AN ABSTRACT OF THE THESIS OF

Maynard C. Falconer for the degree of Master of Science in Electrical and Computer Engineering presented on March 17, 1993.

Title: Time Domain Electromagnetic Simulation of VLSI Interconnects

Redacted for Privacy

Abstract approved: ___

Current trends in integrated circuit technology are continuing towards physically smaller components and related structures. Interconnects between circuit components have reached sub-micron dimensions which can not be modeled accurately by the current ideal microstrip analysis based techniques. These techniques often assume that the structure being modeled is perfectly conducting, smooth, uniform, and infinitely thin. At submicron dimensions the magnitude of the surface roughness may approach the dimensions of the interconnect structure. To investigate the effects of surface roughness in micron and submicron integrated circuit interconnects the finite-difference time-domain technique has been employed to simulate periodic surface roughness on interconnect structures. The simulation technique is validated by comparing the results with known microstrip results. The finite-difference time-domain simulations are in three dimensions and incorporate the finite conductivity of the interconnect, interconnect thickness, and surface roughness. The effects of interconnect size reduction, conductivity, and periodic surface roughness are investigated using the finite-difference time-domain technique.

Time Domain Electromagnetic Simulation of VLSI Interconnects

by

Maynard C. Falconer

A THESIS

submitted to

Oregon State University

in partial fulfillment of

the requirements for the degree of

Masters of Science

Completed March 17, 1993

Commencement June 1993

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 Graduate School

Date thesis is presented March 17, 1993

Typed for researcher by Maynard C. Falconer

ACKNOWLEDGEMENTS

I would like to thank my family for their support in my graduate work and for always being there when I needed them. Also to my graduate advisor, Professor Tripathi, who has helped in many ways from the beginning to the end of this effort. A special thanks to Dmitry Smolyansky who developed the signal processing tools to create the effective dielectric constant plot in the validation section. Thanks also to Neven Orthanovic for the transmission line simulations presented in the current analysis techniques section. A special thanks to Beate, who put up with me through the whole thesis process. And finally a thanks to all my fellow students at Oregon State University who engaged in lively conversation and discussion to help me complete this effort.

TABLE OF CONTENTS

INTRODUCTION	1
CURRENT ANALYSIS TECHNIQUES	3
FINITE-DIFFERENCE TIME-DOMAIN TECHNIQUE	10
Interface and Boundary Conditions	15
Pulse Injection Technique	17
Validation of Simulator	19
STRUCTURES UNDER TEST	21
Conductivity	23
RESULTS	26
Effects of scaling	43
Smooth versus rough case	45
Effects of varying the conductivity	49
Effects of surface roughness amplitude	54
Effects of surface roughness period	56
CONCLUSIONS	58
FUTURE WORK	59
BIBLIOGRAPHY	61
APPENDIX: Program listing	64

LIST OF FIGURES

Figure		Page
1.	Typical VLSI interconnects structure	1
2	a. Transmission line simulation of a 5x0.5μm interconnect. Input and output wave forms	5
	b. Transmission line simulation of a 0.5x0.2μm interconnect. Input and output wave forms	6
3.	Yee cell or unit cell used in Finite-Difference Time-Domain technique	14
4.	Test structure and location of input plane	18
5.	Comparison of FD-TD simulator and results from Zhang	20
6	a. Structure modeled with surface roughness	22
	b. Surface roughness model used in FD-TD simulation	22
7.	Electric field under dielectric interface and corresponding interconnect structure	27
8.	Electric field under dielectric interface for a smooth interconnect (σ=1e6) at	
	a. 200 time steps	28
	b. 300 time steps	29
	c. 400 time steps	30
	d. 500 time steps	31
	e. 600 time steps	32
	f. 700 time steps	33
	g. 800 time steps	34
9.	Electric field under dielectric interface for a rough (rgh=.3μm) interconnect (σ=1e6) at	

a. 200 time steps 35

b. 300 time steps 36

c. 400 time steps 37

d. 500 time steps 38

e. 600 time steps 39

f. 700 time steps 40

g. 800 time steps 41

10. Electric field under center of a smooth interconnect for various scalings 44

11. Electric field under center of a smooth interconnect (σ=1e6) 46

12. Electric field under center of a rough interconnect (σ=1e6) 47

13. Electric field under center of a smooth and rough interconnect (σ=1e6) 48

14. Effects of various conductivities on a perfectly smooth interconnect at a fixed distance (8μm) from the input plane as a function of time 50

15. Effects of various conductivities on a rough interconnect (0.3μm) at a fixed distance (8μm) from the input plane as a function of time 51

16. Electric field under center of rough interconnect (σ=1e3) 52

17. Electric field under center of rough interconnect (σ=1e5) 53

18. Electric field at a fixed distance (10μm) from the input plane as a function of time for various surface roughnesses 55

19. Electric field at a fixed distance (8μm) from the input plane as a function of time for various periods of surface roughness 57

TIME DOMAIN ELECTROMAGNETIC SIMULATION OF VLSI INTERCONNECTS

INTRODUCTION

Current trends in integrated circuit technology are continuing towards physically smaller components and related structures. Interconnects between circuit components have reached submicron dimensions which can not be modeled accurately by the current techniques. Current interconnect modeling techniques rely heavily on the methods for microstrip analysis which often assume that the structure being modeled is perfectly smooth and infinitely thin. Figure 1 shows a typical interconnect structure in VLSI circuits which is modeled in this investigation.

