AN ABSTRACT OF THE THESIS OF

Leonard A. Hayden for the degree of Master of Science in Electrical and Computer Engineering presented on April 25, 1989.

Title: Nonuniformly Coupled Microstrip Transversal Filters for Analog Signal Processing

Redacted for Privacy

Abstract approved: _____
Vijai K. Tripathi

An analog signal processing technique allowing arbitrary step response is presented. A transversal structure is used with nonuniform contradirectional coupling to provide continuous tapping. Sinusoidal and chirp responses are presented as examples implemented using this nonuniformly coupled microstrip structure.

Nonuniformly Coupled Microstrip Transversal Filters for Analog Signal Processing

by

Leonard Hayden

A THESIS

submitted to

Oregon State University

in partial fulfillment of
the requirements for the
degree of

Master of Science

Completed May 4, 1989

Commencement June 1989

APPROVED:

Redacted for Privacy

Professor of Electrical and Computer Engineering in charge of major

Redacted for Privacy

Head of department of Electrical and Computer Engineering

Redacted for Privacy

Dean of Gradu

Date thesis is presented May 4, 1989

Typed by researcher for Leonard A. Hayden

TABLE OF CONTENTS

LIST OF FIGURES

NONUNIFORMLY COUPLED MICROSTRIP TRANSVERSAL FILTERS FOR ANALOG SIGNAL PROCESSING

INTRODUCTION

Nonuniform transmission lines have been used in the past to realize time domain waveforms in various measurement systems [1]. Delay lines with uniformly coupled sections as taps have also been used particularly at low temperatures for broadband superconductive analog signal processing circuits such as chirp filters [3-7].

Synthesis using nonuniform lines has been limited to discrete step changes in impedance which are analyzed using traditional ladder analysis [2] techniques. For coupled lines step changes in coupling act as frequency selective taps with resonances corresponding to the uniform coupled section equal to odd multiples of a quarter wavelength.

In this dissertation a method for analyzing continuous variations in coupling between transmission lines based on the relationship between coupling profile and time domain response is developed. The Fourier transform relationship between taper profile and frequency response for nonuniform transmission lines [8] is used together with Sharpe's equivalence principle [9] to realize a

coupled line transversal filter structure suitable for waveform synthesis and signal processing. It is shown that realizability limitations are minimal allowing generation of arbitrary, continuous, finite-length responses.

The applications of nonuniform coupling are demonstrated by numerical evaluation of representative practical chirp structures designed using a procedure compatible with computer aided design. In order to verify the theoretical results a set of simple experimental circuits were designed and tested. The experimental results are found to be in good agreement with theoretical predictions.

TIME DOMAIN ANALYSIS OF NONUNIFORM COUPLED LINES

In this chapter it is shown that the time domain response of nonuniformly coupled transmission lines is directly related to the coupling profile. The constant characteristic impedance coupled line problem is reduced to the analysis of the equivalent nonuniform transmission line associated with the even mode of excitation. This equivalence translates coupling profile into taper profile, the even mode impedance variation, which completely describes the coupled lines.

Nonuniform transmission lines are readily analyzed in the frequency domain through Bolinder's taper profile -- frequency response transform relation [8]. This analysis is extended to the time domain by identifying the elements in the transform relation that correspond to the impulse response. Upon integration a simpler step response -- taper profile relation is obtained.

Equivalent nonuniform transmission line

Sharpe's equivalence principle [8] asserts that an equivalent pair of dual nonuniform transmission lines exists for every nonuniform directional coupler that is electrically symmetric. It is shown that for symmetric nonuniform coupled lines with constant characteristic impedance, i.e., $Z_{oe}(z)Z_{oo}(z)=Z_o$ constant, the even

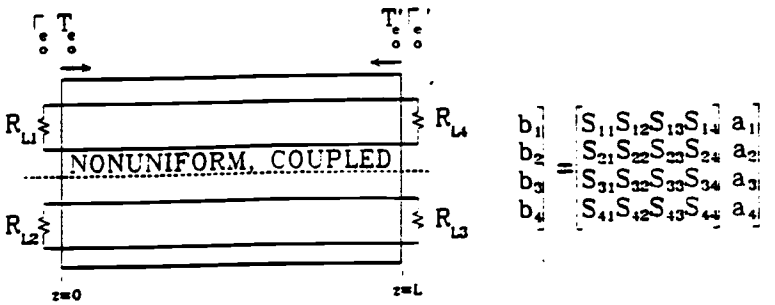
and odd mode characteristic lines are the dual equivalent transmission lines. This enables the even mode nonuniform transmission line to completely describe the system.

A linear, lossless, symmetric four-port network (see Fig.1) can be studied in terms of even and odd modes of excitation [10]. For the even mode of excitation ($a_{1e}=a_{2e}$), $b_{1e}=b_{2e}$, $b_{3e}=b_{4e}$, and $a_{3e}=a_{4e}$, and for the odd mode ($a_{1o}=-a_{2o}$), $b_{1o}=-b_{2o}$, $b_{4o}=-b_{3o}$, and $a_{4o}=-a_{3o}$ from symmetry. Writing explicitly the defining equations for S-parameters using the above relations yields,

$$\begin{aligned} b_{1e,o} &= (s_{11} \pm s_{12})\, a_{1e,o} + (s_{14} \pm s_{13})\, a_{4e,o} \\ &= (s_{22} \pm s_{21})\, a_{1e,o} + (s_{23} \pm s_{24})\, a_{4e,o} = \pm b_{2e,o} \\ b_{4e,o} &= (s_{41} \pm s_{42})\, a_{1e,o} + (s_{44} \pm s_{43})\, a_{4e,o} \\ &= (s_{32} \pm s_{31})\, a_{1e,o} + (s_{33} \pm s_{34})\, a_{4e,o} = \pm b_{3e,o}. \end{aligned} \tag{1}$$

In order to avoid confusion with two-port and four-port terms, the even and odd mode 2x2 S-parameters are written as

$$\begin{bmatrix} b_{1e,o} \\ b_{4e,o} \end{bmatrix} = \begin{bmatrix} \Gamma_{e,o} & T'_{e,o} \\ T_{e,o} & \Gamma'_{e,o} \end{bmatrix} \begin{bmatrix} a_{1e,o} \\ a_{4e,o} \end{bmatrix}. \tag{2}$$



Symmetry: $R_{L1} = R_{L2}$, $R_{L3} = R_{L4}$

Fig. 1 Nonuniformly coupled lines as a four-port network with associated S-parameters

Matching terms in (1) and (2) yields a system of equations with solution:

$$[S] = \tfrac{1}{2}\begin{bmatrix} \Gamma_e+\Gamma_o & \Gamma_e-\Gamma_o & T'_e-T'_o & T'_e+T'_o \\ \Gamma_e-\Gamma_o & \Gamma_e+\Gamma_o & T'_e+T'_o & T'_e-T'_o \\ T_e-T_o & T_e+T_o & \Gamma'_e+\Gamma'_o & \Gamma'_e-\Gamma'_o \\ T_e+T_o & T_e-T_o & \Gamma'_e-\Gamma'_o & \Gamma'_e+\Gamma'_o \end{bmatrix}. \qquad (3)$$

This equation will be used later for analysis of arbitrary structures.

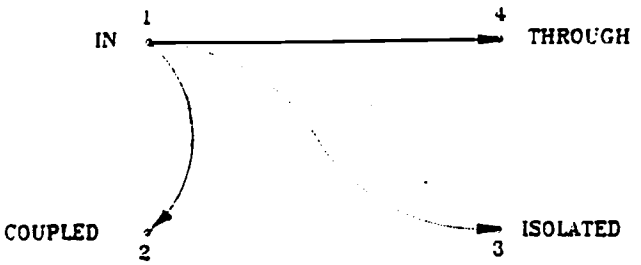


Fig. 2 Contradirectional coupling

The desired structure for synthesis is that of a matched four-port, or contradirectional coupler. The unitary property of lossless, linear networks requires that a matched four-port has one port isolated. In microstrip or stripline structures the coupling is backward-wave or contradirectional, see Fig.2. The condition of matching requires $s_{11}=s_{22}=s_{33}=s_{44}=0$, therefore

$$\Gamma_e = -\Gamma_o, \quad \Gamma'_e = -\Gamma'_o. \qquad (4)$$

Isolation requires $s_{31}=s_{42}=s_{13}=s_{24}=0$, therefore

$$T_e = T_o, \quad T'_e = T'_o. \qquad (5)$$

The simplified S-matrix is

$$[S] = \begin{bmatrix} 0 & \Gamma_e & 0 & T'_e \\ \Gamma_e & 0 & T'_e & 0 \\ 0 & T_e & 0 & \Gamma'_e \\ T_e & 0 & \Gamma'_e & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\Gamma_o & 0 & T'_o \\ -\Gamma_o & 0 & T'_o & 0 \\ 0 & T_o & 0 & -\Gamma'_o \\ T_o & 0 & -\Gamma'_o & 0 \end{bmatrix}. \quad (6)$$

The equivalent nonuniform transmission lines [9] are then

$$[S]_{2x2} = \begin{bmatrix} \pm S_{12} & S_{14} \\ S_{41} & \pm S_{34} \end{bmatrix} = \begin{bmatrix} \Gamma_{e,o} & T'_{e,o} \\ T_{e,o} & \Gamma'_{e,o} \end{bmatrix} \quad (7)$$

which are simply the even and odd mode transmission lines.

The nonuniform coupled line problem is now reduced to the synthesis of the nonuniform even mode transmission line.

Taper profile -- time-domain response relation

As early as 1950, Bolinder [8] identified a Fourier transform relation between a nonuniform transmission line's taper profile and frequency response. The taper profile is the characteristic impedance -- as a function of position along the line -- normalized by one of the termination impedances. This Fourier relation suggests that taper profile is directly related to time domain response.

Time domain reflectometry and lattice diagrams [2] use the relation suggested above for the stepped impedance case. Waveform shaping has been demonstrated for stepped impedance variation [1]. The relation for impedance continuously varying has been neglected although it is perhaps conceptually more fundamental.

The analysis of nonuniform transmission lines results in a nonlinear equation without known solution. The theory of small reflections is used to simplify to a linear relation [11]. Collin [11] makes the case that characteristic impedance loses its usefulness as a single defining parameter under conditions of rapid positional variation. Techniques that do not rely upon the transmission line approximation are necessary for the large reflection case. The starting point for the following derivation is the result from the theory of small reflections expressed in Bolinder's transform relation.

$$\Gamma(z) = \frac{1}{2}\int_{z}^{L} e^{-2jB(u-z)} \frac{d \ln Z}{du}\, du \tag{8}$$

where $Z(z)$ is the taper profile and $\Gamma(z)$ is the reflection coefficient along the line. In this analysis $\Gamma(z=0)=\Gamma_e$ and therefore $S_{21}(w)=\Gamma_e(w)=\Gamma(z=0)=H(w)$ where $H(w)$ is the transfer function of the network. With a variable change

$$H(w) = \frac{1}{2}\int_{0}^{L} e^{-2jBz} \frac{d \ln Z}{dz}\, dz \tag{9}$$

Using the TEM approximation of constant velocity, the exponential term is rewritten

$$\exp[-2j\beta z] = \exp[-jw(2z/v)]. \tag{10}$$

Let $t=2z/v$, then $dt=(2/v)dz$, $dz=(v/2)dt$, $t(z=0)=0$, and $t(z=L)=2L/v$.

$$H(w) = \int_{0}^{2L/v} e^{-jwt}\left[\frac{1}{2} \frac{d \ln Z}{dz} \frac{dz}{dt}\right] dt \tag{11}$$

which is the Fourier transform of the bracketed part,

$$h(t) = \tfrac{1}{2}[d(\ln Z)/dz]dz/dt \qquad 0 < t < 2L/v \tag{12}$$

which must be the impulse response function.

The response g(t) to a unit step u(t) is computed,

$$\begin{aligned} g(t) = h(t) * u(t) &= \int_{-\infty}^{+\infty} h(T)\, u(t-T)\, dT \\ &= \int_0^t h(T)\, dT = \tfrac{1}{2} \int_0^t \frac{d \ln Z}{dz} \frac{dz}{dT}\, dT \\ &= \tfrac{1}{2} \int_0^{vt/2} \frac{d \ln Z}{dz}\, dz = \tfrac{1}{2} \ln Z \Big|_{z=0}^{z=vt/2} \end{aligned} \tag{13}$$

and finally,

$$g(t) = \begin{cases} \tfrac{1}{2} \ln Z(z=vt/2) & 0 < t < 2L/v \\ \tfrac{1}{2} \ln Z(z=L) & t > 2L/v. \end{cases} \tag{14}$$

The step response is the taper profile (scaled appropriately).

Nonuniform coupled lines time domain response

For nonuniform coupled lines equation (14) applies to the even-mode line and the odd-mode line is chosen such that the characteristic impedance is constant (as previously described). That is,

$$g(t) = \begin{cases} \frac{1}{2} \ln Z_e(z=vt/2) & 0 < t < 2L/v \\ \frac{1}{2} \ln Z_e(z=L) & t > 2L/v. \end{cases} \quad (15)$$

A first order approximation can be made for loose coupling. Here $Z_e = 1 + \Delta Z_e$ where $\Delta Z_e \ll 1$. A truncated Taylor series expansion allows

$$\ln Z_e \cong \Delta Z_e \quad (16)$$

and

$$g(t) \cong \frac{1}{2} \Delta Z_e(z=vt/2) \qquad 0 < t < 2L/v. \quad (17)$$

Any change in the even-mode impedance profile directly affects the time-domain step response.

APPLICATION TO WIDE BANDWIDTH ANALOG SIGNAL PROCESSING

Signal processing applications range from waveshaping in digital electronics to sophisticated adaptive filters in communications systems. Modern system requirements are continuously pushing the performance requirements of the processing components. The emphasis in this dissertation is analog signal processing using electromagnetic delay lines. Use of transmission lines as the delay element has the advantage of extremely wide bandwidth although it is difficult to implement long time delays as signals travel at the speed of light.

An application of interest is the linear-frequency-modulated filter. This filter is also known as a 'chirp' filter since the step response is a linear sweep (up or down) in frequency. Chirp filters are used in radar systems for pulse compression, see Fig. 3.

