ANALYSIS OF MILLIMETER AND MICROWAVE INTEGRATED CIRCUITS

Thesis by

Richard C. Compton

In Partial Fulfillment of the Requirements
for the Degree of
Doctor of Philosophy

California Institute of Technology

Pasadena, California

1987

(Submitted May 4, 1987)

TO MY WIFE AND PARENTS

Acknowledgements

I would like to thank my thesis advisor Professor David Rutledge for his guidance, enthusiasm, and encouragement. I would also like to thank Ross McPhedran of the University of Sydney for his continuing help over the years. Thanks also to Professor N. Alexopoulos, and Professor N. Luhmann Jr., at the University of California, Los Angeles. I would like to acknowledge those members of our group with whom I collaborated: Zoya Popović, Kent Potter, Gabriel Rebeiz, Arthur Sheiman, Peter Tong, Wyman Williams, and Chung-en Zah. Thanks to the entire group, past and present, for the many good times. Finally, a special thanks to my mother and father and my wife Annegret for their support.

Analysis of Millimeter and Microwave Integrated Circuits

Abstract

The design and measurement of millimeter and microwave integrated circuits encompasses a diverse spectrum of interesting disciplines. A number of contrasting approaches used for theoretically and experimentally characterizing circuits in this frequency range will be presented. Rigorous methods for calculating antenna patterns and impedances of the planar bow-tie antenna used in millimeter wave receiver systems have been developed. A 94 GHz antenna measurement system and microwave scale models were constructed to confirm these theoretical predictions. Pattern measurements were also made on a linear array of bow-tie antennas, a long strip antenna, and a log-periodic antenna. In modified form, these techniques were applied to the investigation of planar array structures. An equivalent circuit model for an array of squares joined at the corners by discrete devices was formulated. This model was verified with impedance and pattern measurements. Finally, a set of analysis tools that have been assembled into an interactive computer-aided design program, called *Puff*, is discussed. *Puff* has been used in microwave laboratory classes at the California Institute of Technology, Cornell University and the University of California, Los Angeles.

TABLE OF CONTENTS

Chapter 1

Introduction

1.1 Historical perspective

Microwave frequencies cover the region 500 MHz to 30 GHz (1 cm to 60 cm) and millimeter waves range from 30 GHz to 300 GHz (1 mm to 10 mm). Heinrich Hertz is credited with being the first microwave scientist. In the 1890's he used a spark gap generator and dipole to generate and receive electromagnetic radiation believed to be as short as 6 mm [1]. In the mid 1930's, U.S. and British forces became interested in the development of radar in response to the growing offensive threat posed by the Axis powers. This initiated an investigation into microwave systems for high-resolution radar systems to detect enemy planes and ships. A major breakthrough in the development of radar was the invention of the high-power magnetron which appeared in 10-cm radar systems in 1943 [2]. These radar were used to track enemy aircraft and provide direct information for antiaircraft guns.

Progress continued after the war on improved sources and signal processing techniques. Most measurements in the 40's and 50's were performed using slotted-line—a form of transmission line which contains a moveable probe used to measure the position and magnitude of voltage maxima and minima. During this period alternative methods for measuring and guiding waves were developed. New circuit technologies evolved as a mixtures of established coaxial and waveguide microwave systems. The "flat strip" transmission line evolved from a coaxial cable where the outer conductor is deformed into a flat ground plane above which sits the center conductor in the form of a narrow strip. These circuits can be fabricated using standard printed circuit fabrication techniques, and have the advantage of light weight, small size and good high frequency performance [3]. Microwave printed circuits can not be probed in a manner analogous to the slotted-line because the probes dramatically alter the circuit behaviour. During World War II, Dicke [4] worked on the idea of circuit characterization in terms of directed scattered waves rather than voltages and currents. The scattering coefficient s_{ij} is defined as the ratio of an outgoing wave at port j to an incoming wave at port i, with all the ports terminated by the normalizing impedance. Engineers were initially reluctant to accept the s-parameter approach because the analysis was often too complex to perform with a slide rule and pen and paper. In addition, equipment to make accurate s-parameter measurements was not widely available until the late 1960's. In 1967 Hewlett Packard introduced the 8410A vector network analyzer which allowed the direct measurement of scattering parameters and helped to establish the importance of s-parameters.

The development of synchronous earth satellites, in the 60's, opened a new area of applicability for microwave circuits. Propagation through the ionosphere requires frequencies above 100 MHz, while above 30 GHz the absorption due to water vapour in the atmosphere increases rapidly. Figure 1 gives the atmospheric absorption at sea level and also indicates the major applications of microwaves. Typical satellite systems operate in the frequency range 400 MHz to 30 GHz [5].

Major advances were made in microwave circuits with the development of monolithic integrated circuits in the late 60's and early 70's. These circuits have the advantage of being, reproducible, cost effective, reliable and ideal for applications like arrays where large numbers of circuits are required. Gallium Arsenide evolved as the material of choice for these circuits—it's high resistance is well suited for low-loss passive circuit elements and transmission lines. GaAs has also become the most important material for high performance devices like FET transistors because of its high electron mobility.

In contrast to the microwave area, millimeter wave research is still in the fledgling stages. Millimeter waves promise higher resolution and broader bandwidths than microwave systems. However, the push to higher frequencies has encountered obstacles. At millimeter wavelengths it is difficult to build high power sources, low loss guiding structures and sensitive detectors. At present the main ap-

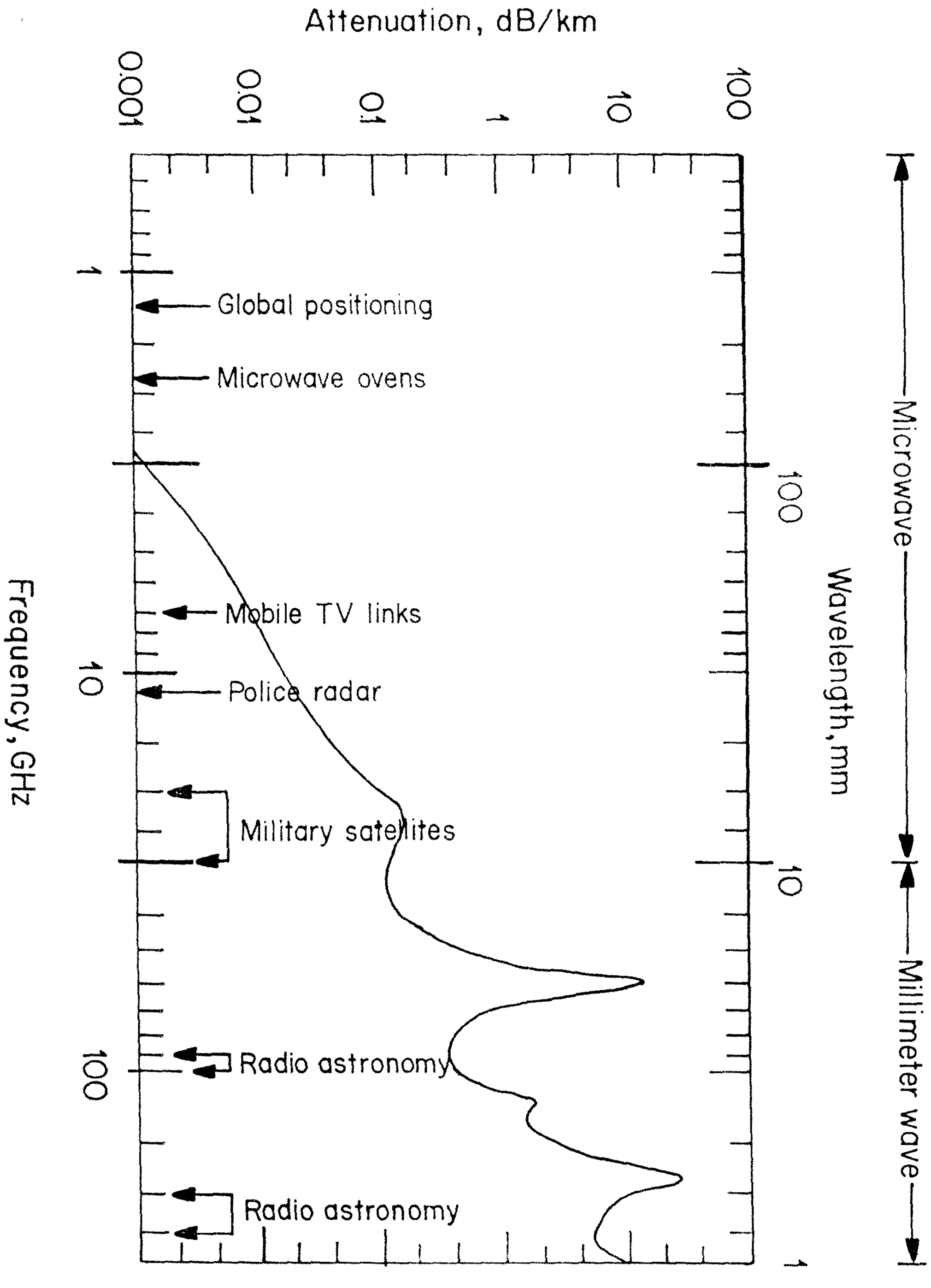


Figure 1. Atmospheric absorption [6,7,8], due mainly to molecular oxygen and water vapour, places strong restrictions on the design of millimeter and microwave systems.

plications have been scientific, in the areas of radio astronomy [9], airborne remote sensing [10], and plasma fusion diagnostics [11].

Circuit design at millimeter and microwave frequencies differ from low frequency design in many respects. High frequency circuits are small and are usually fabricated photolithographically. As a result, once the circuit is built, there is very little in the circuit than can be modified, replaced or adjusted. Furthermore, measurements of the circuits are difficult. Because they cannot be probed, parts of a circuit cannot be individually tested. These fabrication and measurement limitations make the initial design process very important. In addition, constraints such as sensitivity, noise behavior and output power make optimal circuit performance crucial. Major developments in computing resources over the past decade are changing the way engineers design and develop high frequency circuits. In recent years this development has been particularly noticeable in the area of small desk-top computing. Personal computers now provide engineers with the necessary tools for highly interactive computer aided design and measurement work-stations. Figure 2 shows a comparison of run-times for an antenna efficiency calculation [12]. The next generation of PC's that will be introduced over the next few years, will have performance approaching the present day VAX family of computers. These advances in computer speed and power are having a significant impact on the role of computers in the areas of design, measurement [13] and education [14].

1.2 Thesis outline

This thesis contains a collection of techniques used to analyze millimeter and microwave circuits. The analysis process starts with an idea or circuit observation. This is followed by the development of a theory to characterize the circuit performance. The theory results in a set of equations that are numerically solved by computer. The circuit is fabricated and an experiment devised to verify the theory. If necessary, the theory can then be modified or extended to take into account any newly observed phenomena.

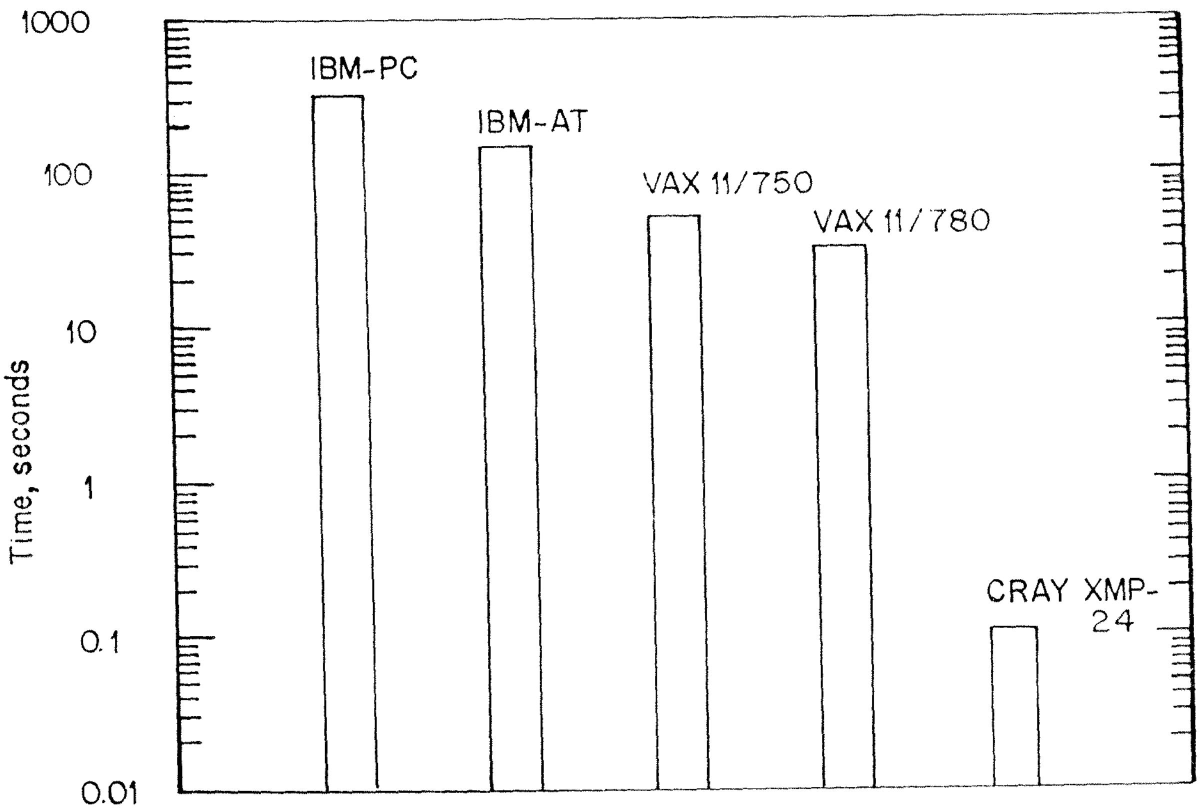


Figure 2. Computer execution times for an antenna efficiency calculation [12]. The calculations performed were sums of products of trigonometric functions weighted with complex admittances.

Nearly all millimeter-wave systems contain some form of antenna. The antenna acts as a transition for coupling radiation between free space and a signal source or detector. Essential to the design of efficient antenna is a good match of the antenna pattern and impedance between the feeding structure and free space. Integrated circuit photolithography techniques allow millimeter-wave antenna systems to be fabricated with micron tolerances. These antennas are supported by a dielectric whose presence can dramatically alter the antenna performance. A rigorous moment method for analyzing planar antennas supported by a dielectric substrate will be presented in Chapter 2. To confirm the theory a computer automated antenna range operating at 94 GHz was constructed. A bow-tie antenna, long strip antenna, log-periodic antenna and linear array were fabricated and measured. Microwave scale models were also built to measure antenna-feed impedances at 2 GHz.

Techniques for studying the performance of planar arrays are discussed in Chapter 3. These arrays consist of periodically spaced holes in a metal sheet, or a planar array of metal disks, and may be fabricated using standard photolithography. They are used primarily in millimeter and sub-millimeter wave systems as transmission and reflection filters. For these applications, it is important to know the mesh filtering characteristics (transmission as function of frequency). The need for sharp filtering characteristics has spurred investigations into meshes with different shaped holes and disks. A number of tricks for analyzing these arrays will be discussed in Chapter 3. The methods developed bear similarities with the techniques used for studying single planar antennas. The integration of active devices into these meshes opens many new exciting possibilities. Such arrays hold promise in applications of beam-steering, high-power signal sources [17], and imaging systems. Equivalent circuit techniques and measurements for an array of metal squares linked with bolometers will be discussed.

The design of complete millimeter and microwave systems are usually performed by modelling each of the components and then combining them using microwave circuit analysis techniques. In Chapter 4 a microwave computer aided design program developed for the IBM PC line of computers will be presented. The program, called *Puff* [14], has been developed primarily for education but has also found use in research applications. *Puff*'s theoretical predictions for a variety of circuits are compared with measurements made using either a network analyzer, spectrum analyzer or noise figure meter.

References

[1] H. Hertz, *Electric Waves*, Dover Publications, New York, 1962.

[2] D. K. Barton, "A half century of radar," *IEEE Trans. Microwave Theory Tech.*, **vol. MTT-32**, pp. 1161–1170, Sept. 1984.

[3] R. M. Barret, "Microwave printed circuits—a historical survey," *IRE Trans. Microwave Theory Tech.*, **vol. MTT-3**, pp. 1–20, March 1955.

[4] C. G. Montgomery, R. H. Dicke, and E. M. Purcell, *Principles of Microwave Circuits*, McGraw-Hill, New York, 1948, chap. 5.

[5] W. L. Pritchard, "The development of satellite communications," *Microwaves and RF*, **vol. 26**, pp. 304-312, March 1987.

[6] CCIR XIIIth Planary Assembly, **vol. 5**, Report 233-3, Geneva 1974.

[7] E. S. Rosenblum, "Atmospheric absorption of 10-400 KMCPS radiation: summary and bibliography to 1961," *Microwave J.*, **vol. 4**, pp. 91–96, March 1961.

[8] Microwave Applications Chart, Wavetek, Sunnyvale CA.

[9] T. G. Phillips and D. B. Rutledge, "Superconducting tunnel detectors in radio astronomy," *Scientific American*, **254**, pp. 96–102, May 1986.

[10] W. J. Wilson, R. J. Howard, A. C. Ibbott, G. S. Parks, and W. B. Ricketts, "Millimeter-wave imaging sensor," *IEEE Trans. Microwave Theory Tech.*, **vol. MTT-34**, pp. 1026–1035, Oct. 1986.

[11] N. C. Luhmann, Jr., "Instrumentation and techniques for plasma diagnostics: an overview," *Infrared and Millimeter Waves*, **chap. 1**, pp. 1–66, K. J. Button, ed., Academic Press, New York, 1983.

[12] C. Zah, R. C. Compton, and D. B. Rutledge, "Efficiencies of elementary integrated-circuit feed antennas," *Electromagnetics*, **3**, pp. 239–254, 1983.

[13] W. L. Williams, R. C. Compton, and D. B. Rutledge, "ELF: computer automation and error correction for a microwave network analyzer," submitted to *IEEE Trans. Instrum. and Meas.*

[14] R. C. Compton, W. L. Williams, Kent Potter, and D. B. Rutledge, "Puff: microwave computer aided design for education using personal computers," submitted to *IEEE Trans. Microwave Theory Tech.*

[15] R. C. Compton, R. C. McPhedran, Z. Popović, G. M. Rebeiz, P. P. Tong, and D. B. Rutledge, "Bow-tie antennas on a dielectric half space: theory and experiment," *IEEE Trans. Antennas Propagat.*, **vol. 35**, June 1987.

[16] R. C. Compton and D. B. Rutledge, "Approximation techniques for planar periodic structures," *IEEE Trans. Microwave Theory Tech.*, **MTT-33**, pp. 1083–1088, Oct. 1985.

[17] W. W. Lam, C. F. Jou, N. C. Luhmann, Jr., and D. B. Rutledge, "Diode grids for electronic beam steering and frequency multiplication," *Int. J. Infrared and Millimeter Waves*, **vol. 7**, pp. 27–41, July 1986.

Chapter 2

Planar Antennas on a Dielectric Half Space

In recent years considerable effort has focused on planar antennas supported by a thick dielectric substrate [1-10]. One such antenna is the bow-tie (Fig. 1). The main advantages of the bow-tie antenna are simple design and broadband impedance. Antennas sitting on substrates with high dielectric constant radiate most of their energy into the dielectric side. This radiation may be coupled out of the dielectric with a substrate lens. The resulting quasi-optical system does not rely on expensive precision-machined waveguide components, and the antennas may be fabricated using standard photolithography. They are suitable for coupling to millimeter-wave detectors that can be integrated into the antenna using similar photolithographic techniques. Bow-tie antennas have been used in SIS and Schottky diode mixers in the frequency range 94–466 GHz [2,3], and in linear imaging arrays [4]. These systems have already been applied in plasma diagnostics and radio astronomy [3,5,8].

The design of an antenna system requires an understanding of the impedance and the antenna patterns, along with the physical mechanisms responsible. Previous theoretical work on planar antennas has concentrated on dipole and slot elements [6,7,9,10]. In this chapter a new formulation is discussed for the rigorous calculation of the radiation pattern of a bow-tie antenna. The antenna has infinitesimal thickness and is placed on a lossless dielectric substrate. The analysis is based on a representation of the current density on the metal surface of the antenna as a sum of an imposed (quasi-static) term and a set of current modes with unknown amplitudes. Free-space fields that are expressed in terms of continuous spectra of symmetrized plane waves are matched to the current modes using the method of moments. The resulting set of equations are solved for the unknown current amplitudes which are then used to reconstruct the electric and magnetic fields in the air and dielectric regions. The magnitude of these fields are related

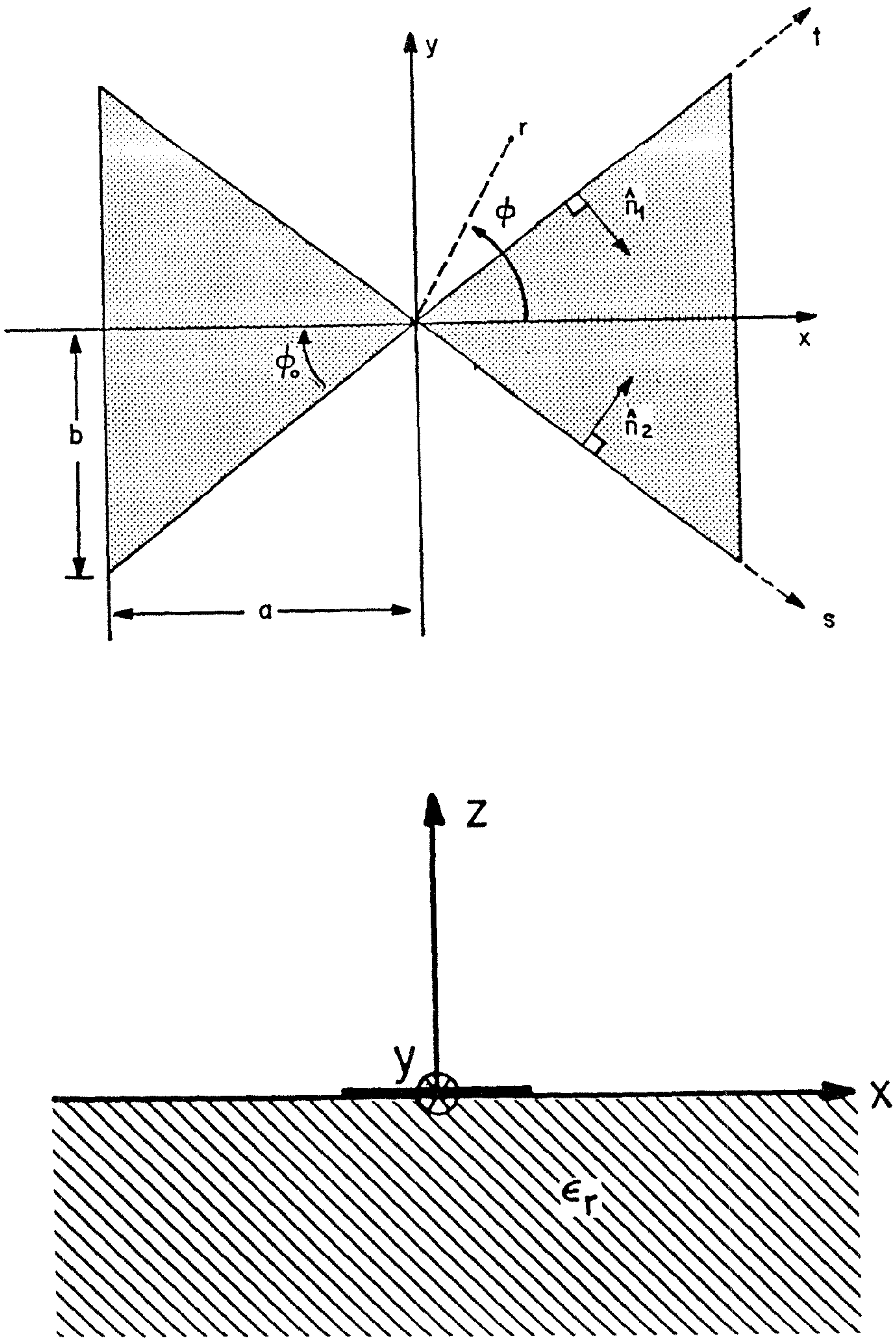


Figure 1. The geometry of a bow-tie antenna in cartesian (x, y, z) and radial coordinates (r, ϕ), and the corresponding oblique coordinates (s, t) with unit vectors $\hat{\mathbf{n}}_1, \hat{\mathbf{n}}_2$.

to the antenna pattern and impedance. Calculations show that for increasing bow length the antenna impedance spirals rapidly to a value predicted by transmission line theory. The theory also shows that the E-plane pattern of a two wavelength, sixty-degree bow-tie antenna is dominated by low-loss current modes propagating at the dielectric wave number. As the bow-tie narrows, the loss of the modes increases, and the dominant wave number tends to the quasi-static value. Pattern measurements made at 94 GHz are shown to agree well with theoretical predictions. Measurements for a long-wire antenna, a linear array of bow-tie elements, and a log-periodic antenna are also presented.

2.1 Theory of the bow-tie antenna

We analyze a bow-tie (Fig. 1) consisting of two perfectly conducting, infinitesimally thin triangular segments that are fed at the bow apex. The antenna sits on a lossless dielectric half-space with dielectric constant ϵ_r. The bow has length a, width b, and half-angle ϕ_0 . The antenna pattern may be calculated by considering the antenna as either a transmitter or receiver—the equivalence of these two situations follows from the reciprocity theorem [11]. The bow will be analyzed in terms of a transmitter fed at the bow apex. The results of this analysis were later verified by comparing them with calculations for the reciprocal case in which the antenna is considered as a receiver. The functional dependence of the feed current is determined by investigating the fields at the bow center. At distances much less than a wavelength from the feed point the fields may be described in terms of static solutions satisfying the Laplace equation. These fields are calculated analytically, using two conformal transformations to determine the bow-tie potentials.

2.1.1 Quasi-static bow-tie solution

The quasi-static impedance Z_{qs} for an infinite bow-tie, obtained by a conformal mapping that transforms the bow into a coplanar waveguide, is given by [1]

$$Z_{qs} = \eta_0 \sqrt{\frac{2}{\epsilon_r + 1}} \quad \frac{K(k)}{K'(k)} \tag{1}$$

where $K'(k)$ and $K(k)$ are elliptic integrals with argument $k = \tan^2(45° - \phi_0/2)$. The above formula agrees well with microwave impedance measurements provided the bow length is greater than a free-space wavelength [1].

A second conformal mapping [13] maps the half space above the coplanar waveguide into a parallel plate box. The known box charge distribution is transformed back to the bow geometry to obtain a $1/\sqrt{\cos^2\phi - \cos^2\phi_0}$ behavior. In the quasi-static limit the current density will have this same edge-singular behavior [14]. In cylindrical coordinates the feed current may be expressed as

$$\mathbf{J} = \frac{\pm e^{-\gamma r}/r}{\sqrt{\cos^2\phi - \cos^2\phi_0}} \quad \hat{\mathbf{r}}. \tag{2}$$

The plus sign applies to the top half of bow and the minus to the bottom half. The $e^{-\gamma r}/r$ term describes an outward-traveling current wave. The propagation constant γ may be written as $\alpha + j\beta$, where a first estimate for β will be provided by the quasi-static wave number $k_{qs} = k_0\sqrt{(\epsilon_r + 1)/2}$. The real part of γ describes the decay of the current along the bow due to radiation.

The above current expression may be used to formulate a simple single-current model for the bow-tie. The antenna pattern and impedance were calculated by considering the integral of the scalar product of the electric field and the current over the bow. The decay constant α may be estimated by comparing theoretical predictions with experiment. Pattern measurements that will be discussed in section 2.5 display features inconsistent with this model. These discrepancies suggest the presence of additional current components. To investigate this in detail, a more complete theory was developed.

2.1.2 Rigorous analysis of the bow-tie.

In the plane z=0 the boundary condition are as follows

$$\begin{aligned} \hat{\mathbf{z}} \times \mathbf{E}^{(a)} &= \hat{\mathbf{z}} \times \mathbf{E}^{(d)} = 0 && \text{(on metal)} \\ \hat{\mathbf{z}} \times \mathbf{E}^{(a)} &= \hat{\mathbf{z}} \times \mathbf{E}^{(d)} && \text{(elsewhere)} \end{aligned} \tag{3}$$

$$\begin{aligned} \hat{\mathbf{z}} \times (\mathbf{H}^{(a)} - \mathbf{H}^{(d)}) &= \mathbf{J} && \text{(on metal)} \\ &= 0 && \text{(elsewhere)} \end{aligned} \tag{4}$$

with the superscripts (a) and (d) denoting air and dielectric fields respectively. To force the normal component of $\mathbf{J}$ to vanish at the end of the bow we modify the current of (2) to obtain the feed (f) current:

$$\mathbf{J}_f = \frac{\pm e^{-\gamma r}/r}{\sqrt{\cos^2\phi - \cos^2\phi_\circ}} \left[1 - \left(\frac{r}{a\cos\phi}\right)^2\right] \hat{\mathbf{r}}. \tag{5}$$

In response to this current a nonzero tangential electric field will form on the metal which will in turn generate additional currents. In general, the total current is

$$\mathbf{J} = \mathbf{J}_f + \sum_{nm} C_{nm} \mathbf{J}_{nm} \tag{6}$$

where the $\mathbf{J}_{nm}$ represent current modes in the bow, and the C_{nm} are unknown coefficients. The unknown coefficients are solved using a moment technique referred to as the Galerkin method. The choice of $\mathbf{J}_{nm}$ will be discussed in detail below. The bow currents are matched to fields in the air region $(z > 0)$ and the dielectric region $(z < 0)$. These fields are expressed as Fourier sums over plane waves $\mathbf{e}^{(i)}_{\mathbf{k}_\parallel p}$, with the TE and TM polarizations indexed as $p = 1$ and 2 respectively. An $e^{j\omega t}$ time dependence is assumed and the direction of propagation of these waves are specified in terms of the transverse component of their k-vector, $\mathbf{k}_\parallel = k_x\hat{\mathbf{x}} + k_y\hat{\mathbf{y}}$. In this notation the electric field in the plane of the bow is given by

$$\mathbf{E} = \sum_p \int d^2\mathbf{k}_\parallel \; A^{(i)}_{\mathbf{k}_\parallel p} \, \mathbf{e}^{(i)}_{\mathbf{k}_\parallel p}. \tag{7}$$

The superscript (i) runs over the air (a) and dielectric (d) regions. Similarly, the magnetic field may be expressed as

$$\hat{\mathbf{z}} \times \mathbf{H}^{(i)} = \mp \sum_p \int d^2\mathbf{k}_\parallel \; Y_{\mathbf{k}_\parallel p} \, A^{(i)}_{\mathbf{k}_\parallel p} \, \mathbf{e}^{(i)}_{\mathbf{k}_\parallel p} \tag{8}$$

where the minus sign holds for the air region ($z > 0$) and the plus sign for the dielectric side ($z < 0$). The $Y_{\mathbf{k}_\parallel p}$ are wave admittances defined in the following way

$$\begin{aligned} Y^{(a)}_{\mathbf{k}_\parallel 1} &= \frac{k_z^{(a)}}{\eta_0 k_0} & Y^{(d)}_{\mathbf{k}_\parallel 1} &= \frac{k_z^{(d)}}{\eta_0 k_0} \\ Y^{(a)}_{\mathbf{k}_\parallel 2} &= \frac{k_0}{\eta_0 k_z^{(a)}} & Y^{(d)}_{\mathbf{k}_\parallel 2} &= \frac{\epsilon k_0}{\eta_0 k_z^{(d)}} \\ k_z^{(a)} &= \sqrt{{k_0}^2 - {k_\parallel}^2} & k_z^{(d)} &= \sqrt{\epsilon {k_0}^2 - {k_\parallel}^2}. \end{aligned} \tag{9}$$

The imaginary part of $k_z^{(i)}$ is chosen to be negative to give decay of the evanescent modes away from the plane of the bow. The electric field must also have the same symmetry as the current in the bow: the $\hat{\mathbf{x}}$ component of $\mathbf{E}$ is even in x and y, and the $\hat{\mathbf{y}}$ component of $\mathbf{E}$ is odd in x and y. These properties may be used to form a simplified set of symmetrized transverse TE and TM fields:

$$\begin{aligned} \mathbf{e}_{\mathbf{k}_\parallel 1} &= \left[k_y \cos k_x x \cos k_y y \;\hat{\mathbf{x}} + k_x \sin k_x x \sin k_y y \;\hat{\mathbf{y}}\right] / k_\parallel \\ \mathbf{e}_{\mathbf{k}_\parallel 2} &= \left[k_x \cos k_x x \cos k_y y \;\hat{\mathbf{x}} - k_y \sin k_x x \sin k_y y \;\hat{\mathbf{y}}\right] / k_\parallel . \end{aligned} \tag{10}$$

The $\hat{\mathbf{z}}$ component (Fig. 1) may be obtained from the requirement that the divergence of the total electric field must be zero. The continuity of the electric field (3), over the entire $z = 0$ plane implies

$$A^{(a)}_{\mathbf{k}_\parallel p} = A^{(d)}_{\mathbf{k}_\parallel p} = A_{\mathbf{k}_\parallel p}. \tag{11}$$

The magnetic field boundary condition (4) requires

$$\begin{aligned} -\sum_p \int d^2\mathbf{k}_\parallel \left(Y^{(a)}_{\mathbf{k}_\parallel p} + Y^{(d)}_{\mathbf{k}_\parallel p}\right) A_{\mathbf{k}_\parallel p}\, \mathbf{e}_{\mathbf{k}_\parallel p} &= \mathbf{J}_f + \sum_{nm} C_{nm} \mathbf{J}_{nm} && \text{(on metal)} \\ &= 0 && \text{(elsewhere).} \end{aligned} \tag{12}$$

Multiplying this equation by $\mathbf{e}_{\mathbf{k}_\parallel p}$ and using the orthogonality relation:

$$\int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} \mathbf{e}_{\mathbf{k}_\parallel p} \cdot \mathbf{e}_{\mathbf{k}_\parallel' q} dx\, dy = \pi^2 \delta_{p,q} \delta(\mathbf{k}_\parallel - \mathbf{k}_\parallel') \tag{13}$$

produces the following reconstruction equation for the Fourier series

$$A_{\mathbf{k}_\parallel p} = -\frac{d^2}{\pi^2} Z_{\mathbf{k}_\parallel p} \left(\sum_{nm} C_{nm} I^{nm}_{\mathbf{k}_\parallel p} + I^f_{\mathbf{k}_\parallel p}\right) \tag{14}$$

in which

$$Z_{\mathbf{k}_\| p} = 1/\left(Y^{(a)}_{\mathbf{k}_\| p} + Y^{(d)}_{\mathbf{k}_\| p}\right)$$
$$I^{nm}_{\mathbf{k}_\| p} = \frac{1}{d^2}\int_{bow} \mathbf{J}_{nm} \cdot \mathbf{e}_{\mathbf{k}_\| p} dx\, dy \tag{15}$$
$$I^{f}_{\mathbf{k}_\| p} = \frac{1}{d^2}\int_{bow} \mathbf{J}_{f} \cdot \mathbf{e}_{\mathbf{k}_\| p} \; dx\, dy.$$

The requirement that the tangential electric field vanish on the metal reduces to

$$\sum_p \int d^2\mathbf{k}_\| \; A_{\mathbf{k}_\| p}\mathbf{e}_{\mathbf{k}_\| p} = 0 \quad \text{(on metal)}. \tag{16}$$

Substituting for $A_{\mathbf{k}_\| p}$ yields the matrix equation

$$\sum_{nm} C_{nm}\left[\sum_p \int d^2\mathbf{k}_\| Z_{\mathbf{k}_\| p}\; I^{nm}_{\mathbf{k}_\| p} I^{NM}_{\mathbf{k}_\| p}\right] = -\sum_p \int d^2\mathbf{k}_\| Z_{\mathbf{k}_\| p}\; I^{f}_{\mathbf{k}_\| p} I^{NM}_{\mathbf{k}_\| p}. \tag{17}$$

$I^{NM}_{\mathbf{k}_\| p}$ is equal to $I^{nm}_{\mathbf{k}_\| p}$ defined in Eq. (15) with the indices NM replacing nm in J_{nm}. This equation can be solved for the unknown current amplitudes C_{nm}, which may be substituted into (14) to reconstruct the fields above and below the bow.

The method outlined above is general. For antennas without the x-y symmetry of the bow-tie, a non-symmetrized version of the plane wave fields should be used.

2.1.3 Current Basis Functions

The basis functions must form a complete set over the metal bow segment. Since they only describe fields in a plane they need not individually satisfy any equation analogous to the Helmholtz wave equation. At the edges of the infinitesimally thick bow they must have zero normal component [14]. The bow symmetry requires that the $\hat{\mathbf{x}}$ component of $\mathbf{J}_{nm}$ must be an even function of y and the $\hat{\mathbf{y}}$ component must be an odd function of y. The modes for the bottom half of the bow follow from the modes for the top half using the x symmetry.

The matrix elements in (17) are two-dimensional integrals of inner products involving the current modes (15). A trigonometric dependence [15] was chosen for the modes to allow analytical evaluation of the inner products and minimize the time spent filling the matrix in (17). The modes are constructed by defining a non-orthogonal coordinate system (s,t) running along the bow edges (see Fig. 1) and defined by

$$\begin{aligned} s &= \frac{d}{2}\left(\frac{x}{a}+\frac{y}{b}\right) & t &= \frac{d}{2}\left(\frac{x}{a}-\frac{y}{b}\right) \\ c &= \sqrt{a^2+b^2} & d &= 2ab/c \\ \hat{\mathbf{n}}_1 &= \left(b\hat{\mathbf{x}}+a\hat{\mathbf{y}}\right)/c & \hat{\mathbf{n}}_2 &= \left(b\hat{\mathbf{x}}-a\hat{\mathbf{y}}\right)/c\ . \end{aligned} \tag{18}$$

A coordinates s value is defined as the perpendicular distance of the point from the t axis. When $x = a \quad y = b$ this perpendicular distance corresponds to the value d. The $\mathbf{J}_{nm}$ current mode is derived by defining intermediate components, $j_{n1} = \mathbf{J}_{nm} \cdot \hat{\mathbf{n}}_1$ and $j_{n2} = \mathbf{J}_{nm} \cdot \hat{\mathbf{n}}_2$ (Fig. 1). The requirements that j_{n1} vanish at s=0 and j_{n2} vanish at t=0 are satisfied by the trigonometric choices

$$\begin{aligned} j_{n1} &= \sin\left(\frac{n\pi s}{d}\right)\cos\left(\frac{m\pi t}{d}\right) \\ j_{n2} &= \cos\left(\frac{m\pi s}{d}\right)\sin\left(\frac{n\pi t}{d}\right). \end{aligned} \qquad (n \neq 0) \tag{19}$$

From the definition of j_{n1} and j_{n2} we can show

$$\mathbf{J}_{nm} = [j_{n1}(\hat{\mathbf{n}}_1 + \cos(2\phi_\circ)\ \hat{\mathbf{n}}_2) + j_{n2}(\hat{\mathbf{n}}_2 + \cos(2\phi_\circ)\ \hat{\mathbf{n}}_1)]/\sin^2(2\phi_\circ). \tag{20}$$

The symmetry properties of $\mathbf{J}$ require that $(n+m)$ be even. Because these modes are finite at the origin, they do not contribute to the current flowing through the bow apex.

In the special case $a = b$, the (s,t) coordinate system becomes orthogonal. In this case the modes are a dual set to the TE and TM waveguide modes that are known to be complete. Mode completeness in the general case $(a \neq b)$ will be discussed in more detail in the following numerical section.

2.2 Numerical Implementation and Verification

The equations were solved using IBM PC and AT computers. The main sections of code were written in *Turbo Pascal*. Frequently called functions involving complex number manipulation were written in assembly language. The total code length was approximately 2000 lines. Typical run times for a full solution with 80 current modes were 12 hours. Most of this time was spent filling the matrix.

All analytical integrals were checked numerically and confirmed using the MU-MATH symbolic manipulator. Another test combines verification of inner products with confirmation of the completeness of a given set of basis functions ($\mathbf{F}_i$ say) [16]. Let $\mathbf{G}$ denote a trial function and express it as a series combination of the $\mathbf{F}_i$. Then the inner product of $\mathbf{G}$ with itself should converge to a sum of products of integrals involving the $\mathbf{F}_i$. The rate of convergence provides information about the errors involved in truncation of the basis sets. In the case where $\mathbf{F}$ are plane wave functions this test is a statement of Parseval's theorem. This completeness test was applied to three separate trial functions $\mathbf{G}$. Firstly, the feed current $\mathbf{J}_f$ was expanded in terms of $\mathbf{e}_{\mathbf{k}_{\parallel}p}$. The area of integration was chosen to be a sub-region excluding the origin and edges where $\mathbf{J}_f$ is not square integrable. The $\mathbf{J}_{nm}$ were also expanded in terms of $\mathbf{e}_{\mathbf{k}_{\parallel}p}$ and checked. In the third test the $\mathbf{e}_{\mathbf{k}_{\parallel}p}$ were expanded in terms of $\mathbf{J}_{nm}$.

Selection of the appropriate $\mathbf{J}_{nm}$ modes to include is important. Too few modes will result in a solution that has not converged, while inclusion of too many unnecessary modes leads to prohibitively-long computation times. To examine convergence in detail, contour plots were constructed in the (n, m) plane of the modal amplitudes $|C_{nm}|$ for a range of bow parameters and total number of current modes. The dominant modes were found to be clustered around the line $m = n$. These modes correspond to standing wave modes that propagate along the bow with wave number m/a. Propagation constants corresponding to k_0, $k_d = \sqrt{\epsilon_r}\, k_0$, and k_{qs} are plotted in Fig. 2 for a sixty-degree bow (full angle $2\phi_0 = 60°$) of length $a = 2\lambda_0$. This figure suggests that for this bow the dominant modes are clustered around k_d. These contour plots allow an efficient selection of modes, thereby improving convergence of the solution. Large mode amplitudes close to the boundary of allowed modes would suggest than an insufficient set of modes has been selected.

A powerful test of the formulation can be made by varying the propagation constants α and β of the current $\mathbf{J}_f$. This does not change the currents near the apex of the bow but does change the imposed current everywhere else. The

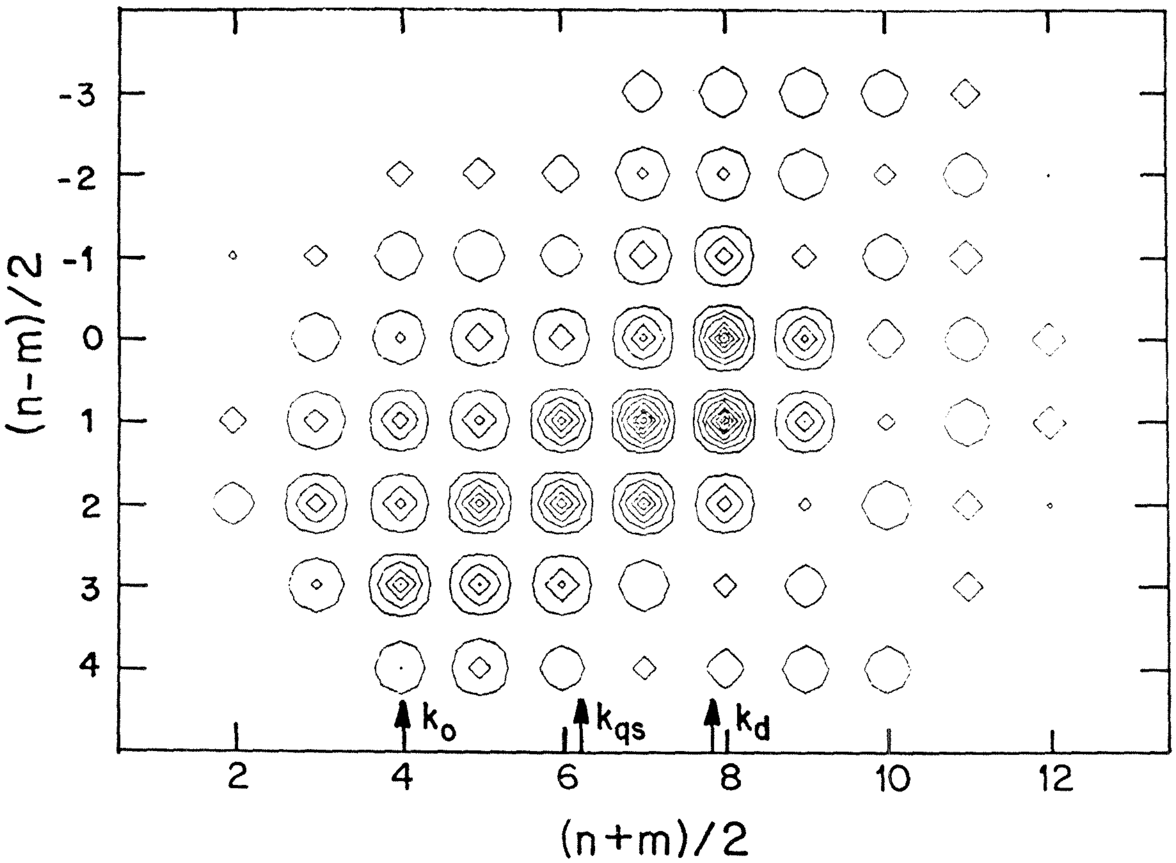


Figure 2. A contour plot of the amplitudes $|C_{nm}|$ in the $n-m$ plane for a bow-tie antenna with $a = 2\lambda_0$, $2\phi_0 = 60°$ and $\epsilon_r = 3.83$.

resulting antenna patterns and impedances were found to be independent of α and β. However, the convergence is not as rapid for non-optimal choices of these constants—the contour plot becomes less localized.

The double integrals over the positive quadrant of the $\mathbf{k}_{\parallel}$ plane in (17) were evaluated numerically using a polar integration in $k_{\parallel}$ and ϕ. Suitable step sizes for each variable were chosen by examining the functional variation of the integrand. The upper limit on $k_{\parallel}$ was obtained by looking at the convergence of the integrand for large $k_{\parallel}$. The integrals were found to converge as a power of $1/k_{\parallel}$ and an error due to truncating the integral was calculated. The integration was terminated when the estimated truncation error dropped below one percent.

The antenna impedances may be calculated by integrating the complex Poynting vector over the $\mathbf{k}_{\parallel}$ plane to obtain

$$
\begin{aligned}
Z &= \eta_0 \frac{\pi^2}{I^2} \int_0^\infty \int_0^\infty \left(P^{(d)}_{k_x k_y} + P^{(a)}_{k_x k_y} \right) \, dk_x \, dk_y \,, \\
P^{(i)}_{k_x k_y} &= Y^{(i)}_{\mathbf{k}_{\parallel} 1} |A_{\mathbf{k}_{\parallel} 1}|^2 + Y^{(i)}_{\mathbf{k}_{\parallel} 2} |A_{\mathbf{k}_{\parallel} 2}|^2 .
\end{aligned}
\tag{21}
$$

The real part of the integrand can be identified with the radiated power, so the normalized antenna gain may be written as

$$
\begin{aligned}
G^{(i)}(\theta, \phi) &= \kappa Re(P^{(i)}_{k_x k_y}) \cos\theta \\
k_x &= k^{(i)} \sin\theta \cos\phi \\
k_y &= k^{(i)} \sin\theta \sin\phi
\end{aligned}
\tag{22}
$$

where the constant κ is chosen to normalize the peak gain to unity, and the superscript (i) refers to the air (a) and dielectric (d) regions. The $\cos\theta$ factor arises from converting the $dk_x \, dk_y$ integral into an integral over the solid angle element $d\Omega$.

A final theoretical check is provided by the reciprocity theorem, in which the bow-tie is treated as a receiving rather than transmitting system. For a particular wave $\mathbf{k}^{\circ}_{\parallel}$ with a particular amplitude $A^{(i)}_{\mathbf{k}^{\circ}_{\parallel} p}$ we calculate the power absorbed by the antenna. For the reciprocal problem

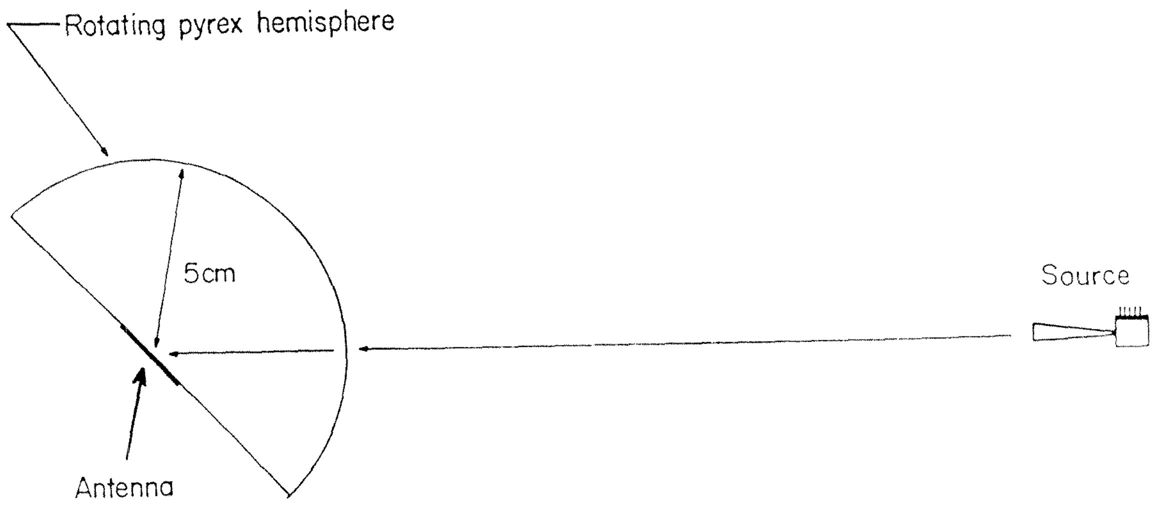
$$
\sum_{nm}{}' C_{nm} \left[\sum_p \int d^2\mathbf{k}_{\parallel} Z_{\mathbf{k}_{\parallel} p} \, I^{nm}_{\mathbf{k}_{\parallel} p} I^{NM}_{\mathbf{k}_{\parallel} p} \right] = -2 \frac{\pi^2}{d^2} \sum_p Z_{\mathbf{k}^{\circ}_{\parallel} p} I^{NM}_{\mathbf{k}^{\circ}_{\parallel} p} (A^{(a)}_{\mathbf{k}^{\circ}_{\parallel} p} Y^{(a)}_{\mathbf{k}^{\circ}_{\parallel} p} + A^{(d)}_{\mathbf{k}^{\circ}_{\parallel} p} Y^{(d)}_{\mathbf{k}^{\circ}_{\parallel} p}).
\tag{23}
$$

The $\sum'_{nm}$ denotes that the sum also includes the $\mathbf{J}_f$ term, whose amplitude is now an unknown. The numerical results for the reciprocal treatments were found to agree to better than one percent.

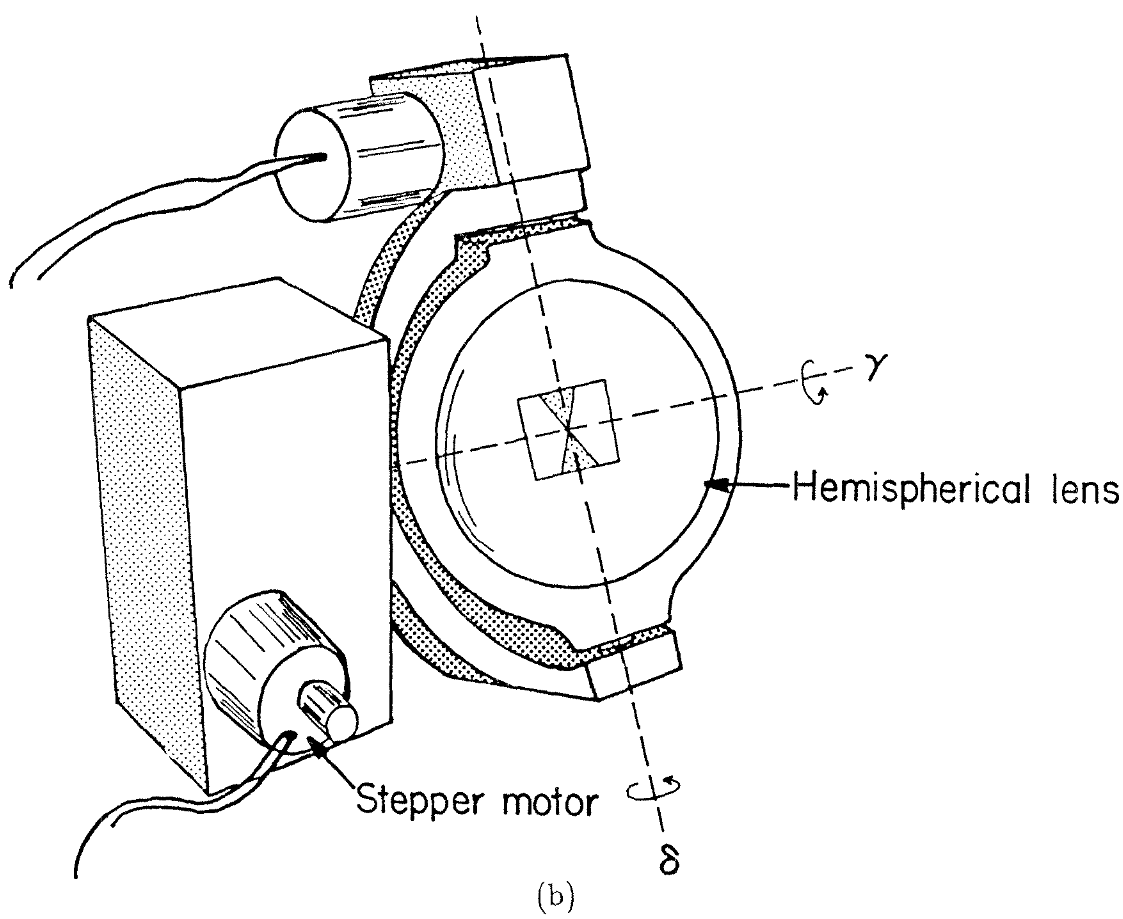
2.3 Antenna measurements at 94 GHz

At the measurement frequency of 94 GHz, it is difficult to feed the antenna and measure its pattern in transmission, so the receiving pattern was measured. The source was a Gunn diode modulated at 1 kHz and operating at a power level of 10mW. The antennas were fabricated on 1″ square, 1/16″ thick Optosil fused silica substrates with integrated bismuth bolometer detectors [17]. The dielectric half space is simulated by placing the antenna at the center of a hemispherical lens. The lens is made of Corning 7940 optical grade fused silica with dielectric constant $\epsilon_r = 3.83$ [18] and diameter 11 cm . The lens diameter is sufficiently large so that the curved surface is in the far field of the antenna. The lens is mounted in a two-axis rotation gimbal system (Fig. 3). The system is rotated by two stepper-motors in increments of $\frac{1}{4}^\circ$. The measurements were controlled with an IBM personal computer. Using an analog/digital interface, the IBM sent TTL pulses to the stepper-motor drivers and then recorded the detected signal from the output of a lock-in amplifier. Variations in the transmitted power were compensated by monitoring the diode power, using a 10-dB coupler and thermistor placed before the horn. Measurements were made in the E and H planes of both the polarized and cross-polarized components, and full two-dimensional scans were also made. The patterns were only measured on the dielectric side since most of the power is radiated into this side.

The reproducibility of patterns depends critically on alignment. A small misalignment in the system produces asymmetries in the patterns. The axis of the system was accurately aligned using a He-Ne laser. The degree of symmetry of the E and H planes was quite sensitive to the position of the antenna and this allowed further improvements in the alignment. Care was also taken to minimize the effect of scattering from the metal frame supporting the lens. Placing absorber around



(a)



(b)

Figure 3. (a) Schematic diagram of measurement antenna measurement configuration. (b) Detailed view of the lens mounting showing the two stepper-motors. The antenna rests behind the lens at the intersection of the two axes of rotation (γ, δ).

the edge of the lens reduces this scattering, at the expense of shadowing the source at large angles. An etalon effect due to multiple reflections within the hemisphere produces many small bumps in the pattern at normal incidence. A quarter-wave anti-reflection coating was formed by drape-molding a 20-mil high-impact styrene sheet onto the hemispherical lens. This reduced the small bumps without changing the overall pattern shape. For large angles of incidence the pattern can be distorted by edge effects if the area of the substrate on which the antenna is fabricated is too small.

2.4 Microwave scale modelling

Bow-tie impedances were measured by building a microwave scale model designed to operate at 2 GHz. The bow-tie was constructed by cutting a copper triangle out of 3M adhesive backed copper foil. The foil was placed on a Stycast block ($\epsilon_r = 4$). The other bow half is simulated by a ground plane through which a coaxial SMA cable connected to the copper triangle is fed. The SMA center pin is then soldered to the apex of the bow. Impedances were obtained using an IBM PC controlled HP 8410 network analyzer.

2.5 Comparison of measurements with theory

Measurements and calculations, in the E-plane, for a sixty-degree bow on fused quartz are compared in Fig. 4. The two are in good agreement, particularly at large angles, where the antenna pattern is strong. The difference in the patterns for the small bumps may be attributed to scattering in the measurement setup. Two-dimensional scans of the measured and calculated response are given in Fig. 5. For this bow-tie the radiated power in the dielectric was calculated to be twenty times larger than the power radiated into free-space. The cross-polarized component in the E and H planes was measured to be more than 20 dB below the main peak. Calculations show that as the bow-length is increased beyond two wavelengths, the number of peaks in the E-plane increases and the major peak moves to larger angles. The positions of the peaks can be accurately predicted by a current which propagates along the axis of the bow with the dielectric wave number $k_d = \sqrt{\epsilon_r}\, k_0$. For bow-lengths less than a wavelength the E and H planes tend towards the pattern of a dipole on an infinite dielectric substrate [1].

The sixty-degree bow-tie impedance (21) was calculated for a range of lengths and compared with experimental values (Fig. 6). The impedances loop at approximately $\lambda_d/2$ and spiral rapidly towards the quasi-static Z_{qs} given in (1).

As the bow angle is decreased with the bow length held fixed, the subsidiary maxima in the antenna pattern weaken. The principal maximum moves towards smaller angles. For $\phi_0 = 1°$, the position of the maximum corresponds to the quasi-static wave number k_{qs}. Also, the amplitudes C_{nm} are strongest for current modes clustered around k_{qs}, rather than the dielectric wave number k_d. Therefore the dominant currents for a narrow bow propagate at k_{qs}. The theoretical E-plane pattern is plotted in Fig. 7.

The distribution of radiation from a narrow bow resembles the experimental pattern for the long strip-dipole antenna of Fig. 7. The fitted model in Fig. 7 was obtained by assuming a damped traveling wave current:

$$\mathbf{J} = J_0 e^{-(\alpha + j\beta)x}\, \hat{\mathbf{x}} \qquad |y| < w/2. \tag{24}$$

The current is assumed to have no variation across the width w of the wire, which is much less than a wavelength. The long-wire model assumes the wire length is infinite. This assumption is justified provided the attenuation is sufficient to make the currents at the end of the wire small. This current can be Fourier transformed and multiplied by the corresponding wave impedances to obtain the antenna pattern. The values for α, β given in the figure caption were adjusted to optimize agreement with measurement. This simple model gives an E-plane pattern in agreement with the measured values. The value of β in Fig. 7 lies close to k_{qs} [19], and the value of α is sufficient to ensure that the currents at the end of the bow are small. The measured and calculated quadrant scans of Fig. 8 confirm the adequacy of the traveling-wave model (24).