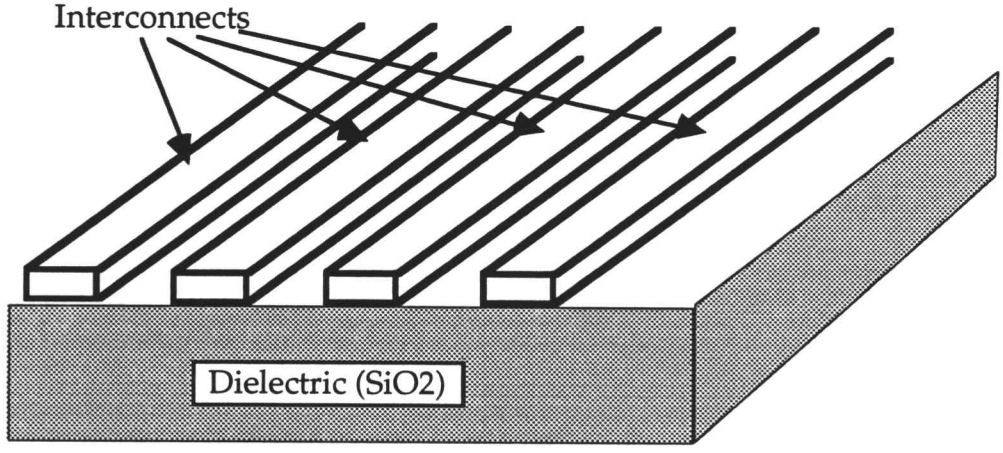


Fig. 1 Typical VLSI interconnects structure

Surface roughness on the interconnect structures occurs for several reasons including roughness of the mask, etching, and method of metalization. For interconnect structures larger than the micron size, surface roughness can typically be ignored, at submicron dimensions the magnitude of the surface roughness begins to approach the dimensions of the interconnect structure. The effects of surface roughness has been investigated for TEM waves and found to be significant in changing the loss and propagation coefficients for transmission line structures [1]. Very little has been reported on efforts to investigate the effects of surface roughness in non-TEM wave structures such as integrated circuit interconnects. To investigate the effects of surface roughness in sub-micron integrated circuit interconnects the finite-difference time-domain technique has been employed to simulate periodic surface roughness on interconnect structures.

CURRENT ANALYSIS TECHNIQUES

Conventional analysis of VLSI interconnects varies from simply computing the approximate RC time constant to implementing numerical techniques on computers. The goal of these techniques is to determine the pulse propagation time and the pulse attenuation to predict the effects of the interconnect on the wave form.

The simplest calculations are first order and give reasonable results only when the interconnect structure is relatively thin, smooth, and of simple geometry. The capacitance is assumed to be that of a parallel plate capacitor and the resistance calculated without skin effects or surface impedance included. Starting with these basic assumptions for R and C, several works have been done investigating the effects of scaling interconnects [2-3]. A more accurate analysis may be performed using microstrip calculations rather than the simple RC method. However the microstrip analysis is based on equations empirically fit to data and assume that the structure is infinitely thin which limits the frequency range which the analysis is valid over and limits the structures to which the analysis can be applied [4]. Additionally, microstrip analysis assumes a TEM field pattern which limits the accuracy achievable using this method.

More commonly the interconnect is modeled, in terms of resistance and capacitance per unit length, as a distributed transmission line. The telegrapher equations are used to calculate the propagation of the wave form down the structure:

$$\frac{\partial v(z,t)}{\partial z} = -Rv(z,t) - L\frac{\partial i(z,t)}{\partial t} \quad (1)$$

$$\frac{\partial i(z,t)}{\partial z} = -Gi(z,t) - C\frac{\partial v(z,t)}{\partial t} \quad (2)$$

The parameters R (resistance), C (capacitance), L (inductance), and G (conductance) are calculated from quasi-static field solutions and often assumed to be independent of frequency. This works well for narrow bandwidth signals and structures where R, C, L, and G can be accurately determined. A more accurate approach is to determine the frequency dependent line parameters R(ω), C(ω), L(ω), and G(ω) then transform the time domain wave form into the frequency domain using the Fourier transform or other applicable technique. Each frequency component is then translated down the interconnect and the time domain wave form is calculated by inverse Fourier transforming the resultant frequency components. A Z (impedance) or Y (admittance) matrix may be used to translate the frequency components down the transmission, or interconnect, structure:

$$\text{Y matrix} -> \begin{bmatrix} Yo \cdot \coth(\gamma\ell) & -Yo \cdot \operatorname{cosech}(\gamma\ell) \\ -Yo \cdot \operatorname{cosech}(\gamma\ell) & Yo \cdot \coth(\gamma\ell) \end{bmatrix} \quad (3)$$

$$\text{Z matrix} -> \begin{bmatrix} Zo \cdot \coth(\gamma\ell) & Zo \cdot \operatorname{cosech}(\gamma\ell) \\ Zo \cdot \operatorname{cosech}(\gamma\ell) & Zo \cdot \coth(\gamma\ell) \end{bmatrix} \quad (4)$$

Where:

$$\gamma = \alpha + j\beta = \sqrt{(R_{(\omega)} + j\omega L_{(\omega)})(G_{(\omega)} + j\omega C_{(\omega)})} \quad (5)$$

$$Zo = \frac{1}{Yo} = \sqrt{\frac{(R_{(\omega)} + j\omega L_{(\omega)})}{(G_{(\omega)} + j\omega C_{(\omega)})}} \quad (6)$$

It is common to assume that G=L=zero, which results in:

$$\gamma = \alpha + j\beta = \frac{1+j}{\sqrt{2}}\sqrt{\omega R_{(\omega)} C_{(\omega)}} \quad (7)$$

$$Zo = \frac{1}{Yo} = \frac{-1+j}{\sqrt{2}}\sqrt{\frac{R_{(\omega)}}{\omega C_{(\omega)}}} \quad (8)$$

R is calculated from the conductivity and cross section. C can be calculated from closed form equations, quasi-static programs, and for simple cases conformal mapping. Figure 2 shows the results of an interconnect simulation based on this transmission line technique.

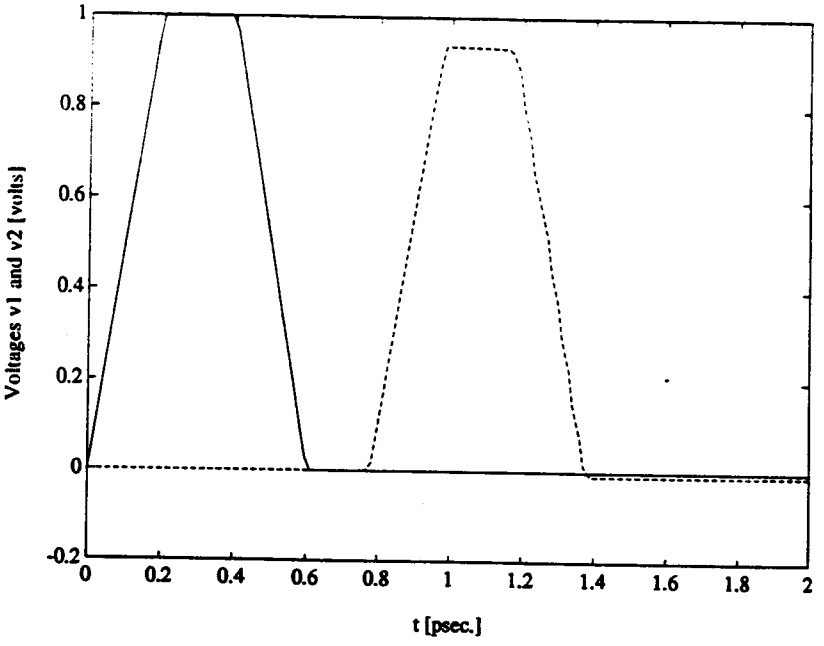


Fig. 2a Transmission line simulation of a 5x0.5μm interconnect. Input and output wave forms