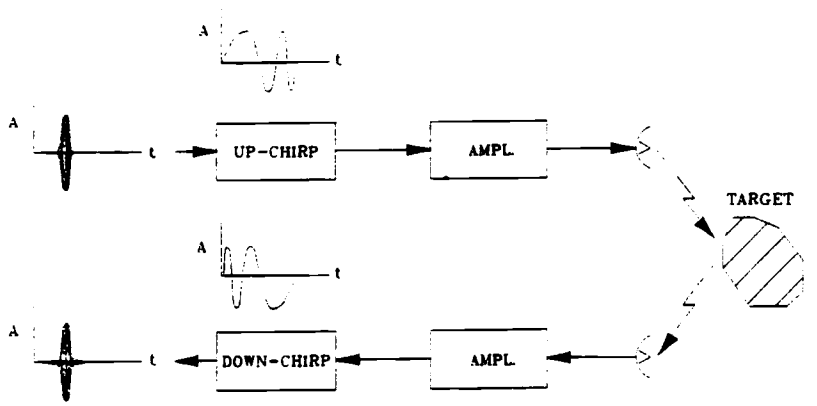


Fig. 3 Use of chirp filters for radar systems

The radar pulse is first expanded in time by the dispersive up-chirp filter. This allows greater gain to be used in the following stages since instantaneous peak power is reduced. The return echo is applied to a down-chirp which acts as a matched filter and compresses the pulse. System dynamic range is improved.

Greater range resolution is achieved by shorter radar pulses, therefore wide bandwidth is essential. Surface acoustic wave devices are used but are limited in bandwidth to less than 1 GHz [12]. Transmission line structures achieve dramatically wider bandwidth [6].

Superconductive tapped delay lines

The work at Lincoln Laboratory, Massachusetts Institute of Technology [3-7], has demonstrated practical chirp filters using superconductive tapped delay lines. Contradirectional couplers are used as the taps, see Fig. 4. Each coupler is thought to be quarter-wave resonant at progressively higher frequencies. The higher frequency has longer round trip from input to output. A linear group delay versus frequency (quadratic phase) relationship is created. This is the characteristic of a chirp filter.

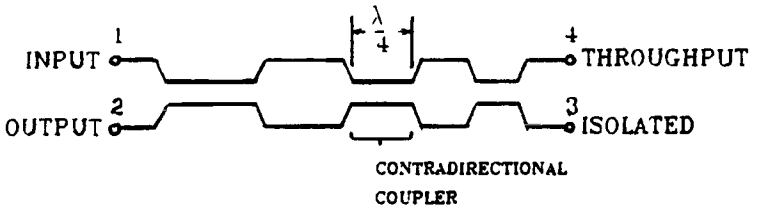


Fig. 4 Chirp filter using cascaded contradirectional couplers

An alternative analysis follows from the results of the previous section of this paper. Each coupling section results in a step change in the time domain response. The step response is simply a squared (signum) version of the chirp response [3-7],

$$g(t) = \mathrm{sgn}\{\sin[(ut+w)t]\} \tag{18}$$

where w is the initial frequency and u is the chirp slope.

The squaring of the response suggests that the filter bandwidth is limited by the 3/4 wavelength reentrance and that an additional filter is required to strip off the higher frequency components. This extra filter is mentioned in [3].

More recent reports [7] indicate that the discontinuity in coupler strength generates distortions in the phase response and amplitude ripple. Continuous variation in coupling will eliminate these effects.

Continuous tapping using nonuniform coupling

As a basis for general signal processing the architecture for the transversal filter is presented in Fig. 5. The output is a weighted sum of samples taken at T intervals. An alternative structure can be envisioned with delay lines in both the input and output signal paths, see Fig. 6. If the taps are bidirectional then small weighting values are required to minimize infinite impulse response behavior.

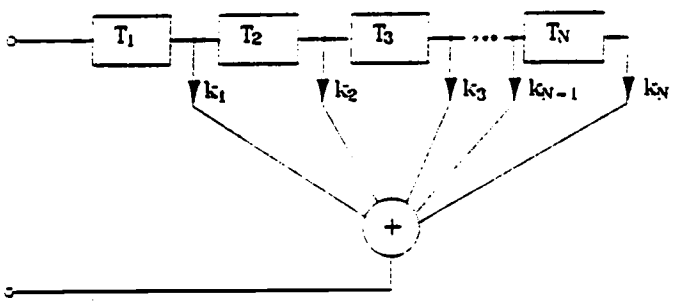


Fig. 5 Transversal filter architecture

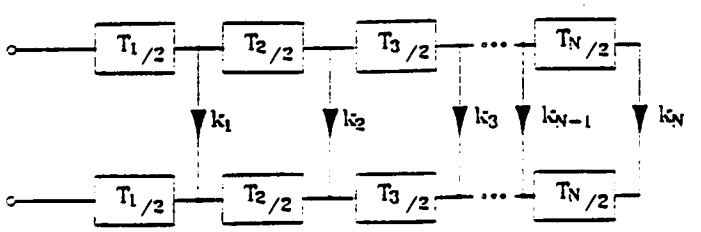


Fig. 6 Alternative transversal filter architecture

A nonuniform coupled line can be thought of as a transversal filter with continuous taps. This is a limiting case with each delay element becoming infinitesimal in length and the number of elements becoming infinite.

Loose coupling is required for two reasons. The finite impulse response is desired to be dominant so the secondary coupling must be kept small, see Fig. 7. The third order to first order ratio of contributions allows reduction of secondary coupling effects to any desired level of insignificance. Also, as an input signal travels along the line it is depleted by the fractions being coupled to the output. Loose coupling allows the approximation of non-depletion.

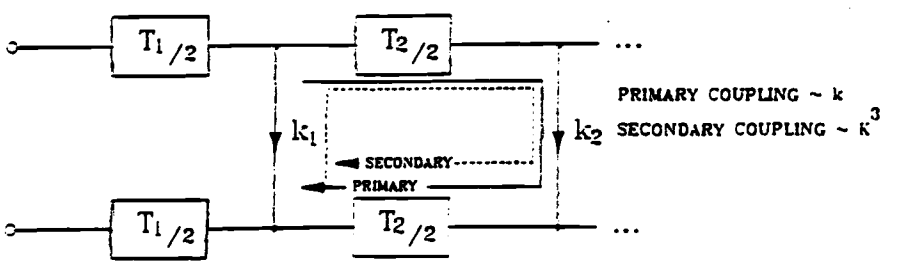


Fig. 7 Infinite impulse response with bidirectional taps

A design method has been developed:

* Synthesize the coupling profile for the desired step response.
* Use analysis (techniques to be discussed later) to determine changes necessary to compensate errors in the response.

The iterations do not interact and will quickly converge to the desired response.

ANALYSIS FROM ELECTRICAL PARAMETERS

The response of a nonuniform transmission line is approximated by dividing it into uniform segments which are then cascaded. The theory of small reflections insures convergence to a solution as the number of uniform segments grows. Even mode and odd mode two-port S-parameters are combined to obtain the four-port S-parameters.

The nonuniform transmission line is approximated by dividing it into uniform segments [13]. The S-parameters for each segment are calculated [14]. T-parameters are calculated using the relations:

$$T_{11} = (S_{12}\, S_{21} - S_{11}\, S_{22})/S_{21} \tag{19}$$

$$T_{12} = S_{11}/S_{21} \tag{20}$$

$$T_{21} = -S_{22}/S_{21} \tag{21}$$

$$T_{22} = 1/S_{21}. \tag{22}$$

The phase shift contributed by a uniform segment is cascaded with a transition to the impedance of the next segment, see Fig. 8. The segments are cascaded to form a composite T-matrix for the nonuniform line. S-parameters are obtained from the resultant T-parameters,

$$S_{11} = T_{12}/T_{22} \tag{23}$$

$$S_{12} = (T_{11}\, T_{22} - T_{12}\, T_{21})/T_{22} \tag{24}$$

$$S_{21} = 1/T_{22} \tag{25}$$

$$S_{22} = -T_{21}/T_{22}. \tag{26}$$

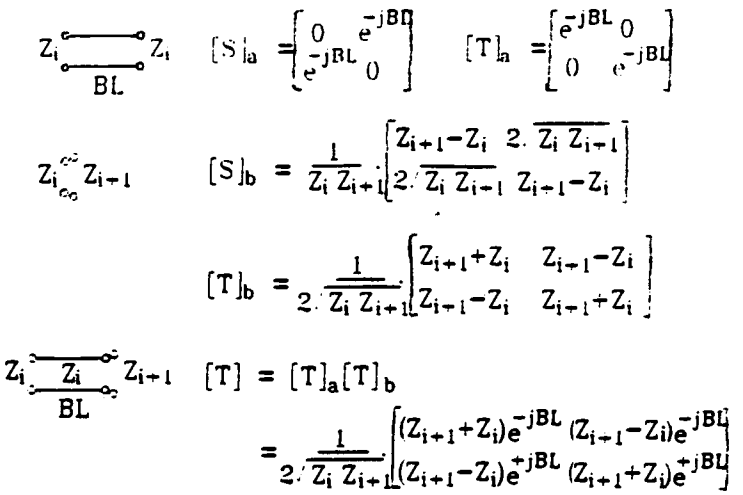


Fig. 8 Characteristics of the uniform segment

The program 'sin.c' uses the above method to analyze nonuniform transmission lines from electrical parameters $[Z_0(z),\beta(z)]$ which are included as functions in the program. A number of TEM [$\beta(z)$=constant] structures were simulated in this manner. The impulse response h(t) was obtained by inverse Fourier transform. This was then integrated to obtain the step response g(t).

A short sinusoidal response was analyzed using a small number of segments, see Fig. 9. The jagged regions correspond to the uniform segments. A finite number of frequency points were computed so the response is band-limited. With more frequency components the uniform segments would correspond to flat regions of the step response. A smoother version of this response was generated using a larger number of segments.

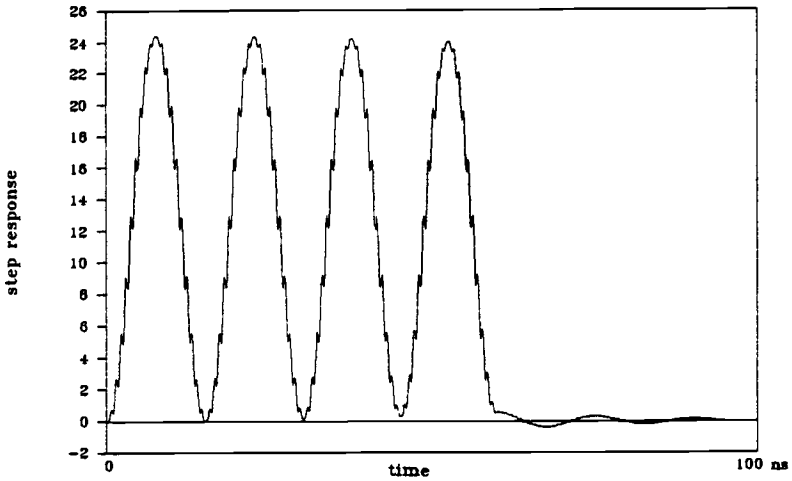


Fig. 9 Four periods of a sinusoid with small number of segments

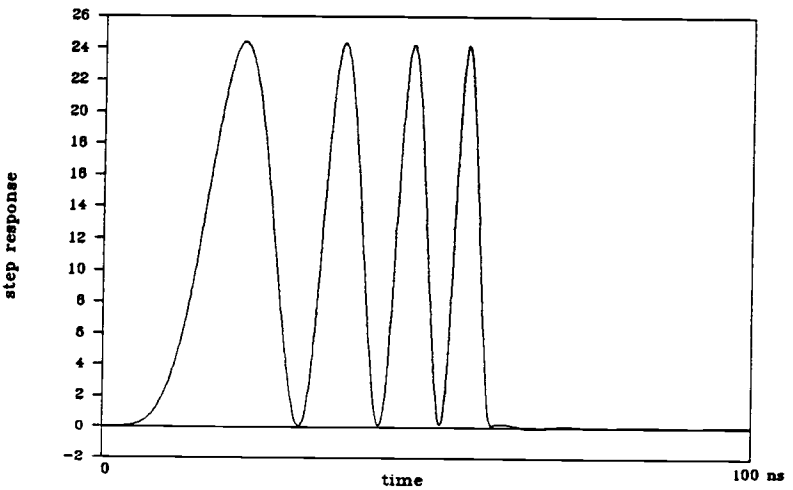


Fig. 10 Simple chirp response

A simple chirp response is shown in Fig. 10. In this example the change in even mode impedance is sinusoidal with quadratic

phase,

$$Z_e = Z_0 + [(Z_{max}-Z_0)/2]\ [1-\cos(ux^2/v^2)]. \qquad (27)$$

The impedance variation was limited to ten percent so that the approximation of equation (17) is valid. This is a chirp with zero initial frequency, the response shows an unlimited number of octaves. A somewhat longer chirp is shown in Fig. 11.

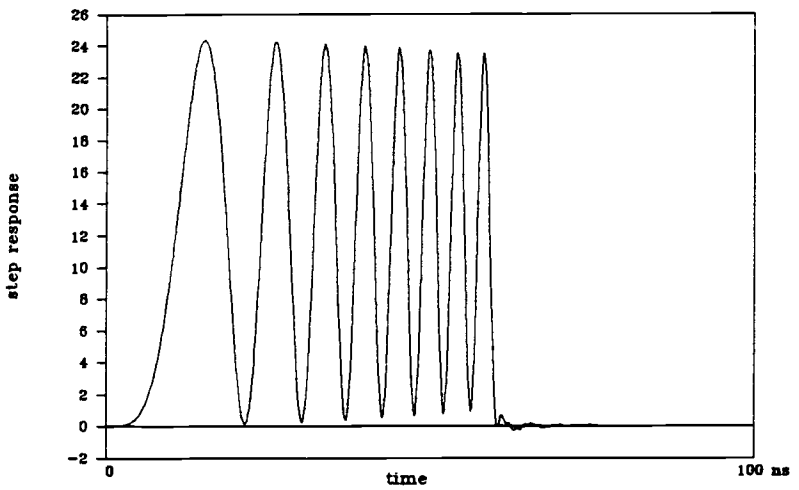


Fig. 11 Longer chirp -- initial frequency=0

A response comparable to that demonstrated with discrete coupling [3-7] is shown in Fig. 12a. Approximately 100 'periods' of chirp over an octave bandwidth centered at 3 GHz are visible. The duration of the response is 30 ns giving a time-bandwidth product of 60. The expanded sections, Fig. 12b-d,e show a very clean waveform with continuous variation in frequency and no visible distortion. No low pass filter is required.

