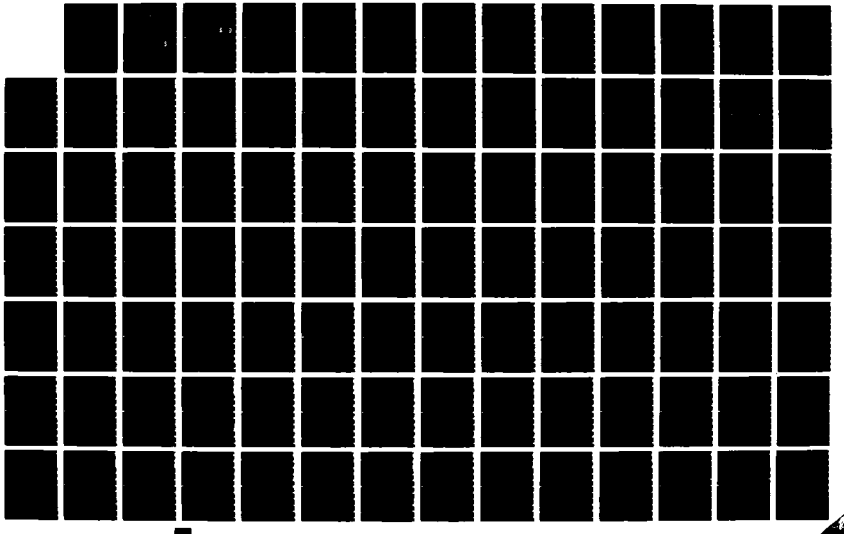
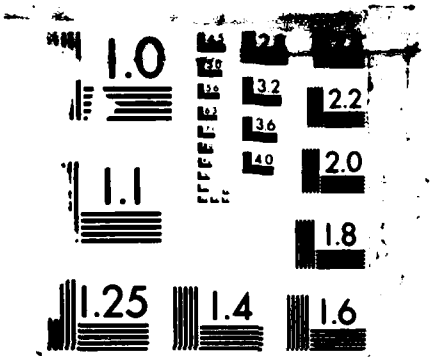
AD-A178 072 DEVELOPMENT OF A MICROCOMPUTER-BASED WORKSTATION FOR ANALYSIS OF DIGITAL.. (U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W C JACKSON 1/2

UNCLASSIFIED DEC 86 AFIT/GE/ENG/86D-38 F/G 17/2 NL





MIC

DEVELOPMENT OF A MICROCOMPUTER-BASED WORKSTATION FOR ANALYSIS OF DIGITAL SATELLITE LINKS

THESIS

William C. Jackson
Captain, USAF

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GE/ENG/86D-38

DEVELOPMENT OF A MICROCOMPUTER-BASED WORKSTATION FOR ANALYSIS OF DIGITAL SATELLITE LINKS

THESIS

William C. Jackson
Captain, USAF

AFIT/GE/ENG/86D-38

AFIT/GE/ENG/86D-38

DEVELOPMENT OF A MICROCOMPUTER-BASED WORKSTATION FOR ANALYSIS OF DIGITAL SATELLITE LINKS

THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
In Partial Fulfillment of the
Requirements for the Degree of
Master of Science in Electrical Engineering

William C. Jackson, B.S.
Captain, USAF

December 1986

Approved for public release; distribution unlimited

Preface

The purpose of this study was to develop a microcomputer-based engineering workstation for the analysis and design of digital satellite links. While there is lots of software for performing link budget analysis already on many mainframe computers, it is not particularly convenient for engineers doing real-time design work. There is also some software available for personal computers, but it exists largely as a poorly-documented hodgepodge of machine-dependent code developed by various organizations for their own specific purposes. I perceived a need for a transportable package of link analysis software which could run on a wide variety of personal computers. I decided to design my software for the IBM PC, since it represents by far the largest class of compatible machines.

As far as coding goes, I needed a programming language that had a good set of color graphics commands, as well as one that could be obtained at low cost. The obvious choice was Turbo Pascal.

I would like to thank Major Dale Hibner for his valuable advice and assistance in the planning and development of this rather massive undertaking. I also wish to thank my wife Pookie for graciously allowing me to put the color monitor on VISA. Now, about that Winchester drive.....

William C. Jackson

Table of Contents

List of Figures

ABSTRACT

The product of this ~~research effort~~ is a comprehensive and integrated software package for the analysis of digital satellite links. The system is designed to run on the IBM PC, and should also run on most compatibles.

The system performs three basic classes of functions: satellite orbital analysis, antenna gain pattern plotting, and link analysis. The first class of functions includes the computation of such quantities as velocity, orbital period, and coverage area for satellites in circular and elliptical orbits. The second class of functions is concerned with plotting the gain patterns for horns, helixes, parabolic reflectors, and phased arrays of dipoles. The last class of ~~functions~~ represents the major thrust of the system, and entails computing such items as the G/T figure of merit, received useful power, carrier-to-noise ratio, bit error rate, maximum data rate, and power margin. Inherent within this class are mathematical models for computing the attenuation due to rainfall and atmospheric absorption.

The link budget itself appears as a color-coded display with two columns: one for the uplink path, and one for the downlink path. The user also has the capability to change certain key inputs, and then have the system automatically recompute the entire link budget with the modified data.

DEVELOPMENT OF A MICROCOMPUTER-BASED WORKSTATION FOR ANALYSIS OF DIGITAL SATELLITE LINKS

I. INTRODUCTION

Orbiting communications satellites play a vital role in information transmission well beyond line-of-sight. One frequently hears the satellite transponding process described as "bouncing" a signal off of a satellite. This is really a misnomer, since modern communications satellites are a far cry from the early ECHO series passive reflector satellites. The transponding process is actually an active one, and consists of two distinct components: the uplink and the downlink. The uplink signal is transmitted from a ground station to the satellite, where it is translated in frequency and re-transmitted as a downlink signal to another ground station. The link itself can best be described as the total transmission path from the first ground station transmitter, through the channel, to the receiver at the second ground station. This includes all the modulators, transmitters, receivers, demodulators, and antennas both on the ground and onboard the satellite. The channel is the total electromagnetic path connecting a given transmitter-receiver pair, and includes the atmosphere as well as free space.

In a typical satellite communications link, the received signal may degrade in one of two ways: through a loss in desired waveform power, or through the addition of noise to the signal. Losses occur when a portion of the signal is

diverted or reflected from its intended route, or is attenuated along its intended route. Noise arises from unwanted signal energy being injected into the link, or from thermal noise generated within the link. In a typical link, free space propagation loss is the single largest loss in signal power. Prominent sources of noise include the internal noise of receiver amplifiers and feed lines, as well as the noise introduced by receiving antennas.

One of the most important tools available to the engineer involved in the design and analysis of communications satellite links is link budget analysis. A system link budget is essentially a balance sheet of gains and losses. It is comprised of a detailed listing of transmission and reception sources, noise sources, and signal attenuators as measured along the entirety of the link. A good part of the link budget is derived from the calculation of received useful power. Link budget analysis is basically an estimation technique for evaluating communications system performance. Link budgets are especially useful for the following (10:1-3):

1. Determining hardware constraints
2. Predicting system performance
3. Discovering system design flaws
4. Experimenting with various design tradeoffs
5. Predicting system availability
6. Highlighting system nuances
7. Searching for an optimal design

The primary purpose of this research effort is to develop a comprehensive set of software routines for performing link budget analysis, and to integrate them into a transportable package for use on a personal computer. Secondary goals include developing software to compute various satellite orbital parameters, as well as software to plot antenna gain patterns. This software in effect turns a personal computer into a dedicated satellite link analysis workstation.

It is assumed that the user has at least a cursory knowledge of digital satellite communications, and is familiar with link budgets. While this project might well be useful as a learning tool, it is not intended to be a tutorial for the novice. No particular knowledge of the software is required to successfully run it, since it is largely menu driven. However, a user's manual is included as Appendix A for those seeking more detailed information. It is also assumed that the user's machine is an IBM PC or compatible with 128K of memory, at least one floppy disk drive, and a color monitor.

Several other technical assumptions were made regarding the link:

1. All noise is thermal additive white gaussian noise, with a flat power spectral density of N_o for all frequencies of interest.
2. Signal bandwidth is taken to be equal to noise bandwidth.
3. The mathematical models for computing atmospheric and rainfall attenuation neglect such ubiquitous

factors as polarization losses, ice crystal effects, multipath propagation, and atmospheric scintillation.

4. No provision is made for error-correcting codes.

5. Pulse shaping techniques are not taken into account, so information transmission is subject to the constraints dictated by the Nyquist/Shannon criteria.

In a software project of this magnitude, it is necessary to use a highly structured design methodology. The first step is to establish an initial set of system requirements, recognizing that these requirements are fluid, and subject to updating as development proceeds. It is important to generate specific and meaningful requirements before any software design takes place, so that the requirements will drive the design, and not vice-versa. Once this is done, the overall system architecture must be defined. This step establishes the primary conceptual issues upon which the rest of the system is based. In addition, the software modules comprising the system are identified. Next, the detailed design is accomplished, where the software modules themselves are designed. It is at this level that fundamental algorithms and data relationships are defined. Finally, the completed software must be validated through a comprehensive testing procedure.

While the design is performed in a top-down manner, the software itself is coded in a bottom-up manner. The general methodology is that the lowest level modules are written first, followed by any higher-level modules (modules which

call other modules). The main program is written last, and the completed modules are integrated into it. Module testing may occur before or after the integration phase. The final system check is performed with all modules tested and in place in the main program.

II. THEORY OF SATELLITE COMMUNICATIONS

This chapter presents a brief overview of satellite communications theory, to include selected aspects of space communications as well as orbital mechanics. This chapter is not intended as a tutorial on the principles of electronic communications per se, but rather as an introduction to the anatomy of a satellite communications system.

Link Description

For the purposes of this paper, a satellite communications system consists of two earth stations separated by some distance, and an orbiting satellite. The complications of satellite-to-satellite communications or multiple users are not considered. Since the satellite transmits the downlink by responding to the uplink, it is referred to as a transponder. If this transponding process were accomplished passively, then the downlink power level would be extremely low due primarily to the large path losses. An active satellite transponder assists this process by adding power amplification to the signal prior to downlink transmission. However, transmitting the downlink at the same frequency as the uplink would cause unwanted feedback, so the transponder must perform some sort of frequency translation prior to amplification and re-transmission. Separation of the uplink and downlink frequency bands also allows the same antenna to be used for both receiving and transmitting (2:7-8).

System Components. The behavior of a satellite link is best understood by analyzing each of its components, the term "component" being taken to mean a source of gain, loss, or noise. Each component belongs to one of three sections of the link: the transmitter, channel, or receiver. Figure 2-1 illustrates the major components in a typical satellite link. This figure could represent either an uplink or a downlink. The transmitter section consists of a high-power microwave transmitter and an antenna, with some type of cable or waveguide between the two. The circuit losses include the effects due to both the lossy line and imperfectly matched couplings. The efficiency of the antenna takes into account losses due to reradiation, scattering, and spillover.

The channel section consists of free-space path losses, atmospheric losses, and rainfall losses. Path loss is due to the decrease in electric field strength with distance, and is unavoidable. Atmospheric losses are due to absorption of the signal by atmospheric oxygen and water vapor. Rainfall losses are due to the attenuation of the signal by rain at the uplink or downlink site, or both. The effects of atmospheric and rainfall attenuation can be minimized by an appropriate choice of frequency.

The receiver section consists of the receiving antenna, a low-noise receiver, and some type of coupling between the two. This side of the link is similar to the transmitter side, with some additional complications. The receiving

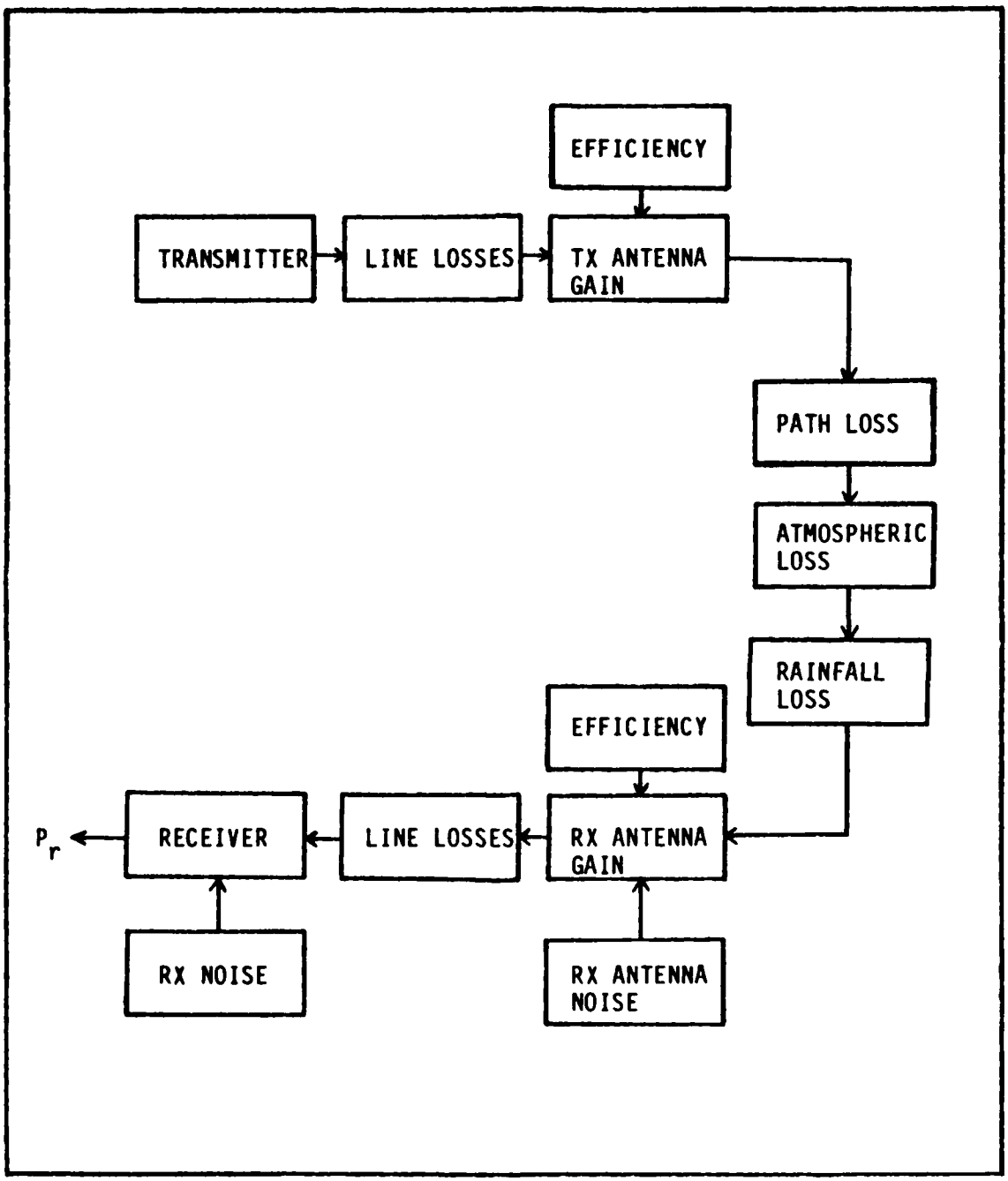


Figure 2-1. Major Components of a Typical Satellite Link

antenna noise temperature now contributes to system noise. Sources of antenna noise may include galactic noise, man-made noise, as well as atmospheric absorption. For earth coverage satellite antennas, 290^{o}K is generally used as the antenna temperature; for ground station antennas it may be much less, since the antenna is looking out toward a cooler sky temperature. Receiver noise is also a fundamental source of system noise. For this reason, the receiver amplifier(s) are sometimes cryogenically cooled to lower their noise temperature (10:1-3).

Design Considerations. Because size and weight are not limiting factors, ground station transmitters may be arbitrarily large. Power outputs of 40-60 dBW are common. In addition, the antenna transmitting the uplink may also be quite large. Large earth stations may employ parabolic dishes with diameters of 13-19 m. Conversely, satellite transmitters and antennas are subject to severe size and weight constraints. Spaceborne transmitter powers may be less than 10 W; spaceborne parabolic dishes are typically less than one meter in diameter.

Uplink and downlink frequencies are not chosen arbitrarily. Above 10 GHz, atmospheric attenuation is a growing concern. Satellite communications bands are specifically allocated to avoid the resonance absorption due to water vapor at 22 GHz, and oxygen at about 60 GHz. In general, it is easiest and most economical to use the lowest available frequencies. However, bandwidth is limited in the

lower frequency bands due to the large number of other users, and interference is more likely (2:106-112; 9:104-105).

Signal Analysis

In order to understand the concept of satellite link analysis, it is necessary to analyze the signals at various points in the link. The behavior of signals in the transmitter, antenna, channel, and receiver is examined separately, and various system performance criteria are given.

Transmitter (2:83). The transmitter field is characterized by its effective isotropic radiated power (EIRP), which is given by

$$EIRP = P_{out} + TxGain + TxCircuitLoss \quad (1)$$

where P_{out}=transmitter power (dB)

TxGain=transmitting antenna gain (dB)
TxCircuitLoss=transmitter circuit losses (dB)

Antenna (2:87-90). The antenna is a key element of a satellite communication system since its gain has a direct effect on the amount of received power. An antenna may be specified in terms of its gain and 3-dB beamwidth. Gain is proportional to the square of the frequency and to the antenna size, while beamwidth varies inversely to frequency and size. Formulas for the gains and beamwidths of several different antenna types are given below:

Horn:

$$G = 4*pi*d^2/L^2 \quad (2a)$$

$$BW = .88L/d \quad (2b)$$

Helix:	$G = 15c1/L^2$	(3a)
	$BW = .91(L/c1^{1/2})$	(3b)
Dish:	$G = (pi*d/L)^2$	(4a)
	$BW = 1.02L/d$	(4b)
Dipole Array:	$G = (N*pi*s/1.4L)^2$	(5a)
	$BW = .87(L/Ns)$	(5b)

where d=antenna dimension
L=wavelength
c=helix circumference
l=helix length
N=number of elements
s=element separation (2:89)

Channel (2:15-16,84-85). The most significant source of signal loss in the channel is free-space propagation or path loss, which is given by

$$\text{Path Loss} = -36.6 - 20 \log[Sf] \quad (6)$$

where S=slant range (mi)
f=frequency (MHz)

The slant range itself is computed from

$$S = (R_e^2 \sin^2 a + 2R_e h + h^2)^{1/2} - R_e \sin a \quad (7)$$

where R_e=radius of the Earth

a=terminal elevation angle
h=satellite altitude

Atmospheric attenuation and rainfall also contribute to total channel losses, and are discussed in detail in Appendices B and C.

Receiver (9:113-118). The amount of receiver noise present is given by the system equivalent temperature. This is the effective equivalent temperature that an external

noise source would need to have in order to produce the same amount of receiver noise. It is given by

$$T_{eq} = \text{RxAntTemp} + [(F/\text{RxCircuitLoss}) - 1]*290 \quad (8)$$

where RxAntTemp=receiving antenna temperature (deg K)
F=noise figure
RxCircuitLoss=receiver circuit losses (W)

A receiver may also be characterized by its G/T figure of merit, which is given by

$$G/T = \text{RxGain}/T_{eq} \quad (9)$$

where RxGain=receiving antenna gain (W)
T_{eq}=system equivalent temperature (deg K)

System Performance (2:104-105; 9:110-113). There are several criteria which indicate the quality of the link. These include received useful power (P_r), carrier power-to-noise power density ratio (CNR), bit energy-to-noise power density ratio (E_b/N_o), bit error rate (BER), and maximum data rate.

Received useful power is given by

$$P_r = P_{out} + \text{TxCircuitLoss} + \text{TxGain} + \text{TxEff} + \text{PathLoss} + \text{AirAtten} + \text{RainAtten} + \text{RxEff} + \text{RxGain} + \text{RxCircuitLoss} \quad (10)$$

where TxEff=transmitting antenna efficiency
RxEff=receiving antenna efficiency
PathLoss=loss due to propagation over a distance
AirAtten=atmospheric attenuation
RainAtten=attenuation due to rainfall
(all units are dB)

The carrier power-to-noise power density ratio is given by

$$CNR = P_r/kT_{eq} \quad (11)$$

where P_r=received useful power (W)

k=Boltzman's constant
T_{eq}=system equivalent temperature (deg K)

Available E_b/N_o is given by

$$E_b/N_o = P_r/RN_o \quad (12)$$

where P_r=received useful power (W)

R=data rate (bits/sec)
N_o=noise power density (W/Hz)

The bit error rate is determined from the modulation type and available E_b/N_o. The expressions for bit error rate for eleven different modulation formats are listed below, where Q(x) is Marcum's Q-function (2:53-68).

BPSK,QPSK,OQPSK,MSK: $Q([2E_b/N_o]^{.5})$ (13)

MPSK: $Q([2\log_2 M\ E_b/N_o]^{.5} \sin \pi/M)/\log_2 M$ (14)

Coherent FSK: $Q([2E_b/N_o]^{.5})$ (15)

Non-Coherent FSK: $.5\exp(-E_b/2N_o)$ (16)

MFSK: $M/4\ \exp(-\log_2 M\ E_b/N_o)$ (17)

ASK: $Q([E_b/N_o]^{.5})$ (18)

MASK: $[(M-1)/M]\ Q\{[6\log_2 M]/[(M^2-1)E_b/N_o]\}/\log_2 M$ (19)

DPSK: $.5\exp(-E_b/N_o)$ (20)

By algebraically manipulating the above expressions, the E_b/N_o required for a given bit error rate and modulation type may be found; these new expressions now involve inverse Q-functions. The difference between the available E_b/N_o and the required E_b/N_o is called the power margin. Finally, the

maximum data rate is given by

$$\text{maxrate} = P_r/(kT_{eq}\ E_b/N_o) \tag{21}$$

where P_r=received power (W)

kT_{eq}=noise power density (W/Hz)

E_b/N_o=required bit energy-to-noise power density ratio

Orbital Aspects

This section presents some of the salient physical aspects of orbiting satellites, beginning with a brief discussion of orbital mechanics. In addition, a methodology for determining look angles is also presented.

Orbital Mechanics (9:11-22). In order to be placed into orbit, a satellite must escape the influence of the Earth's gravitational field. This escape velocity is a function only of altitude, and is given by

$$V_{esc} = [2GM_e\ /\ (Re + h)]^{.5} \tag{22}$$

where G=Earth gravitational constant
h=altitude of launch site
M_e=mass of Earth

R_e=radius of Earth

Once in space, the satellite is placed into either a circular or an elliptical orbit. For the case of a circular orbit, the satellite must achieve a velocity of

$$v = [u\ /\ (R_e + h)]^{.5} \tag{23}$$

where $u=GM_e$ (gravitational coefficient)

The orbital period, or time to complete one revolution, is given by

$$T = 2*pi*[R_e + h]^{1.5}/u^{.5} \quad (24)$$

Another important consideration is coverage area. This refers to the area on the Earth from which the satellite is visible from some pre-determined minimum elevation angle. Refer to the set-up depicted in Figure 2-2. The subtended angles may be determined from simple trigonometry to be

$$s = \cos^{-1}[R_e \cos El / (R_e + h)] - El \quad (25)$$

where h=satellite altitude
El=minimum elevation angle

The coverage area is therefore the area of the spherical cap subtended by the angle s, and is given by

$$A_{cov} = (2*pi*R_e^2)(1 - \cos s) \quad (26)$$

The maximum length of time that an orbiting satellite is visible to an observer on the Earth's surface is referred to as the satellite's duration of visibility. Duration of visibility is a function of subtended angle and orbital period. It is expressed as

$$t_{max} = [2s/360][T/(1 \pm T/T_e)] \quad (27)$$

where T_e=Earth's orbital period

\+ = retrograde orbit
\- = prograde orbit

Note that for the special case of a prograde geostationary orbit, the satellite is visible continuously.

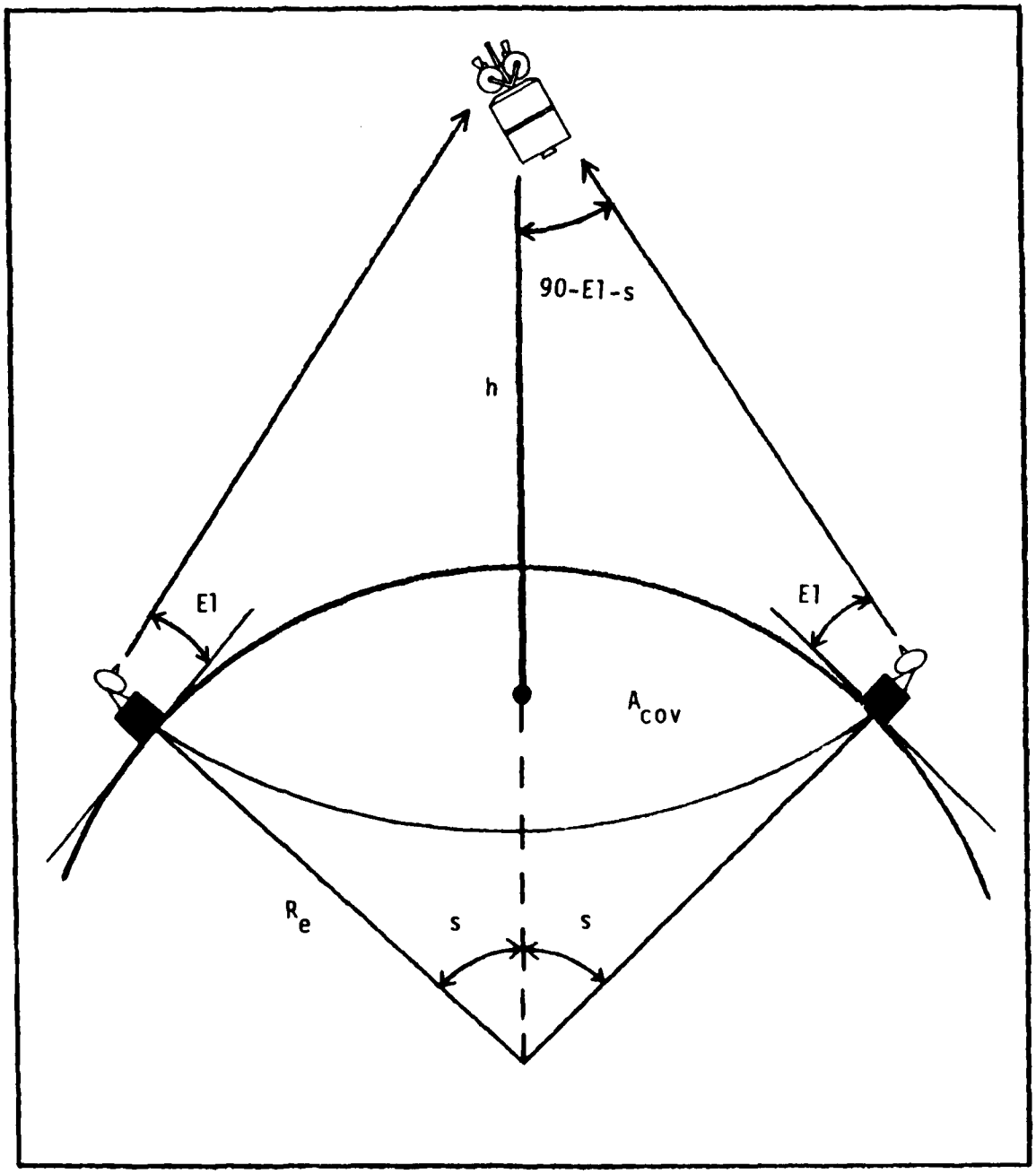


Figure 2-2. Geometry For Determining Coverage Area

For the case of an elliptical orbit, the satellite velocity is not constant, but varies with position. Velocity is highest at the orbital perigee (low point of orbit), and lowest at the orbital apogee (high point of orbit). The major and minor axes a and b of the orbital ellipse are given respectively by

$$a = (R_p R_a)^{.5} \tag{28a}$$

$$b = (R_p + R_a)/2 \tag{28b}$$

where R_p=altitude at perigee

R_a=altitude at apogee

From these, the eccentricity of the orbit may be expressed as

$$E = (1 - b^2/a^2)^{.5} \tag{29}$$

The polar equation for the orbit is then

$$r = a(1 - E^2)/(1 + E \cos O) \tag{30}$$

where r=radial distance from geometric center of ellipse
O=reference angle (zero at perigee)

The satellite's velocity at any given radial distance r is given by

$$v = [u(2/r - 1/a)]^{.5} \tag{31}$$

The orbital period is given by

$$T = 2*pi*a^{1.5}/u^{.5} \tag{32}$$

The formula for coverage area is similar to the one given for a circular orbit, with the exception that the subtended angle is now given by

$$s = \cos^{-1}[R_e \cos El / (R_e + r)] - El \qquad (33)$$

Look Angle Determination (9:22-30). The coordinates to which an earth station antenna must point in order to communicate with a satellite are referred to as the look angles. They are usually specified as azimuth and elevation angle. The geometry of the problem is given in Figure 2-3. Given the coordinates of the earth station (lat,long) and the coordinates of the sub-satellite point (lat_s,$long_s$), the great circle distance between the two is given by

$$R = \cos^{-1}[\cos(lat - lat_s) \cos(long - long_s)] \qquad (34)$$

The azimuth is given by

$$Az = \tan^{-1}[\tan(long - long_s) / \sin(lat - lat_s)] \qquad (35)$$

The tilt angle is the angle between the satellite vertical to the Earth and the line-of-sight to the ground station. Normalizing the satellite's altitude h with respect to the Earth's radius, one may define $p=1+h/R_e$. The tilt angle is then expressed as

$$Tilt = \tan^{-1}[\sin R / (p - \cos R)] \qquad (36)$$

Finally, the elevation angle may be expressed in terms of the tilt angle by

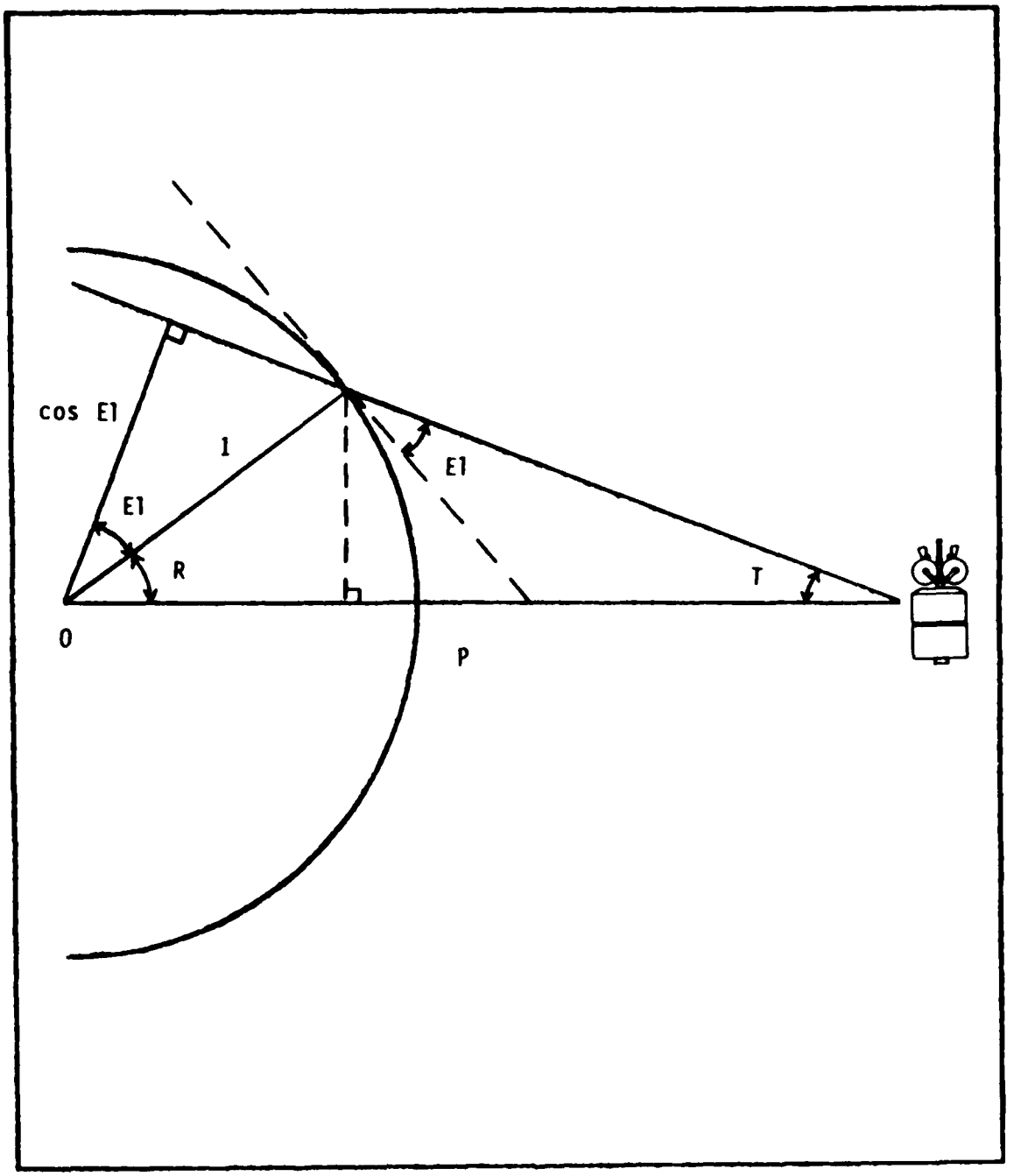


Figure 2-3. Geometry For Determining Look Angles

$$El = \cos^{-1}[p \sin(Tilt)] \qquad (37)$$

This chapter has presented the user with the basic theory needed to understand satellite link analysis. The major components of a typical satellite link were described qualitatively, and several important design considerations were discussed. In addition, the behavior of the signal in each principal section of the link was described quantitatively, and several system performance criteria were stated. Finally, a brief synopsis of orbital mechanics was presented, along with a method of determining look angles.

III. SYSTEM REQUIREMENTS

This section addresses the specification of required functions, interfaces, and performance of the end software product. The intent is to present a complete and unambiguous set of functional specifications describing what the software is required to do, rather than how it is to be implemented at the coding level. This phase of software development is crucial, because inadequate specification of requirements often leads to system deficiencies which are difficult to correct at a later date. In addition, system testing tends to have little real meaning without a concrete set of requirements as a basis.

The essential elements of a good requirements definition are testability and flexibility. This means that it must be possible to verify that the end product satisfies the given specifications. Also, the specifications must not be so rigid as to inhibit program development.