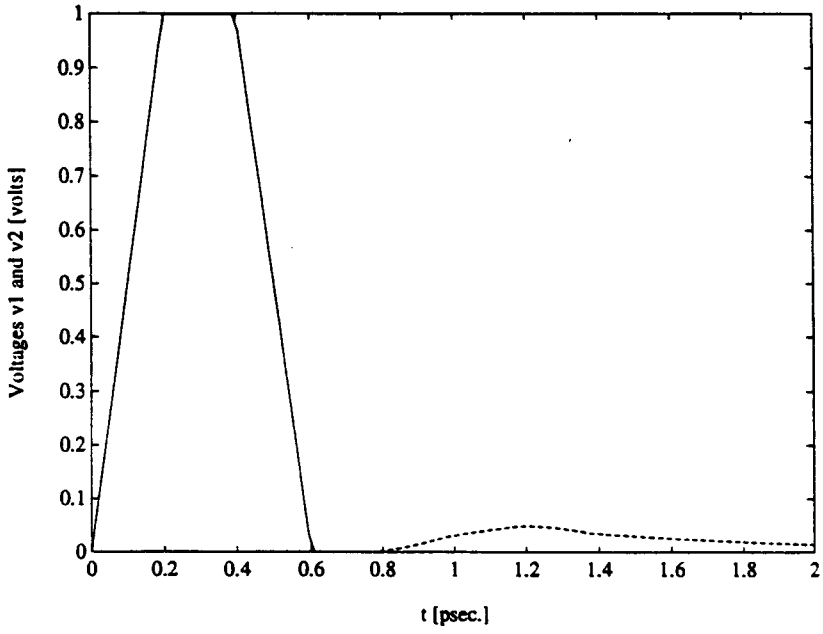


Fig. 2b Transmission line simulation of a 0.5x0.2μm interconnect. Input and output wave forms

If the dimensions of the structure are greater than the skin depth of the frequency components of interest, then skin effects can be incorporated into the resistance R. Conformal mapping can be used for simple geometries to model roughness, but the dimensions of the roughness must be greater than the mean free path length of electrons in the material. All of the transmission methods mentioned above are limited by the inability to account for radiative effects caused by discontinuities.

To incorporate surface roughness into the calculation of the transmission line parameters, Sanderson[1] has introduced the concept of surface displacement. Surface roughness effects all the transmission line

parameters except the conductance, (G), which is typically assumed to be zero. Sanderson uses perturbation theory for TEM waves to develop analytical formulas for the effects of surface roughness. The formulas calculate displacement distances for the surface, either positive or negative, from which each of the transmission line parameters can be re-calculated to incorporate the surface roughness. The surface displacement is the distance which a smooth surface of a transmission line needs to be moved to have the same effect as the presence of the surface roughness. As an example the resistance per unit length of a line is calculated from the conductivity and the cross section of the line. Surface roughness would increase the resistance per unit length and could be modeled by decreasing the cross section, or displacing the smooth surface inwards. Sanderson found that at low frequencies surface roughness causes a delay line effect, and at high frequencies it also increases attenuation and changes the characteristic impedance of the line. At intermediate frequencies, the surface roughness causes the transmission line parameters to become functions of frequency. All of the surface roughness effects were shown to be dependent on the square of the roughness amplitude. Sanderson's work is limited to TEM structures and starts to show large errors when other modes are present. Experimental results show that the surface roughness causes increased effects, compared to the TEM predictions, when the wave structure is not TEM. The displacement surface method has the limits of being applicable to only TEM, or quasi-TEM, wave forms and is applicable only when the displacement distance is small compared to the structure dimension. Both of these restrictions limit the usefulness of the surface displacement method to interconnect structures, however in general it is a powerful concept.

Several numerical methods have been used to analyze VLSI interconnects including spectral domain and method of moments. The spectral domain technique uses a Fourier transform of space to determine the current distribution on the interconnect structure. Thick structures can be modeled by using multiple strips in close proximity to each other. Typically the structure modeled with the spectral domain technique is assumed to be uniform in the direction of propagation, however it maybe possible to model periodic structures in the third dimension. Non-periodic structures can not be modeled with the spectral domain technique which limits the application of this technique. Additionally the spectral domain analysis of the structure is in the frequency domain thus requiring multiple iterations of the calculations to characterize a system as a function of frequency.

The method of moments uses Green's functions to represent the transverse current distribution on the interconnect structure. Three dimensional structures can be modeled with this technique, but they can only be analyzed one frequency at a time. While both the spectral domain and method of moments are very computationally efficient at analyzing a structure at a single frequency, they both require complex preliminary work to design the code for a specific system to be modeled.

The finite-difference time-domain (FD-TD) technique has the capabilities of modeling a complex three dimension structures and characterizing the system under test with a single pulse. The technique is relatively easy to implement, but may require a large amount of computer memory depending on the complexity of the structure being modeled. With the recent advances in computer technology, the memory requirements for the

FD-TD method has become less of a problem and the technique has become much more attractive.

FINITE-DIFFERENCE TIME-DOMAIN TECHNIQUE

The finite-difference time-domain (FD-TD) method was first introduced by K.S. Yee [5] to solve electromagnetic scattering problems. The benefits of the FD-TD method is that it can be readily implemented and used to solve for the time domain response of three dimensional structures. The main draw backs of the method are the large memory requirements and speed needed for the analysis of complex structures. With modern computers becoming faster and having larger memory capabilities, the above draw backs are becoming more manageable. The calculation of the electromagnetic fields are based on the time and space discretisation of Maxwell's equations [6-7].

$$\nabla \times \vec{\mathbf{E}} = -\mu \frac{\partial \vec{\mathbf{H}}}{\partial t} \qquad (9)$$

$$\nabla \times \vec{\mathbf{H}} = \sigma \vec{\mathbf{E}} + \varepsilon \frac{\partial \vec{\mathbf{E}}}{\partial t} \qquad (10)$$