Discrete coupling was simulated by varying the even mode impedance in a signum or squared version of the previous case. As seen in Fig. 13a the overall response is degraded and depletion of

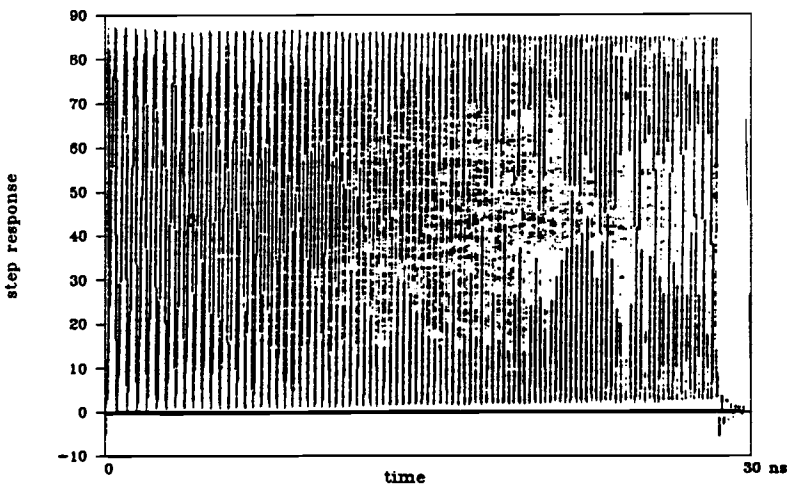


Fig. 12a Chirp of practical duration: overall view

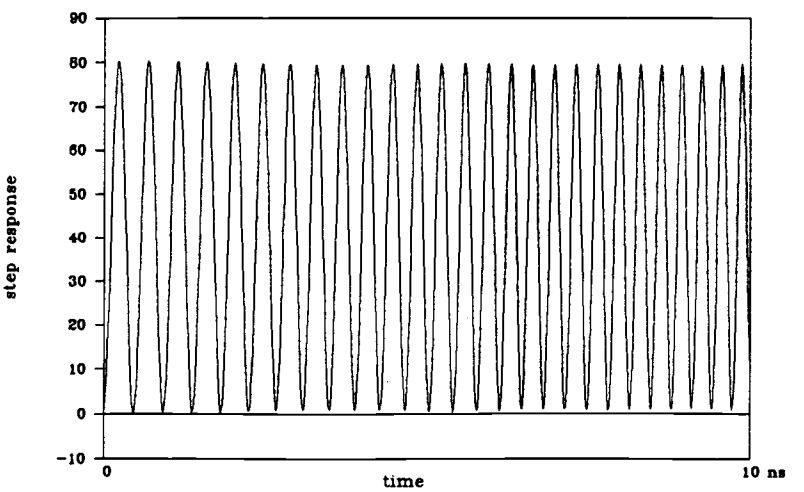


Fig. 12b Chirp of practical duration: first third of response

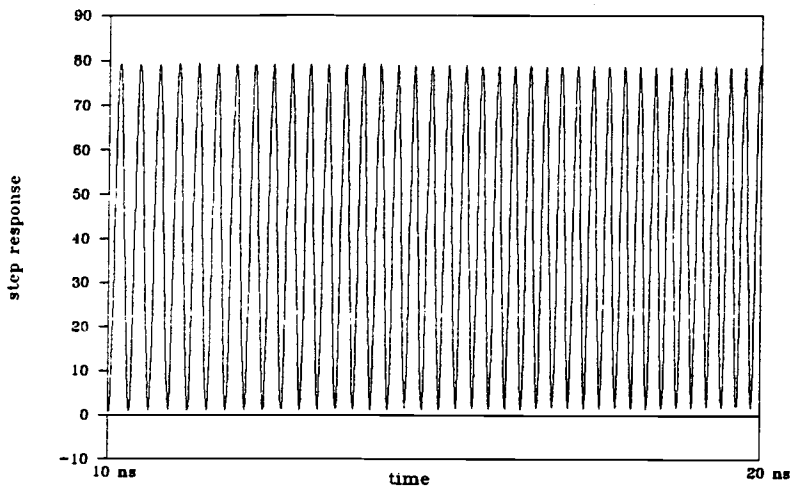


Fig. 12c Chirp of practical duration: second third of response

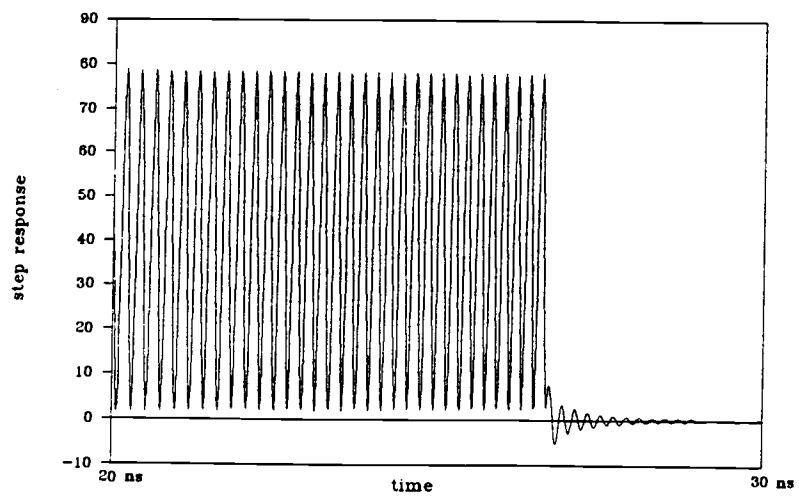


Fig. 12d Chirp of practical duration: third third of response

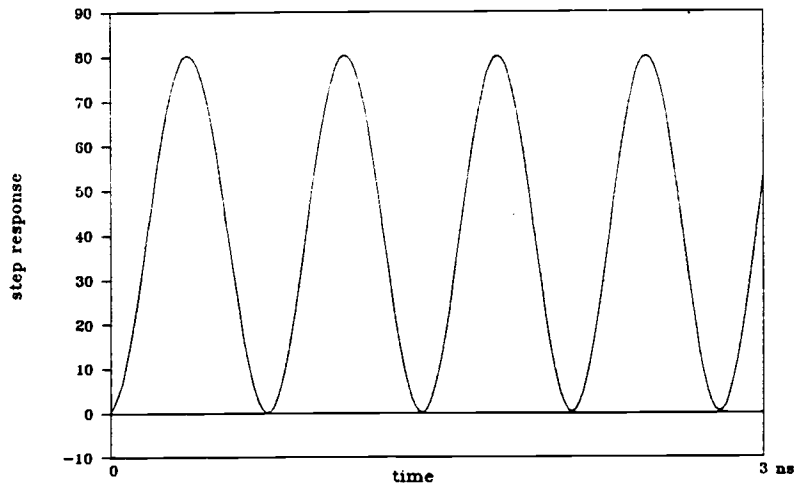


Fig. 12e Chirp of practical duration: expanded view of leading edge

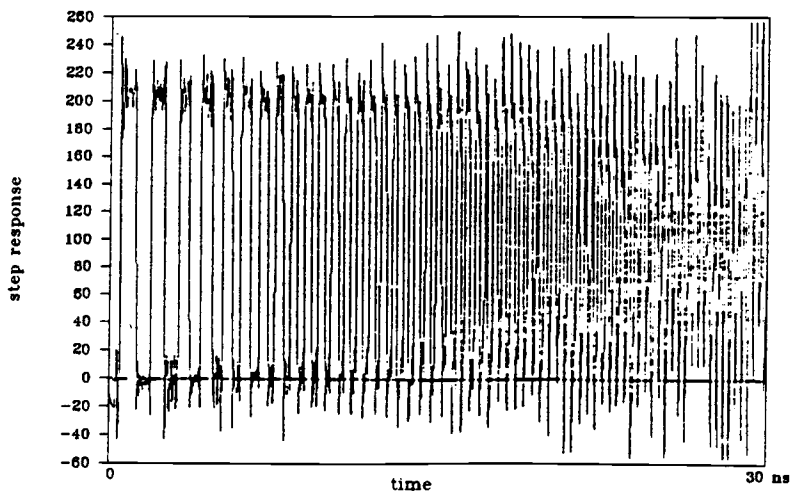


Fig. 13a Squared version of practical chirp: overall view

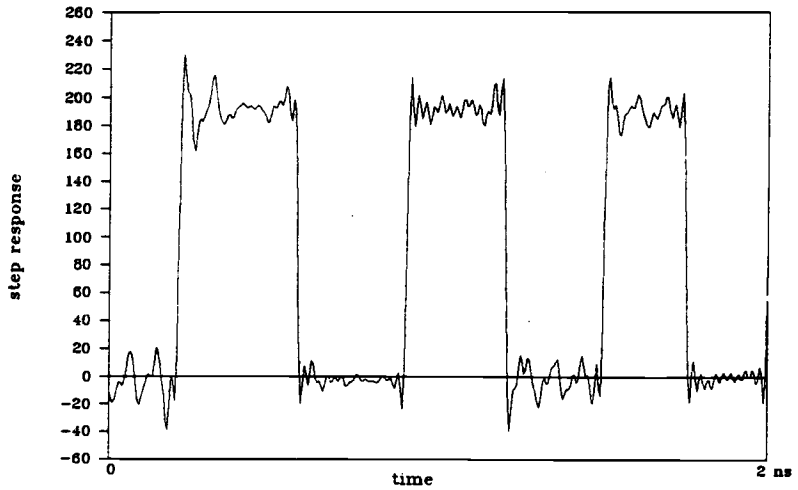


Fig. 13b Squared version of practical chirp: expanded view

the input signal has increased. The squared response is clearly shown in the expanded view of Fig. 13b. A low pass filter is necessary to obtain the desired response. This limits the bandwidth as previously discussed.

ANALYSIS FROM PHYSICAL PARAMETERS

The analysis of the previous section is needed as a design tool but more complete analysis is required to predict the response of a physical structure.

Accurate closed form equations for frequency dependent even and odd mode impedances are available for microstrip [15]. Program 'ntaper.c' uses these equations (found in include 'ms.h') to determine S-parameters for a nonuniform microstrip from physical dimensions. The width of the microstrip as a function of position is described as any number of linearly tapered sections which are specified by length and end widths. Each linear section is divided into uniform segments for cascade analysis.

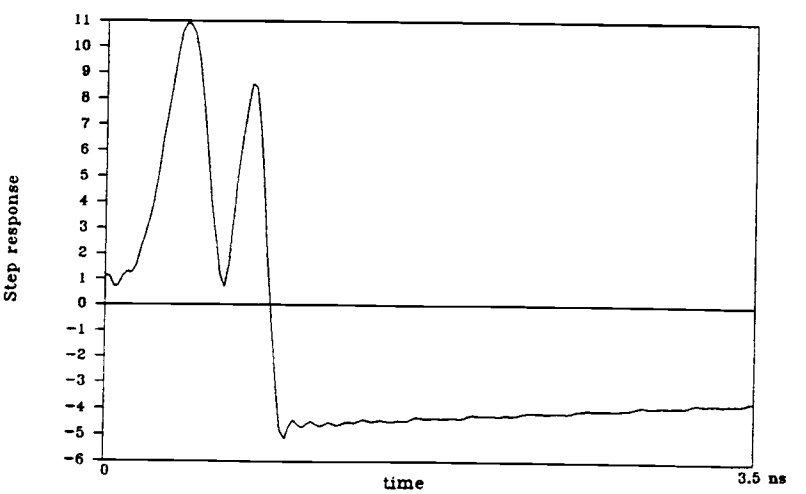


Fig. 14 Short chirp even mode equivalent line simulated response

An even mode equivalent line for a very short chirp response was analyzed. The simulated structure is described in the next section. The response in Fig. 14 reveals a design error (peaks not equal in amplitude) that was not eliminated. Computation time for analysis from physical parameters is long as each uniform segment requires calculation of electrical parameters from the extensive closed form equations [15].

The program 'ncplms.c' determines four-port S-parameters for nonuniformly coupled microstrips from physical dimensions. Additional terms describing the strip spacing at the end points is required for each linear section. Effects of velocity mismatch (between even and odd modes) and impedance variation (Z_{oo} Z_{oe}<>Z_o) are accounted for in this analysis.

The response for the short chirp coupled microstrip structure is shown in Fig. 15. Deviations from design are minimal.

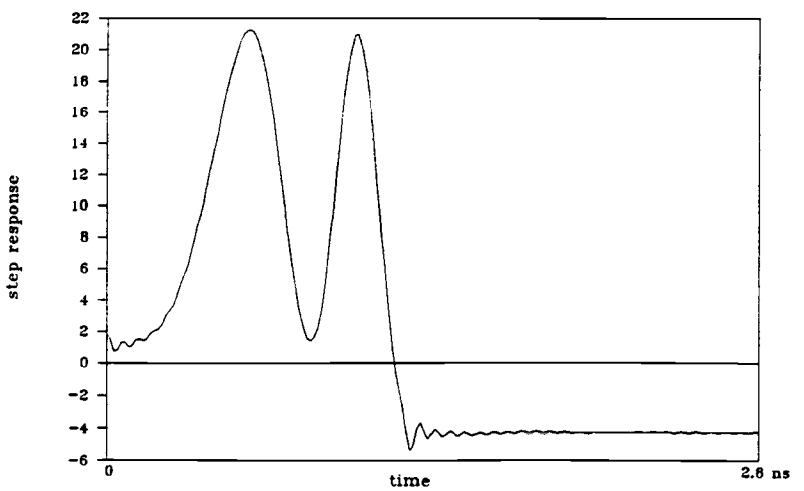


Fig. 15 Short chirp nonuniformly coupled lines simulated response

RESULTS FROM EXPERIMENTAL STRUCTURES

Microstrip test structures were fabricated, see Fig. 16. The substrate used was 30-mils thick with a relative dielectric constant of 2.20. Low dielectric constant material was chosen to reduce the effects of velocity mismatch.