Functional Description

The following is a global description of the system in terms of the functions that it must accomplish. No attempt will be made here to derive equations or develop algorithms, as this is merely a statement of intended capability. For the purposes of this chapter, required functions are classified as primary, secondary, or housekeeping functions. Primary functions are those which represent the major thrust of the thesis, and include antenna analysis,

receiver/transmitter analysis, and various other system design calculations. Secondary functions are those which supplement the overall system, but are not required for system operation, such as calculation of satellite orbital parameters and plotting antenna gain patterns. Housekeeping functions are those which perform no calculations, but are responsible for systems-level tasks such as storing and manipulating data.

Finally, design constraints such as portability, expandability, and audit capability are discussed. Note that design constraints specify how the system is to be constructed and implemented, not necessarily which functions are to be included in the system.

Primary Functions. As a minimum, this set performs the functions listed below. Note that the first item listed applies equally to horns, helixes, parabolic dishes, or phased arrays of dipoles.

1. Compute the maximum antenna gain and half-power beamwidth.
2. Compute the effective isotropic radiated power.
3. Compute the total system noise temperature and G/T figure of merit.
4. Calculate the attenuation due to rainfall and atmospheric absorption.
5. Calculate received useful power as governed by the link equation.
6. Compute the carrier power-to-noise power density ratio (CNR).

7. Determine the maximum data rate as a function of CNR, desired bit error rate, and type of modulation.

8. Compute the bit error rate given the CNR, data rate, and type of modulation.

9. Compute the link margin.

Secondary Functions. This set of functions should be considered optional, and has absolutely no bearing on link budget calculations. These functions concern satellite orbit parameters and antenna gain pattern plotting, and perform the following:

1. Calculate escape velocity at a given altitude above the Earth's surface.

2. Compute the steady-state temperature of a satellite in Earth orbit.

3. Determine velocity, orbital period, coverage area, and duration of visibility for satellites in circular orbit.

4. Determine velocity at apogee and perigee, average velocity, orbit eccentricity, orbital period, and coverage area for satellites in elliptical orbit.

5. Compute the azimuth and elevation angles to a satellite from a given ground site.

6. Plot the antenna gain patterns for horns, helixes, parabolic dishes, and phased arrays of dipoles.

Housekeeping Functions. These functions are primarily concerned with disk access and database management. A capability is included to store and retrieve files of data, as well as to delete unwanted files. The user is also

provided the capability to change the default drive to which data is written, and to review the files residing on a given data disk. These functions are designed to mimic the corresponding functions in DOS.

Design Constraints. There are several design constraints which drastically affect a programming effort of this type. The first is that of portability. Because the resultant software must run on a variety of machines, it is necessary to incorporate a certain amount of flexibility in the early stages of system design in order to avoid the problem of machine dependence. All code is written in TURBO Pascal, which is available in both MS-DOS and CP/M versions, and therefore can run on a wide variety of microcomputers. For this reason, one copy of the program is left as source code, in order to facilitate transportability from an MS-DOS machine to a CP/M machine and vice-versa. The only possible problem area (with regard to machine dependence) is that of display graphics. In order to circumvent this problem, all graphics are generated by a graphics driver module. While this module itself is highly machine dependent, it can be easily removed intact and replaced by another custom driver module for a different machine.

The second constraint is that of expandability. Since system requirements tend to be in a constant state of flux in a project of this nature, it is important not to make the system design too rigid. In other words, as the need for new functions arises, one should not have to completely

revamp the entire architecture in order to accommodate it. The key to expandability is therefore modular design. This allows logical units to be inserted and deleted at will.

The last constraint to be addressed is that of audit capability. This refers to the calculation and storage of intermediate results during data entry or calculations. Incorporating an audit capability serves several purposes. First, it facilitates system testing by storing an audit trail of both module function and module interaction. Secondly, it permits the user to go back through intermediate calculations when seemingly erroneous results are obtained and discover where the error occurred. Finally, in the unlikely event of a program crash, significant data entered prior to that point will be stored in memory for subsequent recovery.

Man-Machine Interface

The success of any interactive software design effort is measured by how well the system insulates the user from technical details of the software. While the programmer must concern himself with system architecture, data structure and flow, and control structure and flow, the user is only concerned with these issues as far as they affect operations at the terminal. As far as the user is concerned, the system is a black box. This section addresses some of the issues encountered in designing such a system.

User Interaction. A variety of user-friendly features are incorporated into a typical interactive software system. The most obvious feature is the use of menus. A menu is simply a display of a set of program options, from which the user is asked to select at least one. This means that the user is able to move freely through the hierarchy of the system simply by entering a response to a list of options. The use of menus eliminates guesswork on the part of the user, and consolidates all necessary user decisions into convenient, easy to use subsets.

Closely associated with menus is the use of prompting and validity checking. Prompting is the use of printed test messages asking the user to enter some specific data, or to perform some other particular function. This eliminates confusion as to exactly what data is required, what format the data should be in, and what units (if any) the data should be in. Validity checking refers to the practice of subsequently analyzing user-entered data and ensuring that it conforms to some set standard (e.g. within a specified range of validity). If the data fails the validity check, then the user is alerted and asked to re-enter it. This ensures that all data going in to the program is valid.

At this point something should be said regarding development of the user's guide. During the initial stages of system planning and design, the programmer must constantly make tentative decisions and assess their impact on the overall system. As time progresses and previously

unseen constraints and limitations appear, it may become necessary to change earlier decisions. The programmer is now falling into the trap of letting the code dictate system operation. This pitfall can be avoided by developing a preliminary version of the user's guide in parallel with the design. This practice is a form of self-imposed programming discipline, and ensures that the original system requirements will in fact drive the implementation.

Visual Display of Data. A significant part of the man-machine interface is the manner in which information is presented to the user. One aspect of this is the use of color. Used intelligently, color can greatly enhance the readability of both text and data. For example, a table consisting of several different sets of data might use a different color for each set in order to more easily differentiate them. With a standard RGB color monitor, the IBM-PC places certain restrictions on the use of color. In the text mode, any one of sixteen foreground colors and eight background colors may be selected; in the medium-resolution graphics mode, the user may choose colors from one of four color palettes, each containing three colors plus the background color; in high-resolution graphics mode, the background color is always black, but the user may choose one of sixteen foreground colors.

In order to be readily interpreted by the user, numerical data must be presented in a clear and intelligible format. Merely filling the screen with numbers is not necessarily

the most efficient way to accomplish this. One way to improve readability is to group related data together; different groups may be separated by collimating the data. Finally, all data should be labeled and units stated where appropriate.

Similar requirements exist for graphical data, but now there are two modes to worry about. When multiple curves are drawn on the same graph, the medium-resolution mode might be more appropriate since it allows multiple colors. However, if the curves are highly detailed, the high-resolution mode might better emphasize these details. So there is a trade-off between color differentiation among data and resolution. Note also that it takes much longer to obtain hardcopy of high-resolution displays than of medium-resolution displays.

Software Validation (5:36-119)

It is absolutely essential that all software be tested not only extensively, but intensively as well. In other words, while the programmer must obviously test all routines in the program under scrutiny, he/she should also "pick apart" each routine to examine how it functions under all possible circumstances. Despite all efforts at objectivity, this testing philosophy nevertheless tends to be counter-intuitive to the programmer testing his own code. For this reason, a rigorous testing protocol should be developed and adhered to. This section will address the basic principles of test case formulation and program testing procedures.

Test Case Formulation. There are five general test case formulation methodologies of interest to the programmer. The first is a so-called "white-box" methodology; the rest are "black-box" methodologies. The term "white-box" implies that the tester is concerned with the internal structure of the program. White-box testing is concerned with the degree to which the established test cases span the logic of the program. Therefore, the test data itself is generated by a detailed examination of program logic. The ultimate white-box test is the execution of every control path in the program. However, this is generally not feasible for programs with many levels of loops. One type of testing, logic-coverage, represents an attempt to establish alternate criteria for program correctness.

There are several approaches to logic-coverage testing. The simplest is known as statement coverage. This criterion requires that every statement in the program be executed at least once. A more stringent test criterion is that of decision coverage, which requires that test data be generated such that each decision has at least one true and one false outcome. However, decision coverage is inadequate if multiple conditions are involved in a decision. Therefore a more complete criterion is that of condition coverage. This criterion requires that each condition in a decision takes on all possible values at least once. The above criteria can also be used together in a hybrid approach.

The remaining discussion of test case formulation concerns "black-box" methodologies. The term "black-box" implies that the tester is not concerned with the internal structure or behavior of the program, but rather with the conditions under which the code fails to meet its initial requirements. Therefore, the test data is derived solely from performance specifications. The ultimate black-box test is to perform exhaustive input testing over all possible inputs. For more complicated programs this number becomes astronomical. However, this seemingly infinite set of test cases may be reduced to a finite and quite manageable set if certain assumptions are made regarding the program's response to carefully chosen data. This is the goal of black-box testing. Black-box methodologies include equivalence partitioning, boundary-value analysis, and error guessing.

In selecting a small subset of inputs for testing, it is obviously important to choose one which has the highest probability of finding the most errors. A well-selected test case should cover a large subset of other possible test cases, and thereby reduce the number of other test cases needed. This may be accomplished by partitioning the input into a finite number of equivalence classes. An equivalence class is a set of input data chosen such that a test performed on a representative value of each class is equivalent to a test performed on any other value in that class. All possible input conditions should be partitioned

into valid and invalid equivalence classes. Test cases should be written covering as many valid equivalence classes as possible for the sake of thoroughness. Individual test cases, each covering a single invalid equivalence class, should also be written. This is necessary because some erroneous input checks may mask others.

In equivalence partitioning, a random member of a class is the subject of the test. However, this methodology ignores the case of a value directly on the edge of a valid equivalence class. The methodology of specifically examining the edges of valid equivalence classes is known as boundary-value analysis. Test cases are derived by considering both input and output equivalence classes. One should check the range limits of test variables as well as just beyond these limits.

The last black-box testing methodology to be discussed is error guessing. Error guessing is an extremely informal technique based on programmer intuition and experience. One approach is to simply list possible error situations, and then write test cases based upon that list. Another approach is to identify test cases based on programmer assumptions made regarding the initial specifications. In any case, the development of test cases generally revolves around the programmer's hunches concerning what types and combinations of input will generate an error.

It should be noted that none of the above methods is complete in itself. A prudent test strategy would certainly

invoke several, if not all, of these methods. A logical order of testing would be boundary-value analysis first, followed by equivalence partitioning, error guessing, and logic-coverage.

Program Testing. A large program is generally not tested all at once, but rather functional modules are tested individually. By narrowing the scope of testing to a more manageable level, it becomes possible to achieve a certain element of parallelism by testing several modules simultaneously. Debugging is also facilitated, since it is easier to discover and trace errors.

There are two considerations in module testing: module function and module interconnection. Corresponding to this are two major philosophies of program testing: non-incremental testing and incremental testing. In non-incremental testing, all modules are first tested individually and then combined to form the program. In order to do this, special driver modules must be written to transmit test cases to the module under test and display results. In addition, one or more stub modules must be written to receive control when a module under test invokes another. The stubs should return a meaningful result to the calling module as well as returning control.

In incremental testing, the module under test is combined with previously checked and validated modules. As modules are progressively linked together, the individual module functions as well as the soundness of their interconnections

are tested, until the entire program has been pieced together. Incremental testing has several advantages over non-incremental testing. First, there are fewer drivers and stubs to write. Also, errors related to interface problems will be detected earlier, since combinations of modules are tested together early. Finally, incremental testing results in a more thorough test, since executing a module as a result of an invocation by another module might trigger a previously undetected error condition.

There are two basic strategies for performing incremental testing: top-down testing and bottom-up testing. They each have their relative advantages and disadvantages, and the choice of which method to use is highly dependent upon the characteristics of the program being tested.

In top-down testing, one starts at the top of the program and tests downward. No drivers are needed since we are beginning execution at the top of the program, but stubs must be prepared for each non-terminal module at the current hierarchical level of test. A terminal module is simply one which does not call any other modules. The test proceeds downwards until the program has been formed. The only criterion for determining the next module to test is that at least one of that module's calling modules must have been previously tested. Top-down testing provides an initial program skeleton that allows early program demonstrations and proves the overall design.

In bottom-up testing, no stubs are needed since testing begins at the terminal modules, but drivers must be written

for the modules under test. The only criterion for choosing the next module to test is that all of its subordinate modules must have been previously tested. Testing proceeds upwards until the program has been formed. Test conditions are easier to create in bottom-up testing, and the observation of test results is easier. However, the program does not exist as an entity or even as a functional subset until the last module has been added.

After module testing has been completed, some type of higher-order testing is appropriate. Software development is largely a communication process, where information regarding the eventual program is translated into various forms. For example, system requirements are translated into program objectives, and these are translated into external specifications, and so forth. Breakdowns in this communication process may readily lead to errors.

The first higher-order test usually performed is function testing. This activity is designed to find discrepancies between a program and its external specification. Test cases are derived from the external specifications, and testing is accomplished using the previously mentioned techniques.

Another class of higher-order tests is system testing. Its purpose is to compare the program to its original objectives. Here, specifications cannot be used to derive test cases, since this would be defeating the purpose of the test. Instead, test cases are designed by analyzing general

program objectives. Some categories of system testing include facility testing (to see if every facility mentioned in the objectives is indeed implemented), usability testing (to determine how well the user interacts with the system), and compatibility testing (to see how easily the software can be installed on other systems).

This section has addressed the issue of system requirements. The problem of stating complete requirements is threefold, since one must address functional descriptions of the system, the user interface with the system, as well as a methodology for validating the end product. The requirements stated in this chapter are considered to be only a minimum set of criteria.

IV. SYSTEM DESIGN

The purpose of this section is to present the overall system design from a global perspective. System design may be thought of in terms of two distinct categories: system architecture and design structure. System architecture refers to the breakdown of the system into its major subsystems, illustrating the scope and relative hierarchy of each subsystem, as well as the relationships between subsystems. System design concerns the functional mapping into modules, and the interaction between modules.

Representation Techniques (8:44-62)

In this section, certain graphical techniques are used to represent system architecture and design structure. The following is a brief explanation of those techniques.

System Architecture. System architecture may be represented by a modified Leighton diagram. This type of diagram presents the issues of scope, hierarchy, and external interfaces without the complexity associated with excessive detail. Its notation consists simply of labels and line segments. The labels represent logical modules, and the line segments represent logical connection. The horizontal dimension portrays overall hierarchy, while the vertical dimension portrays scope of control.

Design Structure. Design structure is represented via structure charts. The structure chart illustrates the overall module hierarchy, as well as the interaction between

modules. Its notation consists of rectangles representing the modules, vectors representing control relationships, and arrows with circular tails representing transfer of data/control (solid circles represent boolean flags).

Synopsis of Overall System Architecture

The overall architecture of the system is concisely illustrated in Figure 4-1. The global module MAIN corresponds to the user's main menu. Via the appropriate menu input, control then passes to one of five subsystems: the satellite parameters group, the antenna gain patterns group, the link analysis group, the database support group, and the exit routine. Except for the exit routine, each subsystem consists of a number of functional modules, and hence are referred to as "groups." Each subsystem will now be discussed individually in detail.

Satellite Parameters Group. The function of this group is to perform several commonly used calculations concerning a satellite's orbit and position. Five modules within this group perform the following tasks:

1. Compute Earth escape velocity at an altitude.
2. Compute the steady-state temperature of a satellite in Earth orbit.
3. Compute various parameters associated with a specified circular orbit.
4. Compute various parameters associated with a specified elliptical orbit.

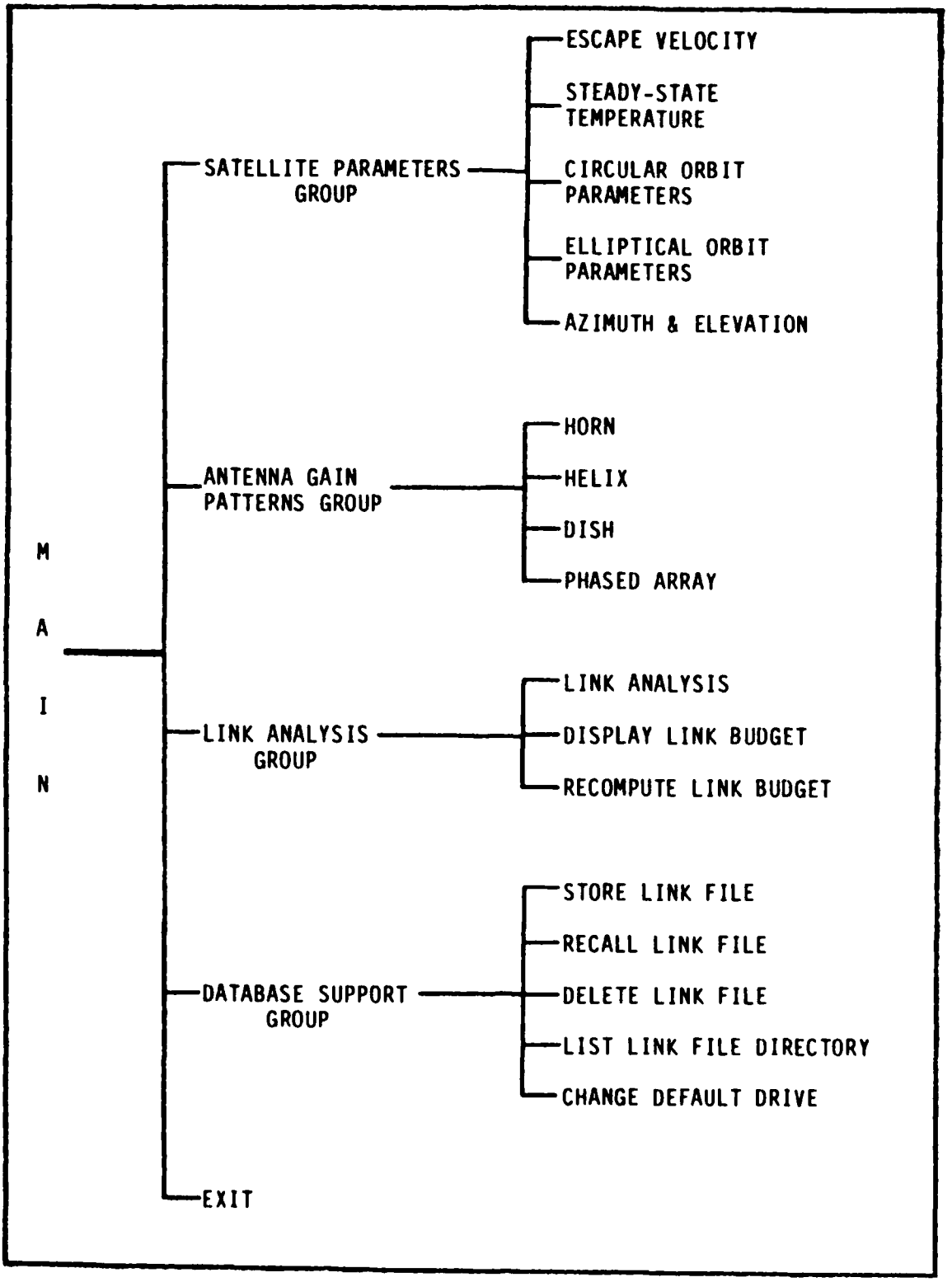


Figure 4-1. Overall System Architecture

5. Compute azimuth and elevation angles to a satellite from two different ground sites.

Only the last module has any external connection with the rest of the system. The computed elevation angles are stored in memory, and may be used as input during link analysis calculations. The remainder of the satellite parameter data is discarded upon return to the main menu.

Antenna Gain Patterns Group. The function of this group is to plot the gain patterns of various types of antennas. Four modules handle the pattern plotting for horns, helixes, parabolic reflectors, and phased arrays of dipoles. The first three modules also display pertinent design information along with the gain pattern for more meaningful hardcopy. The last module does not display this information, but does display each of the six standard polar antenna plots separately.

Link Analysis Group. This functional group constitutes the heart of the system. Although there are a large number of individual modules within this group, the user perceives only three distinct processes: link analysis, display link budget, and recompute link budget. The entire system is, of course, organized around the link analysis process. In this process, the user is guided step by step through a complex series of link calculations, with intermediate results being displayed along the way. The engineer is provided with all of the data necessary to design and analyze a satellite link.

The display link budget process displays all the pertinent data computed above, and organizes it in an easily readable and understandable format. The display has separate columns for uplink and downlink data, and is color-coded according to function and location in the link.

The recompute link budget process is included to allow the design engineer the opportunity to alter various key parameters, in order to determine the overall effect on the link. Given a new value of a parameter, the entire set of link calculations is recomputed with the new parameter without the user having to manually re-enter all of the previous input data. This feature becomes especially valuable when one is engaged in the process of making design tradeoffs, for example, allowing a smaller receiving antenna while increasing the receiver noise figure.

Database Support Group. The purpose of this group is to perform all of the housekeeping and disk management tasks associated with the link database. Thus, the user is spared the tedium of keeping track of what data is in what file, what files are open, and on which disk; the system also prevents the user from inadvertently erasing or not saving an important file. The user interfaces with this group via five options in the main menu. These options are concerned with storing newly computed link files, recalling old link files, deleting unwanted link files, listing the existing files in the link file directory, and changing the default data drive. These options are so complete in their function

and behavior that they strongly resemble the corresponding DOS commands.

Exit Routine. This routine is not associated with any distinct module, and is actually embedded within the master program. All it does is allow the user a clean way to return to DOS. If a recently computed/recomputed link file has not yet been saved, this routine gives the user the opportunity to do so before exiting.

Synopsis of Overall Design Structure

At this point the system design structure is presented in detail. The design structure may be considered one level of detail down from the architecture. The emphasis here is on module functionality and module interaction.

Explanation of Module Functionality. For future reference, the functional modules on the system disk are listed alphabetically below and described according to basic function.

AIR - Computes attenuation due to atmospheric absorption.

ALOG - Converts dB to numbers.

ANTENNA1 - Computes gains and beamwidths for earth station antennas.

ANTENNA2 - Same as above, only for satellite antennas.

ARRAY - Plots gain pattern for phased array of dipoles.

AZEL - Computes azimuth and elevation angle to a satellite from two different ground stations.

BESSEL - Computes first-order Bessel functions of the first kind ($J_1(x)$).

BITRATE - High-level module which computes available E_b/N_o, required E_b/N_o, and max data rate.

BLACKBOD - Computes steady-state temperature of a satellite in Earth orbit.

CAPACITY - Computes Shannon rate and Nyquist rate, and determines whether the link is channel-limited.

CIRCULAR - Computes velocity, orbital period, coverage area, and duration of visibility for a satellite in circular Earth orbit.

CUSTOM - Contains custom plotting and drawing routines.

DISH - Plots gain pattern for parabolic dish antennas.

ELLIPSE - Computes velocity at apogee and perigee, average velocity, eccentricity, orbital period, and coverage area for a satellite in elliptical Earth orbit.

ESCAPE - Computes escape velocity at altitude.

EXIST - Function that determines whether a given file exists on disk.

FUNCTION - Low-level module used by many other modules to perform actual calculations (e.g. gain, beamwidth, EIRP, system temperature, figure of merit, received useful power, CNR, available E_b/N_o, required E_b/N_o, BER, and max data rate).

GETALT - Obtains satellite altitude from the user.

HELIX - Plots gain pattern for helical antennas.

HORN - Plots gain pattern for horn antennas.

INTEGER - Polls the user for an integer input.

LOG10 - Computes log(x).

LOG2 - Computes $\log_2(x)$.

MASTER - Main program.

NOISE - Makes beeping sound (used to signal error conditions).

POWER - High-level module which computes received useful power and CNR.

PWR - Function to perform exponentiation.

Q - Computes Marcum's Q-function.

QINVERSE - Computes inverse Q-function.

RAIN - Computes attenuation due to rainfall.

REAL - Polls the user for a real input.

REC - High-level module which computes system temperature and figure of merit.

SAVEIT - Routine to save a link file on disk.

TRIG - Package of functions to compute arccos(x), arcsin(x), and tan(x).

WAIT - Suspends execution until a key is pressed.

XMIT - High-level module which computes EIRP.

Module Interactions. The interactions among the above modules can best be expressed by structure charts. For the sake of brevity, only those structure charts illustrating significant module inter-relationships are discussed. For example, the case of one module calling another module to compute a simple trigonometric function is not considered significant. All graphics routines call the module CUSTOM; by replacing this module with an alternate graphics driver, the system should be capable of plotting on any MS-DOS

machine. In addition, all modules requiring numeric inputs employ INTEGER and/or REAL to screen user input. Figure 4-2 presents the most important structure charts.

From this figure it may be seen that the modules CIRCULAR, BLACKBOD, and AZEL, as well as the main program, all employ the module GETALT to obtain a value for satellite altitude from the user. The main program interfaces with AZEL for the purpose of obtaining values for elevation angle. If the flag HAVEALPHA is true, then the user may choose to continue using the elevation angles previously computed in AZEL. The modules HORN, HELIX, and DISH all use the module FUNCTION to obtain values for maximum gain. DISH also uses the module BESSEL to actually calculate values for $J_1(x)$. The modules ANTENNA1 and ANTENNA2 both employ the module FUNCTION to obtain values for gain and beamwidth. The module REC uses FUNCTION to compute system equivalent temperature and figure of merit; XMIT uses it to compute EIRP. The module POWER calls AIR and RAIN to obtain values for attenuation due to the atmosphere and to rainfall, respectively, and FUNCTION to compute received power. The module BITRATE uses FUNCTION to compute available E_b/N_o, required E_b/N_o, and max data rate. It also calls CAPACITY to compute the Shannon rate and the Nyquist rate; if one of these rates is lower than the computed max data rate, then an appropriate flag is set and BITRATE makes the corresponding substitution for the max data rate.

This section has given the user an overview of the overall system design. There are two fundamental facets to

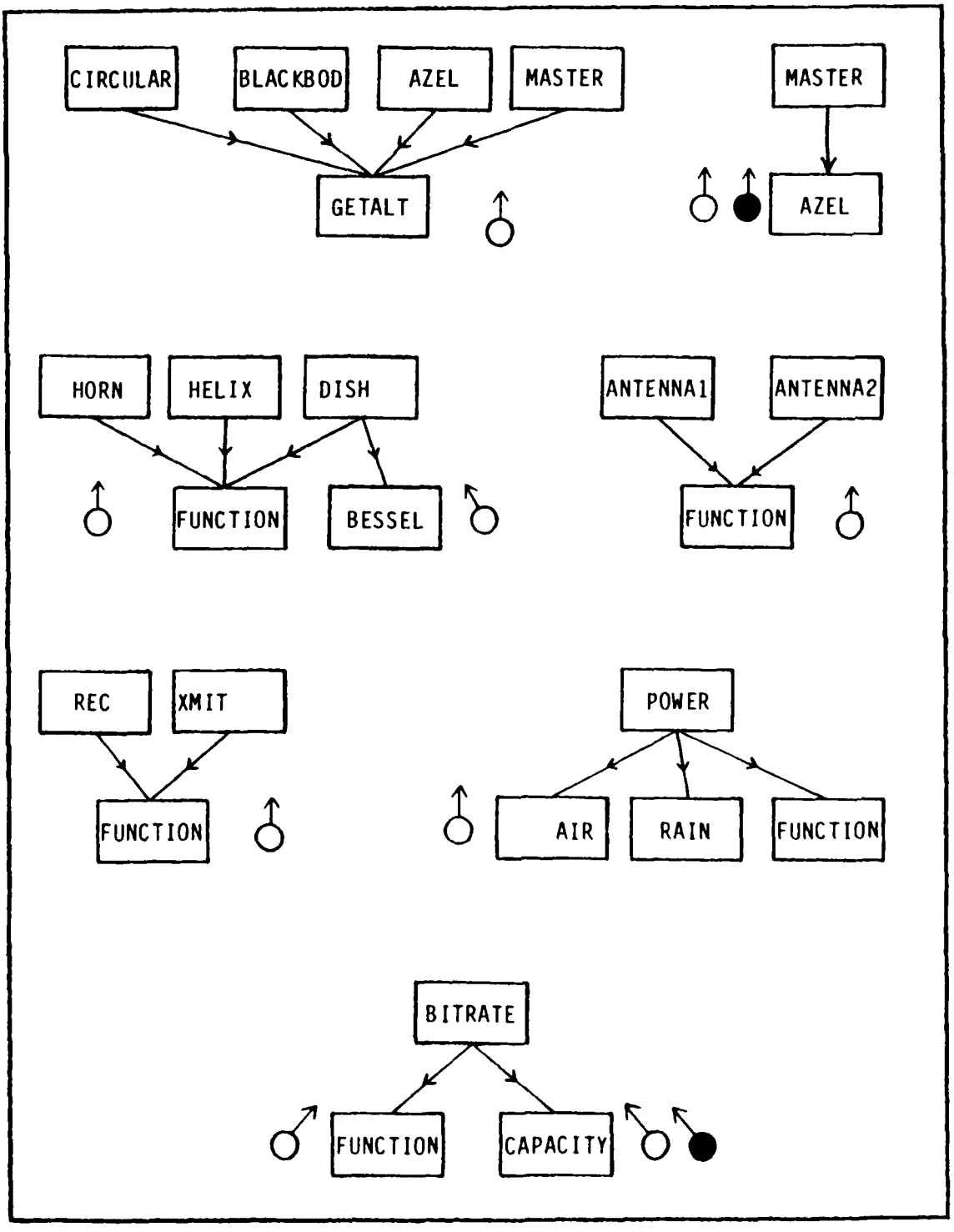


Figure 4-2. Significant Module Interactions

system design: system architecture and design structure. System architecture provides the user with insight regarding the decomposition of the overall system into its constituent subsystems; design structure examines the functions and inter-relationships of the modules which serve as building blocks for the system. A working knowledge of the overall system design gives the user a feel for how the system operates, without the frustration of having to dredge through complicated equations or piles of source code.

V. DETAILED DESIGN

In the previous section the system's functional modules were considered black boxes, and no attempt was made to explain their inner workings. It is the purpose of this section to examine in detail the structure and behavior of these modules. The issues of control flow and database management are also covered.

Representation Techniques

In this section, certain graphical techniques are used to represent control flow and database structure. The following is a brief explanation of those techniques.

Control Graphs (8:96-98). Control graphs are used to show the mechanisms by which a program may transition from one logical state to another. It is implicit that a program can only exist in one state at any given time. The notation consists of node symbols representing a given state, solid lines representing a control path, and dashed lines representing input from some external condition. Control graphs are borrowed from automata theory, and are used to establish that all nodes in a system can be reached, and that once reached they can be exited. The nodes themselves may be only conceptual entities, and the resulting control graph becomes an abstract program model rather than a physical model.

Database Structure (8:77-81). The purpose of database structure representation is twofold: to show flow of data

within the system, and to illustrate the basic file structure. There are many types of data structure representation techniques, each with its own particular merits. Because of the relatively simple database constructs in this system, only a rudimentary technique is needed. The data structure diagram used to describe this system consists simply of boxes representing disk files, and vectors representing directional access. If a key is required to access a particular file, then it is shown in a box adjoining the corresponding file.

Key Detailed Design Issues

The previous chapter on system design examined the system from a logical point of view, and the items discussed therein do not necessarily correspond to physical, implementable sections of the program. In addition, all functional modules were treated as black boxes. It is the purpose of this section to examine the system from a physical perspective, and to describe the system in terms of software behavior. Three major topics are discussed: flow of control, database management, and software design.

Flow of Control. Figure 5-1 illustrates the system flow of control. There are 13 states in the system, which correspond to the main menu, the 11 main menu selections, and DOS. Although DOS is technically not a program state, it is considered a system state. It may be seen from this figure that the initial program state is MAIN, which corresponds to the main menu from the user's perspective.

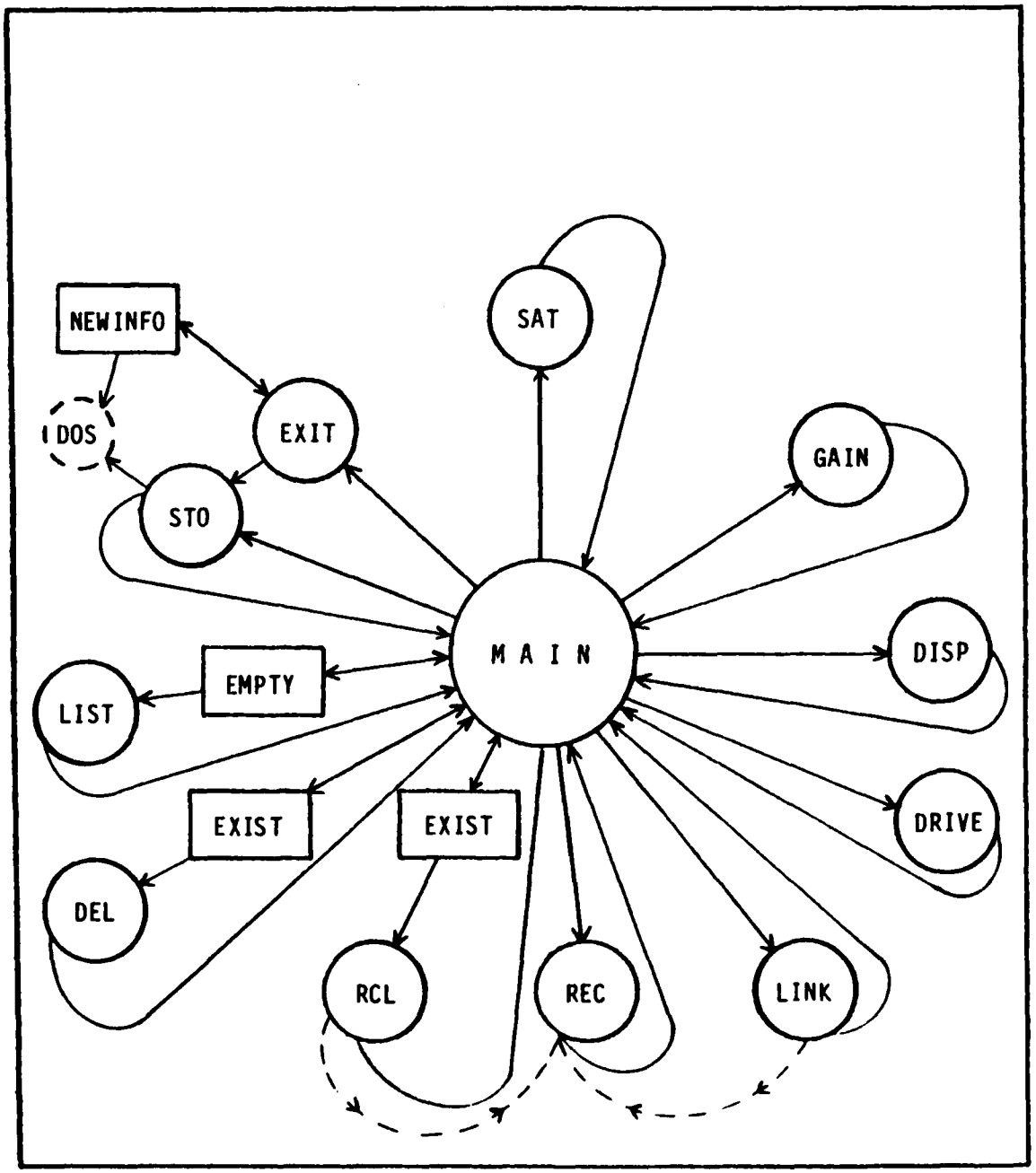


Figure 5-1. System Flow of Control

The states Sat, Gain, Disp, and Drive correspond respectively to four selections from the main menu: satellite parameters, antenna gain patterns, display link budget, and change default drive. These states are all characterized by the fact that they are entered only from MAIN, and once exited, control again returns to MAIN.