The permeability μ ,permittivity ε ,and conductivity σ are time-independent for these calculations, but can be altered to be time-dependent. Expanding the curl operation in equations (9) and (10) and expanding the fields into their x, y, and z components gives:

$$\bar{a}_x\left(\frac{\partial E_z}{\partial y}-\frac{\partial E_y}{\partial z}\right)+\bar{a}_y\left(\frac{\partial E_x}{\partial z}-\frac{\partial E_z}{\partial x}\right)+\bar{a}_z\left(\frac{\partial E_y}{\partial x}-\frac{\partial E_x}{\partial y}\right)= -\mu\frac{\partial}{\partial t}\left(\bar{a}_x H_x+\bar{a}_y H_y+\bar{a}_z H_z\right) \tag{11}$$

$$\bar{a}_x\left(\frac{\partial H_z}{\partial y}-\frac{\partial H_y}{\partial z}\right)+\bar{a}_y\left(\frac{\partial H_x}{\partial z}-\frac{\partial H_z}{\partial x}\right)+\bar{a}_z\left(\frac{\partial H_y}{\partial x}-\frac{\partial H_x}{\partial y}\right)= \sigma\left(\bar{a}_x E_x+\bar{a}_y E_y+\bar{a}_z E_z\right)+\varepsilon\frac{\partial}{\partial t}\left(\bar{a}_x E_x+\bar{a}_y E_y+\bar{a}_z E_z\right) \tag{12}$$

Taking only the x components of equations (11) and (12):

$$\bar{a}_x\left(\frac{\partial E_z}{\partial y}-\frac{\partial E_y}{\partial z}\right)=-\mu\frac{\partial}{\partial t}\left(\bar{a}_x H_x\right) \tag{13}$$

$$\bar{a}_x\left(\frac{\partial H_z}{\partial y}-\frac{\partial H_y}{\partial z}\right)=\sigma\left(\bar{a}_x E_x\right)+\varepsilon\frac{\partial}{\partial t}\left(\bar{a}_x E_x\right) \tag{14}$$

Taking the finite difference of equations (13) and (14) results in:

$$\frac{\left(E_z^n(i,j,k+0.5)-E_z^n(i,j,k+0.5)+E_y^n(i,j+0.5,k+1)-E_y^n(i,j+0.5,k)\right)}{\Delta x}= \frac{-\mu}{\Delta t}\left(H_x^{n+0.5}(i,j+0.5,k+0.5)-H_x^{n-0.5}(i,j+0.5,k+0.5)\right) \tag{15}$$

$$\frac{\left(H_z^{n+0.5}(i+0.5,j+0.5,k)-H_z^{n+0.5}(i+0.5,j-0.5,k)+H_y^{n+0.5}(i+0.5,j,k-.05)-H_y^{n+0.5}(i+0.5,j,k+0.5)\right)}{\Delta x}= \sigma\left(E_x^n(i+0.5,j,k)\right)+\frac{\varepsilon}{\Delta t}\left(E_x^{n+1}(i+0.5,j,k)-E_x^n(i+0.5,j,k)\right) \tag{16}$$

Where the superscript is the time step, i is the number of steps in the x direction, j the number of steps in the y direction, k the number of steps in the

z direction. It is also assumed that $\Delta x=\Delta y=\Delta z$. The field components are positioned within the three dimension discretized space in a Yee, or unit, cell as shown in Figure 3 according to the indices i, j, and k. Solving equations (15) and (16) for the new electric and magnetic field components:

$$H_x^{n+0.5}(i,j+0.5,k+0.5) = H_x^{n-0.5}(i,j+0.5,k+0.5) - \frac{\Delta t}{\mu \cdot \Delta x}\left(E_z^n(i,j,k+0.5) - E_z^n(i,j,k+0.5) + E_y^n(i,j+0.5,k+1) - E_y^n(i,j+0.5,k)\right) \quad (17)$$

$$\left(\frac{\Delta t}{\mu \cdot \Delta x}\right)E_x^{n+1}(i+0.5,j,k) = \left(\frac{2\varepsilon_r\varepsilon_0 - \sigma \cdot \Delta t}{2\varepsilon_r\varepsilon_0 + \sigma \cdot \Delta t}\right)\left(\frac{\Delta t}{\mu \cdot \Delta x}\right)E_x^n(i+0.5,j,k) + \left(\frac{\left(\frac{c \cdot \Delta t}{\Delta x}\right)^2}{\varepsilon_r + \frac{\sigma \cdot \Delta t}{2\varepsilon_0}}\right) \quad (18)$$
$$\left(H_z^{n+0.5}(i+0.5,j+0.5,k) - H_z^{n+0.5}(i+0.5,j-0.5,k) + H_y^{n+0.5}(i+0.5,j,k-0.5) - H_y^{n+0.5}(i+0.5,j,k+0.5)\right)$$

The other four equations follow the same derivation:

$$H_y^{n+0.5}(i,j-0.5,k) = H_y^{n-0.5}(i,j-0.5,k) - \frac{\Delta t}{\mu \cdot \Delta x}\left(E_x^n(i,j-0.5,k+0.5) - E_x^n(i,j-0.5,k-0.5) + E_z^n(i-0.5,j-0.5,k) - E_z^n(i+0.5,j-0.5,k)\right) \quad (19)$$

$$\left(\frac{\Delta t}{\mu \cdot \Delta x}\right)E_y^{n+1}(i+0.5,j,k+0.5) = \left(\frac{2\varepsilon_r\varepsilon_0 - \sigma \cdot \Delta t}{2\varepsilon_r\varepsilon_0 + \sigma \cdot \Delta t}\right)\left(\frac{\Delta t}{\mu \cdot \Delta x}\right)E_y^n(i+0.5,j,k+0.5) + \left(\frac{\left(\frac{c \cdot \Delta t}{\Delta x}\right)^2}{\varepsilon_r + \frac{\sigma \cdot \Delta t}{2\varepsilon_0}}\right) \quad (20)$$
$$\left(H_x^{n+0.5}(i+0.5,j,k+1) - H_x^{n+0.5}(i+0.5,j,k) + H_z^{n+0.5}(i,j,k+0.5) - H_z^{n+0.5}(i+1,j,k+0.5)\right)$$

$$H_z^{n+0.5}(i,j,k+0.5) = H_z^{n-0.5}(i,j,k+0.5) - \frac{\Delta t}{\mu \cdot \Delta x}\left(E_x^n(i,j+0.5,k+0.5) - E_x^n(i,j-0.5,k+0.5) + E_y^n(i-0.5,j,k+0.5) - E_y^n(i+0.5,j,k+0.5)\right) \quad (21)$$