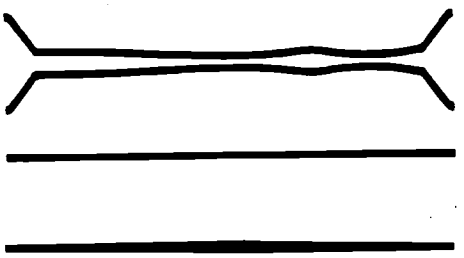


Fig. 16 Microstrip test structures (5.0 inches actual length)

The original pattern was cut by hand from rubylithe at 10x scale. The variation in width for the even mode equivalent line is not apparent in Fig. 16 and is not much more than the errors introduced by the limitation of manual dexterity. The coupled line spacing is plainly discernible and relatively accurate. Some variation in conductor width occurs but coupling profile should dominate the response. The double linear tapered line has large (50%) variation in impedance.

S-parameter measurements were made using an Hewlett-Packard 8510 Network Analyzer. Time Domain measurements were made using a Tektronix 11801 Time Domain Reflectometer system.

The step responses derived by inverse Fourier transform of the Network Analyzer data are shown in Figs. 17-20. The TDR results are shown in Figs. 21-24.

For all cases discontinuities occur at the connectors. For the coupled line cases the unused ports were terminated in 50 ohms.

Given the relatively poor fabrication tolerances the results are acceptable. Good correlation is seen between simulation and measurement. Conductor loss is a possible source of discrepancies as it was not accounted for in the simulations.

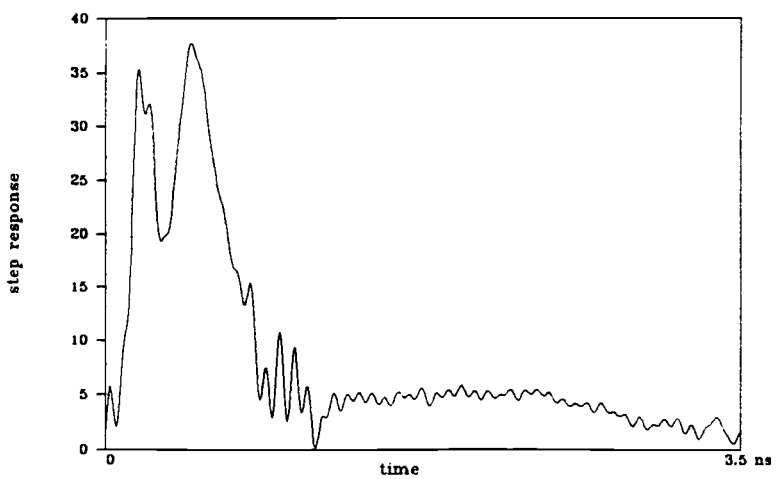


Fig. 17 Even line response from HP8510 data

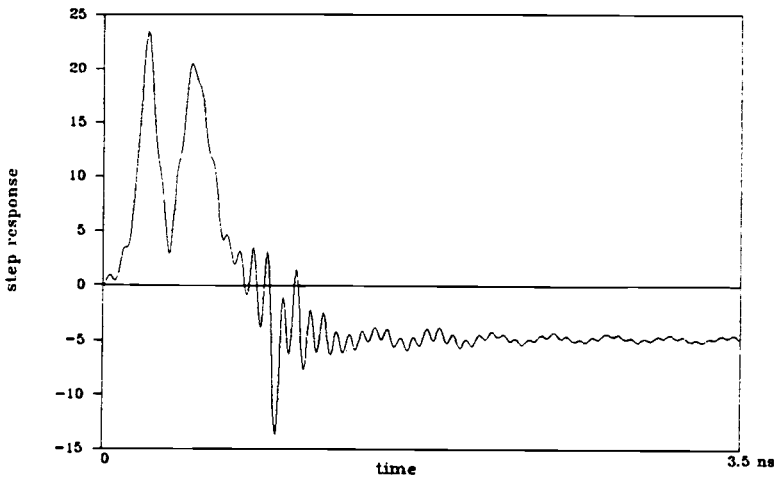


Fig. 18 Down-chirp response from HP8510 data

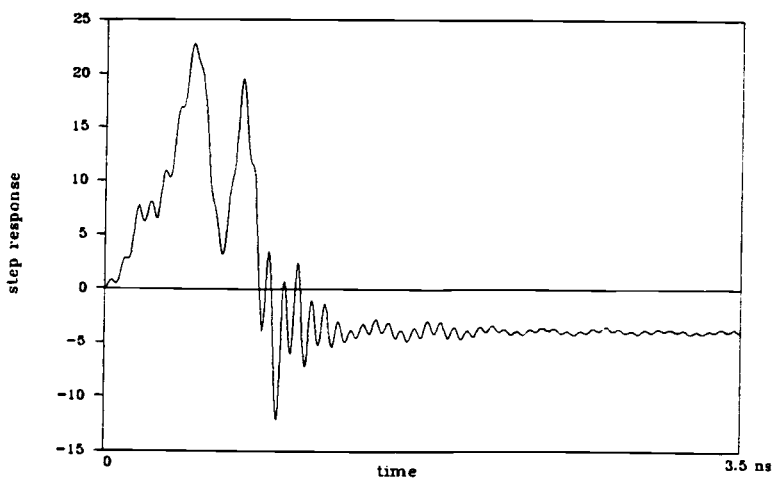


Fig. 19 Up-chirp response from HP8510 data

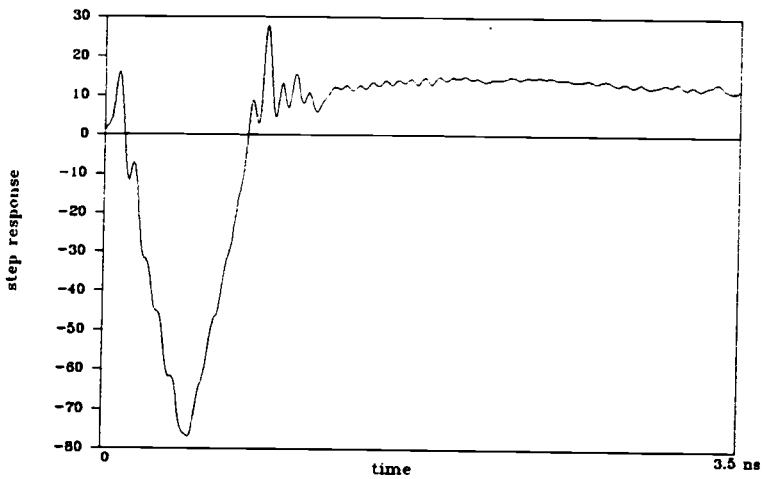


Fig. 20 Linear taper response from HP8510 data

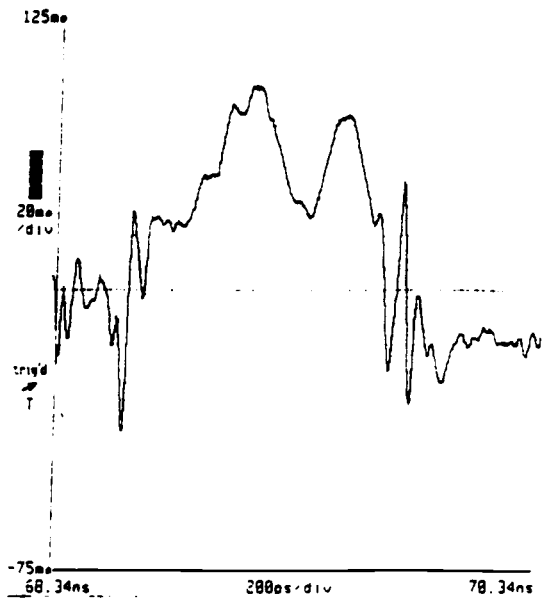


Fig. 21 Even line TDR response

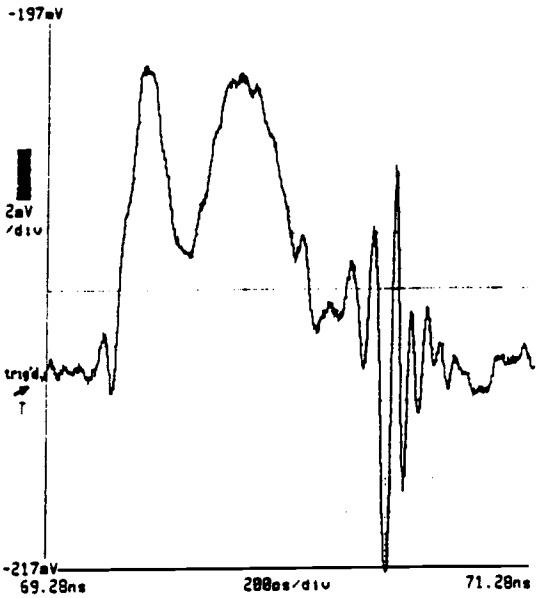


Fig. 22 Down-chirp TDR response

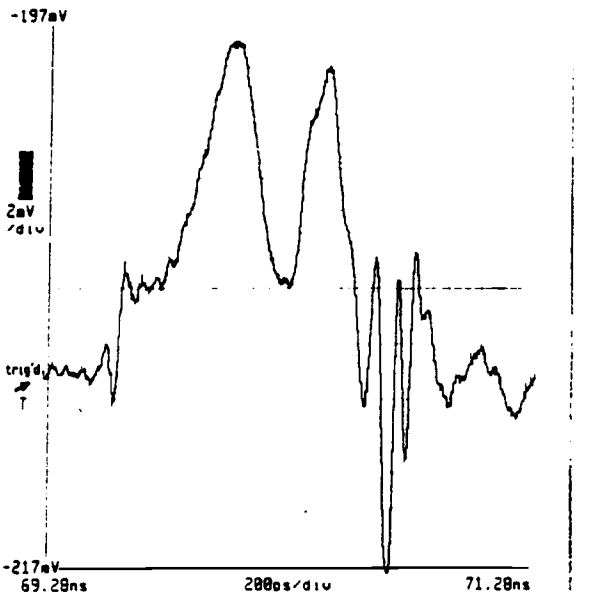


Fig. 23 Up-chirp TDR response

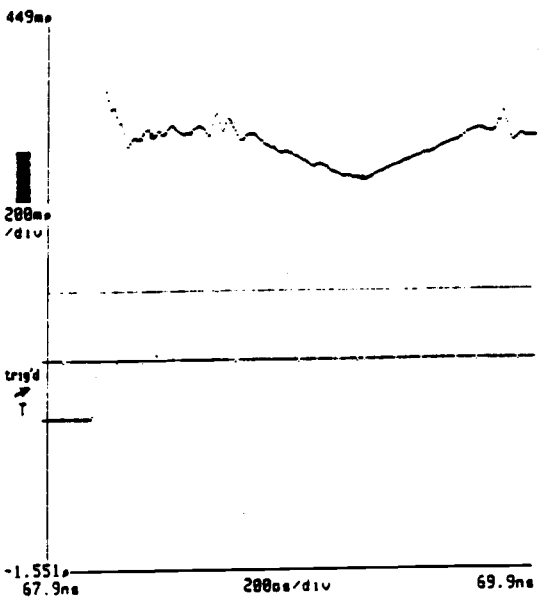


Fig. 24 Linear taper TDR response

CONCLUSION

The approach to design of nonuniform coupled lines presented here has been demonstrated to be useful. A verification analysis technique has been shown to predict with reasonable accuracy the results from measurements. The structures were tolerant of minor fabrication errors.

The application of structures based on the techniques presented are limited to signal processing and waveform synthesis. Practical implementation in either case requires low loss transmission lines that can only be achieved using superconductors. The recent advances in high temperature superconductors suggests that more general applicability remains possible for the future. At present the only known application area is for chirp filters in radar systems and real-time spectrum analysis.

Further investigation in this area might include extending the analysis method to account for loss mechanisms and examining the possibilities for synthesis of structures normally specified in the frequency domain (e.g., wide bandwidth couplers).

Ultimately these methods are limited by the transmission line approximation and associated theory of small reflections. When characteristic impedance contains insufficient information to characterize the line more sophisticated electromagnetic analyses (e.g., moment methods) become necessary.

BIBLIOGRAPHY

[1] K.J. Sternes, "Shaping Fast Rise-Time Pulses with Tapered Transmission Lines" IEEE Trans. Instrumentation and Measurement, Vol. IM-21, No.3, Aug '72

[2] P.C. Magnusson, Transmission Lines and Wave Propagation, Corvallis, Oregon: O.S.U. Book Stores, Inc., 1980

[3] R.S. Withers, A.C. Anderson, P.V. Wright, and S.A. Reible, "Superconductive Tapped Delay Lines for Microwave Analog Signal Processing" IEEE Trans. Magnetics, Vol. MAG-19, No. 3, May '83

[4] A.C. Anderson, R.S. Withers, S.A. Reible, and R.W. Ralston, "Substrates for Superconductive Analog Signal Processing Devices" IEEE Trans. Magnetics, Vol. MAG-21, No. 2, March '83

[5] R.S. Withers, A.C. Anderson, J.B. Green, and S.A. Reible, "Superconductive Delay-Line Technology and Applications" IEEE Trans. Magnetics, Vol. MAG-21, No. 2, March '85

[6] R.W. Ralston, "Signal Processing: Opportunities for Superconductive Circuits" IEEE Trans. Magnetics, Vol. MAG-21, No. 2, March '85

[7] M.S. DiIorio, R.S. Withers, and A.C. Anderson, "Wide-Band Superconductive Chirp Filters" IEEE Trans. Microwave Theory Tech., Vol. MTT-37, No. 4, April '89

[8] E.F. Bolinder, "Fourier Transforms in the Theory of Inhomogeneous Transmission Lines" Proceedings of the IRE, Vol. 38, p.1354, Nov '50

[9] C.B. Sharpe, "An Equivalence Principle for Nonuniform Transmission Line Directional Couplers" IEEE Trans. Microwave Theory Tech., Vol. MTT-15, No. 7, July '67

[10] J. Reed and G.J. Wheeler, "A Method of Analysis of Symmetrical Four-Port Networks" IRE Trans. Microwave Theory Tech., Vol. MTT-4, October '56

[11] R.E. Collin, Foundations for Microwave Engineering, New York, New York: McGraw-Hill, Inc., 1968

[12] G.S. Kino, Acoustic Waves, Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1987

[13] M.A. Mehalic, C.H. Chan, and R. Mittra, "Investigation of Tapered Multiple Microstrip Lines for VLSI Circuits" 1988 IEEE MTT-S Symposium Digest

[14] J. Choma, Electrical Networks, New York, New York: John Wiley & Sons, Inc., 1985

[15] M. Kirschning, and R.H. Jansen, "Accurate Wide-Range Design Equations for the Frequency-Dependent Characteristic of Parallel Coupled Microstrip Lines" IEEE Trans. Microwave Theory Tech., Vol. MTT-32, No. 1, January '84

APPENDIX

Appendix A: Program listings

This appendix contains listings for the programs used in this research. All programs were written in 'C' using the AZTEC C Compiler on an IBM AT style computer. The only libraries used were the standard input/output and mathematics files. The code should be easily portable to other machines.

