIO,'rf1');                                             {Default RF = on}
  Write(GPIO,'fi1');                                    {Default CW Filter = On}
  call_menu_unit;                                {Fancy MainHeadings and plots}
  cursor_off(0,0);                 {Switch off cursor again after graphics mode}
  f_min := 0.01;                                    {The Minimum freq of Sweeper}
  f_max := 26.5;                                    {The Maximum Freq of Sweeper}
  rf_min := -5;                                            {The Minimum RF Level}
  rf_max := 20;                                            {The Maximum RF Level}
  rf := 5;                                               {Default RF level in dBm}
  delta_f := (f_stop - f_start)/10.0;          {frequency increment per division}
  Str(f_start:6:4,start_str);                             {Default start-freq-string}
  Str(f_stop:6:4,stop_str);                               {Default stop-freq-string}
  Write(GPIO,'fa'+start_str+'Gz');             {Program The start+stop frequencies}
  Write(GPIO,'fb'+stop_str+'Gz');
  Str(delta_f:6:4,delta_f_str);
  Str(rf:6:4,rf_str);
  Exit_to_main_menu := False;                {Default flag to exit to the mainmenu}
  Exit_to_dos := False;                                 {Default flag to exit to dos}
  flag_1 := 1;                                           {Default setting for menu#1}
```

```
  flag_2 := 1;                                    {Default setting for menu#2}
  flag_3 := 1;                                    {Default setting for menu#3}
  flag_4 := 1;
  flag_5 := 1;
  flag_8 := 1;                                {Default setting for error model}
  flag_9 := 1;                                    {Default for calibrating S11}
  s_par_flag := S22;                               {default s-parameter = S22}
  freq_response_cal := True;                  {flag for freq-response only cal}
  s_par_calibration_flag := OFF;     {Whether a full calibration was performed}
  atten_flag := 0;                          {Default attenuation in calibration}
  Exit := False;
  outline;                                                {Sketch menu outline}
  heading;                                              {Main heading in outline}
  g:=1;                                                        {Default Menu}
  menu_frame_7(g);
  Full_Error_Correction_active := FALSE;          {Default for full correction}
 end;

{ +++ ------------------------------------------------------------------ +++ }
end.                                                        {END OF UNIT INITIAL}

{*****************************************************************************}
unit MainMenu;
{*****************************************************************************}
{ This unit plots a fancy Title Page and gives the user the option            }
{ of quitting or carrying on with the main program. NOTE when this           }
{ unit has been called, it is essential that the viewport is reset           }
{ The display must be in graphics mode before calling this unit.             }
{*****************************************************************************}
{$O+,F+}                                    {Compiler options for overlaid units}
{*****************************************************************************}
interface
{*****************************************************************************}
uses
  crt,graph;                                              {units used in this unit}

procedure Draw_Analyzer;                {Procedure called from the mian program}

{*****************************************************************************}
implementation
{*****************************************************************************}
 var
  i,j:integer;

{ +++ ------------------------------------------------------------------ +++ }

 procedure Draw_Analyzer;                                          {Draw the 8410C}
  begin
   rectangle(0,0,719,347);                                           {Outer Border}
   Rectangle(15,310,265,340);
   Rectangle(11,306,269,344);
   OutTextXY(40,315,'<ESC> aborts');                                        {Prompts}
   OutTextXY(40,328,'Any other key to continue');
   SetTextStyle(1,0,5);                                  {Triplex Font selected}
   OutTextXY(55,5,' HP8410C NETWORK ANALYZER');
   OutTextXY(55,40,'          UPGRADE       ');
   Line(0,90,719,90);
   SetTextStyle(2,0,4);
{*****************************************************************************}
{ Display all the prompts for configuring the system with parameters that     }
```

```
{ cannot be programmed via the HPIB or 8255 PPI.                                    }
{*************************************************************************************}
   OutTextXY(35,100,'PLEASE ENSURE THE FOLLOWING');
   Rectangle(15,100,225,110);
   OutTextXY(35,120,'HP8410C NETWORK ANALYZER');
   Line(35,130,177,130);
   OutTextXY(35,135,'Test Channel Gain = 55dB');
   OutTextXY(35,145,'Sweep Stability = CW');
   OutTextXY(35,155,'Source = Normal');
   OutTextXY(35,175,'HP8412B PHASE-MAG DISPLAY');
   Line(35,185,180,185);
   OutTextXY(35,190,'Phase Offset = +');
   OutTextXY(35,200,'Phase = 180 degrees');
   OutTextXY(35,210,'BW = 10 kHz');
   OutTextXY(35,230,'HP8746B S-PARAMETER TEST SET');
   Line(35,240,205,240);
   OutTextXY(35,245,'Mode = Remote (Rear of test set)');
   OutTextXY(35,265,'NOTE - THE PARAMETERS LISTED ABOVE MUST');
   OutTextXY(35,275,'UNDER NO CIRCUMSTANCES BE CHANGED DURING');
   OutTextXY(35,285,'PROGRAM EXECUTION .');

   Rectangle(360,150,600,200);                          {Top Main Rectangle}
   Rectangle(360,205,600,255);                            {Middle Rectangle}
   Rectangle(360,260,600,310);                            {Bottom Rectangle}
   Rectangle(510,155,595,195);                              {Top inner rect}
   Line(510,165,595,165);                          {Lines representing a grid}
   Line(510,175,595,175);
   Line(510,185,595,185);
   Line(527,155,527,195);
   Line(544,155,544,195);
   Line(561,155,561,195);
   Line(578,155,578,195);
   Circle(410,175,5);                                 {Top inner left circle}
   Circle(460,175,5);                                {Top inner right circle}
   Rectangle(510,210,595,250);                           {Middle inner rect}
    for i := 1 to 3 do                                  {Middle inner circles}
      begin
        for j := 1 to 4 do Circle(360+(j*30),210+(i*10),4);
      end;
   Circle(440,285,8);                               {Bottom inner left circle}
   Circle(520,285,8);                              {Bottom inner right circle}
   SetViewPort(511,211,594,249,Clipon);              {Plot Real-Time Display}
    repeat
      ClearViewPort;
      MoveTo(0,0);
      For i := 1 to 17 do
       begin
         LineTo(i*5,round((sin(i)/(i))*17+20));          {Plot a sinc function}
         delay(50);
       end;
    until keypressed;
    if readkey = #027 then                          {ESC aborts [#027 = ESC]}
      begin
       closegraph;
       halt;
      end;
  end;                                                        {Draw_Analyzer}

{ +++ ------------------------------------------------------------------ +++ }
```

```
end.                                                    {END OF UNIT MAINMENU}
{*************************************************************************}

unit EdInit;
{*************************************************************************}
{ A unit which is used in order to allow the implementation of Overlaid files}
{ This unit is compiled without the far call or overlaid compiler options    }
{ ($O,$F), and all it does is to initialize the Overlay manager.             }
{*************************************************************************}
interface
implementation
uses Overlay;                                          {Turbo's overlay unit}
begin
OvrInit('MENUFIN.OVR');                      {Initialize the overlay manager}
end.

{*************************************************************************}

unit amp_cal;
{$O+,F+}
{*************************************************************************}
{ A unit to calibrate the amplifiers and network analyzer gains.             }
{ The system drifts with time and every time it is switched on, a different  }
{ level will be outputted at the rear of the HP8412B display.               }
{ This option should be selected before the cad program periodically.        }
{ The values of gain are computed and stored in files - Amp channel in       }
{ ATHRU.DAT and Phase channel in PTHRU.DAT. These values are then read from  }
{ the main program and used as the values Gain_phase and Gain_Amp.           }
{*************************************************************************}

{*************************************************************************}
interface
{*************************************************************************}
{$M $4000, 0, $4000}            {Compiler option necessary for HPIB operation}

uses crt,dos,menu_utility,plot_unt;                {units used in this program}

procedure call_amp_cal;           {The procedure called from the main program}

{*************************************************************************}
implementation
{*************************************************************************}

var
 value1,value2:integer;
 count,i:integer;
 message:string;
 GPIO,OutF:text;
 gain:real;
 EXIT:Boolean;

{ +++ ------------------------------------------------------------- +++ }

function Adsample(channel:integer):integer;
{*************************************************************************}
{ A function to perform an AD conversion on any of the 16 channels.          }
{ The required channel is inputted to the routine                            }
{*************************************************************************}
 var
  i:integer;
begin
```

```
 Port[$702] := (channel SHL 4) + 2;
 Port[$702] := (channel SHL 4) + 3;
 for i:=1 TO 26 do
  begin
  end;
 Adsample :=((Port[$701] AND $0F) SHL 8) + Port[$700];
end;

{ +++ ------------------------------------------------------------------ +++ }

procedure exit_proc;
{*****************************************************************************)
{ A procedure to reprogram all the system conditions that were in effect      }
{ before this unit was called. For example, the correct s-parameters and      }
{ incident attenuation must be reprogrammed because this unit changes their   }
{ values.                                                                     }
{*****************************************************************************)
 begin
  Write(GPIO,'fa'+start_str+'Gz');          {Program The start+stop frequencies)
  Write(GPIO,'fb'+stop_str+'Gz');
  Case s_par_flag  of
   S11: port[$261] := $04;                    {program the selected S-Parameter)
   S21: port[$261] := $06;                    {to ensure that the correct S-par)
   S22: port[$261] := $07;                           {is chosen for calibration)
   S12: port[$261] := $05;
  end;
  Case atten_flag of
    0: port[$260] := $00;                     {Program the selected attenuation)
   10: port[$260] := $04;                   {to ensure that the same attenuation)
   20: port[$260] := $02;           {is chosen for calibration and measurement)
   30: port[$260] := $06;
   40: port[$260] := $01;
   50: port[$260] := $05;
   60: port[$260] := $03;
   70: port[$260] := $07;
  end;
  EXIT := TRUE;                                  {The exit code from this unit)
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure display_msg(message:string);
{*****************************************************************************)
{ A procedure to display the prompts nescessary in order for the user to      }
{ connect a thru which can then be measured by the system and used to compute)
{ the values opf mag and phase channel gains.                                 }
{*****************************************************************************)
 var
  ch:char;
 begin
  repeat
   clrscr;                      {Remove the menus from the screen by clearing it)
   GotoXY(25,2);
   writeln('CALIBRATION OF AMPLIFIERS');                    {Display the messages)
   GotoXY(25,3);                                           {Underline the message)
   writeln('========================');
   GotoXY(23,10);
   Write(message);            {Prompt the user to select 55dB Test channel gain)
   GotoXY(25,24);
```

```
   Writeln('<C-Continue>    <Esc-Quit>');          {Fancy frills on the screen}
   GotoXY(25,25);
   Write(  '▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀▀');                          {Underlining}
   ch := readkey;                              {ESC exits, C continues the program}
  until (ch = 'c') or (ch = 'C') or (ch = #27);
  if (ch = #27) then exit_proc;                                          {ESC code}
  clrscr;                                                {clear screen for menus}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure initial;
{*****************************************************************************}
{ Procedure to initialize the system to a configuaration where the gains can }
{ be computed. The sweeper and test set are programmed to CW mode and S21.   }
{*****************************************************************************}
 begin
  EXIT := FALSE;                                          {Initialize exit code}
  clrscr;                                                    {clear the screen}
  cursor_off(0,0);                                       {switch off the cursor}
  Assign(GPIO,'HPIBDEV');                               {initialize the GPIB bus}
  ReWrite(GPIO);
  Write(GPIO,'cw1.15gz');                                    {program CW=1.5GHz}
  Write(GPIO,'p15dm');                                  {Program power = 15dBm}
  port[$263] := $88;                                     {initialize 8255 card}
  port[$703] := $92;                                       {initialize A/D PPI}
  port[$261] := $07;                  {Remote Mode Selected on s-par test set}
  port[$261] := $06;                                                {Select S21}
  delay(200);                                    {delay to prevent key chatter}
  port[$260] := $00;                            {Program input attenuation = 0dB}
  GotoXY(20,10);Writeln('Connect a Thru between the Test Ports');
  GotoXY(27,12);
  writeln('Hit any key to continue');          {Prompt the user to strike a key}
  Readln;clrscr;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure call_amp_cal;
{*****************************************************************************}
{ The main procedure in this unit. This procedure is called from the main    }
{ program and performs all the necessary calls to facilitate the amplifier   }
{ calibration.                                                               }
{*****************************************************************************}
label
 1;
 begin
  initial;
  for count := 1 to 2 do                          {Repeat for mag and for phase}
   begin
    message := 'Set Test Channel Gain = 55dB'; {Set test channel gain to 55dB}
    display_msg(message);
    if EXIT = TRUE then Goto 1;                          {ESC exits this option}
      delay(500);
      value1 := 0;                                            {Take 100 samples}
      for i := 1 to 100 do
       begin
        value1 := adsample(count) + value1;                      {Sum the total}
       end;
      value1 := value1 div 100;                               {average the values}
```

```
      message := 'Set Test Channel Gain = 45dB';              {set gain = 45dB}
     display_msg(message);
     if EXIT = TRUE then Goto 1;                                      {ESC exits}
       delay(500);
       value2 := 0;
       for i := 1 to 100 do                                  {take 100 samples}
        begin
         value2 := adsample(count) + value2;                          {sum them}
        end;
       value2 := value2 div 100;                                  {average them}

       gain := (10*(value1-value2))/(4096*10*50e-3);      {compute the gain for}
       Case count of                                                {this channel}
        1: Assign(OutF,'ATHRU.DAT');
        2: Assign(OutF,'PTHRU.DAT');          {Write the gain factor to a file}
       end;
       Rewrite(OutF);
       Writeln(OutF,gain:12:10);
       Close(OutF);
       clrscr;
       Case count of      {Prompt the user to swop the mag and phase cables at}
        1: begin                                {rear of the HP8412B display unit}
            GotoXY(15,10);
            Write('Swop the Red(Phase) and Yellow(Amp) Cables');
            GotoXY(13,12);
            Writeln('at the rear of the 8412B Display - Hit Enter');
           end;                                        {ENTER continues the program}
        2: begin
            clrscr;                  {prompt user to return the cables to their}
            GotoXY(28,8);          {original configuration and test gain = 55dB}
            Write('Remember to');
            GotoXY(15,10);
            Write('Swop the Red(Phase) and Yellow(Amp) Cables');
            GotoXY(13,12);
            Writeln('at the rear of the 8412B Display and to reset');
            GotoXY(20,14);
            Writeln('the Test Channel Gain to 55dB');
            GotoXY(29,18);
            Writeln('HIT RETURN');                {RETURN continues the program}
           end;
       end;
       readln;
     end;
     clrscr;

1:   for i := 1 to 5 do              {To signify that an amplifier calibration}
       begin                    {is complete, a message is flashed on the screen}
        GotoXY(26,10);
        write('CALIBRATION COMPLETE');
        delay(150);
        clrscr;
        delay(150);
       end;
     exit_proc;
   end;

{+++ ------------------------------------------------------------------ +++ }
end. {Unit}
```

```
{*************************************************************************}
Unit Math;
{*************************************************************************}
{ A toolbox of mathematical functions which can be used from a Turbo program }
{ The main feature of this unit is the ability to handle Complex Numbers.    }
{ For example Rectangular to polar conversions, complex add,subtract are     }
{ provided                                                                   }
{*************************************************************************}

{$O+,F+}                        {Compiler options for Far Call and Overlaid unit}