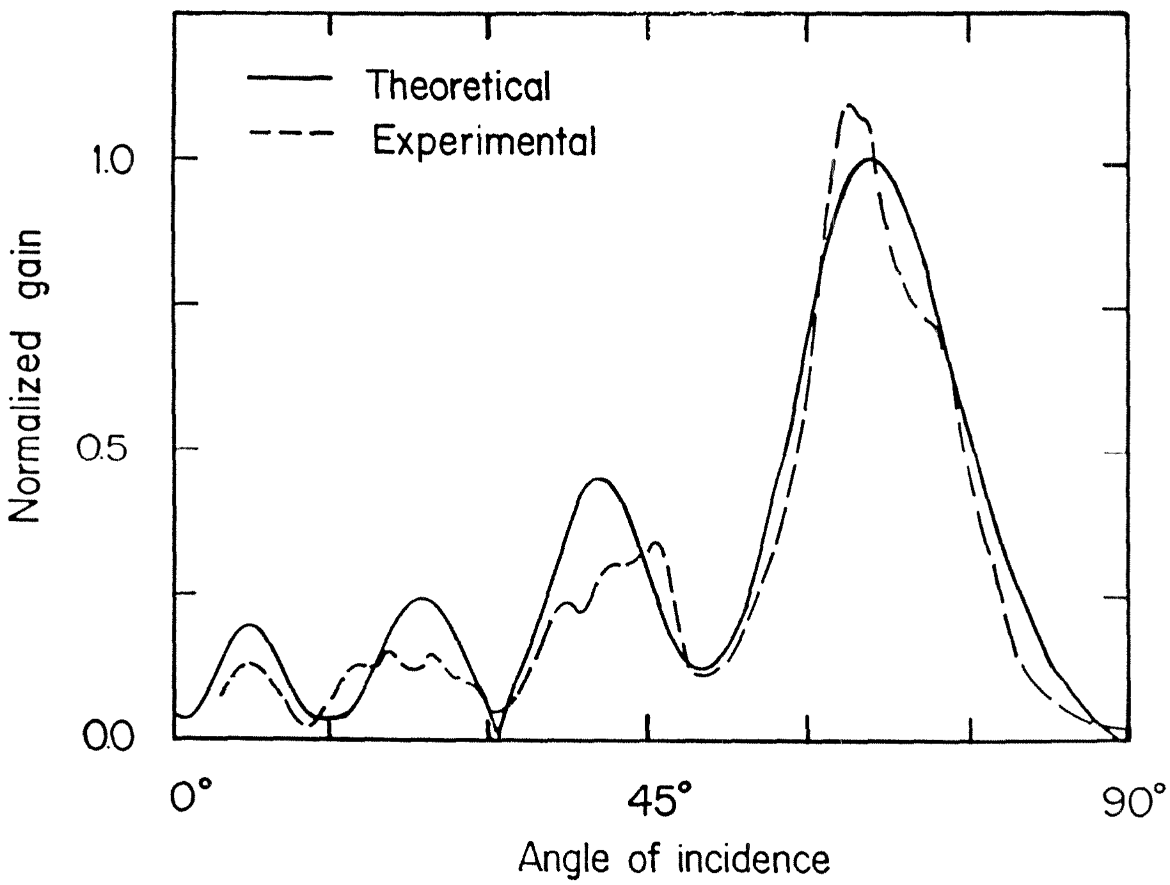


Figure 4. Theoretical (solid line) and experimental (dashed) E-plane patterns, on the dielectric side, for a bow-tie antenna with $a = 2\lambda_0$, $2\phi_0 = 60^\circ$ and $\epsilon_r = 3.83$. The measurements were made every half-degree.

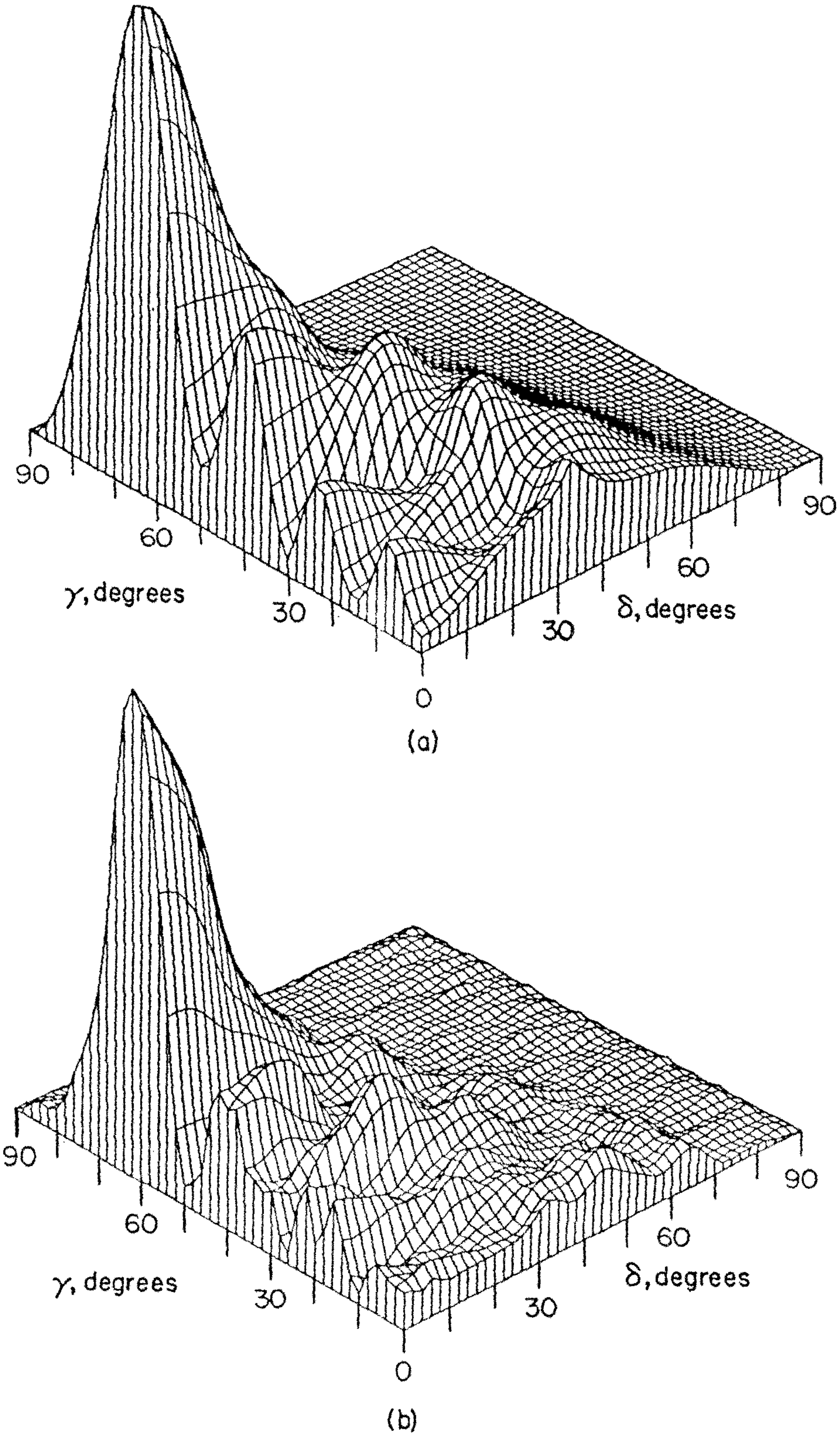


Figure 5. Two-dimensional (a) theoretical and (b) experimental scans of the antenna patten for the bow-tie antenna of Fig. 4.

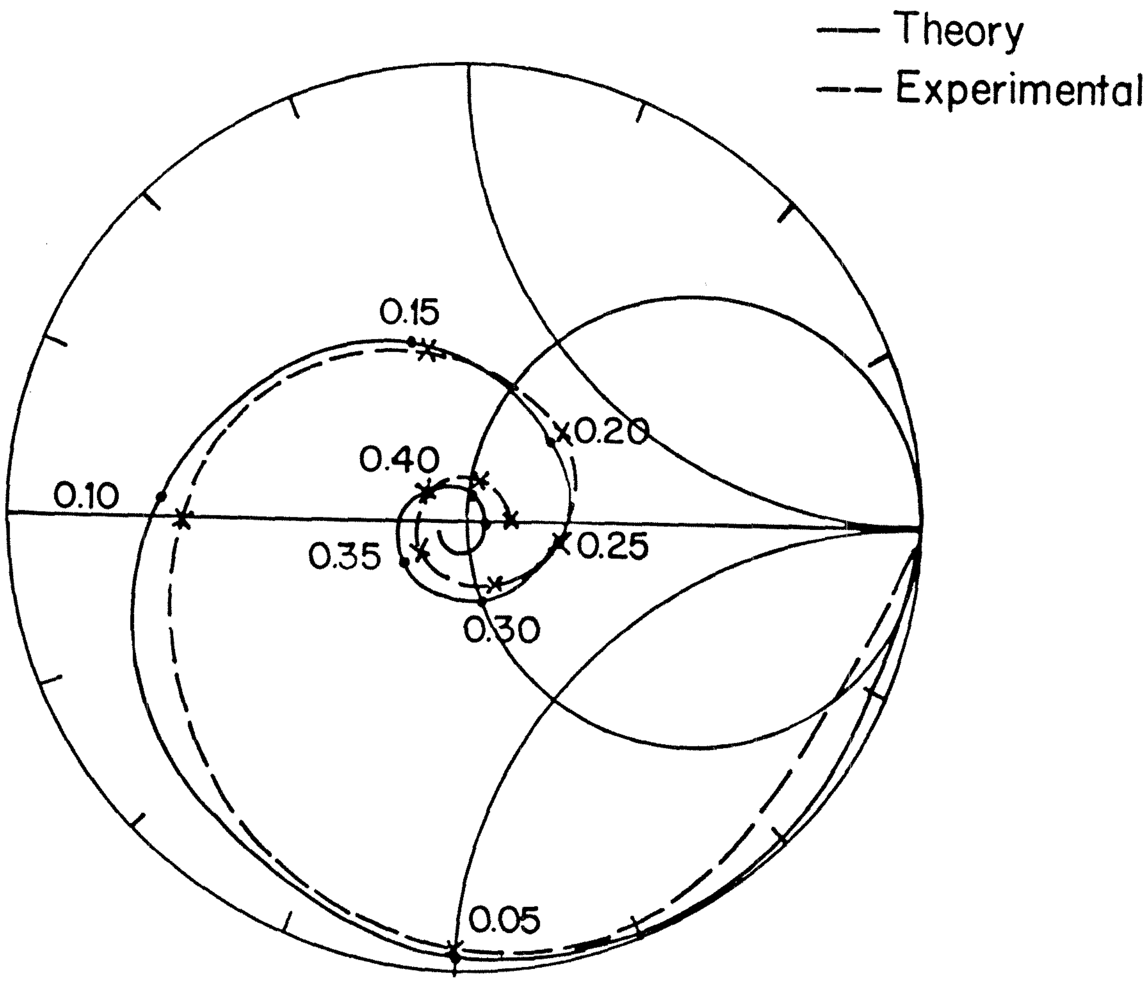


Figure 6. Smith chart plot of bow-tie impedance, normalized to $Z_{qs} = 152\,\Omega$ for $\epsilon_r = 4$ and $2\phi_o = 60°$. The values given correspond to lengths a (Fig. 1) in free space wavelengths.

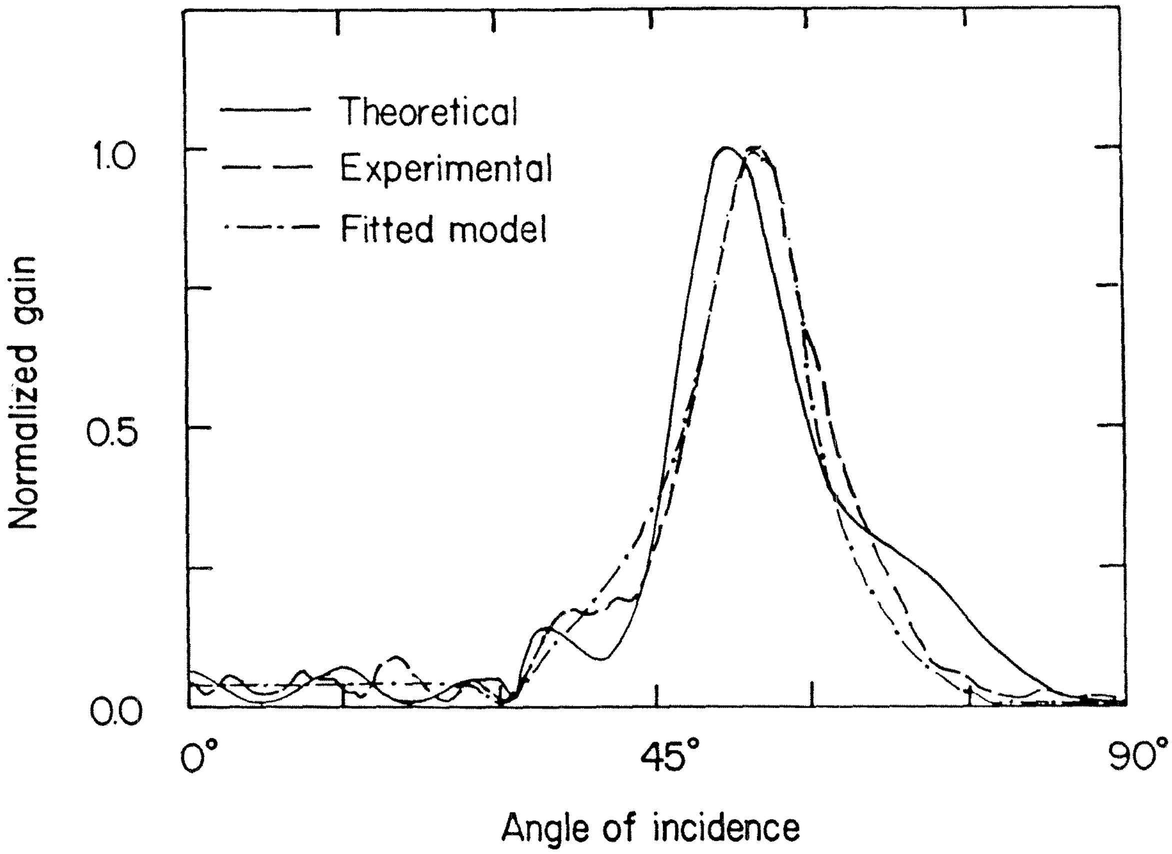


Figure 7. Experimental (dashed line) E-plane patterns on the dielectric side, for a long strip-dipole antenna, $4\lambda_0$ long and $w = 20\mu m$ wide ($\epsilon_r = 3.83$), compared with the bow-tie theory for $\phi_o = 1°$, $a = 2\lambda_0$ (solid) and a fitted long wire model (dot-dash) with $\beta = 1.04k_{q}$, and $\alpha = 7dB/\lambda_0$.

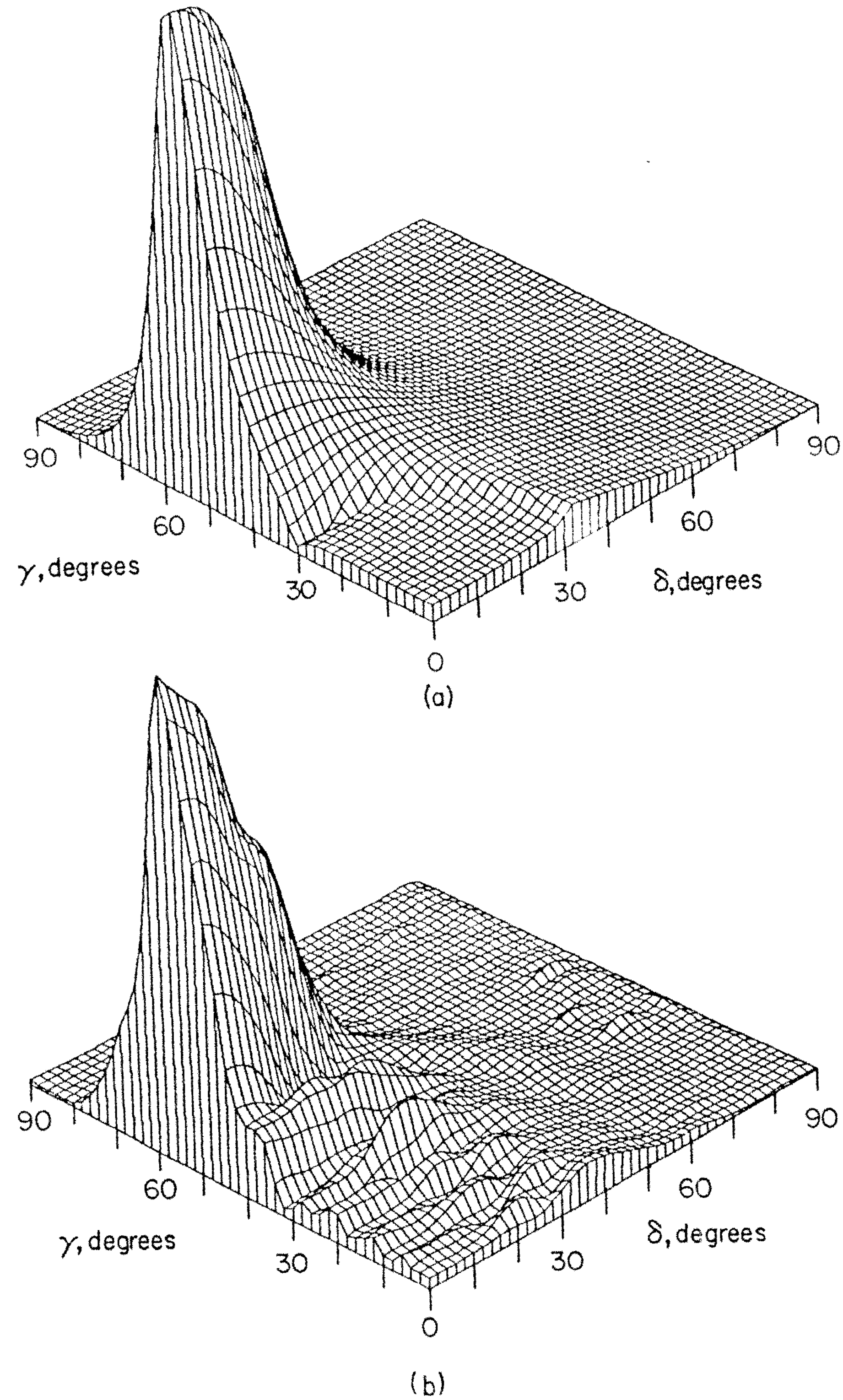


Figure 8. Two-dimensional (a) fitted model and (b) experimental scans of the antenna pattern for the long wire antenna of Fig. 7.

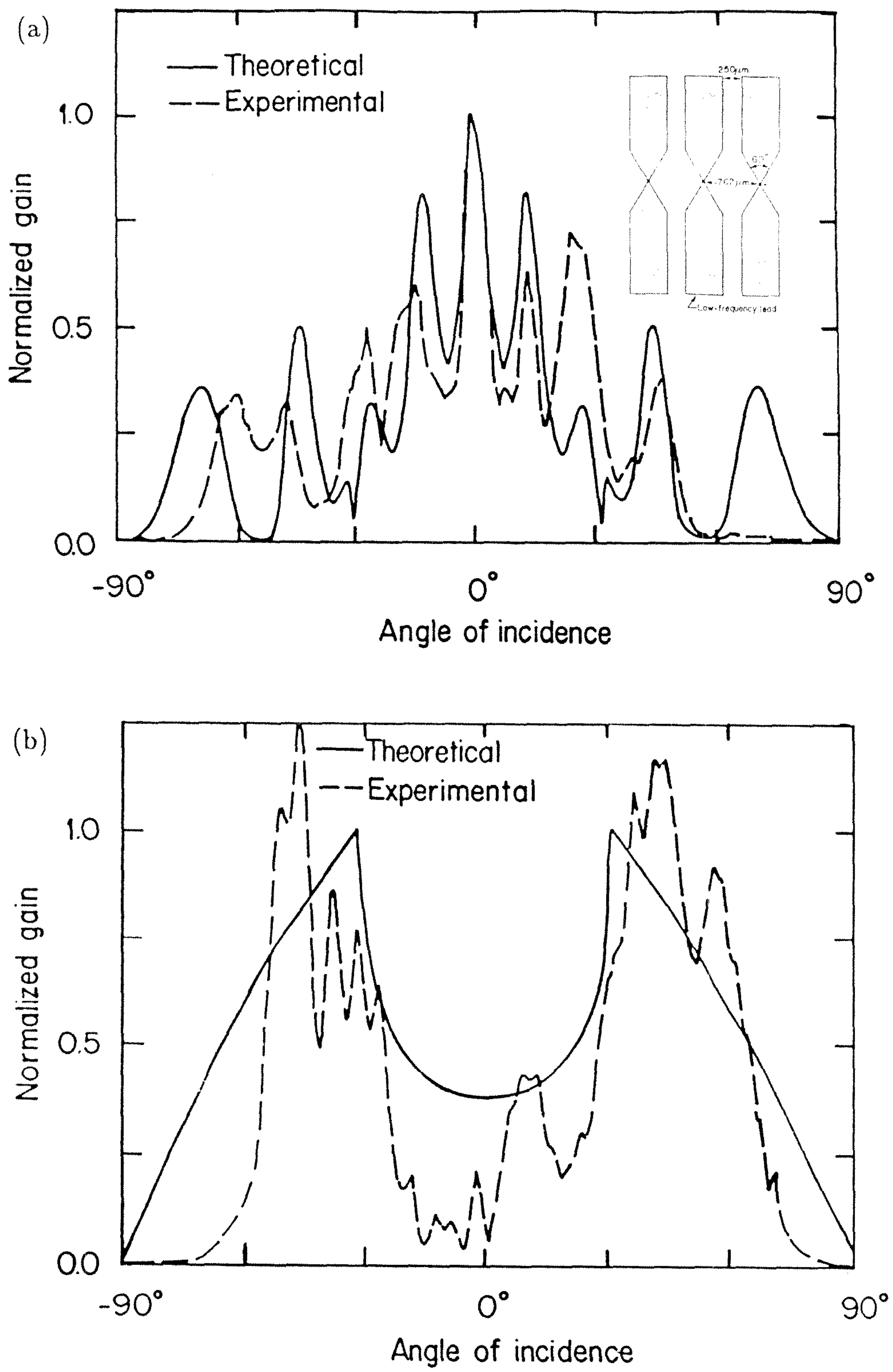
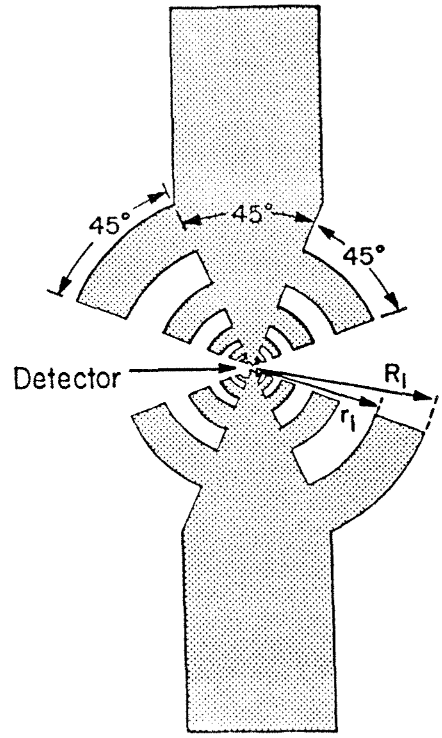


Figure 9. (a) E plane and (b) H plane measurements (dashed line) on the dielectric side for a linear array, compared with a modified dipole pattern (solid), ($\epsilon_r = 3.83$). The antenna spacing is 0.762 mm.

Bow-tie antennas have also been used in linear imaging arrays [1]. Figure 9 shows results for a linear array of 15 elements designed to operate at 94 GHz. The array was fabricated on a 13-mm quartz substrate and mounted on a 64-lead 3M chip carrier. Shown in Fig. 9 are experimental E and H plane measurements for a single array element in the presence of the other elements. At large angles the array measurements are affected by shadowing due to absorber placed around the edges of the lens. The dip in the H-plane at normal incidence is a feature characteristic of the dipole antenna [1]. The fact that it dips to a very low value is believed to result from a small alignment uncertainty for which the H-plane scan corresponds to a scan through one of the sharp troughs adjacent to the center peak in Fig. 9a. The measurements are compared with patterns for a narrow dipole antenna of approximate equivalent length equal to the triangular bow segment. The dipole effect can be visualized as occurring when the expanding wave associated with the current strikes a discontinuity near the end of bow segment. The discontinuity that determines this length may occur when the expanding wave strikes the nearest neighbors. A discontinuity also occurs in the transition from the bow to the low-frequency leads. The E-plane dipole pattern is modulated by the presence of a small current on the low-frequency leads. To construct the theoretical curves of Fig. 9, the lead current was of the form $\sin k_{qs}(x - l)$ where l is the length of the low frequency lead. The relative amplitude of the lead current may not be accurately determined from this experiment. For the theoretical plot in Fig. 9a an amplitude corresponding to five percent of the current flowing in the bow segment gives a modulation comparable with the measurements. In contrast to the two-wavelength bow results, much of the power radiates toward the normal. This allows reasonably efficient coupling to an antenna with a lens. Neikirk [1] measured coupling efficiencies for a single array element of 25% on quartz and Zah [2] measured 34% on silicon.

In the antennas studied so far the current has been allowed to travel freely up and down the antenna. By placing notches and teeth we can force the current to bend and change direction rapidly, as in the log-periodic antenna (Fig. 10) [20,21]. The n^{th} tooth is characterized by an inner radius r_n and an outer radius R_n where

(a)



(b)

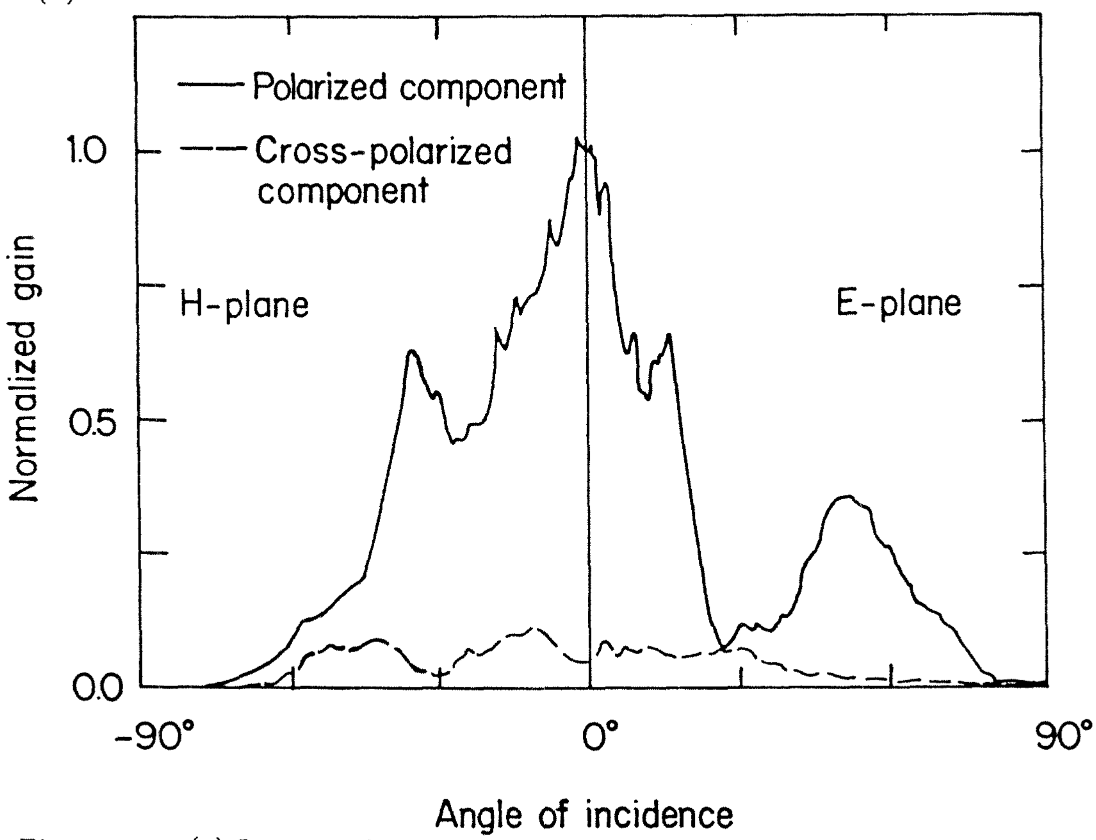


Figure 10. (a) Log-periodic antenna viewed from the free space side. (b) H (left) and E (right) plane measurements on the dielectric side for polarized (solid) and cross-polarized (dashed) components ($\epsilon_r = 3.83$).

$R_n/r_n = \sqrt{2}$ and $R_{n+1}/R_n = \sqrt{2}$. Measurements on the dielectric side [22], of the E and H plane polarized and cross-polarized components are shown in Fig. 10. The antenna orientation was chosen to give a minimum cross-polarization ratio at normal incidence. This orientation corresponds to a polarization along the upper right and lower left teeth of Fig. 10a. The pattern is strongest at broadside and has comparable E and H plane widths. The cross-polarized components are 10 dB below the main peak. Patterns with a centralized peak were also observed for spiral antennas on a dielectric [23]. The polarization properties of the spiral makes it well suited to applications involving circularly-polarized radiation.

References

[1] D. B. Rutledge, D. P. Neikirk, and D. P. Kasilingam, "Integrated-circuit antennas," *Infrared and Millimeter Waves*, **vol. 10**, K. J. Button ed., Academic Press, New York, 1983, pp. 1–90. (Note: The argument k of the elliptic functions of equation (23) should be $\tan^2(45° - \theta/4)$, rather than $\tan(45° - \theta/4)$, where θ refers to the full bow angle.)

[2] C. Zah, D. P. Kasilingam, J. S. Smith, D. B. Rutledge, T. Wang, and S. E. Schwarz, "Millimeter-wave monolithic Schottky diode imaging arrays," *Int. J. Infrared and Millimeter Waves*, **6**, pp. 981–997, 1985.

[3] M. J. Wengler, D. P. Woody, R. E. Miller, and T. G. Phillips, " A low-noise receiver for millimeter and submillimeter waves," *Int. J. Infrared and Millimeter Waves*, **6**, pp. 697–706, 1985.

[4] D. B. Rutledge and M. S. Muha, "Imaging antenna arrays," *IEEE Trans. Antennas Propagat.*, **AP-30**, pp. 535–540, 1982.

[5] P. E. Young, D. P. Neikirk, P. P. Tong, D. B. Rutledge, and N. C. Luhmann Jr., "Multi-channel far-infrared phase imaging for fusion plasmas," *Rev. Sci. Instruments*, **56**, pp. 81–89, 1985.

[6] M. Kominami, D. M. Pozar, and D. H. Schaubert, "Dipole and slot elements and arrays on semi-infinite substrates," *IEEE Trans. Antennas Propagat.*, **AP-33**, pp. 600–607, 1985.

[7] C. R. Brewitt-Taylor, D. J. Gunton, and H. D. Rees, "Planar antennas on a dielectric surface," *Electron. Lett.*, **12**, pp. 729–731, 1981.

[8] T. G. Phillips and D. B. Rutledge, "Superconducting tunnel detectors in radio astronomy," *Scientific American*, **254**, pp. 96–102, May 1986.

[9] C. Zah, R. C. Compton, and D. B. Rutledge, "Efficiencies of elementary integrated-circuit feed antennas," *Electromagnetics*, **3**, pp. 239–254, 1983.

[10] D. R. Jackson and N. G. Alexopoulos, "Microstrip dipoles on electrically thick substrates," *Int. J. Infrared and Millimeter Waves*, **7**, pp. 1–26, 1986.

[11] R. S. Elliott, *Antenna Theory and Design*, Prentice-Hall, Englewood Cliffs, 1981, pp. 41-46.

[12] M. Abramowitz and I. A. Stegun, *Handbook of Mathematical Functions*, National Bureau of Standards, Washington, 1970, p 608.

[13] W. R. Smythe, *Static and Dynamic Electricity*, McGraw Hill, New York, 1968, pp. 101–105.

[14] D. S. Jones, *The Theory of Electromagnetism*, Macmillan, New York, 1964, pp. 566–569.

[15] R. C. Compton and D. B. Rutledge, "Approximation techniques for planar periodic structures," *IEEE Trans. Microwave Theory Tech.*, **MTT-33**, pp. 1083–1088, 1985.

[16] L. C. Botten and R. C. McPhedran, "Completeness and modal expansion methods in diffraction theory," *Optica Acta*, **32**, pp. 1479–1488, 1985.

[17] D. P. Neikirk, W. W. Lam, and D. B. Rutledge, "Far-infrared microbolometer detectors," *Int. J. Infrared and Millimeter Waves*, **5**, pp. 245–278, 1984.

[18] M. N. Afsar, "Dielectric Measurements of Millimeter-Wave Materials," *IEEE Trans. Microwave Theory Tech.*, **MTT-32**, pp. 1598–1609, 1984.

[19] J. R. Wait, "Theory of wave propagation along a thin wire parallel to an interface," *Radio Sci.*, **7**, pp. 675–679, 1972.

[20] R. H. DuHamel and D. E. Isbell, "Broadband logarithmically periodic antenna structures," *IRE National Convention Record*, **part 1**, pp. 119–128, 1957.

[21] P. H. Siegel, "A planar log-periodic mixtenna for millimeter and submillimeter wavelengths," *IEEE MTT-S Digest*, pp. 649–652, 1986.

[22] R. C. Compton, R. C. McPhedran, Z. Popović, G. M. Rebeiz, P. P. Tong, and D. B. Rutledge, "Bow-tie antennas on a dielectric half space: theory and experiment," *IEEE Ant. and Propagation*, **vol. 35**, June 1987.

[23] P. P. Tong, *Millimeter-wave integrated-circuit antenna arrays*, PhD Thesis, California Inst. of Tech., Chpt. 3, 1984.

Chapter 3

Techniques for analyzing two dimensional arrays

A conducting screen periodically perforated with holes acts as a frequency selective reflector and transmitter for incident plane waves (Fig. 1). Meshes or grids as they called, are used in quasi-optical systems as millimeter/microwave filters, polarizers, laser output couplers and beamsplitters. Meshes can also be constructed from planar arrays of metal disks supported by a dielectric substrate. The rigorous calculation of electromagnetic properties of periodic meshes may be performed using the method of moments and requires considerable algebraic work and computer resources. In this chapter a number of approximation techniques for analyzing thin structures with square, rectangular and circular holes are presented. Formulas for the effective impedance of these meshes are described which can take into account non-normal incidence and the presence of a dielectric substrate. In addition, techniques for analyzing more complex shaped apertures such as a cross are discussed. These methods are more accurate than existing approximation techniques and can be applied to a wide range of situations they could not be handled before.

Periodic meshes are becoming increasingly important in the construction of microwave systems [1–5]. To design these systems efficiently it is essential to be able to reliably predict mesh properties. The diffraction properties of meshes may be calculated very accurately using the method of moments [6–8] in which the electromagnetic fields are expanded in terms of Floquet and waveguide modes. Unfortunately this method produces a relatively complicated set of equations which must be solved using a large computer. The difficulties involved with the rigorous moment method has led to development of approximate methods for studying these meshes [9–10].

The equivalent circuit model was one of the first methods introduced to calculate transmission properties of singly and doubly periodic structures [11–12]. This method is widely used because of its simplicity but has many limitations. In this model the mesh is represented by a LC circuit shunted across a transmission line. For a mesh with square holes the inductance is estimated by assuming its long wavelength behavior is like a strip grating whose inductance can be calculated using a conformal mapping [13]. The capacitance is chosen so that the free-space resonant wavelength is equal to the grid period. This approach has several limitations. The inductance and resonance estimate are both very inaccurate—particularly when the squares are small compared to the grid period and a dielectric is present. In addition the circuit model can only be applied to a small number of geometries—strips or square holes with 90° periodicity axes at normal incidence.

For thick meshes very good results have been obtained using methods based on the assumption that most of the energy is carried by one waveguide mode—all other modes being cut off [3]. For thin meshes the monomodal approximation starts to break down (see Fig. 1) because evanescent modes can carry energy through the mesh. Finally if the hole shape is anything other than a rectangle, circle or strip, conventional methods become complicated because the field can no longer be described in terms of simple waveguide modes [14].

The following sections present a number of approximation techniques for thin meshes which overcome the above limitations. The accuracy of the techniques will be displayed by comparing the results with rigorous solutions that are known to give excellent agreement with experiment [3].

3.1 Background theory for planar meshes

For the inductive and capacitive meshes of Fig. 2 at an air-dielectric interface (refractive index n) the equivalent circuit model predicts the following formula for energy transmittance [13]

$$T = \frac{4n}{(1+n)^2 + \left(\frac{Z}{X}\right)^2} \tag{1}$$

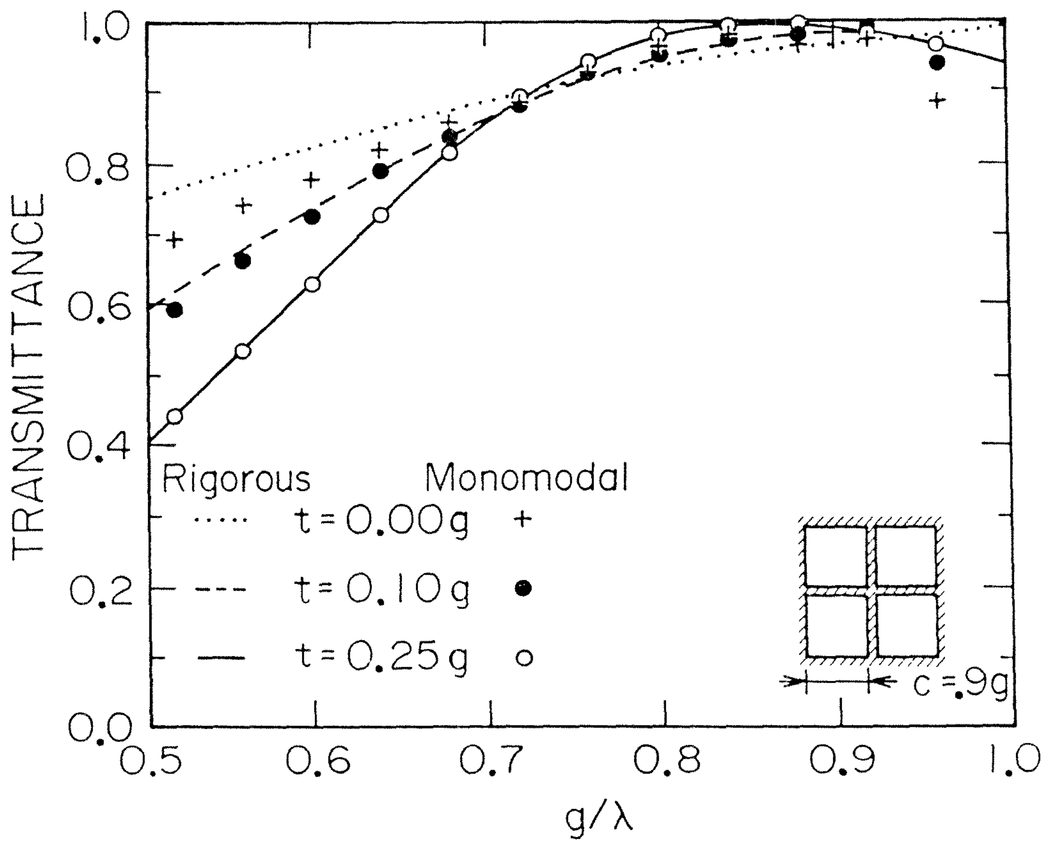


Figure 1. Transmittance curves for normally incident radiation on a mesh with varying thicknesses. The mesh is characterized by a period g, thickness t and a square hole size c. The monomodal calculations [3] make a good approximation to the rigorous calculations [8] for thicknesses ≥ 0.10 g.

$$\frac{X_I}{Z} = -(\omega_o W)\left(\frac{\omega}{\omega_o} - \frac{\omega_o}{\omega}\right)^{-1}$$
$$\frac{X_C}{Z} = \frac{2}{1+n^2}(4\omega_o W)^{-1}\left(\frac{\omega}{\omega_o} - \frac{\omega_o}{\omega}\right) \qquad (2)$$
$$W = \ln(\mathrm{cosec}\frac{\pi a}{g})$$

where $\omega = g/\lambda$ is the normalized frequency and ω_o is the resonant frequency. At long wavelengths ($\omega \rightarrow 0$)these impedances reduce to the strip grating impedances [13]

$$\frac{X_I}{Z} = \omega W$$
$$\frac{X_C}{Z} = \frac{-2}{1+n^2}(4\omega W)^{-1} \quad . \qquad (3)$$

Note that the inductance is unaffected by the presence of a dielectric and the capacitance transforms like two capacitors in parallel.

For perfectly conducting meshes of infinitesimal thickness the method of moments reduces to the following equations [6-7]

$$Y^{Mm} F_m = I^M \quad . \qquad (5)$$

For the inductive mesh the F_m represent expansion coefficients for the electric field where

$$Y^{Mm} = \sum_{pq} \sum_{r=1}^{2} (\xi_{pqr} + y_{pqr}) C^{*}_{pqr}{}^{M} C_{pqr}{}^{m}$$
$$I^M = \sum_{r=1}^{2} A_{00r} \xi_{00r} C_{00r}{}^{M} \qquad (6)$$
$$C_{pqr}{}^{m} = \int \int_{aperture} \Phi^{*}_{pqr} \cdot \Psi^{m}$$

and for the capacitive mesh the F_m are coefficients in the current expansion with

$$Y^{Mm} = \sum_{pq} \sum_{r=1}^{2} \frac{1}{(\xi_{pqr} + y_{pqr})} D^{*}_{pqr}{}^{M} D_{pqr}{}^{m}$$
$$I^m = \sum_{r=1}^{2} A_{00r} \frac{\xi_{00r}}{(\xi_{00r} + y_{00r})} D^{m}_{00r} \qquad (7)$$
$$D_{pqr}{}^{m} = \int \int_{aperture} \Phi^{*}_{pqr} \cdot \Xi^{m} \quad .$$

The Φ_{pqr} are the Floquet modes, the Ψ^m waveguide aperture modes, Ξ^m are current modes and the ξ_{pqr} and y_{pqr} are the impedance on the two sides of the mesh [6–7].

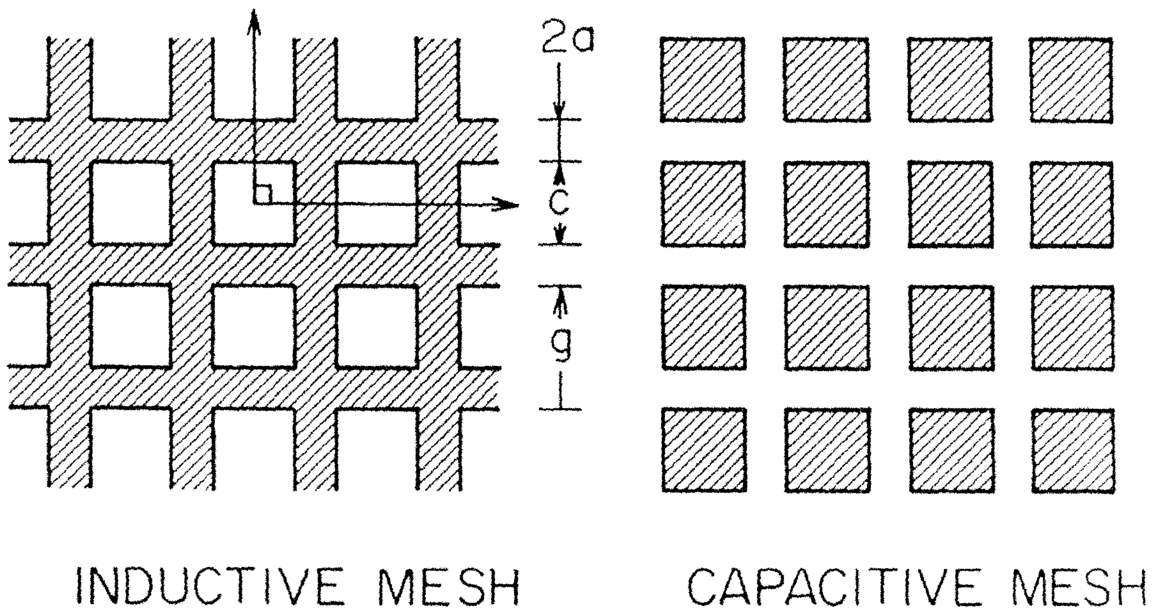


Figure 2. Geometry of inductive and capacitive meshes with square holes and periodicity axes inclined at 90°.

The A_{00r} are the coefficients of the incident field ($r = 1$ is TE and $r = 2$ is TM). The solution of these equations involves the calculation of the integrals C or D and the inversion of matrix Y whose elements are complex numbers. This requires considerable time and computer resources but can be made much easier by making a few simple approximations.

At long wavelengths the matrix Y becomes dominated by the diagonal elements and the elements related to the primary mode (m=0 say). Discarding all other terms results in a set of refined monomodal equations that may be solved analytically to yield the following expression for F_0 (5)

$$F_0 = \frac{I^0 - \sum\limits_{\substack{m \\ m \neq 0}} \frac{Y^{0m} I^m}{Y^{mm}}}{Y^{00} - \sum\limits_{\substack{m \\ m \neq 0}} \frac{Y^{0m} Y^{m0}}{Y^{mm}}} \quad . \tag{8}$$

This may be used to calculate transmittance in a form which reduces to (1) and provides a considerably more accurate estimation of the long wavelength impedance than (3). For a TE incident wave, in the limit when the primary mode dominates, the terms in the sum of (8) are small and the impedance reduces to (2) with

$$W = \frac{\pi |C_{001}{}^0|^2}{\sum\limits_{pq}{}' \; |C_{pq1}{}^0|^2 \sqrt{\alpha_p^2 + \gamma_{pq}^2}} \quad . \tag{9}$$

The α_p and γ_{pq} describe the spatial dependence of the Floquet modes [8] and the $\sum'$ denotes the $p = 0$ $q = 0$ term is not summed. This equation provides a very general and accurate long wavelength mesh impedance for use with (1). In addition this analysis can be used to make a good estimate of the frequency at which the mesh becomes resonant. Numerical studies show that at approximately the resonant frequency the matrix element Y^{00} becomes real so that $X_I \propto W \propto 1/Im(Y^{00})$ becomes infinite.

3.2 New calculation methods

Fig. 3 demonstrates the improved monomodal impedance formula (9) for a freestanding mesh with square holes. The fundamental propagating mode in the aperture is the TE_{10} mode. In this example the metal strips separating the squares are thin and the resonant wavelength is near the grid period. The monomodal formula gives a better estimation of impedance than the circuit model and provides better overall transmittance predictions. Furthermore as the strips become wider the equivalent circuit model becomes more inaccurate because the circuit impedance estimate gets worse and the resonance occurs at longer wavelengths further from the grid period. On the other hand the monomodal impedance estimation becomes better and the resonant frequency can always be accurately calculated.

The monomodal impedance method allows the accurate calculation of transmission curves without large mainframes and long run times. A typical rigorous grating solution can require 600 lines of Fortran code and 10,000 cpu seconds on a Cyber 170-730 series computer [14] as well as large memory requirements. The monomodal impedance method can run on a desktop—for example an IBM PC—with modest memory requirements and considerably less coding ($\sim$ 120 lines). Typical run times on a IBM PC are 2-3 minutes for a complete transmission curve.

The transmission curve of Fig. 4 is for a mesh with periodicity axes inclined at 45°. Such a structure can not be analyzed with existing equivalent circuit theory. The refined monomodal treatment gives an excellent impedance value and a good estimate of the position of the resonance (Y^{00} purely real). Fig. 5 shows the same mesh for radiation incident at 45°—non normal incidence is another situation that could not be properly accounted for with existing circuit theory. At non-normal incidence the transmission formula (1) is modified since the impedance of a TE wave changes from Z to $Z/\cos(\theta)$ where θ is the angle that the magnetic field makes to the plane of the mesh. If θ_i and θ_t are the incident and transmitted angles

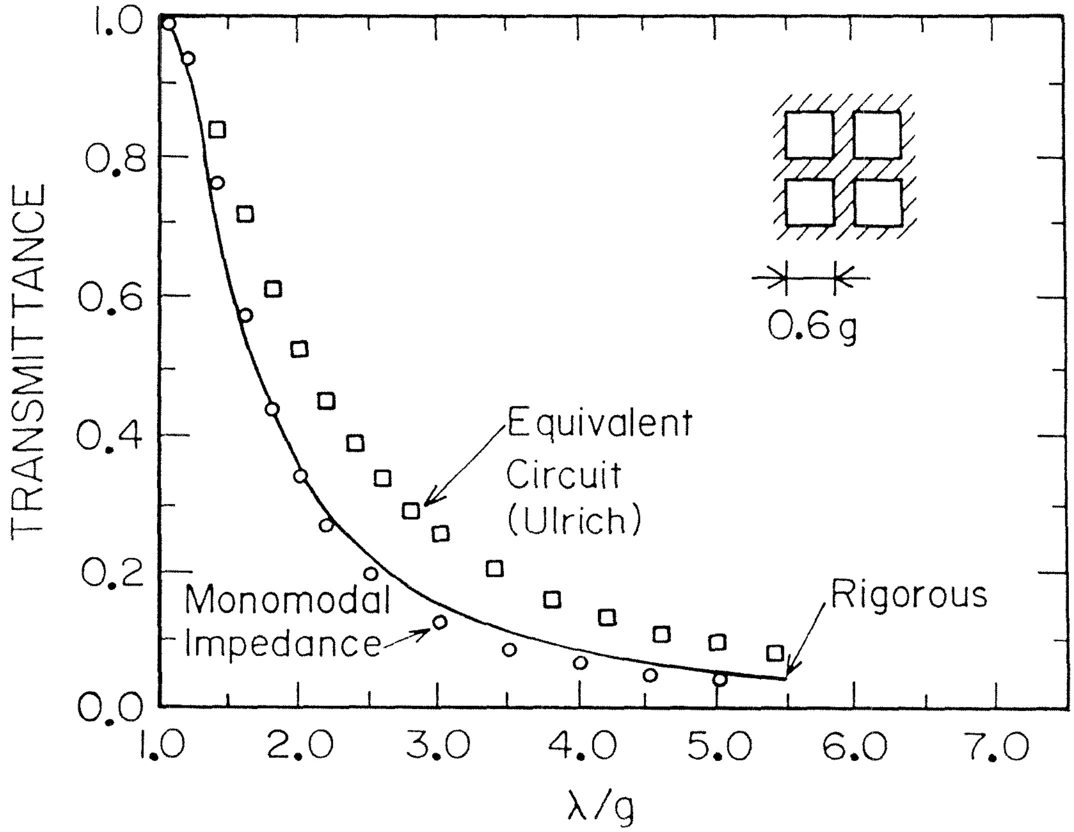


Figure 3. Comparison at normal incidence of monomodal impedance and circuit model transmission curves with rigorous moment solutions.

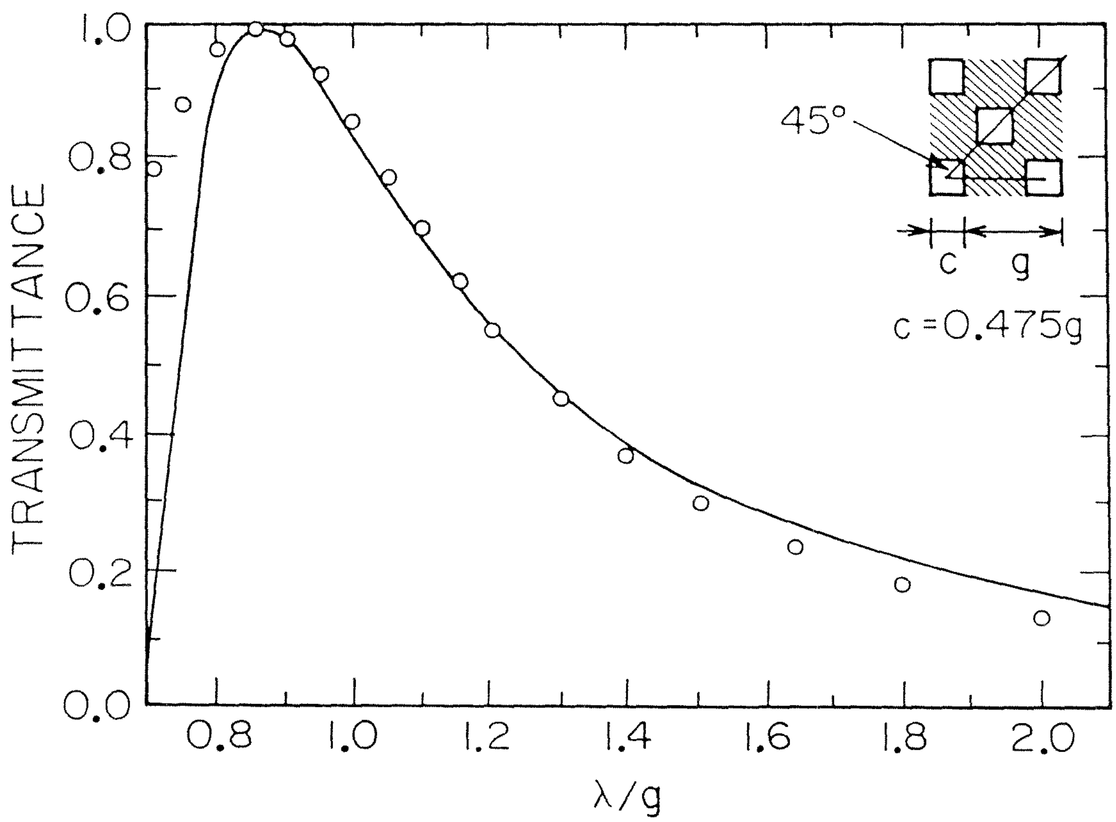


Figure 4. Transmittance curve for a square mesh at normal incidence with periodicity axes inclined at 45° comparing the rigorous solution (solid line) to the monomodal impedance approach (○).

then (1) becomes

$$T = \frac{4n\cos(\theta_i)\cos(\theta_t)}{[\cos(\theta_i) + n\cos(\theta_t)]^2 + \left(\frac{Z}{X}\right)^2} \quad . \tag{10}$$

A similar formula for a TM incident wave may be derived by using $Z\cos(\theta)$ for the transmission line impedance.

The monomodal impedance method can also take into account the presence of a dielectric. Fig. 6 shows the transmittance curve for a mesh at an air-dielectric interface. The inductive mesh impedance is unchanged. The dielectric alters the characteristic impedance on the transmission side of the structure and shifts the resonant wavelength. By modifying the transmission line impedance it is also a simple matter to consider a dielectric slab and allow for absorption loss in the dielectric.

This improved impedance formula can also be applied to circular holes or to any shaped hole or metal plate where the primary energy transmission mode can be calculated. If a more accurate estimation of the impedance is required then more terms in the sum of Equation (8) may be included.

3.3 Analysis of general shaped apertures

When the shape of the aperture becomes anything other than a strip, square or a circle it becomes extremely difficult to find the waveguide or current basis functions Ψ and Ξ used to express the electric field or current. For thin meshes there is no need to use these modes. Any set of independent functions that are continuous over the aperture and satisfy the boundary conditions on the aperture walls should be acceptable. This section will examine the use of some very simple functions for use in a wide range of mesh structures.

One of the simplest set of possible basis functions is the set of delta functions $\delta(x - x_n)$ where the x_n are points in the aperture. The method of moments then reduces to least square matching of the aperture field to the Floquet field above the mesh. Solutions obtained using delta functions are very sensitive to the positioning

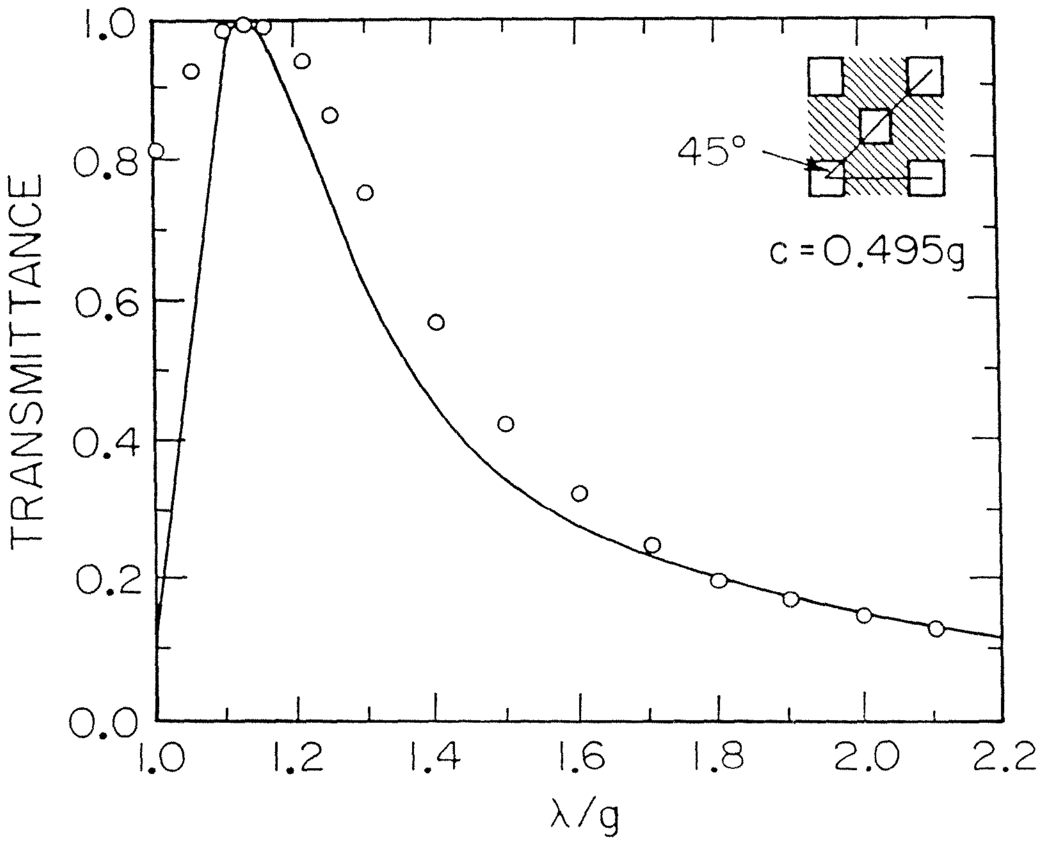


Figure 5. Transmittance curve for a square mesh with periodicity axes inclined at 45° and radiation incident at 45° comparing the rigorous solution (solid line) to the monomodal impedance approach (○).

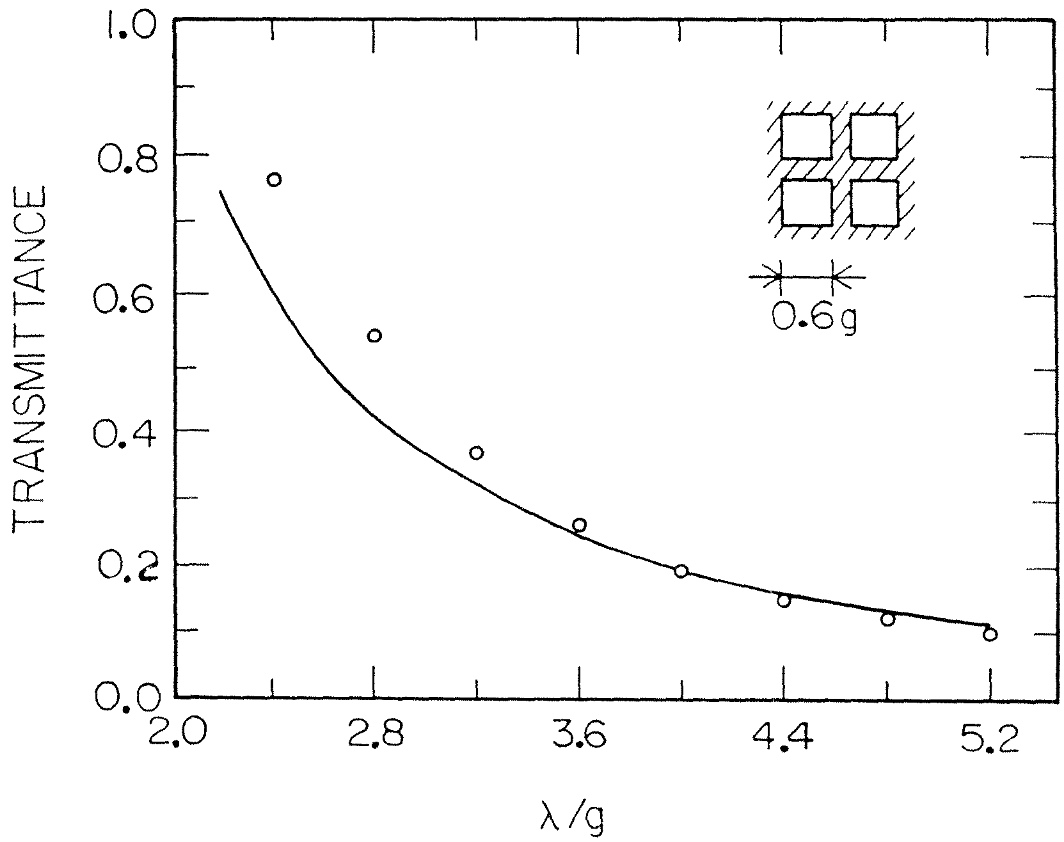


Figure 6. Transmittance curve for a square mesh at normal incidence at an air dielectric boundary (n= 2.1) comparing the rigorous solution (solid line) to the monomodal impedance approach (○).

of the points x_n. In addition, accurate solutions require a very large numbers of points and thus a large matrix (5) needs to be inverted. The major advantages of using delta functions is that they can be easily used for any general shaped aperture.

An examination of a typical aperture field obtained using traditional waveguide basis for a strip grating (Fig. 7 inset) shows that the field may be approximated by a simple polynomial expansion. The size of the matrix to be inverted is equal to the number of basis functions required to expand the field. It is important to exploit all symmetry properties of the structure under consideration because each symmetry reduces the size of the matrix by a factor ot two. For the configuration of Fig. 7 the electric field in the aperture may be expanded as

$$E_x = \sum_n c_n x^n \tag{11}$$

where symmetry (E_{incident} parallel to the x axis) requires that n be even. Fig. 7 shows that such a simple expression for the aperture field, together with (5) and (6), can produce extremely accurate results for the transmittance even though the field differs slightly from the field predicted using waveguide modes.