Program 'ntaper.c' (p. 37-40) analyzes nonuniform microstrip from physical dimensions.

Program 'ncplms.c' (p. 41-44) analyzes nonuniformly coupled microstrips from physical dimensions.

Program 'sin.c' (p. 45-49) analyzes nonuniform transmission lines from subroutines containing impedance and phase constants as functions of position.

Include file 'ms.h' (p. 50-51) is a C-language adaptation of Kirschning & Jansen's microstrip equations.

Include file 'cmath.h' (p. 52-53) contains all the complex mathematics functions used.

Program 'fft.c' (p. 54-55) is the program used for inverse FFT computation.

```
/*   Program "ntaper.c"

     This program was written as part of the research for
     an MS Thesis by Leonard Hayden, Oregon State University,
     1989. All rights are reserved, do not use without
     permission.


     This program computes the two-port S-paramters for a
     nonuniform microstripline. The physical line
     is separated into sections with linear physical taper.
     Each section is divided into a number of uniform segments.
     The T-parameters are determined for each segment, then
     cascaded by multiplying matrices. The resultant T-matrix
     is converted to S-parameters. This method was described
     in:
        "Investigation of tapered multiple microstrip lines for
        VLSI circuits" Mehalic, Chan, and Mittra from the
        Electromagnetics Communication Laboratory, University
        of Illinois, Urbana. 1988 IEEE Microwave Theory and
        Techniques Symposium Digest, page 215.

*/

# include "stdio.h"
# include "math.h"
# include "cmath.h"
# include "ms.h"

/*
     "cmath.h" includes all the complex number declarations
     and routines. "ms.h" is a c-language adaptation of a
     basic program (Kirschning & Jansen, IEEE Transactions
     MTT-1984 with revisions) that calculates microstrip
     electrical parameters from physical dimensions (curve fit).
*/

struct CMATRIX{

struct COMPLEX el[2][2];

}cmat_mul();

struct CMAT4{

struct COMPLEX el[4][4];

}maophase();

struct ZG{

struct COMPLEX a,b,c;

int n;

};

# define C_LIGHT 3e8
# define maxsections 100

float eppsr,subht,spacing,w[maxsections],xp[maxsections];
/*
   rel dielec constant and height of substrate, dummy, line width at
   each junction between linear sections and location of junction
*/
int nls,nseg[maxsections];
/*
   no. of lin sections and no. of segments per section
*/

main()
{
float f_min,f_max;
extern float eppsr,subht,spacing;
extern float w[maxsections],xp[maxsections];
float freq;
int i,j,k,flag,no_of_points;
extern int nls,nseg[maxsections];
struct CMAT4 s;
struct CMATRIX get_s_params(),se;
struct COMPLEX r;
char file[80],*fnm;
FILE *fname;

/*
     Input section
*/

printf("Analysis of linear tapers W1 --> W2 --> W3 ....\n\n");
printf("Enter substrate relative dielectric constant:");
scanf("%f",&eppsr);
printf("Enter substrate height [thickness] in mm:");
scanf("%f",&subht);
printf("Enter fmin, fmax, number of points [frequency in GHz]\n");
scanf("%f,%f,%d",&f_min,&f_max,&no_of_points);
f_min=f_min*1e+9;
f_max=f_max*1e+9;
printf("Enter output file name:");
scanf("%s",file);
printf("Enter number of linear sections:");
scanf("%d",&nls);

xp[0]=0.;
printf("position=0.000 -- enter width (normalized by substrate height):");
scanf("%f",&w[0]);
spacing=0.;

for (i=1;i<=nls;i++)
```

```
 {
 printf("enter width (normalized) at far end of linear section #%d:",i);
 scanf("%f",&w[i]);
 printf("enter position (in meters) at far end of linear section #%d:",i);
 scanf("%f",&xp[i]);
 printf("enter number of uniform segments for linear section #%d:",i);
 scanf("%d",&nseg[i]);
 }


/*
     Start of analysis
*/

printf(
"; freq s11m.p s12m.p s21m,p....\n");

fname=fopen(file,"w");
fprintf(fname,"%d\n",no_of_points);
/*
     Output file format has no. of points as header
*/

for (i=0;i<no_of_points;i++)

{

freq=f_min+ ((f_max-f_min)*i)/(no_of_points-1);

flag=0;
/*  single line analysis--hang-over from coupled line analysis */
        se=get_s_params(freq,flag);
/*  compute s-parameters */

fprintf(fname,"%9.4le %9.4le\n",se.el[0][0].real,se.el[0][0].imag);
/* writes point (S11 real, imag) to file */


for(k=0;k<2;k++){

 for( j=0; j<2; j++){

  r=se.el[k][j];

  se.el[k][j].real=cabs(r);

  se.el[k][j].imag=phase(r);

 }

}
/*  convert real,imag to mag,phase */


   printf("\r%9.3e %9.4le %5.1lf %9.4le %5.1lf %9.4le %5.1lf %9.4le %5.1lf\n",

freq,

 se.el[0][0].real,se.el[0][0].imag,

 se.el[0][1].real,se.el[0][1].imag,

 se.el[1][0].real,se.el[1][0].imag,

 se.el[1][1].real,se.el[1][1].imag);


}
i=fclose(fname);
printf("i=%d",i);
}


struct CMATRIX get_s_params(freq,flag)
float freq;
int flag;
{
extern int nls,nseg[maxsections];
struct COMPLEX gamma_1,z_2,z_1;
struct CMATRIX t_acc,delta_t,s_conv(),get_dt();
struct COMPLEX zdiv,sum,diff,egamma1,emgamma1;
struct ZG z_gamma(),zg;
int section,segment;
struct COMPLEX z(),gamma();
double *length,delta_1,zz;

/*
     Initialize source termination impedance to 50 ohms
*/

section=1;
segment=0;
zz=cabs(z(section,segment,freq,flag));
t_acc.el[0][0].real=(50.+zz)/(2*sqrt(50.*zz));
t_acc.el[0][0].imag=0.;
t_acc.el[0][1].real=(zz-50.)/(2*sqrt(50.*zz));
t_acc.el[0][1].imag=0.;
t_acc.el[1][0]=t_acc.el[0][1];
t_acc.el[1][1]=t_acc.el[0][0];

/*
     Iterate through all sections and segments
*/

for (section=1;section<=nls;section++){
  for (segment=0;segment<nseg[section];segment++){
  if (segment==0) printf("  seg#");
  if (segment>0) printf("\b");  /* backspace character */
```

```
   if (segment>10) printf("\b");
/  if (segment>100) printf("\b");
   if (segment>1000) printf ("\b");

 /* print segment number as status line */

   printf("%d",segment);

 /*
      Determine endpoint impedances and average beta-ell for segment
 */
   gamma_1=gamma(section,segment,freq,flag);
   z_1=z(section,segment,freq,flag);
   z_2=z(section,segment+1,freq,flag);
   delta_1=(xp[section]-xp[section-1])/nseg[section];
   gamma_1=cmul(complex(0.5),cadd(gamma_1,gamma(section,segment+1,freq,flag)));
   gamma_1=cmul(gamma_1,complex(delta_1));


 /* intermediate calculations */
   zdiv=csqrt(cmul(z_1,z_2));
   zdiv.real=zdiv.real*2.;
   zdiv.imag=zdiv.imag*2.;
#  sum=cdiv(cadd(z_1,z_2),zdiv);
   diff=cdiv(csub(z_2,z_1),zdiv);
   egamma1=cexp(gamma_1);
   emgamma1=cexp(cneg(gamma_1));


 /* determine delta-T matrix */
   delta_t.el[0][0]=cmul(sum,emgamma1);
   delta_t.el[0][1]=cmul(diff,emgamma1);
   delta_t.el[1][0]=cmul(diff,egamma1);
   delta_t.el[1][1]=cmul(sum,egamma1);

 /* cascade transmission networks */
    t_acc=cmat_mul(t_acc,delta_t);
    }
   }

 /*
      Complete with load termination of 50 ohms
 */
 zz=cabs(z(nls,nseg[nls],freq,flag));
 delta_t.el[0][0].real=(50.+zz)/(2.*sqrt(50.*zz));
 delta_t.el[0][0].imag=0.;
 delta_t.el[0][1].real=(50.-zz)/(2.*sqrt(50.*zz));
 delta_t.el[0][1].imag=0.;
 delta_t.el[1][0]=delta_t.el[0][1];
 delta_t.el[1][1]=delta_t.el[0][0];

 t_acc=cmat_mul(t_acc,delta_t);

 return (s_conv(t_acc));
 }
```

```
struct CMATRIX s_conv(t_acc)
struct CMATRIX t_acc;
{

/*
     Convert T-matrix to S-parameters
*/

struct CMATRIX s;

s.el[0][0]=cdiv(t_acc.el[0][1],t_acc.el[1][1]);
s.el[0][1]=csub(t_acc.el[0][0],cdiv(cmul(t_acc.el[0][1],
                                t_acc.el[1][0]),t_acc.el[1][1]));
s.el[1][0]=cinv(t_acc.el[1][1]);
s.el[1][1]=cneg(cdiv(t_acc.el[1][0],t_acc.el[1][1]));

return(s);
}


struct COMPLEX gamma(section,segment,freq,flag)
int section,segment,flag;
float freq;
{

/*
     Determine propagation constant at given location
     using physical geometry and curve-fit routine
*/

struct MS mscut;
struct COMPLEX gam;
float width();
extern float epsr,subht,spacing;
mscut=mstrips(epsr,subht,width(section,segment),spacing,freq,flag);
gam.imag=2.*PI*freq*sqrt(mscut.el[0])/C_LIGHT;
gam.real=0.;

return(gam);
}


struct COMPLEX z(section,segment,freq,flag)
int section,segment,flag;
float freq;
{

/*
     Determine impedance at given location using physical
     geometry and curve-fit routine
*/
```

```
struct MS mscut;
struct COMPLEX imp;
float width();
extern float eppsr,subht,spacing;
mscut=mstrips(eppsr,subht,width(section,segment),spacing,freq,flag);
imp.real=mscut.el[1];
imp.imag=0.;

return(imp);
}

float width(section,segment)
int section,segment;
{

/*
     Determine physical geometry at given location using input
     data on sections and number of segments per section
*/

float ww;

ww=(w[section]-w[section-1])*segment/nseg[section]+w[section-1];

return(ww);
}



struct CMATRIX cmat_mul(a,b)
struct CMATRIX a,b;
{

/*
     Multiply two-by-two matrices
*/

struct CMATRIX c;
int i,j;

for (i=0;i<2;i)(
 for (j=0;j<2;j)(
  c.el[i][j]=cadd(cmul(a.el[i][0],b.el[0][j]),cmul(a.el[i][1],b.el[1][j]));
   j++;
   )
 i++;
 )
return(c);
}
```

```
/*   Program "ncplms.c"

     This program was written as part of the research for
     an MS Thesis by Leonard Hayden, Oregon State University,
     1989. All rights are reserved, do not use without
     permission.