{*************************************************************************}
Interface
{*************************************************************************}

type
  complex = record                        {Declaration of Complex variable Types}
    Re:Real;                               {The real part or the mag if in polar}
    Im:Real;                              {The imag part or the angle if in polar}
    CmplxType:Boolean;                        {Polar or rectangular co-ordinates}
  end;

const
  Rectangular = True;                                {Boolean use for complex type}
  Polar = False;
  Infinity = 10000000000000000000000000000000.0;     {default value of infinity}

 {The procedures that can be called from other Turbo programs using this unit}
Procedure RectToPolar(var ComplexArg, ComplexResult : Complex);
Procedure PolarToRect(var ComplexArg, ComplexResult : Complex);
Function Log10(var Arg : Real) : Real;
Function LogN(var Arg,N : Real) : Real;
Procedure CmplxMult(var Arg1,Arg2,Arg3 : Complex);
Procedure CmplxDiv(var Arg1,Arg2,Arg3 : Complex);
Procedure CmplxAdd(var Arg1,Arg2,Arg3 : Complex);
Procedure CmplxSub(var Arg1,Arg2,Arg3 : Complex);
Procedure CmplxMod(var Arg1 : Complex; var Md : Real);
Procedure CmplxAng(var Arg1 : Complex; var Ang : Real);
Procedure CmplxInv(var Arg1,Arg2 : Complex);
function Power(Number, Exponent: real):real;

{*************************************************************************}
Implementation
{*************************************************************************}

Procedure RectToPolar(var ComplexArg,ComplexResult:Complex);
{*************************************************************************}
{ Performs a rectangular to polar conversion on a complex number           }
{ Input ComplexArg and output ComplexResult.                               }
{*************************************************************************}
label
 4;
var
  TmpArg:Complex;
begin
  TmpArg := ComplexArg;
  if ComplexArg.CmplxType then
   begin
    TmpArg.Re := Sqrt(Sqr(ComplexArg.Re)+Sqr(ComplexArg.Im));
```

```
    if ComplexArg.re = 0 then
     begin
      if ComplexArg.re = 0 then
       begin
        TmpArg.Im := 0;
        Goto 4;
       end;
       if ComplexArg.Im < 0 then
        begin
         TmpArg.Im := -90;
         Goto 4;
        end
       else if ComplexArg.Im = 0 then
        begin
         TmpArg.Im := 0;
         Goto 4;
        end
       else
        begin
         TmpArg.Im := 90;
         Goto 4;
        end;
     end;

    if ComplexArg.Im = 0 then
     begin
      if ComplexArg.Re >= 0 then
       begin
        TmpArg.Im := 0;
        goto 4;
       end;
      TmpArg.Im := 180;
      goto 4;
     end;

    TmpArg.Im := ArcTan(ComplexArg.Im/ComplexArg.Re)*180/Pi;
    if ComplexArg.Re < 0 then TmpArg.Im := TmpArg.Im + 180;

4:  TmpArg.CmplxType := Polar;
    ComplexResult := TmpArg;
 end;
 end;

{*************************************************************************}

Procedure PolarToRect(var ComplexArg,ComplexResult:Complex);
{*************************************************************************}
{ Procedure to convert Polar complex numbers into rectangular co-ordinates  }
{ Input is ComplexArg and output is ComplexResult.                          }
{*************************************************************************}
var
  TmpArg : Complex;
begin
  TmpArg := ComplexArg;
  if not ComplexArg.CmplxType then
  begin
    TmpArg.Re := (ComplexArg.Re)*Cos(ComplexArg.Im*pi/180);
    TmpArg.Im := (ComplexArg.Re)*Sin(ComplexArg.Im*pi/180);
    TmpArg.CmplxType := Rectangular;
  end;
```

```
  ComplexResult := TmpArg;
end;

{*****************************************************************************}

Function Log10(var Arg:Real):Real;
{*****************************************************************************}
{ A function to take log to the base 10.                                      }
{*****************************************************************************}
begin
  if Arg <= 0 then Log10 := -Infinity
  else
   Log10 := Ln(Arg)/2.302585093;
end;

{*****************************************************************************}

Function LogN(var Arg,N:Real):Real;
{*****************************************************************************}
{ A function to take the log of a number to any base stipulated.              }
{ The Arg is the number and N is the Base.                                    }
{*****************************************************************************}
begin
  LogN := Ln(Arg)/Ln(N);
end;

{*****************************************************************************}

Procedure CmplxMult(var Arg1,Arg2,Arg3 : Complex);
{*****************************************************************************}
{ Routine to perform a complex multiplication.                                }
{ Arg1 and Arg2 are inputs and Arg3 is the result                             }
{*****************************************************************************}
var
  Rslt1,Rslt2 : Complex;
begin
  Rslt1 := Arg1;
  Rslt2 := Arg2;
  if Arg1.CmplxType = Rectangular then
  begin
    RectToPolar(Arg1,Rslt1);
  end;
  if Arg2.CmplxType = Rectangular then
  begin
    RectToPolar(Arg2,Rslt2);
  end;
  Arg3.Re := Rslt1.Re*Rslt2.Re;
  Arg3.Im := Rslt1.Im+Rslt2.Im;
  Arg3.CmplxType := Polar;
end;

{*****************************************************************************}

Procedure CmplxDiv(var Arg1,Arg2,Arg3 : Complex);
{*****************************************************************************}
{ Routine to perform a complex division on two complex numbers                }
{ Arg1 is divided by Arg2 to yield Arg3.                                      }
{*****************************************************************************}
var
  Rslt1,Rslt2 : Complex;
begin
```

```
  Rslt1 := Arg1;
  Rslt2 := Arg2;
  if Arg1.CmplxType = Rectangular then
  begin
    RectToPolar(Arg1,Rslt1);
  end;
  if Arg2.CmplxType = Rectangular then
  begin
    RectToPolar(Arg2,Rslt2);
  end;
  if Rslt2.Re <> 0 then Arg3.Re := Rslt1.Re/Rslt2.Re else Arg3.Re := Infinity;
  Arg3.Im := Rslt1.Im-Rslt2.Im;
  if Arg3.Re = 0 then Arg3.Im := 0;
  Arg3.CmplxType := Polar;
end;

{*****************************************************************************}

Procedure CmplxAdd(var Arg1,Arg2,Arg3 : Complex);
{*****************************************************************************}
{ Ths routine adds two complex numbers together.                              }
{ Arg1 and Arg2 are added to give Arg3.                                       }
{*****************************************************************************}
var
  Rslt1,Rslt2 : Complex;
begin
  Rslt1 := Arg1;
  Rslt2 := Arg2;
  if Arg1.CmplxType = Polar then
  begin
    PolarToRect(Arg1,Rslt1);
  end;
  if Arg2.CmplxType = Polar then
  begin
    PolarToRect(Arg2,Rslt2);
  end;
  Arg3.Re := Rslt1.Re+Rslt2.Re;
  Arg3.Im := Rslt1.Im+Rslt2.Im;
  Arg3.CmplxType := Rectangular;
end;

{*****************************************************************************}

Procedure CmplxSub(var Arg1,Arg2,Arg3 : Complex);
{*****************************************************************************}
{ Routine to subtract two complex numbers.                                    }
{ Arg2 is subtracted from Arg1 to yield Arg3.                                 }
{*****************************************************************************}
var
  Rslt1,Rslt2 : Complex;
begin
  Rslt1 := Arg1;
  Rslt2 := Arg2;
  if Arg1.CmplxType = Polar then
  begin
    PolarToRect(Arg1,Rslt1);
  end;
  if Arg2.CmplxType = Polar then
  begin
```

```
    PolarToRect(Arg2,Rslt2);
  end;
  Arg3.Re := Rslt1.Re-Rslt2.Re;
  Arg3.Im := Rslt1.Im-Rslt2.Im;
  Arg3.CmplxType := Rectangular;
end;

{*****************************************************************************}

Procedure CmplxMod(var Arg1 : Complex; var Md : Real);
var
  Rslt1 : Complex;
begin
  RectToPolar(Arg1,Rslt1);
  Md := Arg1.Re;
end;

{*****************************************************************************}

Procedure CmplxAng(var Arg1 : Complex; var Ang : Real);
var
  Rslt1 : Complex;
begin
  RectToPolar(Arg1,Rslt1);
  Ang := Arg1.Im;
end;

{*****************************************************************************}

Procedure CmplxInv(var Arg1,Arg2 : Complex);
var
  Num : Complex;
begin
  Num.Re := 1; Num.Im := 0; Num.CmplxType := Polar;
  CmplxDiv(Num,Arg1,Arg2);
end;

{*****************************************************************************}

function Power(Number,Exponent:real):real;
{*****************************************************************************}
{ Function to raise a number to the power of another number.                  }
{ Number represents the number and Power is the power to which it is raised   }
{*****************************************************************************}

begin
   if Number > 0.0 then
      Power := exp(Exponent * ln(Number))
   else
      Power := 0.0
end;

{*****************************************************************************}

end.
```

```
 x_phase[3] := -1;                                          {x-axes origin}
 y_phase[1] := scale*(360);                                        {ymax}
 y_phase[2] := scale*(-360);                                       {ymin}
 y_phase[3] := 0;                                           {y-axes origin}
 graphs_phase[1] := 0;                                       {x lower left}
 graphs_phase[2] := 344;                                     {y lower left}
 graphs_phase[3] := 510;                                    {width of area}
 graphs_phase[4] := 344;                                   {height of area}
                          {graphics plot scaling factors for phase plot}
 x_mag[1] := size+1;                                               {xmax}
 x_mag[2] := -1;                                                   {xmin}
 x_mag[3] := -1;                                            {x-axes origin}
 y_mag[1] := scale*10;                                             {ymax}
 y_mag[2] := -scale*70;                                            {ymin}
 y_mag[3] := 0;                                             {y-axes origin}
 graphs_mag[1] := 0;                                         {x lower left}
 graphs_mag[2] := 344;                                       {y lower left}
 graphs_mag[3] := 510;                                      {width of area}
 graphs_mag[4] := 344;                                     {height of area}

 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure mag_phase_constants;
{*************************************************************************}
{ Part of the routines used to plot rectangular graphs of mag and phase   }
{ These constants are for simultaneous plots of mag and phase.            }
{*************************************************************************}
 begin
 {MAG-constants}
  x_mag[1] := size+1;                                              {xmax}
  x_mag[2] := -1;                                                  {xmin}
  x_mag[3] := -1;                                           {x-axes origin}
  y_mag[1] := scale*10;                                            {ymax}
  y_mag[2] := -scale*70;                                           {ymin}
  y_mag[3] := 0;                                            {y-axes origin}
  graphs_mag[1] := 0;                                        {x lower left}
  graphs_mag[2] := 168;                                      {y lower left}
  graphs_mag[3] := 510;                                     {width of area}
  graphs_mag[4] := 168;                                    {height of area}

 {PHASE-constants}
  x_phase[1] := size+1;                                            {xmax}
  x_phase[2] := -1;                                                {xmin}
  x_phase[3] := -1;                                         {x-axes origin}
  y_phase[1] := scale*360;                                         {ymax}
  y_phase[2] := -scale*360;                                        {ymin}
  y_phase[3] := 0;                                          {y-axes origin}
  graphs_phase[1] := 0;                                      {x lower left}
  graphs_phase[2] := 347;                                    {y lower left}
  graphs_phase[3] := 510;                                   {width of area}
  graphs_phase[4] := 168;                                  {height of area}
 end;

{ +++ ------------------------------------------------------------------ +++ }

function calaxe(x:axesdim;gr:integer):integer;
 var
  grtemp:real;                          {Part of routine to find axes co-ords}
```

```
  grtemp:=gr;
  calaxe:=round(((x[1]-x[3])/(x[1]-x[2]))*grtemp);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure mag_scaling(x_mag,y_mag:axesdim;graphs_mag:graphdim;
VAR scale_mag:scalefactor;VAR centre_mag:centrefactor);

 var
  temp1,temp2:real;

 begin
  centre_mag[3] := x_mag[3];
  centre_mag[4] := y_mag[3];
  gr2_mag[1]    := graphs_mag[1];
  gr2_mag[2]    := graphs_mag[2];
  gr2_mag[3]    := graphs_mag[3];
  gr2_mag[4]    := graphs_mag[4];
  temp1         := gr2_mag[3];
  temp2         := gr2_mag[4];
  scale_mag[1]  := temp1/(x_mag[1]-x_mag[2]);
  scale_mag[2]  := temp2/(y_mag[1]-y_mag[2]);
  centre_mag[1] := gr2_mag[1]+(gr2_mag[3]-calaxe(x_mag,gr2_mag[3]));
  centre_mag[2] := gr2_mag[2]-(gr2_mag[4]-calaxe(y_mag,gr2_mag[4]));
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure phase_scaling(x_phase,y_phase:axesdim;graphs_phase:graphdim;
VAR scale_phase:scalefactor;VAR centre_phase:centrefactor);

 var
  temp1,temp2:real;

 begin
  centre_phase[3] := x_phase[3];
  centre_phase[4] := y_phase[3];
  gr2_phase[1]    := graphs_phase[1];
  gr2_phase[2]    := graphs_phase[2];
  gr2_phase[3]    := graphs_phase[3];
  gr2_phase[4]    := graphs_phase[4];
  temp1         := gr2_phase[3];
  temp2         := gr2_phase[4];
  scale_phase[1]  := temp1/(x_phase[1]-x_phase[2]);
  scale_phase[2]  := temp2/(y_phase[1]-y_phase[2]);
  centre_phase[1] := gr2_phase[1]+(gr2_phase[3]-calaxe(x_phase,gr2_phase[3]));
  centre_phase[2] := gr2_phase[2]-(gr2_phase[4]-calaxe(y_phase,gr2_phase[4]));
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure axe(g:graphdim;c:centrefactor);                          {Plot axes}
 begin
  line(g[1],round(c[2]),(g[1]+g[3]),round(c[2]));
  line(round(c[1]),g[2],round(c[1]),(g[2]-g[4]));
 end;

{ +++ ------------------------------------------------------------------ +++ }
```

```
begin                                          {Initialization section of the unit}
 size := 256;                                   {Effective # of pts in data arrays}
 exit:= False;
end.                                                        {END OF UNIT RECT_SCALING}

{*****************************************************************************}

unit plot_unt;
{*****************************************************************************}
{ A unit to plot the mag and/or phase response versus frequency in a          }
{ rectangular format. It includes the real-time data-aquisition routine       }
{*****************************************************************************}

{$O+,F+}                          {Compiler options for potentially Overlaid units}

{*****************************************************************************}
interface
{*****************************************************************************}

uses
 dos,crt,graph,rect_scaling,menu_utility,math;

type
 fftpoints  = array[0..max_array_size] of longint;
 small_fftpoints  = array[0..max_array_size] of integer;
 microwave_standards = (Short,Open,Thru,S_Load,F_Load);
 com_fftpoints = array[0..max_array_size] of complex;

var
 Amp,Phase:fftpoints;
 Complex_Thru:com_fftpoints;
 gain_amp_channel,gain_phase_channel:real;
 option:integer;
 Full_Error_Correction_active:boolean;
 standard:microwave_standards;
 x_cord,y_cord:integer;
 Statement:string;
 Edf_Amp,Edf_Phase:fftpoints;
 Amp_cal_Open,Phase_cal_Open:fftpoints;
 Amp_cal_Short,Phase_cal_Short:fftpoints;
 Load_Cal:com_fftpoints;                                  {Also used to store Edf}
 Short_cal:com_fftpoints;                                 {Also used to store Esf}
 Open_Cal:com_fftpoints;                                  {Also used to store Erf}
 Esf:com_fftpoints;
 temp_a,temp_p:real;
 Erf:com_fftpoints;
 Complex_value:com_fftpoints;

procedure rect_plot(s_par_calibration_flag:S_Par_type);
procedure cursor_off(bottom,top:byte);
procedure Enable_IRQx(irq:byte);
procedure Disable_IRQx(irq:byte);
procedure Data_Aquisition(VAR Amp,phase:fftpoints);
procedure Display_headings(x,y:integer;Message:string;VAR ch:char);
procedure read_gain;
procedure RealTime_Overlay;                    {Procedure used by RealTimePlot}
procedure freq_response_only_cal;             {performs freq-response only cal}