Polynomial expansions may also be used with great success in doubly periodic meshes. In the case of a mesh with square holes (Fig. 8) the following expression for the two electric field components may be used

$$\begin{aligned} E_x &= \sum_{mn} C_{mn} g_{mn} x^{2n+1} y^{2m+1} (y^2 - \frac{c}{2}) \\ E_y &= \sum_{mn} D_{mn} g_{mn} x^{2n} \quad y^{2m} \quad (x^2 - \frac{c}{2}) \ . \end{aligned} \tag{12}$$

The allowed powers of x and y are determined from the symmetry and the orientation of the incident E field (parallel to the y axis). The factor g_{mn} is chosen so that the integral of the square of the basis functions is normalized. If the functions are not normalized the matrix (5) can become unstable because of large variations in the magnitude of it's elements. The factors in brackets assure that the tangential field goes to zero on the walls of the aperture at $x = \pm\frac{c}{2}$ and $y = \pm\frac{c}{2}$. Fig. 8 shows that this polynomial expansion gives excellent agreement with the results predicted using the square waveguide modes.

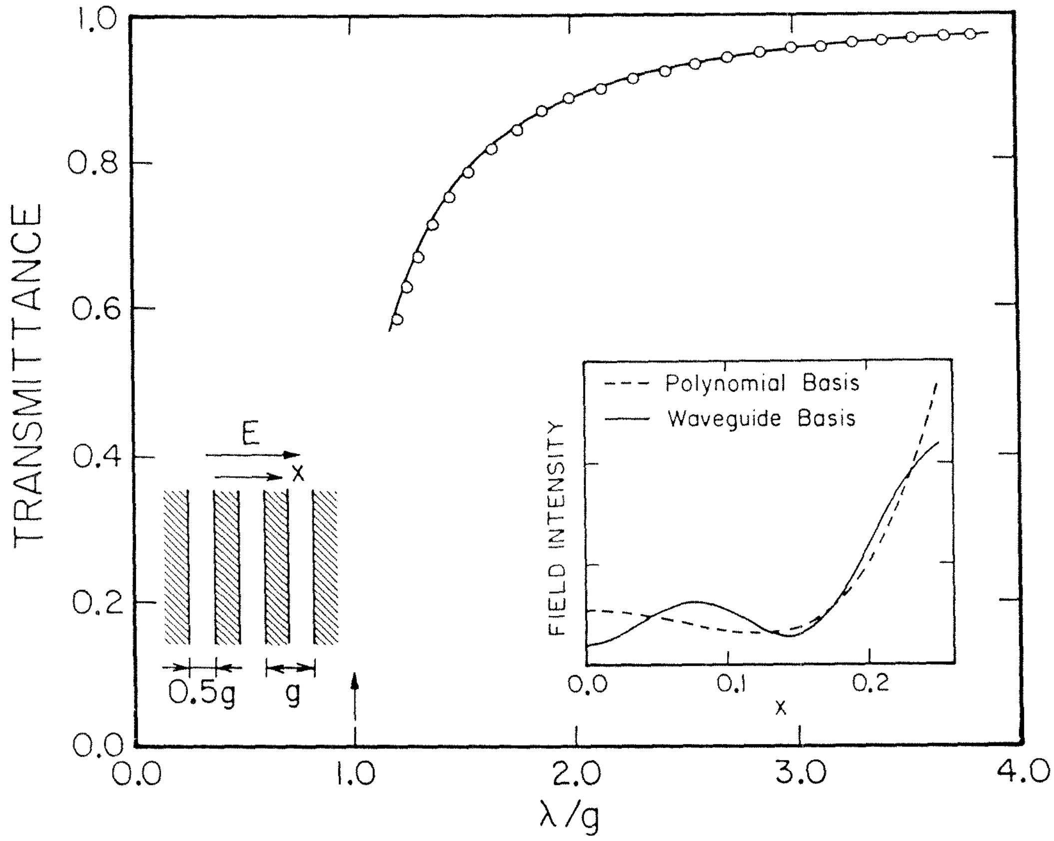


Figure 7. Transmission curve for a strip grating comparing waveguide basis approach (solid line) with a polynomial basis method (○).

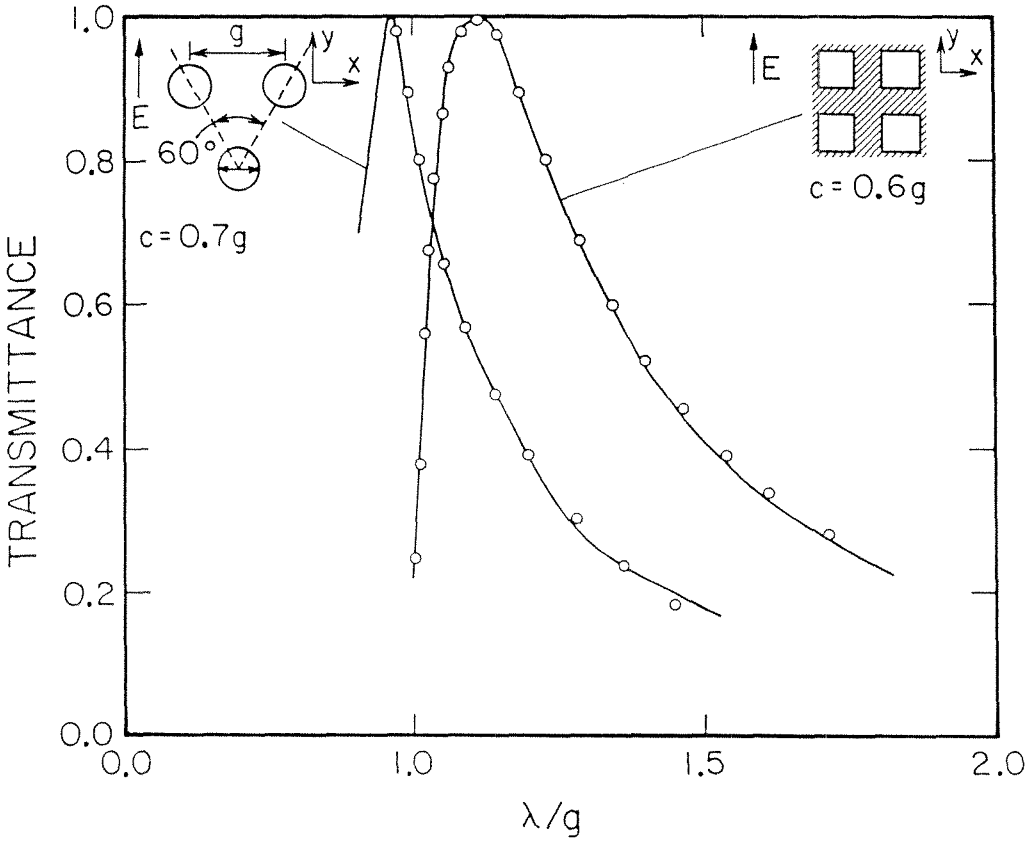


Figure 8. Transmission curve for meshes with square and circular holes [15] comparing waveguide basis approach (solid line) with a polynomial basis method (○).

For meshes with circular holes the radial dependence may be expressed in powers of r instead of the waveguide mode Bessel functions. The allowed angular dependence is chosen to take advantage of the x–y symmetry. The integrals required to fill the matrix, Eqns. (5)–(7) may be evaluated numerically using a modified Simpsons rule integration over the angular and radial variables. Fig. 8 demonstrates the accuracy of this expansion method for circular holes placed in an equilateral array.

If the structure consists of metal plates instead of holes then similar expansions may be used to describe the current in the plates.

There are a number of numerical checks that can be carried out to verify the solutions. Firstly conservation of energy requires that the energy carried away from the mesh equals the energy incident. In this formulation energy conservation is an analytical result but is a good check of the computer implementation. Another check is the common phase properties of the C_{mn}, D_{mn} and in particular

$$\text{Transmittance} = \sin^2(\text{Arg}(C_{mn})) \quad . \tag{13}$$

This provides a quick method of finding the transmittance without having to completely solve the matrix equation and reconstruct the Floquet fields. The solution may also be checked by comparing the aperture fields with the Floquet fields at the plane of the mesh. For the inductive case, numerical analysis suggests that continuity of the electric field is not necessarily a strong test of the solution but continuity of the normal magnetic field is a good measure of completeness of the aperture basis functions. Completeness of the Floquet modes can be checked with the following identity [14]

$$\sum_{pqr} C_{pqr}{}^{m} C^{*}_{pqr}{}^{M} = \int\int_{aperture} \Psi^m \cdot \Psi^M \quad . \tag{14}$$

This equation is an extension of the result obtained for use with waveguide basis functions but does not require that the basis functions be orthogonal or normalized over the aperture. The identity is exact for an infinite number of basis functions and Floquet modes and holds approximately when both series are truncated. Numerical stability may be checked by looking at the determinant of the matrix (5). This is an easy step to do as part of the matrix inversion. When the solution is unstable the determinant undergoes a rapid variation in which the phase changes by 180°.

When the shape of the hole or plate becomes complicated it is no longer possible to find simple functions which are continuous over the entire aperture and satisfy the boundary conditions. If the aperture basis functions are not continuous over the aperture the resulting discontinuity leads to spurious results. This problem can be avoided by splitting the aperture into smaller regions and requiring the functions defined over each region to vanish on the boundaries. If the boundary of the region coincides with the aperture wall then only the tangential electric field or the normal current must vanish.

To illustrate this technique consider the cross shaped aperture. The aperture is split into a number of smaller overlapping regions and the field is expanded in terms of two dimensional triangle functions. Fig. 9 shows good results using only a small number of functions. More accurate results can be obtained by using a larger number of smaller regions.

This approach may be used to solve a large number of boundary problems that could not be previously analyzed. Unlike other methods it makes no assumptions about the form of the fields or currents and places no restrictions on the size or shape of the hole. For example some techniques for analysing loaded slots and crossed dipoles make restricted assumptions about the currents or electric fields present based on the stipulation that the aperture or plate is narrow along one of its transverse directions [16–17].

The time required to obtain the transmittance at a given wavelength depends on the number of aperture modes which determines the number of linear equations which need to be solved. Very quick run times can be achieved by choosing a small number of good basis functions. In general run times are considerably shorter than for the rigorous method using waveguide modes.

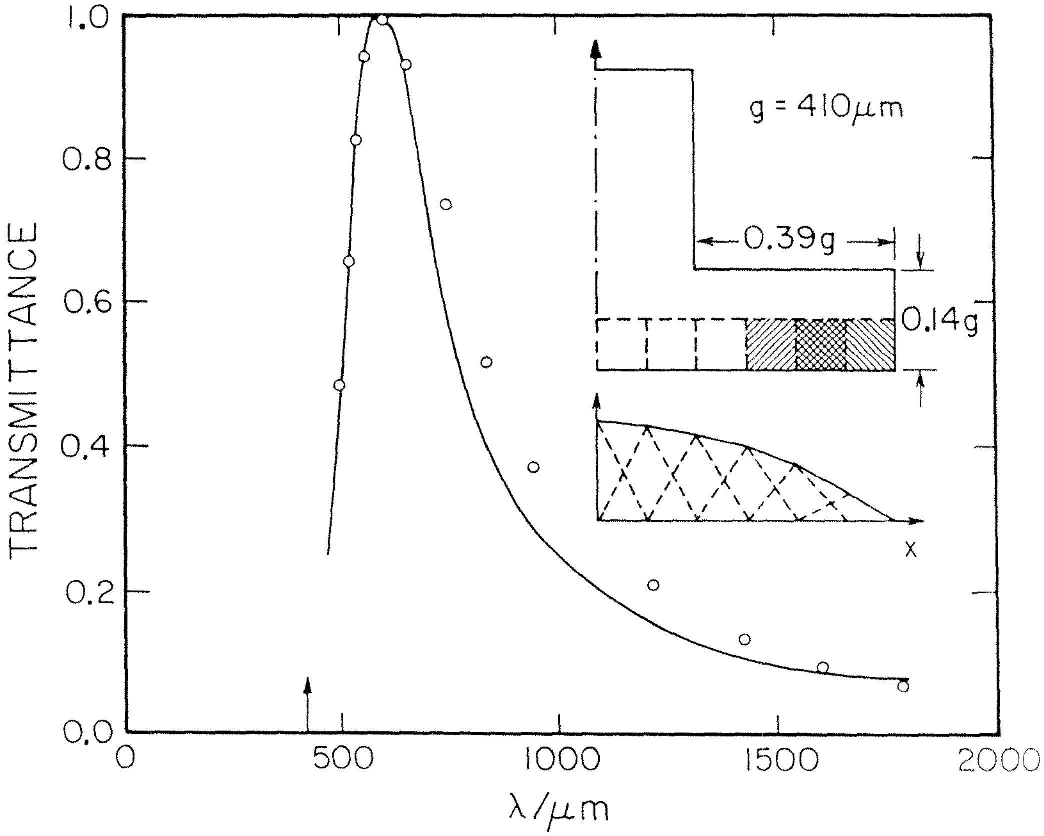


Figure 9. Transmission curve for a meshes with cross shaped holes comparing waveguide basis approach (solid line) with a two dimensional triangular basis method (○).

In the method of moments [18] the unknown field is expanded in terms of a complete set of basis functions and the resulting equation is projected onto a set of trial functions. Little work has been done on examining what constitutes a good set of trial functions and what restrictions need to be placed on the basis functions. In the above analysis the trial functions and basis functions are the same (Rayleigh-Ritz method). In certain applications it may be an advantage to use trial functions that differ from the basis functions. For example, in the case of the cross, step functions could be used as trial functions instead of triangle functions.

3.4 Circuit modeling for arrays with discrete loads

Millimeter wave antennas are becoming increasingly important in many scientific and military applications. In these applications there is a need for integrated detector arrays for use as high efficiency imaging systems [19] (Fig. 10). Such a system might consist of a two dimensional array of squares (Fig. 11) joined by bolometers, SIS detectors or Schottky diode detectors, mounted on a substrate lens [21,22,23]. In this structure each unit cell looks like a 90° bow tie. These arrays may be analyzed using the methods outlined in section 1 to find an equivalent network for the array.

3.5 Equivalent circuit theory

In the design of an efficient array it is important to match the array impedance to the incident wave to get maximum power coupled into the detector. Analysis of the array is simplified by using an equivalent circuit. A plane wave incident on the antenna array is modeled by a traveling wave on a transmission line. The array is represented by a two-port network in which one port is shunted across the transmission line and the second port is connected to the detector (Z_d). The impedance matrix Z for the two-port is calculated by considering the properties of the mesh when the second port is open and closed. These scattering properties are calculated rigorously using the method of moments. The short-circuit case consists of a periodic array of square holes in which the aperture field may be expanded in

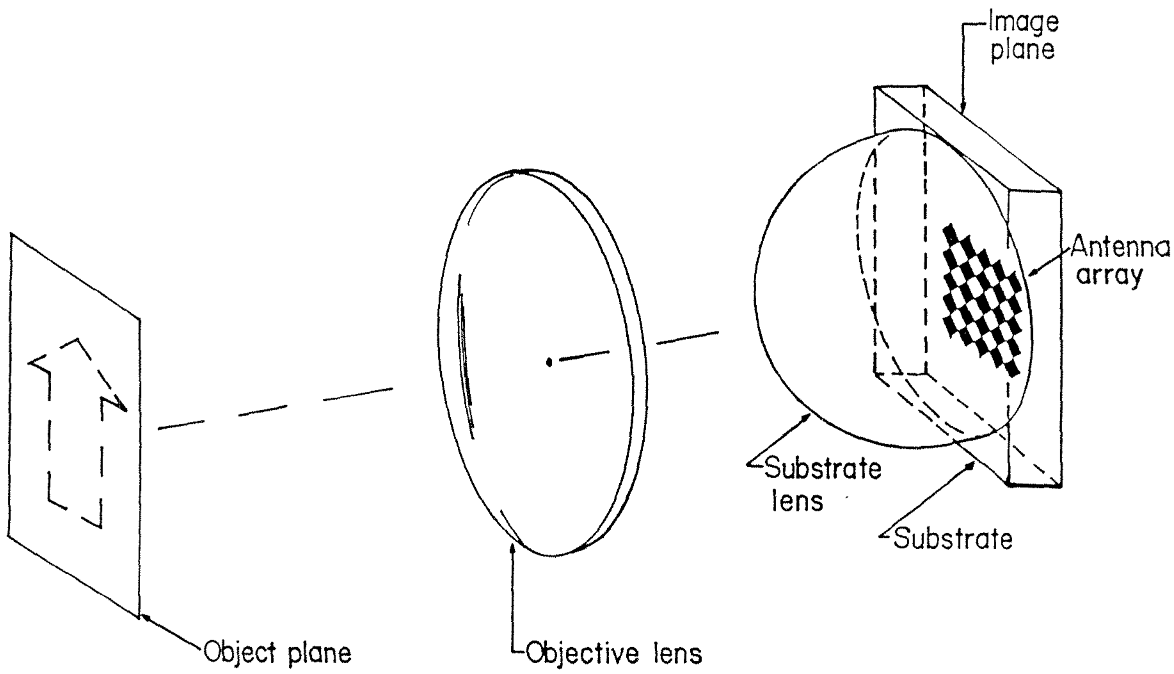


Figure 10. Imaging system in which an array of detectors are fed from antennas placed in the focal plane of an imaging system. The substrate lens couples the radiation into the array while minimizing losses due to substrate modes.

terms of TE/TM waveguide modes. Similarly the open-circuit case may be analyzed as an array of metal squares whose currents may be described by modes that are dual to the waveguide modes. The solutions are checked using conservation of energy, reciprocity and Babinet's principle [5]. For the free-space array the open and short-circuit impedances are related by Babinet's principle. Reciprocity requires that the impedance matrix Z be symmetrical. Once the impedance matrix of the array is obtained the absorption efficiency for any given detector impedance may be calculated.

The impedance matrix Z was calculated for the array sitting on an dielectric substrate with refractive index n. At long wavelengths the two-port was found to be equivalent to a transmission line with characteristic impedance equal to the quasi-static impedance of an infinite bow-tie antenna (Eq. 1 Chapt. 1). The coupling efficiency is defined as the power absorbed by the detectors divided by the incident power. For radiation incident normally from the dielectric side it can be shown that the maximum efficiency occurs when the impedance looking in at port 1 is purely real and given by $377/(n+1)\Omega$. For this impedance the efficiency is $n/(n+1)$. For the bow tie array (Fig. 11) the electrical length of the two-port is small, for $\lambda/d = 4.5$, $l = 0.1\lambda$, so the input impedance becomes very nearly equal to Z_d. This predicts that for bolometers with resistance close to the optimal value very high efficiencies should be obtainable (78% for $n = 3.5$).

3.6 Fabrication and measurement

Several arrays with a period of 1-mm have been built for operation at 94 GHz. They were fabricated on a quartz substrate 1/16″ thick, using conventional lithography and liftoff. The square patches are evaporated silver 1000Å thick. The detectors used are bolometers. They are fabricated by making a photoresist air-bridge at each corner of the square and then evaporating bismuth at an angle on each side of the bridge. The final bismuth thickness was 1000Å and the DC resistance of the bolometers was in the range 100 Ω to 130 Ω.

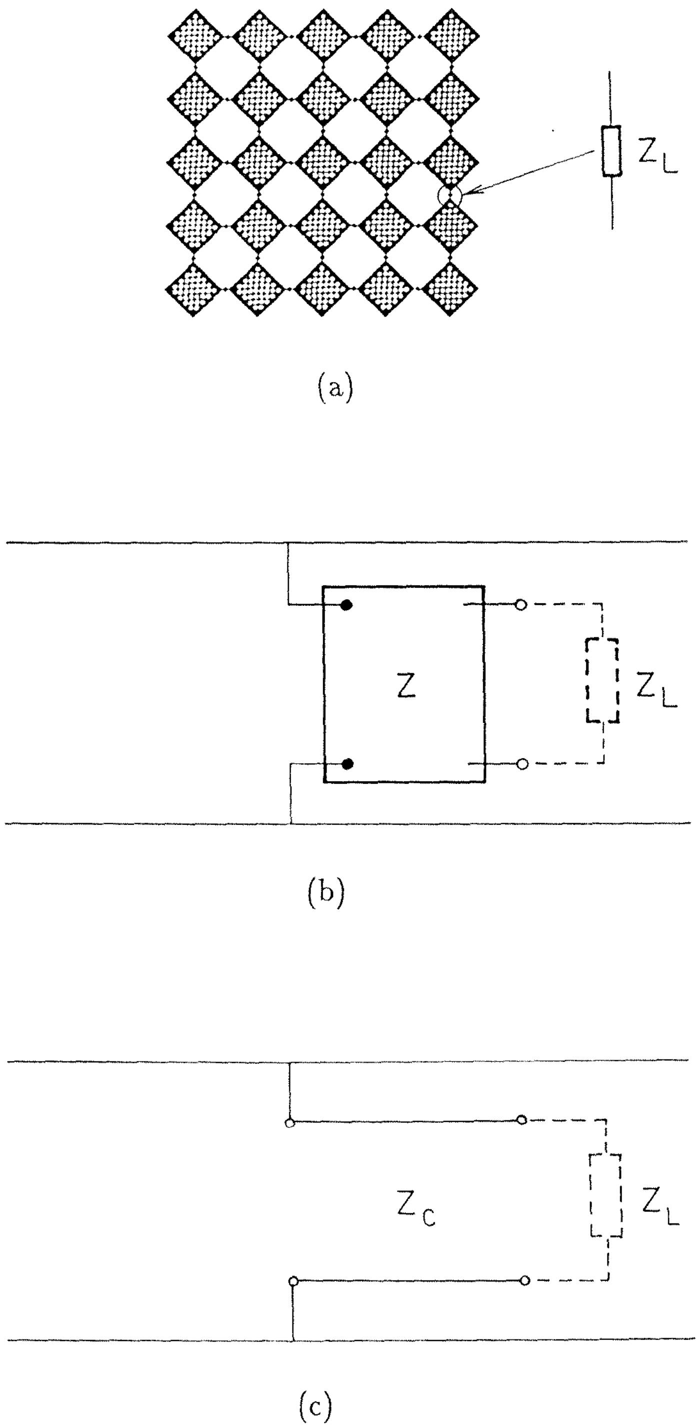
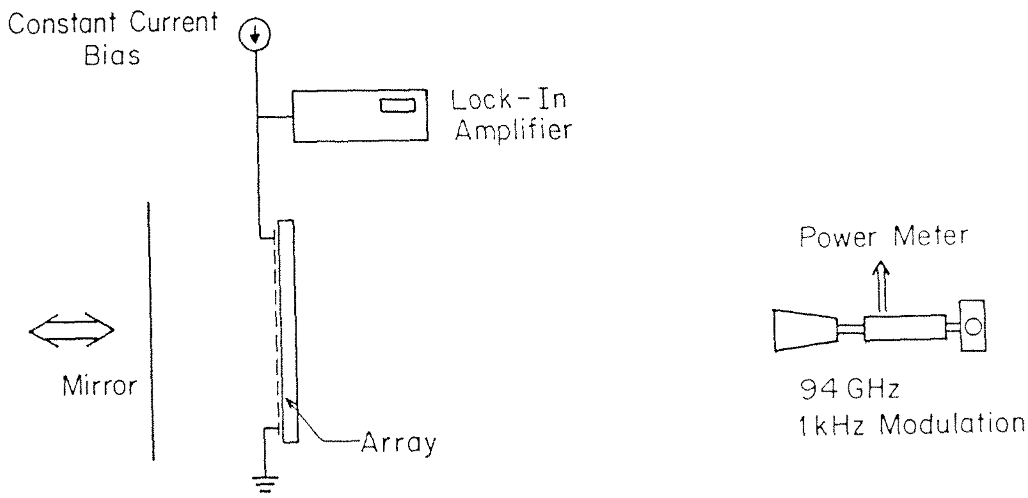
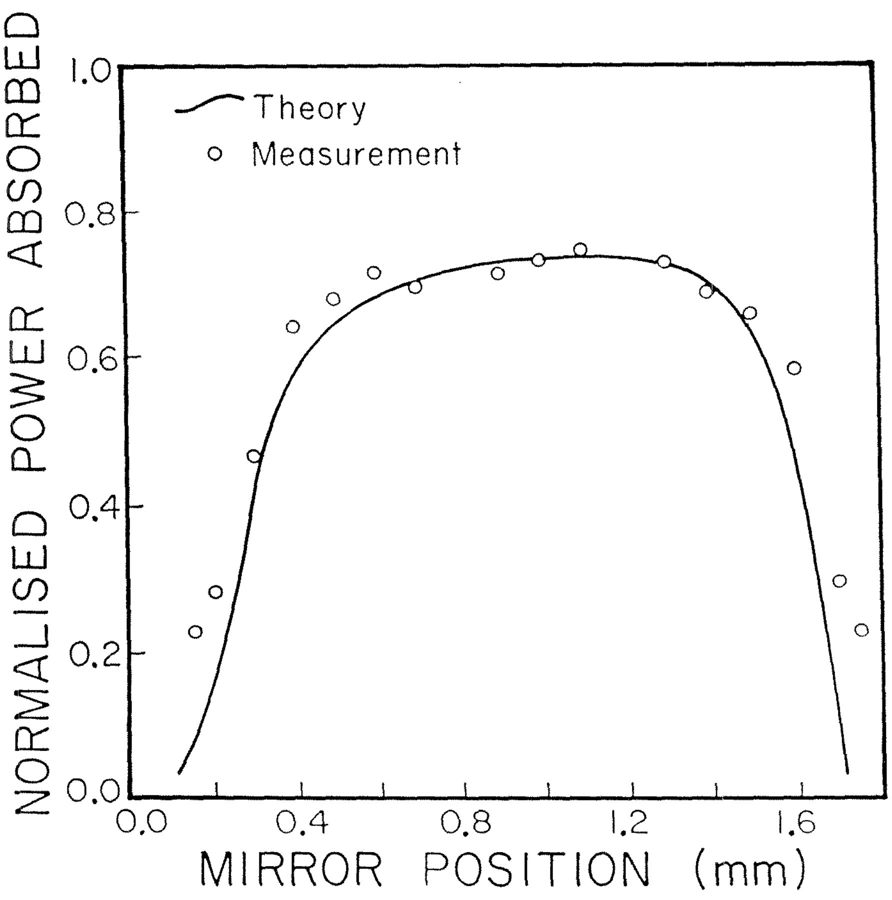


Figure 11. (a) Two dimensional array with discrete loads. (b) Equivalent circuit. (c) Simplified model in which the array is equivalent to a shunt section of line whose impedance approaches the bow-tie antenna quasi-static impedance.



(a)



(b)

Figure 12. (a) Experimental setup for measuring array impedances. (b) Power absorbed for an array on a quartz substrate.

The array was in placed in front of a 94-GHz source and the bolometers were biased with a constant current supply. The voltage across the array is then proportional to the power absorbed by the array. By placing a movable mirror behind the array a tuning curve of power absorbed versus mirror position may be measured. These measurements were compared with the equivalent two-port theory (Fig. 12).

3.7 Array impedance and pattern measurements

By placing the array in the focal plane of an optical system [20] very high efficiencies for the bow tie array can be obtained. In Figure 13 contours of normalized power absorbed for an array on a substrate with $n = 3.5$ are calculated. In these calculations the array is characterized by a uniform sheet resistance and reactance. This impedance depends on the impedance of the detectors and the array geometry. The incident field is decomposed into an angular spectrum [20] and the equivalent mesh impedance is assumed constant for small angles of incidence. The calculations predict very high coupling efficiencies.

Antenna pattern measurements were made at 60 GHz and 94 GHz using the same antenna range discussed in the previous chapter. These measurements are compared with theory in figures 14 and 15. The theoretical calculations were performed by assuming the mesh equivalent impedance remains constant independent of angle of incidence. This assumption breaks down at large angle of incidence.

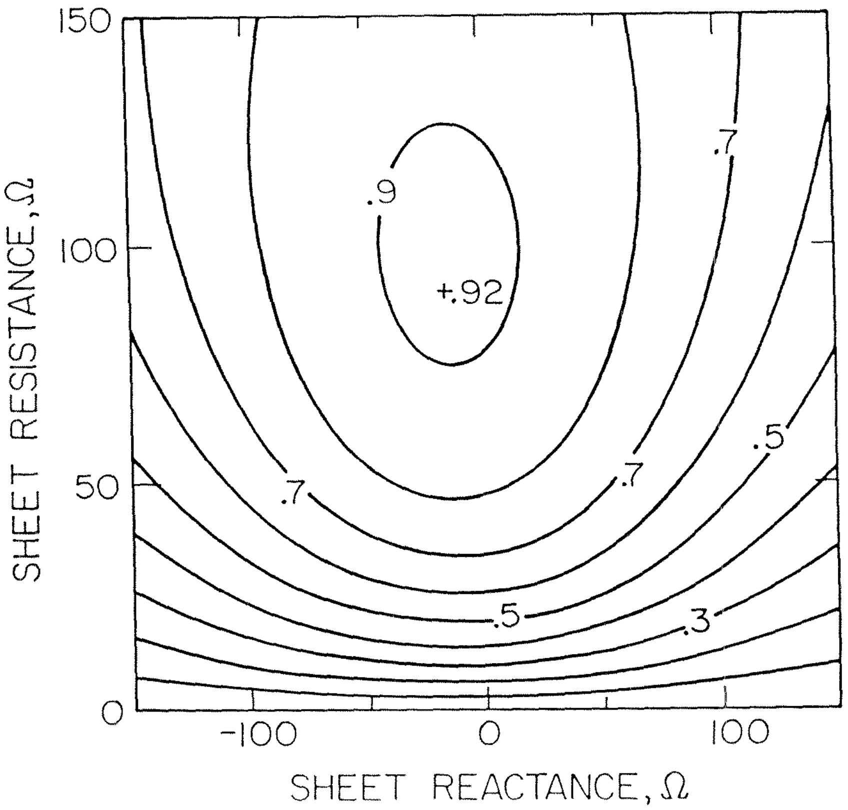


Figure 13. Theoretical efficiency for an imaging array of Fig. 10 as a function of equivalent sheet resistance and reactance.

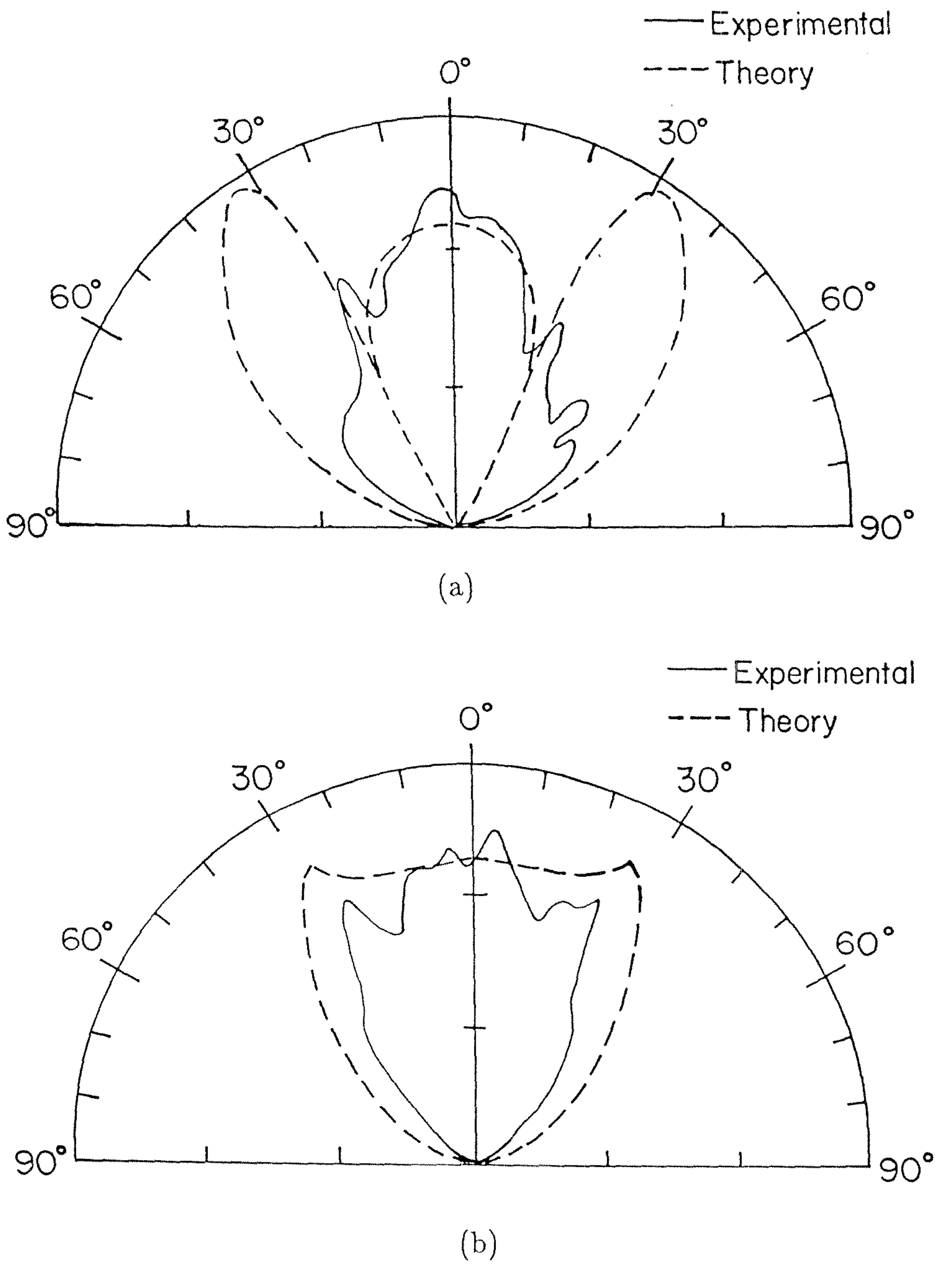


Figure 14. Measured (a) E plane and (b) H plane antenna array patterns at 94 GHz.

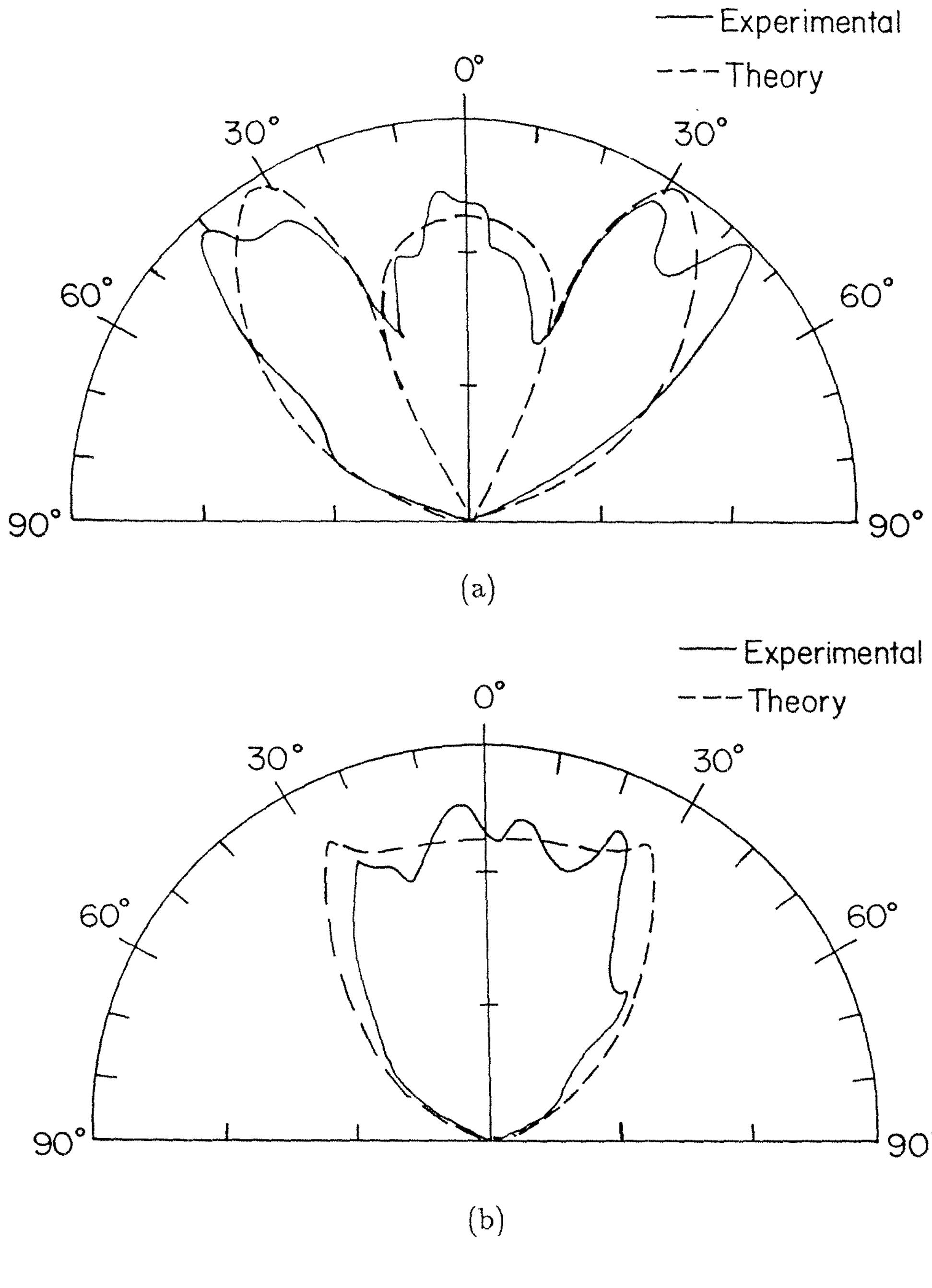


Figure 15. Measured (a) E plane and (b) H plane antenna array patterns at 60 GHz.

References

[1] M. S. Durschlag and T. A. DeTemple, "Far-IR optical properties of freestanding and dielectrically backed metal meshes," *Appl. Opt.*, **vol. 20**, pp. 1245–1253, April 1981.

[2] R. Ulrich, "Interference filters for the far infrared" *Appl. Opt.*, **vol. 7**, pp. 1987–1995, Oct. 1968.

[3] R. C. McPhedran and D. Maystre, "On the theory and solar application of inductive grids" *Appl. Phys.*, **vol. 14**, pp. 1–20, Jan. 1977.

[4] H. M. Pickett, J. Farhoomand, A. E. Chiou, "Performance of metal meshes as a function of incidence angle," *Appl. Opt.*, **vol. 23**, Dec. 1985.

[5] R. C. Compton, J. C. Macfarlane, L. B. Whitbourn, M. M. Blanco and R. C. McPhedran, "Babinet's principle applied to ideal beamsplitters for millimeter waves," *Optica Acta,* **vol. 31**, pp. 515–524, May 1984.

[6] C. C. Chen, "Transmission through a conducting screen perforated periodically with apertures," *IEEE Trans. Microwave Theory Tech.* **vol. MTT-18**, pp. 627–632, Sept. 1970.

[7] C. C. Chen, "Scattering by a two-dimensional periodic array of conducting plates," *IEEE Trans. Antennas Propagat.*, **vol. AP-18**, pp. 660–665, Sept. 1970.

[8] R. C. McPhedran, G. H. Derrick and L. C. Botten, *Electromagnetic Theory of Gratings*, Springer, Berlin, 1980.

[9] R. Ulrich, "Far infrared properties of metallic mesh and its complementary structure," *Infrared Physics*, **vol. 7**, pp. 37–55, Jan. 1967.

[10] R. C. Compton, L. B. Whitbourn and R. C. McPhedran, "Simple formula for the transmittance of strip gratings," *Int. J. Infrared and Millimeter Waves*, **vol. 4**, pp. 901–912, Nov. 1983.

[11] N. Marcuvitz, *Waveguide Handbook*, MIT Radiation Laboratory Series, vol. **10**, McGraw Hill, New York, 1951.

[12] R. C. Compton, L. B. Whitbourn and R. C. McPhedran, "Strip gratings at a dielectric interface and applications of Babinet's principle," *Appl. Opt.*, **vol. 23**, pp. 3236–3242, Sept. 1984.

[13] L. B. Whitbourn and R. C. Compton, "Equivalent circuit formulae for metal grid reflectors at a dielectric boundary," *Appl. Opt.*, **vol. 24**, pp. 217–220, Jan. 1985.

[14] R. C. Compton, R. C. McPhedran, G. H. Derrick and L. C. Botten, "Diffraction properties of a bandpass grid," *Infrared Physics*, **vol. 23**, pp. 239–245, May 1983.

[15] C. C. Chen, "Diffraction of electromagnetic waves by a conducting screen perforated periodically with circular holes," *IEEE Trans. Microwave Theory Tech.*, **vol. MTT-19**, pp. 475–481, May 1971.

[16] R. J. Luebers and B. A. Munk, "Cross polarization losses in periodic arrays of loaded slots," *IEEE Trans. Antennas Propagat.*, **vol. AP-23**, pp. 159–164, March 1975.

[17] C. Tsao and R. Mittra, "Spectral-domain analysis of frequency selective surfaces comprised of periodic arrays of cross dipoles and jerusalem crosses," *IEEE Trans. Antennas Propagat.*, **vol. AP-32**, pp. 478–486, May 1984.

[18] R. F. Harrington, *Field Computation by Moment Methods*, Macmillan, New York, 1968.

[19] D. B. Rutledge et al, *Integrated Circuit Antennas*, Infrared and Millimeter-Waves Series, **vol. 10**, K. J. Button, Ed., Academic Press, New York, 1983.

[20] C. Zah, R. C. Compton and D. B. Rutledge, "Efficiencies of elementary integrated-circuit feed antennas", *Electromagnetics*, **vol. 3**, pp. 239–254, 1983.

[21] R. C. Compton and D. B. Rutledge, "Circuit modeling of planar meshes with discrete loads," SPIE 29th Annual International Technical Symposium on Optical and Electo-Optical Engineering, San Diego California, August 1985.

[22] R. C. Compton, G. M. Rebeiz and D. B. Rutledge, "Developments in two di-

mensional arrays," *Tenth International Conference on Infrared and Millimeter Waves*, Orlando Florida, December 1985.

[23] R. C. Compton and D. B. Rutledge, "Optical techniques at millimeter wavelengths," *Optical Society of America Annual Meeting*, Seattle Washington, October 1986.

Chapter 4

Puff, an interactive program for CAD of microwave circuits

The emergence of low-cost, powerful personal computers has opened the door for many new areas in computer aided design. The personal computer lends itself well to interactive design programs requiring high-resolution graphics and quick response times. A microwave CAD program, called *Puff*, that has been developed for the IBM family of personal computers will be discussed. *Puff* allows new approaches for designing microstrip circuits and has proved to be a useful tool in microwave laboratory classes.

The first IBM Personal Computer was announced in the fall of 1981. Since then, total sales of IBM PC's and compatibles have exceeded 10 million and several faster and more advanced models have been introduced. Today's PC's are capable of many tasks that were previously not possible with desk-top systems. With math-coprocessors such as the Intel 8087, which is capable of 60,000 floating-point multiplications per second [1], complex numerical calculations can be performed with a high degree of accuracy. A number of software companies have recognized the possibilities of the PC and have developed programs for engineering applications, including microwave circuit design. However, there are several problems associated with using commercially available microwave CAD packages in a university. These programs are expensive, and usually come with copy-protection devices. This makes it difficult to distribute them on a large number of machines. The algorithms and modelling are usually proprietary, and this often forces the documentation to be incomplete. This makes it difficult to scrutinize, discuss or improve any of the techniques used. In addition, the source code is not available so the programs cannot be modified or extended. As a result, these programs tend to become black boxes—they provide answers, but little information about how the analysis is performed or the limitations or accuracy associated with the modelling. Furthermore, these packages often do not take full advantage of the interactive capabilities of the PC, in part because they have evolved from larger minicomputer and mainframe versions.

To address these problems, we have written a new microwave computer-aided-design package, called *Puff*, after the magic dragon in the folk song by Peter, Paul and Mary. *Puff* runs on the IBM PC and compatible machines. *Turbo Pascal* was chosen for *Puff* because of its excellent compiling speed, low cost and availability—Borland International has sold over 500,000 copies of this compiler. *Puff* has been designed primarily for use in a microwave laboratory course at the California Institute of Technology [2]. The development of new integrated-circuit technology based on microstrip has created the opportunity for a new kind of microwave circuit class, where the students lay out and analyze circuits, and make the photographic artwork, all on a personal computer. The students at Caltech attend two fifty-minute lectures each week on microwave circuit fundamentals. They design their own circuits as a homework exercise and fabricate and measure them in a two-hour laboratory session. Over a ten-week period, they build a rat-race and branch-line coupler, a low-pass and bandpass filter, a patch antenna with a matching network, an amplifier and an oscillator. The difficulties in manually performing the s-parameter calculations make this an ideal application for computer aided design and education. With *Puff* the user can try a large number of designs in a short time and gain an understanding of circuit behavior without being bogged down with complicated number manipulations. One of the main objectives in developing *Puff* was to make an easy-to-use program for studying microstrip circuits. If the program is too complex the user spends more time learning about a specific program than finding out about microwave circuits. The user interface is also very important for encouraging student interest. *Puff*'s interface makes it easy to interactively tweak circuit parameters and gain a feeling for how the performance varies. For this reason *Puff* does not contain any optimization routines, and effects due to parasitic discontinuites are not automatically included. The object is to have the user manually enter the corrections and compare the predictions with measurements to gain an appreciation of the relative significance of these effects. *Puff* is *not* a program for designing commercial circuits. There is a limited range of circuits that can be designed. In addition, *Puff* neglects dispersion, loss, parasitic effects of discontinuities and nearby lines, radiation, and nonlinearities, and the user-defined devices are limited to two-ports. *Puff* does not analyze nonlinear circuits or noisy networks.

4.1 Using Puff

To run *Puff*, you need an IBM PC, AT, or compatible with at least 512 kilobytes of memory and a math-coprocessor chip, running DOS 2.0 or later. You need a graphics card and a graphics display. The program works best with an *Enhanced Graphics Display* with an *Enhanced Graphics Adapter* (*EGA*) that has 256 kilobytes of video memory. However, it will also work with a *Color Graphics Adapter* (*CGA*) and a single-color graphics display or a *Color Graphics Display*. To make hard copy, you will need a screen-dump routine for your particular display and printer. If you have a *Color Graphics Adapter* and an *IBM Proprinter* you can use the program `graphics.com` that comes with the *DOS* distribution disk. A screen-dump program `egapro.com`, which will make a screen dump with an *EGA* and an *IBM Proprinter* is included on the distribution disk. It works the same way as `graphics.com`. Just run `egapro` once before you use *Puff*. To make a screen dump hit the *PrtSc* key while you are holding down the shift key, ⇑.

Perhaps the best way to learn more about *Puff* is to sit down at a PC and experiment with *Puff*. The following examples give a feeling for how *Puff* can be used. Before you get started, you should make a copy of the original *Puff* disk. To run the program, put the copy in the current drive and type `puff` and the return key. *Puff* will display a copyright notice, and then load a circuit file `setup.puf`. *Puff* should set up the display in Fig 1. The word `setup` should appear on the left side of the screen to indicate that `Puff` used this file to start. You can specify a different starting file, say `yourfile.puf`, on the command line by typing `puff yourfile` when you start *Puff*. *Puff* assumes the extension `.puf` if you do not give one.

The screen is organized in window blocks. You can move between windows by pressing one of the function keys *(F1, F2, F3)*. The block in the upper right corner

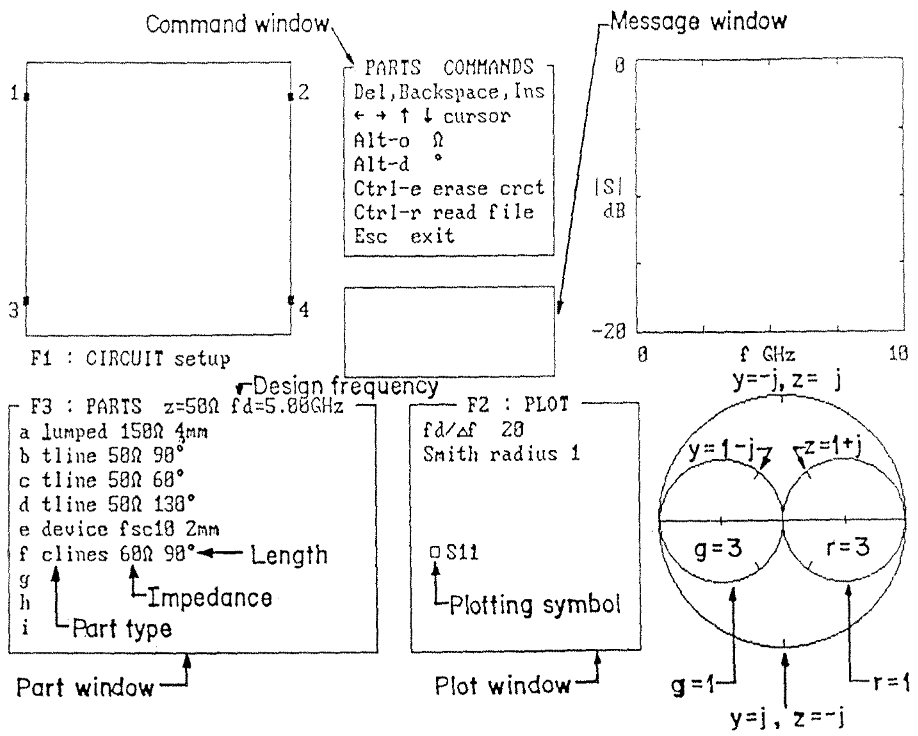


Figure 1. A screen dump from an *EGA* display. The screen dump for the single-color version with a *Color Graphics Adapter* is a little different.

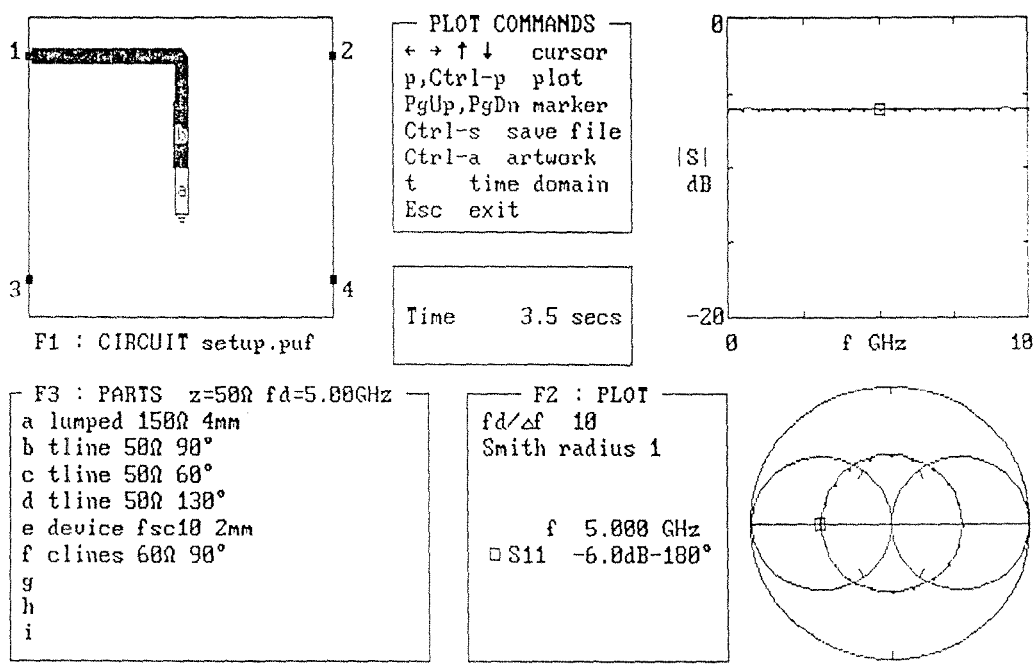


Figure 2. Circuit that consists of a quarter-wavelength section of transmission line with a 150-Ω load.

is the *Circuit* window. This is where the user draws a microstrip circuit. The numbers around the side represent connectors, where in practice measurements on the circuit are made by attaching a coaxial cable. The circuit window is reached by typing *F1*. At top center is the *Command* window, where the currently active commands are listed. You can exit *Puff* and return to *DOS* by hitting the *Esc* key. Below the *Command* window, in the center of the screen, is a three-line *Message* window for displaying error messages and requests for file names. In the upper right corner is the rectangular plot. The rectangular plot can be either a frequency plot that shows the magnitude of the scattering coefficients, or a time domain plot of the scattered waves. On the bottom right is a Smith chart that gives a polar plot of the scattering coefficients. To the left of the Smith chart is the *Plot* window, which can be reached by pushing *F2*. In the plot window, you can generate the plots and see the numerical values of the scattering coefficients. The final window, in the lower left corner, is the *Parts* window, reached by typing *F3*. The *Parts* window gives the current list of parts that may be used in the circuit. You can change or add to the parts list when you are in the *Parts* window.

When *Puff* begins it starts out in the *Parts* window, and this is indicated in several ways. There is a flashing cursor in the *Parts* window, *Parts* commands are listed in the *Command* window, and the F3 at the top of the *Parts* window appears in white. Now hit *F1* to move to the *Circuit* window. Several things will happen. The F3 will become yellow, and the F1 above will turn white. A set of commands for the *Circuit* window will appear in the *Command* window. A large white cursor will appear in the middle of the circuit board. Finally, line a in the parts window will be highlighted in white to indicate that that part is selected. Part a is a lumped 150-Ω resistor, and it is 4 mm long. Now push the down arrow key, ↓. *Puff* will draw a hollow blue box, labeled a, down the screen, and the cursor will move to the other end of this box. In the *Message* window,

△y − 4.00mm

will appear. This shows you the change in the y coordinate of the cursor. It is negative to indicate that the cursor moved down, and it is 4 mm because the resistor is 4 mm long. Now hit the = key and *Puff* will short that end of the resistor to ground. Hit the up arrow key, ↑, to move to the other end of the resistor. A 90° section of 50-Ω transmission line is added by first typing b to select that part. Now type ↑, and *Puff* will draw a filled brown rectangle up the screen, labeled *b*. This represents a quarter-wavelength section at the design frequency, 5 GHz. The width and the length of the line are drawn to scale. For reference, the circuit board itself is 25.4 mm across, and the separation between the connectors is 19 mm. In the *Message* window,

△y 5.69mm

will appear, and this tells you how long the section is. If you type 1. *Puff* will make a gray path up and to the left to join to the first connector. The width of the gray path is the same as the 50-Ω line. The cursor will not move. This completes our circuit, which consists of a section of 50-Ω line with a 150-Ω load. The screen is shown in Fig. 2. You may wonder why we need to ground the resistor. The reason is that without the ground, *Puff* thinks that the end of the resistor is open-circuited. The load would be a 150-Ω resistor in series with an open-circuit, which is just an open circuit.

To do the analysis, push *F2* to go to the *Plot* window and p to make a plot. *Puff* will plot the complex reflection coefficient s_{11} on the circular Smith chart in the bottom left corner of the screen. The calculated points are shown as small squares, and when the analysis is completed they are joined by a spline curve. The interpolation is performed by splining the real and imaginary values of the scattering coefficients separately. The smoothest curves were obtained when the calculated points were parameterized by their spacing on the Smith chart rather than by frequency [3]. When *Puff* is finished with a plot, it will let you know how long the analysis took in the message box. The plot for our circuit is a circle of radius one-half. The magnitude of the reflection coefficient is plotted on the rectangular plot in the upper right corner of the screen. A time domain plot is obtained by pushing t. For the circuit of Fig. 2 you will see a positive pulse of height 0.5, delayed 100 ps. The delay comes from the 90° section of transmission line.

The circuit can be matched by adding a section of transmission line to make s_{11} equal to zero at the design frequency. A quarter-wave section of 87-Ω $(\sqrt{150 * 50})$ line can be used to match a 50-Ω line to a 150-Ω load. To change the impedance of the transmission-line section b, first push *F3* to go to the parts window. Then use the cursor keys to move to the 5 on line b, and type 87 to change the 50 to 87. Push *F2* to return to the *Plot* window. *Puff* will redraw the circuit for you, and you will notice that the transmission-line section has become narrower, because its impedance is larger. Push p to plot again. The screen should look like the one shown in Fig. 3. The s-parameter curve has shrunk so that it passes through the origin at the design frequency, 5 GHz. The design frequency is marked on the plots by a small red box.

We will conclude by finding a stub-matching circuit. Type *F1* to go to the circuit window, and check that the cursor is at the top of the b transmission-line section. Hold down the shift key ⇑ and type 1 to erase the path to the connector. Then hold down the shift key and hit the down arrow key ↓ to erase the b transmission line. The cursor will move to the a resistor. Now type c, ↑, d, →, ←, and 1 to make the stub-matching section shown in Fig. 4. Push *F2* and p to analyze the circuit. This makes a match, too, but it works over a narrower band than the quarter-wave match.

4.1.1 The Parts window

The *Parts* window contains a list of the parts that may be used in the circuit. At the top of the window is the normalizing impedance, z and the design frequency, fd. These numbers are loaded from the circuit file setup.puf. *Puff* also gets its initial parts list from this setup file, and you can edit this file to suit your specific needs. You can also read in a new .puf file in the *Parts* window by typing *Ctrl*-r.

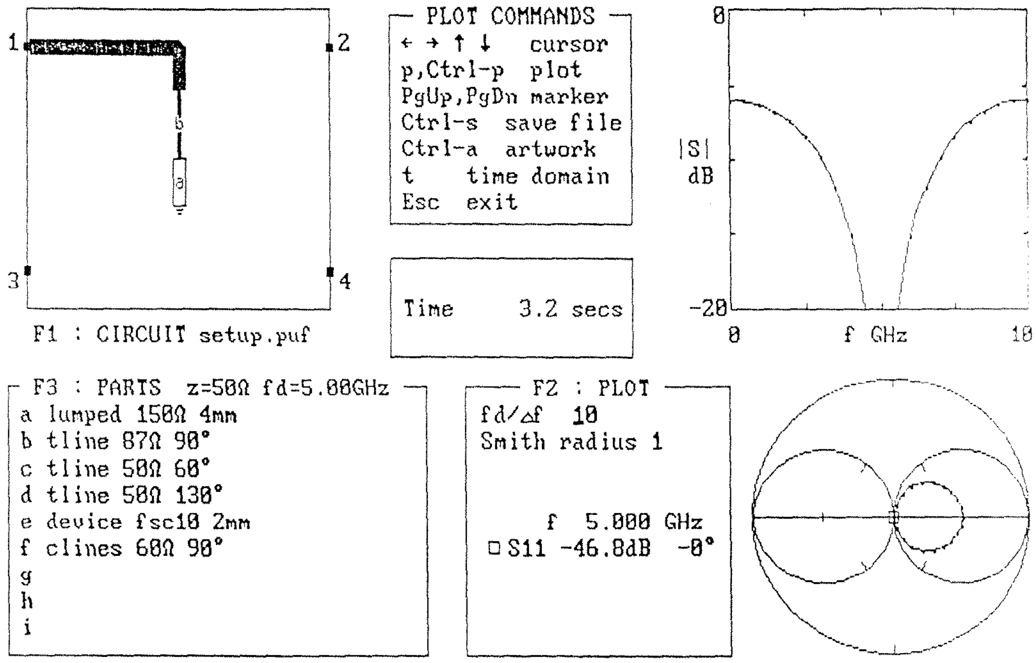


Figure 3. Circuit consisting of a quarter-wavelength matching section of transmission line with a 150-Ω load.

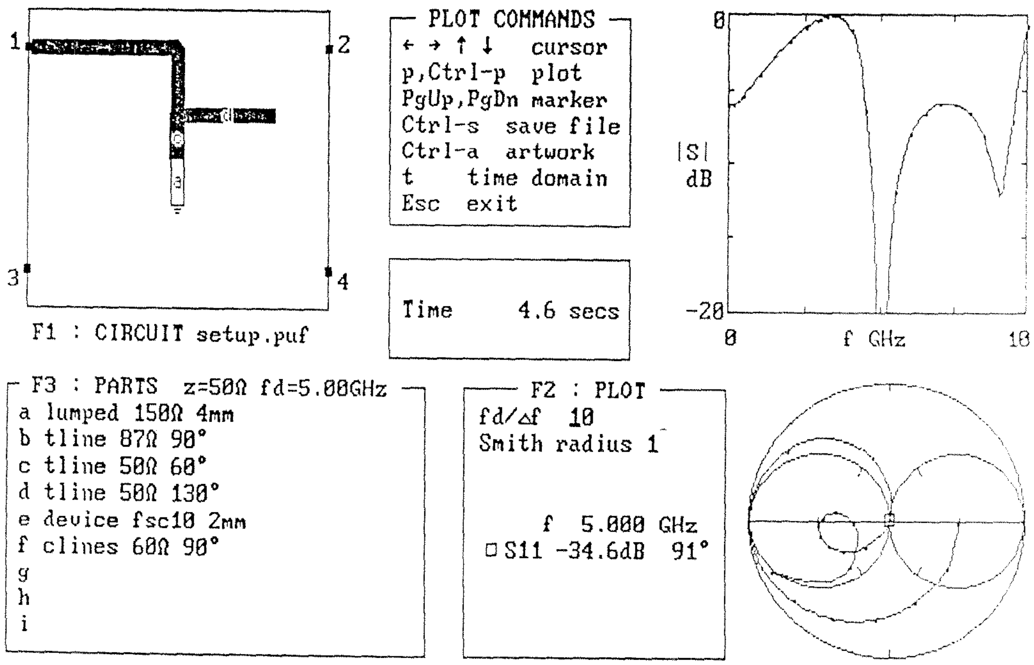


Figure 4. Stub matching circuit.