All of the remaining states also possess this control path, but are also affected by other states or external conditions (such as program flags). For example, when the the state LINK is entered, it provides an external input to RECOMP indicating that link analysis has just been performed. Likewise, when the state RCL is entered, it provides an external input to RECOMP indicating that a link file has been recalled from disk. The purpose of this arrangement is to disallow recomputing a link budget unless a link budget has either been just computed or just recalled from disk. This prevents the fatal errors which would occur if the recompute module were directed to recompute a link file which consisted only of garbage data. Multiple recomputing of the same file is allowed. Note that the external condition EXIST (not a state) will not allow control to pass to RCL unless the file designated for recall does in fact exist on the default drive. Similarly, control cannot pass to DEL unless the file designated for deletion exists on the default drive.

The state LIST cannot be entered unless the link file directory is non-empty. This prevents the display of a blank screen in the case of an empty directory.

The remaining states are closely intertwined. The state STO has the usual control path to and from MAIN. The state EXIT can always be entered from MAIN; however, control does not return to DOS unless either there is no unsaved link data, or there is unsaved link data but the user has given permission to discard it. If neither of these two conditions exists, then control passes to STO so that the data can be stored on disk, and then to DOS. This arrangement eliminates the possibility of exiting the system without saving important data.

Database Management. The system database is relatively simple, and consists of three basic filetypes: link directory, buffer, and link file. In the current system there is one link directory, one buffer, and numerous link files. The link directory is designated LINK.DIR, and contains the names of the link files currently stored on disk. The buffer file TEMP.STO is created only during the file deletion process, and is used to temporarily hold directory entries. Each link file contains all the data for one link budget, and may be given any name from one to eight characters long. All of the above files reside on the default drive under the directory path drive:data\[filename]. Figure 5-2 graphically shows the system database structure.

When a link file is to be stored under a given filename, the system first checks to see if there is already a link file on disk with the same filename. If so, a warning

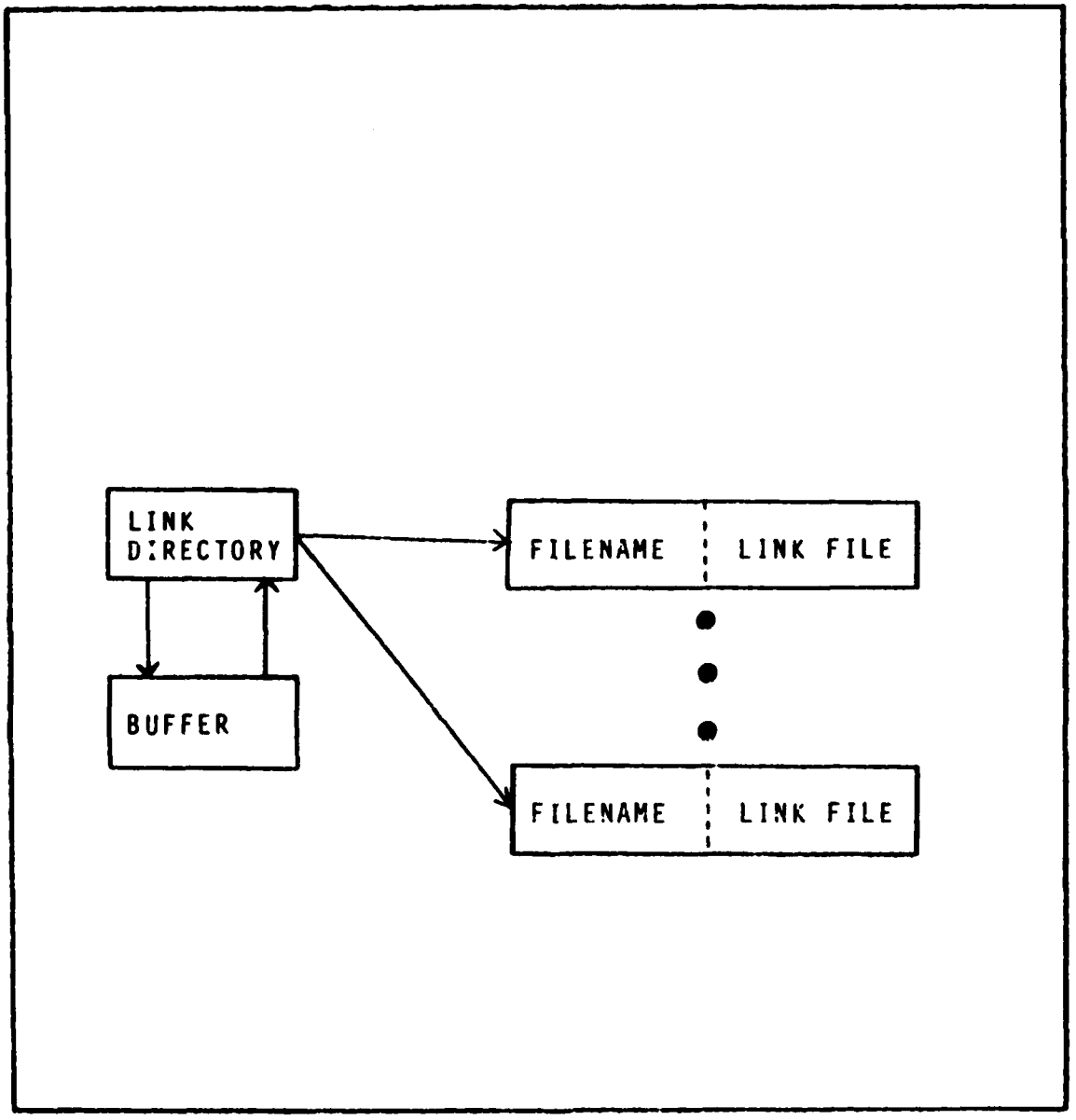


Figure 5-2. System Database Structure

notice is issued and the operation is disallowed. Otherwise the file is written to disk with the extension .LNK, and an appropriate entry is made in the directory.

When a link file is to be recalled from disk, the system first checks to see if the requested file exists on disk. If not, a warning notice is issued and the operation is again disallowed in order to prevent a disk I/O error. If it does exist, then it is opened and read. The user is encouraged to check the directory for the existence of a file before trying to recall it.

When an unwanted link file is to be deleted from disk, the same check for existence is performed as above. If the file does indeed exist, the user is asked to confirm the decision to delete. If the user responds in the affirmative, then the designated link file is erased from disk, and the corresponding entry is removed from the directory. However, since the actual location of the deleted filename in the directory is unknown, the directory LINK.DIR is first copied into the buffer TEMP.STO. Next, LINK.DIR is purged of all records. Finally, all records in TEMP.STO with the exception of the one being deleted are written back into LINK.DIR.

When the directory is to be listed, a check is made to determine whether there is anything in LINK.DIR to display. If not, an appropriate message is printed and no action is taken. If LINK.DIR is non-empty, then it is simply listed record by record.

Changing the default drive merely changes the path to the link data from olddrive:data\[filename] to newdrive:data\[filename]. This permits multiple data disks to be used at the same time.

Software Design

The purpose of this section is not to derive or verify equations used in the calculations, but rather to explain in a somewhat general sense the algorithms, engineering principles, and assumptions involved in the design of each functional module. Flowcharting every module and discussing every line of source code does little to further the user's understanding of the system. Hence, the software is discussed more qualitatively than analytically. For the purposes of this discussion, all software is grouped into one of six packages: utility, math, graphics, low-level functional, high-level functional, and main program. For further detail, the reader should refer to the software listings in Appendix D.

Utility Package. The utility package is a set of small modules which perform miscellaneous program tasks. These modules are described below:

CUSTOM - Contains the procedures CUSTOMPLOT and CUSTOMDRAW, which in turn use the standard Pascal procedures PLOT and DRAW to plot points and draw lines in graphics mode. If the system is transported to another machine which has different graphics protocols, CUSTOM can be rewritten to accommodate the new machine. All

graphics routines use CUSTOM, so modifying it will automatically modify all the graphics routines.

EXIST - Function which checks to see whether a specified file exists on disk. It works by turning on the $I- compiler directive (which activates I/O error checking) and then attempting to reset the file. If the file does not exist, this operation will result in an I/O error. The result of this test is returned to the calling routine by a boolean variable.

GETALT - This module simply prompts the user for a value of satellite altitude. The default value "G" is geosynchronous altitude. This particular routine is made into a module because it is called by many other modules (it is essentially a subroutine).

INTEGER - Polls the user for integer numeric data. If the user does not enter a legal integer, then an error flag is set which can be detected by the calling module.

NOISE - Uses the Pascal procedure SOUND to make the computer's speaker emanate a high-pitched tone. It is used where an audible warning signal is desired.

REAL - Similar to INTEGER, only this module polls the user for real numeric data.

SAVEIT - Saves a specified file on disk. In order to prevent overwriting an existing file, the directory is first checked to see if the specified filename already exists. If so, a warning is issued, and control returns to the main menu. Otherwise, the current link file is

written to the default drive under the specified filename, with the extension ".LNK." This module is accessed in two places: once in the main "Store Link File" routine, and once at the end of the program in the case that the user has mistakenly chosen to exit without storing a needed link file.

WAIT - Suspends execution until a key is pressed. Control enters an infinite loop until the Pascal variable KEYPRESSED is true. This routine is used to temporarily halt program execution so that intermediate results may be displayed.

Math Package. The math package is a set of modules which perform various basic mathematical functions. For more information, refer to any text of standard math tables. These modules are described below:

ALOG - Converts dB to numbers by raising 10 to the power of dB/10.

BESSEL - Computes first order Bessel functions of the first kind, $J_1(x)$. This is accomplished by evaluating the first 200 terms of the series expansion for $J_1(x)$ (as given in any handbook of standard math tables, such as the CRC). Because of the extreme magnitude of some of the terms in the series, an overflow condition will result if they are computed directly. To circumvent this, each term is broken up into smaller pieces which by themselves cannot exceed the capacity of the machine; these pieces are then combined to complete the series.

If more than two consecutive terms do not contribute anything to the series (i.e. they go to zero in the limit) then the series is truncated before the full 200 terms.

LOG10 - Function to compute base ten logarithms via natural logarithms.

LOG2 - Function to compute base two logarithms via natural logarithms.

PWR - Performs real exponentiation by using laws of logarithms. The cases of zero and negative arguments are taken into account.

Q - Computes Marcum's Q-function of the argument. If the argument is greater than 13 then the value zero is returned to avoid an overflow condition. If the argument is between 4.24 and 13, then the exponential approximation of the Q-function is used. Otherwise, the Q-function is computed by first calculating the error function of the argument, and then deriving the Q-function from that. This is done to facilitate the Simpson's rule numerical integration which computes the error function (16:656).

QINVERSE - Computes the inverse Q-function of the argument. The equation for the inverse Q-function is derived from the same exponential Q-function approximation used above. However, since this is a nonlinear equation, and it must be solved numerically. Initially, the method of halving the interval is used to

isolate the general region of the root. This method is predicated upon the fact that if the sign of a continuous function changes between any two points, then a root must exist within that interval; the region of sign change may then be isolated as closely as possible. Once this initial estimate of the root is made, the result is passed to a routine which uses Newton's method to more exactly determine the root. Newton's method extrapolates along a tangent from a point near the root to its intersection with the x-axis. This process is iterated until the tangent lines are sufficiently close together; the final point of x-axis intersection is the root.

TRIG - Computes some simple trigonometric functions not included in the standard Pascal library. The tangent function is computed by dividing the sine function by the cosine function. The inverse sine and inverse cosine functions are computed by evaluating simple formulas involving the inverse tangent function.

Graphics Package. The graphics package is a set of four modules which plot antenna gain patterns. This section does not delve deeply into electromagnetic theory, but rather explains the fundamentals of how these plotting routines work. All of the plotting routines employ the utility routine CUSTOM to perform actual plotting. These modules are described below:

HORN - Plots the gain pattern for a square horn antenna. The gain function for a horn antenna is given by

$G=[(\sin x)/x]^2$, where x=pi*sin(theta). This module allows theta to sweep from -pi/2 to pi/2 in increments of .Ø5, and calculates corresponding values for G. Since these are in terms of polar coordinates, they are converted to rectangular coordinates for the purposes of plotting on the screen. Appropriate offsets and scaling coefficients are also used to insure that the resultant plot fits on the screen as desired.

HELIX - Plots the gain pattern for helical antennas. The gain expression for these antennas is not quite as simple, and is derived via antenna theory. To do this, the helix is modeled mathematically as an array of circular current loops. Two quantities are derived, a pattern factor due to the circular loops, and an array factor due to the iteration of the loops. Then, using pattern multiplication techniques, the overall gain pattern expression is formed. This expression is plotted in a manner similar to the above.

DISH - Plots the gain pattern for parabolic reflector antennas. The gain function for parabolic reflectors is given by $G=(2J_1(x)/x)^2$, where x=pi*d*sin(theta)/L, L is wavelength, and d is dish diameter. Note that the gain function is similar to a sine-over-argument squared function, but instead uses a Bessel function. Consequently, DISH calls the module BESSEL to actually compute these values. The resultant gain function values are then plotted as before.

ARRAY - Plots the gain pattern for a planar phased array of dipoles. The derivation of this depends heavily upon array theory. This module first defines a number of functions to be used later on in the computations; these being the three rectangular direction cosines, the theta and phi components of the element factors corresponding to the three possible vector current directions, and the pattern factors corresponding to the seven possible current distributions. The actual values of these quantities depend on the user input. Next, the main pattern computation routine computes the array factors for the x and y directions, and multiplies them together along with the pattern factor and appropriate element factor to obtain an overall expression for the gain pattern. Finally, a plotting routine is used to actually display the computed gain patterns. Six polar plots are displayed: the theta=pi/2 plane, the phi=Ø plane, and the phi=pi/2 plane (both theta and phi components for each). The user also has the capability to magnify or shrink the resultant plots to any size desired.

Low-Level Functional Package. The modules in this set constitute the basis of the system, since they perform the bulk of the actual link analysis. These modules are described below:

AIR - Computes the total path attenuation due to atmospheric absorption. This attenuation is a function of two variables: elevation angle and frequency. The

computations in this module are based upon a mathematical model of atmospheric attenuation. For detailed information on this model, the user is referred to Appendix B.

AZEL - Computes the azimuth and elevation angles from two specified ground sites to a satellite at a given position. The user is required to enter the latitude and longitude of the uplink site, downlink site, and sub-satellite point; as well as the satellite's altitude. The azimuth is computed in a straightforward manner using basic spherical trigonometry. The elevation angle is derived via simple plane trigonometry from the tilt angle, which is the angle between a line from the satellite to the ground site, and a normal from the satellite to the ground. The altitude is normalized with respect to the Earth's radius, in effect making the Earth a unit circle. The tilt angle follows from basic plane and spherical trigonometry (refer to Chapter II).

BLACKBOD - Computes the steady-state temperature of a satellite given its altitude, total area, and fraction of total area exposed to the sun. The value returned is only a crude estimate, and neglects such factors as rotation of the satellite, internal heat sources, and surface material considerations. The module models the sun as a perfect blackbody, and calculates the total flux at the satellite's altitude. The steady-state temperature is found by setting the flux gained equal to the flux lost.

CAPACITY - Computes the theoretical Shannon rate and Nyquist rate for the specified channel, and determines whether one of these rates is limiting system performance with the parameters specified. The Shannon and Nyquist rates are given by

$$R_{shannon} = B\log_2(1 + P/(BN_o)) \quad (38)$$

$$R_{nyquist} = 2B\log_2 M \quad (39)$$

where R is channel capacity (bits/sec), B is bandwidth (Hz), P is received power (W), N_o is the single-sided noise power spectral density (W/Hz), and M is the number of signalling levels (16:7). If one of these quantities is less than the previously computed bit rate, then an appropriate flag is set.

CIRCULAR - Computes the velocity, orbital period, coverage area, and duration of visibility for a satellite in circular Earth orbit. Formulas for the first two quantities may be found in any orbital mechanics text. The last two quantities are derived via simple trigonometry. For more detail, refer to Chapter II.

ELLIPSE - Computes the velocities at apogee and perigee, average velocity, velocity at a specified orbital point, eccentricity, orbital period, and coverage area for a satellite in elliptical Earth orbit. Formulas for these quantities follow from basic trigonometry and analytic geometry, and may be found in any orbital mechanics text. For more detail, refer to Chapter II.

ESCAPE - Computes escape velocity at a given altitude. The formula for escape velocity may be found in any physics or orbital mechanics text. For more detail, refer to Chapter II.

FUNCTION - Contains a number of link analysis functions which are called by several other modules. FUNCTION contains functions to compute gains and 3-dB beamwidths for four different antenna types, effective isotropic radiated power (EIRP), system equivalent temperature, G/T figure of merit, received useful power, carrier power-to-noise power density ratio, available and required bit energy-to-noise power density ratios, bit error rate, and maximum data rate. For more information on the computation of these quantities, refer to Chapter II.

RAIN - Computes the attenuation due to rainfall in the channel. Like AIR, this module relies upon a mathematical model of rainfall attenuation as a function of rain rate, frequency, and elevation angle. For detailed information, the reader is referred to Appendix C.

High-Level Functional Package. The modules in this package do not perform link calculations per se, but rather organize user input and call the appropriate low-level modules. The high-level system modules are discussed below:

ANTENNA1 - Allows the user to select the antenna for uplink transmitting or downlink receiving at the corresponding ground site. The user may choose a horn,

helix, parabolic reflector, or phased array of dipoles. The module then computes the gain and beamwidth for the particular antenna chosen. These quantities are computed by calling the appropriate routine in the module FUNCTION.

ANTENNA2 - Allows the user to select the antenna for uplink receiving or downlink transmitting on the satellite. These antennas are currently restricted to be parabolic reflectors. However, the user may specify Earth coverage or spot coverage. The gain and beamwidth are computed as before, along with the gain and beamwidth required to achieve the specified coverage format. The user is alerted if either the actual gain or actual beamwidth is insufficient to achieve 25-dB sidelobes.

REC - Prompts the user for noise figure, receiver circuit losses, and receiving antenna temperature; and then calls appropriate routines in FUNCTION to compute the system equivalent temperature and G/T figure of merit.

XMIT - Prompts the user for output power and transmitter circuit losses, and then calls an appropriate routine in FUNCTION to compute the effective isotropic radiated power.

POWER - Organizes all the input necessary to compute the received useful power and CNR. First, the user enters the rain rate, and RAINATTEN is invoked to compute the attenuation due to rainfall. AIRATTEN is also invoked to compute the atmospheric attenuation. Next, the slant

range from ground site to satellite is computed, and the total path loss is determined. FUNCTION is then invoked to compute P_r and CNR.

BITRATE - Organizes all the input necessary to compute available E_b/N_o, required E_b/N_o, margin, and maximum data rate. The user enters the desired data rate (which may or may not be achievable), the type of modulation to be used, and the desired bit error rate. FUNCTION is then invoked to compute available E_b/N_o, required E_b/N_o, and max data rate; the margin is computed by subtracting the required E_b/N_o from the available E_b/N_o. The module CAPACITY is then called to determine whether the Nyquist or Shannon criteria will preclude transmission at the desired rate. If so, the system will not allow the user to use a bit rate which exceeds the capacity of the channel, and either the Nyquist rate or the Shannon rate is used instead.

Main Program The main program is highly modularized to facilitate integration of the functional modules. Of particular interest in the type definition section is the variable type 'link', which is the type assigned to all variables in the link files. It consists of a two-part record: one for uplink and one for downlink. In addition, all link file variables are assigned to a single record, so that an entire link file may be accessed by a single variable. All of the functional modules are included for compilation immediately before the program body by using the

$I compiler directive. Next, there is an initialization section where various flags and strings are initialized and files are assigned. The main menu is then displayed, and the user is prompted for a menu item. The remainder of the program consists of 11 segments which correspond to the 11 main menu selections. Following completion of a segment, control always returns to the main menu unless the EXIT option is selected.

It is neither practical nor desirable to analyze each and every line of code in a document of this nature. Instead, the underlying basis of operation of each module was presented qualitatively. This approach was chosen to give the user a better appreciation for the lower-level design of the system.

VI. TESTING AND EVALUATION

This chapter addresses the general procedures followed during system testing. It is not the purpose of this chapter to document the exact testing procedure for every minor program error detected, but rather to give the reader an understanding of the overall testing process, and to explain how major program flaws were discovered. The end product is also evaluated qualitatively according to the criteria specified in the system requirements.

Code Testing

This section addresses the process of software testing, starting with an explanation of the overall testing strategy. Module testing and higher-order testing are then discussed separately.

Testing Strategy. The system requirements chapter discussed in detail the various types of test case formulations, as well as the specific activities performed during module and higher-order testing. This section does not reiterate this material, but instead discusses the overall testing strategy used to validate the code.

In all cases, black-box testing was performed prior to white-box testing. In addition, by carefully choosing both the type of black-box tests and the order in which modules were tested, it was possible to greatly simplify the testing process. In most cases, it was unnecessary to perform white-box testing after black-box testing, because the process of

black-box testing itself satisfied all of the conditions of logic coverage. In particular, exercising the input screening routines (a type of black-box test) generally guaranteed statement, decision, and condition coverage. The main program (treated as a separate module during testing) was an exception to this rule.

The functional packages were tested in the following order: utility, math, graphics, low-level, high-level, and main program. However, the modules within each package were tested non-incrementally. This, in effect, is a two-tiered testing mechanism. This procedure allowed the modules within each package to be tested independently, with the aid of small driver modules. The only exception to this was in the math package, where PWR had to be tested before ALOG because the latter calls the former.

The main program itself was tested using both black- and white-box methods. In addition, as new modules were completed, it was tested incrementally in a top-down fashion. This was an ongoing process throughout program development.

Lastly, function testing and system testing were performed to insure that the final product satisfied all of its intended objectives.

Module Testing. Module testing was performed in six stages, corresponding to the five functional packages plus the main program. The utility package was tested first. Because these modules are so simple, testing formalism

was dispensed with, and they were tested simply by running them to see if they worked. No errors were found here.

In the math package, PWR and ALOG were tested in order for the reasons mentioned above. All modules were tested using boundary-value analysis and equivalence partitioning. Overflow errors occurred in BESSEL due to large floating point values in the series representation. This problem could have been alleviated by drastically reducing the number of terms in the series, but this would have reduced accuracy. The solution finally decided upon was to break both the numerator and denominator of each term into smaller pieces, and then perform arithmetic on these pieces in such a way as to avoid an overflow condition. The modules LOG1Ø, LOG2, Q, and QINVERSE all fail when given a negative argument; however, input screening prevents this situation from occurring. Inaccuracies in the resulting output by Q led to the discovery of two errors in the numerical integration routine. First, a term was missing from the Simpson's rule approximation. Second, the curve being integrated was not necessarily partitioned into an even number of panels, as required by Simpson's rule. Finally, boundary-value analysis discovered that the inverse cosine function in TRIG produced a division by zero for an argument of zero, as did the inverse sine function for an argument of one. All these errors were easy to correct.

The graphics package was tested next, since the modules within it call a number of the modules in the two previously

tested packages. The plots made by HORN and DISH do not vary with the input due to the ambiguities that would result if the plots were scaled to fit within the screen. In HELIX, stray lines unrelated to the gain pattern would always appear before plotting began. This module was modified to suppress these lines; however, for some combinations of input (usually near the range limits), a few points near the x-axis may be omitted. Lastly, when the object code exceeded the 64K limit set by Pascal, a memory overflow condition occurred. This problem was remedied by converting the module ARRAY into an independent program to be compiled separately from the rest of the system. It is invoked in the main program with the 'chain' command.

Next, the low-level functional package was tested. Because the modules in this package are so simple in design, very few errors were found. There were a few problems encountered with CIRCULAR and ELLIPSE in particular, due to the fact that the sine and cosine functions expect the argument to be in radians, and degrees were sometimes mistakenly used. Slight inaccuracies in AIR and RAIN were traced to some interpolating polynomials not being normalized properly. Finally, FUNCTION contained a litany of errors due to inconsistent units (e.g. bandwidth is specified in MHz, while frequency is in GHz, etc.).

Next, the six high-level functional modules were tested. No problems were found in REC, XMIT, or BITRATE. ANTENNA1 and ANTENNA2 presented some difficulties. Both modules

initially had problems correctly storing the input parameters associated with a given antenna. Because there are four different antenna types, each with a different set of input parameters, the modules had to first correctly identify the antenna type, and then store the input parameters in the correct order. ANTENNA2 also contained a ubiquitous logical error. While it was able to correctly compute the gain and beamwidth required for global or spot coverage, there was no reason to assume that the satellite antenna was capable of achieving the desired performance. Lastly, it was found that POWER failed when the rain rate was identically zero. The above anomalies were easily remedied.

The last module to be tested was the main program itself. Both black- and white-box testing methods were used. As new modules were completed and tested, they were incrementally added to the main program, which was then tested in a top-down fashion. Of course numerous minor errors were found and corrected, but by far the most significant problem was the implementation of the link budget recomputation function. The difficulty arose from the fact that the modules ANTENNA1 and ANTENNA2 prompt the user for antenna type, then for various antenna parameters, and then compute the gain and beamwidth. However, when a recomputation is performed, the antenna type has already been chosen, and the user is just changing its original parameters. This means that the recomputation software has

to know the original antenna type, where the original parameters are stored, and whether it is to work on the uplink or on the downlink. It took a great deal of restructuring before a suitable scheme for accomplishing this was devised. Finally, it was discovered that the order in which the inclusion files are listed at the beginning of the program is crucial, since this governs the order of compilation. Consequently, modules must always be listed before any other modules which call it.

Higher-Order Testing. Two types of higher-order testing were performed: function testing and system testing. This type of testing is much less rigorous than module testing, and tends to be somewhat subjective. The purpose of function testing is simply to determine how well the end product satisfies the original program requirements. The program does, in fact, satisfy the primary, secondary, and housekeeping functional requirements as specified in Chapter III.

System testing, on the other hand, is designed to qualitatively compare the product to its original objectives. First, every facility mentioned as a program objective was indeed implemented. Second, the use of menus, display of intermediate results, and color displays greatly enhanced the usability of the system. Lastly, the high degree of modularity (especially the graphics driver) assures compatibility on other MS-DOS machines. Thus, adherence to the original objectives is very close.

Code Evaluation

This section comments on the merits of the end product from the user's perspective, as opposed to the programmer's perspective. Since the concept of usability has already been addressed, two additional aspects of the code are discussed: accuracy and maintainability.

Accuracy. One can group all of the modules which return numerical results into one of three categories: those which rely upon a formula for generating the result, those which rely upon numerical methods, and those which rely upon a mathematical model. Most modules fall into the first category, and if the formula is correct (as verified by testing), there is really very little room for error.

The modules BESSEL, Q, and QINVERSE comprise the second category. BESSEL evaluates an infinite series, so its accuracy depends upon the number of terms included. Q performs a Simpson's rule numerical integration, and its accuracy depends upon the number of panels that the integrand function is partitioned into. QINVERSE solves a non-linear equation, and its accuracy depends upon the value of the error tolerance specified in the code. It should be noted that this non-linear equation is based on the exponential approximation to the Q-function, and thus has inherent inaccuracies.

The modules AIR and RAIN fall into the last category. Both of these modules suffer from the inevitable errors that result from ill-fitting interpolating polynomials. While

the vast majority of data points could be connected by a polynomial, in several cases the concavity/convexity of the curve was improper. In addition, both models make simplifying assumptions (see Appendices B and C) which limit their accuracy somewhat.

Maintainability. A significant percentage of total software cost is due not to development, but to maintenance. For this reason, software must be intentionally designed to be maintainable. The high degree of modularity of the system greatly facilitates expansion and maintenance. If a new function is to be added, then the modules needed to implement it are written and tested separately, and then included for compilation in the main program (taking care to place them in the proper order). A stub is then inserted in the proper place in the main program. Testing with the stub in place assures that the overall structure is still sound. Finally, the stub is replaced by the code required to call the previously written inclusion modules. In this manner, the system can be expanded continuously without having to alter the system architecture. The system modularity also makes it much easier to trace program bugs.

This chapter has addressed the issues of system testing and evaluation. Since the program itself is highly modular, an efficient testing process must also be modular. Validating a piece of software of this size requires a rigorous and thorough testing plan in order to assure program reliability.

VII. CONCLUSIONS AND RECOMMENDATIONS

The resultant end product of this research effort is a comprehensive, integrated, and transportable software package for the analysis of digital satellite links. While the primary objective of this software package is link budget analysis, the package also includes routines for computing various satellite orbital parameters, as well as for plotting antenna gain patterns. All of the nominal functions specified in the system requirements have been implemented.

System portability to other MS-DOS machines is facilitated by a removable graphics driver. Expandability is straightforward: a new function is written as a separate module and tested independently; it is then included for compilation in the main program via the $I compiler directive. In addition, the problem of tracing perceived errors is mitigated by the display of intermediate results following each major calculation.

The man-machine interface is designed to maximize the user's efficiency as well as to minimize confusion. Menus are used extensively to guide the user through the program, and the use of prompting and input validity checking insure the accuracy of all data entries. In addition, color displays are used to present the data in an organized, easy-to-read format.

The system has successfully withstood the rigors of both module testing and higher-order testing. It is highly

improbable that the user can cause the system to crash by entering inappropriate data.

This software package is far from complete. There are a number of recommendations for future modification and expansion:

1. Develop a mathematical model for calculating antenna noise temperature. The greatest challenge to this will be computing the sky noise temperature component, which is a function of both frequency and elevation angle.
2. Examine the effects of depolarization, ice crystals, multipath propagation, and atmospheric scintillation; and determine if they can be quantized and incorporated into the rain and air attenuation models.
3. Apply some of the results of Navas (6), so that link availability may be estimated based on local rain rate statistics.
4. Give the user the option to employ pulse shaping and error correcting codes.
5. Allow multiple accesses to the link, and introduce a noise term due to interference from other users.

The incorporation of these recommendations should make the package much more flexible, and therefore more applicable to real-world design problems.

APPENDIX A

User's Manual

While the program is highly modular and highly menu-driven, there are a few points regarding its operation which require amplification. It is not the purpose of this manual to teach link analysis, but rather to show the user how to use the program correctly and efficiently.

The host computer must be an IBM PC or equivalent compatible with at least one floppy disk drive. A dot matrix printer is also suggested for obtaining hardcopy of plots or other displays. The system disk contains 36 source files, which occupy 92K. When compiled, the object code occupies 63K. Since DOS requires an additional 27K, it is recommended that the host computer be equipped with at least 128K of memory. Also, the displays will only be readable on a color monitor.

If only one disk drive is available, then the system disk and the data disk will be one in the same. Otherwise, it is recommended that the data disk reside on a separate drive. This is especially useful on machines with Winchester-type hard drives, which can hold very large link analysis databases. The program initially assumes drive A to be the default data drive, until told otherwise by the user. At any rate, it is imperative that the path drive:data\ exists on the data disk, since this constitutes the directory under which all link files are stored. Failure to include this path on the data disk will result in a fatal I/O error.

To start-up the system, boot the operating system, and then place the system disk in drive A and type 'run.' The main menu will then appear (see Figure A-1). The default drive should be switched now if desired. A listing of all link files on the default data disk can be obtained by choosing menu option 9, 'List Link File Directory.' Unwanted files may be deleted by choosing menu option 8, 'Delete Link File,' and specifying the filename to be deleted without the .lnk extension.

Menu item 1, 'Satellite Orbital Parameters,' contains a submenu which permits the user to choose from computing escape velocity, satellite steady-state temperature, circular orbit parameters, elliptical orbit parameters, or azimuth and elevation angles. User input in this section is very straightforward. The circular orbit parameters include satellite velocity, orbital period, coverage area, and duration of visibility. The elliptical orbit parameters include satellite velocity at apogee and perigee, average velocity, velocity at a given reference angle phi (where phi=Ø at perigee and phi=pi at apogee), orbital period, and coverage area. The elevation angles computed in this section may be used later on during link analysis. This is the only connection to the rest of the program.