{*****************************************************************************}
```

```
implementation
{*****************************************************************************}

var                                                  {variables local to this unit}
 i,xmax,ymax:integer;
 Ascii_Key,Ch:char;
 xp:small_fftpoints;
 OutF:Text;
 db_div,degree_div:integer;
 db_str,degree_str:string;
 outline1:pointer;
 size1:word;

{ +++ ------------------------------------------------------------------ +++ }

procedure transmission_freq_response_cal;
{*****************************************************************************}
{ A procedure to display prompts for performing a transmission response.      }
{*****************************************************************************}
 begin
  standard := thru;                    {The calibration standard used = a thru}
  x_cord := 22;y_cord := 10;                 {text co-ordinates of the heading}
  Statement := 'Connect A "Thru" between the Test ports';
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure reflection_freq_response_cal;
{*****************************************************************************}
{ A procedure to display prompts for performing a reflection response.        }
{*****************************************************************************}
 begin
  standard := short;                  {The calibration standard used = a short}
  x_cord := 24;y_cord := 10;            {text-mode co-ordinates of the heading}
  if s_par_flag = S11 then Statement := 'Connect A Short Circuit to port 1'
  else Statement := 'Connect A Short Circuit to port 2';
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Display_headings(x,y:integer;Message:string;VAR ch:char);
{*****************************************************************************}
{A procedure to display the prompts for entering the microwave standards in   }
{ order to perform a full 8-term calibration.                                 }
{*****************************************************************************}
 begin
  clrscr;
  Write('                                                                    ');
  Write('                    MICROWAVE CALIBRATION                           ');
  Write('                                                                    ');
  GotoXY(1,21);
  Write('                                                                    ');
  Write('                    Hit RETURN to continue                          ');
  Write('                         <ESC> quits                                ');
  Write('                                                                    ');
 repeat
  repeat
    InvVideo_Msg(Message,x,y);       {This message prompts to connect a certain}
    delay(700);                                          {microwave standard}
    NormVideo_Msg(Message,x,y);
    delay(700);
```

```
   until keypressed;                          {and the prompt flashes every 700ms}
   ch := readkey;                                   {ESC to exit, return continues}
  until (ch = #13) OR (ch = #27);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure FullPort;
{*****************************************************************************}
{ Set the viewport in graphics mode to the full screen.                       }
{*****************************************************************************}
 begin
  SetViewPort(0,0,719,347,ClipOn);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure cursor_off(bottom,top:byte);
{*****************************************************************************}
{ Control the cursor in text-mode. By selecting different variables as the    }
{ arguments bottom and top, you can draw different shapes of cursors and      }
{ you can clear the cursor completely                                         }
{*****************************************************************************}
 var
  Regs:registers;
 begin
  Regs.ah := 1;
  Regs.al := 0;
  if (top = 0) and (bottom = 0)then regs.ch := 32
  else begin
        regs.ch := top;
        regs.cl := bottom;
       end;
  intr(16,regs);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Enable_IRQx(irq:byte);
{*****************************************************************************}
{ This is a procedure to enable the IRQ line number irq and to store the      }
{ Interrupt Vector of Interrupt service routine number X in the Interrupt     }
{ Vector table IVT.                                                           }
{*****************************************************************************}
 var
  imr,mask:integer;
   begin
    mask       := not ( 1 shl IRQ );
    imr        := PORT [$21];          {get Interrupt mask register frpm 8259}
    imr        := imr and mask;                        {clear mask bit of IRQ}
    port[$21]  := imr;                              {and return to controller}
   end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Disable_IRQx(irq:byte);
{*****************************************************************************}
{ This is a procedure to disable the IRQ line number irq and to store the     }
{ Interrupt Vector of Interrupt service routine number X in the Interrupt     }
{ Vector table IVT.                                                           }
```

```
{*****************************************************************************}
 var
  imr, mask    : INTEGER;
 begin
  mask        :=  ( 1 shl IRQ );
  imr         := PORT [$21];            {get Interrupt mask register from 8259}
  imr         := imr or mask;                          {set mask bit of IRQ}
  port[$21]   := imr;                          {and return to controller}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Scale_Amp_Points(VAR Amp:fftpoints);
{*****************************************************************************}
{ A procedure to scale points for the rectangular mag axes.                   }
{*****************************************************************************}
 begin
  for i := 0 to size do
    begin
     xp[i] := round((i-centre_mag[3])*scale_mag[1]+centre_mag[1]);
     Amp[i] := round(centre_mag[2]-(Amp[i]-centre_mag[4])*scale_mag[2]);
    end;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Scale_Phase_Points(VAR Phase:fftpoints);
{*****************************************************************************}
{ A procedure to scale points for the rectangular phase axes.                 }
{*****************************************************************************}
 begin
  for i := 0 to size do
    begin
     xp[i] := round((i-centre_phase[3])*scale_phase[1]+centre_phase[1]);
     Phase[i] := round(centre_phase[2]-(Phase[i]-centre_phase[4])
     *scale_phase[2]);
    end;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Set_dB_div(d,option:integer);
{*****************************************************************************}
{ Procedure to compute the dB's/division on the mag plot and to plot these    }
{ values next to the grid lines on the display.                               }
{*****************************************************************************}
 var
  y_value,total:integer;

 begin
  {Print the dB's/div on the right side of the grid}
    SetTextStyle(2,0,4);                                {Select Small Text}
    total := 0;                                          {initialize total}
    Str(d,db_str);
    y_value := 0;
    OutTextXY(516,-2,'+'+db_str);                  {output the top value +}
    Case option of
     1: begin
         for i := 1 to 8 do                       {output the other values -}
```

```
          begin                                    {This option is for a single}
           Str(total,db_str);                                          {display}
           y_value := y_value + 43;
           OutTextXY(516,y_value-5,db_str);
           total := total - d;
          end;
        end;
     3: begin
         for i := 1 to 8 do                        {output the other values -}
          begin
           Str(total,db_str);                      {This option is for a dual}
           y_value := y_value + 21;                                    {display}
           OutTextXY(516,y_value-5,db_str);
           total := total - d;
          end;
        end;
     end;
     str(10,db_str);
    SetTextStyle(0,0,1);                                      {Reset TextStyle}
  end;

{ +++ ------------------------------------------------------------------- +++ }

procedure Set_degrees_div(option:integer);
{*****************************************************************************}
{ Procedure to compute the degrees's/division on the phase plot and to plot   }
{ these values next to the grid lines on the display.                         }
{*****************************************************************************}
 var
  y_value:integer;
  max_phase:real;
 begin
  {Print the degrees's/div on the right side of the grid}
    SetTextStyle(2,0,4);                                    {Select Small Text}
    max_phase := y_phase[1]/scale;
    degree_div := round(((y_phase[1]-y_phase[2])/8)/(scale));
    Str(max_phase:6:1,degree_str);
    Case option of
     2: begin                              {This is for a Single display of phase}
         y_value := 0;
         OutTextXY(512,-2,degree_str);       {Write the degrees on the screen}
         for i := 1 to 8 do
          begin
           max_phase := max_phase - degree_div;
           y_value := y_value + 43;
           Str(max_phase:6:1,degree_str);
           OutTextXY(512,y_value-5,degree_str);
          end;
         end;
     3: begin                                {This is for a dual display of phase}
         y_value := 179;
         OutTextXY(512,177,degree_str);
         for i := 1 to 8 do
          begin
           max_phase := max_phase - degree_div;
           y_value := y_value + 21;
           Str(max_phase:6:1,degree_str);
           if i = 8 then OutTextXY(512,y_value-9,degree_str)
           else OutTextXY(512,y_value-5,degree_str);
          end;
```

```
        end;
    end;
                                                    {Refresh the degrees/div}
    SetTextStyle(0,0,1);
    SetViewPort(563,115,718,135,False);
    ClearViewPort;          {Clear the small area that houses the degrees/div}
    Str(degree_div,degree_str);
    OutTextXY(1,10,'Y Scale  '+degree_str+'deg/div');
    FullPort;                                       {Reset the graphics screen}
  end;

{ +++ ------------------------------------------------------------------ +++}

procedure Auto_scale_phase_axes;
{*************************************************************************}
{ Procedure to automatically scale the y-axis of the phase response. The  }
{ array of phase is interrogated and the max and minimum phases are used  }
{ PLUS a margin of 45 degrees to determine the axes values.  At this point }
{ the phase array is the difference between the cal and the phase arrays and }
{ is scaled up by a factor SCALE. Note that phase is now expressed in     }
{ DEGREES*SCALE and no longer as a level.                                 }
{*************************************************************************}
 begin
  y_phase[1] := Phase[0]+scale*45;                {45 degrees as a top margin}
  y_phase[2] := Phase[size]-scale*45;            {45 degrees as a lower margin}
  y_phase[3] := y_phase[1];
  if (y_phase[1]-y_phase[2])<180*scale then        {Ensure a minimum scale of}
   begin                                                  {approx 360 degrees}
    y_phase[1] := y_phase[1]+180*scale;
    y_phase[2] := y_phase[2]-180*scale;
    y_phase[3] := y_phase[1];
   end;
  phase_scaling(x_phase,y_phase,graphs_phase,scale_phase,centre_phase);
  Set_degrees_div(option);                  {Plot the numbers next to the grid}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure single_outline;
{*************************************************************************}
{ Procedure to draw the axes and all the menus that are associated with them }
{ for the rectilinear plots. This is for a single display of mag or phase  }
{*************************************************************************}

 var
  y_value,x_value:integer;
  total:integer;
 begin
  y_value := 0;x_value := 0;
  rectangle(0,0,510,344);
  rectangle(558,0,719,330);
  rectangle(558,335,719,347);

  SetLineStyle(1,0,1);                                    {Select Dotted Line}
  for i := 1 to 7 do                              {Generate the grid background}
   begin
    y_value := y_value + 43;
    line(0,y_value,510,y_value);
   end;
  for i := 1 to 10 do                             {Generate the grid background}
```

```
   begin
    x_value := x_value + 51;
    line(x_value,0,x_value,344);
   end;
  SetLineStyle(0,0,1);                                    {Reset Line Style}

  Case option of                              {plot the db/division next to grid}
   1: begin
       db_div := 10;
       Set_dB_div(db_div,option);
      end;
   2: inline($90);                            {degrees/div is automatically done}
  end;

  SetTextStyle(2,0,4);
  OutTextXY(571,336,'Hit <ESC> for Main Menu');
  Line(558,30,719,30);
  SetTextStyle(1,0,3);
  Case option of
   1: begin
       OutTextXY(568,0,'MAGNITUDE');
       SetTextStyle(0,0,1);                               {Reset TextStyle}
       Str(db_div,db_str);
       OutTextXY(562,125,'Y Scale  '+db_str+'dB/div');
      end;
   2: begin
       OutTextXY(595,0,'PHASE');
       SetTextStyle(0,0,1);                               {Reset TextStyle}
       Str(degree_div,degree_str);
       OutTextXY(562,125,'Y Scale  '+degree_str+'deg/div');
       OutTextXY(562,235,' A  To  Autoscale');
      end;
   3: inline($90);
  end;

  Size1 := ImageSize(0,0,510,344);          {Save the rect plot grid shape for}
  GetMem(outline1,size1);                                {refreshing purposes}
  GetImage(0,0,510,344,outline1^);

  delta_f := (f_stop-f_start)/10;          {Display the current configurations}
  Str(delta_f:6:4,delta_f_str);
  OutTextXY(562,40,'fa       '+start_str+' GHz'); {Display f_start and f_stop}
  OutTextXY(562,55,'fb       '+stop_str+' GHz');
  OutTextXY(562,70,'delta-f  '+delta_f_str+' GHz');          {Display delta_f}
  Case s_par_flag of
   S11: OutTextXY(562,95,'S-Parameter     S11');          {Display the active}
   S21: OutTextXY(562,95,'S-Parameter     S21');                {s-parameter}
   S22: OutTextXY(562,95,'S-Parameter     S22');
   S12: OutTextXY(562,95,'S-Parameter     S12');
  end; {Case}
  if (Full_Error_Correction_Active) then        {Has a full-cal been selected}
   OutTextXY(562,155,'Full Correction  ON')                          {YES}
  else
   OutTextXY(562,155,'Full Correction OFF');                         {NO}
  OutTextXY(562,170,'("T" To Toggle)');          {Prompt to toggle correction}
  OutTextXY(562,315,'<P/S><1> = Hardcopy');        {Prompt to obtain hardcopy}
 end;

{ +++ ------------------------------------------------------------------ +++ }
```

```
procedure dual_outline;
{*****************************************************************************}
{ Procedure to draw the axes and all the menus that are associated with them }
{ for the rectilinear plots. This is for a dual display of mag and phase     }
{*****************************************************************************}
var
 i,y_value,x_value:integer;
 begin
  y_value := 0;x_value := 0;
  rectangle(0,0,510,168);
  rectangle(0,179,510,347);
  rectangle(558,0,719,330);
  rectangle(558,335,719,347);
  SetLineStyle(1,0,1);                                    {Select Dotted Line}
  for i := 1 to 7 do                                {Generate the grid background}
   begin
    y_value := y_value + 21;
    line(0,y_value,510,y_value);
   end;
  y_value := 179;
  for i := 1 to 7 do
   begin
    y_value := y_value + 21;
    line(0,y_value,510,y_value);
   end;
   for i := 1 to 10 do                              {Generate the grid background}
   begin
    x_value := x_value + 51;
    line(x_value,0,x_value,168);
   end;
  x_value := 0;
  for i := 1 to 10 do
   begin
    x_value := x_value + 51;
    line(x_value,179,x_value,347);
   end;
  SetLineStyle(0,0,1);                                    {Reset Line Style}

  SetTextStyle(2,0,4);
  OutTextXY(571,336,'Hit <ESC> for Main Menu');
  Line(558,30,719,30);
  SetTextStyle(1,0,3);
  Case option of
   1: inline($90);
   2: inline($90);
   3: OutTextXY(559,0,'MAG + PHASE');
  end;
  SetTextStyle(0,0,1);                                    {Reset TextStyle}

  Size1 := ImageSize(0,0,510,344);        {Save the grid images for refreshing}
  GetMem(outline1,size1);
  GetImage(0,0,510,344,outline1^);
  delta_f := (f_stop-f_start)/10;          {Display the current configurations}
  Str(delta_f:6:4,delta_f_str);
  OutTextXY(562,40,'fa       '+start_str+' GHz');
  OutTextXY(562,55,'fb       '+stop_str+' GHz');
  OutTextXY(562,70,'delta-f  '+delta_f_str+' GHz');
  Case s_par_flag of
   S11: OutTextXY(562,95,'S-Parameter     S11');  {Display active s-parameter}
   S21: OutTextXY(562,95,'S-Parameter     S21');
```