You will be prompted for a file name. The parts list can be changed with the cursor keys, and the backspace, *Ins*, and *Del* keys. The first letter on each line, (from `a` to `i`), identifies the part in the circuit at the top of the screen. *Puff* will not let you change these identifying letters. For each part, you need to specify the part name, the impedance characteristics, and the length, in that order, and on one line. *Puff* recognizes a part name by the first letter in the name; the rest of the letters are for you, so that you can read the parts list easily. Commas and periods are treated as decimal points. *Puff* generally ignores spaces, and does not pay any attention to whether a letter is upper or lower case. There are four different kinds of parts:

`tline` (pronounce this *tee-line*). This is a transmission-line section, and it is shown as a brown rectangle on an *EGA* display. *Puff* recognizes four units for specifying the characteristic impedance: Ω (typed as *Alt* `o`), `s` (for Siemens), `z` (normalized impedance), or `y` (normalized admittance). For example, a transmission line section with a characteristic impedance of 25 Ω can be specified as either 25Ω or by `0.04s`. The characteristic impedance must be positive. The normalizing impedance is given in the setup file. In `setup.puf`, it is 50 Ω. The length units are ° (degrees at the design frequency, typed as *Alt* `d`) and `mm` (millimeters). A quarter-wavelength line at the design frequency `fd` would be specified by writing 90°. It is usually more convenient to specify the transmission-line length in degrees rather than millimeters, but sometimes the physical units are useful for aligning a `tline` with a transistor or lumped element. In a `.puf` file, Ω is written as `O` and ° as `D`.

`lumped` These are lumped elements that you would solder into the circuit like resistors, capacitors, or diodes. A `lumped` part is drawn as a hollow blue rectangle on an *EGA* display. A real impedance may be specified in the same way as the characteristic impedance of a `tline`. Unlike the characteristic impedance for a transmission line, it may be zero or negative. For example, you may specify a 100-Ω resistor as 100Ω, `0.01s`, 2z, or `0.5y`, assuming that the normalizing impedance is 50 Ω. In addition, you can specify a reactive `lumped` part if you put a `j` either before or after the number, and use impedance units. For example, `50jΩ` and `jz` specify a reactance of 50 Ω at the design frequency `fd`. *Puff* scales reactances with frequency. Positive reactances are proportional and negative reactances inversely proportional to frequency. This means that an inductive 50-Ω reactance becomes 100 Ω at twice the design frequency, but that a capacitive 50-Ω reactance becomes -25j Ω at `2fd`. You can also specify a series circuit by combining a real part, positive imaginary part, and a negative imaginary part, all in the same `lumped` part. For example, `1+j10-j10z` specifies a circuit that is resonant at the design frequency with a Q of 10. The resistance is the normalizing impedance, and the inductive and capacitive reactance are ten times the normalizing impedance at the design frequency. Note that the units only appear once, after all the numbers. You should separate the numbers by a $+$ or $-$ sign, because *Puff* will ignore the spaces. Specifications with admittance units are treated in a dual way. Positive capacitive susceptances are proportional to frequency, and negative inductive susceptances inversely proportional to frequency, and you can specify simple parallel GCL circuits by combining a real part, a positive imaginary part, and a negative imaginary part. The length of `lumped` part is specified in `mm` only.

`clines` (pronounce this *see-line*). This is a pair of coupled transmission lines. *Puff* uses the same impedance and length units for `clines` as for a `tline`, except that either one or two impedances may be specified. It assumes that the even and odd-mode phase velocities on coupled lines are the same. If only one impedance is specified, then the specification looks the same as for a `tline`. If the given impedance is larger than the normalizing impedance, *Puff* interprets it as the even-mode impedance. If it is smaller than the normalizing impedance, *Puff* will take it as the odd-mode impedance. If the impedance is equal to the normalizing impedance, *Puff* will give you an error message. *Puff* will choose the remaining mode impedance to match the lines to the normalizing impedance. If you give two impedances, *Puff* will interpret the bigger one as the even-mode impedance, and the smaller one as the odd-mode impedance.

device A two-port whose scattering coefficients are specified by an s-parameter file. These would be used for transistors that would be soldered into the circuit. A `device` is drawn as an arrowhead, wide end first. The wide end is port 1, and the narrow end is port 2. A file name specifying the s parameters must be given. *Puff* will assume an extension of `.puf` if one is not given. The length of `lumped` part is specified in `mm` only. If you want to make up your own `device` file, you should look at `fsc10.puf` file given on the disk for the Fujitsu $fsc10$ transistor to see the file format.

When you try to leave the *Parts* window, *Puff* will check to make sure that all your parts are properly specified and that the `device` files can be read. *Puff* will give you an error message if a `tline` or `clines` has a negative impedance, or if a part too wide or too long to fit on the circuit board. *Puff* will also want the lengths and widths to bigger than a parameter called the `resolution`, which is specified in the setup file. In `setup.puf`, the `resolution` is 0.2 mm. It will redraw the current circuit if it has changed, and report any parts that now stick off the board.

4.1.2 The circuit window

The *Circuit* window is in the upper left corner of the screen. The square represents the microstrip substrate, and the numbers on the sides show where the connectors are. When you push a cursor key, you draw a part from the *Parts* window in the direction of the arrow. The message window will show you the change in the x and y coordinates. *Puff* starts out drawing part `a`, but you can get the other ones in the list by typing the letter for the part you want. You can ground the circuit at any point by pushing the = key. If there is already a part in the direction of the cursor key, *Puff* will move to the other end of the part rather than draw over it. If one of your parts ends within a distance of `resolution` of the end of another part, *Puff* will connect them together. *Puff* will stop you from drawing a part off the edge of the substrate, but it will not stop you from crossing over a previous part. You can make a path to a connector by pushing one of the number keys 1, 2, 3, or 4 on the top row of the keyboard. Notice that *Puff* does this by moving up or down first and then right or left. The electrical length of the connector paths is not taken into account in the analysis, and they are drawn with a different color or fill pattern to show this. From the point of view of the analysis, it is as there is a terminating resistor right across where the connector paths join the rest of the circuit.

The shift keys ⇑ are not shown on the circuit command list, but they are important in erasing and moving around the board. *Puff* will erase a part rather than move over it if you hold the shift key down when you push the cursor key. In the same way, the path to a connector will be erased if you hold the shift key down when you push the connector number. The shift key also is used to move the cursor when no parts are present. If there is no part already in place and you hold the shift key down while you push the cursor, then you will move half the length of the part, but no part is drawn. *Puff* moves in half steps rather than full steps because it is sometimes convenient for positioning the circuit in a symmetric way on the board. To move the full length of the part, you will need to move twice. In addition, if you are not at the connecting path to a port, holding the shift key down when you press a connector number moves you to that port without drawing the path. *Ctrl*-n moves the cursor to the nearest node of the network. This can be useful if you are off the network and want to get back, or if you want to see if two nodes are connected. If the situation is completely hopeless you can start over by pushing *Ctrl*-e.

The rules for `clines` are a little complicated and you will need to experiment with them. You can move around on `clines` with the cursor keys and you can use the *Ctrl*-n key to jump from line to line. You can connect `clines` to `clines`. If you draw the second `clines` in the same direction as the first, the lines will be connected to the lines in the other pair. This is the usual configuration for directional couplers. If you push the cursor at right angles to the direction of the first `clines` the two `clines` will staggered so that only one of the lines in each pair is connected. This is the usual arrangement for coupled-line filters.

Puff will beep if you type an invalid key. Next *Puff* will check to see that it can actually carry out the command. If it cannot, it will give you an error message in the red message box. For example, if you push the up arrow key in the present circuit, *Puff* will say,

`Cannot go over path to port`

because *Puff* cannot figure out why you would want to go the first connector if it is already in the circuit.

4.1.3 The Plot window

The *Plot* window is reached by typing *F2*. Analysis of the network is performed in the *Circuit* window by typing p. The connector paths are not considered in the analysis. If you type *Ctrl*-p, the old plots on the screen will be plotted again as dashed lines before making a new plot. This allows you to compare the scattering coefficients of different networks. The *Plot* window gives the values of the scattering coefficients at the design frequency `fd`. If you push the *PgUp* and *PgDn* keys, you can see the scattering coefficients at the other frequencies. You can choose which scattering coefficients are plotted and change the plotting parameters just by typing in new values. The ↑ and ↓ cursor keys can be used to move the cursor to different parameters. They cycle through a loop that includes the parameters in the *Plot* window and the x and y axes on the rectangular plot. Then just type over the parameter until you have it the way you want it. You can plot up to four different scattering coefficients at the same time. Just move the cursor down to the bottom of the *Plot* window, and the other marks will appear, together with a letter `s`. If you type in the ports for the s parameters, they will be plotted too. If you leave them blank, the line will be erased when you move the cursor up to a different line.

When you make a plot, *Puff* will do a spline interpolation between the calculated points. If the spline curves start to kink, it is an indication that the frequency interval between points Δ`f` has gotten too large. Δ`f` must also be small enough to catch any narrow resonances. You adjust the frequency plotting interval by specifying the ratio `fd/`Δ`f`. This has to be a positive integer. This rather indirect way of specifying the plotting interval is used because it makes it easy to use the points in calculating the time-domain response.

After an analysis, you can make a time-domain plot by typing `t`. *Puff* will make a fast Fourier transform of the scattering coefficients [4], and plot them on a linear scale with the same maximum amplitude as the Smith chart. The ratio `fd/`Δ`f` determines the time axis for the time-domain plot, which goes from $-1/(10\Delta f)$ to $4/(10\Delta f)$. You can adjust this interval by changing the ratio `fd/`Δ`f`. You can also adjust the number of points that are plotted in the setup file. The lowest and highest frequencies on the frequency axis of the magnitude plot are used for windowing the data for the Fourier transform. A raised-cosine window is used that goes to zero at the highest frequency. In addition, the scattering coefficients are assumed to be zero at frequencies lower than the lowest frequency.

Ctrl-`a` makes the photographic artwork, magnified by a photographic reduction factor specified in the setup file, which starts at 5 to 1. You will be prompted for your name and a network name. Only the `tlines` and `clines` will appear in the artwork and the corners will be mitred.

You can save a network in a file with *Ctrl*-`s`. The *Parts* window and the *Plot* window will be saved along with the *Circuit* window. *Puff* will add the extension `.puf` to the file name if one is not specified.

4.1.4 Initial Puff setup

This is a listing of the setup file, **setup.puf** for reference. Be careful if you change this file. You will need to edit the **setup.puf** file if you want to start with different parameters. You may add comments, in braces, at the end of the lines.

```
\b{oard}  {setup file for puff version e.2}
e  10.2   {dielectric constant of substrate}
t  1.27   {substrate thickness in mm}
s  25.4   {duroid side length in mm}
p  5.0    {photographic reduction ratio. s*p <=203.2mm}
c  19     {connector separation in mm*}
r  0.2    {circuit resolution in mm}
z  50     {normalizing impedance in Ohms. z<5000/sqrt((e+1)/2)}
n  128    {points in fast Fourier transform, power of 2,n<=256}
d  0      {display:0 Puff chooses, 1 EGA, 2 CGA, 3 One color}
f  5      {design_freq}
a  0      {artwork_correction in mm}
\k{ey for plot window}
d>   0    {dB max}
d< -20    {dB min}
f<   0    {frequency min}
f>  10    {frequency max}
fd/df 10
Smith radius 1
S  11
\p{arts window} {O=ohms and D=degrees}
lumped 150O 4mm
tline 50O 90D
tline 50O 60D
tline 50O 130D
device fsc10 2mm
clines 60O 90D
\e{nd}
*{There are two connectors on the left side, and two
connectors on the right, placed symmetrically about
the centerline.  If the connector separation is 0 then
Puff will just make two connectors, one is centered
on the left, and the second is centered on the right.}
```

4.2 Design Examples

Figure 5 shows a variety of microstrip circuits that were designed with *Puff*. *Puff* can generate mask artwork on a standard IBM compatible dot-matrix graphics printer. Two passes are made on each line to achieve a resolution of 120 to 150 dots per inch. A 12-cm square mask can be generated in approximately one minute on an IBM proprinter. Bends are chamfered to minimize the reflection discontinuities [5,6]. At the narrowest point the chamfer is 0.57 times the width of the incoming lines. This mask is reduced 5:1 onto a 2.5-inch-square glass plate using a Pentax 6×7 camera. The mask pattern is transferred using a UV lamp to a positive photoresist film spun onto a *Duroid* substrate. After the resist is developed, the copper is etched with a ferric-chloride solution. The circuit board is then clamped into a brass mounting block and the *s*-parameters are accurately measured with an HP 8410 network analyzer controlled by an IBM PC [7].

4.2.1 Branch-line coupler

A *Puff* screen dump for a branch-line coupler is displayed in Fig. 6. *Puff*'s predictions are compared with measurements for the branch-line coupler in Fig. 7. The artwork mask has been compensated to take into account the parasitic discontinuities at the four tee's. The dominant effect is a phase shift in the stem of the tee. This shift can be accounted for, to first order, by changing the length of the lines in the artwork. In *Puff* this extra length is placed in parentheses in the parts list, next to the transmission-line section involved. The correction is then applied to the artwork mask but is ignored in the analysis. Semi-empirical expressions for the electrical shift have been published by Hammerstad and Bekkadal [8] and provide results in good agreement with measurements.

4.2.2 Low-pass and bandpass filter examples

Figure 8a shows results for a low-pass filter design using alternating sections of high and low impedance line. The widths and lengths were varied interactively until

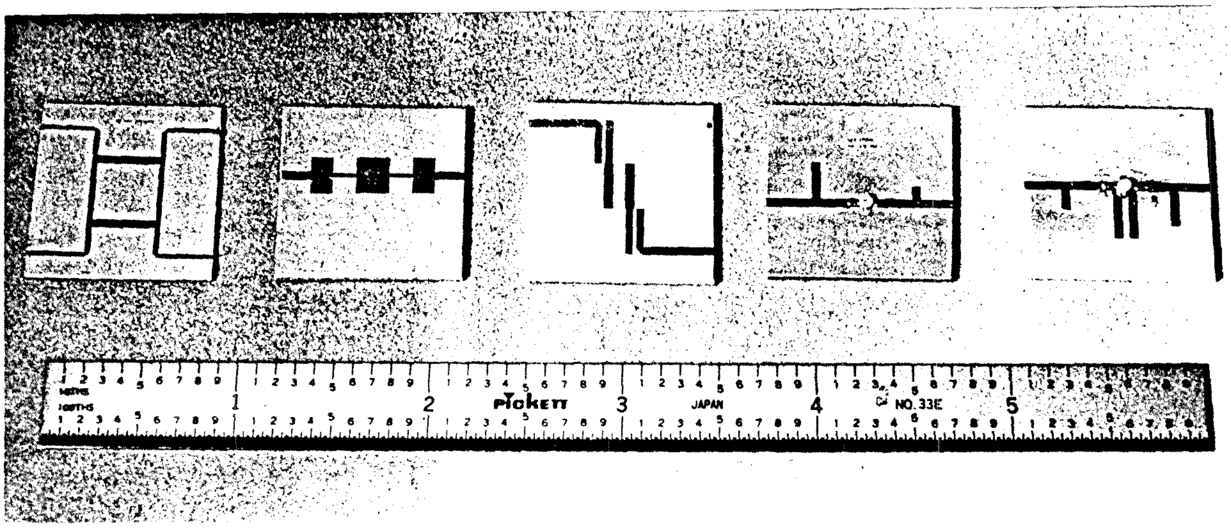


Figure 5. A collection of microwave circuits designed by *Puff* and fabricated on either 0.635-mm (25-mil) or 1.27-mm (50-mil), 1-ounce *Duroid* (6010.2). Pictured from left to right are a branch-line coupler, low-pass filter, bandpass filter, FET amplifier and oscillator. These circuits were designed to operate at frequencies ranging from 2 to 7 GHz.

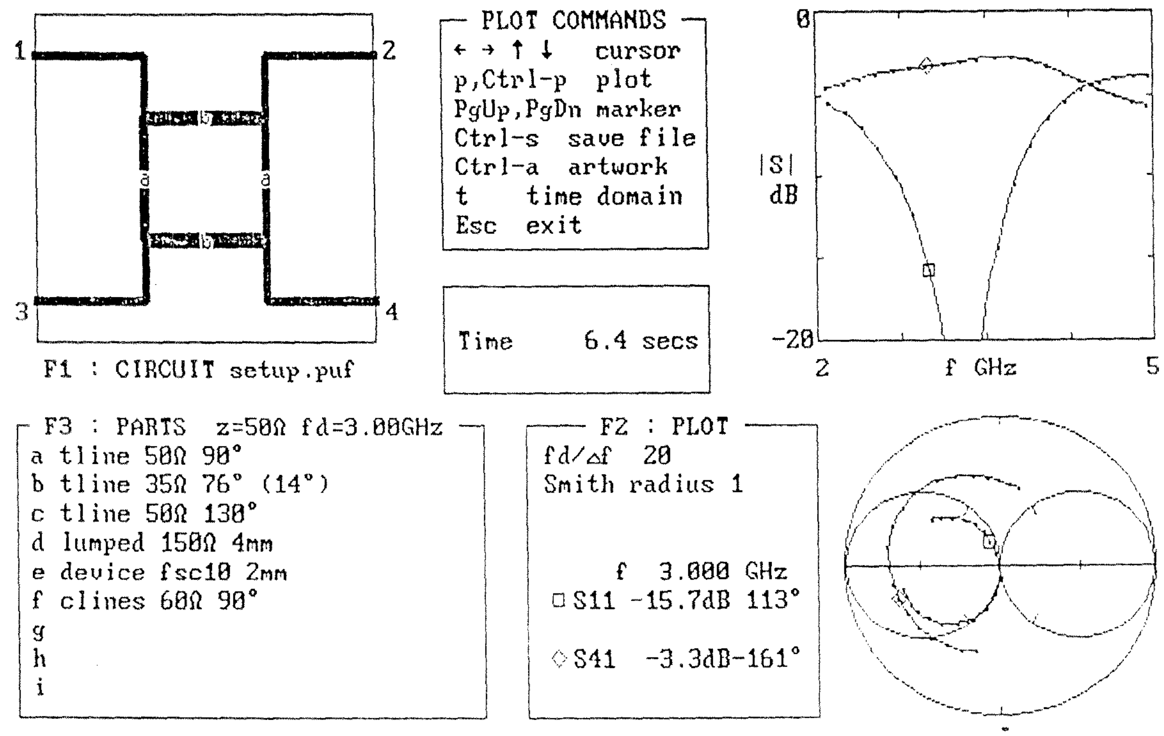


Figure 6. A *Puff* screen dump obtained from an IBM personal computer equipped with an Enhanced Graphics Display. In the top left corner is a layout of a branch-line coupler. Below this is a list of parts. The rectangular plot and Smith chart display s_{11} and s_{41}. The small box in the center shows the time required for computing the plots on the right.

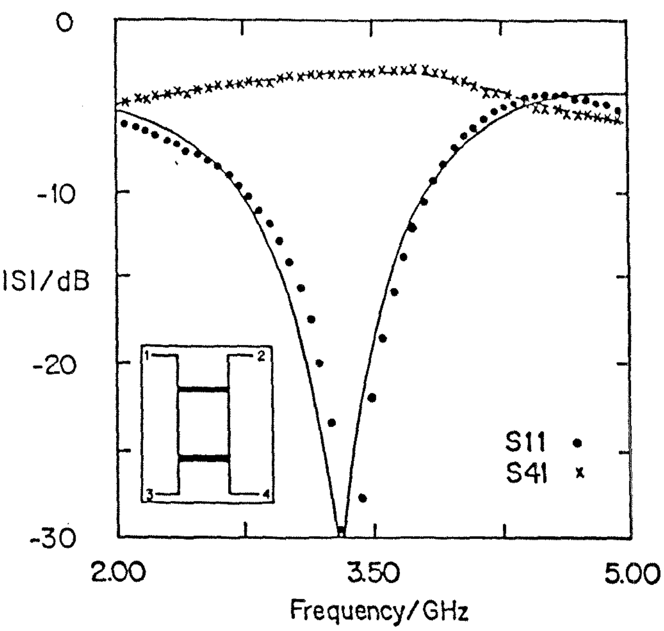


Figure 7. Comparison between predictions made by *Puff* (solid and dashed lines) and measurements (• and ×) made with an IBM PC controlled HP 8410 network analyzer for the branch-line coupler design in Fig. 6.

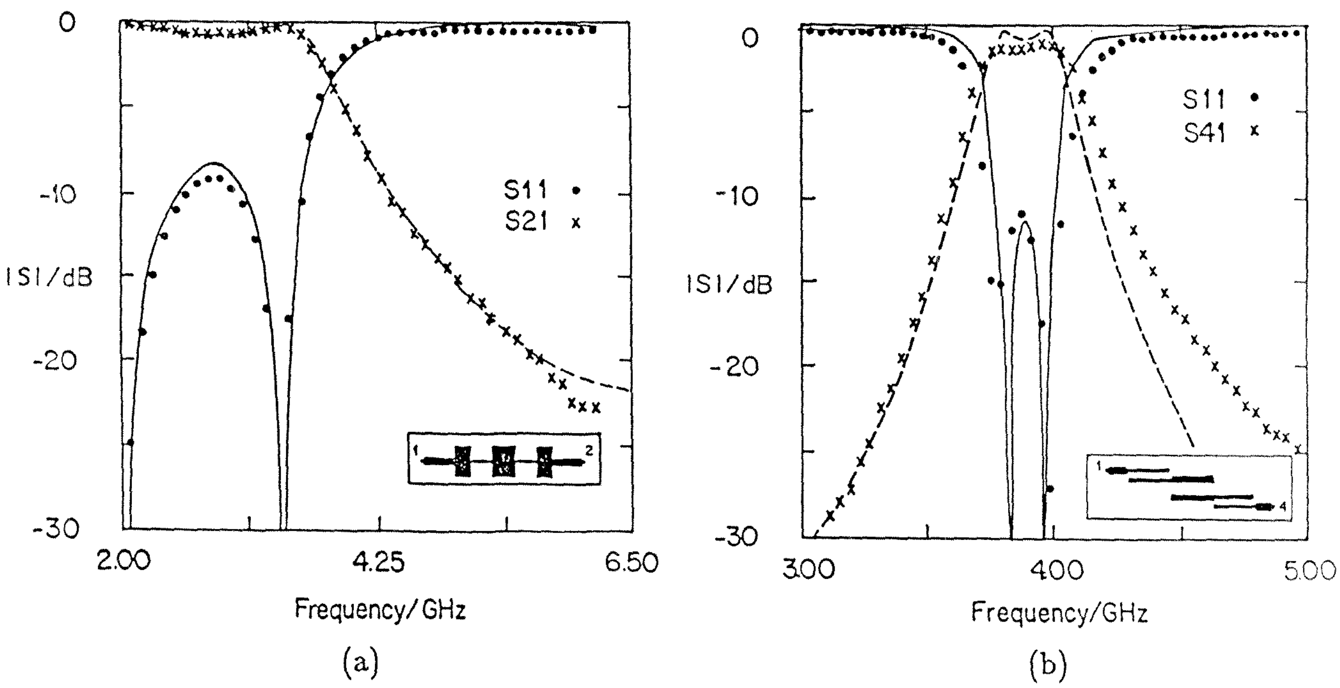


Figure 8. Reflection and transmission results for (a) low-pass and (b) bandpass filter showing theory curves and measurements (• and ×).

the desired filtering performance is obtained on the screen. The steepness of the cutoff can be adjusted by varying the ratio of high to low impedances. The value of the higher characteristic impedance line is limited by the minimum line width that can be accurately etched. More stages may be added to obtain a sharper filtering characteristic. The fringing fields on the low-impedance lines are compensated using the Hammerstad formula [8]. The second design in Fig. 8 is for a bandpass filter using three sections of quarter-wave coupled line. The line lengths are again compensated for the fringing capacitances.

4.2.3 FET amplifier and oscillator design

Figure 9a shows how *Puff* can be used to design a single-stage low-noise amplifier with a Fujitsu FSC10 MESFET. *Puff*'s calculations were performed using s-parameters obtained from the Fujitsu data sheet. The gain of the amplifier was measured using an HP 8970A noise-figure meter, and the transistor bias was applied using two external bias tee's. A maximum gain of 13.2 dB was obtained at 3.7 GHz with a noise figure of 1.9 dB.

Puff has also been used to build feedback oscillators. The approach is different from traditional oscillator design. In *Puff* it is natural to analyze the circuit looking in from the external ports, whereas a conventional design looks at conditions inside the circuit. The oscillation condition is $s_{11} = 1/s_L$, where s_{11} is the reflection coefficient at the input of the transistor and s_L is the reflection coefficient of the load. Fig. 10a shows how this condition can be achieved. In the figure, the s_{11} curve loops around the point $1/s_L$ (Fig. 10a). It is assumed that the s_{11} curve circles clockwise as the frequency increases and that as the transistor saturates, the loop contracts. Oscillations build up in the circuit, the transistor saturates, and the s_{11} loop shrinks until the oscillation condition is met.

However, an interesting problem arises when we try to follow this procedure in *Puff*. The load is the matched port, so that $1/s_L = \infty$. It is not clear how we go about drawing a clockwise loop about the point at infinity. The solution

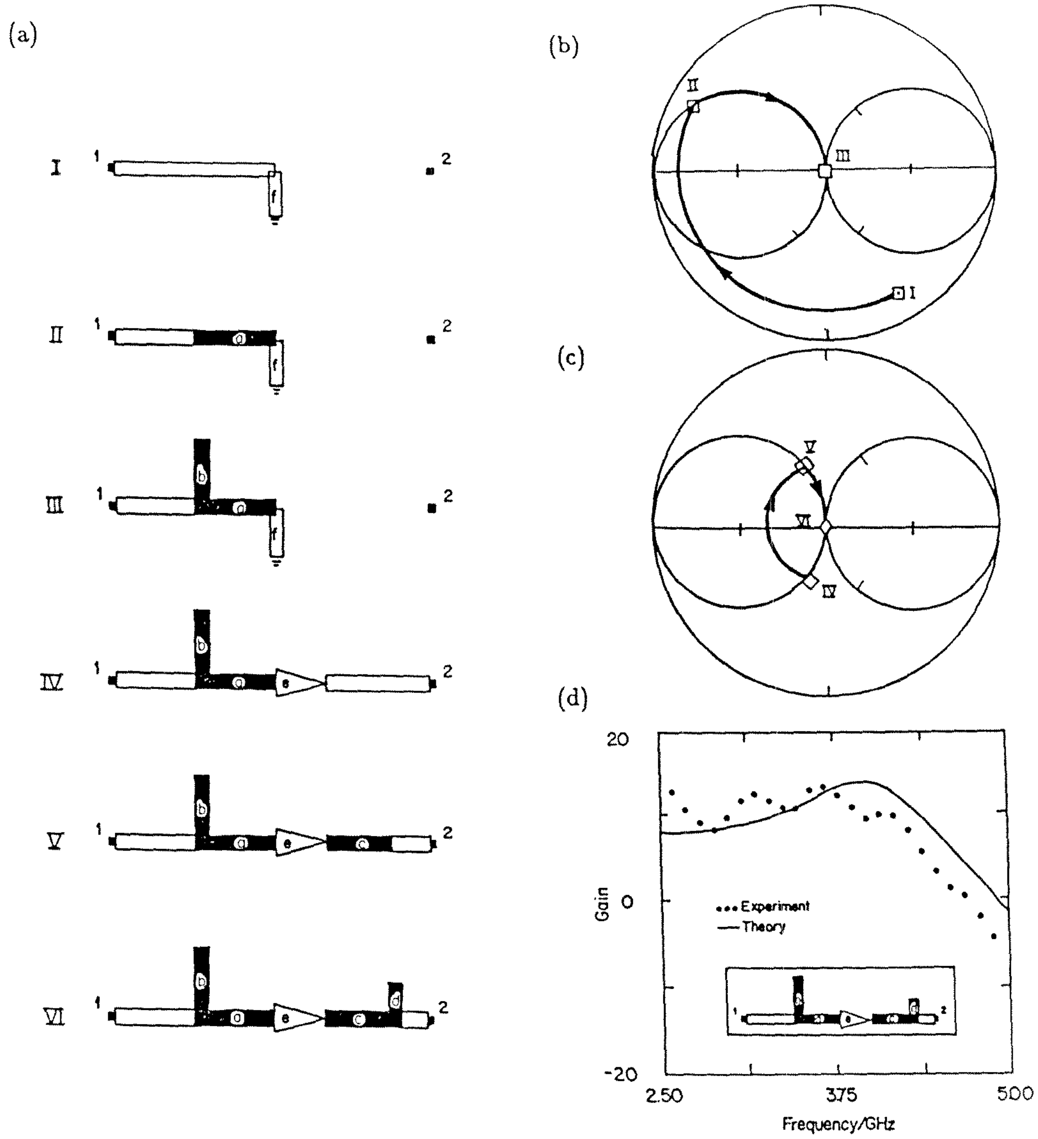


Figure 9. (a) Steps for designing a low-noise FET amplifier. (I) The input stage is designed to have the impedance with the optimum noise figure. Beginning with a lumped impedance (part **f**) which is the conjugate of the optimum source impedance given in the manufacturer's data sheet, (II) line **a** is chosen to move s_{11} around to the g=1 circle and (III) stub **b** brings this into the center of the Smith chart. (IV) An output stub is designed in a similar manner to provide a conjugate match of s_{22}. (V) Line **c** combined with (VI) stub **d** gives the required output match, (b) and (c) show the corresponding Smith charts for the input and output matching steps. (d) Comparison of *Puff*'s gain predictions with measurements made with an HP noise-figure meter.

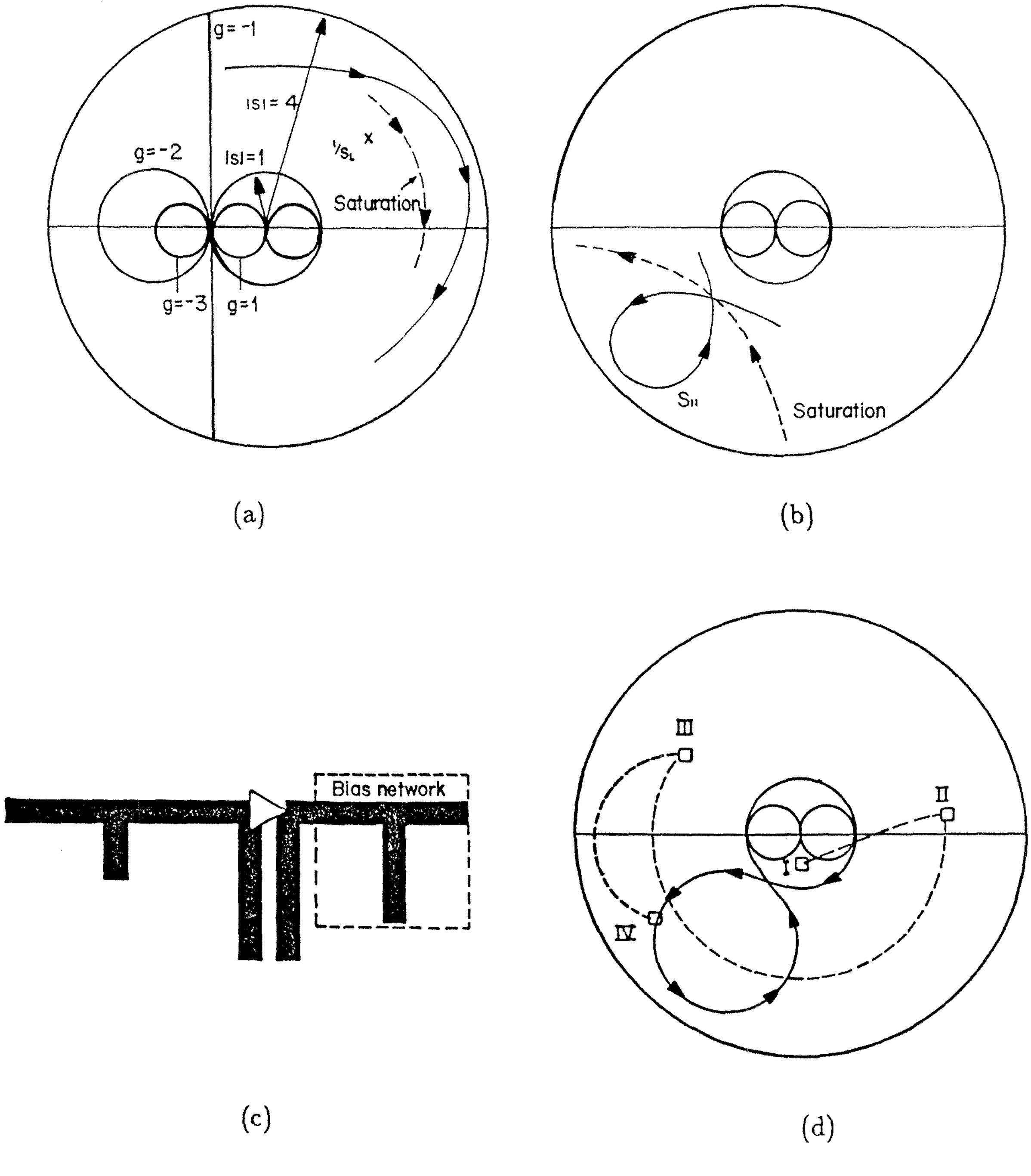


Figure 10. Expanded Smith chart used for the oscillator design. (a) A clockwise loop enclosing the point $1/s_L$ provides oscillations. (b) A counterclockwise loop for the point $1/s_L = \infty$. (c) Layout for a microstrip FET oscillator. (d) The coupled-line feedback loop is varied to get a large s_{11} (I) → (II) and a stub is positioned to produce a counterclockwise loop (III) → (IV). The resulting circuit generated 10 mW at 5.6 GHz.

is to add circuit elements that produce a counterclockwise loop. Because of the properties of bilinear transforms [9], a region *inside* the previous clockwise loop is mapped to the *outside* of the counterclockwise loop. This means that when the transistor saturates the loop expands until it intersects the point at infinity, and the oscillation condition is satisfied (Fig. 10b,c). It is easy to demonstrate this effect by reducing the magnitude of the transistor gain in steps to simulate saturation. This design procedure is shown graphically in Fig. 10d and has been used to build several oscillators.

4.3 Analysis Method

A *Puff* circuit is built up from the following set of parts: microstrip transmission lines, coupled lines, lumped elements, and arbitrary two-port devices specified by s-parameter files, plus tees, crosses, open circuits, and grounds that result from the way these parts are connected. The microstrip dimensions are derived from closed-form expressions given by Schneider [10]. Width corrections due to the metal thickness [11] and systematic errors in the photography and etching can be taken into account in the artwork by a parameter in a setup file. Dispersion and losses are not taken into account. Spacings and widths for the coupled microstrip lines are determined by solving a set of coupled equations derived by Akhtarzad [6,12]. The even and odd electrical lengths are assumed to be equal and are derived by calculating an equivalent guide wavelength for a single line.

The analysis algorithm combines the behavior of these individual parts to determine the overall circuit performance. The method must be a general technique that is efficient for a wide range of circuit configurations. One possible approach is to use impedance or admittance matrices to calculate circuit behavior. A problem with this method is that when microwave networks are measured, they are characterized by scattering parameters. Analyzing microwave circuits in terms of an admittance matrix requires converting the scattering parameters for the individual components to admittance matrices, then combining these to find the admittance matrix of the entire circuit, and then converting back to s-parameters. Usually these conversions take so much time that it is better to work with scattering coefficients throughout.

There are several different approaches for analyzing microwave circuits using scattering parameters. These methods usually involve filling a matrix with the s-parameters of the individual components [13–15]. The matrix tends to be sparse because scattering coefficients relating different subnetworks are zero. This s-matrix is combined with a second matrix which describes the way in which the components are connected. These methods involve the manipulation of big matrices occupying large amounts of memory. For example, a simple circuit like the branch-line coupler requires the solution of 20 linear equations with complex coefficients. Two thirds of the elements in the matrix are zero so that in a Gaussian elimination most of the time is spent multiplying, adding or storing zero. This computational inefficiency can be avoided by combining the individual subnetworks two at a time, and calculating the s-parameters each time a connection is made, until the complete network is assembled. This technique, called the *subnetwork growth* method [13,15], has the advantage that no large sparse matrices are created.

In making a connection there are two distinct operations depending on whether the ports to be joined are on different networks or on the same network (Fig. 11). Formulas for the new s-parameters can be calculated using signal flow-graphs.

Fig. 12a shows the signal flow graphs when the two ports being connected are from different networks. The original pair of networks S and T are combined to form a new network S'. The ports to be joined are k and l. Let the input port be j, and consider an output port i, in S, and another output port m, in T. Mason's rule [17] may be used to write down two formulas for the scattering coefficients:

$$s'_{ij} = s_{ij} + \frac{s_{kj} t_{ll} s_{ik}}{1 - s_{kk} t_{ll}} \tag{1}$$

and

$$s'_{mj} = \frac{s_{kj} t_{ml}}{1 - s_{kk} t_{ll}}. \tag{2}$$

The signal-flow graph for the case when the two ports belong to the same network is shown in Fig. 12b. Mason's rule has to be applied carefully, because there are

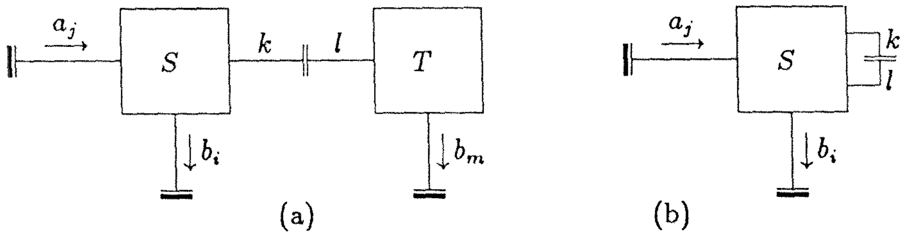


Figure 11. The two kinds of joints. A joint between ports k and l on (a) two different networks , and (b) on the same network.

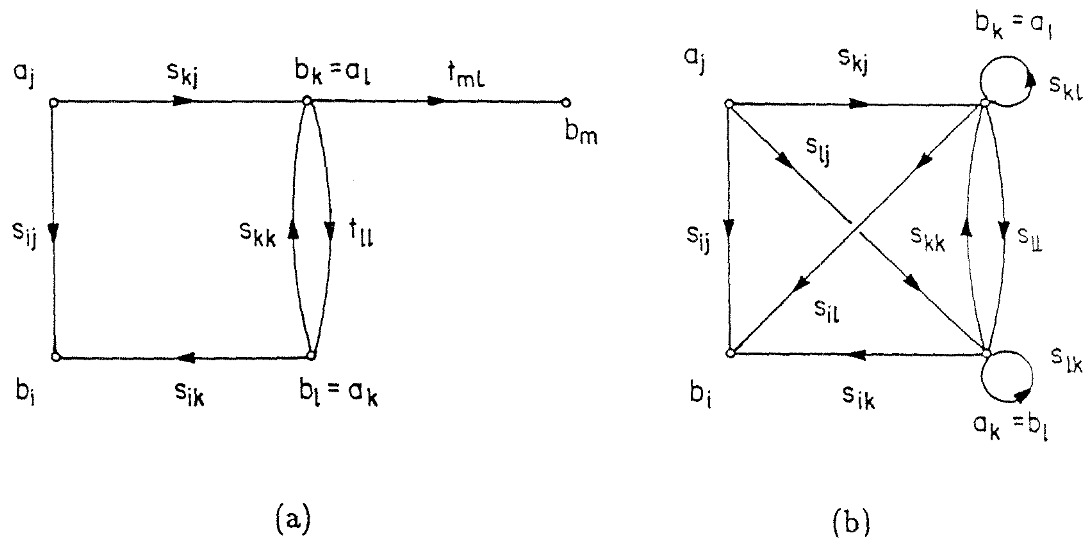


Figure 12. The signal flow graphs for (a) joining ports k and l on different networks S and T, and (b) on the same network.

several loops that must be accounted for. The formula for the new s-parameters becomes

$$s'_{ij} = s_{ij} + \frac{s_{kj}s_{il}(1 - s_{lk}) + s_{lj}s_{ik}(1 - s_{kl}) + s_{kj}s_{ll}s_{ik} + s_{lj}s_{kk}s_{il}}{(1 - s_{kl})(1 - s_{lk}) - s_{kk}s_{ll}}. \tag{3}$$

Equations (1) and (2) may also be derived as special cases of Equation (3).

The execution time of this algorithm is sensitive to the order in which joins are made. It is difficult to derive a general method for prescribing the connection order that will minimize the number of calculations. *Puff* uses a simple approach which searches for a connection that will result in a new subnetwork with the smallest number of new s-parameters to recalculate. This method is close to the optimal, and is easy to implement with a minimum of computational overhead [15]. Programming in Pascal allows the circuit description in *Puff* to be stored as dynamic variables that are addressed using pointers. The process of laying-out a network consists of building up a linked list of subnetworks and connections. In the analysis stage *Puff* collapses these linked lists by making connections until no joints remain. The s-parameter manipulation involves complex additions and multiplications that are performed by two short assembly-language routines which exploit the power of the 8087 math-coprocessor.

There are two classes of singularities to be avoided when calculating the s-parameters. One type occurs when the s-parameters of the parts in the circuit become infinite. For example, the reflection coefficient of a resistor with a resistance equal to the negative of the reference impedance is infinite when connected to ground. To avoid this problem *Puff* multiplies all negative resistances by $1 - 1.235 \times 10^{-12}$. This is equivalent to adding a tiny resistor. The second class of singularities occurs when the denominator and numerator in one of Eqns. (1–3) becomes zero but the s-parameters are well defined. In these cases a simplified expression for the new s-parameters can be derived. However, testing each s-parameter operation for these special cases would considerably slow down the execution and complicate the programing. These singular cases appear as a resonant loop current when a tee, cross, short or open are present. One way to avoid these problems is to add tiny attenuators on the ports of these components. This is implemented in *Puff* by multiplying the s-parameters of these parts by the same factor quoted above for negative resistances.

To illustrate how the subnetwork method proceeds in practice, consider the branch-line coupler shown in Fig. 13a. *Puff* subdivides the branch-line coupler into a network of eight parts: two quarter-wave 50-Ω transmission lines, two quarter-wave 35-Ω transmission lines, and four tees. These components are labeled A through H in the figure, and the external ports are numbered from 1 to 4. The branch-line coupler is redrawn in block form in Fig. 13b. Once *Puff* calculates the s-parameters for each of the circuit elements, parts are combined in pairs to make the four new subnetworks shown in Fig. 13c. The four new subnetworks are all three-ports. These are then combined in pairs to form two four-ports, $ABCD$ and $EFGH$, as shown in Fig. 13d. Finally these two four-ports are joined to make the branch-line coupler. This is slightly more complicated than the previous connections, because these two four-ports are connected at two ports rather than one. *Puff* makes the connection in two steps. First it joins one of the pairs of ports. This leaves a six-port shown in Fig. 13e. Four of the ports are the numbered external ports for the branch-line coupler, but the remaining two are internal ports that need to be connected. So as a final step, *Puff* connects these two internal ports, and the analysis is complete. Unnecessary calculations are avoided by only calculating the parameters related to the scattering coefficients selected by the user.

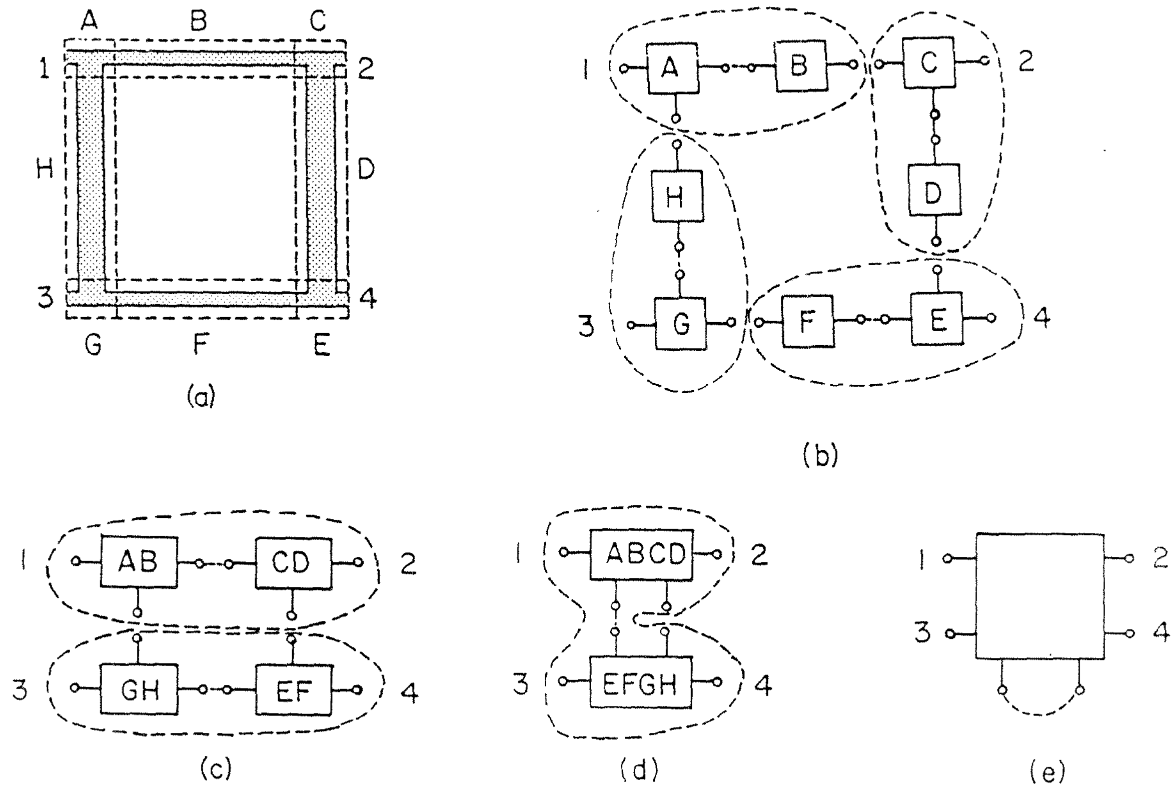


Figure 13. Analysis of a branch-line coupler. (a) Outline of the coupler, showing the different parts. (b) Representing the eight unconnected parts by boxes. (c) Joining the eight parts by pairs to make four three-ports, and (d) joining these to make two four-ports. Then one connection is made to form a single six-port. (e) Finally the two internal ports are connected.

References

[1] *Intel Microsystem Handbook*, 1986 (8086/8087 8MHz clock).

[2] R. Compton, W. Williams, S. Pietrusiak, Z. Popović and D. Rutledge, "A class for computer-aided design and measurement of microwave integrated circuits," *Proceedings of the Conference of Producibility of Millimeter/Microwave Integrated Circuits*, Huntsville, Alabama 1986.

[3] M. A. Covington, "Smooth curves," *PC Tech Journal*, pp. 110–120, Aug. 1986.

[4] W. H. Press, B. P. Flannery, S. A. Teukolsky and W. T. Vetterline, *Numerical Recipes*, Cambridge University Press, New York, 1986.

[5] B. Easter, A. Gopinath, I. M. Stephenson, "Theoretical and experimental methods for evaluating discontinuities in microstrip," *The Radio and Electronic Engineer*, **vol. 48**, pp. 73–84, Jan/Feb. 1978.

[6] T. C. Edwards, *Foundations for Microstrip Circuit Design*, John Wiley & Sons, Avon, Great Britain, 1985.

[7] W. L. Williams, R. C. Compton and D. B. Rutledge, "ELF: computer automation and error correction for a microwave network analyzer," submitted to *IEEE Trans. Instrum. Meas.*

[8] E. O. Hammerstad, and F. Bekkadal, "A Microstrip Handbook," *ELAB Report*, STF 44 A74169, N7034, University of Trondheim, Norway, 1975.

[9] K. Kurokawa, *An Introduction to the Theory of Microwave Circuits*, Academic Press, New York, pp. 26–27, 1969.

[10] M. V. Schneider, "Microstrip lines for microwave integrated circuits," *Bell System Technical Journal*, **vol. 48**, pp. 1421–1443, May/June 1969.

[11] I. J. Bahl, and R. Garg, "Simple and accurate formulas for a microstrip with finite strip thickness," *Proceedings of the IEEE*, **vol. 65**, pp. 1611–1612, Nov. 1977.

[12] S. Akhtarzad, T. R. Rowbotham, and P. B. Johns, "The design of coupled microstrip lines," *IEEE Trans. Microwave Theory Tech.*, **vol. MTT-23**, pp. 486–

492, June 1975.

[13] K. C. Gupta, R. Garg, and R. Chadha, *Computer-aided Design of Microwave Integrated Circuits*, Artech House, Dedham, Massachusetts, 1981.

[14] V. Monaco and P. Tiberio, "Computer-aided analysis of microwave circuits," *IEEE Trans. Microwave Theory Tech.*, **vol. MTT-22**, pp. 249–263, March 1974.

[15] V. Monaco and P. Tiberio, "Automatic scattering matrix computation of microwave circuits," *Alta. Freq.*, **vol. 29**, pp. 165–170, Feb. 1970.

[16] G. Filipsson, "A new general computer algorithm for s-matrix calculation of interconnected multiports," *Proc. 11th Euro. Microwave Conf.*, pp. 700–704, 1981.

[17] S. J. Mason, "Feedback theory—further properties of signal flow graphs," *Proceedings of the IRE*, **vol. 44**, pp. 920–926, July 1956.

[18] L. W. Johnson and R. D. Riess, *Numerical Analysis*, Addison Wesley, Reading, Massachusetts, pp. 237–247, 1982.

[19] R. C. Compton, W. L. Williams, Kent Potter and D. B. Rutledge, "Puff: Microwave computer aided design for education using personal computers," submitted to *IEEE Trans. Microwave Theory Tech.*

[20] R. C. Compton, W. L. Williams, and D. B. Rutledge, "Puff: an interactive microwave computer aided design program for personal computers," *1987 MTT-S International Symposium.*, Las Vegas, Nevada, June 1987.

[21] D. B. Rutledge, R. C. Compton, W. L. Williams, Z. Popović and Kent Potter, "A class for computer aided design and measurement of microstrip circuits," *1987 International Antennas and Propagation Symposium.*, Blacksburg, Virginia, June 1987.

[22] R. C. Compton, W. L. Williams, and D. B. Rutledge, "Computer aided design of microwave integrated circuits," *Proceedings of the 1986 Advanced Educational Projects Conference* San Diego, California, April 1986.

Chapter 5

Conclusions

An efficient antenna is a crucial component in a receiver system. A rigorous theoretical formulation for analyzing the bow-tie antenna on a dielectric substrate has been presented. The method may be generalized to planar antennas of other shapes. Numerical results from the theory agree well with 94-GHz pattern measurements and 2-GHz impedance measurements. These results show that for wide bows the dominant current is a wave propagating along the axis of the bow at the dielectric wave number. For increasing bow-tie length, the impedances spiral rapidly towards a quasi-static value given by transmission-line theory. As the bow narrows the dominant current becomes an edge current with the quasi-static wave number. Long, narrow bow-ties were found to behave like strip dipole antennas and could be modelled using a simple long-wire travelling wave current. Measurements of the linear array show that the neighbors have significant effects on the bow-tie pattern. The log-periodic antenna measurements display a centralized peak in the E and H planes at broadside. Consequently, the log-periodic antenna is more suitable than the bow-tie antenna for substrate-lens receiver systems and should result in improved coupling efficiencies. A detailed experimental and theoretical investigation of the impedance and pattern of the log-periodic antenna over a broad frequency range is necessary to assess the suitability of this antenna for specific applications.

Approximation techniques for calculating the effective impedance of a thin planar periodic meshes have been developed. This refined monomodal impedance technique is faster and easier to use than rigorous moment methods and more accurate and general than existing approximation methods. Also described were a number of ways of choosing nonconventional basis functions for the current on the metal plates or electric field in the aperture. These functions can be used to solve problems like the strip grating, square hole mesh and the circular hole mesh. They can also be extended to solve mesh problems like the cross in which the waveguide basis functions are not readily available. These techniques were expanded to analyze arrays with integrated devices of known impedance. Impedance and pattern meshes performed at 94 GHz and 60 GHz are in good agreement with predictions.

Software requirements in an educational context differ from those in industry. A program called *Puff* has been developed for use in a hands-on microwave integrated laboratory class at the California Institute of Technology and has been introduced into courses at the University of California, Los Angeles and Cornell University. *Puff* combines interactive graphics with an easy to use set of commands to provide a user-friendly interface for the computer aided design of microstrip circuits. Circuits are drawn using elements selected from a parts list and the analysis may be performed at any stage, directly from the screen layout. Camera-ready masks are conveniently generated by *Puff* using a standard IBM dot-matrix printer. With the inclusion of two simple length corrections for tee's and open circuits, good agreement between predictions and experiment for a variety of passive and active circuits have been obtained in the frequency range 2 to 7 GHz. Possible extensions to *Puff* include dispersion [1], discontinuity compensation [2] and automatic artwork corrections. *Puff* will be distributed on diskette as part of a microwave textbook.

References

[1] W. J.Getsinger, "Microstrip dispersion model," *IEEE Trans. Microwave Theory Tech., vol. MTT-21*, pp. 34–39, Jan. 1973.

[2] R. Chada, and K. C. Gupta, "Compensation of discontinuities in planar transmission lines," *IEEE Trans. Microwave Theory Tech., vol. MTT-30*, pp. 2151–2155, Dec. 1982.