     This program computes the two-port S-paramters for a
     nonuniform microstripline. The physical line
     is separated into sections with linear physical taper.
     Each section is divided into a number of uniform segments.
     The T-parameters are determined for each segment, then
     cascaded by multiplying matrices. The resultant T-matrix
     is converted to S-parameters. This method was described
     in:
          "Investigation of tapered multiple microstrip lines for
          VLSI circuits" Mehalic, Chan, and Mittra from the
          Electromagnetics Communication Laboratory, University
          of Illinois, Urbana. 1988 IEEE Microwave Theory and
          Techniques Symposium Digest, page 215.

*/

# include "stdio.h"
# include "math.h"
# include "cmath.h"
# include "ms.h"

/*
     "cmath.h" includes all the complex number declarations
     and routines. "ms.h" is a c-language adaptation of a
     basic program (Kirschning & Jansen, IEEE Transactions
     MTT-1984 with revisions) that calculates microstrip
     electrical parameters from physical dimensions (curve fit).
*/

struct CMATRIX{

struct COMPLEX el[2][2];

}cmat_mul();

struct CMAT4{

struct COMPLEX el[4][4];

}maophase();

struct ZG{

struct COMPLEX a,b,c;

int n;
};

# define C_LIGHT 3e8
# define maxsections 100

float eppsr,subht,w[maxsections],xp[maxsections],sp[maxsections];
/*
   rel dielec constant and height of substrate, dummy, line width at
   each junction between linear sections and location of junction
*/
int nls,nseg[maxsections];
/*
   no. of lin sections and no. of segments per section
*/

main()
{
float f_min,f_max;
extern float eppsr,subht,w1,w2;
extern float w[maxsections],xp[maxsections],sp[maxsections];
float freq;
int i,j,k,flag,no_of_points;
extern int nls,nseg[maxsections];
struct CMAT4 s;
struct CMATRIX get_s_params(),se,so;
struct COMPLEX r;
char file[80],*fnm;
FILE *fname;

/*
     Input section
*/

printf("Analysis of linear tapers W1 --> W2 --> W3 ....\n\n");
printf("Enter substrate relative dielectric constant:");
scanf("%f",&eppsr);
printf("Enter substrate height [thickness] in mm:");
scanf("%f",&subht);
printf("Enter fmin, fmax, number of points [frequency in GHz]\n");
scanf("%f,%f,%d",&f_min,&f_max,&no_of_points);
f_min=f_min*1e+9;
f_max=f_max*1e+9;
printf("Enter output file name:");
scanf("%s",file);
printf("Enter number of linear sections:");
scanf("%d",&nls);

xp[0]=0.;
printf(
"position=0.(000) -- enter width, spacing (normalized by substrate height):");
scanf("%f,%f",&w[0],&sp[0]);

for (i=1;i<=nls;i++)
```

```
 {
 printf(
 "enter width, spacing (normalized) at far end of linear section #%d:",i);
 scanf("%f,%f",&w[i],&sp[i]);
 printf("enter position (in meters) at far end of linear section #%d:",i);
 scanf("%f",&xp[i]);
 printf("enter number of uniform segments for linear section #%d:",i);
 scanf("%d",&nseg[i]);
 }


/*
     Start of analysis
*/

printf(
"; freq s11m,p s12m,p s21m,p....\n");

fname=fopen(file,"w");
fprintf(fname,"%d\n",no_of_points);
/*
     Output file format has no. of points as header
*/

for (i=0;i<no_of_points;i++)

{

freq=f_min+ ((f_max-f_min)*i)/(no_of_points-1);

flag=2;
        se=get_s_params(freq,flag);

flag=1;

sc=get_s_params(freq,flag);

/* printf("%9.4le %9.4le, %9.4le %9.4le \n",se.el[0][0],so.el[0][0]); debug */

/*  compute s-parameters */
s.el[0][0]=cmul(ccmplex(0.5),cadd(se.el[0][0],so.el[0][0]));
s.el[0][1]=cmul(ccmplex(0.5),csub(se.el[0][0],so.el[0][0]));
s.el[0][2]=cmul(ccmplex(0.5),csub(se.el[0][1],so.el[0][1]));
s.el[0][3]=cmul(ccmplex(0.5),cadd(se.el[0][1],so.el[0][1]));
s.el[1][0]=s.el[0][1];
s.el[1][1]=s.el[0][0];
s.el[1][2]=s.el[0][3];
s.el[1][3]=s.el[0][2];
s.el[2][0]=cmul(ccmplex(0.5),csub(se.el[1][0],so.el[1][0]));
s.el[2][1]=cmul(ccmplex(0.5),cadd(se.el[1][0],so.el[1][0]));
s.el[2][2]=cmul(ccmplex(0.5),cadd(se.el[1][1],so.el[1][1]));
s.el[2][3]=cmul(ccmplex(0.5),csub(se.el[1][1],so.el[1][1]));
s.el[3][0]=s.el[2][1];
s.el[3][1]=s.el[2][0];
s.el[3][2]=s.el[2][3];
s.el[3][3]=s.el[2][2];

fprintf(fname,"%9.4le %9.4le\n",s.el[1][0].real,s.el[1][0].imag);
/* writes point (S21 real, imag) to file */


for(k=0;k<4;k++){

 for( j=0; j<4; j++){

  r=s.el[k][j];

  s.el[k][j].real=cabs(r);

  s.el[k][j].imag=phase(r);

 }

}
/*  convert real,imag to mag,phase */

    for ( j=0; j<4; j++){
    printf("\r%9.3e %9.4le %5.1lf %9.4le %5.1lf %9.4le %5.1lf %9.4le %5.1lf\n",

 freq,

  s.el[j][0].real,s.el[j][0].imag,

  s.el[j][1].real,s.el[j][1].imag,

  s.el[j][2].real,s.el[j][2].imag,

  s.el[j][3].real,s.el[j][3].imag);
         }

 }
 i=fclose(fname);
 printf("i=%d",i);
 }


 struct CMATRIX get_s_params(freq,flag)
 float freq;
 int flag;
 {
 extern int nls,nseg[maxsections];
 struct COMPLEX gamma_1,z_2,z_1;
 struct CMATRIX t_acc,delta_t,s_conv(),get_dt();
 struct COMPLEX zdiv,sum,diff,egamma,emgamma;
 struct ZG z_gamma();
 int section,segment;
 struct COMPLEX z(),gamma();
 double *length,delta_l,zz;
```

```
/*
     Initialize source termination impedance to 50 ohms
*/

section=1;
segment=0;
zz=cabs(z(section,segment,freq,flag));
t_acc.el[0][0].real=(50.+zz)/(2*sqrt(50.*zz));
t_acc.el[0][0].imag=0.;
t_acc.el[0][1].real=(zz-50.)/(2*sqrt(50.*zz));
t_acc.el[0][1].imag=0.;
t_acc.el[1][0]=t_acc.el[0][1];
t_acc.el[1][1]=t_acc.el[0][0];

/*
     Iterate through all sections and segments
*/

for (section=1;section<=nls;section++){
  for (segment=0;segment<nseg[section];segment++){
  if (segment==0) printf("  seg#");
  if (segment>0) printf("\b");  /* backspace character */
  if (segment>10) printf("\b");
  if (segment>100) printf("\b");
  if (segment>1000) printf ("\b");

/* print segment number as status line */

  printf("%d",segment);

/*
     Determine endpoint impedances and average beta-ell for segment
*/
   gamma_1=gamma(section,segment,freq,flag);
   z_1=z(section,segment,freq,flag);
   z_2=z(section,segment+1,freq,flag);
   delta_l=(xp[section]-xp[section-1])/nseg[section];
   gamma_1=cmul(ccomplex(0.5),cadd(gamma_1,gamma(section,segment+1,freq,flag)));
   gamma_1=cmul(gamma_1.ccomplex(delta_l));

/* intermediate calculations */
   zdiv=csqrt(cmul(z_1,z_2));
   zdiv.real=zdiv.real*2.;
   zdiv.imag=zdiv.imag*2.;
   sum=cdiv(cadd(z_1,z_2),zdiv);
   diff=cdiv(csub(z_2,z_1),zdiv);
   egamma1=cexp(gamma_1);
   emgamma1=cexp(cneg(gamma_1));

/* determine delta-T matrix */
   delta_t.el[0][0]=cmul(sum,emgamma1);
   delta_t.el[0][1]=cmul(diff,egamma1);
   delta_t.el[1][0]=cmul(diff,egamma1);
   delta_t.el[1][1]=cmul(sum,egamma1);

/* cascade transmission networks */
   t_acc=cmat_mul(t_acc,delta_t);
   }
  }

/*
     Complete with load termination of 50 ohms
*/
zz=cabs(z(nls,nseg[nls],freq,flag));
delta_t.el[0][0].real=(50.+zz)/(2.*sqrt(50.*zz));
delta_t.el[0][0].imag=0.;
delta_t.el[0][1].real=(50.-zz)/(2.*sqrt(50.*zz));
delta_t.el[0][1].imag=0.;
delta_t.el[1][0]=delta_t.el[0][1];
delta_t.el[1][1]=delta_t.el[0][0];

t_acc=cmat_mul(t_acc,delta_t);

return (s_conv(t_acc));
}


struct CMATRIX s_conv(t_acc)
struct CMATRIX t_acc;
{

/*
     Convert T-matrix to S-parameters
*/

struct CMATRIX s;

s.el[0][0]=cdiv(t_acc.el[0][1],t_acc.el[1][1]);
s.el[0][1]=csub(t_acc.el[0][0],cdiv(cmul(t_acc.el[0][1],
                                t_acc.el[1][0]),t_acc.el[1][1]));
s.el[1][0]=cinv(t_acc.el[1][1]);
s.el[1][1]=cneg(cdiv(t_acc.el[1][0],t_acc.el[1][1]));

return(s);
}


struct COMPLEX gamma(section,segment,freq,flag)
int section,segment,flag;
float freq;
{

/*
     Determine propagation constant at given location
     using physical geometry and curve-fit routine
*/
```

```
struct MS mscut;
struct COMPLEX gam;
float width(),spacing();
extern float eppsr,subht;
mscut=mstrips(eppsr,subht,width(section,segment),
 spacing(section,segment),freq,flag);
gam.imag=2.*PI*freq*sqrt(mscut.el[0])/C_LIGHT;
gam.real=0.;

return(gam);
}


struct COMPLEX z(section,segment,freq,flag)
int section,segment,flag;
float freq;
{

/*
     Determine impedance at given location using physical
     geometry and curve-fit routine
*/

struct MS mscut;
struct COMPLEX imp;
float width(),spacing();
extern float eppsr,subht;
mscut=mstrips(eppsr,subht,width(section,segment),
  spacing(section,segment),freq,flag);
imp.real=mscut.el[1];
imp.imag=0.;

return(imp);
}

float width(section,segment)
int section,segment;
{

/*
     Determine physical geometry at given location using input
     data on sections and number of segments per section
*/

float ww;

ww=w[section]-w[section-1]*segment/nseg[section]+w[section-1];

return(ww);
}


float spacing(section,segment)
int section,segment;
{

/*
     Determine physical spacing at given location using input
     data on sections and number of segments per section
*/

float ww;

ww=(sp[section]-sp[section-1])*segment/nseg[section]+sp[section-1];

return(ww);
}



struct CMATRIX cmat_mul(a,b)
struct CMATRIX a,b;
{

/*
     Multiply two-by-two matrices
*/

struct CMATRIX c;
int i,j;

for (i=0;i<2;i){
 for (j=0;j<2;j){
  c.el[i][j]=cadd(cmul(a.el[i][0],b.el[0][j]),cmul(a.el[i][1],b.el[1][j]));
  }++;
  }
 i++;
 }
return(c);
}
```