Main menu item 2, 'Antenna Gain Patterns,' also has no connection to the rest of the program. It contains a submenu from which the user may choose to plot the gain patterns of horn, helical, parabolic reflector, or phased

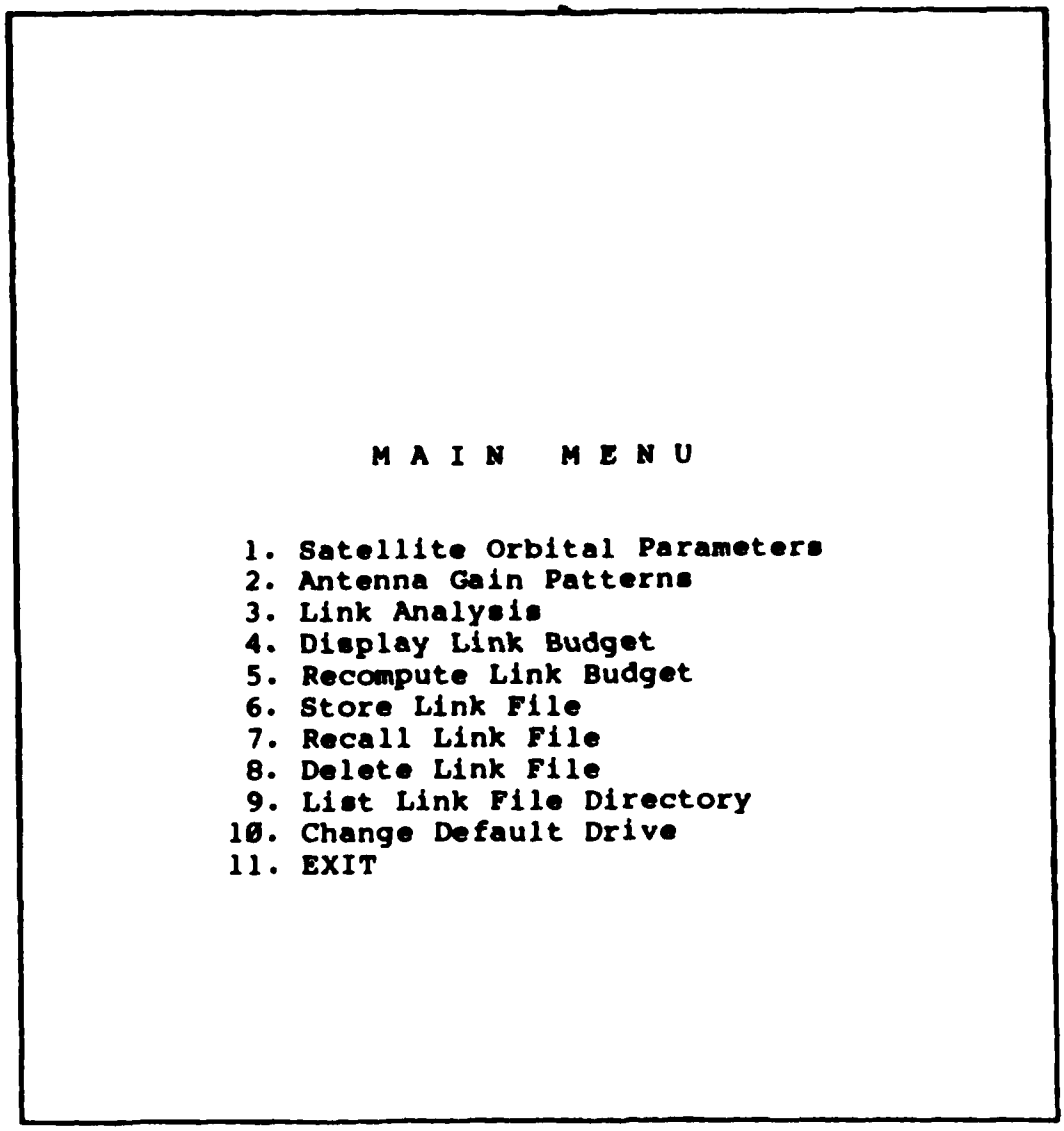


Figure A-1. Main Menu Display

array antennas. The first three selections require the user to enter the frequency and various items relating to the geometry of the particular antenna, and their use is also straightforward. The phased array selection, however, is considerably more complicated.

This selection plots the gain pattern for a planar phased array of identical dipoles. First, the user must enter the length of the individual dipole elements normalized to the wavelength. Next, the user must choose from one of seven current distributions on the dipoles. The desired array must then be decomposed into homogeneous subarrays in both the x- and y-directions. The program prompts the user for the number of x- and y-subarrays, the number of elements in each subarray, the spacing between those elements normalized to wavelength, and the phase taper between consecutive elements. For many cases, the separations and phase tapers between any two adjacent elements may well be identical; consequently, there will only be one subarray in each direction. Given this data, the program will generate six polar plots, those being the theta and phi components of the planes theta=pi/2 (xy-plane), phi=Ø (xz-plane), and phi=pi/2 (yz-plane). After each plot, the user is prompted for a magnification factor. Entering a value greater than unity will cause the same gain pattern to be plotted, only larger than before. The converse holds for values less than unity. Entering a value of zero will cause the screen to be cleared, and the next polar plot to be drawn.

Main menu item 3, 'Link Analysis,' is used to perform link budget analysis. The user is guided step by step through every calculation, and intermediate results are displayed along the way. The final link budget can be displayed by selecting menu item 4, 'Display Link Budget.' Menu item 5, 'Recompute Link Budget,' allows the user to recompute all the link calculations with new values of frequency, antenna design, transmitter power, receiver noise figure, bit error rate, or data rate. It also permits the user to perform the reverse process of computing bit error rate given the data rate. Once a desired link budget is obtained, the user may store it on the data disk by selecting menu item 6, 'Store Link File.' Previously stored link files may be recalled for review or recomputation by selecting menu item 7, 'Recall Link File.'

The following is an ordered listing of instructions for designing and analyzing a typical link, starting from the main menu.

1. Enter the frequencies for the uplink and downlink.
2. Enter the elevation angles and latitudes for the transmitting and receiving ground sites. Elevation angles previously computed in the satellite parameters section may be used again here.
3. Enter the mean sea level altitude of the transmitting and receiving ground sites.
4. Enter the satellite altitude (enter 'G' for geosynchronous altitude).

5. Enter the system bandwidth (The signal bandwidth is taken to be equal to the noise bandwidth).

6. Specify the type of antenna for the uplink transmitter.

7. Enter the appropriate antenna parameters asked for.

8. Specify the type of coverage for the uplink receiver, along with the antenna parameters asked for.

9. Repeat steps 6-8 for the downlink transmitter and receiver. The maximum gain and 3-dB beamwidth for each antenna is displayed as an intermediate result.

10. Enter the output amplifier power level and transmitter circuit losses for the transmitters at the first ground site and onboard the satellite. The effective isotropic radiated power is then calculated and displayed.

11. Enter the amplifier noise figure, circuit losses, and antenna noise temperature for the receivers at the second ground site and onboard the satellite. The effective system temperature and G/T figure of merit are then displayed.

12. Enter the rain rate at the uplink and downlink sites. If it is not known exactly, choose one of the qualitative descriptors that best matches the estimated rain rate.

13. Specify the desired data rate, modulation type, and bit error rate for the uplink and downlink signals. After a slight delay, the available E_b/N_o, required E_b/N_o, and maximum data rate will be displayed.

14. This is the end of user data entry. The link budget may now be displayed by selecting menu item 4, 'Display Link Budget.' The display consists of two columns: one for the uplink and one for the downlink. The display is also organized into color-coded groups for convenience. Cyan corresponds to general information, blue to the transmitter, yellow to the channel, green to the receiver, magenta to link performance criteria, and red to critical link constraints. If the computed data rate exceeds the maximum data rate, then it will blink. The same holds for negative power margins. This is to call attention to the fact that the link cannot exist as specified.

15. The user may now select menu item 5, 'Recompute Link Budget' in order to tailor the system to his needs.

16. The final link budget may be stored on disk for by selecting menu item 6, 'Store Link File.' Previously computed link files may be retrieved for review or additional computation by selecting menu item 7, 'Recall Link File.'

This manual should enable the user to efficiently use the link analysis program. Because all user input is screened, it is doubtful that erroneous input can cause the system to crash. In the unlikely event that this does occur, the user should restart the program. The only data potentially lost will be that data entered subsequent to the last 'Store Link File' command.

APPENDIX B

Mathematical Model for Determining Atmospheric Attenuation

At sufficiently high frequencies, atmospheric attenuation may significantly degrade a satellite link. This attenuation is due to absorption of electromagnetic waves by various constituent gases in the atmosphere. These interactions occur at the gas resonance frequencies. In the 1-100 GHz frequency band, the only significant contributors to atmospheric attenuation are water vapor and oxygen. The water vapor absorption band is centered at 22.235 GHz, while the oxygen absorption lines extend from 53.5 to 65.2 GHz. Satellite communications bands are generally chosen to avoid these frequencies, so resulting attenuation values are typically less than 1 dB (2:94-95; 9:323).

The total atmospheric attenuation depends upon a number of variables; namely frequency, elevation angle, ground station altitude, relative humidity, and surface temperature. Of these, only frequency and elevation angle play a significant role. Since the uncertainty of the link may be on the order of several dB anyway, it is perfectly acceptable to neglect the other variables. Hence, the model described below determines atmospheric attenuation as a function of frequency and elevation angle.

Figure B-1 is a logarithmic plot of atmospheric attenuation versus frequency for a vertical path over the 1-100 GHz range. The mathematical model for atmospheric attenuation was developed by fitting approximating

polynomials to various sections of this curve. The equations of the polynomials were obtained via a commercially available curve-fitting program. Thus, a quantitative relationship between attenuation and frequency was established. Because of the complexity of this curve, it was partitioned into four regions: 1-22 GHz, 22-40 GHz, 40-60 GHz, and 60-100 GHz.

Figure B-2 shows the first region. Note that both axes have logarithmic scales, even though attenuation is in units of dB. While taking the logarithm of a quantity already expressed in dB may not make sense intuitively, it was necessary in order to generate the curves properly. A single approximating polynomial joining all the data points in this region could not be found, so the region was decomposed into three subregions. The first two subregions correspond to areas of the curve where the relationship between attenuation and frequency is approximately linear, so two different linear regression lines are used to model this relationship. The last subregion corresponds to an area of approximately cubic data dependence, so a third-degree polynomial is used.

Figure B-3 shows the second region. The data points in this region display a good quadratic relationship, so a second-degree polynomial provides a good approximation.

Figure B-4 shows the third region. Several higher-degree polynomials were fit to this data, but only the second-degree polynomial provided an acceptable fit. Note that

there is some measurable error along the center of the curve.

Figure B-5 shows the fourth region. This region also had to be decomposed into three subregions, and proved to be the most difficult to model. Only two data points occupy the first subregion, since no way could be found to connect these points with the rest of the curve. These points are joined by a regression line. Similarly, the second subregion is also occupied by only two points. However, a quadratic polynomial is used to connect them in order to more closely approximate the true shape of the curve. The last subregion is modeled quite nicely by a fourth-degree polynomial.

The above discussion has explained in some detail how the mathematical model for vertical path atmospheric attenuation was developed. To include the elevation angle dependence, it is necessary only to multiply the vertical path attenuation by the cosecant of the elevation angle.

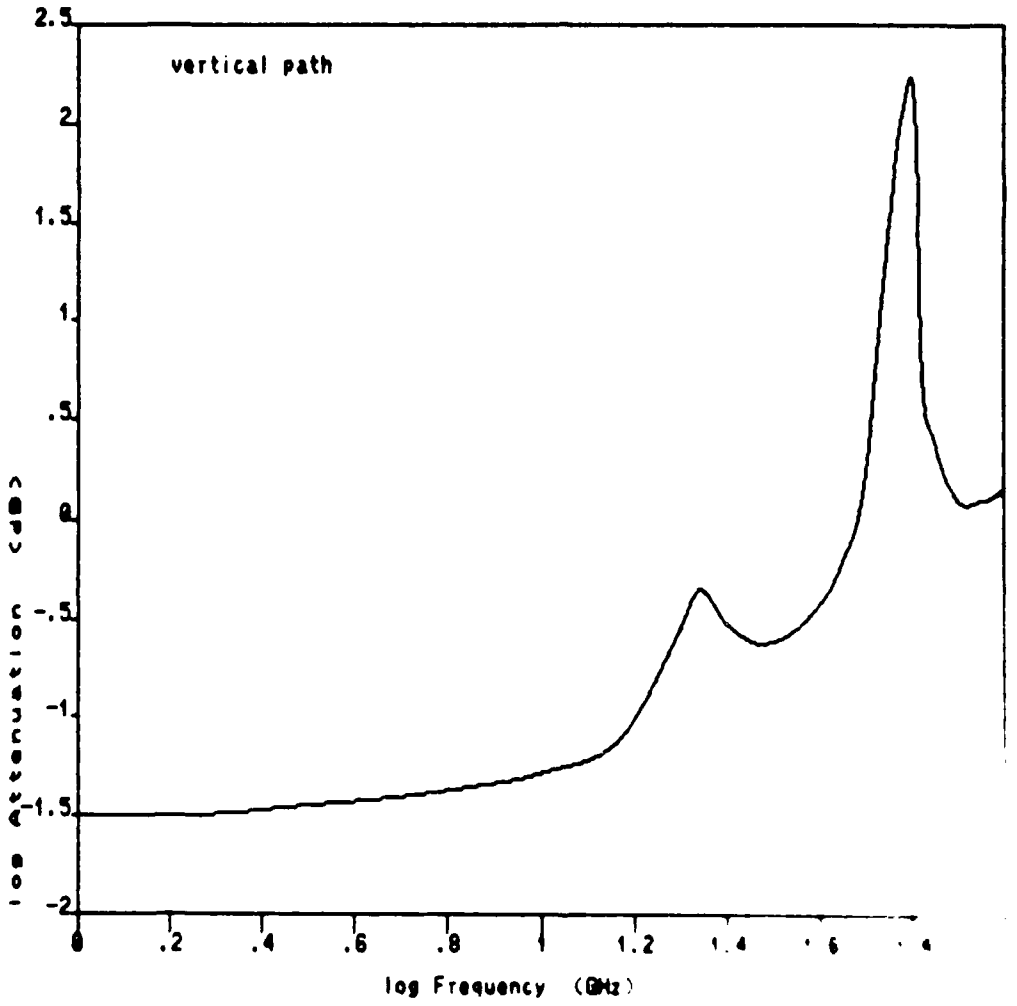
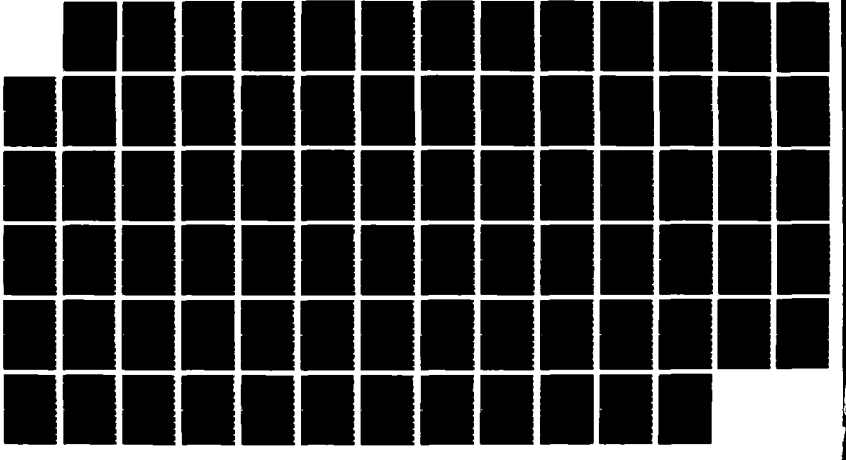
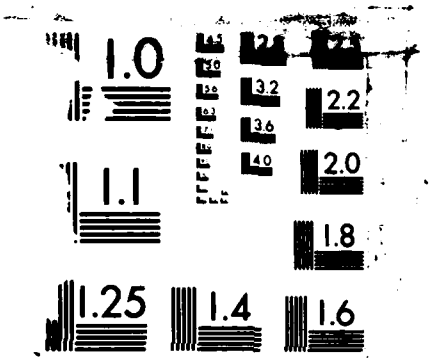


Figure B-1. Atmospheric Attenuation

AD-A178 872 DEVELOPMENT OF A MICROCOMPUTER-BASED WORKSTATION FOR ANALYSIS OF DIGITAL..(U) AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OH SCHOOL OF ENGI.. W C JACKSON 2/2

UNCLASSIFIED DEC 86 AFIT/GE/ENG/86D-38 F/G 17/2 NL





MIC
N

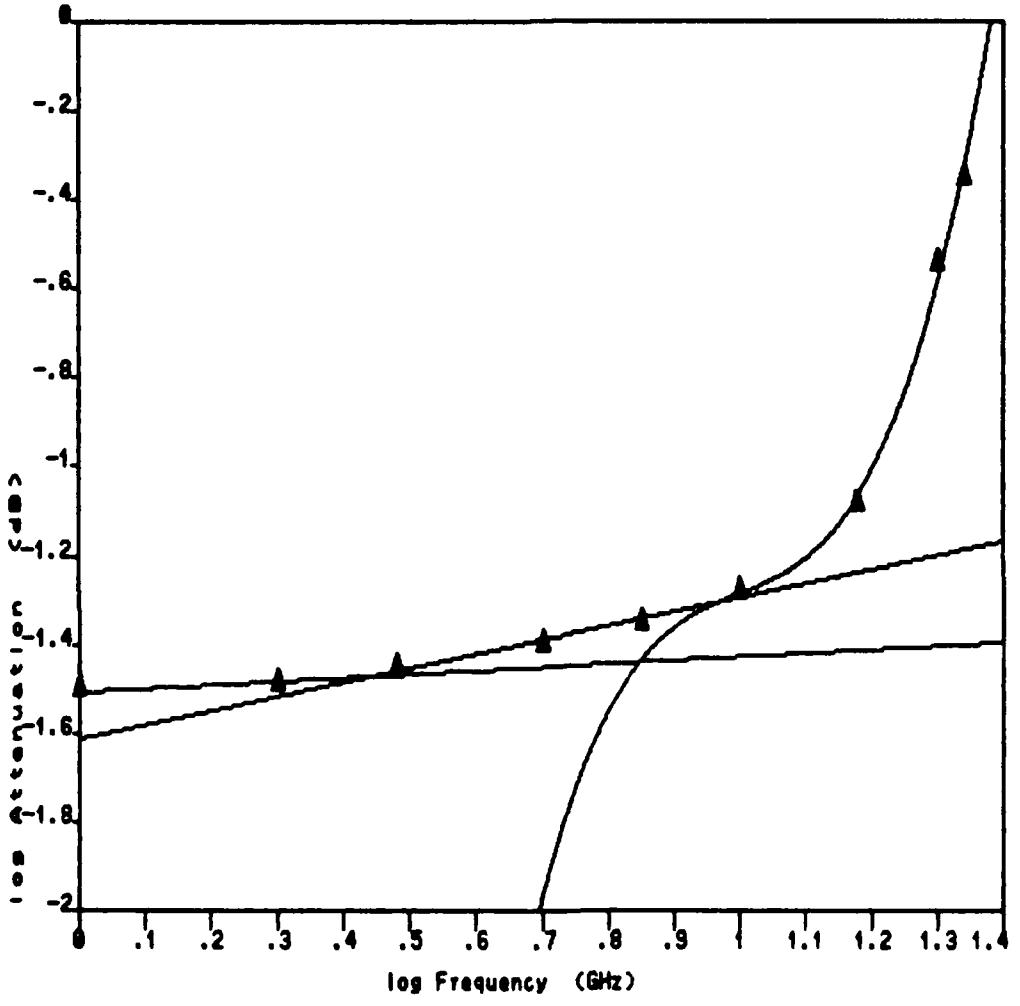


Figure B-2. First Region

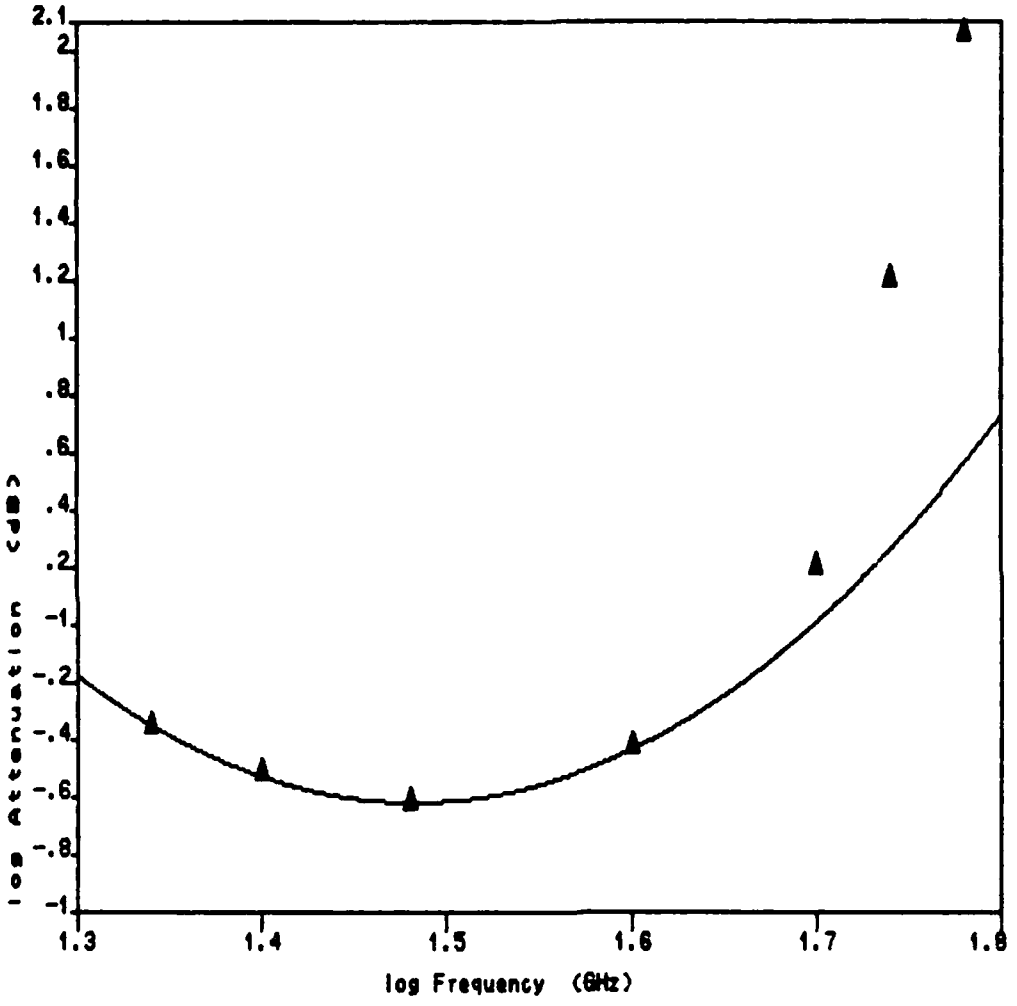


Figure B-3. Second Region

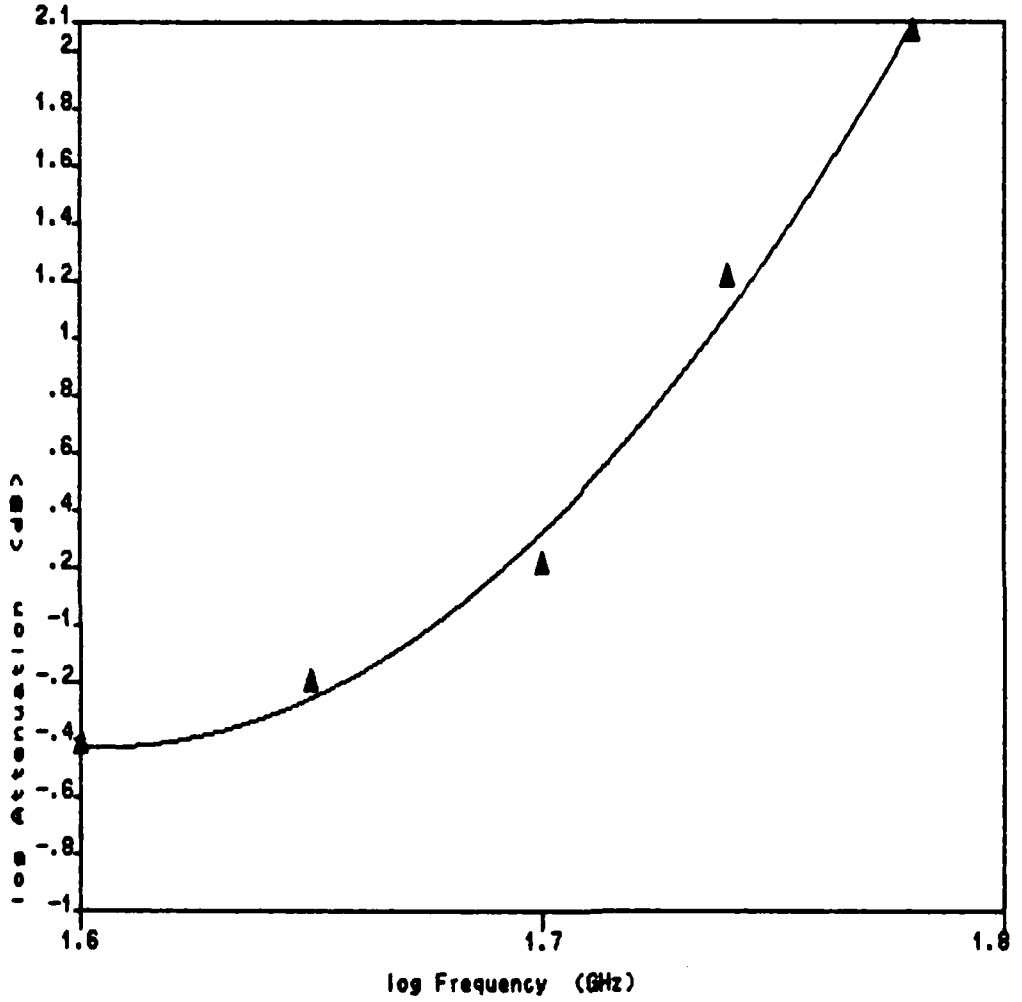


Figure B-4. Third Region

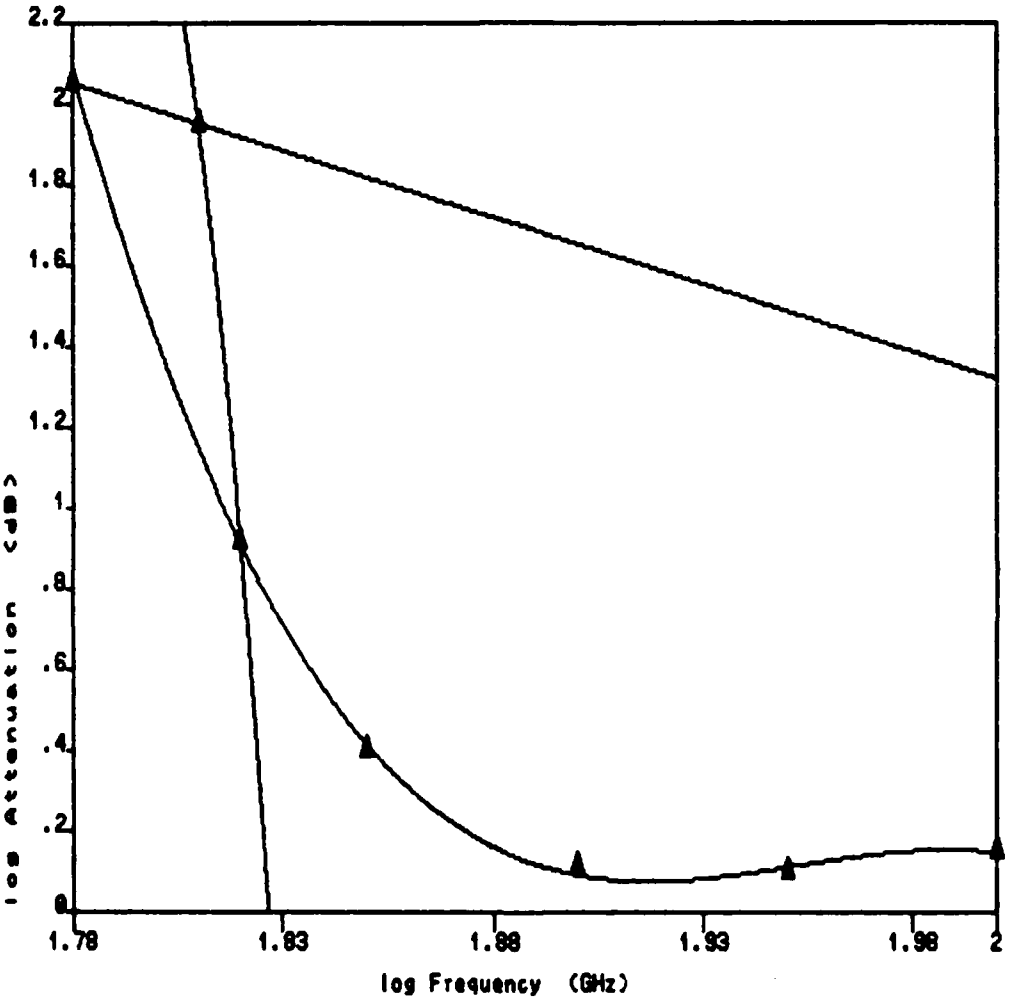


Figure B-5. Fourth Region

APPENDIX C

Mathematical Model for Determining Rain Attenuation

Rainfall is the most serious degrading factor in a satellite link. Rain droplets scatter and absorb impinging electromagnetic radiation, causing potentially severe signal attenuation, particularly at wavelengths approaching the size of the droplets. These interactions depend upon the number of droplets encountered (and thus upon the rain rate) as well as the geometry of the droplets.

There is a wide variety of rain attenuation models from which to choose. However, many of them rely heavily upon the knowledge of extensive local rain rate statistics or rain droplet distributions, which are unavailable to many users. The model presented here estimates rain attenuation as a function of frequency, elevation angle, rain rate, ground station altitude, and ground station latitude.

The attenuation A due to rainfall has been commonly given in the literature by

$$A = aR^bL \quad dB \qquad (C\text{-}1)$$

where a and b are frequency-dependent coefficients, R is rain rate in mm/hr, and L is effective path length. The quantity aR^b by itself is called the specific attenuation, and has units of dB/km. Note that the attenuation depends on the rate of rainfall, rather than on rainfall accumulation. The process of measuring rain rate is inexact at best, so the model allows the user to qualitatively

describe the rain intensity (light, medium, etc.), and then inserts a ballpark value. Hence, calculating attenuation involves two basic steps: determining the coefficients a and b, and then determining the effective path length (2:96-99; 9:327-335).

Approximations for the coefficients a and b are given in [7] as functions of frequency only. However, these coefficients are also dependent upon the microstructure of the rain, such as the rain temperature, droplet shape, and droplet size distribution. With a little extra work, it is possible to obtain a very accurate approximation. In [7], there is extensive tabular data for the coefficients a and b up to 1000 GHz, for rain temperatures of 20°C and 0°C, for several different droplet distributions. These droplet distributions include those of Laws and Parson, Marshall and Palmer, and Joss et al. While it is beyond the scope and intent of this report to present these distributions in detail, it should be mentioned that the Laws and Parsons distribution was chosen because it is a little more accurate than the others above 30 GHz. This distribution also yields different a and b values for low and high rain rates. As for rain temperature, while the assumption of 20°C gives suitable results for terrestrial links, 0°C is more appropriate for satellite links. All of the figures in this appendix were obtained by plotting the tabular data for the coefficients a and b (12:1466-1467).

Figure C-1 is a plot of a(f) versus frequency for low rain rates (i.e. less than 25 mm/hr), and is approximated by

a sixth-degree polynomial. Figure C-2 is simply a magnification of the previous figure for frequencies below 10 GHz, and is also modeled by a sixth-degree polynomial. Figure C-3 is a plot of a(f) versus frequency for high rain rates (i.e. greater than 25 mm/hr), and is approximated by a tenth-degree polynomial. Figure C-4 is also a magnification of the previous figure for frequencies below 10 GHz, and is given by a cubic.

Figure C-5 is a plot of b(f) versus frequency for low rain rates. Due to the nature of this particular curve, a single approximating polynomial could not be found. Hence, the curve was divided into two regions: 1-8 GHz, and 8-100 GHz. Figure C-6 shows the section of the curve from 1-8 GHz, which is approximated by a fifth-degree polynomial. Figure C-7 shows the section of the curve from 8-100 GHz, which is approximated by an eighth-degree polynomial. Figure C-8 is a plot of b(f) versus frequency for high rain rates. Due to the considerable complexity of this curve, it was broken-up into three regions: 1-6 GHz, 6-30 GHz, and 30-100 GHz. Figure C-9 shows the section of the curve from 1-6 GHz, which is approximated by a sixth-degree polynomial. Figure C-10 shows the section of the curve from 6-30 GHz, which is approximated by a fifth-degree polynomial. Lastly, Figure C-11 shows the section of the curve from 30-100 GHz, which is approximated by fourth-degree polynomial.

Once the coefficients a and b have been determined, the effective path length through the rain must be found. For a

given elevation angle El, the path length L through the rain is given by

$$L = (H_e - H_o)/\sin(El) \quad km \qquad (C\text{-}2)$$

where H_e is the effective storm height in km, and H_o is the ground station elevation in km.

The temperature gradient where $T=0^{\circ}C$ is called the zero-degree isotherm, and is generally the point above which no liquid water exists. The height of the zero-degree isotherm H_i, (also known as melting layer height), is related to the latitude by

$$\begin{aligned} H_i &= 4.8 \quad km && abs(lat) <= 30^{\circ} \\ &= 7.8 - 0.1*abs(lat) && abs(lat) > 30^{\circ} \end{aligned} \qquad (C\text{-}3)$$

Seasonal variations of the melting layer height have been neglected. The effective storm height is related to the melting layer height by

$$\begin{aligned} H_e &= H_i && R <= 10 \text{ mm/hr} \\ &= H_i + \log(R/10) && R > 10 \text{ mm/hr} \end{aligned} \qquad (C\text{-}4)$$

Note that C-4 accounts for the fact that higher rain rates are associated with more active convective rain cells, which contain strong updrafts that can carry water well above the zero-degree isotherm.

The above relations are generally accurate for relatively low rain rates (less than 10 mm/hr). However, high rain rates create another wrinkle in the problem. The same

convective rain cells which carry water above the zero-degree isotherm also render the distribution of rain non-uniform in the horizontal direction. Much data on this problem has been collected using rain gauge networks, but these experiments are beyond the scope of this report. The end result of this research was the derivation of a path profile model which takes the presence of convective rain cells into account. Putting all the pieces together, the total path attenuation A due to rain is given by

$$A = aR^{b}L \qquad R<=10 \text{ mm/hr} \qquad (C\text{-}5)$$

$$A = aR^{b}[1 - \exp\{-PL\}]/P \qquad R>10 \text{ mm/hr}$$

where $P = gb \ln(R/10)\cos El$
and $g = 1/22$

By specifying the rain rate as a value which occurs a given percentage of time, the resultant attenuation will also be expressed as a percent time occurrence. Likewise, if a link is designed to operate with a fixed amount of rain attenuation, then the link availability can be determined from the percentage of time that rain rate is exceeded (12:1466-1471).

The above discussion has presented the development of the rain attenuation model. It should be noted that this is only a crude model, and does not take into account such factors as ice crystals, fog, suspended particulates, and depolarization.