Appendix A

Computer program listing of *Puff*

```
{.N PUFF1E3} {Modules used to extend beyond 64K code limit}
{.N PUFF2E3}
{$G512,D-} { Allow input redirection}
{$R+,C-}   {- for public}
{*******************************************************************
Puff Version E.3

 Compile using Borland 8087 Turbo Pascal 3.0 + TurboPower Turbo Extender
 Files needed :
 puffe3.pas, inc0e3.pas, inc1e3.pas, inc2e3.pas, inc3e3.pas, pufffasm.com
 To compile :
 c> shellgen -B -R puffe3
 c> bigmake puffe3
********************************************************************}

program PUFF;
const
 {The following constants are the keyboard return codes.
  See Turbo Manual Appendix K. For the extended codes 27 XX, key=#128+XX}
 not_esc=#0;     Ctrl_a=#1;     Ctrl_d=#4;     Ctrl_e=#5;    backspace=#8;
 Ctrl_n=#14;     Ctrl_p=#16;    Ctrl_r=#18;    Ctrl_s=#19;   Ctrl_u=#21;
 Esc=#27;        sh_1=#33;       sh_3=#35;      sh_4=#36;      sh_2=#64;
 Alt_o=#152;     Alt_d=#160;
 sh_down= #178;  sh_left=#180;  sh_right=#182; sh_up=#184;
 F1=#187;        F2=#188;       F3=#189;
 up_arrow=#200;  PgUp=#201;     left_arrow=#203; right_arrow=#205;
 down_arrow=#208;PgDn=#209;    Ins=#210;      Del=#211;

 {The following characters refer to extended graphics character set}
 {Lambda=#128;} Delta=#129; {Shift_arrow=#130;} bar=#131;
 ground=#132; infin=#133; ity=#134; Omega=#139; Degree=#140;
 lbrack=#123; rbrack=#125;

{GRAPHICS PARAMS}
 hir=0.57143; {20/35 for EGA to CGA conversion of y pixel dot postion}
 ymin = 3; ymax = 161;  yf=0.833; {ratio of y/x on Smith 5/6 for EGA monitor}
 xmin : array[1..3] of integer=(449, 12,462); {1 magn 2 board 3 smith}
 xmax : array[1..3] of integer=(636,202,636);{xmax[2]:=xmin[2]+(ymax-ymin)/yf}
 centery=270; centerx=549; {centerx=(xmax[3]+xmin[3])/2 center of Smith chart}
 rad=87; {rad=(xmax[3]-xmin[3])/2 radius of Smith chart}
 col_window :array[1..3] of integer=(Lightcyan,Lightgreen,Yellow);
 s_color    :array[1..4] of integer=(lightred,lightcyan,lightblue,yellow);
 xbn=2;       ybn=16; {parts list}      charx=8;    chary=14;
 max_net_size=6;    des_len=31; key_max=2000;
 max_ports=4;  max_params=4;
 red_psx=0.2117; {1000 mil*0.0254 mil /mm /(120 dots) in x dirn for artwork}
 red_psy=0.1764; {1000 mil*0.0254 mil /mm /(144 dots) in y dirn}

 one=0.999999999998765;
 {Attenuation factor for tee's, crosses, shorts, opens and negative resistors}
 ln10=2.302585;
 infty=1.0e100;
 step_fn=true; {selects step response or pulse for FFT}
```

```
{CGA and EGA graphics characters drawn on circuit board}
gchar :array[1..9] of array[1..7] of byte=(
{a} ($00,$1c,$02,$1e,$22,$22,$1e),
{b} ($20,$20,$2c,$32,$22,$32,$2c),
{c} ($00,$00,$1c,$22,$20,$22,$1c),
{d} ($02,$02,$1a,$26,$22,$26,$1a),
{e} ($00,$00,$1c,$22,$3e,$20,$1c),
{f} ($0c,$12,$10,$38,$10,$10,$10),
{g} ($00,$1e,$22,$22,$1e,$02,$1c),
{h} ($20,$20,$2c,$32,$22,$22,$22),
{i} ($00,$08,$00,$08,$08,$08,$08));
hchar :array[1..9] of array[1..9] of byte=(
{a} ($00,$7f,$63,$7d,$61,$5d,$5d,$61,$7f),
{b} ($70,$50,$5e,$53,$4d,$5d,$4d,$53,$7e),
{c} ($00,$00,$1c,$63,$5d,$5f,$5d,$63,$1c),
{d} ($07,$05,$3d,$65,$59,$5d,$59,$65,$3f),
{e} ($00,$00,$3c,$63,$5d,$41,$5f,$63,$3e),
{f} ($1c,$73,$6d,$6f,$44,$6c,$6c,$6c,$7c),
{g} ($00,$3f,$61,$5d,$5d,$61,$7d,$63,$3c),
{h} ($70,$50,$5c,$53,$4d,$5d,$5d,$55,$77),
{i} ($00,$1c,$14,$1c,$14,$14,$14,$14,$1c));

{Commands for help window}
 command:array[1..3,1..7,1..2] of string[17]=(
 (( #27' '#26' '#24' '#25,' draw part'),
  ('=',' ground '#132'      '),
  ('1..4',' connect path'),
  ('a..i',' select part'),
  ('Ctrl-e',' erase crct'),
  ('Ctrl-n',' go to node'),
  ('Esc',' exit        ')),
 (( #27' '#26' '#24' '#25,'   cursor '),
  ('p,Ctrl-p',' plot   '),
  ('PgUp,PgDn',' marker '),
  ('Ctrl-s',' save file'),
  ('Ctrl-a',' artwork  '),
  ('t','    time domain '),
  ('Esc',' exit        ')),
 (( #27' '#26' '#24' '#25,'   cursor '),
  ('Del,Backspace,Ins',''),
  ('Alt-o, Alt-d ',#139', '#140),
  ('Ctrl-u',' update    '),
  ('Ctrl-e',' erase crct'),
  ('Ctrl-r',' read file '),
  ('Esc',' exit        ')));

type
 textfile     = text[$800];
 line_string  = String[des_len];
 file_string  =  string[127];
 farray       = array[1..514,1..max_params] of real; {512+2 nft max= 256}
 char_s       = array[1..112] of byte;
```

```
complex      = ^complex_record;
s_param      = ^s_paramater_record;
plot_param   = ^plot_record;
spline_param = ^spline_record;
net          = ^net_record;
conn         = ^connector_record;
compt        = ^compt_record;
marker       = ^byte;

complex_record = record
  r,i : real;
end;

s_paramater_record = record
  z          : complex;
  next_s     : s_param;
end;

plot_record = record
  next_p,prev_p : plot_param;
  filled        : boolean;
  x,y           : real;
end;

spline_record = record
  next_c,prev_c : spline_param;
  sx,sy,h       : real;
end;

net_record = record
   com                     : compt;
   con_start               : conn;
   xr,yr                   : real; {postion in mm}
   node,chamfer,grounded   : boolean;
   next_net,other_net      : net;
   nx1,nx2,ny1,ny2,number_of_con,nodet,ports_connected : integer;
  end;

connector_record = record
  port_type,conn_no : integer; {0 norm 1... max_port external 5,6 internal}
  cxr,cyr           : real  ;{position in mm}
  dir               : byte;
  net               : net;
  next_con,mate     : conn;
  s_start           : s_param;
end;

compt_record = record
  lngth,width,zed,zedo,wavelength,con_space,spec_freq : real;
  xp,xmaxl,x_block,xorig,yp,number_of_con,used        : integer;
  s_begin,s_file,s_ifile,f_file                       : s_param;
  changed,right,parsed,step : boolean;
  next_compt,prev_compt     : compt;
```

```
  descript                 : line_string;
  typ                      : char;
end;

key_record = record
  keyl      : char;
  noden     : integer;
end;

registerset = record
  case integer of
    1: (AX,BX,CX,DX,BP,DI,SE,DS,ES,Flags: integer);
    2: (AL,AH,BL,BH,CL,CH,DL,DH: byte);
  end;

var
 col            : complex;                 {1+j0}
 data           : farray;                  {array for FFT}
 ticks          : integer absolute $0000:$046C;{Timer location Norton p.222}
 c_s            : s_param;
 result         : RegisterSet;             {record for Bios interupts}
 net_file       : textfile;
 conk,ccon      : conn;
 char_p         : ^char_s;
 sresln         : string[8];
 key_list       : array[1..key_max] of key_record;{List decribing circuit}
 board          : array[1..11] of boolean;       {used in reading board setup}
 s_key          : array[1..10] of line_string;   {plot window paramters}
 s_param_table  : array[1..max_params] of compt; {Which s-params to plot}
 bita           : array[0..960] of byte;         {Line bit map for artwrok}
 xvalo,yvalo    : array[1..4] of integer;        {used by plotting in rcplot}
 cross_dot      : array[1..35] of integer;       {Dot colors under cross}
 box_dot        : array[1..26,0..8] of integer;  {Dot colors under markers}
 box_filled     : array[1..8] of boolean;        {true if box_dot set}
 portnet        : array[0..max_ports] of net;    {Record of port}
 inp,out        : array[1..max_ports] of boolean; {Is port input or output?}
 si,sj          : array[1..max_ports] of integer;
 sa             : array[1..max_ports,1..max_ports] of complex; {s-params array}
 mate_node      : array[1..4] of net;       {used in layout of clines}
 message        : array[1..3] of file_string;{Displayed message}
 setup_file     : file_string;
 command_f,window_f       : array[1..3] of compt;
 spline_start,spline_end  : spline_param;  {Start and end of list of s-params}
 dirn,cursor_char         : byte;          {dirn 0=North 1=East 2=West 3=South}
 name,network_name        : line_string;       {Names put on artwork}
 key,key_o,chs,previous_key           : char; {keys for linked list}
 plot_start,plot_end,c_plot,plot_des : array[1..max_params] of plot_param;
                                    {start, end, current and design s-params}
 net_start_ptr1,net_start_ptr2,pbeg,ptrvar,ptrall  : marker;
                                        {for copying circuit with blockmoves}

 fmin,finc,               {frequency minimum,frequency increment}
 Z0,                      {characteristic impedance}
```

```
rho_fac,                   {radius factor of smith chart}
q_fac,                     {fd/df=Q}
resln,                     {resolution of circuit drawing in mm}
sfx1,sfy1,                 {scale factors for pixels for circuit drawing}
xrold,yrold,sigma,
sxmax,sxmin,symax,symin,{max and min values on rectangular plot}
reduction,                 {photographic reduction ratio}
er,                        {relative substrate dielectric constant}
bmax,                      {substrate board size in mm}
substrate_t,               {substrate thickness}
con_sep,                   {connector seperation}
freq,design_freq,          {current frequency and design frequency in GHz}
xm,ym,                     {circuit cursor postion in mm}
psx,psy,csx,csy,artwork_cor,
widthZO,                   {width of normalizing impedance}
lengthxm,lengthym          {length in x and y current part}           : real;

cwidthxZO2,cwidthyZO2,pwidthxZO2,pwidthyZO2, {screen and mask half width ZO}
hires_color,                                 {CGA background color}
message_color,                               {color in message block}
key_presses,                                 {determines cursor flash rate}
key_i,key_end,
nft,                                         {number of FFT points}
xi,xii,yi,yii,
window_number,                               {current window number}
ptmax,                                       {maximum number of graph points}
spx,spy,spp,                                 {Re(s),Im(s),|s| dot position}
displayo,display,
dot_step,ydot,mb,rowl,xdot_max,              {Artwork variables}
idb,ticko,iv,xpt,npts,cx,cx3,min_ports,imin  : integer;

read_kbd,board_read,EGA,insert_key,update_flag,
update_key,spline_in_rect,spline_in_smith,filled_OK,remain,circuit_changed,
marker_OK,p_labels,bad_compt,action,demo_mode,port_dirn_used,debug : boolean;

correction_compt,
q_compt,rho_fac_compt,part_start,coord_start,ccompt,compt1,compt3 : compt;

old_net_start,old_net,netK,netL,cnet,net_start,snet_Start : net;

{plot variables}
  GDSTYLE:    ^Byte;
  GDCOLOR,GDMERGE,GDCUR_X,GDCUR_Y,GDVW_X1,GDVW_X2,GDVW_Y1,GDVW_Y2,
  GDASPC1,GDASPC2,GDGSEG,GDTYPE,GDC_FLG,GDS_FLG:     integer;

{.M PUFF1E3}
{$I INCOE3}
{$I INC1E3}
{.M PUFF2E3}
{$I INC2E3}
{.M MAINMODULE}

{START COMPONENT MANIPULATION}
```

```
procedure del_char(tcompt : compt);
{Delete character -- Del}
begin
  tcompt^.changed:=true;
  delete(tcompt^.descript,cx+1,1);
  write_compt(lightgray,ccompt);write(' ');
  gotoxy(tcompt^.xp+cx,tcompt^.yp);
end; {del_char}

procedure back_char(tcompt : compt);
{Backspace and delete character}
begin
  if cx > tcompt^.x_block then begin
    gotoxy(tcompt^.xp+cx,tcompt^.yp);
    cx:=cx-1;
    del_char(tcompt);
  end;
end; {back_char}

procedure add_char(tcompt : compt);
{Add character to parameter or part}
var
  lendes : integer;
begin
  Textcolor(white);
  with tcompt^ do begin
    if not(insert_key) then delete(descript,cx+1,1);
    insert(key,descript,cx+1);
    lendes:=length(descript) ;
    if lendes > xmaxl then begin
      message[1]:='Line too long';
      delete(descript,lendes,1);
      write_message;
    end;
    cx:=cx+1; if cx > xmaxl then cx:=cx-1;
    if right then
      if(xp+length(descript)-1 >= xorig) or (xp+cx >= xorig) then begin
      gotoxy(xorig-1,yp);write(' ');
      xp:=xp-1;
    end;
    write_compt(lightgray,tcompt);
    changed:=true;
    gotoxy(xp+cx,yp);
  end;{with tcompt}
end; {add_char}

procedure choose_part(ky : char);
{Select one of the parts [a..i]}
var
  tcompt : compt;
  found  : boolean;
begin
```

```
 if ky in ['A'..'I'] then ky:=char(ord(ky)+32);
 tcompt:=nil; found:=false;
 repeat
   if tcompt=nil then tcompt:=part_start else tcompt:=tcompt^.next_compt;
   if (tcompt^.descript[1]=ky) and tcompt^.parsed then found:=true
 until (tcompt^.next_compt=nil) or found;
 if found then begin
   write_compt(lightgray,compt1);
   compt1:=tcompt;
   write_compt(white,compt1);
 end else begin
   message[1]:=ky+' is not a';
   message[2]:='valid part';
   update_key:=false;
 end;
end; {choose_part}

{ START CIRCUIT DRAWING}

procedure draw_net(tnet : net);
{Calls routine to draw net on circuit board}
begin
  case tnet^.com^.typ of
    't'  : draw_tline0(tnet,true,false);
    'l'  : draw_tline0(tnet,false,false);
    'd'  : draw_device0(tnet);
    'c'  : begin
             draw_tline0(tnet^.other_net,true,false);
             draw_tline0(tnet,true,true);
           end;
  end; {case}
end; {draw_net}

function con_found : boolean;
{Looks for ccon on cnet in direction of arrow. On exit cnet=network
 to remove or step over. If ccon is connected to an external port then
 cnet is unchanged.}
var
  found : boolean;
begin
  ccon:=nil;found:=false;port_dirn_used:=false;
  if cnet <> nil then begin
  repeat
    if ccon = nil then ccon:=cnet^.con_start else ccon:=ccon^.next_con;
    if dirn=ccon^.dir then found:=true;
  until found or (ccon^.next_con=nil);
  if found then begin
    if ext_port(ccon) then begin
       message[1]:='Cannot go over';
       message[2]:='path to port';
       port_dirn_used:=true;
       update_key:=false;
    end else cnet:=ccon^.mate^.net;
```

```
  end;{if found}
  end;{if cnet}
  con_found:=found;
end; {con_found}

function new_net(ports : integer; choice : boolean) : net;
{Makes a new network on the end of the linked list.
 If choice then network is node else network is part.}
var
  tnet : net;
begin
  if net_start = nil then begin
    New(net_start);
    tnet:=net_start;
  end else begin
    tnet:=net_start;
    while tnet^.next_net <> nil do tnet:=tnet^.next_net;
    new(tnet^.next_net);
    tnet:=tnet^.next_net;
  end;
  with tnet^ do begin
    next_net:=nil;
    node:=choice;
    con_start:=nil;
    ports_connected:=0;
    number_of_con:=ports;
    xr:=xm;
    yr:=ym;
    if node then begin
      grounded:=false;
      com:=nil;
    end else com:=compt1;
  end;{with}
  new_net:=tnet;
  if not(tnet^.node)then
  if compt1^.typ = 'c' then begin
    new(tnet^.other_net);
    tnet:=tnet^.other_net;
    with tnet^ do begin
      com:=compt1;
      dirn_xy;
      xr:=xm+yii*compt1^.con_space;
      yr:=ym+xii*compt1^.con_space;
    end;{with}
  end;{if ccompt1}
end; {new net}

procedure dispose_net(vnet : net);
{Remove a network form the linked list}
var
  found : boolean;
  tnet  : net;
begin
```

```
  tnet:=nil;found:=false;
  repeat
    if tnet = nil then begin
      tnet:=net_start;
      if tnet=vnet then begin
        net_start:=net_start^.next_net;
        tnet:=net_start;
        found:=true;
      end {if tnet}
    end else begin
      if tnet^.next_net=vnet then begin
        found:=true;
        tnet^.next_net:=tnet^.next_net^.next_net
        end else
        tnet:=tnet^.next_net
    end {if tnet=nil}
  until found or (tnet^.next_net=nil);
  if not(found) then begin
    message[2]:='dispose_net';
    shutdown;
  end;
end; {dispose_net}

function new_con(tnet : net; dirt : integer) : conn;
{Make a new connector}
var
  tcon : conn;
begin
  if tnet^.con_start=nil then begin
    new(tnet^.con_start);
    tcon:=tnet^.con_start;
  end else begin
    tcon:=tnet^.con_start;
    while tcon^.next_con <> nil tcon:=tcon^.next_con;
    new(tcon^.next_con);
    tcon:=tcon^.next_con;
  end;
  with tcon^ do begin
    port_type:=0;
    next_con:=nil;
    net:=tnet;
    cxr:=xm; dir:=dirt;
    cyr:=ym;
  end;{with}
  with tnet^ do
  if node and (number_of_con > 1) then begin
    xr:=(xr*(number_of_con-1)+xm)/number_of_con;
    yr:=(yr*(number_of_con-1)+ym)/number_of_con;
  end;
  new_con:=tcon;
end; {new_con}

procedure dispose_con(vcon : conn);
```

```
{Dispose a connector}
var
  found : boolean;
  tcon  : conn;
  vnet  : net;
  i     : integer;
begin
  tcon:=nil;found:=false;
  vnet:=vcon^.net;
  vnet^.number_of_con:=vnet^.number_of_con-1;
  if vnet^.number_of_con=0 then dispose_net(vnet) else begin
    repeat
      if tcon = nil then begin
        tcon:=vnet^.con_start ;
        if tcon=vcon then begin
          vnet^.con_start:=vcon^.next_con;
          found:=true
        end
      end else begin
        if tcon^.next_con=vcon then begin
          found:=true;
          tcon^.next_con:=tcon^.next_con^.next_con
        end else tcon:=tcon^.next_con
      end;
    until found or (tcon^.next_con=nil);
    if not(found) then begin
      message[2]:='dispose_con';
      shutdown;
    end;
  end;{if vcon}
  with vnet^ do
  if node and (number_of_con > 0) then
  for i:=1 to number_of_con do begin
    if i=1 then begin
      tcon:=vnet^.con_start;
      xr:=0; yr:=0;
    end else tcon:=tcon^.next_con;
    xr:=xr+tcon^.cxr/number_of_con;
    yr:=yr+tcon^.cyr/number_of_con;
  end;{for}
end; {dispose_con}

procedure draw_port(mnet : net; col : integer);
{Draws a box a number for an external port}
var
  x,y,i,j : integer;
begin
  Textcolor(col);
  x:=Round(mnet^.xr/csx);
  y:=Round(mnet^.yr/csy);
  i:=0;j:=0;
  case mnet^.ports_connected of
    1 : i:= 0;
```

```
    2 : i:= 2;
    3 : i:= 0;
    4 : i:= 2;
  end;
  gotoxy((x + xmin[2]) div charx+i,(y+ymin) div chary+j+1);
  write(mnet^.ports_connected:1);
  fill_box(xmin[2]+x-2,ymin+y-2,xmin[2]+x+2,ymin+y+2,col);
end; {draw_port}

procedure filltri(x,y,widthx,widthy,col : integer);
{fills a triangle on screen for chamfer on connections to external ports}
var
  i : integer;
begin
  for i:=-widthy to widthy do
  if i<>0 then puff_draw(x+Round(widthx/widthy*(abs(i)-widthy)),y+i,
                   x-Round(widthx/widthy*(abs(i)-widthy)),y+i,col);
end; {filltri}

procedure draw_to_port(tnet : net; port_number : integer);
{Draws a connectinon to an external port}
var
  xp,yp,offset,xli,yli : integer;
  yval : real;
begin
  portnet[port_number]^.node:=true;
  xli:=Round(tnet^.xr/csx)+xmin[2];
  yli:=Round(tnet^.yr/csy)+ymin;
  case port_number of
    1,3 : offset:=2;
    2,4 : offset:=-2;
    else offset:=0;
  end;{case}
  xp:=Round(portnet[port_number]^.xr/csx)+offset+xmin[2];
  yval:=portnet[port_number]^.yr;
  yp:=Round(yval/csy)+ymin;
  if abs(yp-yli) >= cwidthyzO2 then begin
     fill_box(xli-cwidthxZO2,yli,xli+cwidthxZO2,yp,lightgray);
     if EGA then filltri(xli,yp,cwidthxZO2,cwidthyZO2,lightgray);
  end;
  if abs(xp-xli) > 2 then
      fill_box(xli,yp-cwidthyZO2,xp,yp+cwidthyZO2,lightgray);
  draw_port(portnet[port_number],Red)
end; {draw_to_port}

procedure draw_ports(tnet : net);
{Loops over tnet's connections to external ports}
var
  tcon : conn;
begin
  tcon:=nil;
  repeat
    if tcon=nil then tcon:=tnet^.con_start else tcon:=tcon^.next_con;
```

```
    if ext_port(tcon) then draw_to_port(tnet,tcon^.port_type);
  until tcon^.next_con=nil;
end; {draw_ports}

procedure node_look;
{Looks for a node at current cursor postion}
var
  tnet : net;
begin
  cnet:=nil;
  tnet:=nil;
  if net_start <> nil then
  repeat
    if tnet = nil then tnet:=net_start else tnet:=tnet^.next_net;
      with tnet^ do if (abs(con_start^.cxr-xm)< resln) and node and
                       (abs(con_start^.cyr-ym)< resln) then begin
        cnet:=tnet;
        exit;
      end;
  until tnet^.next_net=nil;
end; {node_look}

procedure goto_port(port_number : integer);
{Go to an external port}
begin
  xm:=portnet[port_number]^.xr;     ym:=portnet[port_number]^.yr;
  if port_number=0 then begin
    xrold:=xm;  yrold:=ym;
  end;
  xi:=Round(xm/csx); yi:=Round(ym/csy);
  node_look;
end; {goto_port}

procedure new_port(x,y : real; port_number : integer);
{Make a new external port}
begin
  new(portnet[port_number]);
  with portnet[port_number]^ do begin
    next_net:=nil;
    number_of_con:=0;
    con_start:=nil;
    xr:=x;
    yr:=y;
    ports_connected:=port_number;
    node:=false; {not_connected yet}
  end; {with}
end; {new port}

procedure draw_circuit;
{Draw the entire circuit}
var
  tnet : net;
  port_number,xb,yb,xo,yo : integer;
```

```
begin
  xo:=xmin[2];  xb:=Round(bmax/csx)+xo;
  yo:=ymin;     yb:=Round(bmax/csy)+yo;
  fill_box(0,1,xb+14,yb,black);
  draw_box(xo,yo,xb,yb,lightcyan);
  if net_start= nil then begin
    new_port(bmax/2.0,bmax/2.0,0);
    new_port(0.0,(bmax-con_sep)/2.0,1);
    new_port(bmax,(bmax-con_sep)/2.0,2);
    min_ports:=2;
    if con_sep <> 0 then begin
      new_port(0.0,(bmax+con_sep)/2.0,3);
      new_port(bmax,(bmax+con_sep)/2.0,4);
      min_ports:=4;
    end;
  end; {if cnet}
  Textcolor(Brown);
  for port_number:=1 to min_ports do draw_port(portnet[port_number],brown);
  if net_start <> nil then begin
    tnet:=nil; iv:=1;
    repeat
      if tnet=nil then  tnet:=net_start else tnet:=tnet^.next_net;
        dirn:=tnet^.con_start^.dir;
        if tnet^.node then begin
          if tnet^.grounded then draw_ground0(tnet^.xr,tnet^.yr)
        end else draw_net(tnet);
    until tnet^.next_net = nil;
    tnet:=nil;
    repeat
      if tnet=nil then  tnet:=net_start else tnet:=tnet^.next_net;
      if tnet^.ports_connected > 0 then draw_ports(tnet);
    until tnet^.next_net = nil;
    xi:=Round(xm/csx);
    yi:=Round(ym/csy);
  end else goto_port(0);
end; {draw_circuit}

procedure get_key;
{Get key from keyboard. See Turbo manual Appendix K}
label
  end_blink;
var
  shift            : byte absolute $0000:$0417;
  cursor_displayed : boolean;
  i,key_ord        : integer;
begin
  if read_kbd then begin
    if demo_mode then begin
      readln(key_ord);key:=char(key_ord)
    end else begin
      shift:=shift and $df; {switch off Num lock(bit 5)}
      cursor_displayed:=false;
      if window_number=1 then draw_cursor0 else ggotoxy(cursor_displayed);
```

```
      if not(keypressed) then {blink cursor}
      if window_number <> 1 then
      repeat
        for i:=1 to key_presses do if keypressed then goto end_blink;
        ggotoxy(cursor_displayed);
        shift:=shift and $df;
      until false;
      end_blink:
      read(kbd,key);
      if (key = #27) and keypressed then  begin
        read(kbd,key);  {get non-ASCII key}
        key_ord:=Ord(key);
        if key_ord > 127 then key:=#0 else key:=char(key_ord+128);
      end;
    if((key in['1'..'9'])and((shift and 3)>0))then key:=Char(Ord(key)+128);
    if key=Alt_o then key:=Omega;  {Ohms symbol}
    if key=Alt_d then key:=Degree;
    if window_number=1 then erase_cursor0
                       else if cursor_displayed then ggotoxy(cursor_displayed)
  end; {if demo}
  end else begin      {redraw_circuit}
    if key_i > 0 then
    if key_list[key_i].noden <> node_number then begin
       key:=F3;
       read_kbd:=true;
       compt3:=ccompt; cx3:=compt3^.x_block;
       message[1]:='A connection';
       message[2]:='has changed';
       exit;
    end;
    key_i:=key_i+1;
    if key_i > key_end then begin {end redraw}
      read_kbd:=true;
      key:=key_o;
      circuit_changed:=false;
      draw_circuit;
    end else key:=key_list[key_i].keyl;
  end;{read_kbd}
end; {get_key}

procedure ground_node;
{Ground a node}
begin
  if cnet <> nil then
  with cnet^ do
  if not(grounded) then begin
    grounded:=true;
    draw_ground0(xr,yr);
  end;
end; {ground_node}

procedure unground;
{Remove a ground}
```

```
begin
  if cnet <> nil then
  with cnet^ do
  if grounded then begin
    grounded:=false;
    if read_kbd then draw_circuit;
  end;
end; {unground}

procedure join_port(port_number,ivt : integer);
{Join cnet to an external port}
var
  found : boolean;
  tcon  : conn;
  dirt  : integer;
begin
  if port_number <= min_ports then
  if ivt=1 then begin {connect}
     if ym > portnet[port_number]^.yr then dirt:=0 else dirt:=3;
     if abs(ym - portnet[port_number]^.yr) < widthz0/2.0 then
     case port_number of
     1,3 : dirt:=2;
     2,4 : dirt:=1;
     end;
    if not(portnet[port_number]^.node) then begin
      if cnet = nil then cnet:=new_net(0,true);
      draw_to_port(cnet,port_number);
      tcon:=new_con(cnet,dirt);
      cnet^.ports_connected:=cnet^.ports_connected+1;
      tcon^.port_type:=port_number;
      tcon^.mate:=nil;
      cnet^.number_of_con:= cnet^.number_of_con+1;
    end else begin
      message[1]:='Port '+char(port_number+ord('0'))+' is';
      message[2]:='already joined';
    end;
  end else begin     {erase}
    tcon:=nil;found:=false;
    if cnet <> nil then
    repeat
      if tcon = nil then tcon:=cnet^.con_start else tcon:=tcon^.next_con;
      if tcon^.port_type=port_number then found:=true;
    until found or (tcon^.next_con=nil);
    if found then begin
       dispose_con(tcon); node_look;
       portnet[port_number]^.node:=false;
       cnet^.ports_connected:=cnet^.ports_connected-1;
       if read_kbd then draw_circuit;
    end else goto_port(port_number);
  end;{ivt}
end; {join_port}

function port_or_node_found : boolean;
```

```
{Calls node look to look for a node then looks for a port at
 current cursor postion}
var
  i : integer;
begin
  node_look;
  if cnet <> nil then port_or_node_found:=true else begin
    port_or_node_found:=false;
    for i:=1 to min_ports do
    if (abs(portnet[i]^.xr-xm)< resln) and not(portnet[i]^.node) and
       (abs(portnet[i]^.yr-ym)< resln) then begin
       port_or_node_found:=true;
       join_port(i,1);
       exit
    end;
  end;{if cnet}
end; {port_or_node_found}

procedure add_net;
{Connect up a new network to circuit}
var
  i    : integer;
  vcon : conn;
  vnet : net;
  special_coupler : boolean;
begin
  special_coupler:=look_back0;
  if not(off_board0(1.0) or occupied_port0) then begin
    compt1^.used:=compt1^.used+1;
    vnet:=new_net(compt1^.number_of_con,false);
    draw_net(vnet);
    for i:=1 to compt1^.number_of_con do begin
      if special_coupler and (i in [1,3]) then begin
         cnet:=mate_node[i];
         cnet^.number_of_con:=cnet^.number_of_con+1
      end else begin
        if port_or_node_found then cnet^.number_of_con:=cnet^.number_of_con+1
                                else cnet:=new_net(1,true);{make node}
      end;
      vcon:=new_con(vnet,dirn);
      vcon^.conn_no:=i;
      ccon:=new_con(cnet,dirn);
      ccon^.mate:=vcon;
      vcon^.mate:=ccon;
      if i <> compt1^.number_of_con then increment_pos(i);
    end;{for i}
  end;{off_board0}
end; {add_net}

procedure rem_net;
{Remove a network from the circuit}
var
  i : integer;
```

```
  mnode,snet,onet : net;
begin
  if not(port_dirn_used) then begin
    snet:=nil;
    cnet^.com^.used:=cnet^.com^.used-1;
    for i:=1 to cnet^.number_of_con do begin
      if i=1 then ccon:=cnet^.con_start else ccon:=ccon^.next_con;
      dispose_con(ccon^.mate);
      mnode:=ccon^.mate^.net;
      with mnode^ do if number_of_con=1 then begin
        onet:=cnet;cnet:=mnode;
        if ext_port(con_start) then join_port(con_start^.port_type,0);
        cnet:=onet;
      end;
      if mnode^.number_of_con > 0 then snet:=mnode;
    end;{for i}
    lengthxy(cnet);    increment_pos(1);
    dispose_net(cnet);
    node_look;
    if cnet=nil then begin
       cnet:=snet;
       if cnet <> nil then increment_pos(0);
    end;
    if read_kbd then draw_circuit;
  end; {if not port}
end; {rem_net}

procedure step_line;
{Step a distance = 1/2 part size, on exit cnet points to node if found.}
begin
  if not(port_dirn_used or off_board0(0.5)) then begin
    compt1^.step:=true;
    increment_pos(-1);
    node_look;
  end;{if not port}
end; {step_line}

procedure step_over_line;
{Step over a line, on exit cnet points to node.}
var
  tcon : conn;
  tnet : net;
begin
  if not(port_dirn_used) then begin
    tcon:=ccon^.mate; {network that you are stepping over}
    tnet:=tcon^.net;
    case tcon^.conn_no of
      1,3 : cnet:=tcon^.next_con^.mate^.net;
      2 : cnet:=tnet^.con_start^.mate^.net;
      4 : cnet:=tnet^.con_start^.next_con^.next_con^.mate^.net;
    end;
    increment_pos(0);
  end;{if not port}
```

```
end; {step_over_line}

procedure move_net(dirnt,ivt:integer);
{Procedure for calling one of rem_net, set_over_line, step_line and add_net}
begin
  if compt1^.parsed then begin
    dirn:=dirnt;  iv:=ivt;
    if con_found then if iv=0 then rem_net else step_over_line
                 else if iv=0 then step_line else add_net;
  end else begin
    if not(read_kbd) then begin
      key:=F3;
      read_kbd:=true;
      compt3:=ccompt; cx3:=compt3^.x_block;
    end;
    message[2]:='Invalid part';
    update_key:=false;
  end;{if parsed}
end; {move_net}

procedure pars_compt_list;
{Pars the component list.
 If action is true then find part dimensions else find s-parameters}
var
  pars   : boolean;
  tcompt : compt;
begin
  tcompt:=nil;
  bad_compt:=false;
  repeat
    if tcompt=nil then tcompt:=part_start else tcompt:=tcompt^.next_compt;
    with tcompt^ do begin
      if changed and ((used > 0) or step) then circuit_changed:=true;
      if action then pars:=changed else pars:=used > 0;
      if pars then begin
        parsed:=true;
        if action then typ:=get_lead_char0(tcompt);
        case typ of
          't'  : tline0(tcompt);
          'c'  : clines0(tcompt);
          'd'  : device0(tcompt);
          'l'  : lumped0(tcompt);
          ' '  : parsed:=false;
         else begin
           parsed:=false;
           bad_compt:=true;
           message[1]:=typ+' is an';
           message[2]:='unknown part';
         end;
        end;{case}
      end;{if pars}
      if not(bad_compt) then changed:=false
                        else if window_number=3 then ccompt:=tcompt;
```

```
    end;{with}
  until ((tcompt^.next_compt=nil) or bad_compt);
  if bad_compt then write_message
end; {pars_compt_list}

{$I inc3e3.pas}

{START ARTWORK}
procedure fill_port(tNt : net);
{Perform artwork connections to external ports}
var
  tptyr,tptxr : real;
  tport,tpt   : net;
  tcon        : conn;
  nodet1,i,x1,y1,x2,y2 : integer;
begin
  tcon:=nil;
  repeat
    if tcon = nil then tcon:=tNt^.con_start else tcon:=tcon^.next_con;
    if ext_port(tcon) then begin
      tport:=portnet[tcon^.port_type];
      y1:=Round(tNt^.yr/psy);
      y2:=Round(tport^.yr/psy);
      if ydot=0 then begin
        tpt:=tport;
        x1:=Round(tNt^.xr/psx);
        x2:=Round(tpt^.xr/psx);
        tptxr:=tpt^.xr;          tptyr:=tpt^.yr;

        new(tpt^.other_net); tpt:=tpt^.other_net;
        tpt^.ny1:=y2-pwidthyZO2; tpt^.ny2:=y2+pwidthyZO2;
        if x1 < x2 then begin  tpt^.nx1:=x1;  tpt^.nx2:=x2
              end else begin  tpt^.nx1:=x2;  tpt^.nx2:=x1  end;

        new(tpt^.other_net); tpt:=tpt^.other_net;
        tpt^.nx1:=x1-pwidthxZO2; tpt^.nx2:=x1+pwidthxZO2;
        if y1 < y2 then begin  tpt^.ny1:=y1;  tpt^.ny2:=y2
              end else begin  tpt^.ny1:=y2;  tpt^.ny2:=y1  end;

        if tNt^.yr > tptyr then begin
          if tNt^.xr > tptxr then nodet1:=12 else nodet1:=10;
        end else begin
          if tNt^.xr > tptxr then nodet1:=5  else nodet1:= 3;
        end;
        new(tpt^.other_net); tpt:=tpt^.other_net;{chamfers}
        tpt^.xr:=tNt^.xr; tpt^.yr:=tptyr;tpt^.nodet:=nodet1;
        tpt^.number_of_con:=2;
        init_chamfer0(tpt,widthZO,widthZO);
      end;
      tport:=tport^.other_net;
      fill_shape0(tport,false);  {horiz line}
      if abs(y2-y1) > pwidthyZO2 then begin
        tport:=tport^.other_net;
```

```
        fill_shape0(tport,false);{vert. line}
        tport:=tport^.other_net;
        fill_shape0(tport,true);
      end;
    end;{if tcon}
  until tcon^.next_con=nil;
end; {fill_port}

procedure net_loop;
{Loop over parts for artwork mask}
var
  tnet : net;
  widthx,widthy : real;
begin
  remain:=false;
  ydot:=ydot+dot_step;
  tnet:=nil ;
  repeat
    if tnet=nil then tnet:=net_start else tnet:=tnet^.next_net;
    if ydot=0 then begin
       if tnet^.node then begin
         get_widthxy0(tnet,widthx,widthy);
         init_chamfer0(tnet,widthx,widthy)
       end else begin
         dirn:=tnet^.con_start^.dir;
         init_line0(tnet);
         if tnet^.com^.typ='c' then init_line0(tnet^.other_net);
       end;{if tnet^.node}
    end;{if ydot}
    if not(tnet^.node) then begin
      if tnet^.com^.typ in ['t','c'] then fill_shape0(tnet,false);
      if tnet^.com^.typ in ['c'] then fill_shape0(tnet^.other_net,false);
    end;
  until tnet^.next_net=nil;
  tnet:=nil ;
  repeat
    if tnet=nil then tnet:=net_start else tnet:=tnet^.next_net;
    if tnet^.node then begin
      if tnet^.ports_connected > 0 then fill_port(tnet);
      if tnet^.chamfer then fill_shape0(tnet,true);
    end;
  until tnet^.next_net=nil;
end; {net_loop}

procedure matrix_artwork;
{Procedure for directing artwork}
label
  exit_artwork;
var
  ix : integer;
begin
  if net_start=nil then begin
    message[1]:='No circuit';
```

```
    message[2]:='to do artwork';
    write_message;
  end else begin
    if reduction*bmax > 8*25.4 then begin
      message[1]:='Reduction ratio';
      message[2]:='is too small to';
      message[3]:='do artwork';
      write_message;
    end else begin
      ydot:=0;dot_step:=0;
      remain:=true;
      xdot_max:=Round(reduction*bmax*120/25.4);
      if xdot_max > 960 then xdot_max:=960;
      mb:=-1;
      p_labels:=top_labels0;
      if p_labels then begin
        message[2]:='Press s to stop';write_message;
        while remain do begin
          if keypressed then begin
            read(kbd,chs);
            if chs in ['s','S'] then begin
              message[2]:='      STOP       ';
              write_message;
              goto exit_artwork;
            end;{if key}
            beep;
          end;
          rowl:=0;
          for ix:=0 to xdot_max do bita[ix]:=0;
          mb:=mb+1;
          net_loop;
          if (rowl > 0) and p_labels then print_labels0;
          if not(p_labels) then begin
            if rowl > xdot_max then rowl:=xdot_max;
            write(lst,#27'L', chr((rowl+1) mod 256), chr((rowl+1) div 256));
            for ix := 0 to rowl do write(lst,chr(bita[ix]));
            write(lst,#13);
            if odd(mb) then write(lst,#27'J',#13) else write(lst,#27'J',#11);
          end;
          if odd(mb) then dot_step:=9 else dot_step:=7;
        end;{while}
        message[2]:='Artwork completed';
        write_message;
      end;{if p_labels}
      exit_artwork:
      reset_printer;
    end;
  end;
end; {matrix_artwork}

procedure time_res;
{Main procedure for FFT to get time response}
var
```

```
  nf,xx,delt,time : real;
  nn,istart,ifinish,x1,y1,i,ij,col : integer;
begin
  erase_message;
  istart:=Round(fmin/finc);
  ifinish:=istart+npts;
  if ifinish > nft then begin
    message[2]:='fmax/'+delta+'f too large';
    write_message;
    exit;
  end;
  if step_fn then begin
    nf:=0;
    for i:= 1 to (nft+1) do begin
      if betweeni(istart,i-1,ifinish) then begin
        if not(odd(i)) then begin
          if odd(i div 2) then nf:=nf+(1.0+cos(pi*(i-1)/(ifinish+1)))/(i-1)
                          else nf:=nf-(1.0+cos(pi*(i-1)/(ifinish+1)))/(i-1);
        end;
      end;
    end;
    sxmin:= -q_fac/(2*design_freq);
    sxmax:=  q_fac/(2*design_freq);
    nf:=(pi*nft)/(nf*0.5*4);
  end else begin
    xx:=pi/(ifinish+1);
    nn:=ifinish-istart+1;
    nf:=2.0*nft/(nn+1-cos(nn*xx/2.0)*sin((nn+1)*xx/2.0)/sin(xx/2.0));
    sxmin:= -q_fac/(8*design_freq);
    sxmax:=3*q_fac/(8*design_freq);
  end;
  marker_OK:=false;
  delt:=1.0/(2.0*nft*finc);
  symax:= rho_fac;
  symin:=-rho_fac;
  draw_graph(xmin[1],ymin,xmax[1],ymax,true);
  sfx1:=(xmax[1]-xmin[1])/(sxmax-sxmin);
  sfy1:=(ymax-ymin)/(symax-symin);
  for ij:=1 to max_params do
   if s_param_table[ij]^.changed then begin
    col:=s_color[ij];
    fill_data0(ij,istart,ifinish,nf);
    realft(data,ij,nft,-1);
    for i:= 1 to 2*nft do begin
      time:=(i-1)*delt;if time > sxmax then time := time-2*nft*delt;
      x1:=xmin[1]+Round((time-sxmin)*sfx1);
      y1:=ymax-Round((data[i,ij]-symin)*sfy1);
      if betweeni(ymin,y1,ymax) and betweeni(xmin[1],x1,xmax[1])
                                   then puff_plot(x1,y1,col);
    end;{i}
  end;{ij}
  message[1]:='Type any key';
  message[2]:='to return to the';
```

```
  message[3]:='frequency domain';
  write_message;read(kbd,chs);if keypressed then read(kbd,chs); erase_message;
  draw_graph(xmin[1],ymin,xmax[1],ymax,false);
end; {time_res}

procedure move_marker(xi : integer);
{Move marker on Smith chart and rectangular plot}
var
  i,ij,k,kk,nb,sfreq : integer;
begin
  if marker_OK then begin
    for ij:=1 to max_params do
    if s_param_table[ij]^.changed then
    case xi of
    0 : begin
          if plot_des[ij]=nil then xpt:=0
          else begin
             c_plot[ij]:=plot_des[ij];
             xpt:=Round((design_freq-fmin)/finc);
             if not(betweeni(0,xpt,npts)) then xpt:=0;
          end;
          if xpt=0 then c_plot[ij]:=plot_start[ij];
        end; { 0: }
    1 : if c_plot[ij]=plot_end[ij] then c_plot[ij]:=plot_start[ij]
                                   else c_plot[ij]:=c_plot[ij]^.next_p;
   -1 : if c_plot[ij]=plot_start[ij] then c_plot[ij]:=plot_end[ij]
                                     else c_plot[ij]:=c_plot[ij]^.prev_p;
    end; {case}
    if xi = 0 then for i:=1 to 2*max_params do box_filled[i]:=false
              else xpt:=xpt+xi;
    if xpt > npts then xpt:=0;
    if xpt < 0    then xpt:=npts;
    write_freq0;
    sfreq:=xmin[1]+Round((freq-sxmin)*sfx1);
    for k:=1 to 3 do
    for ij:=1 to max_params do
    if s_param_table[ij]^.changed then
    case k of
     1 : restore_box0(ij);
     2 : begin
           box_filled[ij]:=false;
           box_filled[ij+max_params]:=false;
           if c_plot[ij]^.filled then begin
             calc_pos0(c_plot[ij]^.x,c_plot[ij]^.y,0,1,sfreq,false);
             if spline_in_smith then move_box0(spx ,spy,ij);
             if spline_in_rect  then move_box0(sfreq,spp,ij+max_params);
           end;
         end; {2:}
     3 : begin
           for kk:=0 to 1 do begin
            nb:=ij+kk*max_params;
            if box_filled[nb] then pattern(box_dot[1,nb],box_dot[2,nb],ij,128)
           end;{kk}
```

```
          if c_plot[ij]^.filled then write_s0(ij);
        end;{3 :}
    end;{case}
  end; {if do time}
end; {move_marker}

procedure plot_manager(do_analysis,clear_plot,do_time,boxes : boolean);
{main procedure for directing anlaysis followed by plotting}
begin
  ticko:=ticks;
  if do_time then begin
    if filled_OK then time_res
  end else begin
    erase_message;
    get_coords0;
    if bad_compt then begin
      write_message;
      cx:=ccompt^.x_block;
      filled_OK:=false;
    end else begin
      draw_smith0;
      draw_graph(xmin[1],ymin,xmax[1],ymax,false); cx:=ccompt^.x_block;
      if not(clear_plot) and filled_OK then begin
        smith_and_magplot(true,true);
        move_marker(0);
      end;
      if do_analysis then analysis;
      if filled_OK then begin
        marker_OK:=true;
        smith_and_magplot(false,boxes);
        ticko:=ticks-ticko;
        erase_message;
        Textcolor(lightgray);
        gotoxy(32,12);write('Time',ticko/18.2:8:1,' secs');
        move_marker(0);beep;
        if demo_mode then rcdelay(300);
      end else marker_OK:=false;
    end;
  end;{if bad_compt}
end; {plot_manager}

procedure erase_circuit;
{Erase circuit board}
var
  tcompt : compt;
begin
  erase_message;
  if compt1 <> nil then write_compt(lightgray,compt1);
  compt1:=part_start;
  if pbeg <> nil then release(pbeg); {release networks}
  tcompt:=nil;
  repeat
    if tcompt=nil then tcompt:=part_start else tcompt:=tcompt^.next_compt;
```

```
    with tcompt^ do begin
      used:=0;  step:=false;
      if typ='d' then begin {only pars device if changed or above pbeg}
        if (f_file=nil) or (pbeg=nil) then changed:=true else
        changed:=changed or
               ((seg(f_file^)-seg(pbeg^)) shl 4 + ofs(f_file^)-ofs(pbeg^)>=0);
        if changed then begin
          action:=true; device0(tcompt); changed:=false;
        end; {if changed}
      end;{if type}
    end; {with}
  until tcompt^.next_compt=nil;
  mark(pbeg); {mark off networks}
  key_i:=0; {set_up for redraw}
  if read_kbd then begin
    filled_OK:=false;  circuit_changed:=false;
    marker_OK:=false;  key_end:=0; {erase key_list}
  end;
  net_start:=nil;  cnet:=nil;
  draw_circuit;
end; {erase_circuit}

procedure redraw_circuit;
{Set up for circuit redraw}
begin
  read_kbd:=false;  erase_circuit;
  key_o:=key; key:=F1;
end; {redraw_circuit}

function setupexists(var fname : file_string) : boolean;
{Look for setup.puf in current directory of in \PUFF}
var
  found : boolean;
begin
  found:=false;
  if fname <> 'setup.puf' then begin
    fname:='setup.puf';
    found:=fileexists(false,net_file,fname);
  end;
  if not(found) then begin
    fname:='\PUFF\setup.puf';
    found:=fileexists(false,net_file,fname);
  end;
  setupexists:=found;
end; {setupexists}

procedure read_setup(fname : file_string);
{Read board parameters in setup.puf}
var
  char1,char2 : char;
begin
  if setupexists(fname) then begin
    repeat
```

```
      readln(net_file,char1,char2);
    until ((char1='\') and (char2='b')) or Eof(net_file);
    if not(EOF(net_file)) then read_board0;
    close(net_file);
  end;
  if not(board_read) then begin
    erase_message;
    message[2]:='Board parameters';
    message[3]:='not found';
    shutdown;
  end;
end; {read_setup}

procedure screen_init;
{Initialise screen}
begin
  displayo:=display;
  EGAgraphics0;   set_up_char0; message_color:=lightred;
  write_compt(col_window[1],window_f[1]);
  draw_box(240,10*chary-6,389,13*chary+4,lightred);
  draw_box(286,202,448,343,green);
  bad_compt:=false;
  {draw_graph(xmin[1],ymin,xmax[1],ymax,false);}
  write_compt(col_window[2],window_f[2]);
  key:=F3;  compt3:=part_start;  cx3:=compt3^.x_block;
end; {screen_init}

procedure read_net(fname : file_string);
{Read .puf file}
var
  char1,char2 : char;
  file_read   : boolean;
  tname       : file_string;
begin
  file_read:=false;  marker_OK:=false;  filled_OK:=false;
  tname:=fname;
  if (pos('.',fname)=0) and (fname<>'') then fname:=fname+'.puf';
  if fileexists(true,net_file,fname) then begin
    read(net_file,char1);
    repeat
      readln(net_file,char2);{writeln(lst,char1,char2);}
      if char1='\' then begin
        case char2 of
          'b' : begin {board parameters}
                  read_board0;
                  if board_read then begin
                    if display <> displayo then screen_init;
                    erase_circuit;
                  end;
                end;
          'k' : begin {read in 'key' = plot parameters}
                  read_key0;   set_up_key0;
                  bad_compt:=false; rho_fac:=get_real(rho_fac_compt,1);
```

```
                if bad_compt or (rho_fac<=0.0) then begin
                   rho_fac:=1;
                   rho_fac_compt^.descript:='Smith radius 1.0';
                end;
              end;
        'p' : read_parts0;    {read parts list}
        's' : read_s_params0; {read in calculated s-parameters}
        'c' : read_circuit0;  {read circuit}
        'e' : ;               {end}
       else read(net_file,char1);
      end;{case}
    end else read(net_file,char1);
  until ((char1='\') and (char2='e')) or EOF(net_file);
  close(net_file);
  file_read:=true;
end;
if not(board_read) then begin
  tname:='setup';
  erase_message;
  message[1]:='Missing board#';
  message[2]:=tname;
  message[3]:='Trying setup';
  write_error(1);
  read_setup(fname);
  if display <> displayo then screen_init;
  erase_circuit;
  file_read:=true;
end;
if file_read then begin
  write_file_name0(tname);
  ccompt:=q_compt;  if filled_OK then plot_manager(false,true,false,true);
  new(pbeg);{mark off block}
  key:=F3;
  ccompt:=part_start;  cx:=ccompt^.x_block;
  compt3:=ccompt;      cx3:=cx;
  action:=true;  pars_compt_list;
  write_parts_list0;
  if not(filled_OK) then begin
    draw_graph(xmin[1],ymin,xmax[1],ymax,false);
    draw_smith0;
  end;
end;
end; {read_net}

procedure save_net;
{Save .puf file}
var
  fname : file_string;
  drive : integer;
begin
  fname:=input_string('File to save ckt');
  if fname='' then exit;
  if Pos(':',fname)=2 then drive:=ord(fname[1])-ord('a')+1
```

```
                              else drive:=-1;
  if enough_space(drive) then begin
    write_file_name0(fname);
    if pos('.',fname)=0 then fname:=fname+'.puf';
    assign(net_file,fname);  {$I-} rewrite(net_file);{$I+}
    if IOresult=0 then begin
      save_board0;
      save_key0;
      save_parts0;
      save_s_params0;
      save_circuit0;
      writeln(net_file,'\e',lbrack,'nd',rbrack);
      close(net_file);
      erase_message
    end else begin
      message[1]:='Invalid filename';
      write_message
    end;{if ioresult}
  end;{if enough}
end; {save_net}

procedure puff_start;
{Start puff}
var
  memK,ij,i : integer;
  cspc      : spline_param;
begin
  ClrScr; Textcolor(white);
  gotoxy(31,4);write('PUFF, TRW version E.3');
  gotoxy(23,6);write('Richard Compton and David Rutledge');
  gotoxy(23,7);write('California Institute of Technology');
  gotoxy(32,8);write('Copyright, 1987');
  message_color:=white;
  intr($11,result); {Get equipment list -- Norton page 233}
  if result.ax and 2 = 0 then begin
    message[2]:='8087 not found';
    shutdown {if 8087 not present must shutdown or program will lock up}
  end;
  {debug=true will print debugging information}
  {read(idb); if idb > 0 then debug:=true else} debug:=false;
  insert_key:=false;  co1:=co(1.0,0);   displayo:=-1; pbeg:=nil;
  key_end:=0;{erase key list} erase_message; {init message[i]}
  gotoxy(1,24);
  new(char_p);  getmem(char_p,14*256); {free memory for extendened characters}
  ticko:=ticks;  for i:=1 to 2295 do if keypressed then ;
  key_presses:=8000 div (ticks-ticko); {rate at which cursor flashes}
  memK:=memavail;{check to see if there is enough memory left yo run Puff}
  if memK < 0 then memK:=Round(memK/64+1024) else memK:=Round(memK/64);
  case memK of
      1..80   : begin
                  message[2]:='Insufficient';
                  message[3]:='memory to run Puff';
                  shutdown
```

```
                end;
      81..120  : ptmax:=100;
     121..240  : ptmax:=500;
     else ptmax:=1000;
   end;{case}
   for xpt:=0 to ptmax do begin
     if xpt=0 then begin
       new(spline_start);
       cspc:=spline_start;
     end else begin
       new(cspc^.next_c);
       cspc^.next_c^.prev_c:=cspc;
       cspc:=cspc^.next_c;
     end;
     for ij:=1 to max_params do
     if xpt=0 then begin
       new(plot_start[ij]);
       c_plot[ij]:=plot_start[ij];
     end else begin
       new(c_plot[ij]^.next_p);
       c_plot[ij]^.next_p^.prev_p:=c_plot[ij];
       c_plot[ij]:=c_plot[ij]^.next_p;
     end;
   end;{xpt}
   new(correction_compt);
   make_coord_and_parts_list0;
   compt1:=nil;
   title0;
   for i:=1 to  6 do s_key[i]:=' ';
   for i:=7 to 10 do s_key[i]:='';
   set_up_key0;
   board_read:=false; for i:=1 to 11 do board[i]:=false;
   read_kbd:=true; update_flag:=true;
   setup_file:=commandline0;
   if pos('.',setup_file)=0 then setup_file:=setup_file+'.puf';
   if not(fileexists(setup_file<>'setup.puf',net_file,setup_file)) then begin
     if not(setupexists(setup_file)) then begin
       erase_message;
       message[2]:='setup.puf';
       message[3]:='not found';
       shutdown;
     end;
   end;
   close(net_file);
 end; {puff_start}

 procedure check_esc;
 {Check that on exit Esc key was not accidently pressed}
 var
   tcompt : compt;
 begin
   message[1]:='Exit? Type Esc to';
   message[2]:='confirm, or other';
```

```
  message[3]:='  key to resume  ';
  write_message;
  tcompt:=ccompt;ccompt:=nil;{so cursor doesn't blink}get_key;ccompt:=tcompt;
  erase_message;
  if key <> Esc then key:=not_esc;
end; {check_esc}

procedure circuit1;
begin
  insert_key:=false;
  write_compt(white,compt1);
  repeat
    get_key; if (key <> F3) and read_kbd then erase_message;
    case key of
      Ctrl_e      : begin erase_circuit; write_compt(white,compt1); end;
      {Ctrl_d      : dump0;
      'c'         : clr_plane0(darkGray);}
      Esc         : check_esc;
      else begin {HKMP bad}
        update_key:=read_kbd;
        case key of
          right_arrow : move_net(1,1);
          left_arrow  : move_net(2,1);
          down_arrow  : move_net(3,1);
          up_arrow    : move_net(0,1);
          sh_right    : move_net(1,0);
          sh_left     : move_net(2,0);
          sh_down     : move_net(3,0);
          sh_up       : move_net(0,0);
          sh_1        : join_port(1,0);
          sh_2        : join_port(2,0);
          sh_3        : join_port(3,0);
          sh_4        : join_port(4,0);
          'a'..'i',
          'A'..'I'    : choose_part(key);
          '1'..'4'    : join_port(ord(key)-ord('1')  +1,1);
          '='         : ground_node;
          '+'         : unground;
          Ctrl_n      : snap0;
          else begin
             update_key:=false;
             if not(key in [F1..F3]) then beep;
          end;
        end;{else case}
        if update_key then update_key_list(node_number);
       end;
    end;{case}
    dx_dy0;
    write_message;
  until key in [F2..F3,Esc];
  write_compt(lightgray,compt1);
end; {circuit1}
```

```
procedure plot2;
begin
  ccompt:=q_compt;  cx:=ccompt^.x_block; previous_key:=' ';
  repeat
    ticko:=ticks;
    get_key;
    case key of
      '0'..'9','.',' ','-','+'
                      : add_char(ccompt);
       Del            : del_char(ccompt);
       backspace      : back_char(ccompt);
       Ins            : insert_key:=not(insert_key);
       Up_Arrow       : move_cursor( 0,-1);
       Down_arrow     : move_cursor( 0, 1);
       Left_arrow     : move_cursor(-1, 0);
       Right_arrow    : move_cursor( 1, 0);
       PgDn           : move_marker(-1);
       PgUp           : move_marker(+1);
       't','T'        : begin
                          if not(previous_key in ['p','P',Ctrl_p]) then
                          plot_manager(true,true,false,false); {analyze}
                          plot_manager(true,true, true,false);  {time}
                        end;
       Ctrl_s         : save_net;
       Ctrl_a         : matrix_artwork;
       Ctrl_p         : plot_manager( true,false,false,true); {replot}
       'p','P'        : plot_manager( true, true,false,false); {analyze}
       Esc            : check_esc;
       else if not(key in [F1,F3]) then beep;
    end;{case}
    previous_key:=key;
  until key in [F1,F3,Esc];
end; {plot2}

procedure parts3;
label
  component_start;
begin
  update_command0;
  ccompt:=compt3;  cx:=cx3;
  component_start:
  repeat
    if read_kbd then get_key;
    case key of
      right_arrow    : move_cursor( 1, 0);
      left_arrow     : move_cursor(-1, 0);
      Ins            : insert_key:=not(insert_key);
      'a'..'z','A'..'Z','0'..'9','+','-','.',',',' ',Omega,Degree,'(',')'
                     : add_char(ccompt);
      del            : del_char(ccompt);
      backspace      : back_char(ccompt);
      down_arrow     : move_cursor( 0, 1);
      up_arrow       : move_cursor( 0,-1);
```

```
      Ctrl_r        : begin
                        read_net(input_string('net file to read'));
                        write_commands0;
                      end;
      Ctrl_u        : begin
                        update_flag:=not(update_flag);
                        update_command0;
                      end;
      Ctrl_e        : erase_circuit;
      Esc           : check_esc;
       else if not(key in [F1..F3]) then beep;
    end; {case}
  until key in [F1..F3,Esc];
  compt3:=ccompt;  cx3:=cx;
  if key <> Esc then begin
    erase_message;
    action:=true;  pars_compt_list;
    if bad_compt then begin
      read_kbd:=true;
      cx:=ccompt^.x_block;
      goto component_start;
    end;
  end;
  if update_flag and circuit_changed and (key <> esc) then redraw_circuit;
end; {parts3}

begin
  puff_start;
  read_net(setup_file); read_kbd:=not(circuit_changed);
  repeat
    window_number:=Ord(key)-Ord(F1)+1;
    write_commands0;
    case window_number of
      1 : circuit1;
      2 : plot2;
      3 : parts3;
    end;
    if not(EGA) then with window_f[window_number]^ do
                      draw(xp*8-1,yp*8-9,xp*8+15,yp*8-9,0);
    write_comptm(3,col_window[window_number],window_f[window_number]);
  until key= Esc;
  textmode(bw80);
end.
```

```
{Include file #0: INCOE3.PAS}

{EGA INLINE  by Kent Cedola, 2015 Meadow Lake Ct., Norfolk, VA, 23518}

procedure GPLINE(X1,Y1 : Integer);
{Draw line for (gdcur_x,gdcur_y) to (x,y)}
begin
  inline
    ($55/$8B/$0E/GDCUR_X /$8B/$1E/GDCUR_Y /$8B/$76/$06 /$8B/$7E/$04
      /$89/$36/GDCUR_X /$89/$3E/GDCUR_Y /$C7/$06/GDC_FLG/>$00 /$33/$C0
      /$3B/$1E/GDVW_Y1 /$7D/$03 /$80/$CC/$08 /$3B/$1E/GDVW_Y2 /$7E/$03
      /$80/$CC/$04 /$3B/$0E/GDVW_X1 /$7D/$03 /$80/$CC/$02 /$3B/$0E/GDVW_X2
     /$7E/$03 /$80/$CC/$01 /$3B/$3E/GDVW_Y1 /$7D/$02 /$0C/$08 /$3B/$3E/GDVW_Y2
      /$7E/$02 /$0C/$04 /$3B/$36/GDVW_X1 /$7D/$02 /$0C/$02 /$3B/$36/GDVW_X2
      /$7E/$02 /$0C/$01 /$0B/$C0 /$75/$03 /$E9/>$99 /$C7/$06/GDC_FLG/>$01
      /$84/$E0 /$74/$09 /$C7/$06/GDC_FLG/>$02 /$E9/$0270 /$0A/$C0 /$75/$06
      /$87/$CE /$87/$DF /$86/$E0 /$A8/$02 /$75/$24 /$A8/$04 /$75/$3C /$A8/$08
      /$75/$54 /$8B/$EF /$2B/$EB /$A1/GDVW_X2 /$2B/$C1 /$F7/$ED /$8B/$EE
      /$2B/$E9 /$F7/$FD /$03/$C3 /$8B/$36/GDVW_X2 /$8B/$F8 /$E9/$FF6E /$8B/$EF
      /$2B/$EB /$A1/GDVW_X1 /$2B/$C1 /$F7/$ED /$8B/$EE /$2B/$E9 /$F7/$FD
      /$03/$C3 /$8B/$36/GDVW_X1 /$8B/$F8 /$E9/$FF52 /$8B/$EE /$2B/$E9
      /$A1/GDVW_Y2 /$2B/$C3 /$F7/$ED /$8B/$EF /$2B/$EB /$F7/$FD /$03/$C1
      /$8B/$F0 /$8B/$3E/GDVW_Y2 /$E9/$FF36 /$8B/$EE /$2B/$E9 /$A1/GDVW_Y1
      /$2B/$C3 /$F7/$ED /$8B/$EF /$2B/$EB /$F7/$FD /$03/$C1 /$8B/$F0
      /$8B/$3E/GDVW_Y1 /$E9/$FF1A /$BA/$03CE /$8A/$26/GDMERGE /$B0/$03 /$EF
      /$B8/$0205 /$EF /$8B/$D6 /$3B/$D1 /$73/$04 /$87/$CA /$87/$DF /$2B/$D1
      /$2B/$FB /$8B/$F3 /$D1/$E6 /$D1/$E6 /$03/$F3 /$D1/$E6 /$D1/$E6 /$D1/$E6
     /$D1/$E6 /$8B/$D9 /$D1/$EB /$D1/$EB /$D1/$EB /$03/$DE /$8B/$F2 /$BA/$03CE
      /$B0/$08 /$EE /$42 /$80/$E1/$07 /$B0/$80 /$D2/$C8 /$83/$3E/GDS_FLG/$00
      /$75/$03 /$E9/>$BE /$56 /$C4/$36/GDSTYLE /$26/$80/$3C/$01 /$75/$08
      /$26/$8A/$64/$01 /$5E /$E9/>$AB /$5E /$89/$76/$04 /$89/$7E/$06 /$0B/$FF
      /$79/$0C /$F7/$DF /$3B/$FE /$77/$03 /$EB/$73/$90 /$EB/$4D/$90 /$3B/$FE
      /$77/$25 /$8B/$CE /$8B/$FE /$D1/$EF /$F7/$DF /$C4/$36/GDSTYLE
      /$26/$8A/$24 /$46 /$E8/$0125 /$D0/$C8 /$83/$D3/$00 /$03/$7E/$06 /$78/$F3
      /$83/$C3/$50 /$2B/$7E/$04 /$EB/$EB /$8B/$CF /$D1/$EF /$F7/$DF
      /$C4/$36/GDSTYLE /$26/$8A/$24 /$46 /$E8/$0102 /$83/$C3/$50 /$03/$7E/$04
      /$78/$F5 /$D0/$C8 /$83/$D3/$00 /$2B/$7E/$06 /$EB/$EB /$8B/$CF /$D1/$EF
      /$F7/$DF /$C4/$36/GDSTYLE /$26/$8A/$24 /$46 /$E8/>$DF /$83/$EB/$50
      /$03/$7E/$04 /$78/$F5 /$D0/$C8 /$83/$D3/$00 /$03/$7E/$06 /$EB/$EB
      /$8B/$CE /$8B/$FE /$D1/$EF /$F7/$DF /$C4/$36/GDSTYLE /$26/$8A/$24 /$46
      /$E8/>$BA /$D0/$C8 /$83/$D3/$00 /$2B/$7E/$06 /$78/$F3 /$83/$EB/$50
      /$2B/$7E/$04 /$EB/$EB /$8A/$26/GDCOLOR /$55 /$8E/$06/GDGSEG /$0B/$FF
      /$79/$08 /$F7/$DF /$3B/$FE /$77/$4C /$EB/$6D /$3B/$FE /$77/$23 /$8B/$CE
      /$8B/$EE /$D1/$ED /$F7/$DD /$EE /$26/$80/$3F/$00 /$26/$88/$27 /$49
      /$78/$79 /$D0/$C8 /$83/$D3/$00 /$03/$EF /$78/$EC /$83/$C3/$50 /$2B/$EE
      /$EB/$E5 /$8B/$CF /$8B/$EF /$D1/$ED /$F7/$DD /$EE /$26/$80/$3F/$00
      /$26/$88/$27 /$49 /$78/$56 /$83/$C3/$50 /$03/$EE /$78/$EE /$D0/$C8
      /$83/$D3/$00 /$2B/$EF /$EB/$E5 /$8B/$CF /$8B/$EF /$D1/$ED /$F7/$DD /$EE
      /$26/$80/$3F/$00 /$26/$88/$27 /$49 /$78/$33 /$83/$EB/$50 /$03/$EE
      /$78/$EE /$D0/$C8 /$83/$D3/$00 /$2B/$EF /$EB/$E5 /$8B/$CE /$8B/$EE
      /$D1/$ED /$F7/$DD /$EE /$26/$80/$3F/$00 /$26/$88/$27 /$49 /$78/$10
      /$D0/$C8 /$83/$D3/$00 /$03/$EF /$78/$EC /$83/$EB/$50 /$2B/$EE /$EB/$E5
      /$5D /$EB/$25 /$EE /$50 /$26/$8A/$04 /$8E/$06/GDGSEG /$26/$8A/$27
```

```
      /$26/$88/$07 /$8E/$06/GDSTYLE+2 /$58 /$FE/$CC /$75/$07 /$8B/$36/GDSTYLE
      /$26/$8A/$24 /$46 /$49 /$78/$01 /$C3 /$58 /$BA/$03CE /$B8/>$03 /$EF
      /$B8/>$05 /$EF /$B8/$FF08 /$EF/$5D);
end; {gpline draw}

procedure GPBOX1(X1,Y1,X2,Y2 : Integer);
{Fillbox(x1,y1,x2,y2) with gdcolor}
begin
  inline
    ($BA/$03CE/$8A/$26/GDMERGE/$B0/$03/$EF/$B8/$0205/$EF/$8B/$46/$0A/$8B/$5E/
     $08/$8B/$7E/$06/$8B/$C8/$80/$E1/$07/$B2/$FF/$D2/$EA/$88/$56/$0A/$8B/$CF/
     $80/$E1/$07/$B2/$80/$D2/$FA/$88/$56/$08/$8B/$4E/$04/$8B/$D0 /$D1/$EA/$D1/
     $EA/$D1/$EA/$D1/$EF/$D1/$EF/$D1/$EF/$2B/$FA/$75/$0B/$50/$8A/$46/$08/$22/
     $46/$0A/$88/$46/$08/$58/$4F/$2B/$CB/$41/$8B/$F0/$D1/$EE/$D1/$EE/$D1/$EE/
     $8B/$C3/$D1/$E0/$D1/$E0/$03/$C3/$05/$A000/$8E/$C0/$BA/$03CE/$B0/$08/$EE/
     $42/$8A/$26/GDCOLOR/$8A/$46/$0A/$EE/$E8/>$18/$0B/$FF/$78/$12/$74/$09/$B0/
     $FF/$EE/$E8/>$0C/$4F/$75/$FA/$8A/$46/$08/$EE/$E8/>$02/$EB/$11/$51/$56/
     $26/$8A/$04/$26/$88/$24/$83/$C6/$50/$E2/$F5/$5E/$59/$46/$C3/$B0/$FF/$EE/
     $4A/$B8/>$03/$EF/$B8/>$05/$EF);
end; {gpbox1}

procedure GPCIR(Radius : Integer);
{Draw circle with center (dcur_x,gdcur_y)}
begin
  inline
    ($83/$EC/$42 /$C7/$06/GDC_FLG/>$02 /$8B/$46/$04 /$0B/$C0 /$75/$01 /$40
      /$8B/$D8 /$A1/GDCUR_X /$2B/$C3 /$3B/$06/GDVW_X2 /$76/$03 /$E9/$02DC
      /$03/$C3 /$03/$C3 /$3B/$06/GDVW_X1 /$73/$03 /$E9/$02CF /$8B/$C3
      /$F7/$26/GDASPC1 /$F7/$36/GDASPC2 /$8B/$D0 /$8B/$0E/GDCUR_Y /$2B/$C8
      /$3B/$0E/GDVW_Y2 /$76/$03 /$E9/$02B4 /$89/$4E/$C2 /$03/$C8 /$03/$C8
      /$3B/$0E/GDVW_Y1 /$73/$03 /$E9/$02A4 /$89/$4E/$C0 /$C7/$06/GDC_FLG/>$00
      /$D1/$E0 /$D1/$E0 /$03/$C2 /$8B/$F0 /$A1/GDCUR_Y /$D1/$E0 /$D1/$E0
      /$03/$06/GDCUR_Y /$05/$A000 /$2B/$C6 /$89/$46/$F6 /$03/$C6 /$03/$C6
     /$89/$46/$F4 /$8B/$36/GDCUR_X /$89/$76/$C6 /$89/$76/$C4 /$8B/$CE /$D1/$EE
      /$D1/$EE /$D1/$EE /$B0/$80 /$80/$E1/$07 /$D2/$C8 /$88/$46/$FE
      /$89/$76/$FC /$88/$46/$FA /$89/$76/$F8 /$8B/$CA /$8B/$C2 /$F7/$E2
      /$89/$46/$F2 /$89/$56/$F0 /$D1/$E0 /$D1/$D2 /$89/$46/$EA /$89/$56/$E8
      /$8B/$C3 /$F7/$E3 /$52 /$50 /$D1/$E0 /$D1/$D2 /$89/$46/$E6 /$89/$56/$E4
      /$58 /$D1/$E1 /$49 /$F7/$E1 /$89/$46/$EE /$89/$56/$EC /$58 /$F7/$E1
      /$01/$46/$EC /$33/$C0 /$89/$46/$D6 /$89/$46/$D4 /$BA/$03CE
      /$8A/$26/GDMERGE /$B0/$03 /$EF /$B8/$0205 /$EF /$B0/$08 /$EE /$E8/$0160
      /$B9/$FFFF /$8B/$46/$D6 /$8B/$56/$D4 /$03/$46/$F2 /$13/$56/$F0
      /$89/$46/$E2 /$89/$56/$E0 /$79/$08 /$33/$C1 /$33/$D1 /$40 /$73/$01 /$42
      /$89/$46/$D2 /$89/$56/$D0 /$8B/$46/$D6 /$8B/$56/$D4 /$2B/$46/$EE
      /$1B/$56/$EC /$89/$46/$DE /$89/$56/$DC /$79/$08 /$33/$C1 /$33/$D1 /$40
      /$73/$01 /$42 /$89/$46/$CE /$89/$56/$CC /$8B/$46/$E2 /$8B/$56/$E0
      /$2B/$46/$EE /$1B/$56/$EC /$89/$46/$DA /$89/$56/$D8 /$79/$08 /$33/$C1
      /$33/$D1 /$40 /$73/$01 /$42 /$89/$46/$CA /$89/$56/$C8 /$8B/$46/$D2
      /$8B/$56/$D0 /$3B/$56/$CC /$77/$42 /$72/$05 /$3B/$46/$CE /$73/$3B
      /$3B/$56/$C8 /$77/$36 /$72/$05 /$3B/$46/$CA /$73/$2F /$D0/$46/$FE
      /$83/$5E/$FC/$00 /$FF/$4E/$C6 /$D0/$4E/$FA /$83/$56/$F8/$00 /$FF/$46/$C4
      /$8B/$46/$E2 /$8B/$56/$E0 /$89/$46/$D6 /$89/$56/$D4 /$8B/$46/$EA
      /$8B/$56/$E8 /$01/$46/$F2 /$11/$56/$F0 /$E9/>$8D /$8B/$46/$CE
```

```
      /$8B/$56/$CC /$3B/$56/$D0 /$77/$3C /$72/$05 /$3B/$46/$D2 /$73/$35
      /$3B/$56/$C8 /$77/$30 /$72/$05 /$3B/$46/$CA /$73/$29 /$83/$46/$F6/$05
      /$FF/$46/$C2 /$83/$6E/$F4/$05 /$FF/$4E/$C0 /$8B/$46/$DE /$8B/$56/$DC
      /$89/$46/$D6 /$89/$56/$D4 /$8B/$46/$E6 /$8B/$56/$E4 /$29/$46/$EE
      /$19/$56/$EC /$EB/$47/$90 /$D0/$46/$FE /$83/$5E/$FC/$00 /$FF/$4E/$C6
      /$D0/$4E/$FA /$83/$56/$F8/$00 /$FF/$46/$C4 /$83/$46/$F6/$05 /$FF/$46/$C2
      /$83/$6E/$F4/$05 /$FF/$4E/$C0 /$8B/$46/$DA /$8B/$56/$D8 /$89/$46/$D6
      /$89/$56/$D4 /$8B/$46/$EA /$8B/$56/$E8 /$01/$46/$F2 /$11/$56/$F0
      /$8B/$46/$E6 /$8B/$56/$E4 /$29/$46/$EE /$19/$56/$EC /$E8/>$1A
      /$8B/$46/$F6 /$3B/$46/$F4 /$74/$03 /$E9/$FEAF /$B0/$FF /$EE /$4A
      /$B8/>$05 /$EF /$B8/>$03 /$EF /$E9/>$A0 /$8A/$26/GDCOLOR /$BA/$03CF
      /$8B/$5E/$C6 /$3B/$1E/GDVW_X1 /$73/$08 /$C7/$06/GDC_FLG/>$01 /$EB/$3B
      /$8A/$46/$FE /$EE /$8B/$76/$FC /$8B/$5E/$C2 /$3B/$1E/GDVW_Y1 /$73/$08
      /$C7/$06/GDC_FLG/>$01 /$EB/$09 /$8E/$46/$F6 /$26/$8A/$04 /$26/$88/$24
      /$8B/$5E/$C0 /$3B/$1E/GDVW_Y2 /$76/$08 /$C7/$06/GDC_FLG/>$01 /$EB/$09
      /$8E/$46/$F4 /$26/$8A/$04 /$26/$88/$24 /$8B/$5E/$C4 /$3B/$1E/GDVW_X2
      /$76/$08 /$C7/$06/GDC_FLG/>$01 /$EB/$3B /$8A/$46/$FA /$EE /$8B/$76/$F8
      /$8B/$5E/$C2 /$3B/$1E/GDVW_Y1 /$73/$08 /$C7/$06/GDC_FLG/>$01 /$EB/$09
      /$8E/$46/$F6 /$26/$8A/$04 /$26/$88/$24 /$8B/$5E/$C0 /$3B/$1E/GDVW_Y2
      /$76/$08 /$C7/$06/GDC_FLG/>$01 /$EB/$09 /$8E/$46/$F4 /$26/$8A/$04
      /$26/$88/$24 /$C3 /$8B/$E5);
end; {gpcir}

function GPRDDOT(X,Y : Integer): Integer;
{Get dot color}
begin
  inline
    ($8B/$46/$04 /$A3/GDCUR_Y /$D1/$E0 /$D1/$E0 /$03/$06/GDCUR_Y /$05/$A000
      /$8E/$C0 /$8B/$76/$06 /$89/$36/GDCUR_X /$8B/$CE /$D1/$EE /$D1/$EE
      /$D1/$EE /$80/$E1/$07 /$B4/$80 /$D2/$CC /$BA/$03CE /$B0/$04 /$EE /$42
      /$33/$DB /$B0/$00 /$EE /$26/$84/$24 /$74/$03 /$80/$CB/$01 /$B0/$01 /$EE
      /$26/$84/$24 /$74/$03 /$80/$CB/$02 /$B0/$02 /$EE /$26/$84/$24 /$74/$03
      /$80/$CB/$04 /$B0/$03 /$EE /$26/$84/$24 /$74/$03 /$80/$CB/$08
      /$89/$5E/$08);
end; {gprdot get pixel color}

procedure puff_plot(x,y,color : integer);
{EGA and CGA plot}
begin
  result.ah := $C;
  result.bh := 0;
  if EGA then begin
    result.al := color;
    result.dx := y;
    result.cx:=x ;
    intr($10,result);
  end else begin
    case color of
     1..127    : color:=1;
     127..255  : color:=129;
    end; {case}
    plot(x,Round(y*hir),color);
  end;
```

```
end; {puff_plot}

procedure puff_draw(x1,y1,x2,y2,color : integer);
{EGA and CGA draw}
var
 xt,i : integer;
begin
  if EGA then begin
    GDCOLOR := Color;
    GDCUR_X := X1;
    GDCUR_Y := Y1;
    gpline(x2,y2);
  end else begin
    case color of
     1..127    : color:=1;
     127..255  : color:=129;
    end; {case}
    if y1=y2 then begin
      if x1 > x2 then begin xt:=x2; x2:=x1; x1:=xt; end;
      if odd(x1) then x1:=x1+1;
      for i:=0 to (x2-x1) div 2 do puff_plot(x1+2*i,y1,color);