$$\left(\frac{\Delta t}{\mu\cdot\Delta x}\right)E_z^{n+1}(i+0.5,j-0.5,k)=\left(\frac{2\varepsilon_r\varepsilon_0-\sigma\cdot\Delta t}{2\varepsilon_r\varepsilon_0+\sigma\cdot\Delta t}\right)\left(\frac{\Delta t}{\mu\cdot\Delta x}\right)E_z^{n}(i+0.5,j-0.5,k)+\left(\frac{\left(\frac{c\cdot\Delta t}{\Delta x}\right)^2}{\varepsilon_r+\frac{\sigma\cdot\Delta t}{2\varepsilon_0}}\right) \quad (22)$$

$$\left(H_y^{n+0.5}(i+1,j-0.5,k)-H_y^{n+0.5}(i,j-0.5,k)+H_x^{n+0.5}(i+0.5,j-1,k)-H_x^{n+0.5}(i+0.5,j,k)\right)$$

As seen in equations (17-22) the new value of a field component is dependent only on the field components at the previous time step, therefore it is sufficient to retain only the information contained in the previous time step.

Using the discretized Maxwell's equations, boundary conditions, and initial field conditions the electromagnetic fields can be solved for in the time and space domains. The required boundary conditions and their implementation are discussed in several papers [8-9].

The accuracy of the FD-TD technique is dependent on the selection of the grid spacing (Δx) and the time increment (Δt). The grid spacing must be small compared to the structure dimensions and to the wavelength of the input signal. The stability criteria relating the time and space increments is:

$$\Delta t \le \frac{\Delta x}{c\sqrt{3}} \quad (23)$$

Further modifications can be made to the FD-TD equations including irregular grid spacing, incorporating metal strips smaller than the grid spacing, and incorporating dielectric structures smaller than the grid spacing [10-11].

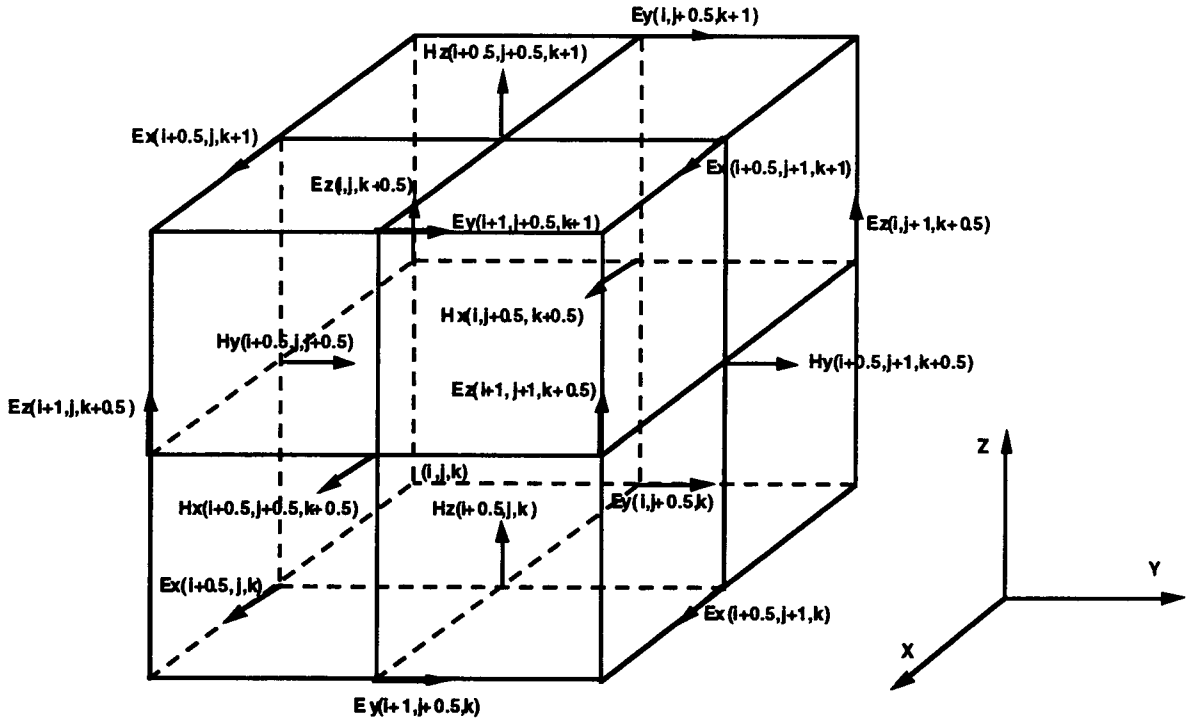


Fig. 3 Yee cell or unit cell used in the Finite-Difference Time-Domain technique.

Interface and Boundary Conditions

The interface between materials with different dielectric constants or different conductivities in the FD-TD method require modifications to the basic equations previously presented. The new FD-TD equations are derived as follows.

Suppose Ex is tangentially situated on the interface between two different materials, media 1 and media 2. The interface plane of the two materials is in the X-Z plane. The relevant equations in the two materials are:

$$\varepsilon_1\left(\frac{\partial E^1x}{\partial t}\right)+\sigma_1 E^1x=\frac{\partial H^1z}{\partial y}-\frac{\partial H^1y}{\partial z} \qquad (24)$$

$$\varepsilon_2\left(\frac{\partial E^2x}{\partial t}\right)+\sigma_2 E^2x=\frac{\partial H^2z}{\partial y}-\frac{\partial H^2y}{\partial z} \qquad (25)$$

Since the tangential electric fields across a boundary are continuous:

$$E^1x=E^2x \qquad (26)$$

Adding equations (24) and (25) yields:

$$\left(\frac{\varepsilon_1+\varepsilon_2}{2}\right)\frac{\partial Ex}{\partial t}+\left(\frac{\sigma_1+\sigma_2}{2}\right)Ex=\frac{\partial Hz}{\partial y}-\frac{\partial Hy}{\partial z} \qquad (27)$$