```
   S22: OutTextXY(562,95,'S-Parameter     S22');
   S12: OutTextXY(562,95,'S-Parameter     S12');
  end; {Case}
  Set_dB_div(db_div,option);
  db_div := 10;
  str(db_div,db_str);
  OutTextXY(562,125,'Y Scale  '+degree_str+'deg/div');
  OutTextXY(562,140,'Y Scale  '+db_str+'dB/div');
  if (Full_Error_Correction_Active) then            {Display various prompts}
   OutTextXY(562,155,'Full Correction  ON')
  else
   OutTextXY(562,155,'Full Correction OFF');
  OutTextXY(562,170,'("T" To Toggle)');          {Prompt to toggle correction}
  OutTextXY(562,235,' A  To  Autoscale');          {Prompt to Autoscale phase}
  OutTextXY(562,315,'<P/S><1> = Hardcopy');        {Prompt to obtain hardcopy}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure options;
{*************************************************************************}
{ Procedure which calls the routines necessary to generate the plots for  }
{ single or dual plots of mag and/or phase                                }
{*************************************************************************}
 begin
  Case option of
   1: begin                                                {Mag only option}
       single_outline;
       mag_scaling(x_mag,y_mag,graphs_mag,scale_mag,centre_mag);
      end;
   2: begin                                              {Phase only option}
       single_outline;
       phase_scaling(x_phase,y_phase,graphs_phase,scale_phase,centre_phase);
      end;
   3: begin                                          {Mag and Phase option}
       dual_outline;
       mag_phase_constants;
       mag_scaling(x_mag,y_mag,graphs_mag,scale_mag,centre_mag);
       phase_scaling(x_phase,y_phase,graphs_phase,scale_phase,centre_phase);
      end;
  end;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Toggle_cal;
{*************************************************************************}
{ Procedure to toggle the status of Full error correction                 }
{*************************************************************************}
 begin
  SetViewPort(562,155,716,162,ClipOn);          {The status is contained in a}
  ClearViewPort;                                              {small viewport}
  FullPort;                                                   {Reset viewport}
  if (Full_Error_Correction_active) then     {Write the new correction status}
    begin                                                  {in the small viewport}
      OutTextXY(562,155,'Full Correction OFF');                {Correction is OFF}
      Full_Error_Correction_active := FALSE;  {Set flag for correction status}
    end
  else
```

```
     begin
      OutTextXY(562,155,'Full Correction  ON');              {Correction is ON}
      Full_Error_Correction_active := TRUE;    {Set flag for correction status}
     end;
  end;

 { +++ ------------------------------------------------------------------ +++ }

procedure Service_Key(s_par_calibration_flag:S_Par_type);
{*****************************************************************************}
{ Service a function key if pressed. Also update all the relevant info that  }
{ is displayed on the screen if a function key has been pressed.              }
{*****************************************************************************}
 begin
   If Ascii_Key = #027 then                                          {ESC exits}
    begin
     exit := True;
     Autoscale := FALSE;                              {Reset the autoscale flag}
    end;
   If ((Ascii_Key = 'T')OR(Ascii_Key = 't')) AND (s_par_calibration_flag <>
   OFF) then
     Case s_par_calibration_flag of          {T indicates a correction toggle}
      S11: If s_par_flag = S11 then Toggle_cal;            {has been selected}
      S21: If s_par_flag = S21 then Toggle_cal;
      S12: If s_par_flag = S12 then Toggle_cal;
      S22: If s_par_flag = S22 then Toggle_cal;
      OFF: inline($90);
     end;
   If (Ascii_Key = 'A') or (Ascii_Key = 'a') then         {autoscale selected}
    begin
     exit := True;
     Autoscale := TRUE;                             {Set the autoscale flag}
    end;
 end;

{ +++ ------------------------------------------------------------------ +++ }

Procedure Phase_conversion(Mp,Lp,Ma,La:small_fftpoints;VAR P,A:fftpoints);
{*****************************************************************************}
{ Convert Phase from its plus minus 180 degrees format to a continuous form  }
{ Inputs  Mp = The input phase array Msbs                                     }
{         Lp = The input phase array Lsbs                                     }
{         Ma = The input amp array Msbs                                       }
{         La = The input amp array Lsbs                                       }
{ Outputs P  = The output Phase array in continuous format(levels*scale)      }
{         A  = The output masked Amp array (levels)                           }
{*****************************************************************************}
 var
  n:array[-1..max_array_size] of integer;
  i:integer;
 begin
  for i := 0 to size do
   begin
    P[i] := ((Mp[i] AND $0F)*256)+Lp[i];              {Mask Amp and Phase data}
    A[i] := ((Ma[i] AND $0F)*256)+La[i];
   end;
  n[-1] := 0;
  for i := 0 to (size-1) do
   begin
    If abs((P[i+1]-P[i])) > (180*gain_Phase_channel*4.096) then n[i] :=
```

```
    n[i-1] + 1
    Else n[i] := n[i-1];
    P[i] := Round(scale*((P[i]-n[i-1]*(360*gain_Phase_channel*4.096))));
   end;
  P[size]:= Round(scale*((P[size]-n[size-1]*(360*gain_Phase_channel*4.096))));
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Data_Aquisition(VAR Amp,phase:fftpoints);
{*****************************************************************************}
{ Procedure to collect 256 points of amplitude(ch1) and phase(ch2) via ADC    }
{*****************************************************************************}
 const
  time_delay = 110;                              {adjust sampling to clock speed}
 var                                  {This is suitable for 12MHz 0 wait states}
  Sample_No,i,count:integer;
  Amp_Msb,Amp_Lsb,Phase_Msb,Phase_Lsb:small_fftpoints;
  OutF:Text;

 begin
  Disable_IRQx(0);                              {Disable all interrupts to prevent}
                                                                 {trace jitter}
  port[$263] := $01;                                {Trigger Amp-Sweep - Bit C0}
  repeat until (port[$262] AND $10 = $00);     {Loop until Sweep starts proper}
   delay(1);                                        {Start at same time as 8410}
   for Sample_No := 0 to size do                          {Collect 256 samples}
    begin
      port[$702] := $12;
      port[$702] := $13;
      for i := 1 to 9 do
       begin
        inline($90/$90/$90/$90/$90/$90/$90/$90);        {generate wait states}
       end;
      Amp_Msb[Sample_No] := port[$701];                                  {MSB}
      Amp_Lsb[Sample_No] := port[$700];                                  {LSB}
      for i := 1 to time_delay do inline($90);
      count := 0;
      repeat
       count := count + 1;                          {counter to prevent lockup}
      until (port[$262] AND $10 = $00) or (count = 100);
    end;
    port[$263] := $00;                                            {Reset Sweep}
{*****************************************************************************}
{ A very important delay. The Sweeper has a dead-time associated with it. It  }
{ takes approx 140ms to settle after a sweep has been reset.                  }
{*****************************************************************************}
    delay(140);
    inline($90);
    port[$263] := $01;                                        {Start Phase Sweep}
   repeat until port[$262] AND $10 = $00;      {Loop until Sweep starts proper}
    delay(1);                                        {Start at same time as 8410}
    for Sample_No := 0 to size do
     begin
      port[$702] := $22;
      port[$702] := $23;
      for i := 1 to 9 do
       begin
        inline($90/$90/$90/$90/$90/$90/$90/$90);
       end;
      Phase_Msb[Sample_No] := port[$701];
```

```
      Phase_Lsb[Sample_No] := port[$700];
      for i := 1 to time_delay do inline($90);
      count := 0;
      repeat
       count := count + 1;
      until (port[$262] AND $10 = $00) or (count = 100);
    end;
    port[$263] := $00;                                          {Reset Sweep}
    Enable_IRQx(0);                                  {Enable system interrupts}
                                                {Convert phase to continuous form}
    Phase_Conversion(Phase_Msb,Phase_Lsb,Amp_Msb,Amp_Lsb,Phase,Amp);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure RealTime_Overlay;
{*************************************************************************}
{ Procedure used by RealTimePlot to perform scaling and manipulation on data }
{*************************************************************************}
 var
  num,den:complex;
 begin
  Case Full_Error_Correction_active of
    TRUE: begin
          If s_par_calibration_flag <> OFF then
            begin
                                        {Convert mag+phase to complex numbers}
              for i := 0 to size do
               begin
                Complex_value[i].Re := -(0.2*Amp[i])/temp_a;
                Complex_value[i].Re := Power(10,Complex_value[i].Re/20);
                Complex_value[i].Im := (Phase[i]/scale)/temp_p;
               end;
          Case s_par_calibration_flag of
           S11,S22: begin                                {8 term calibration}
                     num.CmplxType := Polar;
                     den.CmplxType := Polar;
                     for i := 0 to size do
                      begin
                       CmplxSub(Complex_value[i],Load_Cal[i],num);
                       CmplxMult(num,Esf[i],den);
                       CmplxAdd(den,Erf[i],den);
                       CmplxDiv(num,den,Complex_value[i]);
                       Amp[i] := Round(20*LOG10(Complex_value[i].Re)*scale);
                       Phase[i] := Round(Complex_value[i].Im*scale);
                      end;
                     end;
          S12,S21: If (s_par_flag = S21)OR(s_par_flag = S12) then
                     for i := 0 to size do
                      begin
                       Phase[i] := Round((Phase[i]-Phase_Cal_Short[i])/
                       (temp_p));
                       Amp[i] := -Round(scale*((0.2*(Amp_Cal_Short[i]-
                       Amp[i]))/(temp_a)));
                      end;
              OFF: inline($90);
             end; {Case}
            end;
          end;
    FALSE:              {A Simple freq-response only is performed by default}
```

```
            begin
             for i := 0 to size do
              begin
               Amp[i] := -Round(scale*((0.2*(Amp_Cal_Short[i]-Amp[i]))/
               (4.096*gain_amp_channel)));
               if (s_par_flag = S11)OR(s_par_flag = S22) then
                Phase[i] := Round((Phase[i]-(Phase_Cal_Short[i]-180*
                temp_p*scale))/(temp_p))
               else
                Phase[i] := Round((Phase[i]-Phase_Cal_Short[i])/(temp_p));
              end;
             end;
    end;{Case}
end;

{ +++ ------------------------------------------------------------------ +++ }

procedure RealTimePlot(s_par_calibration_flag:S_Par_type);
{*************************************************************************}
{ Procedure to plot phase and amplitude in rectangular form in real-time  }
{*************************************************************************}
 var
  i:integer;
  TempAmp,TempPhase:fftpoints;

 begin
  delay(500);                              {delay prevents lock up due to Pos z}
  Data_Aquisition(Amp,Phase);                                  {Collect 256 pts}
  j := 0;
  Case option of
   1: begin
       RealTime_Overlay;
       Scale_Amp_Points(Amp);                                {Scale pnts for axes}
       MoveTo(xp[0],Amp[0]);                                {move to the first point}
       for i := 1 to size do LineTo(xp[i],Amp[i]);                      {plot pnts}
        repeat
         Write(GPIO,'t3');          {Program External sweep trigger condition}
          repeat
           TempAmp := Amp;                                        {Store old array}
           Data_Aquisition(Amp,Phase);                    {Collect 256 new pnts}
           RealTime_Overlay;
           Scale_Amp_points(Amp);                                      {Scale them}
           for i := 0 to size-1 do
            begin                                          {Replace old pts 1 by 1}
             SetColor(0);                                      {thereby creating a}
             Moveto(xp[i],TempAmp[i]);                           {real-time effect}
             LineTo(xp[i+1],TempAmp[i+1]);
             SetColor(15);                                  {Plot new pnts 1 by 1}
             Moveto(xp[i],Amp[i]);
             LineTo(xp[i+1],Amp[i+1]);
           end;
           j := j + 1;                                                {Count Sweeps}
           if j = 10 then                           {Refresh grid every 10 sweeps}
            begin
             PutImage(0,0,outline1^,Orput);
             j := 0;                                                {Reset Counter}
            end;
          until keypressed;                                    {Keep on plotting!}
          Ascii_Key := ReadKey;
```

```
      If Ascii_Key = #027 then exit := True;                    {ESC exits}
      if ((Ascii_Key = 'T')OR(Ascii_Key = 't'))then
      Service_Key(s_par_calibration_flag);
    until exit = True;
   end;
2: begin                                {Repeat of above but for phase only}
    RealTime_Overlay;
    Auto_scale_phase_axes;              {Auto Scale points for Phase y-axis}
    Scale_Phase_points(Phase);
    MoveTo(xp[0],Phase[0]);
    for i := 1 to size do LineTo(xp[i],Phase[i]);                {plot pnts}
     repeat
      Write(GPIO,'t3');          {Program External sweep trigger condition}
      repeat
       TempAmp := Phase;                                    {Store old array}
       Data_Aquisition(Amp,Phase);                  {Collect 256 new pnts}
       RealTime_Overlay;
       Scale_Phase_Points(Phase);                                {Scale them}
       for i := 0 to size-1 do
        begin                                       {Replace old pts 1 by 1}
         SetColor(0);
         Moveto(xp[i],TempAmp[i]);
         LineTo(xp[i+1],TempAmp[i+1]);
         SetColor(15);                               {Plot new pnts 1 by 1}
         Moveto(xp[i],Phase[i]);
         LineTo(xp[i+1],Phase[i+1]);
        end;
        j := j + 1;                                          {Count Sweeps}
        if j = 10 then                        {Refresh grid every 10 sweeps}
         begin
          PutImage(0,0,outline1^,Orput);
          j := 0;                                          {Reset Counter}
         end;
      until keypressed;                                  {Keep on plotting!}
     Ascii_Key := ReadKey;
    If Ascii_Key = #027 then  exit := True;                  {<ESC> Exits}
    if ((Ascii_Key = 'T')OR(Ascii_Key = 't'))OR(Ascii_Key = 'A') or
    (Ascii_Key = 'a') then Service_Key(s_par_calibration_flag);
   until exit = True;
   end;
3: begin                              {Same as above but for dual displays}
    RealTime_Overlay;
    Auto_scale_phase_axes;                       {Autoscale the phase data}
    Scale_Amp_Points(Amp);                                {Scale the phase}
    Scale_Phase_Points(Phase);                              {Scale the Amp}
    MoveTo(xp[0],Amp[0]);
    for i := 1 to size do LineTo(xp[i],Amp[i]);  {Plot first array of data}
    MoveTo(xp[0],Phase[0]);
    for i := 1 to size do LineTo(xp[i],Phase[i]);
     repeat
      Write(GPIO,'t3');          {Program External sweep trigger condition}
      repeat
       TempAmp := Amp;
       TempPhase := Phase;
       Data_Aquisition(Amp,Phase);
       RealTime_Overlay;
       Scale_Amp_Points(Amp);
       Scale_Phase_Points(Phase);
       for i := 0 to size-1 do
        begin
```