    end else draw(x1,Round(y1*hir),x2,Round(y2*hir),color);
  end;
end; {draw}

procedure arc(cx,cy,r,sqrr : integer);
{Draw 4 arc's for Smith chart with center (cx,cy) radius r.
 Arcs are symmetric about center of chart and must be within sqrt(sqrr) of
 Smith chart center}
label
  exit_arc;
var
  x1o,x2o,y1o,y2o,x1,x2,y1,y2,xcc : integer;
  rx,ry,pts,th : real;
  stop         : boolean;
begin
  xcc:=Round(2*centerx);
  if EGA then begin
    pts:=10.0/(2.0*r); th:=0;
    x1:=cx+r;  y1:=cy;  x2:=xcc-x1;    y2:=y1;
    repeat
      if th <> 0 then begin
        puff_draw(x1o,y1o,x1,y1,green);
        puff_draw(x2o,y1o,x2,y1,green);
        puff_draw(x1o,y2o,x1,y2,green);
        puff_draw(x2o,y2o,x2,y2,green);
      end;
      x1o:=x1;
      y1o:=y1;
      x2o:=x2;
      y2o:=y2;
      th:=th+pts;
      rx:=r*cos(th);
```

```
     ry:=r*sin(th);
     x1:=Round(cx+rx);
     y1:=Round(cy+ry*yf);
     x2:=xcc-x1;
     y2:=Round(cy-ry*yf);
     if sqrr=0 then stop:=th > pi/1.9
             else stop:=sqr(x1-centerx)+sqr(Round(ry)) > sqrr;
   until stop;
  end else begin
   cy:=Round(cy*hir);
   x1o:=cx; y1o:=cy;
   for y1:=cy to cy+Round(r*yf*hir) do begin
     ry:=(y1-cy)/(yf*hir);
     rx:=sqrt(abs(sqr(r)-sqr(ry)));
     x1:=Round(cx+rx);
     if (Trunc(sqr(x1-centerx)+sqr(ry))>sqrr)and(sqrr<>0) then goto exit_arc;
     x2:=xcc-x1;
     y2:=Round(cy-ry*yf*hir);
     if abs(4*(y1o-y1))+abs(5*(x1o-x1)) > 9 then begin
       x1o:=x1;          y1o:=y1;
       plot(x1,y1,1);  plot(x2,y1,1);
       plot(x1,y2,1);  plot(x2,y2,1)
     end;
   end;
   plot(cx,y1,1);
   plot(cx,y2,1);
  end;{if EGA}
  exit_arc:
end; {arc}

procedure circle(cx,cy,r,color : integer);
{EGA and CGA circle draw}
begin
  if EGA then begin
    GDCOLOR := Color;
    GDCUR_X := CX;
    GDCUR_Y := CY;
    gpcir(r);
  end else arc(cx,cy,r,0);
end; {circle}

procedure draw_box(xs,ys,xm,ym,color : integer);
begin
  puff_draw(xs,ys,xm,ys,color);
  puff_draw(xm,ys,xm,ym,color);
  puff_draw(xm,ym,xs,ym,color);
  puff_draw(xs,ym,xs,ys,color);
end; {draw_box}

procedure fill_box(x1,y1,x2,y2,color : integer);
const
  e=$B800;  o=$BA00;
var
```

```
  xt,a1,a2,i,r8 : integer;
  byt1e,byt2e,byt3e,bytce,fille,byt1o,byt2o,byt3o,bytco,fillo : byte;
begin
  if x1 > x2 then begin xt:=x1;x1:=x2;x2:=xt;end;
  if y1 > y2 then begin xt:=y1;y1:=y2;y2:=xt;end;
  if EGA then begin
    if (x2-x1) < 7 then  for xt:=x1 to x2 do puff_draw(xt,y1,xt,y2,color)
    else begin
      GDCOLOR := Color;
      gpbox1(x1,y1,x2,y2);
    end;
  end else begin
    if odd(x1) and (color<> black) then begin
     x1:=x1+1;
     x2:=x2+1;
    end;
    y2:=Round((y2-y1)*hir);
    y1:=Round(y1*hir);
    y2:=y2+y1;
    {writeln(lst,y1:6,y2:6,y1*hir:6:2,y2*hir:6:2);}
    a1 := x1 div 8;
    a2 := x2 div 8;
    if color=black then begin
      byt1e:=lo(not($ff shr (x1-a1*8)));
      byt2e:=lo(not($ff shl (8*(a2+1)-x2)));
      byt3e:=lo(byt1e or byt2e);
      bytce:=0; bytco:=0;
    end else begin
      if  color = lightgray then begin
       fille:=$88;fillo:=$22;
      end else begin
       fille:=$aa;fillo:=$aa;
      end;
      byt1e := lo(fille and ($ff shr (x1-a1*8)));
      byt2e := lo(fille and ($ff shl (8*(a2+1)-x2-1)));
      byt3e := lo(byt1e and byt2e);
      bytce := fille ;
      byt1o := lo(fillo and ($ff shr (x1-a1*8)));
      byt2o := lo(fillo and ($ff shl (8*(a2+1)-x2-1)));
      byt3o := lo(byt1o and byt2o);
      bytco := fillo ;
    end;
    if a2 > a1 then for i:=y1 to y2 do begin
      r8:=80*(i div 2);
      if odd(i) then begin
        if color=black then begin
          mem[o:a1+r8]:=mem[o:a1+r8] and byt1e;
          mem[o:a2+r8]:=mem[o:a2+r8] and byt2e
        end else begin
          mem[o:a1+r8]:=mem[o:a1+r8] or byt1o;
          mem[o:a2+r8]:=mem[o:a2+r8] or byt2o
        end;
        fillchar(mem[o:a1+r8+1],a2-a1-1,bytco);
```

```
      end else begin
        if color=black then begin
          mem[e:a1+r8]:=mem[e:a1+r8] and byt1e;
          mem[e:a2+r8]:=mem[e:a2+r8] and byt2e
        end else begin
          mem[e:a1+r8]:=mem[e:a1+r8] or byt1e;
          mem[e:a2+r8]:=mem[e:a2+r8] or byt2e
        end;
        fillchar(mem[e:a1+r8+1],a2-a1-1,bytce);
      end;{if odd}
    end
    else for i:=y1 to y2 do begin
      r8:=80*(i div 2);
      if odd(i) then begin
       if color=black then mem[o:a1+r8]:=mem[o:a1+r8] and byt3e
                      else mem[o:a1+r8]:=mem[o:a1+r8] or  byt3o;
      end else begin
       if color=black then mem[e:a1+r8]:=mem[e:a1+r8] and byt3e
                      else mem[e:a1+r8]:=mem[e:a1+r8] or  byt3e;
      end;{if odd}
    end;
  end{if EGA}
end; {fill_box}

procedure pattern(x1,y1,ij,pij : integer);
{Draw marker pattern for s-parameter plots
 ij=1 box, ij=2 X, ij=3 diamond, ij=4 +}
var
 color,j : integer;
begin
  if EGA then begin
    color:=s_color[ij];
    case ij of
      1 : draw_box(x1-3,y1-3,x1+3,y1+3,color);
      2 : begin
            puff_draw(x1-3,y1-3,x1+3,y1+3,color);
            puff_draw(x1-3,y1+3,x1+3,y1-3,color);
          end;
      3 : begin
            puff_draw(x1-4,y1,x1,y1+4,color);
            puff_draw(x1-3,y1-1,x1,y1-4,color);
            puff_draw(x1+4,y1,x1+1,y1+3,color);
            puff_draw(x1+3,y1-1,x1+1,y1-3,color);
          end;
      4 : begin
            puff_draw(x1-4,y1  ,x1+4,y1  ,color);
            puff_draw(x1  ,y1+4,x1  ,y1-4,color);
          end;
    end;{case}
  end else begin
    if pij=128 then color:=129 else color:=1;
    y1:=Round(y1*hir);
    case ij of
```

```
    1: begin
        draw(x1-5,y1-2,x1-5,y1+3,color);
        draw(x1+5,y1-2,x1+5,y1+3,color);
        for j:=-2 to 2 do begin
           plot(x1-2*j,y1-2,color);
           plot(x1-2*j,y1+3,color);
        end;
      end;
    2: begin
        draw(x1-5,y1-3,x1+5,y1+3,color);
        draw(x1-5,y1+3,x1+5,y1-3,color);
      end;
    3: begin
         draw(x1-5,y1+1,x1,y1+3,color);
         draw(x1+5,y1-1,x1,y1-3,color);
         draw(x1-7,y1,x1,y1-3,color);
         draw(x1+7,y1,x1,y1+3,color);
         plot(x1,y1-3,color);plot(x1,y1+3,color);
       end;
    4: begin
        for j:=1 to 3 do begin
          plot(x1-2*j,y1,color);
          plot(x1+2*j,y1,color);
        end;
        draw(x1,y1+3,x1,y1-3,color);
      end;
    end;{case}
  end;
end; {pattern}

procedure box(x,y,ij : integer);
{Draw small box to indicate where s-parameters are calculated}
begin
  if EGA then begin
    case ij of
      2 : begin y:=y-1;x:=x-1;end;
      3 : x:=x-1;
      4 : y:=y-1;
    end;
    puff_plot(x,y,s_color[ij]);
    puff_plot(x+1,y,s_color[ij]);
    puff_plot(x,y+1,s_color[ij]);
    puff_plot(x+1,y+1,s_color[ij]);
  end else begin
    y:=Round(hir*y);
    plot(x,y,1);
    plot(x,y-1,1);
  end;
end; {box}

procedure neg_box(x1,y1,x2 : integer);
{CGA routine for reverse video of graphics}
const
```

```
  e=$B800;  o=$BA00;
var
  i,j,r8 : integer;
begin
  for i:=0 to 3 do
  for j:=x1 to x1+x2-1 do begin
    r8:=80*((y1-1)*4 +i)+j-1;
    mem[o:r8]:=lo(not(mem[o:r8]));
    mem[e:r8]:=lo(not(mem[e:r8]));
  end;
end; {neg_box}

procedure wso(var marker; i : integer);
{Debug routine for listing segment and offset address of a pointer}
begin
 if i=1 then write(lst,' ',seg(marker),':',ofs(marker))
        else writeln(lst,' ',seg(marker),':',ofs(marker));
end;

procedure write_compt(color : integer; tcompt : compt);
{Display a component -- highlighted (CGA reverse video)}
begin
  Textcolor(color);
  with tcompt^ do begin
    gotoxy(xp,yp);write(descript);
    if (color=white) and not(EGA) then neg_box(xp,yp,length(descript));
  end;{with}
end; {write_compt}

procedure write_comptm(m,color : integer; tcompt : compt);
{Display the first m characters of a component}
var
  i : integer;
begin
  Textcolor(color);
  with tcompt^ do begin
    gotoxy(xp,yp);for i:=1 to m do write(descript[i]);
  end;{with}
end; {write_comptm}

procedure beep;
begin
  sound(250);  delay(50);  Nosound;
end;

procedure rcdelay(j : integer);
{Interuptable delay. Used in demo mode}
var
 i : integer;
begin
  for i:=1 to j do begin
    if keypressed then begin
      read(kbd,chs);
```

```
      if chs='S' then begin
         beep;
         textmode(bw80);
         halt(1);
      end else delay(2000);
    end else delay(10);{if keypressed}
  end;{for i}
end; {rcdelay}

procedure write_message;
{Write message in center box}
begin
  textcolor(message_color);
  gotoxy(32+(17-length(message[1])) div 2,11);write(message[1]);
  gotoxy(32+(17-length(message[2])) div 2,12);write(message[2]);
  gotoxy(32+(17-length(message[3])) div 2,13);write(message[3]);
  if (message[1]+message[2]+message[3] <> '') and read_kbd then beep;
  if demo_mode then rcdelay(50);
end; {write_message}

procedure erase_message;
{Erase message in center box}
begin
  gotoxy(32,11);write('                 ');
  gotoxy(32,12);write('                 ');
  gotoxy(32,13);write('                 ');
  message[1]:='';  message[2]:='';  message[3]:='';
end; {erase_message}

procedure write_error(time : real);
begin
  write_message;
  delay(Round(1000*time));
  erase_message;
end; {write_error}

function input_string(mess : line_string) : file_string;
{Prompt user to input string (filename)}
var
  answer : file_string;
begin
  erase_message;
  Textcolor(LightCyan);
  gotoxy(32,12);write(mess);
  gotoxy(32,13);write('?');
  gotoxy(33,13);readln(answer);
  input_string:=answer;
  erase_message;
end; {input_string}

function printer_offline : boolean;
{Check printer is online}
var
```

```
  i : integer;
begin
  i:=0;
  repeat
    result.dx := 0; result.ax := 512; intr($17,result);
    if result.ah <> 144 then begin
      message[2]:=' Printer Offline ';
      write_message;
      delay(1000);
      i := i + 1;
    end;
  until (result.ah =144) or (i = 10);
  if i=10 then begin
    printer_offline:=true;
    message[2]:='Abandoning Print ';{printer is offline}
    write_error(1);
  end else begin
    printer_offline:=false;
    message[2]:='   Printer OK    ';
    write_error(0.25);
  end;{if i=10}
end; {printer offline}

procedure reset_printer;
begin
  result.ah:=1;result.al:=0;result.dx:=0;
  intr($17,result);
end;

procedure dirn_xy;
{Maps dirn into x and y changes}
begin
  xii:=0;yii:=0;
  case dirn of
    1 : xii:= 1; {East}
    2 : xii:=-1; {West}
    3 : yii:= 1; {South}
    0 : yii:=-1; {North}
  end;
end;{drin_xy}

procedure increment_pos(i : integer);
{Increment postion of cursor on circuit}
begin
  dirn_xy;
  if odd(i) then begin
    if i=-1 then begin     {step 1/2 of compt}
      xm:=xm+(compt1^.lngth*xii+yii*compt1^.con_space)/2.0;
      ym:=ym+(compt1^.lngth*yii+xii*compt1^.con_space)/2.0;
    end else begin
      xm:=xm+lengthxm*xii;
      ym:=ym+lengthym*yii;
    end;
```

```
    end else begin
      if i=0 then begin
        xm:=cnet^.xr;
        ym:=cnet^.yr;
      end else begin
        xm:=xm+lengthxm*xii-compt1^.con_space*yii;
        ym:=ym+lengthym*yii-compt1^.con_space*xii;
      end;{if i}
    end;{if odd}
    xi:=Round(xm/csx);
    yi:=Round(ym/csy);
    dirn:=lo(not(dirn+252));
end; {increment pos}

procedure lengthxy(tnet : net);
{Convert part lengths and widths to increments in the x and y directions}
var
  lengths,widths : real;
begin
  dirn_xy;
  if tnet <> nil then begin
    lengths:=tnet^.com^.lngth;
    widths:=tnet^.com^.width;
  end else writeln(lst,'error');
  lengthxm:=lengths*abs(xii)+widths*abs(yii);
  lengthym:=lengths*abs(yii)+widths*abs(xii);
end; {lengthxy}

function betweenr(x1,x2,x3,sigma : real) : boolean;
{True if real x2 is between x1-sigma and x3+sigma}
var
  xt : real;
begin
  if x1 > x3 then begin
    xt:=x1;x1:=x3;x3:=xt;
  end;
  if (x1-sigma<= x2 ) and (x2 <= x3+sigma) then betweenr:=true
                                            else betweenr:=false;
end;

function betweeni(x1,x2,x3 : integer) : boolean;
begin
  if (x1 <= x2 ) and (x2 <= x3) then betweeni:=true else betweeni:=false;
end;

function ext_port(tcon : conn) : boolean;
{Check to see of a connector is connected to an external port}
begin
  ext_port:=betweeni(1,tcon^.port_type,min_ports);
end;

procedure write_gchar(c : char; x,y : integer);
{EGA/CGA procedure for plotting the characters [a..i] on the circuit parts}
```

```
var
 i,j,k,mask : integer;
begin
  if not(EGA) and not(odd(x)) then x:=x+1;
  if betweeni(xmin[2]+4,x,xmax[2]-4)and betweeni(ymin+4,y,ymax-4)then begin
    k:=ord(c)-96;
    for j:=1 to 7 do begin
     mask:=32;
     for i:=1 to 5 do begin
      if (gchar[k,j] and mask) > 0 then
      if EGA then puff_plot(x+i-3,y+j-4,red)
                 else plot(x+i-3,Round(hir*y)+j-4,1);
      mask:=mask shr 1;
     end;
    end;
    for j:=1 to 9 do begin
     mask:=64;
     for i:=1 to 7 do begin
      if (hchar[k,j] and mask) > 0 then
      if EGA then puff_plot(x+i-4,y+j-5,0)else plot(x+i-4,Round(hir*y)+j-5,0);
      mask:=mask shr 1;
     end;
    end;
  end;{between}
end; {write_gchar}
```

```
{Include file #1: INC1E3.PAS}

{START COMPLEX}
function prp(vu,vX,vY : complex) :  complex; external 'PUFFASM.COM';
procedure supr(vu,vX,vY : complex); external Prp[3];

{prp and supr are assembler routines.
See routines below for Pascal equivalents.}

(*
function prp(vu,vX,vY : complex) :  complex;
begin
  vu^.r:=vX^.r*vY^.r-vX^.i*vY^.i;
  vu^.i:=vX^.r*vY^.i+vX^.i*vY^.r;
  prp:=vu;
end;

procedure supr(vu,vX,vY : complex);
begin
  vu^.r:=vu^.r+vX^.r*vY^.r-vX^.i*vY^.i;
  vu^.i:=vu^.i+vX^.r*vY^.i+vX^.i*vY^.r;
end;
*)

function co(s,t : real) : complex;
var
  u : complex;
begin
  new(u);
  u^.r:=s;
  u^.i:=t;
  co:=u;
end; {Makes a complex number}

function di(s,t : complex) : complex;
var
  u : complex;
begin
  new(u);
  u^.r:=s^.r - t^.r;
  u^.i:=s^.i - t^.i;
  di := u;
end; {s - t}

function su(s,t : complex) : complex;
var
  u : complex;
begin
  new(u);
  u^.r := s^.r + t^.r;
  u^.i := s^.i + t^.i;
  su := u;
end; {s+t}
```

```
function rc(z : complex) : complex;
var
  u   : complex;
  mag : real;
begin
  new(u);
  mag:=sqr(z^.r)+sqr(z^.i);
  u^.r := z^.r/mag;
  u^.i :=-z^.i/mag ;
  rc := u;
end; {1/z}

function sm(s : real; t : complex) : complex;
var
  u : complex;
begin
  new(u);
  u^.r:=s * t^.r;
  u^.i:=s * t^.i;
  sm:=u;
end; {s*t}

{START PARSING}
function goto_numeral(n : integer; x : line_string) : integer;
{Find location of nth number in x}
var
  long,i,j : integer;
  found    : boolean;
begin
  i:=0;
  found:=false;
  goto_numeral:=0;
  j:=1;
  long:=length(x);
  if long > 0 then
  repeat
    if x[j]='(' then j:=Pos(')',x);
    if x[j] in ['0'..'9','.','-'] then begin
      i:=i+1;
      if i=n then found:=true
      else repeat{step over number}
             j:=j+1;
           until not(x[j] in ['0'..'9','.','-']) or (j=long+1);
    end else j:=j+1;
  until found or (j=long+1);
  if found then goto_numeral:=j
           else begin
             bad_compt:=true;
             message[2]:='Number is missing';
           end;
end; {goto_numeral}
```

```
function get_correction(tcompt : compt) : line_string;
{Get correction in () in tcompt}
var
  i1,i2 : integer;
begin
  get_correction:='';
  i1:=Pos('(',tcompt^.descript);
  i2:=Pos(')',tcompt^.descript);
  if (i1=0) and (i2=0) then exit;
  if (i2 < i1) or (i1 = 0) then begin
    ccompt:=tcompt;
    bad_compt:=true;
    message[1]:='Unbalanced';
    message[2]:='brackets';
    exit;
  end;
  get_correction:=Copy(tcompt^.descript,i1+1,i2-i1-1);
end; {get_correction}

function get_real(tcompt : compt; n : integer) : real;
{Read nth number in tcompt}
var
  c_string,s_value : line_string;
  j,code,long : integer;
  value       : real;
  found       : boolean;
begin
  c_string:=tcompt^.descript;
  j:=goto_numeral(n,c_string);
  if bad_compt then begin
     ccompt:=tcompt;
     exit;
  end;
  s_value:='';
  long:=length(c_string);
  found:=false;
  repeat
     if c_string[j] in ['0'..'9','.',',','-'] then begin
        if c_string[j]=',' then s_value:=s_value+'.'
                           else s_value:=s_value+c_string[j];
     j:=j+1
     end else found:=true;
  until (found or (j=long+1));
  Val(s_value,value,code);
  if (code<>0) or (length(s_value)=0) then begin
    ccompt:=tcompt;
    bad_compt:=true;
    message[2]:='Invalid number';
    exit;
  end;
  get_real:=value;
end; {get_real}
```

```
procedure get_param(tcompt : compt;n : integer;var value : real;var u1 :char);
{Get nth parameter in tcompt}
var
  c_string,s_value : line_string;
  j,code,long : integer;
  found_value : boolean;
begin
  c_string:=tcompt^.descript;
  j:=goto_numeral(n,c_string);
  if bad_compt then begin
     ccompt:=tcompt;
     exit;
  end;
  long:=length(c_string);
  if j > 0 then begin
    found_value:=false;   s_value:='';
    repeat
     if c_string[j] in ['0'..'9','.',',','-'] then begin
        if c_string[j]=',' then s_value:=s_value+'.'
                           else s_value:=s_value+c_string[j];
        j:=j+1
     end else found_value:=true;
    until (found_value or (j=long+1));
    Val(s_value,value,code);
    if (code<>0) or (length(s_value)=0) then begin
      ccompt:=tcompt;
      bad_compt:=true;
      message[2]:='Invalid number';
      exit;
    end;
  end;
  if j  <=long then u1:=c_string[j  ] else u1:='?';
end; {get_param}

procedure get_lumped_params(tcompt : compt;var v1,v2,v3,v4 : real;var u:char);
{Get paramters for a lumped element}
label
  exit_get_lumped;
var
  c_string,s_value   : line_string;
  i,j,code,long,sign : integer;
  value : real;
  found : boolean;
  ident : char;
begin
  v1:=0;v2:=0;v3:=0;v4:=0;u:='?';
  c_string:=tcompt^.descript;
  j:=1;
  if bad_compt then exit;
  long:=length(c_string);
  for i:=1 to 4 do begin
    s_value:='';
    found:=false;
```

```
    if j > long then goto exit_get_lumped;
    while not(c_string[j] in ['0'..'9','.', ',' ,'j','-','+']) do begin
     j:=j+1;
     if j > long then goto exit_get_lumped;
    end;
    if c_string[j]='+' then begin
      j:=j+1;
    end;
    if c_string[j]='-' then begin
      j:=j+1;
      sign:=-1
    end else sign:=1;
    if c_string[j]='j' then begin
     j:=j+1;
     ident:='j'
    end else ident:=' ';
    repeat
     if c_string[j] in ['0'..'9','.',','] then begin
        if c_string[j]=',' then s_value:=s_value+'.'
                           else s_value:=s_value+c_string[j];
        j:=j+1
      end else found:=true;
    until (found or (j=long+1));
    if (c_string[j] in ['j','m','M']) and (j<>long+1) then begin
      ident:=c_string[j];
      j:=j+1;
    end;
    if j<=long then if c_string[j]in['y','Y','z','Z','s','S',Omega] then begin
      u:=c_string[j];
      j:=j+1;
    end;
    {writeln(j:4,' *',s_value,'*',' ',ident:2,c_string[j]:2);}
    if (ident='j') and (length(s_value)=0) then s_value:='1';
    Val(s_value,value,code);
    if (code<>0) or (length(s_value)=0) then begin
      ccompt:=tcompt;
      bad_compt:=true;
      message[2]:='Invalid number';
      exit;
    end;
    value:=value*sign;
    case ident of
     'j'     : if value > 0 then v2:=value else v3:=value;
     'm','M' : v4:=value;
      else v1:=value;
    end;{case}
  end;{for j}
  exit_get_lumped:
  if u='?' then begin
    bad_compt:=true;
    message[1]:='Missing lumped';
    message[2]:='unit. Use y,';
    message[3]:='z, S, '+Omega;
```

```
  end;
end; {get_lumped_params}

procedure get_device_params(tcompt:compt;var fname:file_string;var long:real);
{Get paramneters for device}
label
  exit_gdp1,exit_gdp2;
var
  p1,p2,code,i,len  : integer;
  s_value  : line_string;
begin
  fname:=tcompt^.descript;
  for i:=1 to 2 do begin
    p1:=Pos(' ',fname);
    Delete(fname,1,p1);
  end;
  p1:=Pos(' ',fname);
  p2:=length(fname);
  s_value:=fname;
  if p1*p2= 0 then begin
    ccompt:=tcompt;
    bad_compt:=true;
    message[1]:='Invalid device';
    message[2]:='file name or';
    message[3]:='missing length';
    exit;
  end else Delete(fname,p1,p2);
  Delete(s_value,1,p1);
  if (Pos('mm',s_value)+Pos('MM',s_value)) = 0 then begin
    ccompt:=tcompt;
    bad_compt:=true;
    message[1]:='device length';
    message[2]:='must be in mm';
    exit;
  end else Delete(fname,p1,p2);
  if length(s_value) > 0 then
  repeat
    if not(s_value[1] in ['0'..'9','.',',','-']) then Delete(s_value,1,1)
                                                  else goto exit_gdp1;
  until length(s_value)=0;
  exit_gdp1:
  len:=length(s_value);
  if len > 0 then
  repeat
    if not(s_value[len] in ['0'..'9','.',',','-']) then Delete(s_value,len,1)
                                                    else goto exit_gdp2;
    len:=length(s_value);
  until length(s_value)=0;
  exit_gdp2:
  for i:=1 to length(s_value) do if s_value[i]=',' then s_value[i]:='.';
  Val(s_value,long,code);
  if (code<>0) or (length(s_value)=0) then begin
    ccompt:=tcompt;
```

```
    bad_compt:=true;
    message[1]:='Invalid length';
    message[2]:='or filename';
    exit;
  end;
end; {get_device_params}

{START COMPONENT FUNCTIONS}

function widtht(zed : real) : real;
{Microstrip models are from Gupta, Garg, and Chadha:CAD of Microwave Circuits
Artech House, 1981, pp. 60-62: Eqs. 3.53a and 3.53b, Widths in mils}
var
  A,B : real;
begin
  A := (zed/60) * sqrt((er+1)/2) + (er-1)/(er+1)*(0.23+0.11/er);
  if A > 1.52 then
    if 2*A > 300 then begin
      bad_compt:=true;
      message[1]:='Impedance';
      message[2]:='too large';
      widtht:=0;
    end else widtht:= substrate_t*8*exp(A)/(exp(2*A)-2)+artwork_cor
  else begin
    B := 60*sqr(pi)/zed/sqrt(er);
  widtht:=substrate_t*2/pi*(B-1-ln(2*B-1)+(er-1)/2/er*(ln(B-1)+0.39-0.61/er))
           +artwork_cor;
  end;
end; {widtht}

function cosh(x : real) : real;
var
  exp1 : real;
begin
  if x > 300 then begin
    exp1:=infty;
    bad_compt:=true;
    message[1]:='cline impedances';
    message[2]:='can'+char(39)+'t be realized';
    message[3]:='in microstrip';
  end else begin
    exp1:=exp(x);
    cosh:=(exp1+1/exp1)/2;
  end;
end; {cosh}

function arccosh(x : real) : real;
{Inverse cosh}
var
  sqx : real;
begin
  sqx:=sqr(x);
  if sqx <= 1 then begin
```

```
    arccosh:=0;
    bad_compt:=true;
    message[1]:='cline impedances';
    message[2]:='can'+char(39)+'t be realized';
    message[3]:='in microstrip';
  end else arccosh:=ln(x+sqrt(sqr(x)-1));
end; {arc_cosh}

procedure error(g,wo,ceven : real; var fg,dfg : real);
{Used to calculate cline dimensions. When error=0 a consistent
solution of the cline equations has been reached. Edwards p139}
var
 hh,sqm1,rsqm1,dcdg,acoshh,acoshg,u1,u2,du1dg,du2dg,dhdg,
 dcdu1,dcdu2,dcdh : real;
begin
  hh:=0.5*((g+1)*ceven+g-1);
  dhdg:=0.5*(ceven+1.0);
  acoshh:=arccosh(hh); if bad_compt then exit;
  acoshg:=arccosh(g);  if bad_compt then exit;

  sqm1:=sqr(hh)-1;
  rsqm1:=sqrt(sqm1);
  dcdh:=(rsqm1+hh)/(hh*rsqm1+sqm1);

  sqm1:=sqr(g)-1;
  rsqm1:=sqrt(sqm1);
  dcdg:=(rsqm1+g)/(g*rsqm1+sqm1);

  u1:=((g+1)*ceven-2)/(g-1);
  du1dg:=((g-1)*ceven-((g+1)*ceven-2))/sqr(g-1);
  u2:=acoshh/acoshg;
  du2dg:=(acoshg*dcdh*dhdg-acoshh*dcdg)/sqr(acoshg);

  sqm1:=sqr(u1)-1;
  rsqm1:=sqrt(sqm1);
  dcdu1:=(rsqm1+u1)/(u1*rsqm1+sqm1);

  sqm1:=sqr(u2)-1;
  rsqm1:=sqrt(sqm1);
  dcdu2:=(rsqm1+u2)/(u2*rsqm1+sqm1);

  fg:=(2*arccosh(u1)+arccosh(u2))/pi-wo; if bad_compt then exit;
  dfg:=(2*dcdu1*du1dg+dcdu2*du2dg)/pi;
end; {error}

procedure width_spacing_coupler(we,wo : real; var widthc,spacing : real);
{This routine uses Netwons method to find the cline width and spacing}
const
  tol=0.0001;
var
  woh,soh,codd,ceven,g,fg,dfg,dg,g1 : real;
  i : integer;
begin
```

```
  ceven:=cosh(pi*we/2);
  codd :=cosh(pi*wo/2);
  soh:=2*arccosh((ceven+codd-2)/(codd-ceven))/pi; if bad_compt then exit;
  g:=cosh(pi*soh/2); {starting guess}
  i:=0;
  repeat{newton algorithm}
    g1:=g;
    error(g,wo,ceven,fg,dfg);if bad_compt then exit;
    dg:=fg/dfg;
    g:=g1-dg;
    i:=i+1;
    if g<= 1.0 then i:=101;
  until ((abs(dg)<abs(tol*g)) or (i > 100));
  if i> 100 then begin
    bad_compt:=true;
    message[1]:='cline impedances';
    message[2]:='can'+char(39)+'t be realized';
    message[3]:='in microstrip';
    exit;
  end;
  soh:=2.0*arccosh(g)/pi;
  woh:=arccosh(0.5*((g+1)*ceven+g-1))/pi-soh/2.0;
  widthc:=woh*substrate_t;
  spacing:=(woh+soh)*substrate_t;
end; {width_spacing_coupler}

procedure shutdown;
{Called when a disastrous error condtion has been reached to stop Puff}
begin
  message[1]:='FATAL ERROR:';
  write_message;
  halt(1);
end;

function fileexists(note:boolean;var inf:textfile;fname:file_string): boolean;
{Performs an Assign and Reset on textfile if the file exists}
begin
  fileexists:=False;
  if fname <> '' then begin
    assign(inf,fname);
    {$I-} reset(inf); {$I+}
    IF IOResult=0 then fileexists:=true
    else begin
      if note then begin
        message[2]:='File not found';
        message[3]:=fname;
        write_message;    delay(1000);
      end;
    end; {if IO}
  end; {if fname}
end; {fileexists}

procedure add_coord(x1,xb,xl,y1 : integer;just : boolean;tdes : line_string);
```

```
{Add a record to the list of paramters used in the plot window}
begin
  if ccompt=nil then ccompt:=coord_start else ccompt:=ccompt^.next_compt;
  with ccompt^ do begin
    xp:=x1; xorig:=x1; xmaxl:=xl; x_block:=xb; yp:=y1;
    right:=just;
    descript:=tdes;
  end;
end; {add_coord}

function atan2(x,y : real) : real;
{Modified arctan to get phase in all quadrants and avoid blow ups when x=0}
begin
  if x=0 then begin
     if y > 0 then atan2:=0 else atan2:=180;
  end else begin
     if x > 0 then atan2:=180*arctan(y/x)/pi
     else begin
     if y > 0 then atan2:=180*arctan(y/x)/pi+180
              else atan2:=180*arctan(y/x)/pi-180;
     end;
  end;
end; {atan2}

function enough_space(defaultdrive : integer) : boolean;
{Look for space on disk -- a: 0, b: 1 ..}
var
  tracks,bytes,sectors,ij : integer;
  totalfreebytes,space_needed : real;
begin
  if defaultdrive < 0 then begin
    result.AX := $1900;                    { Get current Drive number }
    MSDos( result );
    DefaultDrive := (result.AX and $FF) + 1;
  end;

  result.AX := $3600;             { Get Disk free space }
  result.DX := Defaultdrive;
  MSDos( result );
  Tracks := result.BX;
  Bytes := result.CX;
  Sectors := result.AX;
  Totalfreebytes := (( Sectors * Bytes * 1.0 ) * Tracks );
  space_needed:=10240.0;
  if filled_OK then begin
    space_needed:=space_needed+11.0*npts;
    for ij:=1 to max_params do if s_param_table[ij]^.changed then
      space_needed:=space_needed+20.0*npts;
  end;
  if space_needed < totalfreebytes then enough_space:=true
  else begin
    enough_space:=false;
    message[2]:='Disk too full';
```

```
    write_message;
  end;
end; {enough_space}

function node_number : integer;
{Find number of node in linked list of nets}
var
  nn   : integer;
  tnet : net;
begin
  tnet:=nil; nn:=0;
  if net_start <> nil then
  repeat
    if tnet=nil then tnet:=net_start else tnet:=tnet^.next_net;
    if tnet^.node then nn:=nn+1;
  until (tnet^.next_net=nil) or (cnet=tnet);
  if cnet=tnet then node_number:=nn else node_number:=0;
end; {node_number}

procedure update_key_list(nn : integer);
{Update array which contains keystrokes used to layout circuit}
begin
  if key_end=0 then key_end:=1 else key_end:=key_end+1;
  if key_end=key_max then begin
    key_end:=key_max-1;
    message[1]:='Circuit is too';
    message[2]:='complex for';
    message[3]:='redraw';
    write_message;
  end;
  key_list[key_end].key1:=key;
  key_list[key_end].noden:=nn;
end; {update_key_list}

procedure move_cursor(x1,y1 : integer);
{Move character cursor}
var
  long,i : integer;
begin
  if x1 <> 0 then with ccompt^ do begin
    long:=length(descript);
    if cx+x1 <= long then begin
      cx:=cx+x1;
      if cx < x_block then cx:=x_block;
      if right and (cx+xp >= xorig) then begin
        xp:=xp-1;
        write_compt(lightgray,ccompt);write(' ');
      end;
    end;
  end else begin
    erase_message;
    if (window_number=2) and (length(ccompt^.descript)=1)then
    for i:=1 to max_params do if s_param_table[i]=ccompt then begin
```

```
      gotoxy(ccompt^.xp-2,ccompt^.yp);write('   ');
    end;
    if y1=-1 then begin
      if ccompt^.prev_compt = nil then beep else ccompt:=ccompt^.prev_compt;
    end else begin
      if ccompt^.next_compt = nil then beep else ccompt:=ccompt^.next_compt;
    end; {y1=-1}
    if (window_number=2) and (length(ccompt^.descript)=1)then
    for i:=1 to max_params do if s_param_table[i]=ccompt then begin
      pattern(38*charx-1,(20+i)*chary-8,i,0);
      write_comptm(1,lightgray,ccompt);
    end;
    long:=length(ccompt^.descript);
    if cx > long then cx:=long;
    if window_number=2 then cx:=ccompt^.x_block;
    if cx < ccompt^.x_block then cx:=ccompt^.x_block;
  end;
end; {move_cursor}

{START OVERLAY}
overlay procedure update_command0;
begin
  textcolor(white);
  gotoxy(46,5); if update_flag then writeln('ON ') else writeln('OFF');
end;

overlay procedure snap0;
{Move cursor X to nearest node}
var
  distance,distancet : real;
  tnet,pnet : net;
  found      : boolean;
  i          : integer;
begin
  tnet:=nil;found:=false;
  if net_start <> nil then
  if read_kbd then begin
    distance:=1.0e10;
    repeat
      if tnet = nil then tnet:=net_start else tnet:=tnet^.next_net;
      if tnet^.node then begin
        distancet:=sqrt(sqr(tnet^.xr-xm)+sqr(tnet^.yr-ym));
        if betweenr(0,distancet,distance,-resln/2.0) then begin
          distance:=distancet;
          found:=true;
          pnet:=tnet;
        end;
      end;
    until tnet^.next_net=nil;
  end else begin
    i:=0;
    repeat
      if tnet = nil then tnet:=net_start else tnet:=tnet^.next_net;
```

```
      if tnet^.node then begin
        i:=i+1;
        if i=key_list[key_i].noden then begin
          found:=true;
          pnet:=tnet;
        end;
      end;
    until (tnet^.next_net=nil) or found;
  end;{if read_kbd}
  if found then begin
    cnet:=pnet;
    increment_pos(0);
  end;
end; {snap0}

overlay procedure dx_dy0;
{Display change in x an y in mm when cursor X moves}
var
  dx,dy : real;
begin
  dx:=(xm-xrold);  dy:=(ym-yrold);
  xrold:=xm;       yrold:=ym;
  if read_kbd then begin
    Textcolor(lightgray);
    gotoxy(35,12);
    if abs(dx) > resln then begin
      write(delta,'x', dx:7:2,'mm');
      gotoxy(35,13);
    end else gotoxy(35,12);
    if abs(dy) > resln then write(delta,'y',-dy:7:2,'mm');
  end;
end;

overlay function get_lead_char0(tcompt : compt) : char;
{Get lead character to determine if part is tline, clines, device or lumped}
label
  exit_label;
var
  xstr  : line_string;
  char1 : char;
begin
  xstr:=tcompt^.descript;
  Delete(xstr,1,1);
  char1:=xstr[1];
  repeat
    if xstr[1]=' ' then  Delete(xstr,1,1);
    if (length(xstr)=0) then goto exit_label;
    char1:=xstr[1];
  until (xstr[1] <> ' ');
  exit_label:
  if char1 in ['A'..'Z'] then char1:=char(ord(char1)+32);
  get_lead_char0:=char1;
end; {get_lead_char0}
```

```

overlay procedure make_coord_and_parts_list0;
{Set up linked list of components for all parameters in the Plot window}
var
  tcompt : compt;
  i      : integer;
begin
  for i:=1 to 10 do
  if i=1 then begin
    New(coord_start);
    tcompt:=coord_start;
  end else begin
    New(tcompt^.next_compt);
    tcompt^.next_compt^.prev_compt:=tcompt;
    tcompt:=tcompt^.next_compt;
  end;
  tcompt^.next_compt:=coord_start;
  coord_start^.prev_compt:=tcompt;
  for i:=1 to 9 do begin
    if i=1 then begin
       New(part_start);
       tcompt:=part_start;
       tcompt^.prev_compt:=nil;
       tcompt^.yp:=ybn;
    end else begin
       New(tcompt^.next_compt);
       tcompt^.next_compt^.prev_compt:=tcompt;
       tcompt:=tcompt^.next_compt;
       tcompt^.yp:=tcompt^.prev_compt^.yp+1;
    end;
    with tcompt^ do begin
      descript:=char(ord('a')+i-1)+' ';
      changed:=false;  right:=false;  parsed:=false;
      f_file:=nil;     used:=0;
      xp:=xbn;         x_block:=2;  xmaxl:=des_len-1;
    end;{with}
  end;{i}
  tcompt^.next_compt:=nil;
end; {make_coord_and_parts_lists0}

overlay function commandLine0 : file_string;
{Get information which might follow initial PUFF command eg. c> puff lowpass}
label
  exit_cmd;
var
  Buffer : file_string;
  CL     : file_string absolute cseg:$80;
begin
  Buffer := CL;
  if length(buffer) > 0 then
  repeat
    if buffer[1] = ' ' then Delete(buffer,1,1) else goto exit_cmd;
  until length(buffer)=0;
```

```
  exit_cmd:
  if Pos('-D',buffer) > 0 then begin
    buffer:='';   demo_mode:=true;
  end else demo_mode:=false;
  if buffer = '' then commandline0:='setup' else commandline0:=buffer
end; {commandline0}

overlay procedure tline0(tcompt : compt);
{If action is true then get tline parameters
 is action is false calculate tline s-parameters}
var
  i,j   : integer;
  rds   : complex;
  c_ss  : array[1..2] of array[1..2] of complex;
  u     : complex;
  unit1 : char;
  zd,elength,sh,ch,ere,gamma,value : real;
begin
  if action then begin
  with tcompt^ do begin
    correction_compt^.descript:=get_correction(tcompt);if bad_compt then exit;
    get_param(tcompt,1,value,unit1);if bad_compt then exit;
    case unit1 of
      Omega   : zed:=value;
      's','S' : zed:=1.0/value;
      'z','Z' : zed:=z0*value;
      'y','Y' : zed:=z0/value;
       else begin
         bad_compt:=true;
         message[1]:='Invalid tline';
         message[2]:='impedance unit';
         message[3]:='Use y, z, S or '+Omega;
         exit;
       end;
    end;{case}
    if zed <= 0 then begin
      bad_compt:=true;
      message[1]:='tline impedance';
      message[2]:='must be positive';
      exit;
    end;
    width:=widtht(zed); if bad_compt then exit;
    if width < resln then begin
      bad_compt:=true;
      message[1]:='Impedance too big';
      message[2]:='tline too narrow';
      message[3]:='(<'+sresln+')';
      exit;
    end;
    if width > bmax then begin
      bad_compt:=true;
      message[1]:='Impedance is';
      message[2]:='too small';
```

```
      message[3]:='tline too wide';
      exit;
    end;
    ere:=(er+1)/2 + (er-1)/2/sqrt(1+10*substrate_t/width);
    get_param(tcompt,2,value,unit1);if bad_compt then exit;
    case unit1 of
      degree : begin
                 wavelength:=value/360.0;
                 lngth:=(300/design_freq)*wavelength/sqrt(ere);
               end;
      'm','M': begin
                 lngth:=value; {mmlong}
                 wavelength:=(design_freq/300)*lngth*sqrt(ere)
               end;
        else begin
          bad_compt:=true;
          message[1]:='Invalid tline';
          message[2]:='length unit';
          message[3]:='Use mm or '+Degree;
          exit;
        end;
      end;{case}
      if correction_compt^.descript<>'' then begin
        get_param(correction_compt,1,value,unit1);if bad_compt then exit;
        case unit1 of
          Degree : lngth:=lngth+(300/design_freq)*(value/360.0)/sqrt(ere);
          'h'    : lngth:=lngth+value*substrate_t;
          'm'    : lngth:=lngth+value;
        end;{case}
      end;
      if lngth > bmax then begin
        bad_compt:=true;
        message[1]:='tline is longer';
        message[2]:='than board size';
        exit;
      end;
      if lngth < resln then begin
        bad_compt:=true;
        message[1]:='tline too short';
        message[2]:='Length must be';
        message[3]:='>'+sresln;
        exit;
      end;
      con_space:=0.0;
      number_of_con:=2;
    end;{with tcompt}
  end else begin
    elength:=2*pi*tcompt^.wavelength;
    gamma:=freq/design_freq;
    zd:=tcompt^.zed/z0;
    sh:=sin(elength*gamma);
    ch:=cos(elength*gamma);
    rds:=rc(co(2*zd*ch,(sqr(zd)+1.0)*sh));{rc}
```

```
    new(u);
    c_ss[1,1]:=prp(u,co(0.0,(sqr(zd)-1.0)*sh),rds);
    c_ss[2,2]:=co(c_ss[1,1]^.r,c_ss[1,1]^.i);
    c_ss[1,2]:=sm(2*zd,rds);
    c_ss[2,1]:=co(c_ss[1,2]^.r,c_ss[1,2]^.i);
    c_s:=nil;
    for j:=1 to 2 do
    for i:=1 to 2 do begin
      if c_s=nil then begin
        new(tcompt^.s_begin);
        c_s:=tcompt^.s_begin;
      end else begin
        new(c_s^.next_s);
        c_s:=c_s^.next_s;
      end;
      c_s^.next_s:=nil;
      c_s^.z:=c_ss[i,j];
    end;{i}
    if debug then begin
      write(lst,'line',c_ss[1,1]^.r:12:6,c_ss[1,1]^.i:12:6);wso(c_ss[1,1]^.r,2)
    end;
  end;{action}
end; {tline0}

overlay procedure clines0(tcompt : compt);
{Cline equivalent of tline}
var
  i,j,mi,mj : integer;
  rds,s11eo : complex;
  c_s       : s_param;
  unit1     : char;
  seo       : array[1..2] of array[1..2] of array[1..2] of complex;
  wide,wido,zd,elength,sh,ch,gamma,widthc,spacec,value,zt,ere : real;
begin
  if action then begin
    with tcompt^ do begin
      correction_compt^.descript:=get_correction(tcompt);
      if bad_compt then exit;
      number_of_con:=4;
      zed:=0; zedo:=0; wavelength:=0; lngth:=0;
      for i:=1 to 3 do begin
        j:=goto_numeral(i,tcompt^.descript);bad_compt:=false;
        if (j > 0) or (i < 3) then begin
          get_param(tcompt,i,value,unit1); if bad_compt then exit;
          case unit1 of
           omega   : if zed=0 then zed:=value else zedo:=value;
           's','S' : if zed=0 then zed:=1.0/value else zedo:=1.0/value;
           'z','Z' : if zed=0 then zed:=z0*value else zedo:=z0*value;
           'y','Y' : if zed=0 then zed:=z0/value else zedo:=z0/value;
            degree : wavelength:=value/360.0;
           'm','M' : lngth:=value; {mmlong}
            else begin
              bad_compt:=true;
```

```
        message[1]:='Missing clines';
        message[2]:='unit. Use y,';
        message[3]:='z, S, '+Omega+', mm or '+Degree;
        exit;
      end;
    end;{case}
  end;{j}
end;{i}
if (zed = 0) and (zedo = 0) then begin
  bad_compt:=true;
  message[1]:='Both cline even';
  message[2]:='& odd impedances';
  message[3]:='not found or zero';
  exit;
end else begin
  if zed=0 then  zed :=sqr(Z0)/zedo;
  if zedo=0 then zedo:=sqr(Z0)/zed;
end;
if (zed <= 0) or (zedo <= 0) then begin
  bad_compt:=true;
  message[1]:='cline';
  message[2]:='impedances must';
  message[3]:='be positive';
  exit;
end;
if zed < zedo then begin zt:=zed; zed:=zedo; zedo:=zt; end;
if zed*0.98 < zedo then begin
   bad_compt:=true;
   message[1]:='cline even & odd';
   message[2]:='impedances are';
   message[3]:='too close';
   exit;
end;
if not(bad_compt) then begin
  wide:=widtht(zed /2.0)/substrate_t;
  wido:=widtht(zedo/2.0)/substrate_t;if bad_compt then exit;
  width_spacing_coupler(wide,wido,widthc,spacec);
  if not(bad_compt) then begin
    width:=widthc;
    con_space:=spacec;
    if con_space < resln then begin
      bad_compt:=true;
      message[1]:='clines spacing is';
      message[2]:='<'+sresln;
      message[3]:=Omega+'e/'+Omega+'o is too big';
      exit;
    end;
    if width < resln then begin
      bad_compt:=true;
      message[1]:='Even impedance is';
      message[2]:='too large. Widths';
      message[3]:='must be >'+sresln;
      exit;
```

```
          end;
          ere:=(er+1)/2 + (er-1)/2/sqrt(1+10*substrate_t/width);
          if (lngth=0) and (wavelength=0) then begin
            bad_compt:=true;
            message[1]:='Missing cline';
            message[2]:='length. Supply';
            message[3]:='length in mm or '+Degree;
          end else begin
            if lngth=0 then lngth:=(300/design_freq)*wavelength/sqrt(ere)
                       else wavelength:=(design_freq/300)*lngth*sqrt(ere);
            if correction_compt^.descript<>'' then begin
            get_param(correction_compt,1,value,unit1);if bad_compt then exit;
            case unit1 of
             Degree : lngth:=lngth+(300/design_freq)*(value/360.0)/sqrt(ere);
             'h'    : lngth:=lngth+value*substrate_t;
             'm'    : lngth:=lngth+value;
            end;{case}
            end;{if correction}
          end;
        end;{if not_bad}
      end;{if not_bad}
    end;{with}
  end else begin{action}
    gamma:=freq/design_freq;
    for i:=1 to 2 do begin
      if i=1 then zd:=tcompt^.zed/z0 else zd:=tcompt^.zedo/z0;
      elength:=2*pi*tcompt^.wavelength;
      sh:=sin(elength*gamma);
      ch:=cos(elength*gamma);
      rds:=rc(co(2*zd*ch,(sqr(zd)+1.0)*sh));{rc}
      s11eo:=sm((sqr(zd)-1.0)*sh,rds);{j*}
      seo[i,1,1]:=co(0.5*s11eo^.i,-0.5*s11eo^.r);
      seo[i,1,2]:=sm(zd,rds);
      seo[i,2,2]:=seo[i,1,1];
      seo[i,2,1]:=seo[i,1,2];
    end;{for i}
    c_s:=nil;
    for j:=1 to 4 do begin
    if j>2 then mj:=j-2 else mj:=j;
    for i:=1 to 4 do begin
      if c_s=nil then begin
        new(tcompt^.s_begin);
        c_s:=tcompt^.s_begin;
      end else begin
        new(c_s^.next_s);
        c_s:=c_s^.next_s;
      end;
      c_s^.next_s:=nil;
      if i>2 then mi:=i-2 else mi:=i;
      if (i>2) xor (j>2) then c_s^.z:=di(seo[1,mi,mj],seo[2,mi,mj])
                         else c_s^.z:=su(seo[1,mi,mj],seo[2,mi,mj]);
      if debug then writeln(lst,i:3,j:4,c_s^.z^.r:8:3,c_s^.z^.i:8:3);
    end;{i}
```

```
    end;{j}
  end;{action}
end; {clines0}

overlay procedure lumped0(tcompt : compt);
{Lumped equivalent of tline}
var
  value1,value2,value3,value4,ff,zi : real;
  zo2,yb2,s11,s21 : complex;
  i,j   : integer;
  unit1 : char;
begin
  if action then begin
    with tcompt^ do begin
      number_of_con:=2;
      con_space:=0.0;
      get_lumped_params(tcompt,value1,value2,value3,value4,unit1);
      if bad_compt then exit;
      lngth:=value4;
      case unit1 of
        omega : begin
                  zed:=value1/z0;
                  zedo:=value2/z0;
                  wavelength:=value3/z0;
                  spec_freq:=1;
                end;
        'z','Z': begin
                  zed:=value1;
                  zedo:=value2;
                  wavelength:=value3;
                  spec_freq:=1;
                 end;
        's','S': begin
                  zed:=value1*z0;
                  zedo:=value2*z0;
                  wavelength:=value3*z0;
                  spec_freq:=-1;
                end;
        'y','Y': begin
                  zed:=value1;
                  zedo:=value2;
                  wavelength:=value3;
                  spec_freq:=-1;
                end;
       end;{case}
      if zed < 0 then zed:=zed*one;
      if lngth<=0 then begin
         bad_compt:=true;
         message[1]:='lumped length';
         message[2]:='must be in mm';
         message[3]:='and >'+sresln;
      end;
      width:=0;
```

```
    end;{with}
  end else begin
    ff:=freq/design_freq;
    with tcompt^ do
    if spec_freq > 0 then begin {z ,zedo=Ind wavlength=Cap}
      if freq= 0 then begin
        if wavelength=0 then s21:=co(1/(1+zed/2),0) else s21:=co(0.0,0.0);
      end else begin
        zi:=(zedo*ff+wavelength/ff)/2;
        zo2:=co(1+zed/2,zi);
        s21:=rc(zo2);
      end;
      new(s11);
      s11^.r:=1-s21^.r;
      s11^.i:=-s21^.i;
    end else begin              {y}
      if freq= 0 then begin
        if wavelength=0 then s11:=co(1/(1+zed*2),0) else s11:=co(0.0,0.0);
      end else begin
        zi:=2*(zedo*ff+wavelength/ff);
        yb2:=co(1+2*zed,zi);
        s11:=rc(yb2);
      end;
      new(s21);
      s21^.r:=1-s11^.r;
      s21^.i:=-s11^.i;
    end;
    c_s:=nil;
    for j:=1 to 2 do
    for i:=1 to 2 do begin
      if c_s=nil then begin
        new(tcompt^.s_begin);
        c_s:=tcompt^.s_begin;
      end else begin
        new(c_s^.next_s);
        c_s:=c_s^.next_s;
      end;
      c_s^.next_s:=nil;
      if i=j then c_s^.z:=co(s11^.r,s11^.i) else c_s^.z:=co(s21^.r,s21^.i);
    end;{i}
  end;{action}
end; {lumped0}

overlay procedure device0(tcompt : compt);
{Device equivalent of tline}
var
  i,j,k,txpt    : integer;
  fname         : file_string;
  dev_file      : textfile;
  c_s,c_f,c_is  : s_param;
  s1,s2         : array[1..10,1..10] of complex;
  found         : boolean;
  char1,char2   : char;
```

```
  f1,f2,mag,ph,tfreq,tfmin,ffac : real;
begin
  if action then begin
    get_device_params(tcompt,fname,tcompt^.lngth); if bad_compt then exit;
    if tcompt^.lngth<=resln then begin
      bad_compt:=true;
      message[1]:='device length';
      message[2]:='must be >'+sresln;
      exit;
    end;
    if pos('.',fname)=0 then fname:=fname+'.puf';
    if (tcompt^.f_file = nil) or tcompt^.changed then
    if fileexists(true,dev_file,fname) then begin
      with tcompt^ do begin
        repeat
          readln(dev_file,char1,char2);{read \s.....}
        until ((char1='\') and (char2='s')) or EOF(dev_file);
        if EOF(dev_file) then begin
          bad_compt:=true;
          message[1]:='No s parameters';
          message[2]:='found in';
          message[3]:=fname;
          exit;
        end;
        readln(dev_file);{read comments s11 ...}
        number_of_con:=2;   width:=0;
        con_space:=0.0;
        c_s:=nil;
        c_f:=nil;
        repeat
          read(dev_file,char1);{writeln(lst,char1);}
          if char1 <> '\' then begin
            if c_f=nil then begin
              new(tcompt^.f_file);
              c_f:=tcompt^.f_file;
            end else begin
              new(c_f^.next_s);
              c_f:=c_f^.next_s;
            end;
            c_f^.next_s:=nil;
            new(c_f^.z);
            read(dev_file,c_f^.z^.r);{writeln(lst,'freq ',c_f^.z^.r:10:4);}
            for j:= 1 to number_of_con do
            for i:= 1 to number_of_con do begin
              if c_s=nil then begin
                new(tcompt^.s_file);
                c_s:=tcompt^.s_file;
              end else begin
                new(c_s^.next_s);
                c_s:=c_s^.next_s;
              end;
              c_s^.next_s:=nil;
              read(dev_file,mag,ph);
```

```
          new(c_s^.z);
          c_s^.z^.r:=one*mag*cos(ph*pi/180);
          c_s^.z^.i:=one*mag*sin(ph*pi/180);
          {writeln('ij',i:4,j:4,mag:8:1,ph:4:1);}
        end;{for i}
        readln(dev_file);
      end;{if char1 <> '\''}
    until EOF(dev_file) or (char1='\');
  end;{with}
  close(dev_file);
 end else bad_compt:=true;
end else begin
 if xpt=0 then begin
   with tcompt^ do begin
     tfmin:=f_file^.z^.r;
     s_ifile:=nil;
     for txpt:=0 to npts do begin
       tfreq:=fmin+txpt*finc;
       found:=false;{writeln(lst,txpt:4,tfmin:12:4,tfreq:12:4);}
       if tfmin <= tfreq then begin
         c_f:=nil; c_s:=nil;
         repeat
           if c_f=nil then c_f:=f_file else c_f:=c_f^.next_s;
           for j:= 1 to number_of_con do
           for i:= 1 to number_of_con do begin
             if c_s=nil then c_s:=s_file else c_s:=c_s^.next_s;
             s1[i,j]:=c_s^.z
           end;{i j}
           if c_f^.next_s^.z^.r >= tfreq then found:=true;
         until found or (c_f^.next_s^.next_s=nil);
         f1:=c_f^.z^.r;f2:=c_f^.next_s^.z^.r;
         for j:= 1 to number_of_con do
         for i:= 1 to number_of_con do begin
           c_s:=c_s^.next_s;  s2[i,j]:=c_s^.z
         end;{i j}
       end;{if tfmin < tfreq}
       for j:= 1 to number_of_con do
       for i:= 1 to number_of_con do begin
         if s_ifile=nil then begin
           new(s_ifile);
           c_is:=s_ifile;
         end else begin
           new(c_is^.next_s);
           c_is:=c_is^.next_s;
         end;
         c_is^.next_s:=nil;
         if found then begin
           ffac:=(tfreq-f1)/(f2-f1); new(c_is^.z);
           c_is^.z^.r:=s1[i,j]^.r+ffac*(s2[i,j]^.r-s1[i,j]^.r);
           c_is^.z^.i:=s1[i,j]^.i+ffac*(s2[i,j]^.i-s1[i,j]^.i);
         end else c_is^.z:=nil;
       end;{for i}
       if debug then begin
```

```
          write(lst,'out',txpt:4,tfmin:10:4,tfreq:10:4,c_is^.z^.r:10:4);
          wso(c_is^,1);wso(s_ifile^,2);
        end;
      end;{for txpt}
    end; {if fileexists}
  end;{if xpt:=0}
  with tcompt^ do begin
    s_begin:=s_ifile;
    for k:=0 to xpt-1 do
    for j:=1 to number_of_con do
    for i:=1 to number_of_con do begin
      s_begin:=s_begin^.next_s;
      if s_begin=nil then begin
        message[2]:='s out of range';
        shutdown;
      end;
    end;
    if s_begin^.z=nil then begin
      bad_compt:=true;
      message[1]:='Frequency out of';
      message[2]:='   range given   ';
      message[3]:='in device file';
    end;
  end;{with}
 end;{action}
end; {device0}

overlay procedure title0;
{Set up titles for the three windows}
var
  i : integer;
begin
  for i:=1 to 3 do begin
    new(window_f[i]); new(command_f[i]);
    case i of
     1 : begin
          with window_f[1]^ do begin
           xp:=2; yp:=13; descript:=' F1 : CIRCUIT';
          end;
          with command_f[1]^ do begin
           xp:=32;yp:=1; descript:='CIRCUIT  COMMANDS';
          end;
        end;
     2 : begin
          with window_f[2]^ do begin
           xp:=41; yp:=15; descript:=' F2 : PLOT ';
          end;
          with command_f[2]^ do begin
           xp:=33; yp:=1; descript:=' PLOT COMMANDS ';
          end;
        end;
     3 : begin
          with window_f[3]^ do begin
```

```
            xp:=2; yp:=15; descript:=' F3 : PARTS ';
           end;
           with command_f[3]^ do begin
            xp:=32;yp:=1; descript:=' PARTS  COMMANDS ';
           end;
         end;
    end;{case}
  end;{for i}
end; {title0}
```

```
{Include file 2:INC2E3.PAS}

overlay procedure draw_tline0(tnet : net; line,seperate : boolean);
{Draw a transmission line on the circuit board}
var
  x1,x2,y1,y2,x3,y3 : integer;
  x1r,y1r,x2r,y2r   : real;
begin
  lengthxy(tnet);
  x1r:=tnet^.xr-lengthxm*yii/2.0;   x2r:=x1r+lengthxm*(xii+yii);
  y1r:=tnet^.yr-lengthym*xii/2.0;   y2r:=y1r+lengthym*(yii+xii);
  x1:=Round(x1r/csx+xmin[2]);
  x2:=Round(x2r/csx+xmin[2]);
  x3:=Round((x1r+x2r)/(2.0*csx)+xmin[2]);
  y1:=Round(y1r/csy+ymin);
  y2:=Round(y2r/csy+ymin);
  y3:=Round((y1r+y2r)/(2.0*csy)+ymin);
  if line then fill_box(x1,y1,x2,y2,brown) else begin
    if x1=x2 then begin
      x1:=Round(x1-widthZ0/(2.0*csx));
      x2:=Round(x2+widthZ0/(2.0*csx));
    end else begin
      y1:=Round(y1-widthZ0/(2.0*csy));
      y2:=Round(y2+widthZ0/(2.0*csy));
    end;
    draw_box(x1,y1,x2,y2,lightblue);
  end;
  if seperate then begin
   x1r:=(tnet^.xr+tnet^.com^.con_space*yii/2.0)/csx+xmin[2];
   y1r:=(tnet^.yr+tnet^.com^.con_space*xii/2.0)/csy+ymin;
   puff_draw(Round(x1r),Round(y1r),Round(x1r+lengthxm*xii/csx),
                                  Round(y1r+lengthym*yii/csy),black);
  end;
  write_gchar(tnet^.com^.descript[1],x3,y3);
end; {draw_tline0}

overlay procedure draw_device0(tnet : net);
{Draw a triangle to represent a device}
var
  x1,x2,y1,y2,xt,yt : integer;
begin
  lengthxy(tnet);
  x1:=Round(tnet^.xr/csx)+xmin[2];
  y1:=Round(tnet^.yr/csy)+ymin;
  x2:=Round((tnet^.xr+lengthxm*xii)/csx)+xmin[2];
  y2:=Round((tnet^.yr+lengthym*yii)/csy)+ymin;
  xt:=Round(yii*widthz0/csx);  yt:=Round(xii*widthz0/csy);
  puff_draw(x1-xt,y1-yt,x1+xt,y1+yt,lightblue);
  puff_draw(x1+xt,y1+yt,x2,y2,lightblue);
  puff_draw(x2,y2,x1-xt,y1-yt,lightblue);
  x1:=Round((2*x1+x2)/3.0);
  y1:=Round((2*y1+y2)/3.0);
  write_gchar(tnet^.com^.descript[1],x1,y1);
```

```
end; {draw_device0}

overlay procedure set_up_key0;
{Set up parameters for Plot window}
begin
  ccompt:=nil;
  add_coord(57,0, 5, 1, true,s_key[1]);   {dBmax}
  add_coord(57,0, 5,12, true,s_key[2]);   {dBmin}
  add_coord(57,0, 7,13,false,s_key[3]);   {fmin}
  add_coord(81,0, 7,13, true,s_key[4]);   {fmax}
  add_coord(38,7,18,16,false,'fd/'+delta+'f  '+s_key[5]);     q_compt:=ccompt;
  add_coord(38,13,18,17,false,'Smith radius '+s_key[6]);rho_fac_compt:=ccompt;
  add_coord(40,1, 3,21,false,'S'+s_key[7]);          s_param_table[1]:=ccompt;
  add_coord(40,1, 3,22,false,'S'+s_key[8]);          s_param_table[2]:=ccompt;
  add_coord(40,1, 3,23,false,'S'+s_key[9]);          s_param_table[3]:=ccompt;
  add_coord(40,1, 3,24,false,'S'+s_key[10]);         s_param_table[4]:=ccompt;
end; {set_up_key0}

overlay procedure set_up_char0;
{Set up extendended graphics characters}
const
 char_hrs: char_s=(
                   $c0,$60,$30,$38,$3c,$6c,$c6,$00, {Lamdda 128}
                   $00,$00,$10,$28,$44,$c6,$fe,$00, {delta  129}
                   $10,$28,$44,$ee,$28,$28,$38,$00, {shift_arrow 130}
                   $10,$10,$10,$10,$10,$10,$10,$00, {bar 131}
                   $00,$00,$00,$FF,$00,$3c,$00,$18, {ground 132}
                   $00,$00,$3c,$46,$43,$46,$3c,$00,{infin}
                   $00,$00,$78,$c4,$84,$c4,$78,$00, {ity}
                   $00,$00,$00,$00,$00,$00,$00,$00,
                   $00,$00,$00,$00,$00,$00,$00,$00,
                   $00,$00,$00,$00,$00,$00,$00,$00,
                   $00,$00,$00,$00,$00,$00,$00,$00,
                   $38,$6c,$c6,$c6,$6c,$28,$ee,$00, {Omega 139}
                   $30,$48,$48,$30,$00,$00,$00,$00, {degree 140}
                   $00,$00,$00,$00,$00,$00,$00,$00);

char_ega:  char_s=(
     $00,$40,$60,$30,$18,$1c,$1c,$36,$66,$42,$00,$00,$00,$00,{Lambda}
     $00,$00,$00,$00,$08,$1c,$36,$63,$c1,$ff,$00,$00,$00,$00,{delta}
     $00,$08,$1C,$36,$63,$77,$14,$14,$14,$1C,$00,$00,$00,$00,{shift_arrow}
     $10,$10,$10,$10,$10,$10,$10,$10,$10,$10,$10,$00,$00,$00,{bar}
     $00,$00,$00,$00,$00,$FF,$00,$00,$3c,$00,$00,$18,$00,$00,{ground}
     $00,$00,$00,$1c,$27,$23,$21,$22,$1c,$00,$00,$00,$00,$00, {inf}
     $00,$00,$00,$38,$44,$84,$c4,$e4,$38,$00,$00,$00,$00,$00, {ty}
     $00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00,$00);

var
  a,b,i : integer;
begin
  if EGA then begin
    with result do begin
      bh:=2;  ah:=$11;  al:=$30;
```

```
      intr($10,result);
      move(mem[es:bp],char_p^,14*256);
      es:=seg(char_p^);   bp:=ofs(char_p^);
      cx:=14; bl:=0;dl:=14;
      ah:=$11;al:=$21;
      intr($10,result);
    end; {with}
    a:=seg(char_p^); b:=ofs(char_p^);
    for i:=1 to 14 do begin
      mem[a:b+128*14+i]:=char_ega[i];    {lambda}
      mem[a:b+129*14+i]:=char_ega[i+14]; {delta}
      mem[a:b+130*14+i]:=char_ega[i+28]; {shift_arrow}
      mem[a:b+131*14+i]:=char_ega[i+42]; {bar}
      mem[a:b+132*14+i]:=char_ega[i+56]; {ground}
      mem[a:b+133*14+i]:=char_ega[i+70]; {infin}
      mem[a:b+134*14+i]:=char_ega[i+84]; {ity}

      mem[a:b+139*14+i]:=mem[a:b+234*14+i]; {omega}
      mem[a:b+140*14+i]:=mem[a:b+248*14+i]; {degree}
    end;{for i}
  end else begin
    new(char_p);
    with result do begin
      ds:=seg(char_p^);   dx:=ofs(char_p^);
      ah:=$25;   al:=$1f;
    end;
    msdos(result);
    char_p^:=char_hrs;
  end;{if EGA}
end; {set_up_char0}

overlay procedure read_board0;
{Read board parameters from .puf file}
var
  i     : integer;
  value : real;
  strz,strf   : string[6];
  tcompt      : compt;
  char1,char2 : char;
begin
  repeat
    if Eoln(net_file) then begin {ignore blank lines}
       readln(net_file);char1:=' ';
    end else begin
      read(net_file,char1);
      if char1<>'\' then begin
        repeat
          read(net_file,char2);
        until char2=' ';
        readln(net_file,value);{writeln(lst,char1,value:12:4);}
        case char1 of
          'e' : begin
                  er:=value;
```

```
                board[1]:=true;
              end;
        't' : begin
                substrate_t:=value;
                board[2]:=true;
              end;
        's' : begin
                bmax:=value;
                csy:=bmax/(ymax-ymin);
                csx:=csy*yf;
                board[3]:=true;
              end;
        'p' : begin
                reduction:=value;
                if reduction > 0 then begin
                  psx:=red_psx/reduction;
                  psy:=red_psy/reduction;
                  board[4]:=true;
                end;
              end;
        'c' : begin
                con_sep:=value;
                board[5]:=true;
              end;
        'r' : begin
                resln:=value;
                Str(resln:5:2,sresln);
                if sresln[5]='0' then delete(sresln,5,1);
                sresln:=sresln+'mm';
                board[6]:=true;
              end;
        'z' : begin
                Z0:=value;
                board[7]:=true;
              end;
        'n' : begin
                nft:=Round(value);
                if not(nft in [1,2,4,8,16,32,64,128,256]) then nft:=128;
                board[8]:=true;
              end;
        'd' : begin
                display:=Round(value);
                EGA:=false;
                board[9]:=true;
                case display of
                  1 : EGA:=true;             {EGA}
                  2 : hires_color:=black;    {CGA}
                  3 : hires_color:=white;    {BW}
                  else begin                  {default}
                    mem[0:$410]:=mem[0:$410] and 239;{make 5th bit zero
                                                        Norton p52}
                    result.al:=0;   result.ah:=$12;
                    result.bl:=$10; result.bh:=$F0; intr($10,result);
```

```
                     if result.bx = 3 then EGA:=true else hires_color:=white;
                   end;
                 end;{case}
                 if EGA then imin:=1 else imin:=0;
               end;{begin}
         'f' : begin
                 design_freq:=value;
                 if design_freq > 0 then board[10]:=true;
               end;
         'a' : begin
                 artwork_cor:=value;
                 board[11]:=true;
               end;
      end;{case}
    end; {if char1}
  end;{if Eoln}
 until (char1='\') or EOF(net_file);
 for i:=2 to 11 do board[1]:=board[1] and board[i];
 board_read:=board[1];
 if board_read then begin
   if abs(con_sep) > bmax then con_sep:=bmax;
   if z0 > 5000/sqrt((er+1)/2.0) then z0:=50;
   widthZ0:=widtht(z0);
   pwidthxZ02:=Round(widthZ0*0.5/psx);  pwidthyZ02:=Round(widthZ0*0.5/psy);
   cwidthxZ02:=Round(widthZ0*0.5/csx);  cwidthyZ02:=Round(widthZ0*0.5/csy);
   tcompt:=nil;
   repeat
     if tcompt=nil then tcompt:=part_start else tcompt:=tcompt^.next_compt;
     tcompt^.changed:=true;
   until tcompt^.next_compt=nil;
   if z0 >=100 then i:=3 else i:=2;          Str(z0:i:0,strz);
   if design_freq >=10 then i:=5 else i:=4;  Str(design_freq:i:2,strf);
   window_f[3]^.descript:=' F3 : PARTS  z='+strz+Omega+' '+'fd='+strf+'GHz '
 end;
end; {read_board0}

overlay procedure read_key0;
{Read key from .puf file}
var
 len,j,i : integer;
 des     : line_string;
 c1,c2,c3,char1 : char;
begin
  for i:=1 to 6  do s_key[i]:=' ';
  for i:=7 to 10 do s_key[i]:='';
  repeat
    if Eoln(net_file) then begin {ignore blank lines}
       readln(net_file);char1:=' ';
    end else begin
      read(net_file,char1);  des:='';
      if char1 <> '\' then begin
        des:=char1;
        repeat
```

```
        read(net_file,char1);
        des:=des+char1;
      until (char1=lbrack) or Eoln(net_file);
      readln(net_file);
      c1:=des[1];
      c2:=des[2];
      c3:=des[3];
      while not(des[1] in ['0'..'9','.',',','-']) do Delete(des,1,1);
      len:=length(des);
      while not(des[len] in ['0'..'9','.',',','-']) and (len > 0) do begin
        Delete(des,len,1);
        len:=length(des);
      end;
      case c1 of
        'd'     : if c2='>' then s_key[1]:=des else s_key[2]:=des;
        'f'     :  case c2 of
                    '<': s_key[3]:=des;
                    '>': s_key[4]:=des;
                    'd': if c3='/' then s_key[5]:=des
                    end; {case}
        'S','s' : if c2='m' then begin
                      s_key[6]:=des;
                  end else begin
                    j:=6;
                    repeat
                      j:=j+1;
                    until (length(s_key[j])=0) or (j>9);
                    s_key[j]:=des;
                  end;
      end;{case}
    end;{if char1 ..}
  end;{if Eoln}
 until (char1='\') or EOF(net_file);
end; {read_key}

overlay procedure read_parts0;
{Read parts from .puf file}
var
  char1  : char;
  i,j    : integer;
  des    : line_string;
  tcompt : compt;
begin
  for i:=1 to 9 do begin
    if i=1 then tcompt:=part_start else tcompt:=tcompt^.next_compt;
    with tcompt^ do begin
      descript:=char(ord('a')+i-1)+' ';
      used:=0; changed:=false; parsed:=false;
    end;
  end;{i}
  j:=0;
  repeat
    if Eoln(net_file) then begin {ignore blank lines}
```

```
      readln(net_file);char1:=' ';
    end else begin
      read(net_file,char1);
      if char1 <> '\' then begin
        readln(net_file,des);
        j:=j+1;
        if j <= 9 then begin
          i:=Pos('{',des);
          if i> 0 then Delete(des,i,length(des));
          for i:=1 to length(des) do
          case des[i] of
           'O' :  des[i]:=Omega;
           'D' :  des[i]:=Degree;
          end;
          while des[length(des)]=' ' do delete(des,length(des),1);
          if j=1 then tcompt:=part_start else tcompt:=tcompt^.next_compt;
          with tcompt^ do begin
            descript:=descript+char1+des;
            changed:=true;
          end;
        end;{if j}
      end;{if char1<>'\'}
    end;{if Eoln}
  until (char1='\') or EOF(net_file);
end; {read_parts0}

overlay procedure read_circuit0;
{Read in circuit from .puf file}
var
  key_i,nn : integer;
  char1    : char;
begin
  circuit_changed:=true; key_end:=0;
  repeat
    if not(Eof(net_file)) then {read circuit}
    if Eoln(net_file) then begin {ignore blank lines}
      readln(net_file);char1:=' ';
    end else begin
      read(net_file,char1);
      if char1 <> '\' then begin
        readln(net_file,key_i,nn);
        key:=char(key_i);
        update_key_list(nn);
      end;{if char1}
    end;{if Eoln}
  until (char1='\') or EOF(net_file);
  key_i:=0; {set_up for redraw}
end; {read_circuit0}

overlay procedure read_s_params0;
{Read s-parameters from .puf file}
var
  ij          : integer;
```

```
  freq,mag,ph : real;
  char1       : char;
begin
  filled_OK:=true;
  npts:=-1;
  readln(net_file); {comment line}
  for ij:=1 to max_params do begin
    s_param_table[ij]^.changed:=false;
    c_plot[ij]:=nil;
    plot_des[ij]:=nil;
  end;
  repeat
    if Eoln(net_file) then begin {ignore blank lines}
      readln(net_file);char1:=' ';
    end else begin
      read(net_file,char1);
      if (char1<>'\') and (npts+1 < ptmax) then begin
        read(net_file,freq);
        npts:=npts+1; if npts=0 then fmin:=freq;
        ij:=0;
        repeat
          ij:=ij+1;
          if c_plot[ij]=nil then c_plot[ij]:=plot_start[ij]