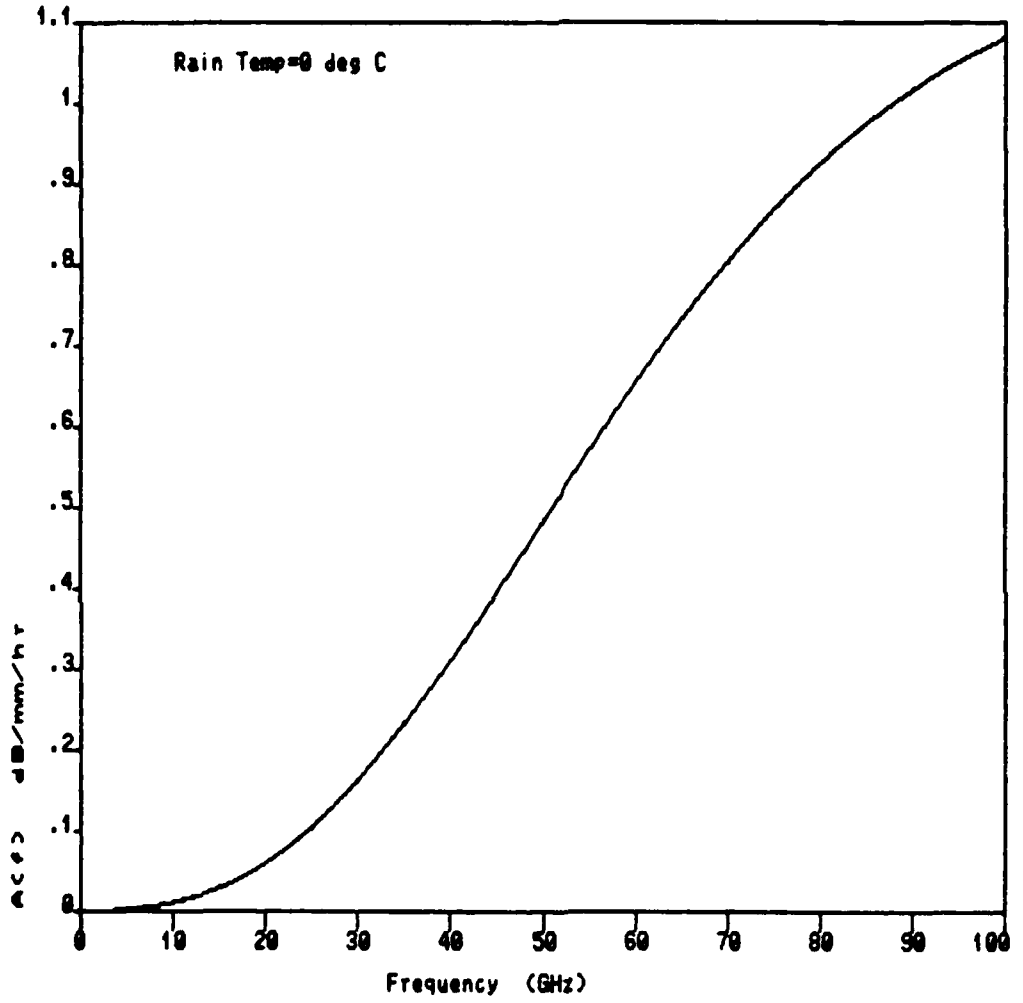


Figure C-1. A(f) vs. Frequency for Low Rain Rates (7:323)

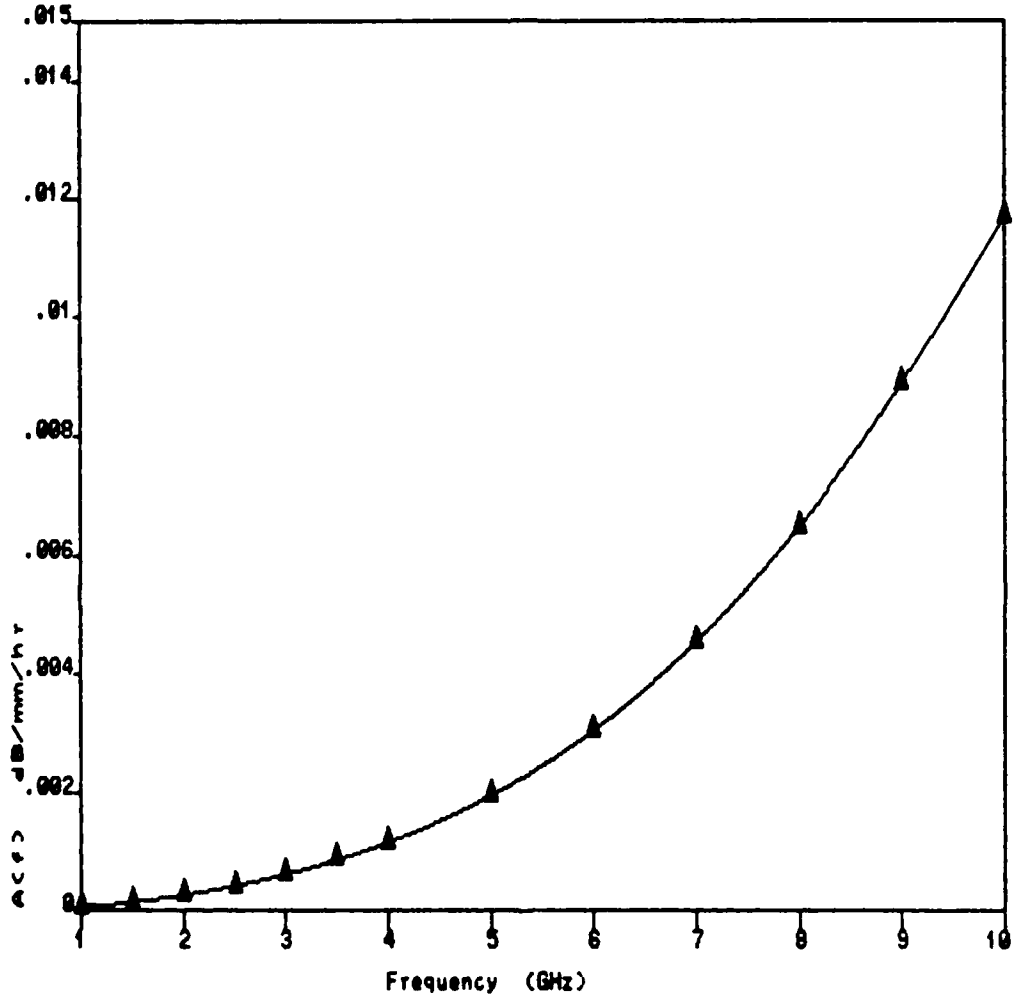


Figure C-2. Magnification of Frequencies Below 10 GHz (7:323)

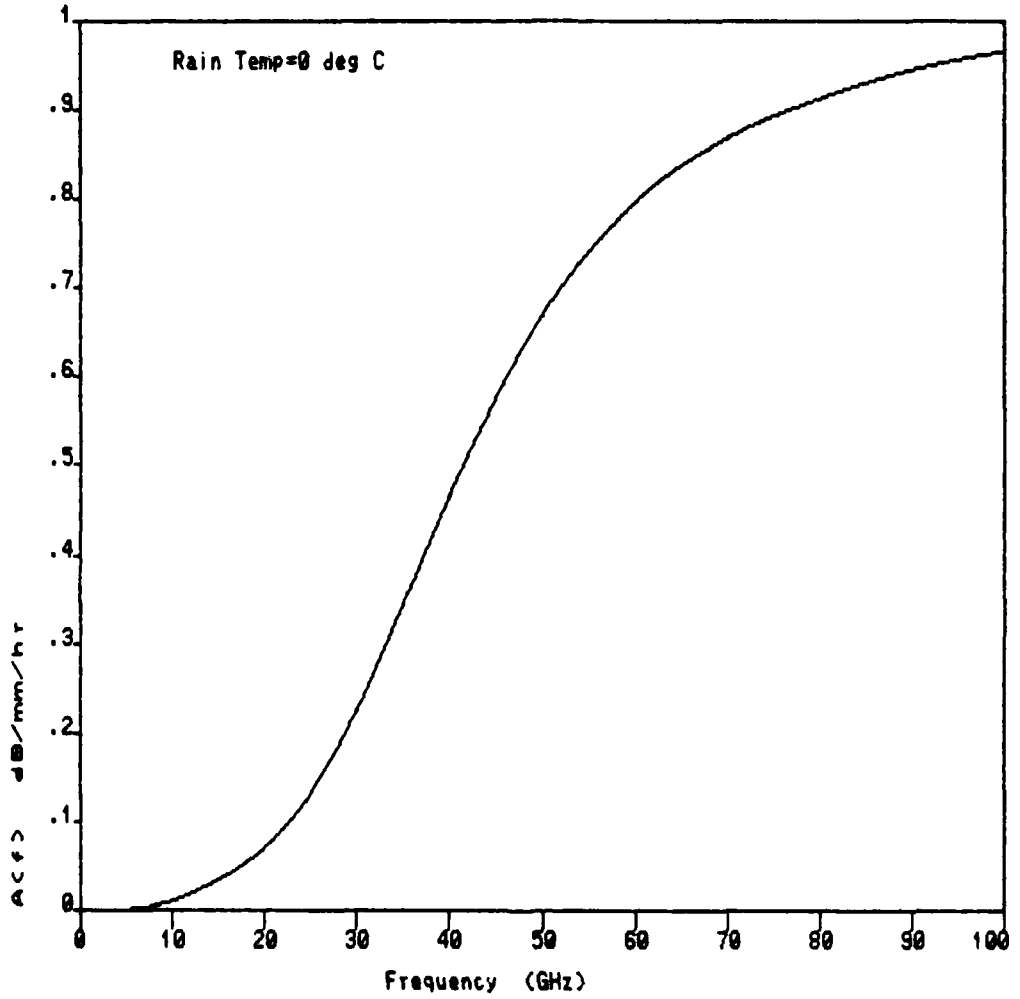


Figure C-3. A(f) vs. Frequency for High Rain Rates (7:323)

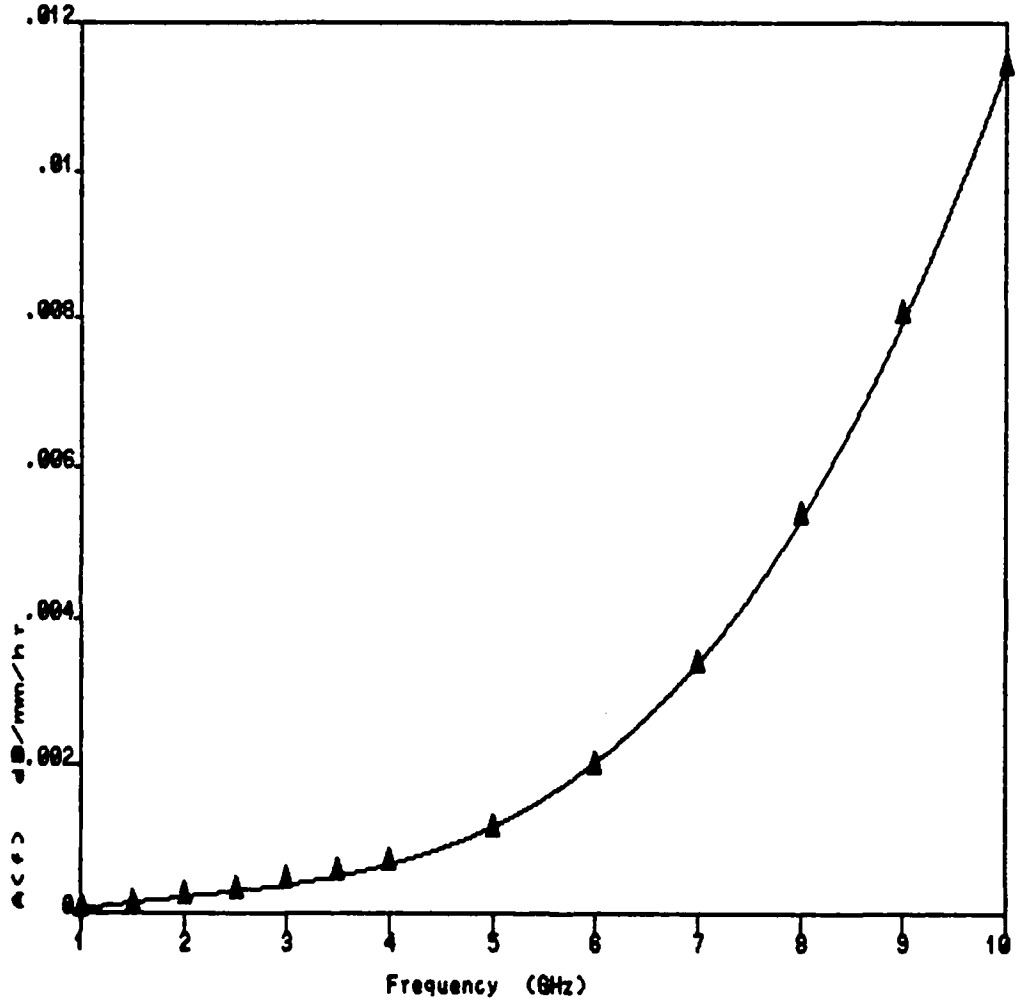


Figure C-4. Magnification of Frequencies Below 10 GHz (7:323)

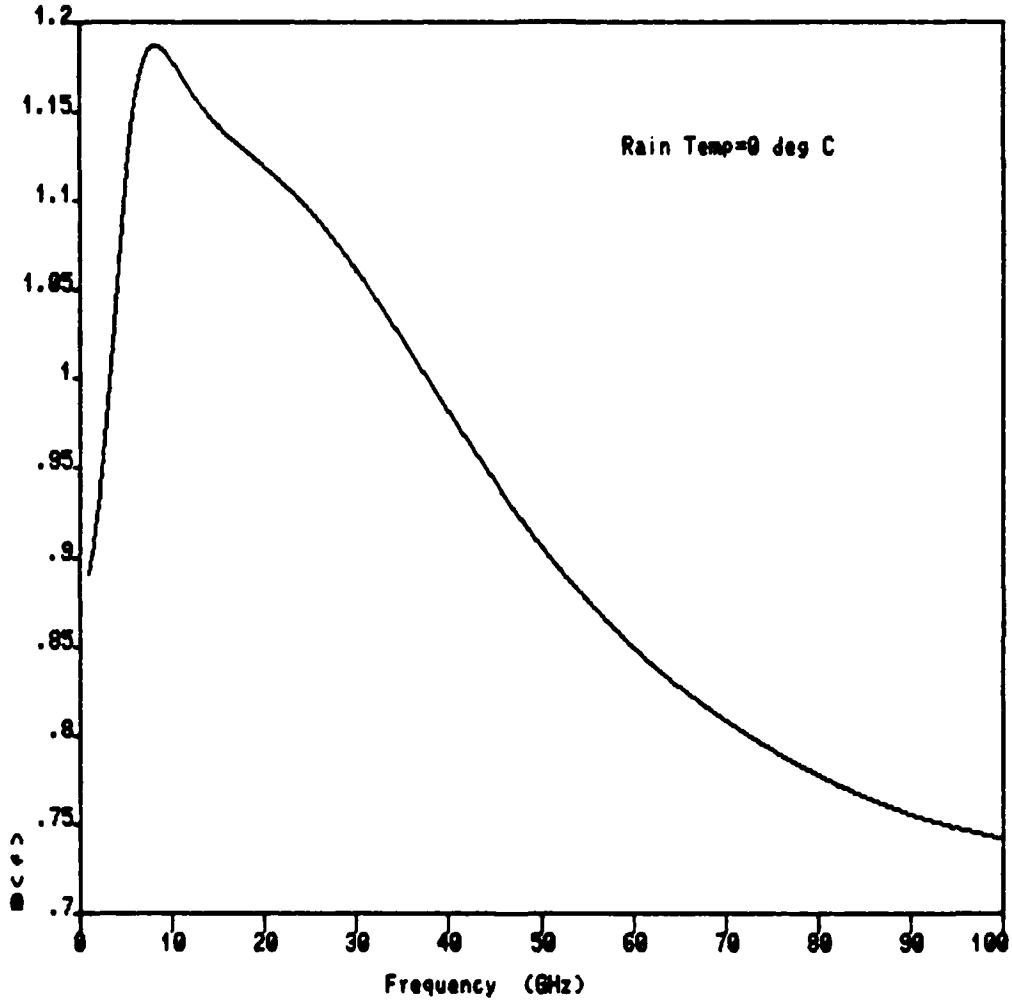


Figure C-5. B(f) vs. Frequency for Low Rain Rates (7:323)

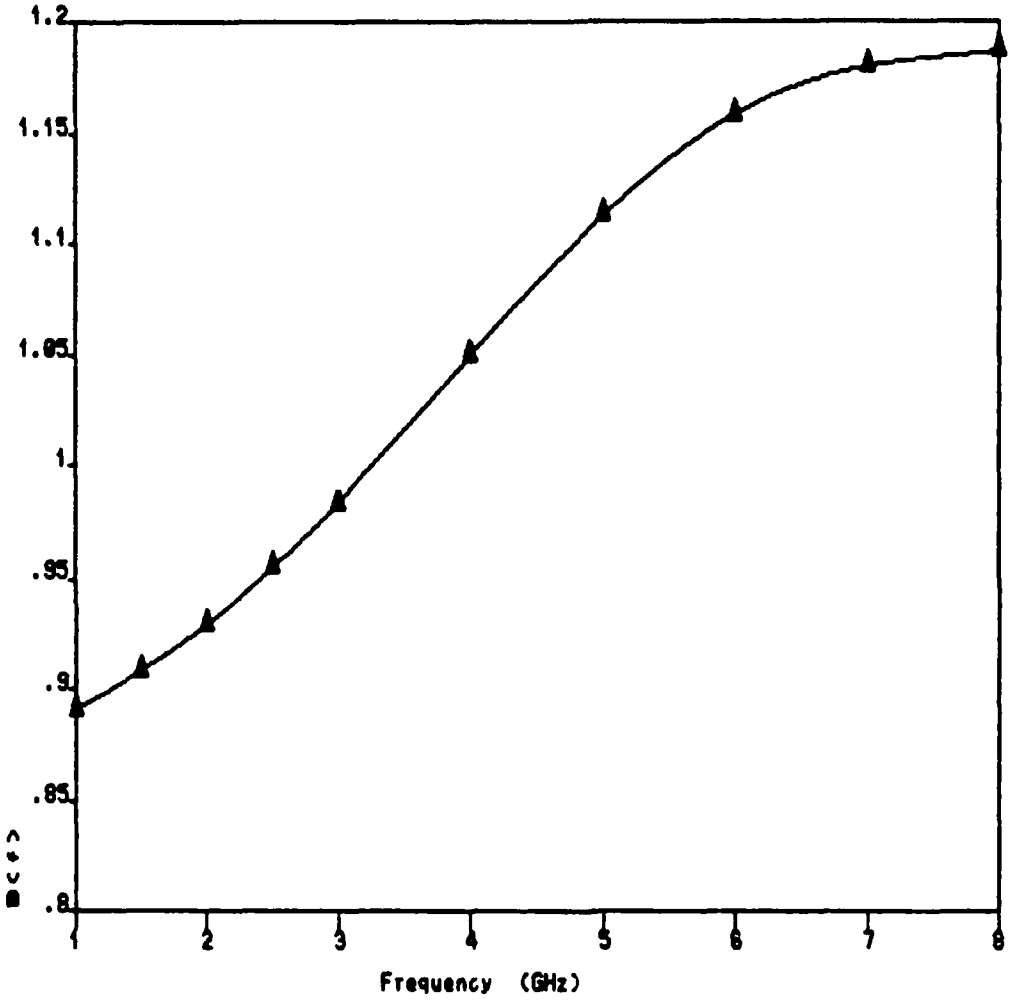


Figure C-6. 1-8 GHz Partition (7:323)

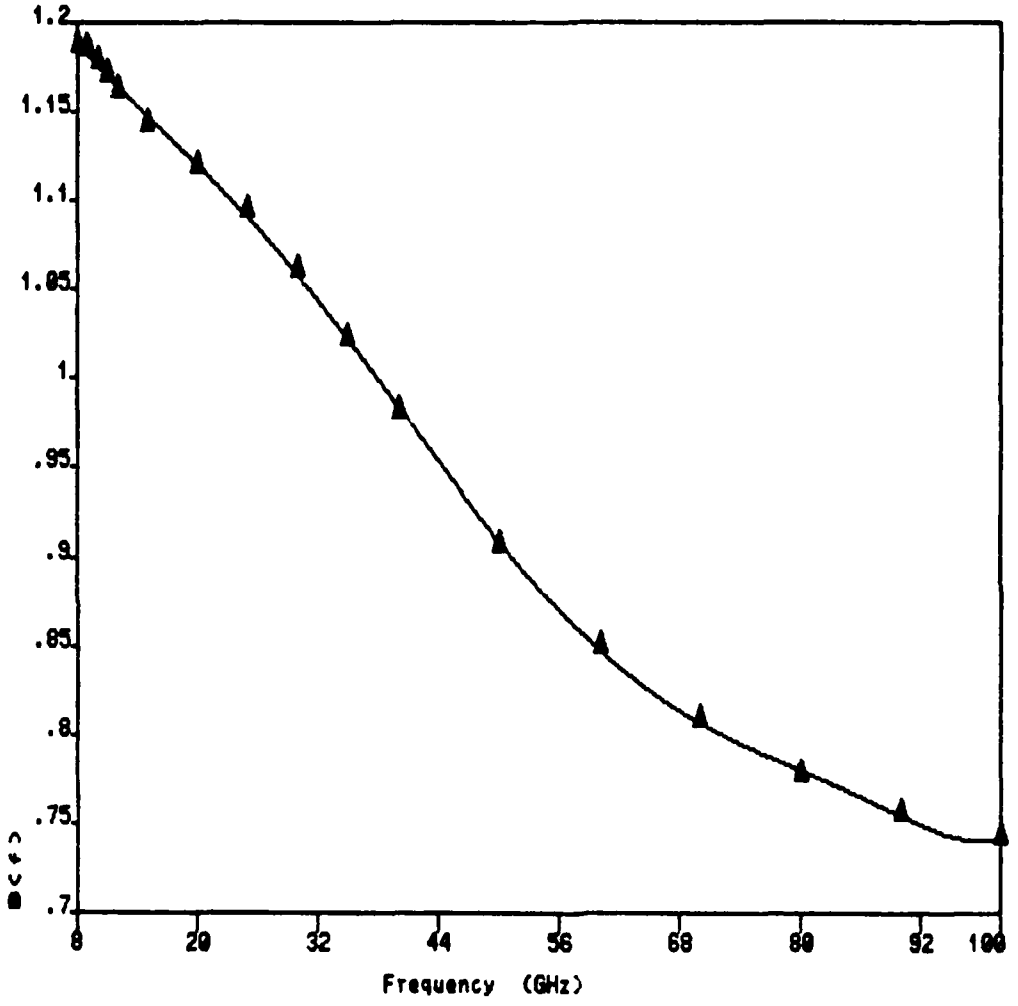


Figure C-7. 8-100 GHz Partition (7:323)

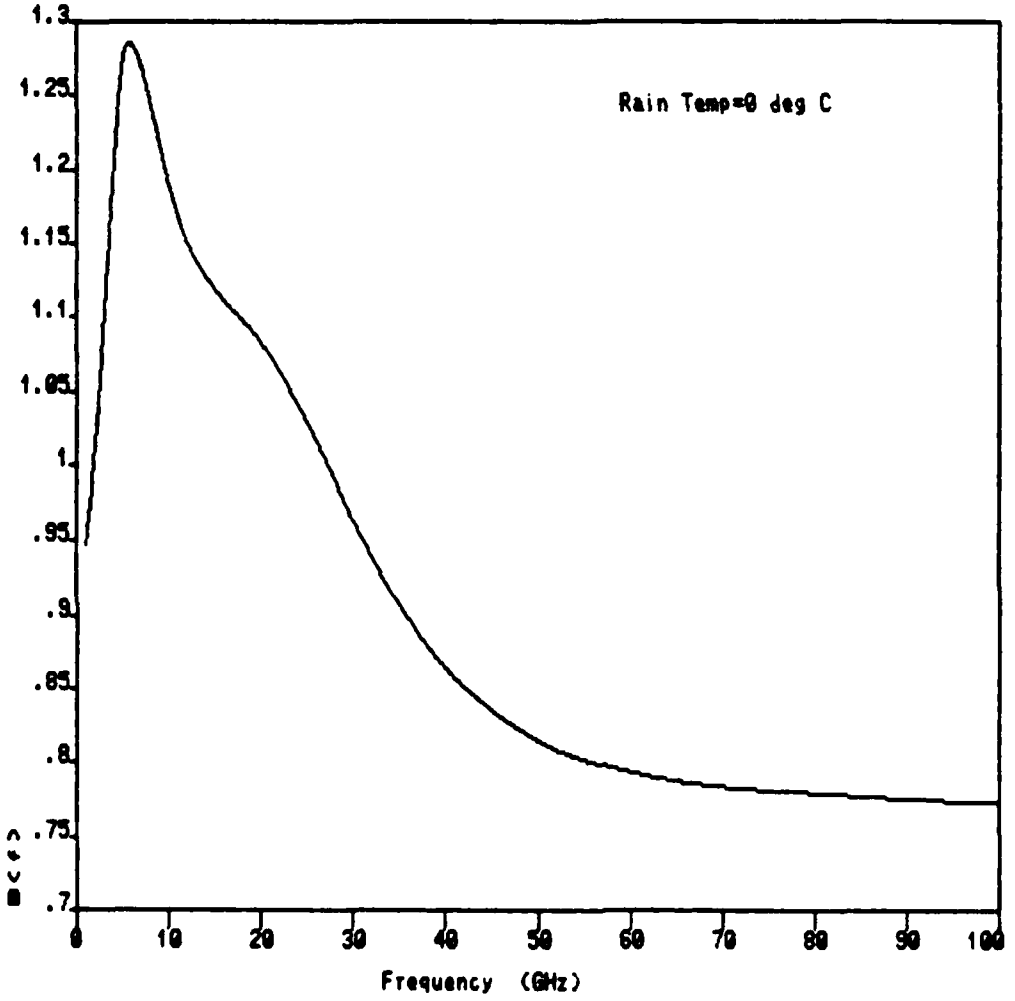


Figure C-8. B(f) vs. Frequency for High Rain Rates (7:323)

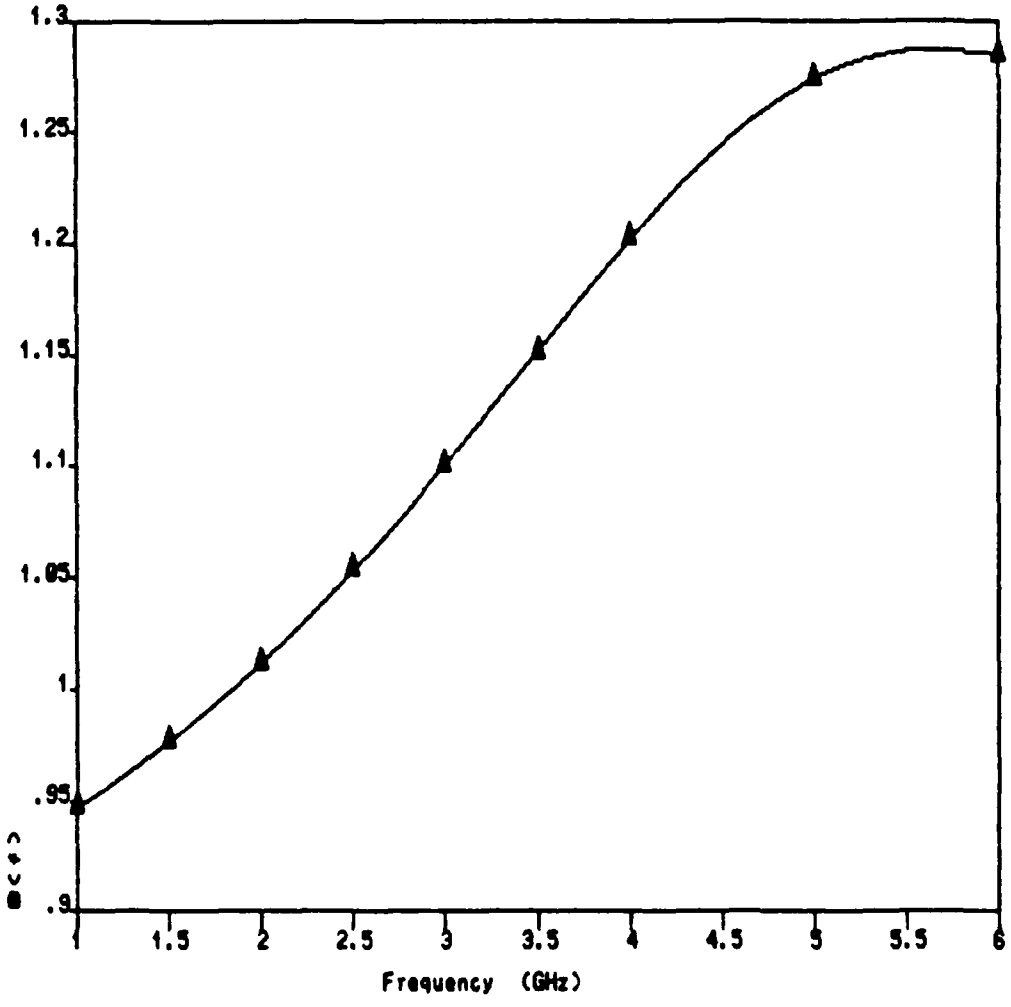


Figure C-9. 1-6 GHz Partition (7:323)

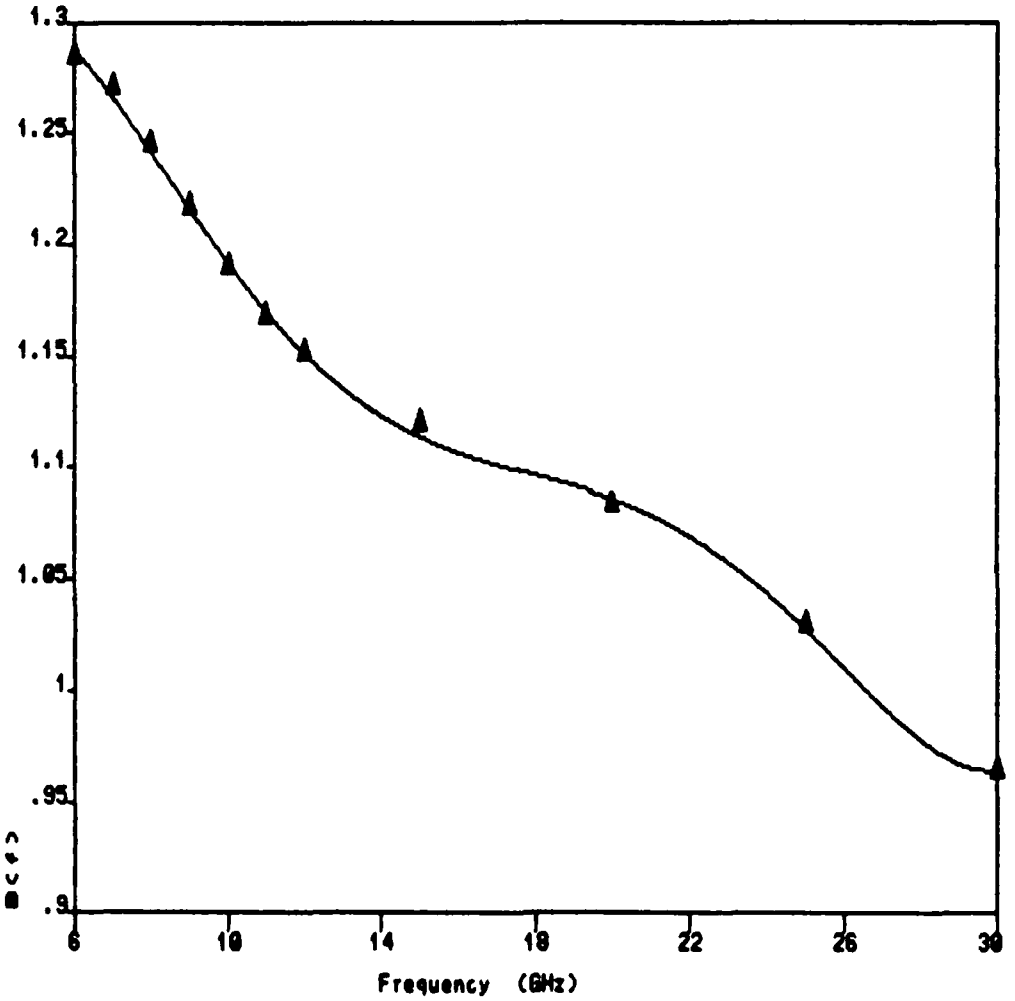


Figure C-10. 6-30 GHz Partition (7:323)

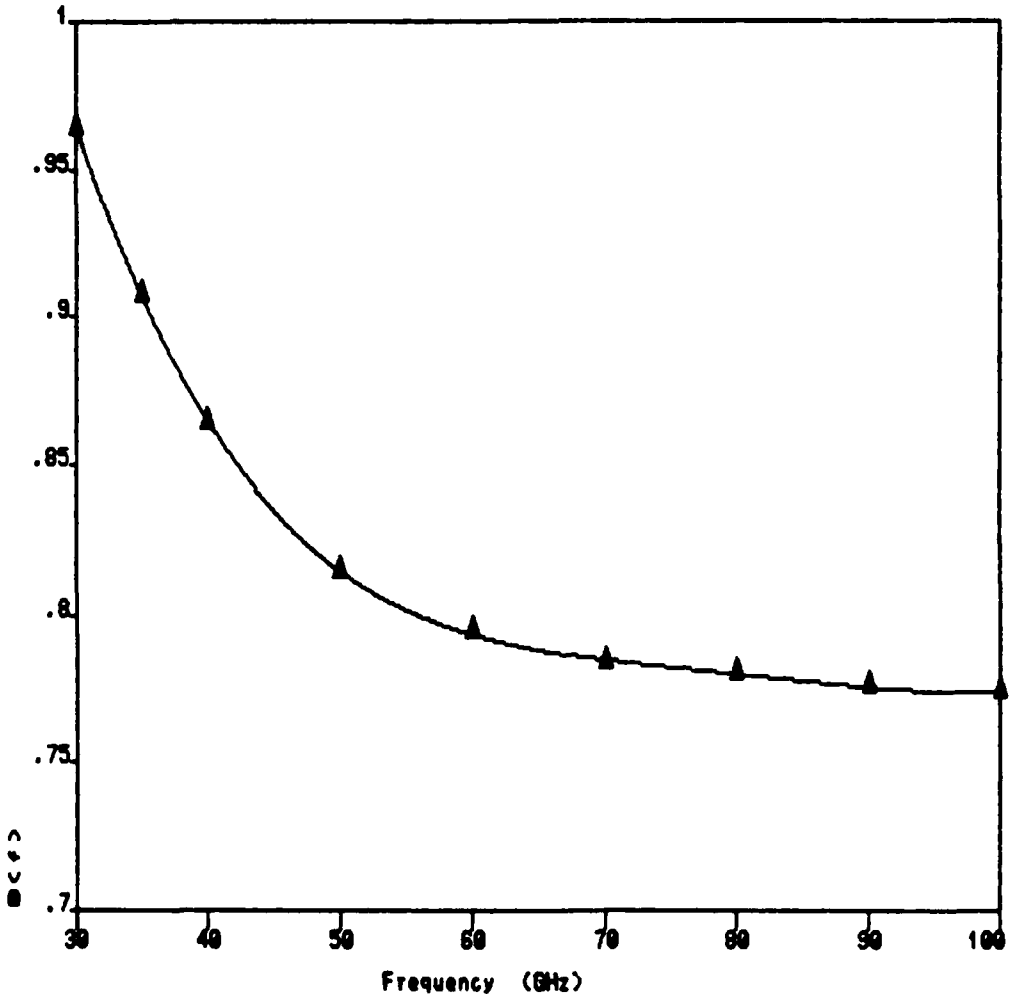


Figure C-11. 30-100 GHz Partition (7:323)