Discretization of equation (27) leads to a modified version of equation (10):

$$\left(\frac{\Delta t}{\mu\cdot\Delta x}\right)E_x^{n+1}(i+0.5,j,k)=\left(\frac{(\varepsilon_{r1}+\varepsilon_{r2})\varepsilon_0-\left(\frac{\sigma_1+\sigma_2}{2}\right)\Delta t}{(\varepsilon_{r1}+\varepsilon_{r2})\varepsilon_0+\left(\frac{\sigma_1+\sigma_2}{2}\right)\Delta t}\right)\left(\frac{\Delta t}{\mu\cdot\Delta x}\right)E_x^{n}(i+0.5,j,k)+$$

$$\left(\frac{\left(\frac{c\cdot\Delta t}{\Delta x}\right)^2}{\frac{(\varepsilon_{r1}+\varepsilon_{r2})}{2}+\frac{\sigma\cdot\Delta t}{2\varepsilon_0}}\right)\left[H_z^{n+0.5}(i+0.5,j+0.5,k)-H_z^{n+0.5}(i+0.5,j-0.5,k)+\right. \quad (28)$$

$$\left.H_y^{n+0.5}(i+0.5,j,k-0.5)-H_y^{n+0.5}(i+0.5,j,k+0.5)\right]$$

Since the simulation is not incorporating any magnetic materials, the magnetic field components are all continuous across the interface and therefore require no modification.

Pulse Injection Technique

The excitation pulse used in this effort was a finite rise-time and finite fall-time pulse which roughly approximates a signal generated by a digital system. Since the exact field patterns are not known for the entire input plane, an approximate field structure is used to inject a pulse into the simulated interconnect structure. After the pulse has propagated a small distance the prominate edge effect field structure of the pulse manifests itself and the appropriate fringing fields are established.

The input pulse is generated by exciting the vertical tangential electric field components of the input plane, between the interconnect and the substrate ground plane (see Figure 4). The horizontal electric field component, on the input plane, is set to zero to simulate a Dirchlet or metal boundary. Elsewhere on the input plane the full Dirchlet boundary condition exists. This injection technique has been used by Zhang [7,12] Liang [13], and Sheen [14]. To further confirm this technique, a pulse was established in the center of the interconnect structure as an initial condition and observed as it propagated down the interconnect structure in both the positive and negative Z directions.

One draw back of injecting the pulse from the input plane is the creation of a large D.C. magnetic field on the front surface due to the Dirchlet boundary condition. The presence of the D.C. magnetic field causes an immediate drop in the pulse amplitude and a distortion of the pulse shape. Both the amplitude loss and the pulse distortion are unfortunate, but do not alter the propagation characteristics of the pulse down the simulated

interconnect structure. A Neumann, or magnetic wall, boundary condition was also tested on the input plane and was found to produce comparable amplitude losses and pulse distortion. Both Zhang [12] and Sheen [14] have also noted the creation and effects of the D.C. magnetic field and concluded that it has no influence on the traveling pulse.

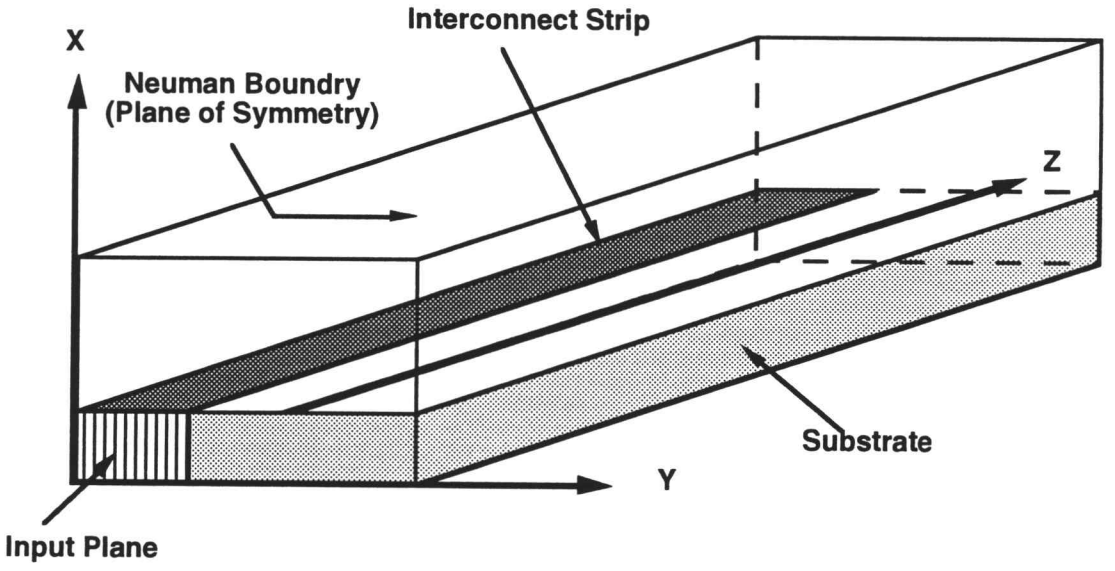
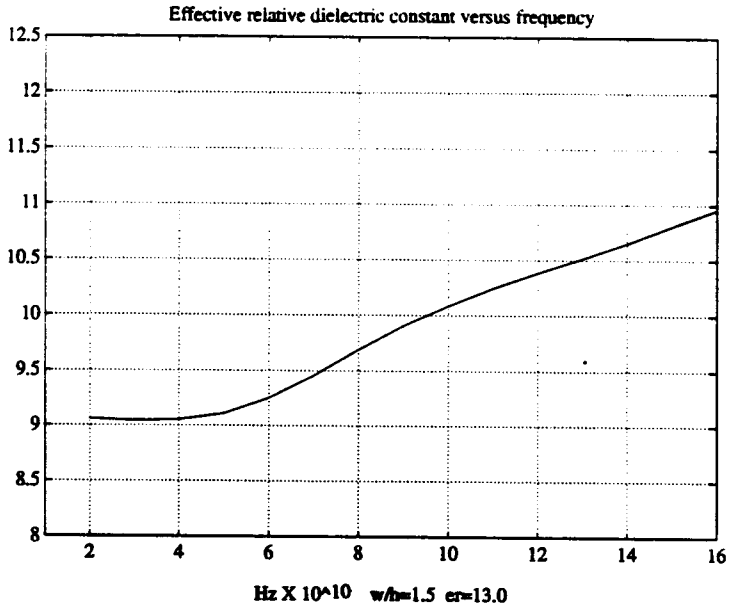