```
            SetColor(0);
            Moveto(xp[i],TempAmp[i]);
            LineTo(xp[i+1],TempAmp[i+1]);
            Moveto(xp[i],TempPhase[i]);
            LineTo(xp[i+1],TempPhase[i+1]);
            SetColor(15);                                  {Plot new pnts 1 by 1}
            Moveto(xp[i],Amp[i]);
            LineTo(xp[i+1],Amp[i+1]);
            Moveto(xp[i],Phase[i]);
            LineTo(xp[i+1],Phase[i+1]);
           end;
           j := j + 1;                                                {Count sweeps}
           if j = 10 then                          {Refresh grid every 10 sweeps}
            begin
             PutImage(0,0,outline1^,Orput);
             j := 0;                                          {Reset the counter}
            end;
         until keypressed;                                    {Keep on plotting!}
         Ascii_Key := ReadKey;
         If Ascii_Key = #027 then  exit := True;                  {<ESC> Exits}
         if ((Ascii_Key = 'T')OR(Ascii_Key = 't'))or(Ascii_Key = 'A') or
         (Ascii_Key = 'a') then Service_Key(s_par_calibration_flag);
       until exit = True;
      end;
    end; {Case}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure read_gain;
{*************************************************************************}
{ Procedure to read the values of the gains on Amp and Phase channels. These }
{ gains are computed and written to files which can be read by the main      }
{ program.                                                                  }
{*************************************************************************}
var
 InF:text;
begin
  Assign(InF,'ATHRU.DAT');                        {Initialize the Mag channel}
  Reset(InF);
  Read(InF,gain_amp_channel);
 {Check for faulty value of gain}
 {NOTE:IF THE GAINS OF THE LM101 AMPS ARE CHANGED, THESE LIMITS WILL HAVE TO }
 {CHANGE}
  if (gain_amp_channel < 1) OR (gain_amp_channel > 3) then gain_amp_channel
  := 1.7236328125;
  Close(InF);
  Assign(InF,'PTHRU.DAT');                       {Initialize the Phase channel}
  Reset(InF);
  Read(InF,gain_phase_channel);
 {Check for faulty value of gain}
  if (gain_phase_channel < 1) OR (gain_phase_channel > 3) then
  gain_phase_channel := 1.7089843750;
  Close(InF);
end;

{ +++ ------------------------------------------------------------------ +++ }
```

```
procedure freq_response_only_cal;
{*****************************************************************************}
{ Procedure to perform a frequency response-only calibration.                 }
{*****************************************************************************}
 var
  ch:char;
 begin
  Window(1,1,80,25);
  if (freq_response_cal) then            {Perform a freq response cal only ??}
   begin
    Case s_par_flag of
     S11,S22: reflection_freq_response_cal;                  {Reflection cal}
     S21,S12: transmission_freq_response_cal;               {Transmission cal}
    end;
    display_headings(x_cord,y_cord,statement,ch);            {Display prompt}
    data_aquisition(Amp_Cal_Short,Phase_Cal_Short);          {Sample the data}
    Window(40,9,80,25);
    freq_response_cal := False;                              {Reset the flag}
   end;
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure rect_plot(s_par_calibration_flag:S_Par_type);
{*****************************************************************************}
{ The main procedure in Unit PLOT_UNT. This procedure calls all others        }
{ necessary to plot rectilinear displays of mag and/or phase.                 }
{*****************************************************************************}
 begin
   freq_response_only_cal;                      {Perform freq response-only cal}
   db_div := 10;                                                 {default db_div}
   repeat
     initialize;
     options;                                                     {Plot the axis}
      repeat
        Write(GPIO,'t3');           {Program External sweep trigger condition}
        realtimeplot(s_par_calibration_flag);     {Plot the data in real-time}
        Service_Key(s_par_calibration_flag);       {If special key is pressed}
      until (exit = True);                                          {Exit code}
    FreeMem(outline1,size1);        {Free the dynamic variables from the heap}
    exit := False;
   until NOT(autoscale);        {If autoscale selected then continue plotting}
   RestoreCrtMode;                                        {Return to text mode}
   Full_Error_Correction_active := FALSE;
   cursor_off(0,0);                                      {Switch off the cursor}
 end;

{ +++ ------------------------------------------------------------------ +++ }

end.                                                    {END OF UNIT PLOT_UNT}
```

```
unit Smithcha;
{*****************************************************************************}
{ A unit to plot a Smithchart and display real-time data on it. This          }
{ real-time data is in the form of sampled mag and phase information which    }
{ is obtained from a HP8410C network analyzer. The smithchart can be zoomed   }
{ into all four quadrants and a cursor is availiable to return the complex    }
{ impedance values at any point on the chart.                                 }
{*****************************************************************************}

{$O+,F+}          {Compiler options necessary for potentially overlaid units}

{*****************************************************************************}
interface
{*****************************************************************************}

Uses Crt,Graph,Math,menu_utility,rect_scaling,plot_unt;          {Units used}

procedure main(start_str,stop_str:string;s_par_flag:s_par_type);

{*****************************************************************************}
implementation
{*****************************************************************************}

const
  StartX = 200;                                  {initial co-ordinates of the}
  StartY = 10;                                          {cursor on the chart}

var
  GraphDriver,GraphMode:integer;
  XSideLength,YSideLength:integer;
  XScale,YScale:Real;
  i,j:Integer;
  Cursor,smith,quad1,quad2,quad3,quad4:Pointer;
  Size2,size4,sizequad1,sizequad2,sizequad3,sizequad4:Word;
  Xa,Ya,inc:integer;
  ulx,uly,lrx,lry:word;
  MaxX,MaxY:word;
  ANS,Ch:char;
  Xmargin:word;
  Xzmin,Xzmax,Yzmin,Yzmax:integer;
  zoomflag:integer;
  xtemp,ytemp:integer;
  Real_phase,Real_mag:array[0..max_array_size] of real;
  exit_smith:boolean;
  xval,yval:fftpoints;

{ +++ ------------------------------------------------------------------ +++ }

procedure dataentry;
{*****************************************************************************}
{ Procedure to sample a set of mag and phase points and scale them for the    }
{ Smithchart.                                                                 }
{*****************************************************************************}
 var
  com:complex;
 begin
  data_aquisition(amp,phase);                          {Sample mag and phase}
  RealTime_Overlay;                  {Decide on whether error corrected}
  for i := 0 to size do
```

```
    begin                                  {Manipulate the data for the smithchart}
     com.Re := Amp[i]/scale;                       {ie. Convert dB's to linear etc}
     com.Im := Phase[i]/scale;
     com.Re := Power(10,com.Re/20);
     com.CmplxType := Polar;
     PolarToRect(com,com);
     Real_mag[i] := com.Re;
     Real_phase[i] := com.Im;
    end;

  for i := 0 to size do
   begin
    xval[i] := round((Real_mag[i]+1)*50*XScale);              {Scale the data}
    yval[i] := round((1-Real_phase[i])*50*YScale);
   end;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure fullport;
{*****************************************************************************}
{ Set the graphics viewport to the entire screen.                             }
{*****************************************************************************}
 begin
  SetViewPort(0,0,MaxX,MaxY,ClipOn);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Toggle_full_correction;
{*****************************************************************************}
{ Procedure to toggle the status of the full error correction. When selected }
{ a refresh of the screen is also performed.                                  }
{*****************************************************************************}
 begin
  SetViewPort(5,5,XSideLength,YSideLength,ClipOn);            {Small viewport}
  ClearViewPort;                                                    {clear it}
  Case zoomflag of
   0: PutImage(0,0,smith^,2);                                {Refresh the chart}
   1: PutImage(0,0,quad1^,2);
   2: PutImage(0,0,quad2^,2);
   3: PutImage(0,0,quad3^,2);
   4: PutImage(0,0,quad4^,2);
  end;
  SetViewPort(XMargin,273,716,282,ClipOn);                   {reset viewport}
  ClearViewPort;                                                    {clear it}
  FullPort;
  if (Full_Error_Correction_active) then       {Toggle the correction message}
    begin
      OutTextXY(XMargin,275,'Full Correction                OFF');       {OFF}
      Full_Error_Correction_active := FALSE;
    end
  else
     begin
      OutTextXY(XMargin,275,'Full Correction                 ON');        {ON}
      Full_Error_Correction_active := TRUE;
     end;
  end;

{ +++ ------------------------------------------------------------------ +++ }
```

```
procedure initialize(VAR MaxX,MaxY:word;VAR XSidelength,YSideLength:integer);
{*************************************************************************}
{ Procedure to initialize the graphics mode and various variables used in the}
{ rest of the unit.                                                       }
{*************************************************************************}
var
 Xasp,Yasp : Word; {X + Y aspect ratios to scale circles on a graphics screen}

begin
  GraphDriver := Detect;                              {Autodetect graphics mode}
  InitGraph(GraphDriver,GraphMode,' ');                     {INIT GRAPHICS MODE}
  MaxX := GetMaxX;MaxY := GetMaxY;
  GetAspectRatio(Xasp,Yasp);
  XSideLength := Round(MaxX/2);                      {Scale the x and y lengths}
  YSideLength := Round((Xasp/Yasp)*XSideLength);              {of the Smithchart}
  ClearDevice;
  Fullport;                                                 {Reset the viewport}
  inc := 1;                                    {Default increment of the cursor}
  exit_smith := False;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure drawcursor(VAR cursor:pointer;VAR size2:word);
{*************************************************************************}
{ Draw the Cursor image and save it for later refreshing purposes.        }
{*************************************************************************}
begin
  Line(StartX+5,StartY,StartX+5,StartY+10);
  Line(StartX,StartY+5,StartX+10,StartY+5);
  ulx := StartX;uly := StartY;                              {Read Cursor Image }
  lrx := StartX+10;lry := StartY+10;
  Size2 := ImageSize(ulx,uly,lrx,lry);        {Save the cursor image in memory}
  GetMem(cursor,size2);
  GetImage(ulx,uly,lrx,lry,Cursor^);
  PutImage(ulx,uly,Cursor^,XORput);               {XORput the image - erase it}
end;

{ +++ ------------------------------------------------------------------ +++ }

procedure drawsmithchart(XSide,YSide:integer;VAR XScale,YScale:real);
{*************************************************************************}
{ Draw the Smithchart image and save it for refreshing later.             }
{*************************************************************************}
begin
  XScale := XSideLength/100;
  YScale := YSideLength/100;
  zoomflag := 0;                                      {Full chart is selected}
  XSideLength := XSideLength+5;YSideLength := YSideLength+5;
  SetViewPort(5,5,XSideLength,YSideLength,ClipOn);
  RectAngle(0,0,XSideLength-5,YSideLength-5);
  Circle(Round(50*XScale),Round(50*YScale),Round(50*XScale));        {Constant}
  Circle(Round(75*XScale),Round(50*YScale),Round(25*XScale));      {R-Circles }
  Circle(round(65*xscale),round(50*yscale),round(35*xscale));
  Circle(round(85*xscale),round(50*yscale),round(15*xscale));
  Line(0,Round(50*YScale),Round(100*XScale),Round(50*YScale));       {Constant}
  Arc(Round(100*XScale),Round(150*YScale),107,143,Round(100*XScale)); {X-Arcs}
  Arc(Round(100*XScale),Round(-50*YScale),217,253,Abs(Round(-100*XScale)));
  Arc(Round(100*XScale),Round(60*YScale),200,249,Abs(Round(10*XScale)));
  Arc(Round(100*XScale),Round(40*YScale),110,160,Abs(Round(-10*XScale)));
```

```
  Arc(Round(100*XScale),Round(83.333*YScale),138,201,Abs(Round(100/3*
  XScale)));
  Arc(Round(100*XScale),Round(16.6667*YScale),158,222,Abs(Round(-33.33*
  XScale)));
  Arc(Round(100*XScale),Round(100*YScale),123,270,Round(50*XScale));
  Arc(Round(100*XScale),0,180,237,Abs(Round(-50*XScale)));

  Size4 := ImageSize(0,0,XSidelength,YSidelength);      {Save the smith-chart}
  GetMem(smith,size4);
  GetImage(0,0,XSidelength,YSidelength,smith^);
end;

{ +++ ------------------------------------------------------------------ +++ }

procedure chartfancy(start_str,stop_str:string;s_par_flag:s_par_type);
{*************************************************************************}
{ Procedure to provide all the fancy menus for the smithchart. Also the   }
{ status of the measurement is shown eg f_start and s-parameter.          }
{*************************************************************************}
begin
  Rectangle(Round(MaxX*0.55),5,MaxX,MaxY);
  SetTextJustify(centertext,centertext);
  SetTextStyle(1,0,2);  {fancy heading}
  Outtextxy(Round(MaxX*0.775),13,'SMITH CHART MENU');
  Line(Round(MaxX*0.55),25,MaxX,25);
  SetTextJustify(LeftText,TopText);
  Xmargin := Round(MaxX*0.55)+8;
  SetTextStyle(0,0,1);
  OutTextXY(XMargin,40,'<F1>   Zoom 1st quadrant');
  OutTextXY(XMargin,55,'<F2>   Zoom 2nd quadrant');
  OutTextXY(XMargin,70,'<F3>   Zoom 3rd quadrant');
  OutTextXY(XMargin,85,'<F4>   Zoom 4th quadrant');
  OutTextXY(XMargin,100,'<F5>   Reset Zoom');
  OutTextXY(XMargin,115,'<F6>   Toggle Cursor increment');
  OutTextXY(XMargin,145,'<ESC>  Exit to Main Menu');
  Rectangle(5,Round(MaxY*0.90),XSideLength,MaxY);
  SetTextJustify(LeftText,CenterText);
  OutTextXY(7,Round(MaxY*0.95),'Cursor');
  Line(55,Round(MaxY*0.90),55,MaxY);
  Line(55,Round(MaxY*0.95),XSideLength,Round(MaxY*0.95));
  SetViewPort(55,Round(MaxY*0.90),XSideLength,MaxY,ClipOff);
  OutTextXY(25,9,'X-Value ');
  OutTextXY(125,9,'Y-Value ');
  OutTextXY(215,9,'Cursor inc');
  Fullport;
  Line(Round(MaxX*0.55),215,MaxX,215);
{ Include the active parameters in menu eg S-Parameter,calibration status }
{ f-start and f-stop.                                                     }
  OutTextXY(XMargin,230,'Start frequency [fa]       '+start_str+' GHz');
  OutTextXY(XMargin,245,'Stop frequency [fb]        '+stop_str+' GHz');
  Case s_par_flag of
   S11: OutTextXY(XMargin,260,'S-Parameter                          S11');
   S21: OutTextXY(XMargin,260,'S-Parameter                          S21');
   S22: OutTextXY(XMargin,260,'S-Parameter                          S22');
   S12: OutTextXY(XMargin,260,'S-Parameter                          S12');
  end; {Case}
  OutTextXY(XMargin,275,'Full Correction                OFF');
  Full_Error_Correction_active := FALSE;
  OutTextXY(XMargin,290,'("T" To Toggle)');
```

```
  OutTextXY(XMargin,340,'<Print-Screen><1>            Hardcopy');
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure ZOOMquad1;
{*************************************************************************}
{ Procedure for plotting and saving the Zoomed image of the 1st quadrant of  }
{ a Smithchart.                                                              }
{*************************************************************************}
begin
  Circle(Round(50*XScale),Round(50*YScale),Round(50*XScale));
  Circle(round(65*xscale),round(50*yscale),round(35*xscale));
  Arc(Round(100*XScale),Round(-50*YScale),217,0,Abs(Round(-100*XScale)));

  sizequad1 := Imagesize(0,0,XSidelength,YSidelength);        {Save the image}
  GetMem(quad1,sizequad1);
  GetImage(0,0,XSidelength,YSidelength,quad1^);
end;