APPENDIX D

Source Code Listing

```
(*****************************************************************************)
(* SATELLITE LINK ANALYSIS WORKSTATION FOR IBM PC     (Ver 1.1)              *)
(* Copyright 1986 by Bill Jackson                                            *)
(*****************************************************************************)
program master(input,output);
(*****************************************************************************)
const mu=3.964e14;      (* gravitational constant times Earth mass *)
      Re=6.378e6;       (* Earth radius *)
      dr=1.7453e-2;     (* degrees-to-radians conversion *)
      rd=57.2958;       (* radians-to-degrees conversion *)
      k=1.38e-23;       (* Boltzman's constant *)
(*****************************************************************************)
type legal=set of char;
     filename=string[23];
     link=record
               upl:real;
               dnl:real;
          end;
     linkrecord=record
                    altitude (* satellite altitude *):real;
                    lat (* ground station latitude *),
                    Hb (* ground station MSL elevation *),
                    alpha (* elevation angle *),
                    freq (* frequency *),
                    TxEfficiency (* transmitting antenna efficiency *),
                    RxEfficiency (* receiving antenna efficiency *),
                    TxGain (* transmitting antenna gain *),
                    RxGain (* receiving antenna gain *),
                    beamwidth (* 3-dB beamwidth *),
                    Pout (* transmitter output power *),
                    TxCircuitLoss (* transmitter circuit losses *),
                    RxCircuitLoss (* receiver circuit losses *),
                    EIRP (* effective isotropic radiated power *),
                    NoiseFigure (* noise figure *),
                    RxAntTemp (* receiving antenna noise temperature *),
                    Teq (* system equivalent temperature *),
                    GT (* G/T figure of merit *),
                    SlantRange (* slant range to satellite *),
                    rainrate (* rain rate *),
                    RainAtten (* rainfall attenuation *),
                    AirAtten (* atmospheric attenuation *),
                    PathLoss (* free space propagation loss *),
                    Pr (* received useful power *),
                    CNR (* carrier-to-noise power density ratio *),
                    bandwidth (* system bandwidth *),
                    datarate (* transmitted data rate *),
                    modtype (* modulation type *),
                    m (* number of signalling levels *),
```

```
                    BER (* bit error rate *),
                    AvailEbNo (* available Eb/No *),
                    ReqEbNo (* required Eb/No *),
                    MaxDatarate (* maximum data rate *),
                    margin (* link power margin *),
                    AntennaType (* antenna type *),
                    beamtype (* beam type *),
                    arg1,arg2,arg3,arg4,arg5 (* dummy variables *):link;
                end;
(*****************************************************************************)
var InputError (* illegal numeric input *):boolean;
    integerdata (* integer input data *):integer;
    realdata (* real input data *):real;
    legalinteger,legalreal (* valid input sets *):legal;
    dumdum (* dummy file *):file;
    data (* link file record containing all pertinent variables *):linkrecord;
    linkfile (* link file *):file of linkrecord;
    buffer (* storage buffer for use in deleting link files *),
    library (* directory of current link files *):file of string[23];
    i (* loop variable *),
    option (* menu item option *),
    choice (* submenu item choice *),
    site (* antenna site *):integer;
    EbNo (* temporary storage for Eb/No *),
    rate (* temporary storage for datarate *),
    err (* temporary storage for bit error rate *),
    limit (* limiting data transmission rate *):real;
    junk (* dummy variable *):real;
    path (* link file directory path *):string[11];
    oldname,name,dummyname (* link filename temporary storage *):string[23];
    letter (* dummy variable *):char;
    inp (* dummy variable *):string[8];
    killname (* name of file to be deleted *):string[8];
    havealpha (* elevation angles previously computed *),
    skip (* skip reentry of elevation angles in link analysis *),
    recompute (* it is possible to recompute the existing link budget *),
    AlreadyThere (* filename already exists *),
    uplink (* uplink portion is being computed *),
    newinfo (* new unsaved data has neen generated *),
    standard (* FALSE when computing BER given datarate *),
    shannonflag (* indicates Shannon rate is limiting factor *),
    nyquistflag (* indicates Nyquist rate is limiting factor *):boolean;
    title (* display titles *):array [1..10] of string[20];
(*****************************************************************************)
(*                    INCLUSION FILES INSERTED HERE                          *)
(*****************************************************************************)
§$I A:integer.past
§$I A:real.past
§$I A:exist.past
§$I A:trig.past
§$I A:pwr.past
§$I A:bessel.past
§$I A:log10.past
```

```
§$I A:alog.past
§$I A:log2.past
§$I A:noise.past
§$I A:custom.past
§$I A:wait.past
§$I A:getalt.past
§$I A:escape.past
§$I A:blackbod.past
§$I A:circular.past
§$I A:ellipse.past
§$I A:azel.past
§$I A:q.past
§$I A:qinverse.past
§$I A:function.past
§$I A:horn.past
§$I A:helix.past
§$I A:dish.past
§$I A:antenna1.past
§$I A:antenna2.past
§$I A:xmit.past
§$I A:rec.past
§$I A:rain.past
§$I A:air.past
§$I A:power.past
§$I A:saveit.past
§$I A:capacity.past
§$I A:bitrate.past
(*****************************************************************************)
(*                          INITIALIZATION AND SET-UP                        *)
(*****************************************************************************)
begin
title[1]:='UPLINK TRANSMITTER';
title[2]:='UPLINK RECEIVER';
title[3]:='DOWNLINK TRANSMITTER';
title[4]:='DOWNLINK RECEIVER';
title[5]:='GROUND TERMINAL';
title[6]:='SATELLITE';
title[7]:='UPLINK SITE';
title[8]:='DOWNLINK SITE';
title[9]:='U P L I N K';
title[10]:='D O W N L I N K';
path:='a:data@';
legalinteger:=['0'..'9'];
legalreal:=['0'..'9','.','-','+','e','E'];
havealpha:=false;
skip:=false;
recompute:=false;
AlreadyThere:=false;
newinfo:=false;
standard:=true;
assign(dumdum,'a:array.chn');
assign(library,path+'link.dir');
assign(buffer,path+'temp.sto');
```

```
if not exist(path+'link.dir') then
begin
     rewrite(library);
     close(library);
end;
(*****************************************************************************)
repeat
textmode;
textbackground(green);
clrscr;
gotoXY(30,5);
textbackground(yellow);
textcolor(black);
writeln(' M A I N   M E N U ':19);
textbackground(green);
textcolor(white);
gotoXY(1,10);
writeln('     1.  Satellite Orbital Parameters');
writeln('     2.  Antenna Gain Patterns');
writeln('     3.  Link Analysis');
writeln('     4.  Display Link Budget');
writeln('     5.  Recompute Link Budget');
writeln('     6.  Store Link File');
writeln('     7.  Recall Link File');
writeln('     8.  Delete Link File');
writeln('     9.  List Link File Directory');
writeln('    10.  Change Default Drive');
writeln('    11.  EXIT');
repeat
      writeln;writeln;write(' ENTER OPTION :');
      Get Integer;
      option:=integerdata;
      if not (option in [1..11]) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
case option of
     1:begin (* Satellite Orbital Parameters *)
            textbackground(red);
            clrscr;
            gotoXY(1,10);
            writeln('     1.  Escape Velocity');
            writeln('     2.  Steady-State Temperature');
            writeln('     3.  Circular Orbit Parameters');
            writeln('     4.  Elliptical Orbit Parameters');
            writeln('     5.  Azimuth and Elevation Angle');
            repeat
                 writeln;writeln;write(' ENTER CHOICE :');
                 Get Integer;
                 choice:=integerdata;
                 if not (choice in [1..5]) then InputError:=true;
                 if InputError=true then noise;
            until InputError=false;
            case choice of
```

```
            1:EscapeVelocity;
            2:blackbody;
            3:circular;
            4:elliptical;
            5:azel;
        end;
  end;
2:begin (* Antenna Gain Patterns *)
        textbackground(red);
        clrscr;
        gotoXY(1,10);
        writeln('     1.  Horn');
        writeln('     2.  Helix');
        writeln('     3.  Parabolic Dish');
        writeln('     4.  Phased Array of Dipoles');
        repeat
              writeln;writeln;write(' ENTER CHOICE :');
              GetInteger;
              choice:=integerdata;
              if not (choice in [1..4]) then InputError:=true;
              if InputError=true then noise;
        until InputError=false;
        case choice of
             1:horn;
             2:helix;
             3:dish;
             4:chain(dumdum);
        end;
  end;
3:begin (* Link Analysis *)
        recompute:=true;
        newinfo:=true;
        textbackground(red);
        clrscr;
        gotoXY(1,5);
        repeat
              write(' ENTER UPLINK FREQUENCY  (1 - 100 GHz) :');
              GetReal;
              data.freq.upl:=realdata;
              if not((data.freq.upl>=1.0) and (data.freq.upl<=100.0))
              then InputError:=true;
              if InputError=true then noise;
        until InputError=false;
        repeat
              write(' ENTER DOWNLINK FREQUENCY  (1 - 100 GHz) :');
              GetReal;
              data.freq.dnl:=realdata;
              if not((data.freq.dnl>=1.0) and (data.freq.dnl<=100.0))
              then InputError:=true;
              if InputError=true then noise;
        until InputError=false;
        writeln;
        if havealpha=true then
```

```
begin
     write(' USE ELEVATION ANGLES FROM MENU ITEM #1 ?  ');
     readln(letter);
     if ((letter='Y') or (letter='y')) then skip:=true else
     skip:=false;
end;
if skip=false then
begin
     repeat
          write(' ENTER TRANSMITTING TERMINAL ELEVATION ANGLE'+
          ' (deg) :');
          GetReal;
          data.alpha.upl:=realdata;
          if not((data.alpha.upl>=0) and (data.alpha.upl<=360))
          then InputError:=true;
          if InputError=true then noise;
     until InputError=false;
     repeat
          write(' ENTER TRANSMITTING TERMINAL LATITUDE (deg) :');
          GetReal;
          data.lat.upl:=realdata;
          if (abs(data.lat.upl)>90) then InputError:=true;
          if InputError=true then noise;
     until InputError=false;
     writeln;
     repeat
          write(' ENTER RECEIVING TERMINAL ELEVATION ANGLE (deg)'+
          ' :');
          GetReal;
          data.alpha.dnl:=realdata;
          if not((data.alpha.dnl>=0) and (data.alpha.dnl<=360))
          then InputError:=true;
          if InputError=true then noise;
     until InputError=false;
     repeat
          write(' ENTER RECEIVING TERMINAL LATITUDE (deg) :');
          GetReal;
          data.lat.dnl:=realdata;
          if (abs(data.lat.dnl)>90) then noise;
     until InputError=false;
end;
writeln;
repeat
      write(' ENTER TRANSMITTING SITE ALTITUDE (m) :');
      GetReal;
      data.Ho.upl:=realdata;
      if (abs(data.Ho.upl)>Re) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      write(' ENTER RECEIVING SITE ALTITUDE (m) :');
      GetReal;
      data.Ho.dnl:=realdata;
```

```
            if (abs(data.Ho.dnl)>Re) then InputError:=true;
        until InputError=false;
        writeln;
        getaltitude;
        data.altitude:=realdata;
        writeln;
        repeat
            write(' ENTER SYSTEM BANDWIDTH  (MHz) :');
            GetReal;
            data.bandwidth.upl:=realdata;
            data.bandwidth.dnl:=realdata;
            if not((data.bandwidth.upl>0) and (data.bandwidth.upl<1000.0))
            then InputError:=true;
            if InputError=true then noise;
        until InputError=false;
        for site:=1 to 4 do
        begin
            if ((site=1) or (site=2)) then uplink:=true else
            uplink:=false;
            case site of
                1,4:antenna1;
                2,3:antenna2;
            end;
        wait;
        end;
        uplink:=true;
        transmitter;
        uplink:=false;
        transmitter;
        uplink:=true;
        receiver;
        uplink:=false;
        receiver;
        RecPower;
        uplink:=true;
        bitrate;
        uplink:=false;
        bitrate;
    end;
4:begin (* Display Link Budget *)
        textbackground(black);
        clrscr;
        textcolor(white);
        writeln('UPLINK':36,'DOWNLINK':25);
        with data do
        begin
            textcolor(lightcyan);
            writeln('Frequency (GHz)      ':21,freq.upl:15:2,freq.dnl:25:2);
            writeln('Slant Range (km)     ':21,Slantrange.upl:15:2,
            SlantRange.dnl:25:2);
            writeln('3-dB Beamwidth (deg) ':21,beamwidth.upl:15:3,
            beamwidth.dnl:25:3);
            textcolor(lightblue);
```

```
                writeln('Tx Power (W)          ':21,Pout.upl:15:2,Pout.dnl:25:2);
                writeln('Tx Circuit Losses     ':21,TxCircuitLoss.upl:15:2,
                TxCircuitLoss.dnl:25:2);
                writeln('Tx Antenna Gain       ':21,TxGain.upl:15:2,
                TxGain.dnl:25:2);
                writeln('EIRP                  ':21,EIRP.upl:15:2,EIRP.dnl:25:2);
                textcolor(yellow);
                writeln('Path Loss             ':21,PathLoss.upl:15:2,
                PathLoss.dnl:25:2);
                writeln('Rain Attenuation      ':21,RainAtten.upl:15:2,
                RainAtten.dnl:25:2);
                writeln('Atmos Attenuation     ':21,AirAtten.upl:15:2,
                AirAtten.dnl:25:2);
                textcolor(lightgreen);
                writeln('Received Useful Power':21,Pr.upl:15:2,Pr.dnl:25:2);
                writeln('Rx Antenna Gain       ':21,RxGain.upl:15:2,
                RxGain.dnl:25:2);
                writeln('Rx Circuit Losses     ':21,RxCircuitLoss.upl:15:2,
                RxCircuitLoss.dnl:25:2);
                writeln('Rx Noise Figure       ':21,NoiseFigure.upl:15:2,
                NoiseFigure.dnl:25:2);
                writeln('System Eq Temp (K)    ':21,Teq.upl:15:2,Teq.dnl:25:2);
                textcolor(lightmagenta);
                writeln('G/T Fig of Merit      ':21,GT.upl:15:2,GT.dnl:25:2);
                writeln('C N R                 ':21,CNR.upl:15:2,CNR.dnl:25:2);
                write('Data Rate (Mb/sec)    ':21);
                if datarate.upl>MaxDatarate.upl then
                textcolor(lightmagenta+blink);
                write(datarate.upl:15:2);
                textcolor(lightmagenta);
                if datarate.dnl>MaxDatarate.dnl then
                textcolor(lightmagenta+blink);
                writeln(datarate.dnl:25:2);
                textcolor(lightmagenta);
                writeln('B E R                 ':21,'       ',BER.upl:8,
                '               ',BER.dnl:8);
                writeln('Available Eb/No       ':21,AvailEbNo.upl:15:2,
                AvailEbNo.dnl:25:2);
                writeln('Required Eb/No        ':21,ReqEbNo.upl:15:2,
                ReqEbNo.dnl:25:2);
                textcolor(lightred);
                write('Margin                ':21);
                if margin.upl<0 then textcolor(lightred+blink);
                write(margin.upl:15:2);
                textcolor(lightred);
                if margin.dnl<0 then textcolor(lightred+blink);
                writeln(margin.dnl:25:2);
                textcolor(lightred);
                write('Max Datarate (Mb/sec)':21,MaxDatarate.upl:15:2,
                MaxDatarate.dnl:25:2);
        end;
    end;
5:if recompute=true then (* Recompute Link Budget *)
```

```
begin
     newinfo:=true;
     textbackground(red);
     clrscr;
     gotoXY(1,5);
     writeln('     1.  Frequency');
     writeln('     2.  Antenna Design');
     writeln('     3.  Transmitter Power');
     writeln('     4.  Receiver Noise Figure');
     writeln('     5.  Bit Error Rate');
     writeln('     6.  Data Rate');
     writeln('     7.  Compute BER given data rate');
     repeat
          writeln;writeln;write(' ENTER CHOICE :');
          GetInteger;
          choice:=integerdata;
          if not (choice in [1..7]) then InputError:=true;
          if InputError=true then noise;
     until InputError=false;
     clrscr;
     if choice<>2 then
     begin
     repeat
          writeln;write(' (U)plink or (D)ownlink ?');
          readln(letter);
          if not ((letter in ['a'..'z']) or (letter in ['A'..'Z']))
          then noise;
     until (letter in ['a'..'z']) or (letter in ['A'..'Z']);
     if ((letter='u') or (letter='U')) then uplink:=true
     else uplink:=false;
     end;
     case choice of
          1:begin (* enter new frequency *)
                repeat
                     gotoXY(2,5);
                     write(' ENTER NEW FREQUENCY  (GHz) :');
                     GetReal;
                     junk:=realdata;
                     if not((junk>=1.0) and (junk<=100.0)) then
                     InputError:=true;
                     if InputError=true then noise;
                until InputError=false;
                if uplink=true then data.freq.upl:=junk else
                data.freq.dnl:=junk;
            end;
          2:begin (* enter new antenna design *)
                repeat
                     gotoXY(1,5);
                     writeln('     1.  Uplink Transmitter');
                     writeln('     2.  Uplink Receiver');
                     writeln('     3.  Downlink Transmitter');
                     writeln('     4.  Downlink Receiver');
                     writeln;writeln;write(' ENTER ANTENNA SITE :');
```

```
                GetInteger;
                site:=integerdata;
                if not(site in [1..4]) then InputError:=true;
                if InputError=true then noise;
            until InputError=false;
            if site<=2 then uplink:=true else uplink:=false;
            if ((site=1) or (site=4)) then antenna1 else antenna2;
        end;
    3:begin (* enter new transmitter power *)
            repeat
                gotoXY(2,5);
                write(' ENTER OUTPUT POWER LEVEL (W) :');
                GetReal;
                junk:=realdata;
                if junk<=0 then InputError:=true;
                if InputError=true then noise;
            until InputError=false;
            if uplink=true then data.Pout.upl:=junk else
            data.Pout.dnl:=junk;
        end;
    4:begin (* enter new receiver noise figure *)
            repeat
                gotoXY(2,5);
                write(' ENTER NEW NOISE FIGURE :');
                GetReal;
                junk:=realdata;
                if not((junk>0) and (junk<=1.0)) then
                InputError:=true;
                if InputError=true then noise;
            until InputError=false;
            if uplink=true then data.NoiseFigure.upl:=junk else
            data.NoiseFigure.dnl:=junk;
        end;
    5:begin (* enter new bit error rate *)
            standard:=true;
            repeat
                gotoXY(2,5);
                write(' ENTER NEW DESIRED BIT ERROR RATE :');
                GetReal;
                junk:=realdata;
                if not((junk>0) and (junk<1.0)) then
                InputError:=true;
            until InputError=false;
            if uplink=true then data.BER.upl:=junk else
            data.BER.dnl:=junk;
        end;
    6:begin (* enter new data rate *)
            standard:=true;
            repeat
                gotoXY(2,5);
                write(' ENTER DATA RATE DESIRED  (Mb/sec) :');
                GetReal;
                junk:=realdata;
```

```
                    if junk<1e-3 then InputError:=true;
                    if InputError=true then noise;
               until InputError=false;
               if uplink=true then data.datarate.upl:=junk else
               data.datarate.dnl:=junk;
          end;
     7:begin (* compute BER given data rate *)
               standard:=false;
               repeat
                    gotoXY(2,5);
                    write(' ENTER MAX DATA RATE DESIRED  (Mb/sec) :');
                    GetReal;
                    rate:=realdata;
                    if rate<1e-3 then InputError:=true;
                    if InputError=true then noise;
               until InputError=false;
          end;
end;
with data do
begin (* perform actual recomputations *)
     case trunc(AntennaType.upl) of
          1:begin
                    TxGain.upl:=10*log10(gain1(arg1.upl,freq.upl));
                    beamwidth.upl:=bw1(arg1.upl,freq.upl);
               end;
          2:begin
                    TxGain.upl:=10*log10(gain2(arg1.upl,arg2.upl,
                    freq.upl));
                    beamwidth.upl:=bw2(arg1.upl,arg2.upl,freq.upl);
               end;
          3:begin
                    TxGain.upl:=10*log10(gain3(arg1.upl,freq.upl));
                    beamwidth.upl:=bw3(arg1.upl,freq.upl);
               end;
          4:begin
                    TxGain.upl:=10*log10(gain4(arg3.upl,arg1.upl,
                    freq.upl));
                    beamwidth.upl:=bw4(arg3.upl,arg1.upl,freq.upl);
               end;
     end;
     case trunc(AntennaType.dnl) of
          1:RxGain.dnl:=10*log10(gain1(arg1.dnl,freq.dnl));
          2:RxGain.dnl:=10*log10(gain2(arg1.dnl,arg2.dnl,
          freq.dnl));
          3:RxGain.dnl:=10*log10(gain3(arg1.dnl,freq.dnl));
          4:RxGain.dnl:=10*log10(gain4(arg3.dnl,arg1.dnl,
          freq.dnl));
     end;
     RxGain.upl:=10*log10(gain3(arg5.upl,freq.upl));
     TxGain.dnl:=10*log10(gain3(arg5.dnl,freq.dnl));
     EIRP.upl:=isotropic(Pout.upl,TxGain.upl,TxCircuitLoss.upl);
     EIRP.dnl:=isotropic(Pout.dnl,TxGain.dnl,TxCircuitLoss.dnl);
     Teq.upl:=eqtemp(RxAntTemp.upl,NoiseFigure.upl,
```

```
RxCircuitLoss.upl);
Teq.dnl:=eqtemp(RxAntTemp.dnl,NoiseFigure.dnl,
RxCircuitLoss.dnl);
GT.upl:=figmerit(RxGain.upl,Teq.upl);
GT.dnl:=figmerit(RxGain.dnl,Teq.dnl);
PathLoss.upl:=-36.6-20*log10(slantrange.upl*1e3*6.214e-4*
freq.upl*1e3);
PathLoss.dnl:=-36.6-20*log10(slantrange.dnl*1e3*6.214e-4*
freq.dnl*1e3);
AirAtten.upl:=air(freq.upl,alpha.upl);
AirAtten.dnl:=air(freq.dnl,alpha.dnl);
RainAtten.upl:=rain(rainrate.upl,freq.upl,alpha.upl,Ho.upl,
lat.upl);
RainAtten.dnl:=rain(rainrate.dnl,freq.dnl,alpha.dnl,Ho.dnl,
lat.dnl);
Pr.upl:=power(Pout.upl,TxCircuitLoss.upl,TxGain.upl,
TxEfficiency.upl,PathLoss.upl,AirAtten.upl,RainAtten.upl,
RxEfficiency.upl,RxGain.upl,RxCircuitLoss.upl);
Pr.dnl:=power(Pout.dnl,TxCircuitLoss.dnl,TxGain.dnl,
TxEfficiency.dnl,PathLoss.dnl,AirAtten.dnl,RainAtten.dnl,
RxEfficiency.dnl,RxGain.dnl,RxCircuitLoss.dnl);
CNR.upl:=cnratio(Pr.upl,Teq.upl);
CNR.dnl:=cnratio(Pr.dnl,Teq.dnl);
if standard=true then
begin
     AvailEbNo.upl:=EbNoAvail(Pr.upl,Teq.upl,datarate.upl);
     AvailEbNo.dnl:=EbNoAvail(Pr.dnl,Teq.dnl,datarate.dnl);
     ReqEbNo.upl:=EbNoReq(trunc(modtype.upl),m.upl,BER.upl);
     ReqEbNo.dnl:=EbNoReq(trunc(modtype.dnl),m.dnl,BER.dnl);
     MaxDataRate.upl:=MaxRate(Pr.upl,Teq.upl,ReqEbNo.upl);
     capacity(bandwidth.upl,Pr.upl,Teq.upl,MaxDatarate.upl,
     M.upl);
     if ((shannonflag=true) or (nyquistflag=true)) then
     MaxDatarate.upl:=limit;
     MaxDataRate.dnl:=MaxRate(Pr.dnl,Teq.dnl,ReqEbNo.dnl);
     capacity(bandwidth.dnl,Pr.dnl,Teq.dnl,MaxDatarate.dnl,
     M.dnl);
     if ((shannonflag=true) or (nyquistflag=true)) then
     MaxDatarate.dnl:=limit;
     margin.upl:=AvailEbNo.upl-ReqEbNo.upl;
     margin.dnl:=AvailEbNo.dnl-ReqEbNo.dnl;
end else
begin
     if uplink=true then
     begin
          EbNo:=CNR.upl-10*log10(rate*1e6);
          err:=biterror(trunc(modtype.upl),m.upl,
          EbNo);
     end else
     begin
          EbNo:=CNR.dnl-10*log10(rate*1e6);
          err:=biterror(trunc(modtype.dnl),m.dnl,
          EbNo);
```

```
                    end;
                    clrscr;
                    gotoXY(1,5);
                    if uplink=true then writeln(title[9]:46) else
                    writeln(title[10]:46);
                    gotoXY(1,10);
                    textcolor(lightmagenta);
                    writeln(' Desired Data Rate = ',rate:0:2,' Mbits/sec');
                    writeln(' Available Eb/No = ',EbNo:0:2,' dB');
                    writeln(' Bit Error Rate = ',err:8);
               end;
          end;
     end else
     begin
          noise;
          textcolor(yellow);
          writeln;writeln(' *** UNABLE TO RECOMPUTE ***');
          textcolor(white);
     end;
   6:begin (* Store Link File *)
          newinfo:=false;
          textbackground(red);
          clrscr;
          gotoXY(1,5);
          saveit;
     end;
   7:begin (* Recall Link File *)
          recompute:=true;
          newinfo:=false;
          textbackground(red);
          clrscr;
          gotoXY(1,5);
          repeat
               write(' ENTER FILENAME  (w/o extension) :');
               readln(name);
               if not((length(name)>=1) and (length(name)<=8)) then noise;
          until (length(name)>=1) and (length(name)<=8);
          name:=path+name+'.lnk';
          if exist(name)=true then
          begin
               assign(linkfile,name);
               reset(linkfile);
               with data do read(linkfile,data);
               close(linkfile);
          end else
          begin
               noise;
               textcolor(yellow+blink);
               writeln;writeln(' *** FILE  NONEXISTENT ***');
               textcolor(white);
          end;
     end;
   8:begin (* Delete Link File *)
```

```
          textbackground(red);
          clrscr;
          gotoXY(1,10);
          repeat
               write(' DELETE WHICH FILE ?');
               readln(killname);
               if not((length(killname)>=1) and (length(killname)<=8))
               then noise;
          until (length(killname)>=1) and (length(killname)<=8);
          writeln;write(' ARE YOU SURE ?  (Y or N)');
          readln(letter);
          if ((letter='Y') or (letter='y')) then
          begin
               name:=path+killname+'.lnk';
               if exist(name)=true then
               begin
                    clrscr;
                    gotoXY(1,12);
                    textcolor(yellow+blink);
                    writeln('D E L E T I N G   F I L E':52);
                    assign(linkfile,name);
                    erase(linkfile); (* erase disk file *)
                    reset(library);
                    rewrite(buffer);
                    while not EOF(library) do
                    begin (* copy all other filenames to buffer *)
                         read(library,name);
                         if pos(killname,name)=0 then write(buffer,name);
                    end;
                    reset(buffer);
                    rewrite(library);
                    while not EOF(buffer) do (* copy buffer into library *)
                    begin
                         read(buffer,name);
                         write(library,name);
                    end;
                    close(buffer);
                    close(library);
                    clrscr;
               end else
               begin
                    textcolor(yellow+blink);
                    noise;
                    writeln;writeln(' *** FILE NONEXISTENT ***');
                    textcolor(white);
               end;
          end;
     end;
9:begin (* List Link File Directory *)
          reset(library);
          textbackground(red);
          clrscr;
          if filesize(library)=0 then
```

```
            begin
                clrscr;
                noise;
                textcolor(yellow+blink);
                writeln;writeln(' *** LIBRARY EMPTY ***');
            end;
            while not EOF(library) do
            begin
                read(library,name);
                writeln('     ',name);
            end;
            close(library);
        end;
    10:begin (* Change Default Drive *)
            textbackground(red);
            clrscr;
            gotoXY(1,5);
            writeln('NOTE: the path disk:®data must exist on the DATA disk !');
            repeat
                 writeln;write(' ENTER NEW DEFAULT DRIVE :');
                 readln(letter);
                 if not((letter in ['a'..'e']) or (letter in ['A'..'E']))
                 then noise;
            until (letter in ['a'..'e']) or (letter in ['A'..'E']);
            path:=letter+':data®';
            assign(library,path+'link.dir');
            assign(buffer,path+'temp.sto');
            if not exist(path+'link.dir') then
            begin
                rewrite(library);
                close(library);
            end;
       end;
end;  (* end of case *)
if option<>11 then wait;
until option=11; (* EXIT routine *)
if newinfo=true then (* last chance to save data *)
begin
     writeln;write(' WARNING: DATA NOT SAVED ... save ? (Y,N)  ');
     read(letter);
     if ((letter='Y') or (letter='y')) then saveit;
end;
textcolor(white);
textbackground(black);
clrscr;
end.
```

```
(* custom graphics driver *)
procedure customplot(x,y,n:integer);
(* plot a point *)
begin
     plot(x,y,n);
end;
(**)
procedure customdraw(x1,y1,x2,y2,n:integer);
(* draw a line between two points *)
begin
     draw(x1,y1,x2,y2,n);
end;
(**)
(* determine if a disk file exists *)
function exist(name:filename):boolean;
var fil(* dummy filename *):file;
begin
assign(fil,name);
§$I-†
reset(fil);
§$I+†
exist:=(IOresult=0);
end;
(**)
(* poll user for satellite altitude *)
procedure getaltitude;
begin
repeat
      write(' ENTER SATELLITE ALTITUDE (m)  ( or G for geosynchronous ) :');
      GetReal;
      if realdata<1.5e5 then InputError:=true;
      if InputError=true then noise;
until InputError=false;
end;
(**)
(* store an integer input *)
procedure GetInteger;
var i (* loop variable *),
    code (* error code *):integer;
    character (* individual input character *):char;
    datain (* input data string *):string[5];
begin
InputError:=false;
readln(datain);
for i:=1 to length(datain) do
begin
     character:=copy(datain,i,1);
     if not (character in legalinteger) then InputError:=true else
     val(datain,integerdata,code);
end;
end;
(**)
(* make beeping sound *)
```

```
procedure noise;
begin
sound(1000);
delay(500);
nosound;
end;
(**)
(* store a real input *)
procedure GetReal;
var i (* loop variable *),
    code (* error code *):integer;
    character (* individual input character *):char;
    datain (* input data string *):string[15];
begin
InputError:=false;
readln(datain);
if ((copy(datain,1,1)='g') or (copy(datain,1,1)='G')) then realdata:=3.5784e7
else for i:=1 to length(datain) do
begin
     character:=copy(datain,i,1);
     if not(character in legalreal) then InputError:=true else
     val(datain,realdata,code);
end;
end;
(**)
(* save a file on disk *)
procedure saveit;
begin
repeat
      writeln;write(' ENTER FILENAME  ( 1 - 8 chars ) :');
      readln(name);
      if not((length(name)>=1) and (length(name)<=8)) then noise;
until (length(name)>=1) and (length(name)<=8);
name:=path+name+'.lnk';
dummyname:=name;
delete(dummyname,1,length(path));
reset(library);
(* see if file already in library *)
while not EOF(library) do
begin
     read(library,oldname);
     if pos(oldname,dummyname)<>0 then AlreadyThere:=true;
end;
(* if not already in library, write to disk and update library *)
if AlreadyThere=false then
begin
     assign(linkfile,name);
     rewrite(linkfile);
     with data do write(linkfile,data);
     close(linkfile);
     reset(library);
     if filesize(library) > 0 then seek(library,filesize(library));
     write(library,dummyname);
```

```
     close(library);
end else
begin
     AlreadyThere:=false;
     noise;
     textcolor(yellow+blink);
     writeln;writeln(' *** FILE ALREADY EXISTS ***');
     textcolor(white);
end;
end;
(**)
(* suspend execution pending keyboard entry *)
procedure wait;
label here;
var wide(* width of message *):integer;
begin
gotoXY(1,25);
textcolor(yellow);
if ((option=2) and ((choice=1) or (choice=3))) then wide:=34 else wide:=54;
write('[press any key to continue]':wide);
textcolor(white);
here:if keypressed=false then goto here;
end;
(**)
(* convert dB to numbers *)
function alog(db:real):real;
begin
alog:=pwr(10,db/10);
end;
(**)
(* first-order bessel functions of the first kind *)
function bessel(x:real):real;
var ctr (* counter to truncate insignificant series terms *),
    lim (* max # terms *),
    sign,i,l,z:integer;
    a,subterm,term,old:real; (* constants and loop variables *)
begin
ctr:=0;
lim:=200;
old:=0;
a:=x/2;
term:=1;
sign:=1;
i:=0;
(* evaluate series until 200 terms added or 3 consecutive terms not
contribute to sum *)
while ((i<lim) and (ctr<=2)) do
begin
     i:=i+2;
     sign:=-sign;
     subterm:=1;
     for l:=1 to i div 2 do subterm:=subterm/((1+l)*l);
     for z:=1 to i do subterm:=subterm*x;
```

```
     term:=term+(subterm/pwr(2,i))*sign;
     if a*term=old then ctr:=ctr+1;
     old:=a*term;
end;
bessel:=old;
end;
(**)
(* natural logarithms *)
function log2(x:real):real;
begin
log2:=ln(x)/ln(2.0);
end;
(**)
(* common logarithms *)
function log10(x:real):real;
begin
log10:=ln(x)/ln(10);
end;
(**)
(* raise x to the power of n *)
function pwr(x,n:real):real;
begin
     if x=0 then pwr:=0 else if n=0 then pwr:=1 else
     if x<0 then
          if (trunc(n) mod 2)=0 then pwr:=exp(n*ln(-x)) else pwr:=-exp(n*ln(-x))
     else pwr:=exp(n*ln(x));
end;
(**)
(* Marcum's Q-function *)
function q(u:real):real;
var l (* lower limit of integration *),
    N (* # partitions *),
    inc (* partition width *),
    sum,loop (* loop variables *),
    area (* intermediate result *),
    erf (* value of corresponding error function *):real;
(**)
(* approximation of Q-function *)
function qapprox(x:real):real;
begin
     if x>13 then qapprox:=0 else
     qapprox:=exp(-(sqr(x)/2))/(x*sqrt(2*pi));
end;
(**)
begin
clrscr;
textcolor(yellow+blink);
gotoXY(1,12);
writeln('S T A N D B Y':46);
(* evaluate error function using Simpson's rule *)
if u<4.24 then
begin
     l:=0;
```

```
        u:=u/sqrt(2);
        N:=1000.0;
        inc:=(u-l)/N;
        (* get first and last points *)
        sum:=exp(-sqr(l))+exp(-sqr(u));
        loop:=l+inc;
        (* get remaining points *)
        while loop<=l+(N-1)*inc do
        begin
             sum:=sum+4*exp(-sqr(loop));
             loop:=loop+2*inc;
        end;
        loop:=l+2*inc;
        while loop<=l+(N-2)*inc do
        begin
             sum:=sum+2*exp(-sqr(loop));
             loop:=loop+2*inc;
        end;
    area:=sum*inc/3.0;
    erf:=2/sqrt(pi)*area;
    (* extract Q-function from corresponding error function *)
    q:=(1-erf)/2.0;
    end else q:=qapprox(u); (* if u>=4.24 then use approximation *)
    textcolor(white);
    clrscr;
    end;
    (**)
    (* inverse Q-function *)
    function qinverse(q:real):real;
    var x1,x2,x3 (* dummy variables *):real;
    (**)
    (* equation for argument given the corresponding Q-function *)
    function f(x:real):real;
    begin
         f:=sqr(x)+2*ln(x)+2*ln(q*sqrt(2*pi));
    end;
    (**)
    (* derivative of above equation *)
    function g(x:real):real;
    begin
         g:=2*x+2/x;
    end;
    (**)
    begin
    clrscr;
    textcolor(yellow+blink);
    gotoXY(1,12);
    writeln('S T A N D B Y':46);
    (* establish search range [x1,x2] *)
    x1:=0.1;
    x2:=13;
    (* use method of halving the interval to localize root *)
    while 0.5*abs(x1-x2)>=1e-6 do
```

```
begin
     x3:=(x1+x2)/2.0;
     if abs(f(x3)) + abs(f(x1)) <> abs(f(x3)+f(x1)) then x2:=x3 else x1:=x3;
end;
x2:=x3;
(* use Newton's method to precisely isolate root *)
while abs(x2-x1)>=1e-6 do
begin
     x1:=x2;
     x2:=x1-f(x1)/g(x1);
end;
qinverse:=x2;
textcolor(white);
clrscr;
end;
(**)
(* inverse cosine *)
function arccos(x:real):real;
begin
     if x=0 then arccos:=pi/2 else
     arccos:=arctan(sqrt(1-sqr(x))/x);
end;
(**)
(* inverse sine *)
function arcsin(x:real):real;
begin
     if x=1 then arcsin:=pi/2 else
     arcsin:=arctan(x/sqrt(1-sqr(x)));
end;
(**)
(* tangent *)
function tan(x:real):real;
begin
     tan:=sin(x)/cos(x);
end;
(**)
(* plot gain pattern for horn *)
procedure horn;
var flag (* plot first point flag *):boolean;
    fx,fy (* scaling factors *),
    d (* horn dimension *),
    f(* frequency *),
    inc (* plotting resolution *),
    theta (* polar angle *),
    x,gain,max (* gain variables *):real;
    oldx,oldy,xplot,yplot (* plotting coordinates *):integer;
begin
flag:=false;
fy:=100;fx:=220;
textbackground(blue);
clrscr;
repeat
      write(' ENTER FREQUENCY  (1 - 100 GHz) :');
```

```
      GetReal;
      f:=realdata;
      if not((f>=1.0) and (f<=100.0)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      write(' ENTER HORN DIMENSION (m) :');
      GetReal;
      d:=realdata;
      if d<=0 then InputError:=true;
      if InputError=true then noise;
until InputError=false;
max:=10*log10(gain1(d,f));
graphcolormode;
palette(3);
graphbackground(black);
theta:=-pi/2;
inc:=0.05;
(* sweep theta 180 degrees to generate polar plot *)
while theta<pi/2 do
begin
     x:=pi*sin(theta);
     gain:=sqr(sin(x)/x);
     xplot:=50+round(gain*cos(theta)*fx);
     yplot:=100-round(gain*sin(theta)*fy);
     (* plot first point, then draw lines to each successive point *)
     if flag=false then
     begin
          flag:=true;
          customplot(xplot,yplot,2);
          oldx:=xplot;oldy:=yplot;
     end else
     begin
          customdraw(oldx,oldy,xplot,yplot,2);
          oldx:=xplot;oldy:=yplot;
     end;
theta:=theta+inc;
end;
(* draw little horn to represent antenna *)
customdraw(47,94,47,106,3);
customdraw(47,106,40,103,3);
customdraw(40,103,36,103,3);
customdraw(36,103,36,97,3);
customdraw(36,97,40,97,3);
customdraw(40,97,47,94,3);
gotoXY(1,1);
writeln('FREQ = ',f:0:1,' GHz');
writeln('HORN DIMENSION = ',d:0:1,' m');
gotoXY(1,20);
writeln('MAX GAIN = ',max:0:1,' dB');
writeln;writeln('FIRST SIDELOBE LEVEL = ',max-13.0:0:1,' dB');
end;
(**)
```

```
(* plot gain pattern for helix *)
procedure helix;
var flag (* plot first point flag *),
    check (* flag to supress garbage on screen *):boolean;
    fx,fy (* scaling factors *),
    max (* maximum gain *),
    f (* frequency *),
    k (* wave number *),
    radius (* helix radius *),
    alpha (* pitch angle *),
    turns (* # turns *),
    d (* height of one coil *),
    theta (* polar angle *),
    inc (* plotting resolution *),
    AF,PF (* array factor, pattern factor *),
    fudge(* plotting error adjustment *):real;
    oldx,oldy,xplot,yplot (* plotting coordinates *):integer;
begin
textbackground(blue);
clrscr;
fudge:=0;
check:=false;
flag:=false;
fy:=500;
repeat
      write(' ENTER FREQUENCY  (.1 - 1.0 GHz) :');
      GetReal;
      f:=realdata;
      if not ((f<=1.0) and (f>=0.1)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
k:=2*pi*f*1e9/3e8;
repeat
      write(' ENTER HELIX RADIUS  (optimum = ',1/k:0:3,' m) :');
      GetReal;
      radius:=realdata;
      if not ((radius>=0.9/k) and (radius<=1.1/k)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      write(' ENTER PITCH ANGLE  (12 - 18 deg, 14 = optimum) :');
      GetReal;
      alpha:=realdata;
      if not ((alpha)>=12) and (alpha<=18)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
alpha:=alpha*dr;
repeat
      write(' ENTER NUMBER OF TURNS  (N > 3) :');
      GetReal;
      turns:=int(realdata);
      if turns<4.0 then InputError:=true;
      if InputError=true then noise;
```

```
until InputError=false;
d:=2*pi*radius*tan(alpha);
max:=gain2(2*radius,turns*d,f);
fx:=fy*2.2+turns*40.0;
hires;
hirescolor(lightblue);
customdraw(20,180,620,180,1);
customdraw(320,0,320,180,1);
inc:=1/(6.0*turns); (* adjust resolution *)
oldx:=-500;
theta:=pi+0.1;
(* sweep theta *)
while theta<2*pi do
begin
if theta<=2*pi-fudge+0.01 then
begin (* total pattern=array factor X pattern factor *)
     AF:=abs(sin(cos(theta)*turns*pi*d)/(turns*sin(cos(theta)*pi*d)));
     PF:=cos(k*radius*sin(theta)-0.75*pi)*(1.0/(pi*k*radius*sin(theta)));
     xplot:=320+round(AF*PF*cos(theta)*fx);
     yplot:=180+round(AF*PF*sin(theta)*fy);
     (* suspend plotting until first change in direction *)
     if check=false then if xplot>oldx then oldx:=xplot else if xplot>300 then
     begin
          check:=true;
          fudge:=theta-pi;
     end;
     (* plot first point, then draw lines to each successive point *)
     if check=true then
        if flag=true then
        begin
             customdraw(oldx,oldy,xplot,yplot,1);
             oldx:=xplot;oldy:=yplot;
        end else
        begin
             flag:=true;
             customplot(xplot,yplot,1);
             oldx:=xplot;oldy:=yplot;
        end;
end;
theta:=theta+inc;
end;
gotoXY(1,1);
writeln('FREQ = ',f:0:1,' GHz');
writeln('RADIUS = ',radius:0:2,' m');
writeln('PITCH = ',alpha:0:1,' deg');
writeln('# TURNS = ',turns:0:0);
writeln('MAX GAIN = ',max:0:1,' dB');
end;
(**)
(* plot gain pattern of parabolic reflector *)
procedure dish;
var xl,yl,fx,fxl,fy,fyl (* scaling factors *),
    f (* frequency *),
```

```
    diameter (* dish diameter *),
    dl (* diameter/wavelength *),
    x,gain,max (* gain variables *),
    theta (* polar angle *),
    inc (* plotting resolution *):real;
    oldx,oldy,xplot,yplot (* plotting coordinates *):integer;
begin
fy:=100.0;fx:=130.0;
fy1:=20.0;fx1:=10.0;
textbackground(blue);
clrscr;
repeat
      write(' ENTER FREQUENCY  (1 - 100 GHz) :');
      GetReal;
      f:=realdata;
      if not ((f>=1.0) and (f<=100.0)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      write(' ENTER DISH DIAMETER (m) :');
      GetReal;
      diameter:=realdata;
      if diameter<=0 then InputError:=true;
      if InputError=true then noise;
until InputError=false;
dl:=diameter*f*1e9/3e8;
max:=10*log10(gain3(diameter,f));
graphcolormode;
palette(3);
customdraw(160,100,300,100,1);
customdraw(160,40,160,160,1);
oldx:=160;oldy:=100;
inc:=0.1;
x1:=0;
(* draw small parabola to represent antenna *)
while x1<1.0 do
begin
     y1:=sqrt(x1);
     xplot:=160+round(x1*fx1);
     yplot:=100-round(y1*fy1);
     customdraw(oldx,oldy,xplot,yplot,1);
     oldx:=xplot;oldy:=yplot;
     x1:=x1+inc;
end;
x1:=0;
oldx:=160;oldy:=100;
while x1<1.0 do
begin
     y1:=sqrt(x1);
     xplot:=160+round(x1*fx1);
     yplot:=100+round(y1*fy1);
     customdraw(oldx,oldy,xplot,yplot,1);
     oldx:=xplot;oldy:=yplot;
```

```
    x1:=x1+inc;
end;
oldx:=160;oldy:=100;
theta:=-pi/2;
(* sweep theta over 180 degrees and plot gain pattern *)
while theta<pi/2 do
begin
    x:=pi*sin(theta);
    gain:=sqr(2*bessel(x)/x);
    xplot:=160+round(gain*cos(theta)*fx);
    yplot:=100-round(gain*sin(theta)*fy);
    customdraw(oldx,oldy,xplot,yplot,2);
    oldx:=xplot;oldy:=yplot;
    theta:=theta+inc;
end;
gotoXY(1,1);
writeln('FREQ = ',f:0:1,' GHz');
writeln('DIAMETER = ',diameter:0:1,' m');
gotoXY(1,20);
writeln('MAX GAIN = ',max:0:1,' dB');
writeln;writeln('FIRST SIDELOBE LEVEL = ',max-17.6:0:1,' dB');
end;
(**)
(* plot gain pattern of dipole array *)
program PhasedArray;
type legal=set of char;
var InputError (* illegal numeric input *):boolean;
    integerdata (* integer input data *):integer;
    realdata (* real input data *):real;