Program "sim.c"

```
# include "stdio.h"
# include "math.h"

struct COMPLEX{

double real,imag;

}cneg(),cinv(),cadd(),csub(),cmul(),cdiv(),csqrt(),cexp(),complex();

double cabs(),phase();

struct CMATRIX{

struct COMPLEX el[2][2];

}cmat_mul();

struct CMAT4{

struct COMPLEX el[4][4];

}magphase();

struct ZG{

struct COMPLEX a,b,c;

int n;

};

# define delta_z .05

# define AND &&
# define OR ||
# define PI 3.14159265358979323846264338328
# define C_LIGHT 3e8
# define delta_gl PI/10.

float zw1,zw2;
float totlength;
int n_max;
main()
{
float f_min, f_max;
extern float totlength,zw1,zw2;
float freq;
int i,j,k,flag,no_of_points;
extern int n_max;
struct CMAT4 s;
struct CMATRIX get_s_params(),se,so;
struct COMPLEX r;
char file[80],*fnm;

FILE *fname;

printf("Analysis of cosine electrical taper \n");
/* printf("Enter endpoint even mode impedance:");
scanf("%f",&zw1);*/
printf("Enter midpoint even mode impedance:");
scanf("%f",&zw2);
printf("\nEnter total length:");
scanf("%f",&totlength);
printf("Enter fmin, fmax, number of points (frequency in GHz)\n");
scanf("%f,%f,%d",&f_min,&f_max,&no_of_points);
f_min=f_min*1e+9;
f_max=f_max*1e+9;
printf("Enter number of segments:");
scanf("%d",&n_max);
printf("Enter output file name:");
scanf("%s",file);

printf(
"| freq s11m,p s12m,p s13m,p s14m,p s21m,p....\n");

fname=fopen(file,"w");
fprintf(fname,"%d\n",no_of_points);

for (i=0;i<no_of_points;i++)

{

freq=f_min+ ((f_max-f_min)*i)/(no_of_points-1);

flag=2; /* even */
        se=get_s_params(freq,flag);

flag=1; /* odd */
/*
so=get_s_params(freq,flag);
s.el[0][0]=cmul(complex(0.5),cadd(se.el[0][0],so.el[0][0]));
s.el[0][1]=cmul(complex(0.5),csub(se.el[0][0],so.el[0][0]));
s.el[0][2]=cmul(complex(0.5),csub(se.el[0][1],so.el[0][1]));
s.el[0][3]=cmul(complex(0.5),cadd(se.el[0][1],so.el[0][1]));
s.el[1][0]=s.el[0][1];
s.el[1][1]=s.el[0][0];
s.el[1][2]=s.el[0][3];
s.el[1][3]=s.el[0][2];
s.el[2][0]=cmul(complex(0.5),csub(se.el[1][0],so.el[1][0]));
s.el[2][1]=cmul(complex(0.5),cadd(se.el[1][0],so.el[1][0]));
s.el[2][2]=cmul(complex(0.5),cadd(se.el[1][1],so.el[1][1]));
s.el[2][3]=cmul(complex(0.5),csub(se.el[1][1],so.el[1][1]));
s.el[3][0]=s.el[2][1];
s.el[3][1]=s.el[2][0];
s.el[3][2]=s.el[2][3];
s.el[3][3]=s.el[2][2];  */

fprintf(fname,"%9.4le %9.4le\n",se.el[0][0].real,se.el[0][0].imag);
```

```
for(k=0;k<2;k++){

 for(j=0;j<2;j++){

  r=se.el[k][j];

  se.el[k][j].real=cabs(r);

  se.el[k][j].imag=phase(r);
/*
 printf("\r%9.4le %5.1lf ",se.el[k][j].real,se.el[k][j].imag); */

 }
/*
printf("\n"); */

}

    printf("\r%9.3e %9.4le %5.1lf %9.4le %5.1lf %9.4le %5.1lf %9.4le %5.1lf\n",

freq,

 se.el[0][0].real,se.el[0][0].imag,

 se.el[0][1].real,se.el[0][1].imag.

 se.el[1][0].real,se.el[1][0].imag,

 se.el[1][1].real,se.el[1][1].imag);


}
i=fclose(fname);
printf("i=%d",i);
}


struct CMATRIX get_s_params(freq,flag)
float freq;
int flag;
{
extern int n_max;
struct COMPLEX gamma_1,z_2,z_1;
struct CMATRIX t_acc,delta_t,s_conv(),get_dt();
struct COMPLEX zdiv,sum,diff,egamma1,emgamma1;
struct ZG z_gamma(),zg;
int n;
struct COMPLEX z(),gamma();
double *length,delta_1,zz;

/* printf("get_s_params[%f]\n",freq); */
n=0;

zz=cabs(z(n,length,freq,flag));
t_acc.el[0][0].real=(50.+zz)/(2*sqrt(50.*zz));
t_acc.el[0][0].imag=0.;
t_acc.el[0][1].real=(zz-50.)/(2*sqrt(50.*zz));
t_acc.el[0][1].imag=0.;
t_acc.el[1][0]=t_acc.el[0][1];
t_acc.el[1][1]=t_acc.el[0][0];


while (n< n_max){
/* if (n==0) printf("     seg#");
if (n>0) printf("\b");
if (n>10) printf("\b");
if (n>100) printf("\b");
if (n>1000) printf ("\b"); 

printf("%d",n); */

gamma_1=gamma(n,freq,flag);
z_1=z(n,length,freq,flag);
delta_1=n*(*length)/n_max;
/* while ((cabs(csub(z_1,z(n,length,freq,flag)))<delta_z) AND (n<n_max) AND
(cabs(cmul(cadd(gamma_1,gamma(n,freq,flag)),
ccomplex(0.5*((n*(*length)/n_max)-delta_1))))<delta_g1)){ */
n++;
/* printf("in loop n=%d, gam=%lf,%lf",n,gam); */
/* } */

/* printf("at D n=%d",n); */

z_2=z(n,length,freq,flag);
/* printf("at A n=%d. *length=%lf, delta_1=%lf, gam=%lf,%lf\n",
n,*length,delta_1,gam); */

delta_1=(n*(*length)/n_max)-delta_1;
gamma_1=cmul(ccomplex(0,5),cadd(gamma_1,gamma(n,freq,flag)));

/*  printf("at B n=%d, *length=%lf, delta_1=%lf, z_2=%lf,%lf\n",
n,*length,delta_1,(*z_2).real,(*z_2).imag); */

gamma_1=cmul(gamma_1,ccomplex(delta_1));


/*
printf("an=%d\n",n);
*/
zdiv=csqrt(cmul(z_1,z_2));
zdiv.real=zdiv.real*2.;
zdiv.imag=zdiv.imag*2.;
sum=cdiv(cadd(z_1,z_2),zdiv);
diff=cdiv(csub(z_2,z_1),zdiv);
/* printf("sum=%lf,%lf",sum); */

egamma1=cexp(gamma_1);
```

```
emgammal=cexp(cneg(gamma_l));
/* printf("egammal=%lf,%lf",egammal); */

delta_t.el[0][0]=cmul(sum,emgammal);
delta_t.el[0][1]=cmul(diff,emgammal);
delta_t.el[1][0]=cmul(diff,egammal);
delta_t.el[1][1]=cmul(sum,egammal);


/*
printf("d_t[11]=%f,%f, d_t[12]=%f,%f, d_t[21]=%f,%f, d_t[22]=%f,%f\n",delta_t);
*/

 t_acc=cmat_mul(t_acc,delta_t);
 }
zz=cabs(z(n_max,length,freq,flag));
delta_t.el[0][0].real=(50.+zz)/(2.*sqrt(50.*zz));
delta_t.el[0][0].imag=0.;
delta_t.el[0][1].real=(50.-zz)/(2.*sqrt(50.*zz));
delta_t.el[0][1].imag=0.;
delta_t.el[1][0]=delta_t.el[0][1];
delta_t.el[1][1]=delta_t.el[0][0];

t_acc=cmat_mul(t_acc,delta_t);

return (s_conv(t_acc));
}


struct CMATRIX s_conv(t_acc)
struct CMATRIX t_acc;
{
struct CMATRIX s;

s.el[0][0]=cdiv(t_acc.el[0][1],t_acc.el[1][1]);
s.el[0][1]=csub(t_acc.el[0][0],cdiv(cmul(t_acc.el[0][1],
                                t_acc.el[1][0]),t_acc.el[1][1]));
s.el[1][0]=cinv(t_acc.el[1][1]);
s.el[1][1]=cneg(cdiv(t_acc.el[1][0],t_acc.el[1][1]));

return(s);
}



struct COMPLEX gamma(n,freq,flag)
int n,flag;
float freq;
{
struct COMPLEX gam;
gam.imag=2.*PI*freq*3./C_LIGHT;
gam.real=0.;
```

```
/* printf("gamma=%f,%f  n=%d  freq=%f\n",gam.n,freq);  */

return(gam);
}


struct COMPLEX z(n,length,freq,flag)
int n,flag;
double *length;
float freq;
{
float xpos;
double sqn();
extern float totlength;
struct COMPLEX imp;
*length=totlength;
/* if (3*n<2*n_max){
if ((3*n)>n_max) OR (3*n<(n_max))
    imp.real=50.+100.*sin(3*PI*(3.*n-n_max)/n_max)*n_max/(3*PI*(3.*n-n_max));
if (3*n==n_max) imp.real=50.+100.;
}
else imp.real=50.;*/

/* if (2*n<=n_max) imp.real=(zw1+((zw2-zw1)*n)/(0.5*n_max));
 else imp.real=(zw2+((zw1-zw2)*(n-n_max/2.)))/(0.5*n_max));  zw1-zw2-zw1*/

xpos=n*totlength/n_max;
imp.real=50.+(zw2-50.)/2.*(1-(cos(xpos*(67.83+117.3*xpos))));

imp.imag=0.;
/*
 printf("z=%f,%f  n=%d  *length=%f",imp.n,*length);
*/
return(imp);
}

double sqn(arg)
double arg;
{
if (fabs(arg)<1e-30)  return(0);
else return(arg/fabs(arg));
}



double cabs(c)

struct COMPLEX c;

/* Function cabs() returns a double precision number which
```

```
   is the magnitude of the complex argument.

*/

{
return((sqrt(c.real*c.real+c.imag*c.imag)));
}

double phase(c)


/*
Function phase() returns a double precision number

which is the phase in degrees of the complex arg.  */

struct COMPLEX c;

{

/* printf("phase(%f,%f]\n",c.real,c.imag); */

if ((c.real==0.) AND (c.imag==0.)) return(0.);
        else return((180.*atan2(c.imag,c.real)/PI));

}

struct COMPLEX cneg(x)
struct COMPLEX x;
{
x.real=-x.real;
x.imag=-x.imag;
return(x);
}

struct COMPLEX cadd(x,y)
struct COMPLEX x,y;
{
x.real=x.real+y.real;
x.imag=x.imag+y.imag;
return(x);
}

struct COMPLEX cinv(x)
struct COMPLEX x;
{
double z;

z=x.real*x.real+x.imag*x.imag;

x.real=x.real/z;
x.imag=-x.imag/z;
return(x);
}

struct COMPLEX csub(x,y)
struct COMPLEX x,y;
{
struct COMPLEX z;
z.real=x.real-y.real;
z.imag=x.imag-y.imag;
return(z);
}

struct COMPLEX cmul(x,y)
struct COMPLEX x,y;
{
struct COMPLEX z;
z.real=x.real*y.real-x.imag*y.imag;
z.image=x.real*y.imag+x.imag*y.real;
return(z);
}

struct COMPLEX cdiv(x,y)
struct COMPLEX x,y;
{
y=cinv(y);
x=cmul(x,y);
return(x);
}

struct COMPLEX csqrt(x)
struct COMPLEX x;
{
double mag,ang;

mag=sqrt(cabs(x));
ang=phase(x)/2.;

x.real=mag*cos(ang);
x.imag=mag*sin(ang);

return(x);
}

struct COMPLEX cexp(x)
struct COMPLEX x;
{
double mag,ang;

mag=exp(x.real);

x.real=mag*cos(x.imag);
x.imag=mag*sin(x.imag);

return(x);
```

```
}

struct COMPLEX complex(x)
double x;
{
struct COMPLEX y;

y.real=x;
y.imag=0;

return(y);
}


struct CMATRIX cmat_mul(a,b)
struct CMATRIX a,b;
{
struct CMATRIX c;
int i,j;

for (i=0;i<2;i){
 for (j=0;j<2;j){
  c.el[i][j]=cadd(cmul(a.el[i][0],b.el[0][j]),cmul(a.el[i][1],b.el[1][j]));
   j++;
   }
 i++;
 }
return(c);
}
```

```
struct MS {        include "ms.h"

double el[2];

} mstrips();

/* e=eff diel const of substrate
   h=height of substrate in mm
   u=normalized width of line (w/h)
   g=normalized spacing of coupled lines (s/h)
  f9=freq !!! not in GHz
   double e,h,u,g,f9
flag=0,1,2 for single line, odd mode line, even mode line
   int flag */


struct MS mstrips(e,h,u,g,f9,flag)
float f9;
float e,h,u,g;
int flag;
{
double p[41],r[51];
double f,e1,a1,e2,e3,e4,a2,e5,a3,e6,zo,ro,z1,z2,z3,z4,z5,z6;
struct MS mscut;

 f9=f9/1e+9;
 f=f9*h;
 p[1]=1.+1ca((pow(u,4.)+pow((u/52.),2.))/(pow(u,4.)+.432))/49.;
 p[1]=p[1]+1ca(1.+pow((u/18.1),3.))/18.7;
 p[2]=.564*pow(((e-.9)/(e+3.)),.053);
 e1=(e+1.)/2.+(e-1.)*pow((1.+10./u),(-p[1]*p[2]))/2.;
 p[3]=.27488+(.6315+.525/pow((1.+.0157*f),20.))*u;
 p[3]=p[3]-.065683*exp(-8.7513*u);
 p[4]=.33622*(1.-exp(-.0442*e));
 p[5]=.0363*exp(-4.6*u)*(1.-exp(-pow((f/38.7),4.97)));
 p[6]=1.+2.751*(1.-exp(-pow((e/15.916),8.)));
 q1=p[3]*p[4]*pow(((.1844+p[5]*p[6])*f),1.5763);
 e2=e-(e-e1)/(1.+q1);
 p[7]=a*exp(-a)+u*(20.+pow(a,2.))/(10.+pow(a,2.));
 p[8]=1.+1ca((pow(p[7],4.)+pow((p[7]/52.),2.))/(pow(p[7],4.)+.432))/(49.);
 p[8]=p[8]+1ca(1.+pow((p[7]/18.1),3.))/18.7;
 p[9]=.564*pow(((e-.9)/(e+3.)),.053);
 e3=(e+1.)/2.+(e-1.)/2.*pow((1.+10./p[7]),(-p[8]*p[9]));
 p[10]=.747*e/(.15+e);
 p[11]=.7287*(e1-(e+1.)/2.)*(1.-exp(-.179*u));
 p[12]=p[10]-(p[10]-.207)*exp(-.414*u);
 p[13]=.593+.694*exp(-.562*u);
 e4=e1-(e1-(e+1.)/2.-p[11])*exp(-p[12]*pow(a,p[13]));
 p[14]=.334*exp(-3.3*pow((e/15.),3.))+.746;
 p[15]=p[14]*exp(-pow((f/18.),.368));
 p[16]=1.+4.069*p[15]*pow(a,.479)*exp(-1.347*pow(a,.595)-.17*pow(a,2.5));
 q2=p[3]*p[4]*pow(((.1844+p[16]+p[5]*p[6])*f),1.5763);
 e5=e-(e-e3)/(1.+q2);
 p[17]=.7168*(1.+1.076/(1.+.0576*(e-1.)));
 p[18]=.7913*(1.-exp(-pow((f/20.),1.424)))*atan(2.481*pow((e/8.),.946));
 p[18]=p[17]-p[18];
 p[19]=.242*pow((e-1.),.55);
 p[20]=.6366*(exp(-3.401*f/10.)-1.)*atan(1.263*pow((u/3.),1.629));