{ +++ ------------------------------------------------------------------ +++ }

procedure ZOOMquad2;
{*************************************************************************}
{ Procedure for plotting and saving the Zoomed image of the 2nd quadrant of  }
{ a Smithchart.                                                              }
{*************************************************************************}
 begin
  Circle(0,Round(50*YScale),Round(50*XScale));
  Circle(Round(25*XScale),Round(50*YScale),Round(25*XScale));
  Circle(round(15*xscale),Round(50*YScale),round(35*xscale));
  Circle(round(37.5*xscale),Round(50*YScale),round(12.5*xscale));
  Arc(Round(50*XScale),0,180,270,Round(50*XScale));
  Circle(Round(50*XScale),Round(-50*YScale),Round(100*XScale));
  Arc(Round(50*XScale),Round((50/3)*YScale),158,270,Abs(Round(100/3*XScale)));
  Arc(Round(50*XScale),Round((40)*YScale),112,280,Abs(Round(10*XScale)));

  sizequad2 := Imagesize(0,0,XSidelength,YSidelength);        {Save the image}
  GetMem(quad2,sizequad2);
  GetImage(0,0,XSidelength,YSidelength,quad2^);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure ZOOMquad3;
{*************************************************************************}
{ Procedure for plotting and saving the Zoomed image of the 3rd quadrant of  }
{ a Smithchart.                                                              }
{*************************************************************************}
 begin
  Circle(Round(50*XScale),0,Round(50*XScale));
  Circle(Round(65*XScale),0,Round(35*XScale));
  Arc(Round(100*XScale),Round(100*YScale),0,143,Round(100*XScale));

  sizequad3 := Imagesize(0,0,XSidelength,YSidelength);        {Save the image}
  GetMem(quad3,sizequad3);
  GetImage(0,0,XSidelength,YSidelength,quad3^);
 end;

{ +++ ------------------------------------------------------------------ +++ }
```

```
procedure ZOOMquad4;
{*************************************************************************)
{ Procedure for plotting and saving the Zoomed image of the 4th quadrant of  }
{ a Smithchart.                                                              }
{*************************************************************************)
 begin
  Circle(0,0,Round(50*XScale));
  Circle(Round(25*XScale),0,Round(25*XScale));
  Circle(round(15*xscale),0,round(35*xscale));
  Circle(round(37.5*xscale),0,round(12.5*xscale));
  Arc(Round(50*XScale),Round(50*YScale),90,180,Round(50*XScale));
  Circle(Round(50*XScale),Round(100*YScale),Round(100*XScale));
  Arc(Round(50*XScale),Round((100/3)*YScale),90,202,Abs(Round(100/3*XScale)));
  Arc(Round(50*XScale),Round(10*YScale),90,249,Abs(Round(10*XScale)));

  sizequad4 := Imagesize(0,0,XSidelength,YSidelength);        {Save the image}
  GetMem(quad4,sizequad4);
  GetImage(0,0,XSidelength,YSidelength,quad4^);
end;

{ +++ ------------------------------------------------------------------ +++ }

procedure DisplayData(Xa,Ya,inc,zoomflag:integer;Xscale,Yscale:real);
{*************************************************************************)
{ A procedure to compute and display the complex impedance co-ordinates of   }
{ the position of the cursor on the chart.                                   }
{*************************************************************************)
label
 30;
const
 infinity = 1000000;                                  {Prevent a divide by zero}
var
 xvalue,yvalue:integer;
 xstr,ystr:string[8];
 curstr:string[8];
 pIm,pReal:real;
 Res,React,den:real;

begin
 Case inc of
   1:  curstr := 'small';              {The cursor increments are 1 of 3 sizes}
   10: curstr := 'medium';
   20: curstr := 'large';
  end;
  SetViewPort(56,Round(MaxY*0.95)+1,XsideLength-12,MaxY-1,ClipOff);
  ClearViewport;

  Case zoomflag of
   0: begin
       xvalue := Xa;yvalue := Ya;
      end;
   1: begin
       XScale := (XSidelength-5)/50;
       YScale := (YSidelength-5)/50;
       xvalue := Xa;yvalue := Ya;
      end;
   2: begin
       XScale := (XSidelength-5)/100;
       YScale := (YSidelength-5)/100;
```

```
        xvalue := round(Xa/2+180);yvalue := round(Ya/2);
       end;
    3: begin
        XScale := (XSidelength-5)/100;
        YScale := (YSidelength-5)/100;
        xvalue := round(Xa/2);yvalue := round(Ya/2+135);
       end;
    4: begin
        XScale := (XSidelength-5)/100;
        YScale := (YSidelength-5)/100;
        xvalue := round(Xa/2+180);yvalue := round(Ya/2+135);
       end;
  end; {case}

  pReal := xvalue/(50*Xscale)-1;pIm := 1-yvalue/(50*Yscale);
  den := (sqr(pReal-1)+sqr(pIm));
  If den = 0 then
   begin
     Res := infinity;
     React := 0;
     Str(Res:6:2,xstr);Str(React:6:2,ystr);
     goto 30;
    end;
    Res := (1-sqr(pReal)-sqr(pIm))/den;
    React := (2*pIm)/den;
    Str(Res:6:2,xstr);Str(React:6:2,ystr);
30: SetTextJustify(LeftText,CenterText);
    OutTextXY(15,8,xstr);
    OutTextXY(115,8,ystr);
    OutTextXY(230,8,curstr);
    FullPort;
   end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Zoom(VAR XScale,YScale:real;xtemp,ytemp:integer);
{*************************************************************************}
{ Clear the viewport in preparation for a zoom into 1 of 4 quadfrants.    }
{*************************************************************************}
 begin
  SetViewPort(5,5,xtemp,ytemp,ClipOn);
  ClearViewport;
  RectAngle(0,0,xtemp-5,ytemp-5);
  XScale := (XSidelength)/50;
  YScale := (YSidelength)/50;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure initimage;
{*************************************************************************}
{ Procedure to plot all four quadrants of the chart and save the images.  }
{ While the charts are being plotted, the 2nd hercules screen is displayed }
{ with a mesage to that effect.                                           }
{*************************************************************************}
 begin
  SetActivePage(1);
  OutTextXY(230,170,'INITIALIZATION IN PROGRESS');
  SetActivePage(0);SetVisualPage(1);
  zoom(XScale,YScale,XSidelength+5,YSidelength+5);
  zoomquad1;
```

```
  zoom(XScale,YScale,XSidelength+5,YSidelength+5);
  zoomquad2;
  zoom(XScale,YScale,XSidelength+5,YSidelength+5);
  zoomquad3;
  zoom(XScale,YScale,XSidelength+5,YSidelength+5);
  zoomquad4;
  SetVisualPage(0);
  ClearDevice;
  Fullport;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure quit_to_main_menu;
{*************************************************************************}
{ Procedure to service the quitting from this unit. All the dynamic variables}
{ are freed from the heap thus preventing heap overflows.                   }
{*************************************************************************}
 begin
  Rectangle(XMargin+65,180,maxx-95,200);
  SetViewPort(XMargin+65,180,maxx-95,200,ClipOff);
  SetTextJustify(1,2);
  OutTextXY(round((maxx-xmargin-160)/2),7,'Are You Sure <Y/N>');
  ans := readkey;
  if (ans = 'y') or (ans = 'Y') then
   begin
     FreeMem(Cursor,size2);
     FreeMem(smith,size4);
     FreeMem(quad1,sizequad1);
     FreeMem(quad2,sizequad2);
     FreeMem(quad3,sizequad3);
     FreeMem(quad4,sizequad4);
    exit_smith := True;
   end;
  clearviewport;fullport;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure funckeys;
{*************************************************************************}
{ A procedure to service the pressing of special keys during the real-time  }
{ plot. The special keys are F1,F2,F3,F4,F5,ESC,F6 and all arrow keys.       }
{*************************************************************************}
label
 10;
begin
    Ch := ReadKey;
    if Ch = #27 then quit_to_main_menu;                          {ESC exits}
    if ((Ch = 't') OR (Ch = 'T')) AND (s_par_calibration_flag <> OFF) then
     Case s_par_calibration_flag of
      S11: If s_par_flag = S11 then Toggle_full_correction;
      S21: If s_par_flag = S21 then Toggle_full_correction;
      S12: If s_par_flag = S12 then Toggle_full_correction;
      S22: If s_par_flag = S22 then Toggle_full_correction;
      OFF: inline($90);
     end;
    If Ch <> #0 then goto 10 else                          {Special Key only}
    Ch := ReadKey;
    If (Ch<>#72) and (Ch<>#80) and (Ch<>#75) and (Ch<>#77)
```

```
and (Ch<>#59) and (Ch<>#61) and (Ch<>#62)
and (Ch<>#63) and (Ch<>#64) and (Ch<>#60) then goto 10;
Case Ch of
   #72: begin                                                    { Up arrow }
          If (Ya-inc) >= 0 then              {Restrict moves to the screen}
            begin
              PutImage(Xa,Ya,Cursor^,XORput);                   {erase image}
              Ya := Ya-inc;
              PutImage(Xa,Ya,Cursor^,XORput);                    {draw image}
            end
          Else if ((Ya-1)<0) then
            goto 10
          Else if ((Ya-inc) < 0) and (inc <> 1) then
           begin
             inc := 1;
             PutImage(Xa,Ya,Cursor^,XORput);                    {erase image}
             Ya := Ya-inc;
             PutImage(Xa,Ya,Cursor^,XORput);                    {draw image}
           end;
        end;
   #80: begin                                                  { Down Arrow }
          If (Ya+inc+5) <= YSideLength then      {Restrict cursor movement}
            begin
              PutImage(Xa,Ya,Cursor^,XORput);
              Ya := Ya+inc;
              PutImage(Xa,Ya,Cursor^,XORput);
            end
          Else if ((Ya+6)) > YSideLength then
            goto 10
          Else if ((Ya+inc+5) > YSideLength) and (inc <> 1) then
           begin
             inc := 1;
             PutImage(Xa,Ya,Cursor^,XORput);                    {erase image}
             Ya := Ya+inc;
             PutImage(Xa,Ya,Cursor^,XORput);                    {draw image}
           end;
        end;
   #75: begin                                                   {left arrow }
          If (Xa-inc) >= 0 then                               {Restrict cursor}
            begin
              PutImage(Xa,Ya,Cursor^,XORput);
              Xa := Xa-inc;
              PutImage(Xa,Ya,Cursor^,XORput);
            end
          Else if ((Xa-1)<0) then
            goto 10
          Else if ((Xa-inc) < 0) and (inc <> 1) then
           begin
             inc := 1;
             PutImage(Xa,Ya,Cursor^,XORput);                    {erase image}
             Xa := Xa-inc;
             PutImage(Xa,Ya,Cursor^,XORput);                    {draw image}
           end;
        end;
   #77: begin                                                 { right arrow }
          If (Xa+inc) <= XSideLength-5 then                   {Restrict cursor}
            begin
              PutImage(Xa,Ya,Cursor^,XORput);
              Xa := Xa+inc;
              PutImage(Xa,Ya,Cursor^,XORput);
```

```
            end
          Else if ((Xa+1)>XSidelength-5) then
            goto 10
          Else if ((Xa+inc) > XSideLength-5) and (inc <> 1) then
           begin
             inc := 1;
             PutImage(Xa,Ya,Cursor^,XORput);                {erase image}
             Xa := Xa+inc;
             PutImage(Xa,Ya,Cursor^,XORput);                 {draw image}
           end;
         end;
    #59: begin                                          {F1 ZOOMS to U.L.H.}
          zoomflag := 1;
          Zoom(XScale,YScale,XSidelength,YSidelength);
          PutImage(0,0,quad1^,0);
          PutImage(Xa-5, Ya-5, Cursor^, XORput);
         end;
    #60: begin                                          {F2 Zooms to U.R.H.}
          zoomflag := 2;
          Zoom(XScale,YScale,XSidelength,YSidelength);
          PutImage(0,0,quad2^,0);
          PutImage(Xa-5, Ya-5, Cursor^, XORput);
         end;
    #61: begin                                    {F3 Zooms to lower L.L.H.}
          zoomflag := 3;
          Zoom(XScale,YScale,XSidelength,YSidelength);
          PutImage(0,0,quad3^,0);
          PutImage(Xa-5, Ya-5, Cursor^, XORput);
         end;
    #62: begin                                    {F4 Zooms to lower L.R.H.}
          zoomflag := 4;
          Zoom(XScale,YScale,XSidelength,YSidelength);
          PutImage(0,0,quad4^,0);
          PutImage(Xa-5, Ya-5, Cursor^, XORput);
         end;
    #63: begin                                          {F5 Zooms to normal}
          zoomflag := 0;
          Zoom(XScale,YScale,XSidelength,YSidelength);
          PutImage(0,0,smith^,0);
          PutImage(Xa-5,Ya-5,Cursor^,XORput);
          XScale := (XSidelength-5)/100;
          YScale := (YSidelength-5)/100;
         end;
    #64: begin                                {F6 toggles cursor jump-size }
          if (inc = 1) then                           {3 sizes are possible}
           inc := 10
          else if
           inc = 10 then
            inc := 20
          else
           inc := 1
         end;
  end; { case }
10: end;

{ +++ ------------------------------------------------------------------ +++ }
```

```
procedure readanddisplaydata;
{*************************************************************************}
{ Real time plot procedure.                                               }
{*************************************************************************}
 var
  tempX,tempY:fftpoints;
  counter:integer;
 begin
  counter:=0;
  DisplayData(Xa,Ya,inc,zoomflag,Xscale,Yscale);
  if zoomflag = 1 then
   begin
    XScale := (XSideLength-5)/50;
    YScale := (YSideLength-5)/50;
   end
  else
   begin
    XScale := (XSideLength-5)/100;
    YScale := (YSideLength-5)/100;
   end;
  SetViewPort(5,5,XSideLength,YSideLength,ClipOn);
  dataentry;
 repeat
  counter := counter + 1;
  if counter >= 10 then
   begin
    Case zoomflag of
     0: PutImage(0,0,smith^,2);
     1: PutImage(0,0,quad1^,2);
     2: PutImage(0,0,quad2^,2);
     3: PutImage(0,0,quad3^,2);
     4: PutImage(0,0,quad4^,2);
    end;
    counter := 0;
   end;
  tempX := xval;
  tempY := yval;
  dataentry;
  case zoomflag of                    {Real Time plot of data on entire chart)
   0: for i := 0 to (size-1) do                {or on a quadrant of the chart)
        begin
         SetColor(0);                                              {FULL chart)
         MoveTo(tempX[i],tempY[i]);
         LineTo(tempX[i+1],tempY[i+1]);
         SetColor(15);
         MoveTo(xval[i],yval[i]);
         LineTo(xval[i+1],yval[i+1]);
        end;
   1: for i := 0 to (size-1) do                                  {1st quadrant)
        begin
         SetColor(0);
         MoveTo(tempX[i],tempY[i]);
         LineTo(tempX[i+1],tempY[i+1]);
         SetColor(15);
         MoveTo(xval[i],yval[i]);
         LineTo(xval[i+1],yval[i+1]);
        end;
   2: for i := 0 to (size-1) do                                  {2nd quadrant)
        begin
         SetColor(0);
```

```
          MoveTo((tempX[i]-180)*2,tempY[i]*2);
          LineTo((tempX[i+1]-180)*2,tempY[i+1]*2);
          SetColor(15);
          MoveTo((xval[i]-180)*2,yval[i]*2);
          LineTo((xval[i+1]-180)*2,yval[i+1]*2);
         end;
   3: for i := 0 to (size-1) do                                    {3rd quadrant}
       begin
        SetColor(0);
        MoveTo(tempX[i]*2,(tempY[i]-135)*2);
        LineTo(tempX[i+1]*2,(tempY[i+1]-135)*2);
        SetColor(15);
        MoveTo(xval[i]*2,(yval[i]-135)*2);
        LineTo(xval[i+1]*2,(yval[i+1]-135)*2);
       end;
   4: for i := 0 to (size-1) do                                    {4th quadrant}
       begin
        SetColor(0);
        MoveTo((tempX[i]-180)*2,(tempY[i]-135)*2);
        LineTo((tempX[i+1]-180)*2,(tempY[i+1]-135)*2);
        SetColor(15);
        MoveTo((xval[i]-180)*2,(yval[i]-135)*2);
        LineTo((xval[i+1]-180)*2,(yval[i+1]-135)*2);
       end;
  end; {case}
 until keypressed;
 fullport;
end;