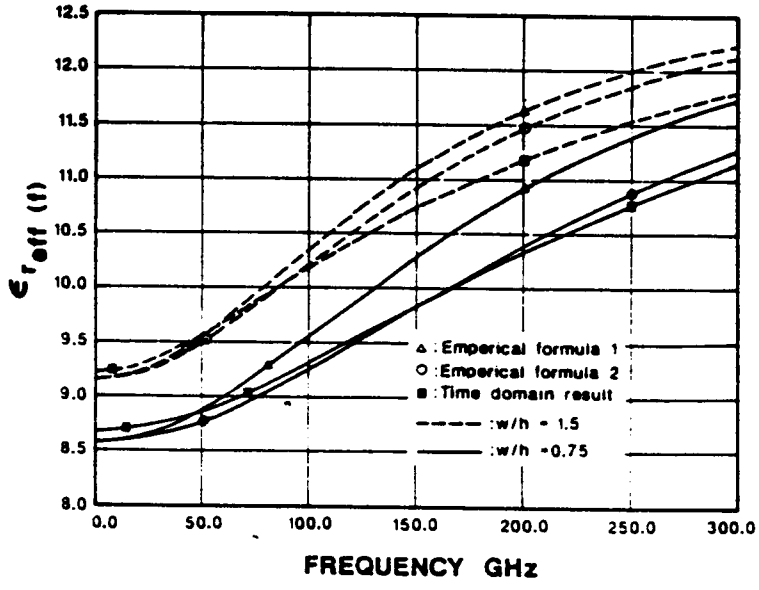
Fig. 4 Test structure and location of input plane

Validation of Simulator

The work published by Zhang [7] offers an easy structure to compare results with. Zhang simulated a microstrip structure on top of a GaAs substrate. The microstrip used was infinitely thin and a perfect conductor. In an earlier version of the FD-TD simulator the results of Zhang were duplicated, but since the current version of the simulator incorporates many features which also need to be checked a slightly altered test structure was used. The main difference is Zhang used an infinitely thin microstrip while this test run utilized a thickness of 12.5μm, or one grid spacing. Zhangs' simulation did not incorperate losses and used perfect conductors, to simulate this the loss factors were set to zero and the conductivity of the metal was set to 10^24 [siemens/m] to simulate a perfect conductor. Figure 5 shows the comparision of the two simulator results for the effective dielectric constant of a microstrip with w/h=1.5 and er=13.0. There is a slight difference between the two simulation results, but this is to be expected due to the finite thickness of the metal strip.



FD-TD simulation of effective dielectric constant



Effective dielectric constant from Zhang

Fig. 5 Comparison of FD-TD simulator and results from Zhang

STRUCTURES UNDER TEST

The FD-TD method was used to model surface roughness on a metal strip above a ground plane as shown in figure 6. Periodic, square surface roughness, as shown in figure 6, was compared to a smooth strip with no roughness for several different conductivities, magnitudes of roughness, periods of roughness, and strip widths. The interconnects modeled were all located 2μm above a perfect ground plane and were 0.5μm thick in the vertical or 'X' direction. The dielectric under the strip had a relative dielectric constant of er=4.0, and elsewhere the relative dielectric constant was set to unity. The typical three dimensional grid used to model the interconnect structures was 80x24x179 Yee cells with a step size of 0.1μm and a stability factor of 0.515.