    legalinteger,legalreal (* valid input sets *):legal;
    inc (* plotting resolution *),
    fy,fx (* scaling factors *),
    m,mag (* plot magnification variables *),
    theta,phi (* spherical coordinate angles *),
    leng (* length of elements *),
    k (* particular direction cosine *),
    AF,AF1,AF2 (* array factors *),
    PF (* pattern factor *),
    EFtheta,EFphi (* element factors *),
    Gtheta,Gphi (* gain for theta and phi components *):real;
    n,i,j (* loop variables *),
    cd (* vector current direction *),
    dist (* current distribution *),
    x,y (* # subarrays in x and y directions *),
    xplot,yplot,oldx,oldy (* plotting coordinates *):integer;
    xnum,ynum (* # elements in a subarray *),
    xspc,yspc (* spacing between elements *),
    xphase,yphase (* phase taper between elements *),
    kdx,kdy,gammax,gammay (* array factor components *),
    numx,denx,numy,deny (* interim results *),
    AFx,AFy (* interim array factors *):array[1..10] of real;
    title,subtitle (* polar plot labels *):string[20];
    flag (* plot first point flag *):boolean;
```

```
   myfile (* dummy file *):file;
(* * * * * * * * * * * * * * * * * * * * * *)
§$I A:custom.past
§$I A:noise.past
§$I A:integer.past
§$I A:real.past
(* * * * * * * * * * * * * * * * * * * * * *)
(* procedure/function declaration area *)
(* * * * * * * * * * * * * * * * * * * * * *)
(* define direction cosines *)
function kx(theta,phi:real):real;
begin
     kx:=sin(theta)*cos(phi);
end;
(**)
function ky(theta,phi:real):real;
begin
     ky:=sin(theta)*sin(phi);
end;
(**)
function kz(theta,phi:real):real;
begin
     kz:=cos(theta);
end;
(* define both components of element factor *)
function ZEFtheta(theta,phi:real):real;
begin
     ZEFtheta:=-sin(theta);
end;
(**)
function ZEFphi(theta,phi:real):real;
begin
     ZEFphi:=0.0;
end;
(**)
function YEFtheta(theta,phi:real):real;
begin
     YEFtheta:=cos(theta)*sin(phi);
end;
(**)
function YEFphi(theta,phi:real):real;
begin
     YEFphi:=cos(phi);
end;
(**)
function XEFtheta(theta,phi:real):real;
begin
     XEFtheta:=cos(theta)*cos(phi);
end;
(**)
function XEFphi(theta,phi:real):real;
begin
     XEFphi:=-sin(phi);
```

```
end;
(* define pattern factors    *)
function pattern1(length,k:real):real;
begin
     pattern1:=1.0;
end;
(**)
function pattern2(leng,k:real):real;
begin
     if k=0 then pattern2:=1.0 else
     pattern2:=leng*sin(pi*leng*k)/(pi*leng*k);
end;
(**)
function pattern3(leng,k:real):real;
begin
     if k=0 then pattern3:=1.0 else
     pattern3:=sqr(sin(pi*leng/2*k)/(pi*leng/2*k))*leng/2;
end;
(**)
function pattern4(leng,k:real):real;
begin
     pattern4:=cos(pi*leng*k)*(2*leng/pi)/(1-sqr(2*leng*k));
end;
(**)
function pattern5(leng,k:real):real;
begin
     pattern5:=(sin(pi*leng*k)/(1-sqr(leng*k)))*leng/pi;
end;
(**)
function pattern6(leng,k:real):real;
begin
     if k=0 then pattern6:=1.0 else
     pattern6:=(leng/2)*(sin(pi*leng*k)/(pi*leng*k))/(1-sqr(leng*k));
end;
(**)
function pattern7(leng,k:real):real;
begin
     pattern7:=sqr(cos(pi*leng/2*k))/(1-sqr(leng*k))*2*leng/pi;
end;
(* get numerical values for direction cosines and element factors *)
procedure casey;
begin
case cd of
     1:begin
            k:=kx(theta,phi);
            EFtheta:=XEFtheta(theta,phi);
            EFphi:=XEFphi(theta,phi);
       end;
     2:begin
            k:=ky(theta,phi);
            EFtheta:=YEFtheta(theta,phi);
            EFphi:=YEFphi(theta,phi);
       end;
```

```
     3:begin
          k:=kz(theta,phi);
          EFtheta:=ZEFtheta(theta,phi);
          EFphi:=ZEFphi(theta,phi);
       end;
end;
case dist of (* get numerical values for pattern factors *)
     1:PF:=pattern1(leng,k);
     2:PF:=pattern2(leng,k);
     3:PF:=pattern3(leng,k);
     4:PF:=pattern4(leng,k);
     5:PF:=pattern5(leng,k);
     6:PF:=pattern6(leng,k);
     7:PF:=pattern7(leng,k);
end;
end;
(* pattern computation procedure *)
procedure compute;
begin
AF1:=1.0;
(* compute x array factor *)
for i:=1 to x do
begin
     gammax[i]:=kdx[i]*kx(theta,phi)+xphase[i];
     numx[i]:=sin(xnum[i]*gammax[i]/2);
     denx[i]:=xnum[i]*sin(gammax[i]/2);
     if denx[i]<>0 then
     begin
          AFx[i]:=abs(numx[i]/denx[i]);
          AF1:=AF1*AFx[i];
     end;
end;
AF2:=1.0;
(* compute y array factor *)
for j:=1 to y do
begin
     gammay[j]:=kdy[j]*ky(theta,phi)+yphase[j];
     numy[j]:=sin(ynum[j]*gammay[j]/2);
     deny[j]:=ynum[j]*sin(gammay[j]/2);
     if deny[j]<>0 then
     begin
          AFy[j]:=abs(numy[j]/deny[j]);
          AF2:=AF2*AFy[j];
     end;
end;
AF:=AF1*AF2; (* total array factor = product of individual array factors *)
(* use pattern multiplication to get gain pattern *)
Gtheta:=AF*PF*abs(EFtheta);
Gphi:=AF*PF*abs(EFphi);
end;
(* routine to draw 6 polar plots   *)
procedure plotit;
var loop,arg:real;
```

```
begin
m:=1.0;
while m<>0 do
begin
     hires;
     hirescolor(lightblue);
     customdraw(0,100,639,100,1);
     customdraw(320,0,320,199,1);
     (* choose appropriate plane,component,and titles *)
     case n of
          1,4:begin
                   theta:=pi/2;phi:=0;
                   title:='THETA = pi/2 PLANE';
                   if n=1 then subtitle:='theta component' else
                   subtitle:='phi component';
              end;
          2,5:begin
                   theta:=0;phi:=0;
                   title:='PHI = 0 PLANE';
                   if n=2 then subtitle:='theta component' else
                   subtitle:='phi component';
              end;
          3,6:begin
                   theta:=0;phi:=pi/2;
                   title:='PHI = pi/2 PLANE';
                   if n=3 then subtitle:='theta component' else
                   subtitle:='phi component';
              end;
     end;
     writeln(title);
     writeln(subtitle);
     inc:=0.05;
     loop:=0;
     flag:=false;
     (* sweep angle and perform plotting *)
     while loop<=2*pi do
     begin
          if ((n=1) or (n=4)) then phi:=phi+inc else theta:=theta+inc;
          casey;
          compute;
          if n<=3 then arg:=Gtheta else arg:=Gphi;
          xplot:=320+round(arg*cos(loop-pi/2)*fx*m);
          yplot:=100-round(arg*sin(loop-pi/2)*fy*m/2);
          (* plot first point, then draw lines between successive points *)
          if flag=false then
          begin
               flag:=true;
               customplot(xplot,yplot,1);
               oldx:=xplot;oldy:=yplot;
               end else
          begin
               customdraw(oldx,oldy,xplot,yplot,1);
               oldx:=xplot;oldy:=yplot;
```

```
            end;
        loop:=loop+inc;
        end;
repeat
        write(' ENTER MAGNIFICATION FACTOR :');
        GetReal;
        mag:=realdata;
        if not (mag>=0) then InputError:=true;
        if InputError=true then noise;
until InputError=false;
m:=m*mag;
end;
end;
(* * * * * * * * * * * * *)
(* main array computation section *)
(* * * * * * * * * * * * *)
begin
legalinteger:=['0'..'9'];
fy:=200;fx:=440;
clrscr;
assign(myfile,'a:master.com');
repeat
write(' ENTER LENGTH OF ELEMENTS (units of wavelength) :');
GetReal;
leng:=realdata;
if not ((leng>0) and (leng<4)) then InputError:=true;
if InputError=true then noise;
until InputError=false;
repeat
        write(' ENTER VECTOR CURRENT DIRECTION  (1=x,2=y,3=z) :');
        GetInteger;
        cd:=integerdata;
        if not (cd in [1..3]) then InputError:=true;
        if InputError=true then noise;
until InputError=false;
repeat
        writeln;writeln;
        writeln(' 1. UNIFORM');
        writeln(' 2. RECTANGULAR PULSE');
        writeln(' 3. TRIANGULAR  PULSE');
        writeln(' 4. COSINE');
        writeln(' 5. SINE');
        writeln(' 6. COSINE-SQUARED');
        writeln(' 7. ABS(SINE)');
        writeln;writeln(' SPECIFY CURRENT DISTRIBUTION : ');
        GetInteger;
        dist:=integerdata;
        if not(dist in [1..7]) then InputError:=true;
        if InputError=true then noise;
until InputError=false;
writeln;
repeat
        write(' ENTER NUMBER OF X-DIRECTED SUBARRAYS :');
```

```
      GetInteger;
      x:=integerdata;
      if x>10 then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      write(' ENTER NUMBER OF Y-DIRECTED SUBARRAYS :');
      GetInteger;
      y:=integerdata;
      if y>10 then InputError:=true;
      if InputError=true then noise;
until InputError=false;
for i:=1 to x do
begin
     repeat
          write(' ENTER # ELEMENTS IN X',i,' :');
          GetReal;
          xnum[i]:=realdata;
          if (xnum[i]<=0) then InputError:=true;
          if InputError=true then noise;
     until InputError=false;
     repeat
          write(' ENTER ELEMENT SPACING FOR X',i,'  (units of wavelength) :');
          GetReal;
          xspc[i]:=realdata;
          if not ((xspc[i]>0) and (xspc[i]<1)) then InputError:=true;
          if InputError=true then noise;
     until InputError=false;
     repeat
          write(' ENTER PHASE TAPER FOR X',i,'  (units of pi) :');
          GetReal;
          xphase[i]:=realdata;
          if not ((abs(xphase[i])>=0) and (abs(xphase[i])<=2)) then
          InputError:=true;
          if InputError=true then noise;
     until InputError=false;
     kdx[i]:=2*pi*xspc[i];
     xphase[i]:=xphase[i]*pi;
end;
writeln;
for j:=1 to y do
begin
     repeat
          write(' ENTER # ELEMENTS IN Y',j,'  :');
          GetReal;
          ynum[j]:=realdata;
          if (ynum[j]<=0) then InputError:=true;
          if InputError=true then noise;
     until InputError=false;
     repeat
          write(' ENTER ELEMENT SPACING FOR Y',j,'  (units of wavelength) :');
          GetReal;
          yspc[j]:=realdata;
```

```
            if not ((yspc[j]>0) and (yspc[j]<1)) then InputError:=true;
            if InputError=true then noise;
        until InputError=false;
        repeat
            write(' ENTER PHASE TAPER FOR Y',j,'  (units of pi) :');
            GetReal;
            yphase[j]:=realdata;
            if not ((abs(yphase[j])>=0) and (abs(yphase[j])<=2)) then
            InputError:=true;
            if InputError=true then noise;
        until InputError=false;
        kdy[j]:=2*pi*yspc[j];
        yphase[j]:=yphase[j]*pi;
end;
for n:=1 to 6 do plotit;
execute(myfile);
end.
(**)
(* compute atmospheric attenuation *)
function air(freq,elev:real):real;
var x (* normalized frequency *),
    y (* log of attenuation *),
    f (* log of frequency *),
    atten (* frequency-dependent part of atmospheric attenuation *):real;
begin
f:=log10(freq);
if ((f>=0) and (f<1.34)) then
begin
     x:=f/1.4;
     if ((f>=0) and (f<0.48)) then y:=7.823e-2*f-1.504;
     if ((f>=0.48) and (f<1.0)) then y:=3.223e-1*f-1.614;
     if f>=1.0 then y:=52.3*pwr(x,3)-1.119e2*sqr(x)+80.67*x-20.86;
end;
if ((f>=1.34) and (f<1.6)) then
begin
     x:=(f-1.3)/0.5;
     y:=3.323*sqr(x)-2.42*x-0.175;
end;
if ((f>=1.6) and (f<=1.78)) then
begin
     x:=(f-1.6)/0.2;
     y:=3.25*sqr(x)-0.1298*x-0.425;
end;
if ((f>=1.78) and (f<=2.0)) then
begin
     x:=(f-1.78)/0.22;
     if ((f>=1.78) and (f<1.81)) then y:=-3.331*f+7.979;
     if ((f>=1.81) and (f<1.82)) then y:=-1.031e2*sqr(x)+10.15*x+2.483;
     if f>=1.82 then y:=5.114*pwr(x,4)-16.58*pwr(x,3)+20.43*sqr(x)-
     11.19*x+2.373;
end;
atten:=pwr(10,y);
air:=-atten/sin(elev*dr);
```

```
end;
(**)
(* compute azimuth and elevation angles *)
procedure azel;
var alt (* satellite altitude *),
    lat1 (* latitude of first ground site *),
    long1 (* longitude of first ground site *),
    lat2 (* latitude of second ground site *),
    long2 (* longitude of second ground site *),
    lats (* latitude of subsatellite point *),
    longs (* longitude of subsatellite point *),
    r1,r2 (* ground site-subsatellite point great circle distances *),
    tilt1,tilt2 (* tilt angles *),
    p (* altitude normalized w/resp to Earth radius *),
    az1,az2 (* azimuth angles *),
    elev1,elev2 (* elevation angles *):real;
begin
textcolor(white);
textbackground(blue);
clrscr;
getaltitude;
alt:=realdata;
p:=1+alt/Re;
writeln;writeln(' ENTER COORDINATES OF FIRST GROUND SITE  (deg)');
repeat
     write('     LATITUDE : ');
     GetReal;
     lat1:=realdata;
     if (abs(lat1)>90) then InputError:=true;
     if InputError=true then noise;
until InputError=false;
lat1:=lat1*dr;
repeat
     write('     LONGITUDE : ');
     GetReal;
     long1:=realdata;
     if (abs(long1)>90) then InputError:=true;
     if InputError=true then noise;
until InputError=false;
long1:=long1*dr;
writeln;writeln(' ENTER COORDINATES OF SECOND GROUND SITE  (deg) :');
repeat
     write('     LATITUDE : ');
     GetReal;
     lat2:=realdata;
     if (abs(lat2)>90) then InputError:=true;
     if InputError=true then noise;
until InputError=false;
lat2:=lat2*dr;
repeat
     write('     LONGITUDE : ');
     GetReal;
     long2:=realdata;
```

```
      if (abs(long2)>90) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
long2:=long2*dr;
writeln;writeln(' ENTER COORDINATES OF SUB-SATELLITE POINT  (deg)');
repeat
      write('     LATITUDE : ');
      GetReal;
      lats:=realdata;
      if (abs(lats)>90) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
lats:=lats*dr;
repeat
      write('     LONGITUDE : ');
      GetReal;
      longs:=realdata;
      if (abs(longs)>90) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
longs:=longs*dr;
r1:=arccos(cos(lat1-lats)*cos(long1-longs));
r2:=arccos(cos(lat2-lats)*cos(long2-longs));
(* take all cases into account and adjust azimuth accordingly *)
if lat1=lats then if long1-longs<0 then az1:=90.0 else az1:=270.0
   else az1:=rd*arctan(tan(long1-longs)/sin(lat1-lats));
if lat2=lats then if long2-longs<0 then az2:=90.0 else az2:=270.0
   else az2:=rd*arctan(tan(long2-longs)/sin(lat2-lats));
if lat1-lats>0 then az1:=az1+180.0;
if lat2-lats>0 then az2:=az2+180.0;
if az1<0 then az1:=az1+360.0;
if az2<0 then az2:=az2+360.0;
tilt1:=arctan(sin(r1)/(p-cos(r1)));
tilt2:=arctan(sin(r2)/(p-cos(r2)));
elev1:=arccos(p*sin(tilt1));
elev2:=arccos(p*sin(tilt2));
(* store elevation angles and ground site latitudes *)
data.alpha.up1:=rd*elev1;
data.alpha.dn1:=rd*elev2;
data.lat.up1:=lat1;
data.lat.dn1:=lat2;
havealpha:=true;
textcolor(lightmagenta);
writeln;
writeln(' FIRST GROUND STATION');
writeln;writeln('     azimuth = ',az1:0:2,' deg');
writeln('     elevation = ',elev1*rd:0:2,' deg');
writeln;writeln;writeln(' SECOND GROUND STATION');
writeln;writeln('     azimuth = ',az2:0:2,' deg');
writeln('     elevation = ',elev2*rd:0:2,' deg');
end;
(**)
(* compute steady-state temperature *)
```

```
procedure blackbody;
const PHIsun=3.91e26; (* average flux emitted from sun *)
var z (* satellite altitude *),
    TotArea (* total satellite surface area *),
    ExpArea (* percent of total area exposed to sun *),
    Eorbit (* irradiance at the satellite's orbit *),
    Tsat (* steady-state satellite temperature *):real;
begin
textbackground(blue);
clrscr;
getaltitude;
z:=realdata;
repeat
     writeln;
     write(' ENTER TOTAL SATELLITE SURFACE AREA (m^2) :');
     GetReal;
     TotArea:=realdata;
     if TotArea<=0 then InputError:=true;
     if InputError=true then noise;
until InputError=false;
repeat
     writeln;
     write(' ENTER PERCENT OF TOTAL SURFACE AREA EXPOSED TO SUN :');
     GetReal;
     ExpArea:=realdata;
     if not ((ExpArea>0) and (ExpArea<=100)) then InputError:=true;
     if InputError=true then noise;
until InputError=false;
ExpArea:=ExpArea/100*TotArea;
Eorbit:=PHIsun/(4*pi*sqr(1.5e11-z));
Tsat:=sqrt(sqrt(Eorbit*ExpArea/(TotArea*5.67e-8)));
textcolor(lightmagenta);
writeln;writeln;writeln('Steady-State Temperature = ',Tsat:0:2,' deg K');
end;
(**)
(* compute Shannon rate and Nyquist rate *)
procedure capacity(bandwidth,Pr,Teq,rate,M:real);
var shannon (* Shannon rate *),
   nyquist (* Nyquist rate *):real;
begin
shannonflag:=false;
nyquistflag:=false;
shannon:=bandwidth*1e6*log2(1+alog(Pr)/(bandwidth*1e6*k*Teq))/1e6;
nyquist:=2*bandwidth*1e6*log2(M)/1e6;
(* use Shannon rate if it is limiting factor *)
if shannon<nyquist then
begin
     limit:=shannon;
     shannonflag:=true;
end else
(* use Nyquist rate if it is limiting factor *)
begin
     limit:=nyquist;
```

```
     nyquistflag:=true;
end;
(* otherwise use previously computed data rate *)
if limit>rate then
begin
     limit:=rate;
     shannonflag:=false;
     nyquistflag:=false;
end;
end;
(**)
(* compute circular orbit parameters *)
procedure circular;
var h (* satellite altitude *),
    alpha (* minimum elevation angle *),
    theta (* subtended angle *),
    velocity (* satellite velocity *),
    period (* orbital period *),
    coverage (* area of coverage *),
    Te (* Earth's rotation period *),
    visibility (* duration of satellite visibility *):real;
begin
textbackground(blue);
clrscr;
getaltitude;
h:=realdata;
velocity:=sqrt(mu/(Re+h));
period:=2*pi/sqrt(mu)*pwr(Re+h,1.5)/3600;
repeat
      writeln;
      write(' ENTER MINIMUM TERMINAL ELEVATION ANGLE (deg) :');
      GetReal;
      alpha:=realdata;
      if ((alpha<=0) or (alpha>360.0)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      writeln;
      write(' [R]etrograde or [P]rograde orbit ? ');
      readln(letter);
      if not (letter in ['p','P','r','R']) then noise;
until letter in ['p','P','r','R'];
if letter in ['p','P'] then Te:=-24.0 else Te:=24.0;
theta:=arccos(Re*cos(alpha*dr)/(Re+h))*rd-alpha;
coverage:=2*pi*sqr(Re)*(1-cos(theta*dr))/1e6;
visibility:=2*theta*period/(360*(1+period/Te));
textcolor(lightmagenta);
writeln;writeln;
writeln('Velocity = ',velocity:0:2,' m/sec');
writeln('Orbital Period = ',period:0:2,' hr');
writeln('Coverage Area = ',coverage:8,' sq km');
write('Duration of Visibility = ');
if h=3.5784e7 then writeln('CONTINUOUS') else writeln(visibility:0:2,' hr');
```

```
end;
(**)
(* compute elliptical orbit parameters *)
procedure elliptical;
var apogee (* satellite apogee *),
    perigee (* satellite perigee *),
    a,b (* major and minor axes of orbit *),
    r,phi (* polar coordinates of orbiting satellite *),
    alpha (* elevation angle *),
    theta (* subtended angle *),
    VA,VP (* velocities at apogee and perigee *),
    Vavg (* average velocity *),
    velocity (* velocity at a specified reference angle phi *),
    eccentricity (* orbit eccentricity *),
    period (* orbital period *),
    coverage (* coverage area *):real;
begin
textbackground(blue);
clrscr;
repeat
      writeln;
      write(' ENTER ALTITUDE @ APOGEE (m) :');
      GetReal;
      apogee:=realdata;
      if not (apogee>0) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
apogee:=apogee+Re;
repeat
      write(' ENTER ALTITUDE @ PERIGEE (m) :');
      GetReal;
      perigee:=realdata;
      if not (perigee>0) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
perigee:=perigee+Re;
a:=(apogee+perigee)/2;
b:=sqrt(apogee*perigee);
eccentricity:=sqrt(1-(sqr(b)/sqr(a)));
va:=sqrt(mu/a*((1-eccentricity)/(1+eccentricity)));
vp:=sqrt(mu/a*((1+eccentricity)/(1-eccentricity)));
repeat
      writeln;
      write(' ENTER REFERENCE ANGLE (deg) :');
      GetReal;
      phi:=realdata;
      if not (phi>=0) and (phi<=360) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
vavg:=sqrt(vp*va);
r:=a*(1-sqr(eccentricity))/(1+eccentricity*cos(phi*dr));
velocity:=sqrt((2/r-1/a)*mu);
period:=2*pi*pwr(a,1.5)/sqrt(mu)/3600;
```

```
repeat
      writeln;
      write(' ENTER TERMINAL ELEVATION ANGLE (deg) :');
      GetReal;
      alpha:=realdata;
      if not ((alpha>=0) and (alpha<=360)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
theta:=arccos(Re*cos(alpha*dr)/(Re+r))*rd-alpha;
coverage:=2*pi*sqr(Re)*(1-cos(theta*dr))/1e6;
textcolor(lightmagenta);
writeln;writeln;
writeln('Velocity @ Apogee = ',VA:0:2,' m/sec');
writeln('Velocity @ Perigee = ',VP:0:2,' m/sec');
writeln('Average Velocity = ',Vavg:0:2,' m/sec');
if phi<>0 then writeln('Velocity @ phi = ',phi:0:0,' deg =',velocity:0:2,
' m/sec');
writeln('Eccentricity = ',eccentricity:0:2);
writeln('Orbital Period = ',period:0:2,' hr');
writeln('Coverage Area = ',coverage:8,' sq km');
end;
(**)
(* compute escape velocity at altitude *)
procedure EscapeVelocity;
var z (* launch site altitude *),
    Vesc (* escape velocity *):real;
begin
textbackground(blue);
clrscr;
repeat
      writeln;
      write(' ENTER LAUNCH SITE ALTITUDE (m) :');
      GetReal;
      z:=realdata;
      if z<0 then InputError:=true;
      if InputError=true then noise;
until InputError=false;
vesc:=sqrt(2*mu/(Re+z));
textcolor(lightmagenta);
writeln;writeln;writeln('Escape Velocity = ',Vesc:0:2,' m/sec');
end;
(**)
(* compute gain for horn *)
function gain1(d,f:real):real;
begin
     gain1:=4*pi*sqr(d*f*1e9/3e8);
end;
(**)
(* compute beamwidth for horn *)
function bw1(d,f:real):real;
begin
     bw1:=0.88*3e8/(f*1e9*d)*rd;
end;
```

```
(**)
(* compute gain for helix *)
function gain2(d,l,f:real):real;
begin
     gain2:=15.0*pi*d*l*sqr(f*1e9/3e8);
end;
(**)
(* compute beamwidth for helix *)
function bw2(d,l,f:real):real;
begin
     bw2:=52.0*3e8/(f*1e9*pi*d*sqrt(l));
end;
(**)
(* compute gain for parabolic dish *)
function gain3(d,f:real):real;
begin
     gain3:=sqr(pi*d*f*1e9/3e8);
end;
(**)
(* compute beamwidth for parabolic dish *)
function bw3(d,f:real):real;
begin
     bw3:=1.02*3e8/(f*1e9*d)*rd;
end;
(**)
(* compute gain for phased dipole array *)
function gain4(n,d,f:real):real;
begin
     gain4:=pi*n*d*f*1e9/(1.4*3e8);
end;
(**)
(* compute beamwidth for phased dipole array *)
function bw4(n,d,f:real):real;
begin
     bw4:=50.0*3e8/(n*d*f*1e9);
end;
(**)
(* compute required beamwidth for global coverage *)
function bw5(alt,elev:real):real;
begin
     bw5:=(2*arcsin(Re*cos(elev*dr)/(Re+alt)))*rd;
end;
(**)
(* compute required gain for global coverage *)
function gain5(bw:real):real;
var c:real;
begin
     c:=1.27; (* typical value *)
     gain5:=sqr(pi*c/(bw*dr));
end;
(**)
(* compute required beamwidth for spot coverage *)
function bw6(alt,nadir:real):real;
```

```
begin
     bw6:=(2*arctan(nadir/(2*alt)))*rd;
end;
(**)
(* compute required gain for spot coverage *)
function gain6(alt,nadir:real):real;
var c:real;
begin
     c:=1.27; (* typical value *)
     gain6:=sqr(pi*c*alt/nadir);
end;
(**)
(* compute EIRP *)
function isotropic(Pout,TxGain,TxCircuitLoss:real):real;
begin
     isotropic:=10*log10(Pout)+TxGain+TxCircuitLoss;
end;
(**)
(* compute equivalent temperature *)
function eqtemp(RxAntTemp,NoiseFigure,RxCircuitLoss:real):real;
begin
     eqtemp:=RxAntTemp+(NoiseFigure/alog(RxCircuitLoss)-1.0)*290.0;
end;
(**)
(* compute G/T figure of merit *)
function figmerit(RxGain,Teq:real):real;
begin
     figmerit:=RxGain-10*log10(Teq);
end;
(**)
(* compute received useful power *)
function power(Pout,TxCircuitLoss,TxGain,TxEfficiency,PathLoss,AirAtten,
               RainAtten,RxEfficiency,RxGain,RxCircuitLoss:real):real;
begin
     power:=10*log10(Pout)+TxCircuitLoss+TxGain+10*log10(TxEfficiency)+
            PathLoss+AirAtten+RainAtten+10*log10(RxEfficiency)+RxGain+
            RxCircuitLoss;
end;
(**)
(* compute CNR *)
function cnratio(Pr,Teq:real):real;
begin
     cnratio:=Pr-10*log10(k*Teq);
end;
(**)
(* compute available Eb/No *)
function EbNoAvail(Pr,Teq,datarate:real):real;
var No (* noise power density *):real;
begin
     No:=k*Teq;
     EbNoAvail:=Pr-10*log10(No*datarate*1e6);
end;
(**)
```

```
(* compute bit error rate *)
function biterror(modtype:integer;m,AvailEbNo:real):real;
begin
     AvailEbNo:=alog(AvailEbNo);
     if (((m>1) and (AvailEbNo*log2(m)<=86.0)) or (AvailEbNo<=86.0)) then
     case modtype of
          1,2,3,4:biterror:=q(sqrt(2*AvailEbNo));
          5:biterror:=q(sqrt(2*AvailEbNo*log2(m))*sin(pi/m))/log2(m);
          6,9:biterror:=q(sqrt(AvailEbNo));
          7:biterror:=0.5*exp(-0.5*AvailEbNo);
          8:biterror:=m/4.0*exp(-AvailEbNo*log2(m));
          10:biterror:=((m-1)/m*q(sqrt(6*log2(m)*AvailEbNo/(sqr(m)-1.0))))/
          log2(m);
          11:biterror:=0.5*exp(-AvailEbNo);
     end else biterror:=0;
     nosound;
end;
(**)
(* compute required Eb/No *)
function EbNoReq(modtype:integer;m,BER:real):real;
var EbNo:real;
begin
     case modtype of
          1,2,3,4:EbNo:=sqr(qinverse(BER))/2.0;
          5:EbNo:=sqr(qinverse(log2(m)*BER)/sin(pi/m))/(2.0*log2(m));
          6,9:EbNo:=sqr(qinverse(BER));
          7:EbNo:=-2.0*ln(2.0*BER);
          8:EbNo:=-ln(4.0/m*BER)/log2(m);
          10:EbNo:=(sqr(m)-1.0)/(6.0*log2(m))*sqr(qinverse(m/(m-1)*
          log2(m)*BER));
          11:EbNo:=-ln(2*BER);
     end;
EbNoReq:=10*log10(EbNo);
nosound;
end;
(**)
(* compute max datarate *)
function maxrate(Pr,Teq,ReqEbNo:real):real;
var No (* noise power density *):real;
begin
     No:=k*Teq;
     maxrate:=(alog(Pr)/(No*alog(ReqEbNo)))/1e6;
end;
(**)
(* compute rainfall attenuation *)
function rain(rainrate,f,elev,alt,lat:real):real;
var x (* normalized frequency *),
    y (* dummy variable *),
    a,b (* coefficients a(f) and b(f) *),
    gamma (* empirical constant *),
    MeltLayer (* melting layer height *),
    StormHeight (* effective storm height *),
    PathLength (* path length through storm *),
```

```
    EffDist (* total effective distance *):real;
begin
if rainrate>0 then
begin
(* * * * * * * * * * * * * * *)
(* calculate a(f) and b(f) *)
(* * * * * * * * * * * * * * *)
if rainrate<=25.0 then
begin
     if f<=10.0 then
     begin
          x:=(f-1.0)/9.0;
          a:=3.475e-3*pwr(x,6)-1.802e-2*pwr(x,5)+2.768e-2*pwr(x,4)-
          1.036e-2*pwr(x,3)+7.878e-3*sqr(x)+9.815e-4*x+6.508e-5;
     end else
     begin
          x:=f/100.0;
          a:=-5.231*pwr(x,6)+20.12*pwr(x,5)-28.12*pwr(x,4)+15.28*pwr(x,3)-
          1.067*sqr(x)+0.1094*x-1.707e-3;
     end;
     if f<8.0 then
     begin
          x:=(f-1.0)/7.0;
          b:=1.238*pwr(x,5)-2.749*pwr(x,4)+1.495*pwr(x,3)+7.341e-2*sqr(x)+
          0.2389*x+0.8908;
     end else
     begin
          x:=(f-8.0)/92.0;
          b:=3.734*pwr(x,8)-5.39*pwr(x,7)-1.103*pwr(x,6)+0.2915*pwr(x,5)+
          7.045*pwr(x,4)-5.547*pwr(x,3)+1.116*sqr(x)-0.5919*x+1.188;
     end;
end else
begin
     if f<=10.0 then
     begin
          x:=(f-1.0)/9.0;
          a:=1.453e-2*pwr(x,3)-5.037e-3*sqr(x)+1.9e-3*x+4.672e-5;
     end else
     begin
          x:=f/100.0;
          a:=28.64*pwr(x,10)-49.08*pwr(x,9)-24.49*pwr(x,8)+57.94*pwr(x,7)+
          27.15*pwr(x,6)-37.87*pwr(x,5)-27.01*pwr(x,4)+29.23*pwr(x,3)-
          3.764*sqr(x)+0.2253*x-2.531e-3;
     end;
     if ((f)>=1.0) and (f<=6.0)) then
     begin
          x:=(f-1.0)/5.0;
          b:=1.719*pwr(x,6)-4.176*pwr(x,5)+3.083*pwr(x,4)-0.9744*pwr(x,3)+
          0.4254*sqr(x)+0.2611*x+0.9468;
     end else if ((f>6.0) and (f<=30.0)) then
     begin
          x:=(f-6.0)/24.0;
          b:=4.418*pwr(x,5)-10.51*pwr(x,4)+8.005*pwr(x,3)-1.783*sqr(x)-
```

```
            0.4557*x+1.288;
        end else
        begin
            x:=(f-30.0)/70.0;
            b:=0.5882*pwr(x,4)-1.709*pwr(x,3)+1.858*sqr(x)-0.927*x+0.964;
        end;
end;
(* * * * * * * * * * * * * * * * * * *)
(* calculate effective distance  *)
(* * * * * * * * * * * * * * * * * * *)
if abs(lat)<=30.0 then MeltLayer:=4.8 else MeltLayer:=7.8-0.1*abs(lat);
if rainrate<=10.0 then StormHeight:=MeltLayer else
StormHeight:=MeltLayer+log10(rainrate/10.0);
PathLength:=(StormHeight-alt/1e3)/sin(elev*dr);
gamma:=1.0/22.0;
if rainrate<=10.0 then EffDist:=PathLength else
begin
     y:=gamma*b*ln(rainrate/10.0)*cos(elev*dr);
     EffDist:=(1-exp(-y*PathLength))/y;
end;
(* compute total rainfall attenuation *)
rain:=-a*pwr(rainrate,b)*EffDist;
end else rain:=0;
end;
(**)
(* ground station antenna analysis *)
procedure antenna1;
var gain (* antenna gain *),
    BW (* antenna beamwidth *),
    rho (* antenna efficiency *),
    f (* frequency *),
    d,l,n (* antenna dimensions *):real;
    AntType (* antenna type *):integer;
begin
textbackground(blue);
clrscr;
writeln;writeln('A N T E N N A   A N A L Y S I S':55);
gotoXY(1,5);
writeln(title[site]:50);
gotoXY(1,10);
writeln('     1.  Horn');
writeln('     2.  Helix');
writeln('     3.  Parabolic Dish');
writeln('     4.  Phased Array of Dipoles');
repeat
     writeln;writeln;write(' ENTER ANTENNA TYPE :');
     GetInteger;
     AntType:=integerdata;
     if not (AntType in [1..4]) then InputError:=true;
     if InputError=true then noise;
until InputError=false;
repeat
     write(' ENTER ANTENNA EFFICIENCY :');
```

```
      GetReal;
      rho:=realdata;
      if not((rho>=0) and (rho<=1.0)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
if uplink=true then f:=data.freq.upl else f:=data.freq.dnl;
case AntType of
     1:begin (* horn *)
            repeat
                 write(' ENTER DIMENSION OF HORN (m) :');
                 GetReal;
                 d:=realdata;
                 if d<=0 then InputError:=true;
                 if InputError=true then noise;
            until InputError=false;
            gain:=gain1(d,f);
            BW:=bw1(d,f);
       end;
     2:begin (* helix *)
            repeat
                 write(' ENTER DIAMETER OF HELIX :');
                 GetReal;
                 d:=realdata;
                 if d<=0 then InputError:=true;
                 if InputError=true then noise;
            until InputError=false;
            repeat
                 write(' ENTER LENGTH OF HELIX :');
                 GetReal;
                 l:=realdata;
                 if l<=0 then InputError:=true;
                 if InputError=true then noise;
            until InputError=false;
            gain:=gain2(d,l,f);
            BW:=bw2(d,l,f);
       end;
     3:begin (* parabolic dish *)
            repeat
                 write(' ENTER DIAMETER OF DISH (m) :');
                 GetReal;
                 d:=realdata;
                 if d<=0 then InputError:=true;
                 if InputError=true then noise;
            until InputError=false;
            gain:=gain3(d,f);
            BW:=bw3(d,f);
       end;
     4:begin (* dipole array *)
            repeat
                 write(' ENTER # LINEAR ELEMENTS :');
                 GetReal;
                 n:=int(realdata);
                 if n<2.0 then InputError:=true;
```

```
                if InputError=true then noise;
            until InputError=false;
            repeat
                write(' ENTER ELEMENT SPACING (m) :');
                GetReal;
                d:=realdata;
                if not((d>0) and (d<1.0)) then InputError:=true;
                if InputError=true then noise;
            until InputError=false;
            gain:=gain4(n,d,f);
            BW:=bw4(n,d,f);
        end;
end;
textcolor(lightmagenta);
writeln;writeln;
gain:=10*log10(gain);
writeln('MAX GAIN = ',gain:0:2,' dB');
writeln('3-dB BEAMWIDTH = ',BW:0:3,' deg');
(* store antenna data *)
if uplink=true then with data do
begin
     AntennaType.upl:=int(AntType);
     arg1.upl:=d;
     arg2.upl:=l;
     arg3.upl:=n;
     TxGain.upl:=gain;
     beamwidth.upl:=BW;
     TxEfficiency.upl:=rho;
end else with data do
begin
     AntennaType.dnl:=int(AntType);
     arg1.dnl:=d;
     arg2.dnl:=l;
     arg3.dnl:=n;
     RxGain.dnl:=gain;
     RxEfficiency.dnl:=rho;
end;
textcolor(white);
end;
(**)
(* satellite antenna analysis *)
procedure antenna2;
var beam (* beam type *):integer;
    gain (* actual antenna gain *),
    reqgain (* required antenna gain *),
    BW (* actual antenna beamwidth *),
    reqBW (* required antenna beamwidth *),
    f (* frequency *),
    elev (* elevation angle *),
    rho (* antenna efficiency *),
    d (* dish diameter *),
    nadir (* desired nadir diameter *):real;
begin
```

```
if uplink=true then f:=data.freq.upl else f:=data.freq.dnl;
if uplink=true then elev:=data.alpha.upl else elev:=data.alpha.dnl;
textbackground(blue);
clrscr;
writeln;writeln('A N T E N N A  A N A L Y S I S':55);
gotoXY(1,5);
writeln(title[site]:50);
gotoXY(1,10);
writeln('     1.  Earth Coverage');
writeln('     2.  Spot Coverage');
repeat
     writeln;writeln;write(' ENTER DESIRED BEAM TYPE :');
     GetInteger;
     beam:=integerdata;
     if not (beam in [1..2]) then InputError:=true;
     if InputError=true then noise;
until InputError=false;
repeat
     write(' ENTER ANTENNA EFFICIENCY :');
     GetReal;
     rho:=realdata;
     if not((rho>0) and (rho<=1.0)) then InputError:=true;
     if InputError=true then noise;
until InputError=false;
repeat
     write(' ENTER ANTENNA DIAMETER (m) :');
     GetReal;
     d:=realdata;
     if d<=0 then InputError:=true;
     if InputError=true then noise;
until InputError=false;
case beam of
    1:begin (* global coverage *)
          reqBW:=bw5(data.altitude,elev);
          reqgain:=gain5(reqBW);
          BW:=bw3(d,f);
          gain:=gain3(d,f);
      end;
    2:begin (* spot coverage *)
          repeat
               write(' ENTER NADIR DIAMETER (m) :');
               GetReal;
               nadir:=realdata;
               if nadir<=0 then InputError:=true;
               if InputError=true then noise;
          until InputError=false;
          reqBW:=bw6(data.altitude,nadir);
          reqgain:=gain6(data.altitude,nadir);
          BW:=bw3(d,f);
          gain:=gain3(d,f);
      end;
end;
textcolor(lightmagenta);
```

```
writeln;writeln;
gain:=10*log10(gain);
reqgain:=10*log10(reqgain);
writeln('MAX GAIN = ',gain:0:2,' dB');
writeln('3-dB BEAMWIDTH = ',BW:0:3,' deg');
(* check if required parameters are attainable *)
if (gain<reqgain) and (uplink=false) then
begin
     textcolor(yellow);
     writeln;writeln('NOTE: Required Gain = ',reqgain:0:2,
     ' dB for 25-dB sidelobes');
end;
if (BW<reqBW) and (uplink=false) then
begin
     textcolor(yellow);
     writeln;writeln('NOTE: Required Beamwidth = ',reqBW:0:2,
     ' deg for 25-dB sidelobes');
end;
(* store antenna data *)
if uplink=true then with data do
begin
     beamtype.upl:=int(beam);
     arg4.upl:=nadir;
     arg5.upl:=d;
     RxGain.upl:=gain;
     RxEfficiency.upl:=rho;
end else with data do
begin
     beamtype.dnl:=int(beam);
     arg4.dnl:=nadir;
     arg5.dnl:=d;
     TxGain.dnl:=gain;
     TxEfficiency.dnl:=rho;
     beamwidth.dnl:=BW;
end;
textcolor(white);
end;
(**)
(* receiver analysis *)
procedure receiver;
var NF (* noise figure *),
    loss (* receiver circuit losses *),
    temp (* receiving antenna noise temperature *):real;
begin
textbackground(blue);
clrscr;
writeln;writeln('R E C E I V E R   A N A L Y S I S':56);
gotoXY(1,5);
if uplink=true then writeln(title[5]:47) else writeln(title[6]:44);
gotoXY(1,10);
repeat
      writeln;
      write(' ENTER RECEIVER AMPLIFIER NOISE FIGURE :');
```

```
      GetReal;
      NF:=realdata;
      if not (NF>=1.0) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      writeln;
      write(' ENTER RECEIVER CIRCUIT LOSSES (-dB) :');
      GetReal;
      loss:=realdata;
      if not (loss<=0) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      writeln;
      write(' ENTER RECEIVING ANTENNA TEMPERATURE (deg K) :');
      GetReal;
      temp:=realdata;
      if not (temp>0) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
(* store input data and perform computations *)
with data do
begin
     if uplink=true then
     begin
          NoiseFigure.upl:=NF;
          RxCircuitLoss.upl:=loss;
          RxAntTemp.upl:=temp;
          Teq.upl:=eqtemp(RxAntTemp.upl,NoiseFigure.upl,RxCircuitLoss.upl);
          GT.upl:=figmerit(RxGain.upl,Teq.upl);
          textcolor(lightmagenta);
          writeln;writeln(' SYSTEM EQUIVALENT TEMPERATURE = ',
          Teq.upl:0:2,' deg K');
          writeln(' G/T FIGURE OF MERIT = ',GT.upl:0:2,' dB');
     end else
     begin
          NoiseFigure.dnl:=NF;
          RxCircuitLoss.dnl:=loss;
          RxAntTemp.dnl:=temp;
          Teq.dnl:=eqtemp(RxAntTemp.dnl,NoiseFigure.dnl,RxCircuitLoss.dnl);
          GT.dnl:=figmerit(RxGain.dnl,Teq.dnl);
          textcolor(lightmagenta);
          writeln;writeln(' SYSTEM EQUIVALENT TEMPERATURE = ',
          Teq.dnl:0:2,' deg K');
          writeln(' G/T FIGURE OF MERIT = ',GT.dnl:0:2,' dB');
     end;
end;
wait;
textcolor(white);
end;
(**)
(* transmitter analysis *)
```

```
procedure transmitter;
var power (* transmitter power *),
    loss (* transmitter circuit losses *):real;
begin
textbackground(blue);
clrscr;
writeln;writeln('T R A N S M I T T E R  A N A L Y S I S':60);
gotoXY(1,5);
if uplink=true then writeln(title[5]:47) else writeln(title[6]:44);
gotoXY(1,10);
repeat
      writeln;
      write(' ENTER OUTPUT AMPLIFIER POWER LEVEL (W) :');
      GetReal;
      power:=realdata;
      if not (power>0) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      writeln;
      write(' ENTER TRANSMITTER CIRCUIT LOSSES (-dB) :');
      GetReal;
      loss:=realdata;
      if not (loss<=0) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
(* store input data and perform computations *)
with data do
begin
     if uplink=true then
     begin
          Pout.upl:=power;
          TxCircuitLoss.upl:=loss;
          EIRP.upl:=isotropic(Pout.upl,TxGain.upl,TxCircuitLoss.upl);
          textcolor(lightmagenta);
          writeln;writeln(' EIRP = ',EIRP.upl:0:2,' dB');
     end else
     begin
          Pout.dnl:=power;
          TxCircuitLoss.dnl:=loss;
          EIRP.dnl:=isotropic(Pout.dnl,TxGain.dnl,TxCircuitLoss.dnl);
          textcolor(lightmagenta);
          writeln;writeln(' EIRP = ',EIRP.dnl:0:2,' dB');
     end;
end;
textcolor(white);
wait;
end;
(**)
(* link receiving performance analysis *)
procedure RecPower;
var i (* counter *):integer;
    rate (* rain rate *):real;
```

```
begin
textbackground(blue);
for i:=1 to 2 do
begin
clrscr;
gotoXY(1,5);
writeln(title[6+i]:46);
gotoXY(1,10);
writeln('     1.  Negligible  ( < .25 mm/hr )');
writeln('     2.  Drizzle  ( .25 - 1.25 mm/hr )');
writeln('     3.  Light Rain  ( 1.25 - 12.5 mm/hr )');
writeln('     4.  Medium Rain  ( 12.5 - 25 mm/hr )');
writeln('     5.  Heavy Rain  ( 25 - 100 mm/hr )');
writeln('     6.  Tropical Downpour  ( > 100 mm/hr )');
writeln('     7.  SPECIFY EXACT RAINRATE  ( 0 - 150 mm/hr )');
repeat
      writeln;writeln;write(' ENTER RAINRATE :');
      GetInteger;
      choice:=integerdata;
      if not choice in [1..7] then InputError:=true;
      if InputError=true then noise;
until InputError=false;
case choice of
     1:rate:=0.125;
     2:rate:=0.75;
     3:rate:=6.88;
     4:rate:=18.75;
     5:rate:=62.5;
     6:rate:=125.0;
     7:begin
           repeat
                writeln;write(' ENTER RAINRATE (mm/hr) :');
                GetReal;
                rate:=realdata;
                if not((rate>=0) and (rate<=150)) then InputError:=true;
                if InputError=true then noise;
           until InputError=false;
           if rate=0 then if i=1 then data.RainAtten.upl:=0 else
           data.RainAtten.dnl:=0;
       end;
end;
if rate>0 then with data do
   if i=1 then RainAtten.upl:=rain(rate,freq.upl,alpha.upl,Ho.upl,lat.upl)
   else RainAtten.dnl:=rain(rate,freq.dnl,alpha.dnl,Ho.dnl,lat.dnl);
if i=1 then data.rainrate.upl:=rate else data.rainrate.dnl:=rate;
end;
data.AirAtten.upl:=air(data.freq.upl,data.alpha.upl);
data.AirAtten.dnl:=air(data.freq.dnl,data.alpha.dnl);
(* compute slant range,path loss,recvd power,and CNR *)
with data do
begin
     SlantRange.upl:=(sqrt(sqr(Re)*sqr(sin(alpha.upl*dr))+2*Re*data.altitude+
     sqr(data.altitude))-Re*sin(alpha.upl*dr))/1e3;
```

```
      SlantRange.dnl:=(sqrt(sqr(Re)*sqr(sin(alpha.dnl*dr))+2*Re*data.altitude+
      sqr(data.altitude))-Re*sin(alpha.dnl*dr))/1e3;
      PathLoss.upl:=-36.6-20*log10(SlantRange.upl*1e3*6.214e-4*freq.upl*1e3);
      PathLoss.dnl:=-36.6-20*log10(SlantRange.dnl*1e3*6.214e-4*freq.dnl*1e3);
      Pr.upl:=power(Pout.upl,TxCircuitLoss.upl,TxGain.upl,TxEfficiency.upl,
      PathLoss.upl,AirAtten.upl,RainAtten.upl,RxEfficiency.upl,RxGain.upl,
      RxCircuitLoss.upl);
      Pr.dnl:=power(Pout.dnl,TxCircuitLoss.dnl,TxGain.dnl,TxEfficiency.dnl,
      PathLoss.dnl,AirAtten.dnl,RainAtten.dnl,RxEfficiency.dnl,RxGain.dnl,
      RxCircuitLoss.dnl);
      CNR.upl:=cnratio(Pr.upl,Teq.upl);
      CNR.dnl:=cnratio(Pr.dnl,Teq.dnl);
end;
end;
(**)
(* link transmission performance analysis *)
procedure bitrate;
var rate (* datarate *),
    err (* bit error rate *),
    num (* number of signalling levels *):real;
    modulation (* modulation type *):integer;
begin
clrscr;
gotoXY(1,5);
if uplink=true then writeln(title[1]:50) else writeln(title[3]:50);
gotoXY(1,10);
repeat
      write(' ENTER DESIRED DATA RATE  (Mb/sec) :');
      GetReal;
      rate:=realdata;
      if rate<1e-3 then InputError:=true;
      if InputError=true then noise;
until InputError=false;
repeat
      writeln;writeln;
      writeln('    1. BPSK');
      writeln('    2. QPSK');
      writeln('    3. OQPSK');
      writeln('    4. MSK');
      writeln('    5. M-PSK');
      writeln('    6. Coherent FSK');
      writeln('    7. Non-Coherent FSK');
      writeln('    8. M-FSK');
      writeln('    9. ASK');
      writeln('   10. M-ASK');
      writeln('   11. DPSK');
      writeln;
      write(' ENTER MODULATION TYPE :');
      Get Integer;
      modulation:=integerdata;
      if not(modulation in [1..11]) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
```

```
num:=2.0;
case modulation of
     5,8,10:begin
            repeat
                 write(' ENTER M :');
                 GetReal;
                 num:=realdata;
                 if ((num<2.0) or (frac(num)<>0)) then InputError:=true;
                 if InputError=true then noise;
            until InputError=false;
            end;
end;
repeat
      write(' ENTER DESIRED BER :');
      GetReal;
      err:=realdata;
      if not((err>0) and (err<1.0)) then InputError:=true;
      if InputError=true then noise;
until InputError=false;
(* store input data and perform computations *)
with data do if uplink=true then
begin
     datarate.upl:=rate;
     modtype.upl:=int(modulation);
     M.upl:=num;
     BER.upl:=err;
     AvailEbNo.upl:=EbNoAvail(Pr.upl,Teq.upl,datarate.upl);
     ReqEbNo.upl:=EbNoReq(trunc(modtype.upl),M.upl,BER.upl);
     margin.upl:=AvailEbNo.upl-ReqEbNo.upl;
     MaxDatarate.upl:=maxrate(Pr.upl,Teq.upl,ReqEbNo.upl);
     capacity(bandwidth.upl,Pr.upl,Teq.upl,MaxDatarate.upl,M.upl);
     textcolor(lightmagenta);
     writeln;writeln(' Available Eb/No = ',AvailEbNo.upl:0:2,' dB');
     writeln(' Required Eb/No = ',ReqEbNo.upl:0:2,' dB');
     writeln(' Max Data Rate = ',limit:0:2,' Mb/sec');
     if shannonflag=true then
     begin
          noise;
          writeln('*** SHANNON RATE USED ***');
          MaxDatarate.upl:=limit;
     end;
     if nyquistflag=true then
     begin
          noise;
          writeln('*** NYQUIST RATE USED ***');
          MaxDatarate.upl:=limit;
     end;
 end else
 begin
      datarate.dnl:=rate;
      modtype.dnl:=int(modulation);
      M.dnl:=num;
      BER.dnl:=err;
```

```
        AvailEbNo.dnl:=EbNoAvail(Pr.dnl,Teq.dnl,datarate.dnl);
        ReqEbNo.dnl:=EbNoReq(trunc(modtype.dnl),M.dnl,BER.dnl);
        margin.dnl:=AvailEbNo.dnl-ReqEbNo.dnl;
        MaxDatarate.dnl:=maxrate(Pr.dnl,Teq.dnl,ReqEbNo.dnl);
        capacity(bandwidth.dnl,Pr.dnl,Teq.dnl,MaxDatarate.dnl,M.dnl);
        textcolor(lightmagenta);
        writeln;writeln(' Available Eb/No = ',AvailEbNo.dnl:0:2,' dB');
        writeln(' Required Eb/No = ',ReqEbNo.dnl:0:2,' dB');
        writeln(' Max Data Rate = ',limit:0:2,' Mb/sec');
        if shannonflag=true then
        begin
             noise;
             writeln('*** SHANNON RATE USED ***');
             MaxDatarate.dnl:=limit;
        end;
        if nyquistflag=true then
        begin
             noise;
             writeln('*** NYQUIST RATE USED ***');
             MaxDatarate.dnl:=limit;
        end;
    end;
    textcolor(white);
    if uplink=true then wait;
    end;
    (**)
```