 p[21]=p[18]+(1.-p[18])/(1.+1.183*pow(u,1.376));
 p[22]=1.695*p[19]/(.414+1.605*p[19]);
 p[23]=.8928+.1072*(1.-exp(-.42*pow((f/20.),3.215)));
 p[24]=fabs(1.-.8928*p[21]/p[23]*(1.+p[20])*exp(-p[22]*pow(a,1.092)));
 q3=p[3]*p[4]*pow(((.1844+p[5]*p[6])*p[24]*f),1.5763);
 e6=e-(e-e4)/(1.+q3);
 zo=376.77;
 ro=6.+(2.*3.14159-6.)*exp(-pow((30.666/u),.7528));
 z1=zo/2./3.14159/sqrt(e1)*lca(ro/u+sqrt(1.+pow((2./u),2.)));
 r[1]=.0891*pow(e,1.4);
 if (r[1]>20.) r[1]=20.;
 r[2]=.267*pow(u,7.);
 if (r[2]>20.) r[2]=20.;
 r[3]=4.766*exp(-3.228*pow(u,.641));
 r[4]=.016+pow((.0514*e),4.524);
 r[5]=pow((f/28.843),12.);
 r[6]=22.2*pow(u,1.92);
 if (r[6]>20.) r[6]=20.;
 r[7]=1.206-.3144*exp(-r[1])*(1.-exp(-r[2]));
 r[8]=1.+1.275*(1.-exp(-.004625*r[3]*pow(e,1.674)*pow((f/18.365),2.745)));
 r[9]=5.086*r[4]*r[5]/(.3386*r[4])*exp(-r[6])/(1.+1.2992*r[5]);
 r[9]=r[9]*pow((e-1.),6.)/(1.+10.*pow((e-1.),6.));
 r[10]=.0004*pow(e,2.136)+.0184;
 r[11]=pow((f/19.47),6.)/(1.+9.619999e-02*pow((f/19.47),6.));
 r[12]=1./(1.+.00245*u*u);
 r[13]=.9408*pow(e2,r[8])-.9603;
 r[14]=(.9408-r[9])*pow(e1,r[8])-.9603;
 r[15]=.707*r[10]*pow((f/12.3),1.097);
 r[16]=1.+.0503*pow(e,2.)*r[11]*(1.-exp(-pow((u/15.),6.)));
 r[17]=r[7]*(1.-1.1241*r[12]/r[16]*exp(-.026*pow(f,1.15656)-r[15]));
 z2=z1*pow((r[13]/r[14]),r[17]);
 r[18]=.8695*pow(u,.194);
 r[19]=1.+q/1.33*pow(a,2.31)/5.29;
 r[20]=.1975*pow((16.6+pow((8.393993/a),6.)),(-.387));
 r[20]=r[20]+lca(pow(a,10.)/(1.+pow((a/3.4),10.)))/241.;
 r[21]=2.*r[18]/r[19]/(exp(-a)*pow(u,r[20])+(2.-exp(-a))*pow(u,(-r[20])));
 z3=z1*sqrt(e1/e3)/(1.-z1*sqrt(e1)*r[21]/zo);
 r[22]=1.794+1.14*lca(1.+.638/(a+.517001)*pow(a,2.43));
 r[23]=.2364+lca(pow(a,10.1)/(1.+pow((a/5.8),10.)))/281.3;
 r[23]=r[23]+lca(1.+.598*pow(a,1.154))/5.1;
 r[24]=(10.+190.*q*q)/(1.+82.3*q*q);
 r[25]=6.5+.95*lca(a)+pow((a/.15),5.);
 if (r[25]>20.) r[25]=20.;  /* use to compare with 29). (believed typo) */
 r[25]=exp(-r[25]);
 r[26]=(1./16.5*r[25])*lca(r[24]);
 r[27]=r[21]-r[22]/r[19]*exp(r[23]*pow(u,(-r[26])))*lca(u);
 z4=z1*sqrt(e1/e4)/(1.-z1*sqrt(e1)*r[27]/zo);
 r[28]=.893*(1.-.3/(1.+.7*(e-1.)));
 r[29]=2.121*pow((f/20.),4.91)/(1.+r[28]*pow((f/20.),4.91));
```

```
r[29]=r[29]*pow(g,.902)*exp(-2.87*a);
r[30]=1.+.088*pow((e/8.),5.1);
r[31]=1.+1.20*pow((e/15.),4.)/(1.+pow((e/15.),4.));
r[32]=1.887*pow(g,r[31]);
r[32]=r[32]/(1.+.41*pow((f/15.),3.)
                          *pow(u,(2./r[30]))/(.125+pow(u,(1.626/r[30]))));
r[32]=r[32]*exp(-1.5*pow(g,.84));
r[33]=r[32]*(1.+9./(1.+.40*pow((e-1.),2.)));
r[34]=.394*(1.-exp(-1.47*pow((u/7.),.672)))*(1.-exp(-4.25*pow((f/20.),1.87)));
r[35]=.61*(1.-exp(-2.13*pow((u/8.),1.593)))/(1.+6.54*pow(g,4.17));
r[36]=.21*pow(g,4.)/((1.+.018*pow(g,4.9))*(1.+.1*u*u)*(1.+pow((f/24.),3.)));
r[37]=r[36]*(9.00000e-02+1./(1.+.1*pow((e-1.),2.7)));
r[38]=42.54*pow(g,.133)*exp(-.812*g)*pow(u,2.5)/(1.+.033*pow(u,2.5));
r[38]=fabs(1.-r[38]);
r[8]=r[8]-r[29]+r[33]-r[34]+r[35]+r[37];
r[4]=.016+pow((.0514*r[38]*e),4.524);
r[9]=5.086*r[4]*r[5]/(.3838+.386*r[4])*exp(-r[6])/(1.+1.2992*r[5]);
r[9]=r[9]*pow((e-1.),6.)/(1.+10.*pow((e-1.),6.));
z5=pow(((.9408*pow(e2,r[8])-.9603)/((.9408-r[9])*pow(e1,r[8])-.9603)),r[17]);
z5=z3*z5;
r[42]=.3*f*f/(10.+f*f)*(1.+.7*pow((e-1.),2.)/.3/(5.+pow((e-1.),2.)));
r[46]=15.16/(1.+.196*pow((e-1.),2.));
r[43]=30.-r[46]-22.2*pow(((e-1.)/13.),12.)/(1.+3.*pow(((e-1.)/13.),12.));
r[44]=.4*pow(g,.84)*(1.+2.5*pow((e-1.),1.5)/(5.+pow((e-1.),1.5)));
r[45]=.149*pow((e-1.),3.)/(94.5+.038*pow((e-1.),3.));
r[39]=.925*pow((f/r[43]),1.536)/(1.+.3*pow((f/30.),1.536));
r[40]=1.+.005*f*r[44]/(1.+.812*pow((f/15.),1.9))/(1.+.025*u*u);
r[41]= 2.506*r[45]*pow(u,.834)/(3.575+pow(u,.834));
r[41]=r[41]*pow((f*(1.+1.3*u)/99.25),4.29);
z6=z2+(z4*pow((e6/e4),r[39]))-r[40]*z2)/(1.+r[41]+r[42]*pow((.46*q),2.2));
if (flag==0) mscut.el[0]=e2; /* single line */
if (flag==1) mscut.el[0]=e6; /* odd mode */
if (flag==2) mscut.el[0]=e5; /* even mode */
if (flag==0) mscut.el[1]=z2; /* single line */
if (flag==1) mscut.el[1]=z6; /* odd mode */
if (flag==2) mscut.el[1]=z5; /* even mode */

/*
printf("eeffsol(f=0) =%f\n",e1);
printf("eeffsol(f)   =%f\n",e2);
printf("eeffeve(f=0) =%f\n",e3);
printf("eeffodd(f=0) =%f\n",e4);
printf("eeffeve(f)   =%f\n",e5);
printf("eeffodd(f)   =%f\n",e6);
printf("zlsol(f=0)   =%f\n",z1);
printf("zlsol(f)     =%f\n",z2);
printf("zleve(f=0)   =%f\n",z3);
printf("zlodd(f=0)   =%f\n",z4);
printf("zleve(f)     =%f\n",z5);
printf("zlodd(f)     =%f\n",z6);
*/
return(mscut);
}
```

```
# define PI 3.14159265358979323846264338328      include "cmath.h"
# define AND &&
# define OR ||

struct COMPLEX{

double real,imag;

}cneg(),cinv(),cadd(),csub(),cmul(),cdiv(),csqrt(),cexp(),complex();

double cabs(),phase(),sgn();

/*
   sgn(double) returns the sign (signum) of the argument.
   +1 for positive, -1 for negative, 0 for small (<1e-30)
*/

double sgn(arg)
double arg;
{
if (fabs(arg)<1e-30)  return(0);
else return(arg/fabs(arg));
}



/*
     Function cabs() returns a double precision number which
     is the magnitude of the complex argument.
*/

double cabs(c)

struct COMPLEX c;
{
return((sqrt(c.real*c.real+c.imag*c.imag)));
}


/*
     Function phase() returns a double precision number
     which is the phase in degrees of the complex arg.
 */

double phase(c)

struct COMPLEX c;

{

if ((c.real==0.) AND (c.imag==0.)) return(0.);
        else return((180./PI)*atan2(c.imag,c.real));

}
```

```
/*
     Function cneg() returns a complex number which is
     the negative of the complex argument.
*/

struct COMPLEX cneg(x)
struct COMPLEX x;
{
x.real=-x.real;
x.imag=-x.imag;
return(x);
}


/*
     Function cadd() returns a complex number which is the
     sum of the two complex arguments.
*/

struct COMPLEX cadd(x,y)
struct COMPLEX x,y;
{
x.real=x.real+y.real;
x.imag=x.imag+y.imag;
return(x);
}


/*
     Function cinv() returns a complex number which is the
     inverse of the complex argument.
*/

struct COMPLEX cinv(x)
struct COMPLEX x;
{
double z;
z=x.real*x.real+x.imag*x.imag;
x.real=x.real/z;
x.imag=-x.imag/z;
return(x);
}


/*
     Function csub() returns a complex number which is the
     difference between the two complex arguments.
*/

struct COMPLEX csub(x,y)
struct COMPLEX x,y;
{
```

```
struct COMPLEX z;
z.real=x.real-y.real;
z.imag=x.imag-y.imag;
return(z);
}


/*
     Function cmul() returns a complex number which is the
     product of the two complex arguments.
*/

struct COMPLEX cmul(x,y)
struct COMPLEX x,y;
{
struct COMPLEX z;
z.real=x.real*y.real-x.imag*y.imag;
z.imag=x.real*y.imag+x.imag*y.real;
return(z);
}


/*
     Function cdiv() returns a complex number which is the
     quotient of the two complex arguments.
*/

struct COMPLEX cdiv(x,y)
struct COMPLEX x,y;
{
y=cinv(y);
x=cmul(x,y);
return(x);
}


/*
     Function csqrt() returns a complex number which is the
     square root of the complex argument.
*/

struct COMPLEX csqrt(x)
struct COMPLEX x;
{
double mag,ang;
mag=sqrt(cabs(x));
ang=phase(x)/2.;
x.real=mag*cos(ang);
x.imag=mag*sin(ang);
return(x);
}


/*
     Function cexp() returns a complex number which is the
     exponential of the complex argument.
*/

struct COMPLEX cexp(x)
struct COMPLEX x;
{
double mag,ang;
mag=exp(x.real);
x.real=mag*cos(x.imag);
x.imag=mag*sin(x.imag);
return(x);
}


/*   Function ccomplex() returns a complex number which is the
     type conversion of the (real) double argument.
*/

struct COMPLEX complex(x)
double x;
{
struct COMPLEX y;
y.real=x;
y.imag=0;
return(y);
}
```

```
# include "stdio.h"      program "fft.c"
# include "math.h"

# define PI 3.14159265358979323846264338328

  # define SWAP(a,b) tempr=(a);(a)=(b);(b)=tempr


main()
{
int i, i.k,no_of_points,isign;
char file[80],out[80],*fnm;
FILE *fname,*ofnam;
float dat[8200];
void realft();

printf("\n\nFFT routine\n\nEnter '1' for fft, '-1' for inv-fft:");
scanf("%d",&isign);
printf("\nEnter file name of data:");
 scanf("%s",file);
printf("\nEnter output file name:");
scanf("%s",out);
 fname=fopen(file,"r");
ofnam=fopen(out,"w");
 fscanf(fname,"%d",&no_of_points);
for (i=1;i<=4*no_of_points;i++) dat[i]=0;
 for(i=1;i<=no_of_points;i++) fscanf(fname,"%f %f",&dat[2*i-1],&dat[2*i]);

no_of_points=2*no_of_points;

/* for(i=1;i<=no_of_points;i++) {
if (i0*i<no_of_points) dat[2*i-1]=1.;
else dat[2*i-1]=0.;
dat[2*i]=0.;
} */
/* printf("\n before..."); */
realft(dat,no_of_points,isign);
/* printf("after fft"); */
fprintf(ofnam,"%d\n",2*no_of_points);
/* for(i=no_of_points;i>=1;i--) printf("%f\n%f\n",dat[2*i-1],dat[2*i]);*/
for(i=2*no_of_points;i>=1;i--) fprintf(ofnam,"%f\n",dat[i]);



}


void realft(data,n,isign)
float data[];
int n,isign;
{
int i,i1,i2,i3,i4,n2p3;
float c1=0.5,c2,h1r,h1i,h2r,h2i;
double wr,wi,wpr,wpi,wtemp,theta;
void four1();

theta=3.14159265358979323/(double) n;
if (isign==1){
  c2=-0.5;
  four1(data,n,1);
  } else {
  c2=0.5;
  theta=-theta;
  }
wtemp=sin(0.5*theta);
wpr= -2.0*wtemp*wtemp;
wpi=sin(theta);
wr=1.0+wpr;
wi=wpi;
n2p3=2*n+3;
for (i=2;i<=n/2;i++){
  i4=1+(i3=n2p3-(i2=1+(i1=i+i-1)));
  h1r=c1*(data[i1]+data[i3]);
  h1i=c1*(data[i2]-data[i4]);
  h2r= -c2*(data[i2]+data[i4]);
  h2i= c2*(data[i1]-data[i3]);
  data[i1]=h1r+wr*h2r-wi*h2i;
  data[i2]=h1i+wr*h2i+wi*h2r;
  data[i3]=h1r-wr*h2r+wi*h2i;
  data[i4]= -h1i+wr*h2i+wi*h2r;
  wr=(wtemp=wr)*wpr-wi*wpi+wr;
  wi=wi*wpr+wtemp*wpi+wi;
  }
if (isign==1) {
  data[1] = (h1r=data[1])+data[2];
  data[2] = h1r-data[2];
  } else {
  data[1]=c1*((h1r=data[1])+data[2]);
  data[2]=c1*(h1r-data[2]);
  four1(data,n,-1);
  }
}



void four1(data,nn,isign)
float data[];
int nn,isign;
{
int n,mmax,m, i,istep,j;
double wtemp,wr,wpr,wpi,wi,theta;
float tempr,tempi;
n=nn << 1;
j=1;
for(i=1;i<n;i+=2){
  if (j>i) {
    SWAP(data[j],data[i]);
```

```
    SWAP(data[i+1],data[j+1]);
    }
  m=n >> 1;
  while (m>=2 && j>m){
    j-=m;
    m >>=1;
    }
  j+=m;
  }
mmax=2;
while (n>mmax) {
  istep=2*mmax;
  theta=6.28318530717959/(isign*mmax);
  wtemp=sin(0.5*theta);
  wpr= -2.0*wtemp*wtemp;
  wpi=sin(theta);
  wr=1.0;
  wi=0.0;
  for (m=1;m<mmax;m+=2){
    for (i=m;i<=n;i+=istep){
      j=i+mmax;
      tempr=wr*data[j]-wi*data[j+1];
      tempi=wr*data[i+1]+wi*data[j];
      data[j]=data[i]-tempr;
      data[j+1]=data[i+1]-tempi;
      data[i] += tempr;
      data[i+1] += tempi;
      }
    wr=(wtemp=wr)*wpr-wi*wpi+wr;
    wi=wi*wpr+wtemp*wpi+wi;
    }
    mmax=istep;
  }
}
```