{ +++ ------------------------------------------------------------------ +++ }

procedure main(start_str,stop_str:string;s_par_flag:s_par_type);
{*************************************************************************}
{ The main procedure in the unit. It calls all the other procedures necessary}
{ to plot and display a real time plot of complex impedance.              }
{*************************************************************************}
 begin
    freq_response_only_cal;                          {Perform normalization}
    initialize(MaxX,MaxY,XSideLength,YSideLength);          {Initialize vars}
    initimage;                                      {init 4 images of chart}
    drawcursor(cursor,size2);                            {init cursor image}
    chartfancy(start_str,stop_str,s_par_flag);                {Fancy menus}
    drawsmithchart(XSideLength,YSidelength,XScale,YScale);   {Draw full chart}
    FullPort;
    Xa := 20;Ya := 255;                                       {Move the cursor}
    PutImage(Xa, Ya, Cursor^, XORput);                {to its initial position}
    repeat
      readanddisplaydata;                                  {REAL-TIME DISPLAY}
      funckeys;                                           {service a keypress}
    until exit_smith = true;
    CloseGraph;                                          {return to text mode}
    cursor_off(0,0);                                    {switch of the cursor}
end;

{ +++ ------------------------------------------------------------------ +++ }

end.                                                    {END OF UNIT SMITHCHA}
```

```
unit menu_utility;
{*****************************************************************************}
{ A unit to sketch all the pull-down menus availiable in the program and to  }
{ service all selections made in these menus.                                 }
{*****************************************************************************}
{$O+,F+}                          {Compiler option for potentially overlaid units}
{$M $4000,0,$4000}                {Compiler option nescessary for HPIB operation}
{*****************************************************************************}
interface
{*****************************************************************************}
uses crt,dos;                                          {units used in this unit}

type
 Astring = string[255];
 string_20 = string[6];
 s_par_type = (S11,S21,S22,S12,OFF);       {The possible s-parameter selection}
 menu_message_4 = array[1..4]of string;   {messages in menus with four options}
 menu_message_3 = array[1..3]of string;  {messages in menus with three options}

var
 start_str,stop_str:string;                    {selected start and stop strings}
 delta_f_str:string;                                  {selected delta_f string}
 delta_f:real;                                         {frequency per division}
 f_start,f_stop:real;                       {selected start and stop frequencies}
 rf:real;
 rf_str:string;
 main_menu_flag:integer;                      {which option chosen from mainmenu}
 flag_1:integer;                             {whether START or STOP freq selected}
 flag_2:integer;                             {whether Smith or Rectangular chosen}
 ch_real:char;                              {char read from the readreal procedure}
 strng:string_20;                           {the string read in from proc readreal}
 j:integer;                                        {the number of chars in strng}
 k:integer;                                  {used in proc start_freq as a dummy}
 Ans:char;                                    {response from keyboard in submenu}
 f_min,f_max:real;                           {Min and Max sweep range of sweeper}
 rf_min,rf_max:real;
 Exit_to_main_menu:boolean;                    {flag inside freq_set menu to exit}
 flag_5:integer;                            {whether Phase or Mag display chosen}
 flag_4:integer;                                  {Which S-Parameter was chosen}
 GPIO:text;
 s_par_flag:s_par_type;
 s_par_calibration_flag:s_par_type;
 atten_flag:integer;
 temp_str:string;
 flag_8:integer;
 flag_9:integer;
 g,g3,g4,g5,g6:integer;
 msg_mag_phase:menu_message_4;
 msg_s_par_cal:menu_message_3;
 msg_smith_rect:menu_message_3;
 msg_touchstone:menu_message_3;
 block_size:integer;
 freq_response_cal:boolean;

procedure NormVideo_Msg(S:Astring;R1,C1:integer);
procedure InvVideo_Msg(S:Astring;R1,C1:integer);
procedure outline;                                  {Draws outline of mainmenu}
procedure heading;                                 {writes headings in outline}
procedure freq_block;                            {Draws block containing fa and fb}
procedure Beep;                                    {Generate an error beep - 900Hz}
```

```
procedure getkey (VAR key : CHAR);              {Returns the value of keypressed}
function  readreal:real;                       {Read in the start and stop freqs}
procedure reset_menu(block_size:integer);     {Restore pull-down menu's status}
procedure smaller_block;                {Draw start/stop_submenu block outline}
procedure start_freq;                     {Input the Start Freq from keyboard}
procedure Stop_freq;                      {Input the Stop Freq from keyboard}
procedure process_response_1;                          {Set freq-range selected}
procedure freq_set;                        {Set-frequency range is serviced}
procedure box;                           {Draw box around display format menu}
procedure s_parameter;
Procedure SelectHPIB(Isc:integer);
procedure Read_via_HPIB(parameter:string;VAR result:real);
procedure klein_box;
procedure menu_frame_7(F:integer);
procedure menu_frame_4(F:integer;msg:menu_message_4;VAR flag:integer);
procedure menu_frame_3(F:integer;msg:menu_message_3;VAR flag:integer);
procedure read_keyboard(VAR ans:char;VAR Exit:Boolean);
procedure block(f:integer);

{*****************************************************************************}
implementation
{*****************************************************************************}

const
 Sweep_Address = 719;                          {Address of the sweep oscillator}

var
 f,f2:integer;
 msg_set_freq:menu_message_4;


{ +++ ------------------------------------------------------------------ +++ }

procedure read_keyboard(VAR ans:char;VAR Exit:Boolean);
{*****************************************************************************}
{ Procedure to read a key from the keyboard.                                  }
{*****************************************************************************}
 begin
   Ans := ReadKey;
   if (Ans = #0) then Ans := ReadKey;
   if (Ans = #27) then Exit := True;
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure NormVideo_Msg(S:Astring;R1,C1:integer);
{*****************************************************************************}
{ Procedure to write a message to screen in text-mode in normal attributes.   }
{*****************************************************************************}
  begin
    gotoXY(R1,C1);write(S);
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure InvVideo_Msg(S:Astring;R1,C1:integer);
{*****************************************************************************}
{ Procedure to write a message to screen in text-mode in inverse attributes.  }
{*****************************************************************************}
  begin
    textBackground(15);textcolor(0);       {Set the background colour = white}
```

```
    GotoXY(R1,C1);write(S);                    {Set the text colour = black}
    textBackground(0);textcolor(7);                  {Reset the attributes}
  end;

{ +++ ------------------------------------------------------------------- +++ }

procedure outline;
{*****************************************************************************}
{ Draw the main menu outline in text mode.                                    }
{*****************************************************************************}
 var
  i:integer;
 begin
  Write('
                                ┌──────────────────────┐
              ');
  Write('┌─────────────────────┤                      ├──────────────
  ──────────┐');
  Write('│                     │                      │
              │');
  Write('│                     └──────────────────────┘
              │');
  Write('│
              │');
  Write('│
              │');
  Write('│                  ┌──────────────────────────┐
              │');
  for i := 1 to 13 do Write('│                  │                          │
                          │');
  Write('│                  └──────────────────────────┘
              │');
  Write('│
              │');
  Write('│
              │');
  Write('│
              │');
  Write('└──────────────────────────────────────────────────────────────────
  ──────────┘');
 end;

{ +++ ------------------------------------------------------------------- +++ }

procedure freq_block;
{*****************************************************************************}
{ Draw a block in the main menu containing the start and stop frequencies     }
{ and the rf level that was selected.                                         }
{*****************************************************************************}
 var
  i:integer;
 begin
  Window(60,3,78,9);
  GotoXY(1,1);
  writeln('┌──────────────────┐');
  for i := 1 to 4 do writeln('│                  │');
  writeln('└──────────────────┘');
  NormVideo_Msg('fa =         GHz',2,2);                  {Start frequency}
  NormVideo_Msg('fb =         GHz',2,3);                   {Stop frequency}
  NormVideo_Msg('rf =         dBm',2,4);                         {RF level}
  NormVideo_Msg('S-Parameter    ',2,5);
  Str(f_start:6:4,start_str);
  Str(f_stop:6:4,stop_str);
```

```
  Str(rf:6:4,rf_str);

  NormVideo_Msg(start_str,7,2);
  NormVideo_Msg(stop_str,7,3);
  NormVideo_Msg(rf_str,7,4);
  Case s_par_flag of                    {Also write out the presently selected}
   S11: temp_str := 'S11';                                    {s-parameter}
   S12: temp_str := 'S12';
   S21: temp_str := 'S21';
   S22: temp_str := 'S22';
   OFF: inline($90);
  end;
  NormVideo_Msg(temp_str,15,5);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure heading;
{*****************************************************************************}
{ Put all the headings inside the main menu frame.                            }
{*****************************************************************************}
 begin
   HighVideo;
   NormVideo_Msg('HP8410C NETWORK ANALYZER',29,2);
   NormVideo_Msg(' UPGRADE ',36,3);
   LowVideo;
   GotoXY(2,23);
   Write('————————————————————————————   ');
   GotoXY(46,23);
   Write('      ————————————————————————————');
   HighVideo;
   NormVideo_Msg('Press ESC to quit',34,23);
   LowVideo;
   freq_block;
  Window(1,1,80,25);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure menu_frame_7(F:integer);
{*****************************************************************************}
{ Write all the messages in the menu frame where there are seven different    }
{ options. In this case, it is the main menu.                                 }
{*****************************************************************************}
 var
  i:integer;
  message:array[1..7] of string;
 begin
  message[1]:='Frequency Range/Rf Level';
  message[2]:='S-Parameter/Attenuation';
  message[3]:='Frequency Domain Display';message[4]:='Full Error Correction';
  message[5]:='Text Files';message[6]:='Amplifier Calibration';
  message[7]:='Exit To Dos';
  for i := 1 to 7 do
    if i <> F then NormVideo_Msg(message[i],19,2*i+5)
    else
     begin
      main_menu_flag := i;
      InvVideo_Msg(message[i],19,2*i+5)
     end;
 end;
```

```
{ +++ ------------------------------------------------------------------ +++ }

 procedure block(f:integer);
 {*************************************************************************}
 { A submenu structure.                                                    }
 {*************************************************************************}
  var
   i:integer;
  begin
    GotoXY(1,1);
    Writeln('┌──────────────────────────┐');
    for i := 1 to f do Writeln('│                          │');
    Writeln('└──────────────────────────┘');
   end;

{ +++ ------------------------------------------------------------------ +++ }

procedure menu_frame_4(F:integer;msg:menu_message_4;VAR flag:integer);
{*************************************************************************}
{ Write all the messages in the menu frame where there are four different  }
{ options. This structure is used by several different submenu structures. }
{*************************************************************************}
 var
  i:integer;
 begin
  for i := 1 to 4 do
    if i <> F then NormVideo_Msg(msg[i],4,2*i+1)
    else
     begin
      flag := i;
      InvVideo_Msg(msg[i],4,2*i+1)
     end;
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure menu_frame_3(F:integer;msg:menu_message_3;VAR flag:integer);
{*************************************************************************}
{ Write all the messages in the menu frame where there are three different }
{ options. This structure is used by several different submenu structures. }
{*************************************************************************}
 var
  i:integer;
 begin
  for i := 1 to 3 do
    if i <> F then NormVideo_Msg(msg[i],4,2*i+1)
    else
     begin
      flag := i;
      InvVideo_Msg(msg[i],4,2*i+1)
     end;
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Beep;
{*************************************************************************}
{ Procedure to generate an 900 Hz tone.                                   }
{*************************************************************************}
 begin
```

```
  sound(900);                                              {900Hz = frequency}
  delay(300);                                               {sound for 300ms}
  nosound;                                                          {silence}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure getkey (VAR key : CHAR);
{*****************************************************************************}
{ Procedure to read the keyboard.                                             }
{*****************************************************************************}
Begin
  if KeyPressed then key := Readkey
  else key := chr(0);
end;

{ +++ ------------------------------------------------------------------ +++ }

function readreal: real;
{*****************************************************************************}
{ Foolproof routine!!! for entering reals with up to max significant digits. }
{ Only 0..9,.,RETURN and BACKSPACE are allowed. Incorrect chars, cause an     }
{ error beep to sound.                                                        }
{*****************************************************************************}
var
   max,k:integer;
   i:real;
   cr,OK,point:boolean;
   dig:integer;

begin
   strng:='00000000000';                                {Initialize the string}
   cr:=false;
   j:=1;
   max:=6;                          {max is the max number of significant digits}
   point:=false;
   repeat
     ok := false;
     repeat
       getkey(ch_real);
       if ORD(ch_real) IN [8,13,46,48..57] then OK:=true
       else if ORD(ch_real)<>0 then Beep;          {If a wrong key was entered}
     until OK ;

     Case Ord(ch_real) of
         46,48..57 : begin                                              {0-9 and .}
                       if (j<max) AND ((NOT(point) AND (ch_real='.'))
                       OR (ord(ch_real) IN [48..57])) then
                        begin
                          Write (ch_real);
                          if ch_real='.' then
                          begin
                             point:=TRUE;
                             max:=max+1;
                          end;
                          strng[j]:=ch_real;
                          j:=j+1;
                        end else Beep;
                     end;
                13 : begin
```

```
                                {If no point is entered then insert a point}
                        if ((j=2) or (j=3)) and (not(point)) then
                         begin
                          strng[J]:='.';
                         end;
{*************************************************************************}
{ If the first digit is a point then move all the digits one place right and }
{ insert leading 0.                                                         }
{*************************************************************************}
                        if strng[1] = '.' then
                         begin
                          dig := 6;
                          repeat
                           strng[dig] := strng[dig-1];
                           dig := dig-1;
                          until dig = 2;
                          strng[dig] := '.';
                          strng[dig-1] := '0';
                         end;

                          {if return is pressed with no new entry then default}
                       if (j = 1)AND(flag_1 = 1) then strng := start_str;
                       if (j = 1)AND(flag_1 = 2) then strng := stop_str;
                       if (j = 1)AND(flag_1 = 3) then strng := rf_str;
                       Val(strng,i,k);
                       cr:=TRUE;
                       readreal := i;
                     end;
                  8 : begin
                        if j>1 then
                        begin
                           if strng[j-1] = '.' then point:=False;
                           Write (ch_real);                    {one backspace}
                           Write ('');              {wipe out old character}
                           Write (ch_real);                    {one backspace}
                           strng[j-1]:='0';
                        end;
                        j:=j-1;
                        if j>0 then
                        begin
                           if strng[j]='.' then point:=False;
                           strng[j]:='0';
                        end else
                        begin
                           j:=1;
                           Beep;
                        end;
                     end;
      end; (* case *)
   until cr;
end; (* readreal *)

{ +++ ------------------------------------------------------------------ +++ }

procedure reset_menu(block_size:integer);
{*************************************************************************}
{ Procedure to redraw the menu structures the way they were before a graphics}
{ call was made.                                                            }
{*************************************************************************}
 begin
  clrscr;                                  {Replace menus the way they were}
```

```
  Window(1,1,80,25);                                    {Default full-screen Window}
  outline;                                                 {Sketch mainmenu outline}
  heading;                                                {Main heading in mainmenu}
  menu_frame_7(g);
  Window(30,6,78,25);
  block(block_size);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure smaller_block;
{*************************************************************************}
{ Menu structure used to house the set freq menus.                        }
{*************************************************************************}
 var
  i:integer;
 begin
  Window(40,10,80,25);
  GotoXY(1,1);
  Writeln('┌──────────────────────────────────────┐');
  for i := 1 to 6 do Writeln('│                                      │');
  Writeln('└──────────────────────────────────────┘');
  NormVideo_Msg('──────                          ──────',2,7);
  HighVideo;
  NormVideo_Msg('RETURN For Default',10,7);
  LowVideo;
 end;