Bibliography

1. Balanis, Constantine A. Antenna Theory - Analysis and Design. New York: Harper & Row Publishers, 1982.

2. Gagliardi, Robert M. Satellite Communications. Belmont, Ca: Lifetime Learning Publications, 1984.

3. Hansen, R. C. (editor). Microwave Scanning Antennas. Los Altos, Ca: Peninsula Publishing, 1985.

4. Ippolito, Louis J. et al. Propagation Effects Handbook for Satellite Systems Design. NASA Reference Publication 1082(03), National Aeronautics and Space Administration, Washington, D.C., Jun 83.

5. Myers, Glenford J. The Art of Software Testing. New York: John Wiley & Sons, 1979.

6. Navas, Michael J. Prediction of Power Margins on Satellite Communications Links as a Function of Local Rain Rate Statistics and Desired Availability. MS Thesis. School of Engineering, Air Force Institute of Technology (AU), Wright-Patterson AFB, Oh, Dec 1985.

7. Olsen, R. L. et al. "The aR^b Relation in the Calculation of Rain Attenuation," IEEE Transactions on Antennas and Propagation, AP-26: 318-329 (Mar 78).

8. Peters, Lawrence J. Software Design: Methods and Techniques. New York: Yourdon Press, 1981.

9. Pratt, Timothy and Charles W. Bostian. Satellite Communications. New York: John Wiley & Sons, 1986.

10. Sklar, Bernard. "What the System Link Budget Tells the System Engineer or How I Learned to Count in Decibels," presented at IEEE WESCON/78, Session 15, Los Angeles, 1978.

11. Spilker, James J. Digital Communications by Satellite. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1977.

12. Stutzman, W. L. and W. K. Dishman. "A Simple Model for the Estimation of Rain-Induced Attenuation Along Earth-Space Paths at Millimeter Wavelengths," Radio Science, 17: 1465-1476 (Nov-Dec 82).

13. Van Trees, Harry L. (editor). Satellite Communications. New York: IEEE Press, 1979.

14. Waite, Mitchell and Christopher L. Morgan. Graphics Primer for the IBM PC. Berkeley, Ca: Osborne/McGraw-Hill, 1983.

15. Wu, William W. Elements of Digital Satellite Communication, Volume 1. Rockville, Md: Computer Science Press, 1984.

16. Ziemer, Rodger E. and Roger L. Peterson. Digital Communications and Spread Spectrum Systems. New York: Macmillan Publishing Co., 1985.

VITA

Captain William C. Jackson was born on 28 July 1960 in Coral Gables, Florida. He graduated from Northwestern University in 1982 with dual degrees in Electrical Engineering and Computer Science. Following graduation he received his commission from USAF Officer Training School, and was subsequently assigned to HQ Foreign Technology Division, WPAFB, Ohio. He served there as a threat integration engineer until entering the School of Engineering, Air Force Institute of Technology, in May 1985.

Permanent Address: 2001 SW 33 Ave.
Miami, Fl.
33145

The product of this research effort is a comprehensive and integrated software package for the analysis of digital satellite links. The system is designed to run on the IBM PC, and should also run on most compatibles.

The system performs three basic classes of functions: satellite orbital analysis, antenna gain pattern plotting, and link analysis. The first class of functions includes the computation of such quantities as velocity, orbital period, and coverage area for satellites in circular and elliptical orbits. The second class of functions is concerned with plotting the gain patterns for horns, helixes, parabolic reflectors, and phased arrays of dipoles. The last class of functions represents the major thrust of the system, and entails computing such items as the G/T figure of merit, received useful power, carrier-to-noise ratio, bit error rate, maximum data rate, and power margin. Inherent within this class are mathematical models for computing the attenuation due to rainfall and atmospheric absorption.

The link budget itself appears as a color-coded display with two columns: one for the uplink path, and one for the downlink path. The user also has the capability to change certain key inputs, and then have the system automatically recompute the entire link budget with the modified data.

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

AD-A178 872

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S): AFIT/GE/ENG/86D-38	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: School of Engineering; 6b. OFFICE SYMBOL (If applicable): AFIT/ENG	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State, and ZIP Code): Air Force Institute of Technology, Wright-Patterson AFB, Ohio, 45433-6583	7b. ADDRESS (City, State, and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
Development of a Microcomputer-Based Workstation for Analysis of Digital Satellite Links (U)

12. PERSONAL AUTHOR(S)
William C. Jackson, B.S., Capt, USAF

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
MS Thesis	FROM _____ TO _____	1986 December	174

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	communication satellites, microcomputers, computer programs, computer aided design
09	02		
22	02		

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Thesis Advisor: Dale V. Hibner, Major, USAF
Department of Electrical and Computer Engineering

Approved for public release: IAW AFR 190-1
JOHN E. WOLAVER
Dean for Research and Professional Development
Air Force Institute of Technology (AU)
5 March 87

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b TELEPHONE (Include Area Code)	22c OFFICE SYMBOL
Dale V. Hibner, Major, USAF	513-255-6027	AFIT/ENG

DD Form 1473, JUN 86 — Previous editions are obsolete.

SECURITY CLASSIFICATION OF THIS PAGE
UNCLASSIFIED

END

5-87

DTIC