                            else c_plot[ij]:=c_plot[ij]^.next_p;
          if abs(freq-design_freq)< fmin/100.0 then plot_des[ij]:=c_plot[ij];
          s_param_table[ij]^.changed:=true;
          c_plot[ij]^.filled:=true;
          read(net_file,mag,ph);
          c_plot[ij]^.x:=mag*cos(ph*pi/180);
          c_plot[ij]^.y:=mag*sin(ph*pi/180);
        until Eoln(net_file) or (ij=max_params);
        readln(net_file);
      end;{if char}
    end;{if Eoln}
  until (char1='\') or EOF(net_file);
  if npts<=1 then filled_OK:=false;
  for ij:=1 to max_params do plot_end[ij]:=c_plot[ij];
  finc:=(freq-fmin)/npts;
end; {read_s_params0}

overlay procedure save_board0;
{Save board parameters to .puf file}
begin
  writeln(net_file,'\b',lbrack,'oard',rbrack);
  writeln(net_file,'e',er:10:3);
  writeln(net_file,'t',substrate_t:10:3);
  writeln(net_file,'s',bmax:10:3);
  writeln(net_file,'p',reduction:10:3);
  writeln(net_file,'c',con_sep:10:3);
  writeln(net_file,'r',resln:10:3);
  writeln(net_file,'z',z0:10:3);
  writeln(net_file,'n',nft:8);
  writeln(net_file,'d',display:8);
```

```
  writeln(net_file,'f',design_freq:17:9);
  writeln(net_file,'a',artwork_cor:17:9);
end; {save_board0}

overlay procedure save_key0;
{Save Plot window parameters to .puf file}
var
  tcompt : compt;
  i      : integer;
  temp   : line_string;
begin
  writeln(net_file,'\k',lbrack,'ey',rbrack);
  for i:=1 to 10 do begin
    if i=1 then tcompt:=coord_start else tcompt:=tcompt^.next_compt;
    with tcompt^ do
    case i of
      1   :  writeln(net_file,'d>  '+descript);
      2   :  writeln(net_file,'d<  '+descript);
      3   :  writeln(net_file,'f<  '+descript);
      4   :  writeln(net_file,'f>  '+descript);
      5   :  begin
               temp:=descript;
               delete(temp,1,5);
               writeln(net_file,'fd/df'+temp);
             end;
      6   :  writeln(net_file,descript);
     7..10:  begin
               temp:=descript;
               delete(temp,1,1);
               if length(temp) > 0 then writeln(net_file,'S   '+temp);
             end;
    end;{case}
  end;{i}
end; {save_key0}

overlay procedure save_parts0;
{Save parts to .puf file}
var
  tcompt : compt;
  des    : line_string;
  i      : integer;
begin
  tcompt:=nil;
  writeln(net_file,'\p',lbrack,'arts list',rbrack);
  repeat {write component list}
    if tcompt=nil then tcompt:=part_start else tcompt:=tcompt^.next_compt;
    des:=tcompt^.descript;
    if length(des) > 2 then begin
      Delete(des,1,2);
      for i:=1 to length(des) do
      case des[i] of
        Omega  : des[i]:='O';
        Degree : des[i]:='D';
```

```
      end;
      writeln(net_file,des);
    end;
  until tcompt^.next_compt=nil;
end; {save_parts0}

overlay procedure save_circuit0;
{Save circuit to .puf file}
begin
  for key_i:=1 to key_end do begin
    if key_i=1 then writeln(net_file,'\c',lbrack,'ircuit',rbrack);
    write(net_file,ord(key_list[key_i].keyl):4,key_list[key_i].noden:4);
    case key_list[key_i].keyl of
        right_arrow : writeln(net_file,'  right');
        left_arrow  : writeln(net_file,'  left');
        down_arrow  : writeln(net_file,'  down');
        up_arrow    : writeln(net_file,'  up');
        sh_right    : writeln(net_file,'  shift-right');
        sh_left     : writeln(net_file,'  shift-left');
        sh_down     : writeln(net_file,'  shift-down');
        sh_up       : writeln(net_file,'  shift-up');
        sh_1        : writeln(net_file,'  shift-1');
        sh_2        : writeln(net_file,'  shift-2');
        sh_3        : writeln(net_file,'  shift-3');
        sh_4        : writeln(net_file,'  shift-4');
        '+'         : writeln(net_file,'  shift-=');
        Ctrl_n      : writeln(net_file,'  Ctrl-n');
          else        writeln(net_file,'  ',key_list[key_i].keyl);
    end; {case}
  end; {for key_i}
end; {save_circuit0}

overlay procedure save_s_params0;
{Save s-parameters to .puf file}
var
  number_of_parameters,ij,txpt : integer;
  first_line : string[120];
  mag,deg    : real;
begin
  if filled_OK then begin
    writeln(net_file,'\s',lbrack,'parameters',rbrack);
    number_of_parameters:=0;first_line:='';
    for ij:=1 to max_params do
     if s_param_table[ij]^.changed then begin
     c_plot[ij]:=nil;
     if first_line=''
     then first_line:='   freq_GHz       '+s_param_table[ij]^.descript
     else first_line:=first_line+'             '+s_param_table[ij]^.descript;
     number_of_parameters:=number_of_parameters+1;
    end;
    writeln(net_file,first_line);
    for txpt:=0 to npts do begin
      freq:=fmin+finc*txpt;
```

```
      write(net_file,freq:9:5);
      for ij:=1 to max_params do
      if s_param_table[ij]^.changed then begin
        if c_plot[ij]=nil then c_plot[ij]:=plot_start[ij]
                         else c_plot[ij]:=c_plot[ij]^.next_p;
        mag:=sqrt(sqr(c_plot[ij]^.x)+sqr(c_plot[ij]^.y));
        deg:=atan2(c_plot[ij]^.x,c_plot[ij]^.y);
        if betweenr(0.1,mag,99.0,0.0)then write(net_file,mag:10:5,' ',deg:6:1)
                                    else write(net_file,' ',mag:9,' ',deg:6:1)
      end;
      writeln(net_file);
    end;
  end;{if filled_OK}
end; {save_s_params0}

overlay procedure draw_ground0(xr,yr : real);
{Draw ground on circuit}
var
  x1,y1,i : integer;
begin
  x1:=Round(xr/csx)+xmin[2];
  if EGA then begin
    y1:=Round(yr/csy)+ymin;
    puff_draw(x1-4,y1  ,x1+4,y1,yellow);
    puff_draw(x1-2,y1+2,x1+2,y1+2,yellow);
    puff_draw(x1-1,y1+4,x1+1,y1+4,yellow);
  end else begin
    y1:=Round(((yr/csy)+ymin)*hir);
    for i:=0 to 6 do
     if odd(i) then draw(x1-i  ,y1+5-i,x1+i  ,y1+5-i,1)
               else draw(x1-i-2,y1+5-i,x1+i+2,y1+5-i,0);
  end;
end; {draw_ground0}

overlay procedure dump0;
{Debug utility to get dump of network information}
var
  tnet : net;
  tcon : conn;
begin
  tnet:=nil;
  writeln(lst);
  if net_start <> nil  then
  repeat
    if tnet = nil then tnet:=net_start else tnet:=tnet^.next_net;
    if tnet^.node then write(lst,'nd') else write(lst,'nt');
    write(lst,tnet^.number_of_con:2);wso(tnet^,1);
    write(lst,tnet^.xr:6:2,tnet^.yr:6:2,' p',tnet^.ports_connected:2);
    tcon:=tnet^.con_start;wso(tcon^,2);
    while tcon <> nil do begin
      gotoxy(32,12);write(xi:4,xi:4,xi:4,xi:4);
      with tcon^do write(lst,'c',port_type:2,conn_no:2,dir:2,cxr:6:2,cyr:6:2);
      wso(tcon^,1);
```

```
      if tcon^.mate=nil then write(lst,' nil') else wso(tcon^.mate^,1);
      tcon:=tcon^.next_con;
      writeln(lst);
    end;{end while tcon}
  until tnet^.next_net= nil;
end; {dump0}

overlay procedure clr_plane0(col : integer);
{Clear intensified plane. Used when photographing Puff screen}
begin
 port[$3C4]:=2; port[$3C5]:=col;
 fillchar(mem[$A000:00],350*80,0);
 port[$3C4]:=2; port[$3C5]:=white;
end;

overlay function look_back0 : boolean;
{Look back at network to see how cline connection should be made}
var
  tcon,scon,mtcon,mscon : conn;
  x1,x2,y1,y2   : real;
  coupler_found : boolean;
  tnet  : net;
  d2    : integer;
  cs,cm : real;
begin
  look_back0:=false;
  coupler_found:=false;
  if cnet <> nil then
  if compt1^.typ = 'c' then begin
    tcon:=nil;
    repeat
      if tcon=nil then tcon:=cnet^.con_start else tcon:=tcon^.next_con;
      if tcon^.mate <> nil then
      if tcon^.mate^.net^.com^.typ='c' then begin
         cs:=(tcon^.mate^.net^.com^.con_space+compt1^.con_space)/2.0;
         cm:=(tcon^.mate^.net^.com^.con_space-compt1^.con_space)/2.0;
         d2:=tcon^.dir;
         mtcon:=tcon^.mate;
         case tcon^.mate^.conn_no of
          1,2 :  mscon:=mtcon^.next_con^.next_con;
          3   :  mscon:=mtcon^.net^.con_start;
          4   :  mscon:=mtcon^.net^.con_start^.next_con;
         end;
         scon:=mscon^.mate;
         Mate_Node[1]:=cnet;        x1:=Mate_Node[1]^.xr; y1:=Mate_Node[1]^.yr;
         Mate_Node[3]:=scon^.net;  x2:=Mate_Node[3]^.xr; y2:=Mate_Node[3]^.yr;
         coupler_found:=true;
      end;
    until tcon^.next_con=nil;
   if coupler_found then
   if abs(x1-x2) < resln then begin
    if y1 > y2 then begin
      case dirn of
```

```
    1 : begin
          ym:=ym-cs;
          tnet:=Mate_Node[1];
          Mate_Node[1]:=Mate_Node[3];
          Mate_Node[3]:=tnet;
          look_back0:=true;
        end;
      2 : begin
          ym:=ym-cm;
          look_back0:=true;
        end;
      3 : if d2 =1 then begin
          dirn:=2;
          ym:=ym+compt1^.con_space;
        end else dirn:=1;
     end; {case}
   end else begin
     case dirn of
      2 : begin
          ym:=ym+cs;
          tnet:=Mate_Node[1];
          Mate_Node[1]:=Mate_Node[3];
          Mate_Node[3]:=tnet;
          look_back0:=true;
        end;
      1 : begin
          ym:=ym+cm;
          look_back0:=true;
        end;
      0 : if d2 =2 then begin
          dirn:=1;
          ym:=ym-compt1^.con_space;
        end else dirn:=2;
     end; {case}
   end; {y1 > y2}
  end else begin
   if x1 > x2 then begin
     case dirn of
      3 : begin
          xm:=xm-cs;
          tnet:=Mate_Node[1];
          Mate_Node[1]:=Mate_Node[3];
          Mate_Node[3]:=tnet;
          look_back0:=true;
        end;
      0 : begin
          xm:=xm-cm;
          look_back0:=true;
        end;
      1 : if d2=3 then begin
          dirn:=0;
          xm:=xm+compt1^.con_space;
        end else dirn:=3;
```

```
        end; {case}
      end else begin
        case dirn of
          0 : begin
                xm:=xm+cs;
                tnet:=Mate_Node[1];
                Mate_Node[1]:=Mate_Node[3];
                Mate_Node[3]:=tnet;
                look_back0:=true;
              end;
          3 : begin
                xm:=xm+cm;
                look_back0:=true;
              end;
          2 : if d2=0 then begin
                dirn:=3;
                xm:=xm-compt1^.con_space;
              end else dirn:=0;
        end; {case}
      end; {x1 > x2}
    end;{if abs}
  end; {if compt1}
end; {look_back0}

overlay function occupied_port0 : boolean;
{Don't allow cursor to step over path to external port}
var
 i:integer;
begin
  occupied_port0:=false;
  for i:=1 to min_ports do
  with portnet[i]^ do begin
  if (abs(xr-xm)<resln) and (abs(yr-ym)<resln) and node then begin
     message[1]:='Use Ctrl-n';
     message[2]:='to move back';
     message[2]:='onto circuit';
     occupied_port0:=true;
     update_key:=false;
  end;end;
end; {occupied_port0}

overlay function off_board0(step_size : real) : boolean;
{Check to see if part will fit on circuit board}
var
  xrep,yrep,xrem,yrem : real;
  off_boardt   : boolean;
begin
  dirn_xy;
  with compt1^ do
  if step_size=1 then begin
    xrep:=xm+lngth*xii+yii*(width/2.0+con_space);
    yrep:=ym+lngth*yii+xii*(width/2.0+con_space);
    xrem:=xm+lngth*xii-yii*width/2.0;
```

```
     yrem:=ym+lngth*yii-xii*width/2.0;
     if  betweenr(0,xrep,bmax,0) and betweenr(0,yrep,bmax,0)
     and betweenr(0,xrem,bmax,0) and betweenr(0,yrem,bmax,0)
       then off_boardt:=false else off_boardt:=true;
   end else begin
     xrep:=xm+(lngth*xii+yii*con_space)/2.0;
     yrep:=ym+(lngth*yii+xii*con_space)/2.0;
     if  betweenr(0,xrep,bmax,0) and betweenr(0,yrep,bmax,0)
       then off_boardt:=false else off_boardt:=true;
   end;
   if off_boardt then begin
     if not(read_kbd) then begin
       key:=F3;
       read_kbd:=true;
       compt3:=ccompt; cx3:=compt3^.x_block;
     end;
     message[1]:='The part lies';
     message[2]:='outside the board';
     update_key:=false;
   end;
   off_board0:=off_boardt;
end; {off_board0}

overlay procedure write_s0(ij : integer);
{Write s-parameters in plot window box as marker is moved}
var
  rho,lnrho,deg : real;
begin
  gotoxy(43,20+ij);
  rho:=sqr(c_plot[ij]^.x)+sqr(c_plot[ij]^.y);
  deg:=atan2(c_plot[ij]^.x,c_plot[ij]^.y);
  if rho>1.0e-10 then begin
    if rho<1.0e+10 then begin
    lnrho:=10*ln(rho)/ln10;
    if s_param_table[ij]^.descript[2]in['f','F'] then
                                            lnrho:=lnrho/2.0;
    write(lnrho:6:1,'dB',deg:4:0,degree,' ');
    end else write('   ',infin,ity,'       ');
  end else write('   0        ');
end; {write_s0}

overlay procedure write_freq0;
begin
  freq:=fmin+xpt*finc;
  Textcolor(green);gotoxy(43,20);write('f',freq:7:3,' GHz');
end; {write_freq0}

overlay procedure write_commands0;
{Write commands in help box}
var
  i    :integer;
begin
 if not((window_number=3) and read_kbd and circuit_changed)then erase_message;
```

```
  for i:=1 to 7 do begin
    gotoxy(32,i+1); {position of command window}
    Textcolor(white);     write(command[window_number,i,1]);
    Textcolor(lightgray); write(command[window_number,i,2]);
  end;
  gotoxy(32,1);  write('                    ');
  draw_box(240,ymin+3,389,116,col_window[window_number]);
  gotoxy(window_f[window_number]^.xp+1,window_f[window_number]^.yp);
  Textcolor(white); write('F',window_number);
  if not(EGA) then
  with window_f[window_number]^ do begin
   neg_box(xp+1,yp,2);
   draw(xp*8-1,yp*8-9,xp*8+15,yp*8-9,1);
   draw(xp*8-1,yp*8-9,xp*8-1 ,yp*8-1,1);
  end;
  write_compt(col_window[window_number],command_f[window_number]);
end; {write_commands0}

overlay procedure write_parts_list0;
var
  tcompt : compt;
begin
  fill_box(1,198,261,342,black);
  draw_box(262,343,0,202,col_window[3]);
  write_compt(col_window[3],window_f[3]);
  if part_start <> nil then begin
    tcompt:=nil;
    repeat
      if tcompt=nil then tcompt:=part_start else tcompt:=tcompt^.next_compt;
      write_compt(lightgray,tcompt);
    until tcompt^.next_compt=nil;
  end;
end; {write_parts_list0}

overlay procedure write_coordinates0(time : boolean);
{Write paramters in plot window}
var
  i      : integer;
  tcompt : compt;
  temp   : line_string;
begin
  if time then begin
     Textcolor(lightgray);
     temp:=rho_fac_compt^.descript;
     Delete(temp,1,13);
     gotoxy(56-Length(temp),1);write(temp);
     temp:='-'+temp;
     tcompt:=tcompt^.next_compt;
     gotoxy(56-Length(temp),12);write(temp);
     gotoxy(56,13);write(sxmin:6:3);
     gotoxy(75,13);write(sxmax:6:3);
     gotoxy(53,6);  write('S');
     gotoxy(66,13); write('t nsec');
```

```
  end else begin
    for i:=1 to 10 do begin
      if i=1 then tcompt:=coord_start else tcompt:=tcompt^.next_compt;
      with tcompt^ do begin
        if right then xp:=xorig-Length(descript);
        if length(descript) > x_block then begin
           write_compt(lightgray,tcompt);
           if i in [7..10] then pattern(38*charx-1,(20+i-6)*chary-8,i-6,0);
        end;
      end;
    end;
    gotoxy(53,6);  write(bar,'S',bar);
    gotoxy(53,7);  write(' dB');
    gotoxy(66,13); write('f GHz');
  end;{do_time}
end; {write_coordinates0}

overlay procedure write_list0(tnet : net; ch : char);
{Debug utilty to dump s-parameters of tnet}
var
  s    : s_param;
  tcon : conn;
begin
  write(lst,'wl ',ch);wso(tnet^,2);
  tcon:=tnet^.con_start;
  while tcon <> nil begin
    write(lst,' t',tcon^.cxr:6:1,tcon^.cyr:6:1);
    gotoxy(32,12);write('t',tcon^.cxr:8:3,tcon^.cyr:8:3);
    s:=nil;
    repeat
      if s=nil then s:=tcon^.s_start else s:=s^.next_s;
      if s^.z=nil then write(lst,'        nil ')
                  else write(lst,s^.z^.r:6:2,s^.z^.i:6:2);
    until s^.next_s=nil;
    tcon:=tcon^.next_con;
    writeln(lst);
  end;{while tcon}
end; {write_list0}

overlay procedure EGAgraphics0;
{Initilise EGA or CGA mode}
begin
  if EGA then begin
    gdtype:=5;
  inline
    ($A0/GDTYPE/$3C/$03/$74/$0A/$3C/$04/$74/$0A/$3C/$05/$74/$0A/$EB/$0E/$B0/
     $0E/$EB/$06/$B0/$0F/$EB/$02/$B0/$10/$32/$E4/$CD/$10);
  GDGSEG  := $A000;
  GDCOLOR := Green;
  GDMERGE := 0;
  GDCUR_X := 0;
  GDCUR_Y := 0;
  GDVW_X1 := 0;
```

```
  GDVW_X2 := 639;
  GDVW_Y1 := 0;
  GDVW_Y2 := 349;
  GDS_FLG := 0;
  GDASPC1 := 5;
  GDASPC2 := 6;
  end else begin
    hires; hirescolor(hires_color);{black for CGA white otherwise}
  end;
end; {EGAgraphics0}

overlay procedure restore_box0(ij : integer);
{Restore pixels that were covered by marker}
var
  nb,k,i,j,xn,yn : integer;
begin
  for k:=0 to 1 do begin
    nb:=ij+k*max_params;
    if box_filled[nb] then begin
      xn:=box_dot[1,nb]; yn:=box_dot[2,nb];
      if EGA then begin
      case ij of
  1 : begin
        j:=1;
        for i:=-3 to 3 do begin
          j:=j+2;
          puff_plot(xn+i,yn-3,box_dot[j,nb]);
          puff_plot(xn+i,yn+3,box_dot[j+1,nb]);
        end;{i}
        for i:=-2 to 2 do begin
          j:=j+2;
          puff_plot(xn-3,yn+i,box_dot[j,nb]);
          puff_plot(xn+3,yn+i,box_dot[j+1,nb]);
        end;{i}
      end;
  2 : begin
        j:=3;
        for i:=1 to 3 do begin
          puff_plot(xn+i,yn+i,box_dot[j,nb]);
          puff_plot(xn+i,yn-i,box_dot[j+1,nb]);
          puff_plot(xn-i,yn+i,box_dot[j+2,nb]);
          puff_plot(xn-i,yn-i,box_dot[j+3,nb]);
          j:=j+4;
        end;
        puff_plot(xn,yn,box_dot[j,nb]);
      end;
  3 : begin
        j:=1;
        for i:=0 to 4 do begin
          j:=j+2;
          puff_plot(xn+i,yn+4-i,box_dot[j,nb]);
          puff_plot(xn-i,yn-(4-i),box_dot[j+1,nb]);
        end;{i}
```

```
        for i:=1 to 3 do begin
          j:=j+2;
          puff_plot(xn+i,yn-(4-i),box_dot[j,nb]);
          puff_plot(xn-i,yn+(4-i),box_dot[j+1,nb]);
        end;{i}
      end;
  4 : begin
        j:=3;
        for i:=1 to 4 do begin
          puff_plot(xn+i,yn,box_dot[j,nb]);
          puff_plot(xn-i,yn,box_dot[j+1,nb]);
          puff_plot(xn,yn+i,box_dot[j+2,nb]);
          puff_plot(xn,yn-i,box_dot[j+3,nb]);
          j:=j+4;
        end;
        puff_plot(xn,yn,box_dot[j,nb]);
      end;
    end;{case}
     end else pattern(xn,yn,ij,128);
   end;{if box_filled}
  end;{k}
end; {restore_box0}

overlay procedure move_box0(xn,yn,nb : integer);
{Move marker by first storing the dots that will be covered}
var
  i,j:integer;
begin
  box_filled[nb]:=true;
  box_dot[1,nb]:=xn; box_dot[2,nb]:=yn;
  if EGA then
    case nb of
  1,5 : begin
        j:=1;
        for i:=-3 to 3 do begin
          j:=j+2;
          box_dot[j,nb]:=gprddot(xn+i,yn-3);
          box_dot[j+1,nb]:=gprddot(xn+i,yn+3);
        end;
        for i:=-2 to 2 do begin
          j:=j+2;
          box_dot[j,nb]:=gprddot(xn-3,yn+i);
          box_dot[j+1,nb]:=gprddot(xn+3,yn+i);
        end;
      end;
   2,6 : begin
        j:=3;
        for i:=1 to 3 do begin
          box_dot[j,nb]   :=gprddot(xn+i,yn+i);
          box_dot[j+1,nb]:=gprddot(xn+i,yn-i);
          box_dot[j+2,nb]:=gprddot(xn-i,yn+i);
          box_dot[j+3,nb]:=gprddot(xn-i,yn-i);
          j:=j+4;
```

```
          end;
          box_dot[j,nb]:=gprddot(xn,yn);
        end;
   3,7 : begin
          j:=1;
          for i:=0 to 4 do begin
            j:=j+2;
            box_dot[j  ,nb]:=gprddot(xn+i,yn+(4-i));
            box_dot[j+1,nb]:=gprddot(xn-i,yn-(4-i));
          end;
          for i:=1 to 3 do begin
            j:=j+2;
            box_dot[j  ,nb]:=gprddot(xn+i,yn-(4-i));
            box_dot[j+1,nb]:=gprddot(xn-i,yn+(4-i));
          end;
        end;
   4,8 : begin
          j:=3;
          for i:=1 to 4 do begin
            box_dot[j,nb]  :=gprddot(xn+i,yn);
            box_dot[j+1,nb]:=gprddot(xn-i,yn);
            box_dot[j+2,nb]:=gprddot(xn,yn+i);
            box_dot[j+3,nb]:=gprddot(xn,yn-i);
            j:=j+4;
          end;
          box_dot[j,nb]:=gprddot(xn,yn);
        end;
  end;{case}
end; {move_box0}

overlay procedure draw_cursor0;
{Draw circuit cursor X by first storing the dots that will be covered}
var
  i,j,x_ii,y_ii,xo,yo : integer;
begin
  cross_dot[1]:=xi+xmin[2]; cross_dot[2]:=yi+ymin; j:=4;
  xo:=cross_dot[1];   yo:=cross_dot[2];
  if EGA then begin
   cross_dot[j-1]:=gprddot(xo,yo);puff_plot(xo,yo,white);
   x_ii:=1;   y_ii:=1;
   for i:=1 to 8 do begin
    cross_dot[j  ]:=gprddot(xo+x_ii,yo+y_ii);puff_plot(xo+x_ii,yo+y_ii,white);
    cross_dot[j+1]:=gprddot(xo+x_ii,yo-y_ii);puff_plot(xo+x_ii,yo-y_ii,white);
    cross_dot[j+2]:=gprddot(xo-x_ii,yo+y_ii);puff_plot(xo-x_ii,yo+y_ii,white);
    cross_dot[j+3]:=gprddot(xo-x_ii,yo-y_ii);puff_plot(xo-x_ii,yo-y_ii,white);
    if odd(i) then x_ii:=x_ii+1 else y_ii:=y_ii+1;
    j:=j+4;
   end;
  end else begin
    puff_draw(xo-7,yo-7,xo+7,yo+7,129);
    puff_draw(xo-7,yo+7,xo+7,yo-7,129);
  end; {if EGA}
end; {draw_cursor0}
```

```
overlay procedure erase_cursor0;
{Erase cursor and restore covered pixels. CGA uses XOR}
var
  i,j,x_ii,y_ii,xo,yo:integer;
begin
  xo:=cross_dot[1];   yo:=cross_dot[2];  j:=4;
  if EGA then begin
    x_ii:=1;   y_ii:=1;
    puff_plot(xo,yo,cross_dot[3]);
    for i:=1 to 8 do begin
      puff_plot(xo+x_ii,yo+y_ii,cross_dot[j  ]);
      puff_plot(xo+x_ii,yo-y_ii,cross_dot[j+1]);
      puff_plot(xo-x_ii,yo+y_ii,cross_dot[j+2]);
      puff_plot(xo-x_ii,yo-y_ii,cross_dot[j+3]);
      if odd(i) then x_ii:=x_ii+1 else y_ii:=y_ii+1;
      j:=j+4;
    end;
  end else begin
    puff_draw(xo-7,yo-7,xo+7,yo+7,129);
    puff_draw(xo-7,yo+7,xo+7,yo-7,129);
  end; {if EGA}
end; {erase_cursor0}

overlay procedure draw_ticks0(x1,y1,x2,y2 : integer; incx,incy : real);
{Draw ticks on rectangular plot}
var
  i,xinc,yinc : integer;
begin
  for i:=1 to 4 do begin
    xinc:=Round(i*incx);
    yinc:=Round(i*incy);
    puff_draw(x1+xinc,y1,x1+xinc,y1+3,lightgreen);
    puff_draw(x1+xinc,y2,x1+xinc,y2-3,lightgreen);
    puff_draw(x1,y1+yinc,(x1+3),y1+yinc,lightgreen);
    puff_draw(x2,y1+yinc,(x2-3),y1+yinc,lightgreen);
  end; {for}
end; {draw_ticks0}

overlay procedure four10(var data : farray; ij,nn,isign : integer);
{FFT as per Press et. al. Numerical Recipes p 394}
label
  one,two;
var
  n,i,m,j,mmax,istep : integer;
  wr,wi,wpr,wpi,wtemp,theta,tempr,tempi : real;
begin
  n:=2*nn;
  j:=1;
  i:=1;
  while i <= n do begin
    if j > i then begin
       Tempr:=Data[j,ij];
```

```
        Tempi:=Data[j+1,ij];
        Data[j,ij]:=Data[i,ij];
        Data[j+1,ij]:=Data[i+1,ij];
        Data[i,ij]:=Tempr;
        Data[i+1,ij]:=Tempi;
     end;
     m:= n div 2;
one: if (m >=2) and (j > m) then begin
     j:=j-m;
     m:=m div 2;
     goto one;
     end;
     j:=j+m;
     i:=i+2;
     end;{while i}
     mmax:=2;
two: if (n > mmax) then begin
     istep:=2*mmax;
     theta:=2*pi/(isign*mmax);
     wpr:=-2*sqr(sin(theta/2));
     wpi:=sin(theta);
     wr:=1.0;
     wi:=0.0;
     m:=1;
     while m <= mmax do begin
         i:=m;
         while i<=n do begin
          j:=i+mmax;
          tempr:=wr*data[j,ij]-wi*data[j+1,ij];
          tempi:=wr*data[j+1,ij]+wi*data[j,ij];
          data[j,ij]:=data[i,ij]-tempr;
          data[j+1,ij]:=data[i+1,ij]-tempi;
          data[i,ij]:=data[i,ij]+tempr;
          data[i+1,ij]:=data[i+1,ij]+tempi;
          i:=i+istep;
        end;{while i}
      wtemp:=wr;
     wr:=wr*wpr-wi*wpi+wr;
      wi:=wi*wpr+wtemp*wpi+wi;
      m:=m+2;
    end;{while m}
    mmax:=istep;
    goto two;
   end;{if n}
end; {four10}

overlay procedure fill_data0(ij,istart,ifinish : integer; nf : real);
{Fill array data for FFT}
var
  wf   : real;
  i    : integer;
  cplt : plot_param;
begin
```

```
  for i:= 1 to (nft+1) do
    if betweeni(istart,i-1,ifinish) then begin
      if (i-1)=istart then cplt:=plot_start[ij]
                      else cplt:=cplt^.next_p;
      if step_fn then begin
        if not(odd(i)) then begin
          wf:=nf*0.5*(1.0+cos(pi*(i-1)/(ifinish+1)))*4/(pi*(i-1));
          data[2*i-1,ij] := wf*cplt^.y;
          data[2*i,ij]   :=-wf*cplt^.x;
        end else begin
          data[2*i-1,ij] :=0;
          data[2*i,ij]   :=0;
        end;
      end else begin
        wf:=nf*0.5*(1.0+cos(pi*(i-1)/(ifinish+1)));
        data[2*i-1,ij] :=wf*cplt^.x;
        data[2*i,ij]   :=wf*cplt^.y;
      end;{if step_fn}
    end else begin
      data[2*i-1,ij] :=0;
      data[2*i,ij]   :=0;
    end;
    data[2,ij]:=data[2*nft+1,ij];
end; {fill_data0}

overlay procedure draw_smith0;
{Draw the smith chart with radius rho_fac}
var
  i,yrad,xrad,reye,centerys : integer;
begin
  fill_box(xmin[1]+9,ymax+23,639,349,black);
  if EGA then puff_draw(centerx-rad,centery,centerx+rad,centery,2) {2=green}
  else begin
    centerys:=Round(hir*centery);
    for i:=1 to rad div 4 do begin
      plot(centerx-i*4,centerys,1);
      plot(centerx+i*4,centerys,1);
    end;
  end;
  circle(centerx,centery,rad,green);
  if rho_fac <= 0.003 then begin
    puff_draw(centerx,Round(centery-rad*yf),centerx,Round(centery+rad*yf),2)
  end else begin
    reye:=Round(rad/rho_fac);
    if rho_fac < 0.99 then
      arc(centerx-reye div 2,centery,reye div 2,sqr(rad))
    else begin
      circle(centerx-reye div 2,centery,reye div 2,green);
      circle(centerx+reye div 2,centery,reye div 2,green);
      circle(centerx,centery,reye,green);
    end;
    if rho_fac <= 2.0 then
    if rho_fac > 0.44 then begin
```

```
      xrad:=Round(rad*0.2/rho_fac);       yrad:=Round(rad*yf*0.4/rho_fac);
      puff_draw(centerx-xrad,centery-yrad+1,centerx-xrad-3,centery-yrad+5,2);
      puff_draw(centerx-xrad,centery+yrad-1,centerx-xrad-3,centery+yrad-5,2);
      puff_draw(centerx+xrad,centery-yrad+1,centerx+xrad+3,centery-yrad+5,2);
      puff_draw(centerx+xrad,centery+yrad-1,centerx+xrad+3,centery+yrad-5,2);
      if rho_fac > 0.5 then begin
   puff_draw(centerx-reye div 2-1,centery-2,centerx-reye div 2-1,centery+2,2);
   puff_draw(centerx+reye div 2+1,centery-2,centerx+reye div 2+1,centery+2,2);
        if rho_fac >= 1.0 then begin
          yrad:=Round(rad*yf/rho_fac);
          puff_draw(centerx,centery+yrad,centerx,centery+yrad-4,2);{ticks}
          puff_draw(centerx,centery-yrad,centerx,centery-yrad+4,2);
        end; {1.0}
      end;{0.5}
    end;{0.44}
  end; {if rho_fac}
end; {draw_smith0}

{START ARTWORK OVERLAYS}

overlay function top_labels0 : boolean;
{Prompt user for labels to on the top of artwork mask}
begin
  top_labels0 := false;
  if printer_offline then exit;
  reset_printer;
  top_labels0 := true;
  name:=input_string('Enter your name');
  network_name:=input_string('Network name');
end; {top_labels0}

overlay procedure print_labels0;
{Prinpt labels on the top of artwork mask}
begin
  write(lst,#27'E',#27'G'); {switch on  emphasised, double strike}
  writeln(lst,name:26    + length(name) div 2);
  writeln(lst,network_name:26 + length(network_name) div 2);
  write(lst,#27'F',#27'H'); {switch off emphasised, double strike}
  p_labels := false;
end; {print_labels0}

overlay procedure get_widthxy0(tnet : net; var widthx,widthy : real);
{Given node tnet, find out the width in the x and y direction
 of connecting parts for chamfer}
var
 direction : integer;
 width     : real;
 tcon      : conn;
begin
  widthx:=0; widthy:=0;
  tnet^.nodet:=0;
  tcon:=nil;
  repeat
```

```
    if tcon=nil then tcon:=tnet^.con_start else tcon:=tcon^.next_con;
    direction:=tcon^.dir;
    if tcon^.mate=nil then width:=widthZ0
                      else width:=tcon^.mate^.net^.com^.width;
    if width <> 0 then
    case direction of
     0 : begin
           tnet^.nodet:=tnet^.nodet+1;
           widthx:=width;
         end;
     1 : begin
           tnet^.nodet:=tnet^.nodet+2;
           widthy:=width;
         end;
     2 : begin
           tnet^.nodet:=tnet^.nodet+4;
           widthy:=width;
         end;
     3 : begin
           tnet^.nodet:=tnet^.nodet+8;
           widthx:=width;
         end;
    end; {case}
  until tcon^.next_con=nil;
end; {get_widthxy0}

overlay procedure init_chamfer0(tnet : net; widthx,widthy : real);
{Calculate corners of white triangle for chamfer}
begin
  with tnet^ do begin
    if (widthx*widthy=0) or (number_of_con<>2) then chamfer:=false
                                               else chamfer:=true;
    if chamfer then case nodet of
     10:begin
         nx1:= Round((xr-widthx/2.0)/psx);
         ny1:= Round((yr-widthy/2.0)/psy);
         nx2:= Round(1.25*widthx/psx);
         ny2:= Round(1.25*widthy/psy);
        end;
     12:begin
         nx1:= Round((xr+widthx/2.0)/psx);
         ny1:= Round((yr-widthy/2.0)/psy);
         nx2:=-Round(1.25*widthx/psx);
         ny2:= Round(1.25*widthy/psy);
        end;
      5:begin
         nx1:= Round((xr+widthx/2.0)/psx);
         ny1:= Round((yr+widthy/2.0)/psy);
         nx2:=-Round(1.25*widthx/psx);
         ny2:=-Round(1.25*widthy/psy);
        end;
      3:begin
         nx1:= Round((xr-widthx/2.0)/psx);
```

```
        ny1:= Round((yr+widthy/2.0)/psy);
        nx2:= Round(1.25*widthx/psx);
        ny2:=-Round(1.25*widthy/psy);
       end;
     else chamfer:=false;
    end;{case}
    if debug then writeln(lst,'m ',chamfer:8,xr:8:4,yr:8:4,
      nx1:6,nx2:6,ny1:6,ny2:6,xii:4,yii:4,widthx:8:3,widthy:8:3,nodet:4);
  end;{with tnet}
end; {init_chamfer0}

overlay procedure init_line0(tnet : net);
{Calculate dot positons of line tnet for artwork}
var
  xt : integer;
begin
  lengthxy(tnet);
  with tnet^ do begin
    nx1:=Round((xr-yii*lengthxm/2.0)/psx);
    ny1:=Round((yr-xii*lengthym/2.0)/psy);
    nx2:=nx1+Round(lengthxm*(xii+yii)/psx);
    ny2:=ny1+Round(lengthym*(yii+xii)/psy);
    if nx1 > nx2 then begin xt:=nx1;nx1:=nx2;nx2:=xt;end;
    if ny1 > ny2 then begin xt:=ny1;ny1:=ny2;ny2:=xt;end;
    if debug then writeln(lst,'t ',xr:8:4,yr:8:4,
      nx1:6,nx2:6,ny1:6,ny2:6,xii:4,yii:4,lengthxm:8:3,lengthym:8:3);
  end;{with tnet}
end; {init_line0}

overlay procedure fill_shape0(tnet : net; corner : boolean);
{fill array bita[1..xdot_max] that will be sent to printer}
var
  yval,xbeg,xend,ybeg,yend,ix,i : integer;
  slope : real;
  temp  : byte;
begin
  with tnet^ do
  if corner then begin {chamfer corner}
    ybeg:=ny1;
    if ny2 < 0 then ybeg:=ybeg+ny2;
    yend:=ybeg+abs(ny2);
    if yend +10 > ydot then remain:=true;
    xbeg:=nx1;
    if nx2 < 0 then xbeg:=xbeg+nx2;
    xend:=xbeg+abs(nx2);
    if xbeg < 0 then xbeg:=0; if xbeg > xdot_max then xbeg:=xdot_max;
    if xend < 0 then xend:=0; if xend > xdot_max then xend:=xdot_max;
    slope:=(yend-ybeg)/(xend-xbeg);
    temp:=128;
    for i:=0 to 7 do begin
      yval:=ydot+2*i;
      if i<> 0 then temp:=temp shr 1;
      if (ybeg <= yval) and (yval <= yend) then begin
```

```
      if xend > rowl then rowl:=xend;
      for ix:=xbeg to xend do begin
      case nodet of
         10: if yval < (yend-Round((ix-xbeg)*slope)) then
                                   bita[ix]:=bita[ix] and not(temp);
         12: if yval < (ybeg+Round((ix-xbeg)*slope)) then
                                   bita[ix]:=bita[ix] and not(temp);
          5: if yval > (yend-Round((ix-xbeg)*slope)) then
                                   bita[ix]:=bita[ix] and not(temp);
          3: if yval > (ybeg+Round((ix-xbeg)*slope)) then
                                   bita[ix]:=bita[ix] and not(temp);
      end;
      end;
      end;
    end;{for i}
  end else begin    {fill_shape}
    if ny2 +10 > ydot then remain:=true;
    temp:=0;
    for i:=0 to 7 do begin
      temp:=temp shl 1;
      if (ny1 <= ydot+2*i) and (ydot+2*i <= ny2) then begin
         temp:=(temp + 1);
         if  nx2 > rowl then rowl:=nx2;
      end;{if y1}
    end;{for i}
    if nx1 < 0 then nx1:=0; if nx1 > xdot_max then nx1:=xdot_max;
    if nx2 < 0 then nx2:=0; if nx2 > xdot_max then nx2:=xdot_max;
    if temp>0 then for ix:=nx1 to nx2 do bita[ix]:=bita[ix] or temp;
  end; {if corner}
end; {fill_shape0}

overlay procedure get_coords0;
{Get values of coordintes in Plot window}
var
  tcoord : compt;
begin
  bad_compt:=false;
  tcoord:=coord_start;          symax:=get_real(tcoord,1);
  if bad_compt then exit;
  tcoord:=tcoord^.next_compt;  symin:=get_real(tcoord,1);
  if bad_compt then exit;
  if symin >= symax then begin
    bad_compt:=true;
    message[1]:='Must have';
    message[2]:='dB(max) > dB(min)';
    ccompt:=tcoord;  exit;
  end;
  tcoord:=tcoord^.next_compt;  sxmin:=get_real(tcoord,1);
  if bad_compt then exit;
  if sxmin < 0 then begin
    bad_compt:=true;
    message[1]:='Must have';
    message[2]:='frequency >= 0';
```

```
    ccompt:=tcoord;  exit;
  end;
  tcoord:=tcoord^.next_compt;  sxmax:=get_real(tcoord,1);
  if bad_compt then exit;
  if sxmax < 0 then begin
    bad_compt:=true;
    message[1]:='Must have';
    message[2]:='frequency >= 0';
    ccompt:=tcoord;  exit;
  end;
  if sxmin >= sxmax then begin
    bad_compt:=true;
    message[1]:='Must have';
    message[2]:='f(max) > f(min)';
    ccompt:=tcoord;  exit;
  end;
  sfx1:=(xmax[1]-xmin[1])/(sxmax-sxmin);
  sfy1:=(ymax-ymin)/(symax-symin);
  sigma:=(symax-symin)/100.0;
  rho_fac:=get_real(rho_fac_compt,1);
  if (rho_fac<=0.0) or bad_compt then begin
    bad_compt:=true;
    message[1]:='The Smith chart';
    message[2]:='radius must be >0';
    ccompt:=rho_fac_compt;
  end;
end; {get_coords0}

overlay procedure get_s_and_f0;
{Find out paremter in plot box ie. which s to calc at df/fd}
var
  pt_end,pt_start,ij,i,j,code1,code2 : integer;
begin
  ccompt:=q_compt; cx:=ccompt^.x_block; bad_compt:=false;
  q_fac:=Trunc(get_real(q_compt,1)); {df/fd}
  if bad_compt then begin
     message[2]:='Invalid fd/'+delta+'f';
     exit;
  end;
  bad_compt:=true;
  if q_fac < 1 then begin
     message[1]:='fd/'+delta+'f too small';
     message[2]:='or negative';
     exit;
  end;
  finc:=design_freq/q_fac;
  if (sxmin/finc < 10000) and (sxmax/finc < 10000) then begin
    if abs(sxmin/finc-Round(sxmin/finc)) < 0.001
    then pt_start:=Round(sxmin/finc) else pt_start:=Trunc(sxmin/finc)+1;
    fmin:=finc*pt_start;
    pt_end:=Trunc(sxmax/finc);
  end else begin
    message[2]:='fd/'+delta+'f too large';
```

```
    exit;
  end;
  npts:=pt_end-pt_start;
  if npts < 1 then begin
    message[2]:='fd/'+delta+'f too small';
    exit;
  end;
  if npts > ptmax then begin
    message[2]:='fd/'+delta+'f too large';
    exit;
  end;
  for ij:=1 to min_ports do begin
    inp[ij]:=false;
    out[ij]:=false;
  end;
  for ij:=1 to max_params do
  with s_param_table[ij]^ do begin
    changed:=false;
    if length(descript)>=3 then begin
      Val(descript[2],i,code1);
      Val(descript[3],j,code2);
      if (code1=0) and (code2=0) then
      if betweeni(1,i,min_ports)and betweeni(1,j,min_ports)then begin
        si[ij]:=i;        sj[ij]:=j;
        if portnet[i]^.node and portnet[j]^.node then begin
          inp[j]:=true;  out[i]:=true;
          changed:=true; bad_compt:=false;
        end;{if port}
      end;{if between}
    end;{if length}
  end; {with}
  if bad_compt then begin
     ccompt:=rho_fac_compt;
     move_cursor( 0, 1);
     message[1]:='No pair of Sij';
     message[2]:='correspond to';
     message[3]:='connected ports';
     exit;
  end;
end; {get_s_and_f0}

overlay procedure clip0(i : integer);
{Set boundary on rectangular plot for clipping}
begin
  if i=1 then begin
    GDVW_X1 := 0;
    GDVW_Y1 := 0;
    GDVW_X2 := 639;
    GDVW_Y2 := 349;
  end else begin
    GDVW_X1 := xmin[1];
    GDVW_Y1 := ymin-1;
    GDVW_X2 := xmax[1];
```

```
    GDVW_Y2 := ymax+1;
  end;
end; {clip0}

overlay procedure write_file_name0(fname : file_string);
{Write file name next to F2 : CIRCUIT}
var
 i : integer;
begin
  Textcolor(col_window[1]);
  repeat
    i:=pos('\',fname);
    if i > 0 then delete(fname,1,i);
  until i=0;
  gotoxy(window_f[1]^.xp+14,window_f[1]^.yp); write(fname);
  for i:=length(fname) to 10 do write(' ');
end;

overlay procedure smith_plot0(x1,y1,col : integer; Var line : boolean);
{Plot curve on Smith plot}
begin
  if spline_in_smith then begin
    if line then begin
       if EGA then puff_draw(xvalo[1],yvalo[1],x1,y1,col)
              else draw(xvalo[1],yvalo[1],x1,y1,1);
     end;
    line:=true;
  end else line:=false;
  xvalo[1]:=x1; yvalo[1]:=y1;
end; {smith_plot0}

overlay procedure rect_plot0(x1,y1,col : integer;Var line : boolean);
{Plot curve on rectangular plot}
begin
  GDVW_Y2 := ymax;
  if EGA then begin
    {if line then writeln(lst,xvalo[2]:4,yvalo[2]:4,x1:4,y1:4,col:4);}
    if line then puff_draw(xvalo[2],yvalo[2],x1,y1,col);
    line:=true;
  end else begin
    if spline_in_rect then begin
      if line then draw(xvalo[2],yvalo[2],x1,y1,1);
      line:=true;
    end else line:=false;
  end;
  xvalo[2]:=x1; yvalo[2]:=y1;
  GDVW_Y2 := 349;
end; {rect_plot0}

overlay procedure calc_pos0(x,y,theta,scf: real;sfreq: integer;dash: boolean);
{Given co(x,y):=rho find dot postion on screen}
var
  p2 : real;
```

```
begin
 p2:=sqr(x)+sqr(y);
 if sqrt(p2)<1.02*rho_fac then begin
   spline_in_smith:=true;
   if abs(theta) > 0 then begin
     {thet:=theta*freq/design_freq;
     sint:=sin(thet);  cost:=cos(thet);
     spx:=Round(centerx+(x*cost-y*sint)*rad/rho_fac);
     spy:=Round((centery-(x*sint+y*cost)*rad*yf/rho_fac))*scf);}
   end else begin
     spx:=Round(centerx+(x*rad/rho_fac));
     spy:=Round((centery-(y*rad*yf/rho_fac))*scf);
   end;
 end else spline_in_smith:=false;
 if p2 > 1/infty then p2:=10*ln(p2)/ln10 else p2:=-infty; {10 log(p2)}
 if betweenr(symin,p2,symax,sigma) then begin
   spline_in_rect:=true;
   spp:=Round((ymax-(p2-symin)*sfy1)*scf);
 end else begin
   spline_in_rect:=false;
   if p2 > symax then spp:=Round((ymin-5)*scf) else spp:=Round((ymax+5)*scf);
 end;
 if dash and not(betweeni(xmin[1],sfreq,xmax[1])) then begin
   spline_in_rect:=false;
   spline_in_smith:=false;
 end;
end; {calc_pos0}

overlay procedure spline0(ij : integer);
{Calculate spline coefficients. Johnson and Riess Numerical Analysis p41,241}
var
 zx,zy,u : array[0..1000] of real;
 m,i     : integer;
 li      : real;
 cplt    : plot_param;
 cspc    : spline_param;
begin
 m:=npts-1;
 for i:=0 to m do begin
   if i=0 then begin
     cplt:=plot_start[ij];
     cspc:=spline_start;
   end else begin
     cplt:=cplt^.next_p;
     cspc:=cspc^.next_c;
   end;
   with cspc^ do with cplt^ do begin
     h:=sqrt(sqr(yf*(next_p^.y-y))+sqr(next_p^.x-x));
     if h<0.000001 then h:=0.000001;
   end;
 end; {for i}
 spline_end:=cspc^.next_c;
 cplt:=plot_start[ij];     cspc:=spline_start;
```

```
  u[1]:=2*(cspc^.next_c^.h+cspc^.h);                    {u_11=a_11}
  zx[1]:=6*((cplt^.next_p^.next_p^.x-cplt^.next_p^.x)/cspc^.next_c^.h
    -(cplt^.next_p^.x-cplt^.x)/cspc^.h);   {y_1=b_1}
  zy[1]:=6*((cplt^.next_p^.next_p^.y-cplt^.next_p^.y)/cspc^.next_c^.h
    -(cplt^.next_p^.y-cplt^.y)/cspc^.h);   {y_1=b_1}
  for i:= 2 to m do begin
    cplt:=cplt^.next_p;    cspc:=cspc^.next_c;
    with cspc^ do with cplt^ do begin
      li:=h/u[i-1];             {a_i-1i=a_ii-1=h_i-1,a_ii=2(h_i+h_i-1)}
      u[i]:=2*(next_c^.h+h)-h*li;       {u_ii=a_ii-L_i.i-1a_i-1,i}
      zx[i]:=6*((next_p^.next_p^.x-next_p^.x)/next_c^.h-
                (next_p^.x-x)/h)-li*zx[i-1];                {2.33}
      zy[i]:=6*((next_p^.next_p^.y-next_p^.y)/next_c^.h-
                (next_p^.y-y)/h)-li*zy[i-1];
    end;{with}
  end;{for i}
  cspc:=spline_end;    cspc^.sx:=0;             cspc^.sy:=0;
  cspc:=cspc^.prev_c;  cspc^.sx:=zx[m]/u[m];   cspc^.sy:=zy[m]/u[m];
  for i:=1 to m-1 do begin
   cspc:=cspc^.prev_c;
   with cspc^ do begin
     sx:=(zx[m-i]-h*next_c^.sx)/u[m-i];
     sy:=(zy[m-i]-h*next_c^.sy)/u[m-i];
   end;
  end;
  cspc:=cspc^.prev_c;  cspc^.sx:=0;             cspc^.sy:=0;
end; {spline0}

procedure smith_and_magplot(dash,boxes : boolean);
{Plot s-parameter curves}
var
  jxsdif,scf,
  cx1,cx2,cx3,cx4,cy1,cy2,cy3,cy4,sqfmfj,sqfjmf,fmfj,fjmf,spar1,spar2 : real;
  sfreq,jfreq,j,nopts,ij,col,txpt : integer;
  line_s,line_r : boolean;
  cplt          : plot_param;
  cspc          : spline_param;
begin
  clip0(2);
  jxsdif:=sfx1*finc;
  if EGA then scf:=1 else scf:=hir;
  for ij:=1 to max_params do
   if s_param_table[ij]^.changed then begin
    spline0(ij);
    col:=s_color[ij];  if dash then col:=col-8;
    line_s:=false;     line_r:=false;
    for txpt:=0 to npts do begin
      if keypressed then begin
        read(kbd,key);
        if key='s' then begin
          message[2]:='      STOP       ';
          write_message;
          exit;
```

```
        end;{if key}
        beep;
      end;{if key_pressed}
      if txpt=0 then begin
        cplt:=plot_start[ij];
        cspc:=spline_start;
      end else begin
        cplt:=cplt^.next_p;
        cspc:=cspc^.next_c;
      end;
      freq:=fmin+txpt*finc;
      sfreq:=xmin[1]+Round((freq-sxmin)*sfx1);
      if cplt^.filled then begin
        calc_pos0(cplt^.x,cplt^.y,0,scf,sfreq,dash);
        rect_plot0(sfreq,spp,col,line_r);
        if spline_in_rect and boxes then box(sfreq,Round(spp/scf),ij);
        smith_plot0(spx,spy,col,line_s);
        if spline_in_smith and boxes then box(spx,Round(spy/scf),ij);
        if txpt < npts then
        with cspc^ do with cplt^ do begin
          cx1:=sx/(6*h);   cx2:=next_c^.sx/(6*h);
          cy1:=sy/(6*h);   cy2:=next_c^.sy/(6*h);
          cx3:=next_p^.x/h-next_c^.sx*h/6;  cx4:=x/h-sx*h/6;
          cy3:=next_p^.y/h-next_c^.sy*h/6;  cy4:=y/h-sy*h/6;
          if h*rad/rho_fac>40 then nopts:=10
                               else nopts:=Round(h*rad/(rho_fac*4))+1;
          for j:=1 to nopts-1 do begin
            fmfj:=j*h/nopts;    fjmf:=h-fmfj;
            sqfmfj:=sqr(fmfj);  sqfjmf:=sqr(fjmf);
            spar1:=(cx1*sqfjmf+cx4)*fjmf+(cx2*sqfmfj+cx3)*fmfj;
            spar2:=(cy1*sqfjmf+cy4)*fjmf+(cy2*sqfmfj+cy3)*fmfj;
            jfreq:=Round(j*jxsdif/nopts);
            calc_pos0(spar1,spar2,0,scf,sfreq+jfreq,dash);
            rect_plot0(sfreq+jfreq,spp,col,line_r);
            smith_plot0(spx,spy,col,line_s);
          end;{j}
        end;{if txpt}
      end;{if cpt^ filled}
    end;{for txpt}
  end;{ij}
  gds_flg:=0;
  clip0(1);
end; {smithplot}

procedure ggotoxy(var cursor_displayed : boolean);
{Activate flashing cursor}
var
  x,y,i,imax : integer;
begin
  if ccompt <> nil then begin
    x:=ccompt^.xp+cx; if x > 80 then x:=80;
    y:=ccompt^.yp;
    if cursor_displayed then begin
```

```
      Textcolor(lightgray); write(char(cursor_char)); gotoxy(x,y);
    end else begin
      gotoxy(x,y);
      result.ah := 8;  result.bh := 0; intr($10,result); {get char at x,y}
      cursor_char:=result.al;
      if insert_key then imax:=6 else imax:=2;
      for i:=imin to imax do
    if EGA then puff_draw(charx*(x-1),chary*y-i-2,charx*x-2,chary*y-i-2,white)
           else draw(charx*(x-1),8*y-i-1,charx*x-2,8*y-i-1,white);
    end; {if cursor_displayed}
    cursor_displayed:=not(cursor_displayed);
  end; {if ccompt <> nil}
end; {ggotoxy}

procedure draw_graph(x1,y1,x2,y2 : integer; time : boolean);
{Draw rectangular graph}
begin
  fill_box(x1-7*charx,0,x2+3,y2+18,black);
  if not(time) then fill_box(37*charx,252-3*chary,55*charx-1,340,black); {KEY}
  write_coordinates0(time);
  draw_box(x1,y1,x2,y2,lightgreen);
  draw_ticks0(x1,y1,x2,y2,(x2-x1)/4.0,(y2-y1)/4.0);
end; {draw_graph}

{START FFT}
procedure realft(var data : farray; ij,n,isign : integer);
{Perfrom real FFT as per Press et. al. Numerical Recipes p 400}
var
  i,i1,i2,i3,i4,n2p3 : integer;
  h1r,h1i,h2i,h2r,c1,c2,wr,wi,wpr,wpi,wtemp,theta : real;
begin
  theta:=2*pi/(2.0*n);
  wr:=1.0;
  wi:=0;
  c1:=0.5;
  if isign=1 then begin
    c2:=-0.5;
    theta:=-theta;
    four10(data,ij,n,-1);
    data[2*n+1,ij]:=data[1,ij];
    data[2*n+2,ij]:=data[2,ij];
  end else begin
    c2:= 0.5;
    theta:= theta;
    data[2*n+1,ij]:=data[2,ij];
    data[2*n+2,ij]:=0.0;
    data[2,ij]:=0.0;
  end;
  wpr:=-2*sqr(sin(theta/2.0));
  wpi:=sin(theta);
  n2p3:=2*n+3;
  for i:=1 to n div 2 + 1 do begin
    i1:=2*i-1;
```

```
      i2:=i1+1;
      i3:=n2p3-i2;
      i4:=i3+1;
      h1r:= c1*(data[i1,ij]+data[i3,ij]);
      h1i:= c1*(data[i2,ij]-data[i4,ij]);
      h2r:=-c2*(data[i2,ij]+data[i4,ij]);
      h2i:= c2*(data[i1,ij]-data[i3,ij]);
      data[i1,ij]:= h1r+wr*h2r-wi*h2i;
      data[i2,ij]:= h1i+wr*h2i+wi*h2r;
      data[i3,ij]:= h1r-wr*h2r+wi*h2i;
      data[i4,ij]:=-h1i+wr*h2i+wi*h2r;
      wtemp:=wr;
      wr:=wr*wpr-   wi*wpi+wr;
      wi:=wi*wpr+wtemp*wpi+wi;
   end;{i}
 if isign=1  then data[2,ij]:=data[2*n+1,ij] else four10(data,ij,n, 1);
 if isign=-1 then for i:=1 to (2*n+2) do data[i,ij]:=data[i,ij]/n;
end; {realft}
```

```
{Include file 3:INC3E3.PAS}

{START ANALYSIS}
procedure copy_networks(var nstart : marker; first_copy : boolean);
{Make a copy of the circuit for 'destructive' analysis.}
var
  num : integer;
begin
  if first_copy then begin
    new(nstart);
    num:=(seg(nstart^)-seg(pbeg^)) shl 4 +ofs(nstart^)-ofs(pbeg^);
    getmem(nstart,num);
    move(pbeg^,nstart^,num);
    snet_start:=net_start;
  end else begin
    num:=(seg(nstart^)-seg(pbeg^)) shl 4 +ofs(nstart^)-ofs(pbeg^);
    move(nstart^,pbeg^,num);
    net_start:=snet_start;
  end;
  if debug then begin
    write(lst,' copy_net ',num:4);wso(pbeg^,1);wso(nstart^,2);
    dump0;
  end;
end; {copy_networks}

procedure fill_sa;
{ Fill array sa with s-parameters and then load into linked list of
plot parameters ready for spline}
var
  i,j,ij,sfreq : integer;
  tcon,scon    : conn;
  s : s_param;
begin
  cnet:=nil;
  if not(bad_compt) then
  repeat
    if cnet=nil then cnet:=net_start else cnet:=cnet^.next_net;
    tcon:=nil;
    if cnet^.con_start <> nil then
    repeat
      if tcon=nil then tcon:=cnet^.con_start else tcon:=tcon^.next_con;
      j:=tcon^.port_type;
      scon:=nil;
      repeat
        if scon=nil then begin
           s:=tcon^.s_start;
           scon:=cnet^.con_start;
        end else begin
           s:=s^.next_s;
           scon:=scon^.next_con;
        end;
        i:=scon^.port_type;
        if i*j > 0 then begin
```

```
          new(sa[i,j]);
          sa[i,j]^.r:=s^.z^.r;
          sa[i,j]^.i:=s^.z^.i;
        end;
      until scon^.next_con=nil;
    until tcon^.next_con=nil;
  until cnet^.next_net=nil;
  sfreq:=xmin[1]+Round((freq-sxmin)*sfx1);
  for ij:=1 to max_params do begin
    write_freq0;
    if s_param_table[ij]^.changed then begin
      if xpt=0 then begin
        c_plot[ij]:=plot_start[ij];
        plot_des[ij]:=nil;
      end else c_plot[ij]:=c_plot[ij]^.next_p;
      plot_end[ij]:=c_plot[ij];
      if xpt=Round((design_freq-fmin)/finc) then plot_des[ij]:=c_plot[ij];
      c_plot[ij]^.filled:=false;
      case s_param_table[ij]^.descript[1] of
      's','S' : begin
                  i:=si[ij];  j:=sj[ij];
                  if sa[i,j] <> nil then begin
                    c_plot[ij]^.x:=sa[i,j]^.r;  c_plot[ij]^.y:=sa[i,j]^.i;
                  end else begin
                    c_plot[ij]^.x:=0;           c_plot[ij]^.y:=0;
                  end;
                  c_plot[ij]^.filled:=true;
                end;
      end;{case}
      if not(bad_compt) then begin
        write_s0(ij);
        calc_pos0(c_plot[ij]^.x,c_plot[ij]^.y,0,1,sfreq,false);
        if spline_in_smith then box(spx,spy,ij);
        if spline_in_rect  then box(sfreq,spp,ij);
      end;
    end;{if s_param}
  end;{for ij}
end; {fill_sa}

function get_s_and_remove(index : integer; var start : s_param):s_param;
{Get an s-parameter from a linked list and remove it}
var
  i : integer;
  s : s_param;
begin
  if index=1 then begin
    get_s_and_remove:=start;
    start:=start^.next_s;
  end else begin
    for i:=1 to index-1 do begin
      if i=1 then s:=start else s:=s^.next_s;
      if s=nil then begin
        message[2]:='get_s_and_remove';
```

```
        shutdown;
      end;
    end; {For i}
    get_s_and_remove:=s^.next_s;
    s^.next_s:=s^.next_s^.next_s;
  end;{if index}
end; {get_c_con_and_remove}

function get_c_and_remove(index : integer; var start : conn) : conn;
{Get a connector from a linked list and remove it}
var
  i : integer;
  s : conn;
begin
  if index=1 then begin
     get_c_and_remove:=start;
     start:=start^.next_con;
  end else begin
    for i:=1 to index-1 do begin
      if i=1 then s:=start else s:=s^.next_con;
      if s=nil then begin
        message[2]:='get_c_and_remove';
        shutdown;
      end;
    end; {For i}
    get_c_and_remove:=s^.next_con;
    s^.next_con:=s^.next_con^.next_con;
  end;{if index}
end; {get_c_con_and_remove}

function get_kL_from_con(tnet : net;tcon : conn) : integer;
{Find k given tnet and tcon}
var
  kL    : integer;
  found : boolean;
  vcon  : conn;
begin
  found:=false;
  kL:=0;
  vcon:=nil;
  repeat
  if vcon=nil then vcon:=tnet^.con_start else vcon:=vcon^.next_con;
    kL:=kL+1;
    if vcon=tcon then found:=true;
  until ((vcon^.next_con=nil) or (found));
  if not(found) then begin
    message[2]:='get_kL_from_con';
    shutdown;
  end;
  if debug then writeln(lst,' get_kL_from ',kL:4);
  get_kL_from_con:=kL;
end; {get_kL_from_con}
```

```
function internal_joint_remaining : boolean;
{Look for next joint to make connection.
If no joint found then internal_joint_remaining:=false}
var
  csize,size : integer;
  cNmate : net;
begin
  if debug then writeln(lst,'bintern_net ');
  Conk:=nil;
  cnet:=nil;
  csize:=1000;
  repeat{cnet}
    if cnet = nil then cnet:=net_start else cnet:=cnet^.next_net;
    ccon:=nil;
    if cnet^.con_start <> nil then
    repeat{ccon}
      if ccon=nil then ccon:=cnet^.con_start else ccon:=ccon^.next_con;
      if ccon^.port_type <=0 then begin
        cNmate:=ccon^.mate^.net;
        if cNmate=cnet then size:=cnet^.number_of_con-2
                       else size:=cnet^.number_of_con+cNmate^.number_of_con-2;
        if betweeni(1,size,csize) then begin
          csize:=size-1;
          Conk:=ccon;
        end; {if size - found simpler net to remove}
      end;{if ccon^.mate}
    until ccon^.next_con=nil;
  until cnet^.next_net=nil;
  if Conk <> nil then begin
    netK:=Conk^.net;
    netL:=Conk^.mate^.net;
    internal_joint_remaining:=true;
  end else internal_joint_remaining:=false;
  if debug then writeln(lst,' intern_net ');
end; {internal_joint_remaining}

function calc_con(Conj : conn) : boolean;
{Does connector belong to set for which s-parameters need to be calculated}
begin
  if ext_port(Conj) then calc_con:=inp[Conj^.port_type]
                    else calc_con:=true;
end;{calc_con}

procedure join_net;
{Join connectors from different networks}
var
  biL,bLL,akj,akk,aij,aik,start:s_param;
  sizea,sizeb,i,j,k,L : integer;
  num1,num2,num3,num  : complex;
  ConL,Conj           : conn;
  u      : complex;
begin
  k:=get_kL_from_con(netK,ConK);
```

```
L:=get_kL_from_con(netL,Conk^.mate);
if debug then begin
  writeln(lst,'begin join_net kl'); dump0;
  write_list0(netK,'k');  write_list0(netL,'L');
end;
Conk:= get_c_and_remove(k,netK^.con_start);
ConL:= get_c_and_remove(L,netL^.con_start);
akk := get_s_and_remove(k,Conk^.s_start);
bLL := get_s_and_remove(L,ConL^.s_start);
new(u);
num2:=rc(di(co(1.0,0.0),prp(u,akk^.z,bLL^.z)));{rc}
sizea:=netK^.number_of_con-1;
sizeb:=netL^.number_of_con-1;

new(u);
num1:=prp(u,bLL^.z,num2);
for j:=1 to sizea do begin
  if j=1 then Conj:=netK^.con_start else Conj:=Conj^.next_con;
  if calc_con(Conj) then begin
    akj:=get_s_and_remove(k,Conj^.s_start);
    new(u);
    num :=prp(u,num1,akj^.z);
    new(u);
    num3:=prp(u,num2,akj^.z);

    aij:=Conj^.s_start;
    aik:=Conk^.s_start;
    if aij^.z <> nil then supr(aij^.z,aik^.z,num);
    for i:=2 to sizea do begin
      aij:=aij^.next_s;
      aik:=aik^.next_s;
      if aij^.z <> nil then supr(aij^.z,aik^.z,num);
    end;{ for i}

    for i:=1 to sizeb do begin
      if i=1 then biL:=ConL^.s_start else biL:=biL^.next_s;
      new(aij^.next_s);
      aij:=aij^.next_s;
      if biL^.z <> nil then begin
        new(u);
        aij^.z:=prp(u,biL^.z,num3)
      end else aij^.z:=nil;
    end;{ for i}
    aij^.next_s:=nil;
    end;{if conj}
  end;{end j}
  if debug and (sizea=0) then dump0;
  if sizea= 0 then netK^.con_start:=netL^.con_start
              else Conj^.next_con:=netL^.con_start;
  if debug and (sizea=0) then dump0;

  new(u);
  num1:=prp(u,akk^.z,num2);
```

```
    for j:=1 to sizeb do begin
      if j=1 then Conj:=netL^.con_start else Conj:=Conj^.next_con;
      if calc_con(Conj) then begin
      Conj^.net:=netK;
      akj:=get_s_and_remove(L,Conj^.s_start);
      new(u);
      num :=prp(u,num1,akj^.z);
      new(u);
      num3:=prp(u,num2,akj^.z);

      aij:=Conj^.s_start;
      aik:=ConL^.s_start;
      if aij^.z <> nil then supr(aij^.z,aik^.z,num);
      for i:=2 to sizeb do begin
        aij:=aij^.next_s;
        aik:=aik^.next_s;
        if aij^.z <> nil then supr(aij^.z,aik^.z,num);
      end;{ for i}

      for i:=1 to sizea do begin
        if i=1 then begin
          biL:=Conk^.s_start;
          new(start);
          aij:=start;
        end else begin
          biL:=biL^.next_s;
          new(aij^.next_s);
          aij:=aij^.next_s;
        end;
        aij^.next_s:=Conj^.s_start;
        if biL^.z <> nil then begin
          new(u);
          aij^.z:=prp(u,biL^.z,num3)
        end else aij^.z:=nil;
      end;{for i}
      if sizea > 0 then Conj^.s_start:=start;
    end;{if conj}
  end;{for j}
  dispose_net(netL);
  netK^.number_of_con:=sizea+sizeb;
  if debug then write_list0(netK,'k');
end; {join_net}

procedure reduce_net;
{Join connectors from the same networks}
var
  akj,akk,aij,aik,aLL,aiL,akL,aLk,aLj : s_param;
  num1,num2,num3,num4 : complex;
  ConL,Conj : conn;
  u,u1      : complex;
  i,k,L     : integer;
begin
  k:=get_kL_from_con(netK,Conk);
```

```
  L:=get_kL_from_con(netK,Conk^.mate);
  if k < L then begin i:=k;k:=L;L:=i;end;
   Conk := get_c_and_remove(k,netK^.con_start);
   ConL := get_c_and_remove(L,netK^.con_start);
   akk  := get_s_and_remove(k,Conk^.s_start);
   aLk  := get_s_and_remove(L,Conk^.s_start);
   akL  := get_s_and_remove(k,ConL^.s_start);
   aLL  := get_s_and_remove(L,ConL^.s_start);
   new(u);
   new(u1);
   num1 :=rc(di(prp(u1,di(co1,akL^.z),di(co1,aLk^.z)),prp(u,aLL^.z,akk^.z)));
    Conj:=netK^.con_start;
    while Conj <> nil do begin
    if calc_con(Conj) then begin
     akj := get_s_and_remove(k,Conj^.s_start);
     aLj := get_s_and_remove(L,Conj^.s_start);
      num4:=co(0.0,0.0);
      supr(num4,akj^.z,di(co1,aLk^.z));
      supr(num4,aLj^.z,akk^.z);
      new(u);
      num2:=prp(u,num1,num4);
      num4:=co(0.0,0.0);
      supr(num4,aLj^.z,di(co1,akL^.z));
      supr(num4,akj^.z,aLL^.z);
      new(u);
      num3:=prp(u,num1,num4);

     aij:=Conj^.s_start;
     aiL:=ConL^.s_start;
     aik:=Conk^.s_start;
     if aij^.z<>nil then begin
       supr(aij^.z,aiL^.z,num2);
       supr(aij^.z,aik^.z,num3);
     end;
     while aij^.next_s<> nil do begin
       aij:=aij^.next_s;
       aiL:=aiL^.next_s;
       aik:=aik^.next_s;
       if aij^.z<>nil then begin
         supr(aij^.z,aiL^.z,num2);
         supr(aij^.z,aik^.z,num3);
       end;
     end;{aij}
     end;{if conj}
     Conj:=Conj^.next_con;
   end;{Conj}
   netK^.number_of_con:=netK^.number_of_con-2;
end; {reduce net}

procedure rem_node(tnet : net);
{Remove nodes in network with 2 ports}
var
  tcon : conn;
```

```
  i,j  : integer;
begin
  ccon:=tnet^.con_start;
  if tnet^.ports_connected =2 then begin {2 port node connect to two ports}
    for j:=1 to 2 do begin
      if j=1 then ccon:=tnet^.con_start else ccon:=ccon^.next_con;
      for i:=1 to 2 do begin
        if i=1 then begin
          new(ccon^.s_start);
          c_s:=ccon^.s_start;
        end else begin
          new(c_s^.next_s);
          c_s:=c_s^.next_s;
        end;
        new(c_s^.z);
        if i<>j then c_s^.z^.r:=-1.0 else c_s^.z^.r:= 0.0;
        c_s^.z^.i:=0.0;
      end;{i}
      c_s^.next_s:=nil;
    end;{j}
  end else begin
    if tnet^.ports_connected > 0 then begin{if node is connected to port}
      if ext_port(ccon) then begin
        tcon:=ccon^.next_con^.mate;
        tcon^.port_type:=ccon^.port_type
      end else begin
        tcon:=ccon^.mate;
        tcon^.port_type:=ccon^.next_con^.port_type;
      end;
      tcon^.mate:=nil;
      tcon^.net^.ports_connected:=tnet^.ports_connected;
    end else begin
      ccon^.mate^.mate:=ccon^.next_con^.mate;
      ccon^.next_con^.mate^.mate:=ccon^.mate;
    end;
    dispose_net(tnet);
  end;
  if debug then writeln(lst,' rem_Node ');
end; {rem_node}

procedure set_up_element(ports : integer);
{Set up s-parameters for each network}
var
  i,j,jj : integer;
  pta    : array[1..max_net_size] of integer;
  onlyinp,onlyout : array[1..max_net_size] of boolean;
begin
  if ports=1 then begin {open or short}
    new(cnet^.con_start^.s_start);
    c_s:=cnet^.con_start^.s_start;
    c_s^.next_s:=nil;
    if cnet^.grounded then c_s^.z:=co(-one,0.0)  { short }
                      else c_s^.z:=co(1.0 ,0.0); { open  }
```

```
end else begin
  jj:=1;
  for j:=1 to ports do begin {check to see if input or output port}
    if j=1 then ccon:=cnet^.con_start else ccon:=ccon^.next_con;
    if ext_port(ccon) then begin
      pta[jj]:=ccon^.port_type;
      if debug then writeln(lst,'jj ',pta[jj]:4,cnet^.number_of_con:3);
      onlyout[jj]:=out[pta[jj]] and not(inp[pta[jj]]);
      onlyinp[jj]:=inp[pta[jj]] and not(out[pta[jj]]);
      if not(out[pta[jj]] or inp[pta[jj]]) then begin
        ccon:=get_c_and_remove(jj,cnet^.con_start);
        cnet^.number_of_con:=cnet^.number_of_con-1;
        jj:=jj-1;
      end;{if not(out..)}
    end else begin
      onlyout[jj]:=false;
      onlyinp[jj]:=false;
    end;{if ext}
    jj:=jj+1;
  end;{j}

  for j:=1 to cnet^.number_of_con do begin
    if j=1 then ccon:=cnet^.con_start else ccon:=ccon^.next_con;
    for i:=1 to cnet^.number_of_con do begin
      if i=1 then begin
        new(ccon^.s_start);
        c_s:=ccon^.s_start;
      end else begin
        new(c_s^.next_s);
        c_s:=c_s^.next_s;
      end;
      if onlyinp[i] or onlyout[j] then begin
        c_s^.z:=nil;
        if debug then writeln(lst,'net nil',i:3,pta[i]:3,' ',inp[pta[i]]);
      end else begin
        new(c_s^.z);
        if cnet^.node then begin
          c_s^.z^.i:=0.0;
          if cnet^.grounded then begin {if grounded}
            if i=j then c_s^.z^.r:=-one else c_s^.z^.r:= 0.0;
          end else begin
            if i=j then c_s^.z^.r:=one*((2/ports)-1)
                   else c_s^.z^.r:=one*(2/ports);
          end;{if grounded}
        end;{if tee etc}
      end;{if onlyinp}
    end;{i}
    c_s^.next_s:=nil;
  end;{j}
  if debug then begin
    write_list0(cnet,'c');
    writeln(lst,' set_up_elements');
  end;{if debug}
```

```
  end;{if 1 port}
end; {set_up_element}

procedure fill_compts;
{Transfer s-parameters from parts to networks}
var
  i,j,ii,jj : integer;
  v_s       : s_param;
  coni,conj : conn;
begin
  action:=false;     pars_compt_list;
  cnet:=nil;
  if not(bad_compt) then
  repeat
    if cnet=nil then cnet:=net_start else cnet:=cnet^.next_net;
    if not(cnet^.node) and (cnet^.number_of_con > 0) then begin
      coni:=nil;
      repeat
        if coni=nil then coni:=cnet^.con_start else coni:=coni^.next_con;
        ii:=coni^.conn_no;
        conj:=nil;
        repeat
          if conj=nil then begin
            conj:=cnet^.con_start;
            c_s:=coni^.s_start;
          end else begin
            conj:=conj^.next_con;
            c_s:=c_s^.next_s;
          end;
          jj:=conj^.conn_no;
          v_s:=nil;
          if c_s^.z <> nil then begin
            for i:=1 to cnet^.com^.number_of_con do
            for j:=1 to cnet^.com^.number_of_con do begin
              if v_s=nil then v_s:=cnet^.com^.s_begin else v_s:=v_s^.next_s;
              if (ii=i) and (jj=j) then begin
                c_s^.z^.r:=v_s^.z^.r;
                c_s^.z^.i:=v_s^.z^.i;
              end;{if ii}
            end;{for j}
          end;{if c_s}
        until conj^.next_con=nil;
      until coni^.next_con=nil;
      if debug then write_list0(cnet,'c');
    end;{if not(cnet^.node)};
  until cnet^.next_net=nil;
  if debug then writeln(lst,' fill_compts ');
end; {fill_compts}

procedure copy_circuit_and_fill_s;
{Procedure for copying circuit and filling s-parameters}
var
  i : integer;
```

```
begin
  if debug then for i:=1 to min_ports do writeln(lst,'**',inp[i]:6,out[i]:6);
  if xpt=0 then begin
    for i:=1 to 2 do begin {set_up_nets}
      cnet:=nil;
      repeat
        if cnet=nil then cnet:=net_start else cnet:=cnet^.next_net;
        with cnet^ do
        if i=1 then begin
        if((number_of_con=2)and node and not(grounded)and(ports_connected<>2))
                then rem_node(cnet);
        end else set_up_element(number_of_con);
      until cnet^.next_net=nil;
    end;{for i}
    if debug then writeln(lst,' set_up_nets ');
  end;{set_up_nets and do freq indept stuff}
  copy_networks(net_start_ptr2,xpt=0);
  fill_compts;
end; {copy_circuit_and_fill_s}

procedure analysis;
{Main procedure for directing the analysis}
label
  exit_analysis;
var
  i,j : integer;
begin
  filled_OK:=false;
  if net_start = nil then begin
    message[1]:='No circuit';
    message[2]:='to analyze';
    write_message;
  end else begin
    get_s_and_f0;
    if bad_compt then begin
      write_message;
      cx:=ccompt^.x_block;
    end else begin
      filled_OK:=true;
      Textcolor(lightgray);gotoxy(32,12);write(' Press s to stop ');
      Textcolor(white);    gotoxy(39,12);write('s');
      old_net_start:=net_start;
      old_net:=cnet;
      mark(ptrall);
      if debug then begin dump0;wso(ptrall^,2)end;
      copy_networks(net_start_ptr1,true);
      for xpt:=0 to npts do begin
        {writeln(lst,xpt:6,memavail:19);}
        freq:=fmin+xpt*finc;
        mark(ptrvar);
        copy_circuit_and_fill_s;
        for i:=1 to max_params do for j:=1 to max_params do sa[i,j]:=nil;
        if bad_compt then filled_OK:=false else begin
```

```
          if debug then dump0;
          while internal_joint_remaining do {loop over joints}
            if netK=netL then reduce_net {join connectors on same net}
                         else join_net;  {join two nets}
        end;
        fill_sa;
        if debug then begin dump0; write_list0(net_start,'a') end;
        if xpt > 0 then release(ptrvar);
        if keypressed then begin
          read(kbd,chs);
          if chs in ['s','S'] then begin
            filled_OK:=false;
            message[2]:='      STOP       ';
            write_message;
            goto exit_analysis;
          end;
          beep;
        end;{if keypreseed}
      end;{xpt}
      exit_analysis:
      if debug then wso(ptrall^,2);
      copy_networks(net_start_ptr1,false);
      release(ptrall);
      cnet:=old_net;
      net_start:=old_net_start;
      if debug then dump0;
    end;{if bad_compt}
  end;{if cnet}
end; {analysis}
```