{ +++ ------------------------------------------------------------------ +++ }

Procedure SelectHPIB(Isc:integer);
{*************************************************************************}
{ Procedure to select the HPIB device. Config.sys sets up a default device }
{ which is the sweeper. If you wish to change this device, use this call.  }
{*************************************************************************}
 var
  Addr : String[8];
 begin
  Str(Isc,Addr);
  Addr := 'HPIB='+Addr;                                    {Address of device}
  Exec('HPIBMODE.COM',Addr);                             {execute a DOS command}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Program_HPIB(flag:integer;freq:real);
{*************************************************************************}
{ Program the HPIB to select the start or stop frequencies.               }
{*************************************************************************}
 var
  i,Dev : Integer;
  Comm : String[12];
  St:String[6];
 begin
  SelectHPIB(Sweep_Address);
  Case flag of
   1: begin
       Str(freq:6:4,St);                                {Start frequency selected}
       Comm := 'fa'+St+'Gz';
       Write(GPIO,Comm);
      end;
```

```
   2: begin
       Str(freq:6:4,St);
       Comm := 'fb'+St+'Gz';                          {Stop frequency selected}
       Write(GPIO,Comm);
      end;
   end;{Case}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Read_via_HPIB(parameter:string;VAR result:real);
{*************************************************************************}
{ A procedure to READ in via the HPIB bus various parameters. Parameter   }
{ refers to the HPIB-type parameter that is to be interrogated eg OPFA    }
{ refers to f-start. Result refers to the returned value and is of type real.}
{*************************************************************************}
 var
  F :String[12];
  OutF,InF:Text;
  j:integer;
  begin
   Assign(OutF,'HPIBDEV');Rewrite(OutF);
   Assign(InF,'HPIBDEV');Reset(InF);
   Write(OutF,parameter);
   read(InF,F);
   val(F,result,j);
   result := result/1e9;
   Close(OutF);Close(InF);
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure start_freq;
{*************************************************************************}
{ Input the Start Freq from keyboard and program it via HPIB.             }
{*************************************************************************}
 begin
  flag_1 := 1;                          {flag to determine start(not STOP)freq}
  smaller_block;
  Str(f_start:6:4,start_str);
  NormVideo_Msg('Present Start Freq =        GHz',4,3);
  NormVideo_Msg(start_str,25,3);
  repeat                                {Until fa and fb are within range}
    NormVideo_Msg('New Start Freq = ▒▒▒▒▒▒▒ GHz',4,5);
    GotoXY(21,5);
    f_start := readreal;
    val(strng,f_start,k);
  until (f_start >= f_min)AND(f_start <=f_max);
  if (f_start >= f_stop) then f_stop := f_start;
{*************************************************************************}
{ Check to see if the selected freq is in the range of the sweeper and if  }
{ the start freq is smaller than the stop freq.                           }
{*************************************************************************}
  Program_HPIB(flag_1,f_start);
  s_par_calibration_flag := OFF;
  block_size:=9;
  reset_menu(block_size);
 end;

{ +++ ------------------------------------------------------------------ +++ }
```

```
procedure Stop_freq;
{*************************************************************************}
{ Input the Stop Freq from keyboard and program it via HPIB.              }
{*************************************************************************}
 begin
  flag_1 := 2;                          {flag to determine stop(not START)freq}
  smaller_block;
  Str(f_stop:6:4,stop_str);
  NormVideo_Msg('Present Stop Freq =         GHz',4,3);
  NormVideo_Msg(stop_str,25,3);
  repeat                                {Until fa and fb are within range}
    NormVideo_Msg('New Stop Freq = ▒▒▒▒▒▒▒ GHz',4,5);
    GotoXY(20,5);
    f_stop := readreal;
    val(strng,f_stop,k);
  until (f_stop >= f_min)AND(f_stop <= f_max);
  if (f_stop <= f_start) then f_start := f_stop;
{*************************************************************************}
{ Check to see if the selected freq is in the range of the sweeper and if }
{ the stop freq is larger than the start freq.                            }
{*************************************************************************}
  Program_HPIB(flag_1,f_stop);
  s_par_calibration_flag := OFF;
  block_size:=9;
  reset_menu(block_size);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure Set_RF_level;
{*************************************************************************}
{ Input the Rf Level from keyboard and program it via HPIB.               }
{*************************************************************************}
 begin
  flag_1 := 3;
  smaller_block;
  Str(rf:6:4,rf_str);
  NormVideo_Msg('Present Rf Level  =         dBm',4,3);
  NormVideo_Msg(rf_str,25,3);
  repeat
    NormVideo_Msg('New Rf Level  = ▒▒▒▒▒▒▒ dBm',4,5);
    GotoXY(20,5);
    rf := readreal;
    val(strng,rf,k);
  until (rf >= rf_min)AND(rf <= rf_max);
  Str(rf:6:4,rf_str);
  Write(GPIO,'pl'+rf_str+'dm');
  s_par_calibration_flag := OFF;
  block_size:=9;
  reset_menu(block_size);
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure process_response_1;
{*************************************************************************}
{ Part1 of the menu structure needed to set freq-range and rf-level.      }
{*************************************************************************}
 begin
  Case flag_1 of                        {flag_1 determines which option chosen}
```

```
   1: begin
       Start_freq;
       freq_response_cal:=TRUE;    {The start freq has been changed ie we need}
       f2:=2;                                {a new transmission thru calibration}
       menu_frame_4(f2,msg_set_freq,flag_1);
      end;
   2: begin
       Stop_freq;
       freq_response_cal:=TRUE;     {The stop freq has been changed ie we need}
       f2:=3;                                {a new transmission thru calibration}
       menu_frame_4(f2,msg_set_freq,flag_1);
      end;
   3: begin
       Set_RF_level;
       freq_response_cal:=TRUE;      {The rf level has been changed ie we need}
       f2:=4;                                {a new transmission thru calibration}
       menu_frame_4(f2,msg_set_freq,flag_1);
      end;
   4: begin                                                  {Return to Main Menu}
       Exit_to_main_menu := true;
      end;
   end;{Case}
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure freq_set;
{*************************************************************************}
{ Part2 of the menu structure needed to set freq-range and rf-level.      }
{*************************************************************************}
 begin
  block_size:=9;
  block(block_size);
  f2:=1;
  menu_frame_4(f2,msg_set_freq,flag_1);
  repeat
    read_keyboard(Ans,Exit_to_main_menu);
    if (Ans = #72) or (Ans = #80) then
     begin
       Case Ans of
        #72: begin
              if flag_1 = 1 then f2 := 4                          {up arrow}
              else f2 := flag_1-1;
              menu_frame_4(f2,msg_set_freq,flag_1);
             end;
        #80: begin
              if flag_1 = 4 then f2 := 1                        {down arrow}
              else f2 := flag_1+1;
             menu_frame_4(f2,msg_set_freq,flag_1);
            end;
       end;{Case Ans}
     end;
    if Ans = #13 then process_response_1;
  until Exit_to_main_menu = True;
  flag_1 := 1;                                  {Reset this flag to default}
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure box;
{*************************************************************************}
```

```
{ A Sub-menu housing structure.                                                }
{******************************************************************************}
 var
  i:integer;
 begin
  Window(40,9,80,25);
  GotoXY(1,1);
  Writeln('┌──────────────────────────────┐');
  for i := 1 to 9 do Writeln('│                              │');
  Writeln('└──────────────────────────────┘');
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure menu_frame_13(F:integer);
{******************************************************************************}
{ Write all the messages in the menu frame where there are thirteeen           }
{ different options.                                                           }
{******************************************************************************}
 var
  i:integer;
  s:string;
  message:array[1..13] of string;
 begin
  message[1]:='S11 <F-Reflection>';message[2]:='S21 <F-Transmission>';
  message[3]:='S22 <R-Reflection>';message[4]:='S12 <R-Transmission>';
  for i := 5 to 12 do
   begin
    str((i-5)*10,s);
    message[i]:=s+'dB Attenuation'
   end;
  message[13]:='Return to Main Menu';
  for i := 1 to 13 do
    if i = F then
     begin
      flag_4 := i;
      if i > 4 then InvVideo_Msg(message[i],4,i+2)
      else InvVideo_Msg(message[i],4,i+1);
     end
    else  if i > 4 then NormVideo_Msg(message[i],4,i+2)
          else NormVideo_Msg(message[i],4,i+1)
 end;

{ +++ ------------------------------------------------------------------ +++ }

procedure process_response_4;
{******************************************************************************}
{ Part1 of the menu structure needed to set s-paremeters and incident          }
{ attenuation.                                                                 }
{******************************************************************************}
 begin
  Case flag_4 of
   1: begin                                                    {S11 selected}
       port[$261] := $04;
       if s_par_flag <> S11 then freq_response_cal:=TRUE;
       s_par_flag := S11;                    {Reset Transmission thru cal flag}
       f:=5;
       menu_frame_13(f);
      end;
   2: begin
       port[$261] := $06;                                      {S21 selected}
```

```
     if (s_par_flag <> S21)AND(s_par_flag <> S12) then
     freq_response_cal:=TRUE;              {Reset Transmission thru cal flag}
     s_par_flag := S21;
     f:=5;
     menu_frame_13(f);
    end;
 3: begin                                                    {S22 selected}
     port[$261] := $07;
     if s_par_flag <> S22 then freq_response_cal:=TRUE;
     s_par_flag := S22;                    {Reset Transmission thru cal flag}
     f:=5;
     menu_frame_13(f);
    end;
 4: begin                                                    {S12 selected}
     port[$261] := $05;
     if (s_par_flag <> S12)AND(s_par_flag <> S21) then
     freq_response_cal:=TRUE;              {Reset Transmission thru cal flag}
     s_par_flag := S12;
     f:=5;
     menu_frame_13(f);
    end;
 5: begin
     port[$260] := $00;                          {0dB attenuation selected}
     if atten_flag <> 0 then freq_response_cal:=TRUE;
     atten_flag := 0;                      {Reset Transmission thru cal flag}
     s_par_calibration_flag := OFF;
     f:=13;
     menu_frame_13(f);
    end;
 6: begin
     port[$260] := $04;                         {10dB attenuation selected}
     if atten_flag <> 10 then freq_response_cal:=TRUE;
     atten_flag := 10;                     {Reset Transmission thru cal flag}
     s_par_calibration_flag := OFF;
     f:=13;
     menu_frame_13(f);
    end;
 7: begin
     port[$260] := $02;                         {20dB attenuation selected}
     if atten_flag <> 20 then freq_response_cal:=TRUE;
     atten_flag := 20;
     s_par_calibration_flag := OFF;        {Reset Transmission thru cal flag}
     f:=13;
     menu_frame_13(f);
    end;
 8: begin
     port[$260] := $06;                         {30dB attenuation selected}
     if atten_flag <> 30 then freq_response_cal:=TRUE;
     atten_flag := 30;
     s_par_calibration_flag := OFF;        {Reset Transmission thru cal flag}
     f:=13;
     menu_frame_13(f);
    end;
 9: begin
     port[$260] := $01;                         {40dB attenuation selected}
     if atten_flag <> 40 then freq_response_cal:=TRUE;
     atten_flag := 40;
     s_par_calibration_flag := OFF;        {Reset Transmission thru cal flag}
     f:=13;
     menu_frame_13(f);
```

```
      end;
   10: begin
       port[$260] := $05;                          {50dB attenuation selected}
       if atten_flag <> 50 then freq_response_cal:=TRUE;
       atten_flag := 50;
       s_par_calibration_flag := OFF;       {Reset Transmission thru cal flag}
       f:=13;
       menu_frame_13(f);
      end;
   11: begin
       port[$260] := $03;                          {60dB attenuation selected}
       if atten_flag <> 60 then freq_response_cal:=TRUE;
       atten_flag := 60;
       s_par_calibration_flag := OFF;       {Reset Transmission thru cal flag}
       f:=13;
       menu_frame_13(f);
      end;
   12: begin
       port[$260] := $07;                          {70dB attenuation selected}
       if atten_flag <> 70 then freq_response_cal:=TRUE;
       atten_flag := 70;
       s_par_calibration_flag := OFF;       {Reset Transmission thru cal flag}
       f:=13;
       menu_frame_13(f);
      end;
   13: begin                                          {Return to Main Menu}
       Exit_to_main_menu := true;
      end;
    end;{Case}
   end;

{ +++ ------------------------------------------------------------------ +++ }

procedure s_par_block;
{*************************************************************************}
{ A Sub-menu housing structure used for s-parameter and incident attenuation.}
{*************************************************************************}
 var
  i:integer;
  begin
    GotoXY(1,1);
    Writeln('┌──────────────────────────┐');
    for i := 1 to 14 do Writeln('│                          │');
    Writeln('└──────────────────────────┘');
   end;

{ +++ ------------------------------------------------------------------ +++ }

procedure s_parameter;
{*************************************************************************}
{ Part2 of the menu structure needed to set s-paremeters and incident     }
{ attenuation.                                                            }
{*************************************************************************}
 begin
  s_par_block;
  f:=1;
  menu_frame_13(f);
  repeat
    read_keyboard(Ans,Exit_to_main_menu);
    if (Ans = #72) or (Ans = #80) then
     begin
```

```
        Case Ans of
         #72: begin
               if flag_4 = 1 then f := 13                          {up arrow}
               else f := flag_4-1;
               menu_frame_13(f);
              end;
         #80: begin
               if flag_4 = 13 then f := 1                          {down arrow}
               else f := flag_4+1;
               menu_frame_13(f);
              end;
        end;{Case Ans}
      end;
     if Ans = #13 then process_response_4;
   until Exit_to_main_menu = True;
   flag_4 := 1;                                  {Reset this flag to default}
  end;

{ +++ ------------------------------------------------------------------ +++ }

procedure klein_box;
{*************************************************************************}
{ A Sub-menu housing structure.                                           }
{*************************************************************************}
 var
  i:integer;
 begin
  Window(40,9,80,25);
  GotoXY(1,1);
  Writeln('┌──────┐');
  for i := 1 to 7 do Writeln('│      │');
  Writeln('└──────┘');
 end;

{*************************************************************************}
{ Initialization section of the unit. All the messages used in the menus are }
{ initialized here.                                                       }
{*************************************************************************}

begin
 msg_set_freq[1]:='Set Start Frequency';
 msg_set_freq[2]:='Set Stop Frequency';
 msg_set_freq[3]:='Set Rf Level';
 msg_set_freq[4]:='Return to Main Menu';
 msg_mag_phase[1]:='Magnitude Display only';
 msg_mag_phase[2]:='Phase Display only';
 msg_mag_phase[3]:='Magnitude and Phase';
 msg_mag_phase[4]:='Return to Main Menu';
 msg_s_par_cal[1]:='S11';
 msg_s_par_cal[2]:='S22';
 msg_s_par_cal[3]:='Quit';
 msg_smith_rect[1]:='Rectangular Display';
 msg_smith_rect[2]:='Smith Chart Display';
 msg_smith_rect[3]:='Return to Main Menu';
 msg_touchstone[1]:='Nothing/Niks';
 msg_touchstone[2]:='Nothing/Niks';
 msg_touchstone[3]:='Return to Main Menu';
end.                                          {END OF UNIT MENU_UTILTY}

{=========================================================================}
```