Appendix B

Assembly language additions to *Puff*

```
;Print Screen Utility written by Authur E. Sheiman
;=======AES - 15:58 01/29/87  - print screen for mode 010H.
;=======AES - 01/13/87  - From Byte 1986 extra edition - inside the PC.
LPPRO   EQU     0               ;IBM Proprinter.
LPLJA   EQU     1               ;HP Laserjet+.
LPITOH  EQU     2               ;ITOH.
LPTYPE  EQU     LPITOH
;LPTYPE  EQU     LPLJA           ;Assemble for Laserjet+.
;=======INTERRUPT VECTOR STRUCTURE
VECTOR  STRUC
REGIP   DW      ?
REGCS   DW      ?
VECTOR  ENDS
;=======KEYBOARD SHIFT FLAG RECORD
RIGHT   EQU     00000001B
;=======CODE SEGMENT
CODE    SEGMENT
;*******EXECUTION STARTS WITH THIS PIECE
        ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
; PROGRAM SEGMENT PREFIX
;=======PROGRAM SEGMENT PREFIX
                        INT     020H            ;INT 020H
TOP_OF_MEMORY           DW      ?
                        DB      ?               ;IOCTL
                        DB      5 DUP(?)        ;DOS_JMP
TERMINATE               VECTOR  <>
CTRL_BREAK              VECTOR  <>
CRITICAL_ERROR          VECTOR  <>
                        DW      ?               ;USED BY DOS
                        DB      20 DUP(?)       ;USED BY DOS
ENVIRONMENT             DW      ?
                        DW      7 DUP(?)        ;USED BY DOS
                        DB      20 DUP(?)       ;UNUSED ?
DOS_CALL                PROC    FAR
                        INT     021H            ;INT 021H
                        RET                     ;RET
DOS_CALL                ENDP
                        DB      9 DUP(?)        ;UNUSED ?
FORMATTED_AREA_1        DB      16 DUP(?)
FORMATTED_AREA_2        DB      16 DUP(?)
                        DB      4 DUP(?)        ;UNUSED
UNFORMATTED_LENGTH      DB      ?
UNFORMATTED_AREA        DB      127 DUP(?)
;
; LABEL FOR END STATEMENT
IP      LABEL   NEAR
        JMP     START
;*******INTERRUPT INTERCEPT PIECE
        ASSUME  CS:CODE,DS:NOTHING,ES:NOTHING
        ASSUME  SS:NOTHING
ORG05   VECTOR  <>              ;ORIGINAL INT
                                ; 005H VECTOR
VINUSF  DB      ?               ;In use flag.
```

```
VX      DW      ?               ;For x := 0 to 79D.
VY      DW      ?               ;For y := 349D downto 0.
VY80    DW      ?               ;80*VY.
VY80X   DW      ?               ;80*VY+VX.

LFF     DB 00CH                 ;FF.
        DB 000H                 ;Termination.

        IFE LPTYPE-LPPRO        ;If Proprinter.
LPTSET  DB 01BH,041H,008H       ;LF to 8/72.
        DB 01BH,032H            ;Variable linefeed mode.
        DB 000H                 ;Termination.
LINTCH  DB 01BH,04BH,94D,1D     ;480 bit graphics 350 printed.
        DB 000H                 ;Termination.
LCRLF   DB 00DH,00AH            ;CR,LF.
        DB 000H                 ;Termination.
LSIGN   DB 00DH,00AH            ;Signon.
        DB 'EGA print screen utility. '
        DB 'Version: Proprinter 1.00.  A.E.S. (c) Caltech 1987.'
        DB 00DH,00AH
        DB 024H                 ;DOS termination.
        ENDIF

        IF LPTYPE-LPLJA         ;If Laserjet+.
LPRSET  DB 01BH,'E'             ;Laserjet+ serial reset needed.
        DB 000H                 ;Termination.
LPTSET  DB 01BH,'*t100R'        ;100 dpi.
        DB 000H                 ;Termination.
LINTCH  DB 01BH,'*r1A,01BH,'*b80W'      ;Graphics.
        DB 000H                 ;Termination.
LEXIT1  DB 01BH,'*rB',01BH,'*p' ;Exit graphics and step part 1.
        DB 000H                 ;Termination.
LEXIT2  DB 01BH,'Y'             ;Exit graphics and step part 2.
        DB 000H                 ;Termination.
LSIGN   DB 00DH,00AH            ;Signon.
        DB 'EGA print screen utility. '
        DB 'Version: Laserjet+ X.00.  A.E.S. (c) Caltech 1987.'
        DB 00DH,00AH
        DB 024H                 ;DOS termination.
VVINX   DB 10 DUP (?)           ;ASCII vertical index.
        DB 000H                 ;Termination.
        ENDIF

        IFE LPTYPE-LPITOH       ;If ITOH.
LPTSET  DB 01BH,'N'             ;10 cpi (80 dpi).
        DB 01BH,'<'             ;Bi-directional. {Uni - '>'}
        DB 01BH,'T12'           ;12/144 inch spacing between lines.
        DB 000H                 ;Termination.
LINTCH  DB 01BH,'S0350'         ;Graphics - 700 bytes follow.
        DB 000H                 ;Termination.
LCRLF   DB 00DH,00AH            ;CR,LF.
        DB 000H                 ;Termination.
LSIGN   DB 00DH,00AH            ;Signon.
```

```
        DB 'EGA print screen utility. '
        DB 'Version: ITOH 1.00.  A.E.S. (c) Caltech 1987.'
        DB 00DH,00AH
        DB 024H                 ;DOS termination.
        ENDIF

;-------INTERCEPT ROUTINE
INT05   PROC    FAR
        STI                     ;Enable interrupts.
        PUSH    AX              ;Save registers.
        PUSH    BX
        PUSH    CX
        PUSH    DX
        PUSH    BP
        PUSH    SI
        PUSH    DI
        PUSH    DS
        PUSH    ES
; Examine mode - if not mode 10H, go to old handler, else process.
        MOV     AH,15D          ;Get VDG state.
        INT     010H
        CMP     AL,010H         ;Mode 010H?
        JZ      BMOD10          ;Yes, process mode 10H.
; DO ORIGINAL INT 005H
        POP     ES
        POP     DS
        POP     DI
        POP     SI
        POP     BP
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        JMP     ORG05
; Process mode 10H.
BMOD10: MOV     AX,CS           ;Set segment registers.
        MOV     DS,AX
        MOV     ES,AX
        ASSUME DS:CODE,ES:CODE
        CLI                     ;No interrupts.
        TEST    VINUSF,0FFH     ;In use?
        JZ      BM1000          ;No.
        STI                     ;Is in use - reenab ints and exit.
        JMP     INT05X
BM1000: MOV     AL,-1           ;Set the in use flag.
        MOV     VINUSF,AL
        STI                     ;Interrupts go.
        MOV     AH,1            ;Reset the LPT.
        XOR     DX,DX           ;LPT0.
        INT     017H
        IFE LPTYPE-LPLJA        ;Laserjet+ needs ESC,E.
        MOV     BX,OFFSET LPRSET        ;Reset LPT
        CALL    MLPT            ;Error is exit error.
```

```
        ENDIF
; Check that printer is ready to go.
        MOV     CX,8000H        ;Must delay for printer after reset.
BMWFP:  XOR     DX,DX           ;Printer 0.
        MOV     AH,2            ;Get status. Pro-printer returns:
        INT     017H            ; 00 - offline; 90 - online.
        XOR     AH,090H         ;Invert bits, so all bits are errors.
        TEST    AH,0B0H         ;Busy or out of paper?
        JZ      BM1001          ;No, all okay.
        LOOP    BMWFP           ;Wait for printer.
BMEREX: XOR     AL,AL           ;Error exit - stop in use and exit.
        MOV     VINUSF,AL
        JMP     INT05X
BM1001: MOV     BX,OFFSET LPTSET        ;Set up LPT
        CALL    MLPT            ;Error is exit error.

        IFE LPTYPE-LPPRO        ;If Proprinter.
; Print loop here.
        XOR     AX,AX           ;For VX := 0 to 79D.
        MOV     VX,AX
BM1002: MOV     BX,OFFSET LINTCH        ;Initial string.
        CALL    MLPT            ;Error (off-line to abort) quits.
        MOV     AX,349D         ;For VY := 349D downto 0.
        MOV     VY,AX
        MOV     AX,349D*80D     ;Set up VY80.
        MOV     VY80,AX
BM1003: MOV     AX,VY80         ;VY80X = 80*VY+VX.
        ADD     AX,VX
        MOV     VY80X,AX
        XOR     DL,DL           ;DL=0.  DL = byte to send.
        MOV     CX,4            ;Loop 4 times.
BM1004: MOV     AL,4            ;Port $3CE := 4.
        PUSH    DX              ;Save DX.
        MOV     DX,03CEH
        OUT     DX,AL
        MOV     AL,CL           ;Compute 3..0.
        DEC     AL
        MOV     DX,03CFH        ;Send to port.
        OUT     DX,AL
        POP     DX              ;Restore DX.
        MOV     AX,VY80X        ;AX = 80*VY+VX.
        PUSH    ES              ;Get VDG byte.
        MOV     BX,0A000H
        MOV     ES,BX
        MOV     BX,AX
        MOV     AH,ES:[BX]
        POP     ES
        OR      DL,AH           ;Or this page in.
        LOOP    BM1004
        CALL    MCHLPT          ;Send DL to the printer. Error exit.
        MOV     AX,VY           ;VY = 0?
        OR      AX,AX
        JZ      BM1005          ;Yes.
```

```
        DEC     AX                  ;VY := VY-1.
        MOV     VY,AX
        MOV     AX,VY80             ;VY80 := VY80-80.
        SUB     AX,80D
        MOV     VY80,AX
        JMP     BM1003              ;Next VY.
BM1005: MOV     BX,OFFSET LCRLF ;Send CR,LF.
        CALL    MLPT                ;Error is exit error.
        MOV     AX,VX               ;VX = 79D?
        CMP     AX,79D
        JZ      BM1006              ;Yes done.
        INC     AX                  ;VX := VX+1.
        MOV     VX,AX
        JMP     BM1002
        ENDIF

        IFE LPTYPE-LPLJA            ;If Laserjet+.
; Print loop here.
        XOR     AX,AX               ;For VY := 0D to 349D.
        MOV     VY,AX
        MOV     VY80,AX             ;VY80=VY*80.
BM1002: MOV     BX,OFFSET LINTCH        ;Initial string.
        CALL    MLPT                ;Error (off-line to abort) quits.
        XOR     AX,AX               ;For VX := 0D to 79D.
        MOV     VX,AX
BM1003: MOV     AX,VY80             ;VY80X = 80*VY+VX.
        ADD     AX,VX
        MOV     VY80X,AX
        XOR     DL,DL               ;DL=0.  DL = byte to send.
        MOV     CX,4                ;Loop 4 times.
BM1004: MOV     AL,4                ;Port $3CE := 4.
        PUSH    DX                  ;Save DX.
        MOV     DX,03CEH
        OUT     DX,AL
        MOV     AL,CL               ;Compute 3..0.
        DEC     AL
        MOV     DX,03CFH            ;Send to port.
        OUT     DX,AL
        POP     DX                  ;Restore DX.
        MOV     AX,VY80X            ;AX = 80*VY+VX.
        PUSH    ES                  ;Get VDG byte.
        MOV     BX,0A000H
        MOV     ES,BX
        MOV     BX,AX
        MOV     AH,ES:[BX]
        POP     ES
        OR      DL,AH               ;Or this page in.
        LOOP    BM1004
        CALL    MCHLPT              ;Send DL (unused) to the printer. Error exit.
        MOV     AX,VX               ;VX = 79D?
        CMP     AX,349D
        JZ      BM1005              ;Yes.
        INC     AX                  ;VX := VX+1.
```

```
        MOV     VX,AX
        JMP     BM1003          ;Next VX.
BM1005: ;Compute variable index.
        MOV     BX,OFFSET LEXIT1        ;Send part 1.
        CALL    MLPT            ;Error is exit error.
        MOV     BX,OFFSET VVINX ;Send computed #.
        CALL    MLPT
        MOV     BX,OFFSET LEXIT2        ;Send part 2.
        CALL    MLPT            ;Error is exit error.
        MOV     AX,VY           ;VY = 349D?
        CMP     AX,349D
        JZ      BM1006          ;Yes done.
        INC     AX              ;VY := VY+1.
        MOV     VX,AX
        MOV     AX,VY80         ;VY80 := VY80+80.
        ADD     AX,80D
        MOV     VY80,AX
        JMP     BM1002
        ENDIF

        IFE LPTYPE-LPITOH       ;If ITOH.
; Print loop here.
        MOV     AX,79D          ;For VX := 79D downto 0D.
        MOV     VX,AX
BM1002: MOV     BX,OFFSET LINTCH        ;Initial string.
        CALL    MLPT            ;Error (off-line to abort) quits.
        XOR     AX,AX           ;For VY := 0D to 349D.
        MOV     VY,AX
       ;MOV     AX,0D*80D       ;Set up VY80.
        MOV     VY80,AX
BM1003: MOV     AX,VY80         ;VY80X = 80*VY+VX.
        ADD     AX,VX
        MOV     VY80X,AX
        XOR     DL,DL           ;DL=0.  DL = byte to send.
        MOV     CX,4            ;Loop 4 times.
BM1004: MOV     AL,4            ;Port $3CE := 4.
        PUSH    DX              ;Save DX.
        MOV     DX,03CEH
        OUT     DX,AL
        MOV     AL,CL           ;Compute 3..0.
        DEC     AL
        MOV     DX,03CFH        ;Send to port.
        OUT     DX,AL
        POP     DX              ;Restore DX.
        MOV     AX,VY80X        ;AX = 80*VY+VX.
        PUSH    ES              ;Get VDG byte.
        MOV     BX,0A000H
        MOV     ES,BX
        MOV     BX,AX
        MOV     AH,ES:[BX]
        POP     ES
        OR      DL,AH           ;Or this page in.
        LOOP    BM1004
```

```
        CALL    MCHLPT          ;Send DL (unused) to the printer. Error exit.
        MOV     AX,VY           ;VY = 349D?
        CMP     AX,349D
        JZ      BM1005          ;Yes.
        INC     AX              ;VY := VY+1.
        MOV     VY,AX
        MOV     AX,VY80         ;VY80 := VY80+80.
        ADD     AX,80D
        MOV     VY80,AX
        JMP     BM1003          ;Next VY.
BM1005: MOV     BX,OFFSET LCRLF ;Send CR,LF.
        CALL    MLPT            ;Error is exit error.
        MOV     AX,VX           ;VX = 0D?
        OR      AX,AX
        JZ      BM1006          ;Yes done.
        DEC     AX              ;VX := VX-1.
        MOV     VX,AX
        JMP     BM1002
        ENDIF

BM1006: MOV     BX,OFFSET LFF   ;Send FF.
        CALL    MLPT            ;Error is exit error.
        MOV     AH,1            ;Reset the LPT.
        XOR     DX,DX           ;LPT0.
        INT     017H
        IFE LPTYPE-LPLJA        ;Laserjet+ needs ESC,E.
        MOV     BX,OFFSET LPRSET        ;Reset LPT
        CALL    MLPT            ;Error is exit error.
        ENDIF
        JMP     BMEREX          ;Legal exit - but treat as error to clear
                                ; the in use flag.

;*** NOTE STACK MUST BE BALANCED FOR CALLS TO MCHLPT AND MLPT ***
; MCHLPT - Send DL to LPT.  *** - ONE LEVEL CALL ONLY ***.
;           Does not use DL.
MCHLPT  PROC    NEAR
        MOV     AL,DL
        XOR     AH,AH
        XOR     DX,DX           ;LPT0.
        INT     017H            ;Stat: 10 - no error.
        MOV     DL,AL           ;Restore DL.
        TEST    AH,029H         ;Printer error?
        JZ      BBHLT0          ;No error.
        POP     AX              ;Balance stack, error exit.
        JMP     BMEREX
BBHLT0: RET
MCHLPT  ENDP

; MLPT - Send string at BX to LPT.  *** - ONE LEVEL CALL ONLY ***.
MLPT    PROC    NEAR
        MOV     AL,[BX]
        OR      AL,AL           ;000H terminator?
        JZ      BLPT1           ;Yes, done.
```

```
        XOR     AH,AH
        XOR     DX,DX
        INT     017H            ;Stat: 10 - no error.
                                ; IN7 017H preserves BX.
        TEST    AH,029H         ;Printer error?
        JZ      BLPTO           ;No error.
        POP     AX              ;Balance stack, error exit.
        JMP     BMEREX
BLPTO:  INC     BX              ;Next.
        JMP     MLPT
BLPT1:  RET
MLPT    ENDP

INTO5X  LABEL   NEAR            ;Return from interrupt - restore.
        POP     ES
        POP     DS
        POP     DI
        POP     SI
        POP     BP
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        IRET
INTO5   ENDP
;*******INITIALIZATION PIECE
        ASSUME  CS:CODE,DS:CODE,ES:CODE,SS:CODE
START   LABEL   NEAR
; Print signon message
        MOV     AH,009H         ;Signon.
        MOV     DX,OFFSET LSIGN
        INT     021H
;-------GET INT 5 VECTOR AND SAVE IN ORGO5
        PUSH    ES
        MOV     AH,035H
        MOV     AL,005H
        INT     021H
        ASSUME  ES:NOTHING
        MOV     ORGO5.REGIP,BX
        MOV     ORGO5.REGCS,ES
        POP     ES
        ASSUME  ES:CODE
;-------Clear in use flag.
        XOR     AL,AL
        MOV     VINUSF,AL
;-------SET INT 5 VECTOR TO INTO5
        MOV     AH,025H
        MOV     AL,005H
        MOV     DX,OFFSET INTO5
        INT     021H
;-------FREE MEMORY ALLOCATED TO THE ENVIRONMENT
        PUSH    ES
        MOV     AH,049H
```

```
        MOV     ES,ENVIRONMENT
        ASSUME  ES:NOTHING
        INT     021H
        POP     ES
        ASSUME  ES:CODE
;-------TERMINATE PROTECTING MEMORY BELOW START
        MOV     DX,OFFSET START
        INT     027H
CODE    ENDS
        END     IP      ;NOTE IP FOR EXE2BIN
```

```
;PUFFASM.ASM
;Contains external function Prp and Supr
;*******************************************************************
;To assemble
;\asm\masm %1 /r;
;if errorlevel 1 goto :quit
;link %1;
;exe2bin %1.exe %1.com
;:quit
;*******************************************************************
code   segment                  ;complex utilities
       assume   cs:code
pass   proc     near

jmp prp                            ;Prp  u:=z1*z2   call u:=prp(u,z1,z2)
jmp supr                           ;Supr u:=u+z1*z2 call supr(u,z1,z2)

prp:                               ;begin Prp
       push     bp                 ;save environment
       mov      bp,sp

       mov bx,[bp+4]
       mov dx,[bp+6]
       mov es,dx
       fld qword ptr es:[bx]       ;x2
       fld qword ptr es:[bx]+8     ;y2

       mov bx,[bp+8]
       mov dx,[bp+10]
       mov es,dx
       fld qword ptr es:[bx]+8     ;y1
       fld qword ptr es:[bx]       ;x1
       fxch st(3)

       mov ax,[bp+12]
       mov bx,ax
       mov dx,[bp+14]
       mov es,dx

                                   ;0        1       2  3  4  5  6  7
                                   ;x2       y1      y2 x1
       fld     st(0)               ;x2       x2      y1 y2 x1
       fmul    st,  st(4)          ;x2*x1    x2      y1 y2 x1
       fld     st(2)               ;y1       x2*x1   x2 y1 y2 x1
       fmul    st,  st(4)          ;y1*y2    x2*x1   x2 y1 y2 x1
       fsubp   st(1),st            ;x1*x2-y1*y2 x2   y1 y2 x1
       fstp qword ptr es:[bx]      ;x2       y1      y2 x1
       fmulp   st(1),st            ;x2*y1    y2      x1
       fxch    st(2)               ;x1       y2  x2*x1
       fmulp   st(1),st            ;x1*y2 x2*x1
       faddp   st(1),st            ;y2*x1+x2*y1
       fstp qword ptr es:[bx]+8

```

```
        mov     sp,bp               ;restore environment
        pop     bp
        ret     16

supr:                               ;begin Supr
        push    bp                  ;save environment
        mov     bp,sp

        mov bx,[bp+4]
        mov dx,[bp+6]
        mov es,dx
        fld qword ptr es:[bx]       ;x2
        fld qword ptr es:[bx]+8     ;y2

        mov bx,[bp+8]
        mov dx,[bp+10]
        mov es,dx
        fld qword ptr es:[bx]+8     ;y1
        fld qword ptr es:[bx]       ;x1
        fxch st(3)

        mov ax,[bp+12]
        mov bx,ax
        mov dx,[bp+14]
        mov es,dx

                                    ; 0          1       2  3  4  5  6  7
                                    ;x2         y1      y2 x1
        fld     st(0)               ;x2         x2      y1 y2 x1
        fmul    st,  st(4)          ;x2*x1      x2      y1 y2 x1
        fld     st(2)               ;y1         x2*x1   x2 y1 y2 x1
        fmul    st,  st(4)          ;y1*y2      x2*x1   x2 y1 y2 x1
        fsubp   st(1),st            ;x1*x2-y1*y2 x2     y1 y2 x1
        fadd qword ptr es:[bx]
        fstp qword ptr es:[bx]      ;x2         y1      y2 x1
        fmulp   st(1),st            ;x2*y1      y2      x1
        fxch    st(2)               ;x1         y2   x2*x1
        fmulp   st(1),st            ;x1*y2 x2*x1
        faddp   st(1),st            ;y2*x1+x2*y1
        fadd qword ptr es:[bx]+8
        fstp qword ptr es:[bx]+8

        mov     sp,bp               ;restore environment
        pop     bp
        ret     12

pass    endp
code    ends
        end
```