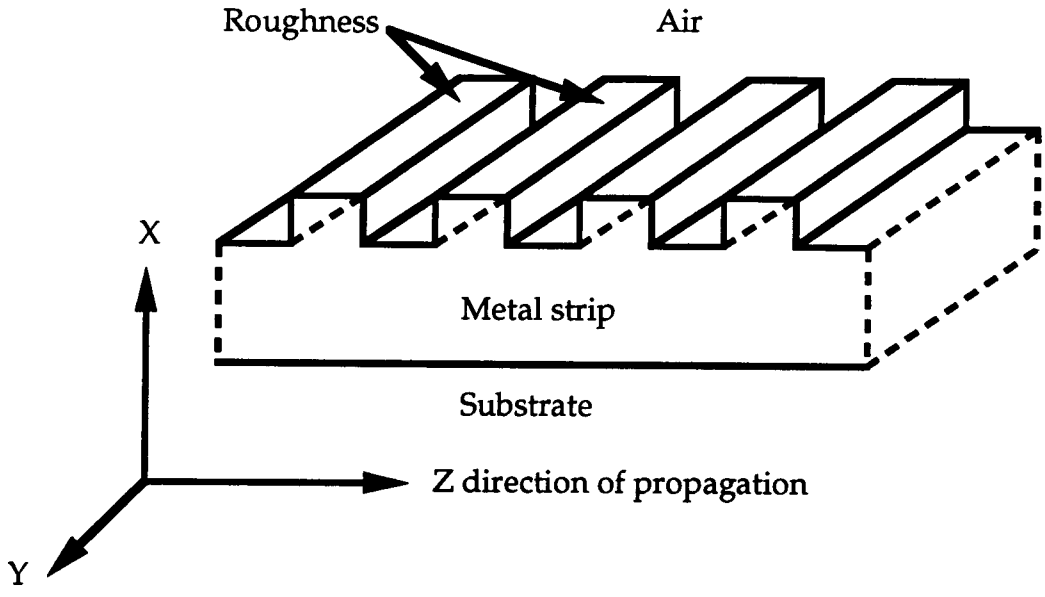


Fig. 6a Structure modeled with surface roughness

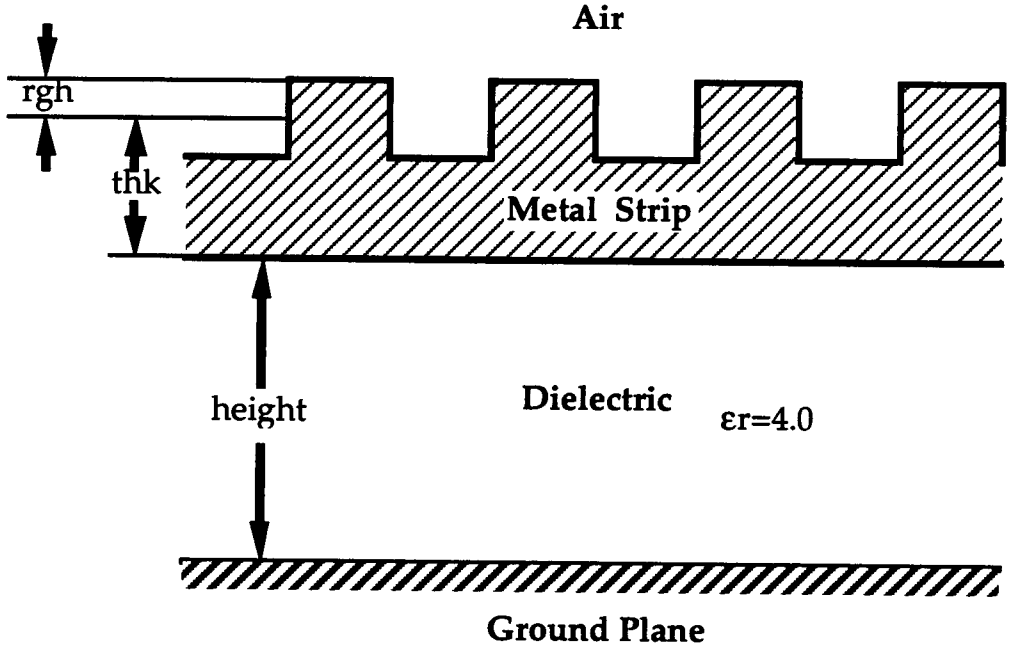


Fig. 6b Surface roughness model used in FD-TD simulation

Conductivity

The electrical conductivity of the material used to construct an interconnect strongly influences the effects of surface roughness, loss, and propagation delay along the interconnect structure. Several materials of varying electrical conductivity are utilized in the manufacture of solid state interconnects for various reasons including; adhesion to substrate, electromigration, oxidation properties, etc... Due to these diverse reasons it is not always possible to select a material with a high conductivity. Table 1 lists some of the materials used in the construction of interconnects and the bulk conductivity associated with each. The typical interconnect conductivity from Table 1 is $4x10^7$ [1/Ω-m] or less. Interconnects are also constructed from layers of several different materials, but these composite structures are not considered here.

The electrical bulk conductivity of a medium is derived from several factors including the mean free electron path length.

$$\sigma_B = \frac{q^2 n \lambda_0}{m^* v} \quad (29)$$

Where q is the electron charge, n is the number of free electrons per unit volume, m* is the electrons effective mass, and v is the electron velocity at the Fermi surface [15]. When the physical dimensions of the structure are on the order of, or smaller than, the mean free electron path length ,λ_0., the conductivity of the material will deviate from the bulk conductivity. This deviation occurs due to the collision of electrons with the material surfaces

Table 1 interconnect materials and associated conductivities

Heavily doped poly-SI [16]	$\sigma = 10x10^{-3} - 50x10^{3}$ [1/Ω-m]
Aluminum [17]	$\sigma = 38x10^{6}$ [1/Ω-m]
Refractory Metals: [17]	
Molybdenum [Mo]	$\sigma = 21x10^{6}$ [1/Ω-m]
Tungsten [W]	$\sigma = 18x10^{6}$ [1/Ω-m]
Refractory Metal Silicides: [15]	
$MoSi_2$	$\sigma = 1x10^{6} - 1.1x10^{6}$ [1/Ω-m]
WSi_2	$\sigma = 1.43x10^{6}$ [1/Ω-m]
$TaSi_2$	$\sigma = 1.8x10^{6} - 2.9x10^{6}$ [1/Ω-m]
$TiSi_2$	$\sigma = 4x10^{6} - 7.7x10^{6}$ [1/Ω-m]
Refractory Metal Nitrides: [15]	
Electrical conductivity is comparable to those of refractory metal silicides.	
Amorphous Metal Films: [15]	
include (Nb-Ni, Mo-Ni, Si-W, Mo-Si)	$\sigma = 1x10^{6}$ [1/Ω-m]

causing a shorter mean free electron path length than would normally occur in the material bulk. For a rectangular structure with thickness t and width w, the modified conductivity due to surface collisions can be determined from [15].

$$\frac{\sigma}{\sigma_B} = 1 - \frac{3}{8}\left(\frac{1}{\kappa_1} + \frac{1}{\kappa_2}\right) + \frac{4}{5\pi\kappa_1\kappa_2} \qquad \kappa_1 >> 1 \quad \kappa_2 >> 1 \quad (30)$$

$$\frac{\sigma}{\sigma_B} = \frac{3}{4}(\kappa_1 + \kappa_2)\ln\left(\sqrt{2}+1\right) - \frac{1}{4}\left(\frac{\kappa_2^2}{\kappa_1} + \frac{\kappa_1^2}{\kappa_2}\right)\left(1-\sqrt{2}\right) \qquad \kappa_1 << 1 \quad \kappa_2 << 1 \quad (31)$$

Where σ_B is the bulk conductivity, k1=t/λo, and k2=w/λo. A general integral form for calculating the modified conductivity can be found in the work by Cohen [18].

The aforementioned equations for the modified conductivity assume that all surface scattering is purely inelastic. This assumption was studied by Ziman [19] who showed that surface scattering is essentially inelastic when the surface roughness is larger than an electrons' wavelength (0.5nm for Au) which is a valid assumption for this effort. Both elastic and inelastic collisions can be included in the modified conductivity equations [20].

As an example of the modified conductivity, the mean free path lengths for metals at room temperature are on the order of 10^{-7}[m] or less [18]. In a structure 0.5um x 0.5um with $\lambda=10^{-7}$[m] the new conductivity would be:

$$\sigma=0.86\ \sigma_B$$

As the interconnect dimensions become smaller, the conductivity of the material becomes smaller, thus altering the values given in Table 1. At submicron sizes the this effect invalidates simple scaling rules.

RESULTS

Using the time-domain method described previously and the interconnect structures presented above the effects of periodic surface roughness on the delay and distortion of typical time domain signal waveforms have been investigated. The FD-TD simulations determine space-time distribution of the field structures everywhere in the three dimensional simulation space. The resulting field patterns are quite clear and offer insights to the specific electromagnetic structures. For all simulations, except the scaling ones, the structure under test was 0.5μm wide, 0.5μm thick, 2μm above the ground plane, and the substrate dielectric constant was 4. Figures 8 through 9 show the vertical electric field on the plane directly under the interconnect strip for a conductivity of 1e6 [1/Ω-m] for the smooth case and a roughness of 0.3 [μm] as a function of space at a set time. Figure 7 shows the corresponding interconnect structure for figures 8 and 9. These figures show the prominate edge effect start to form and then dominate in a very short time. In the final time steps the pulse is reflected from the simulation edge with a reflection coefficient of -1 due to the Dirchlet boundary condition. The time step used in all the simulations was dt=172e-18[sec] unless otherwise noted. By the time the simulation reached 800 time steps in figures 5 and 6, significant deviations between the smooth and rough cases can be seen. Note that the scale of the electric field amplitude is not consistent from figure to figure unless a scale is given.

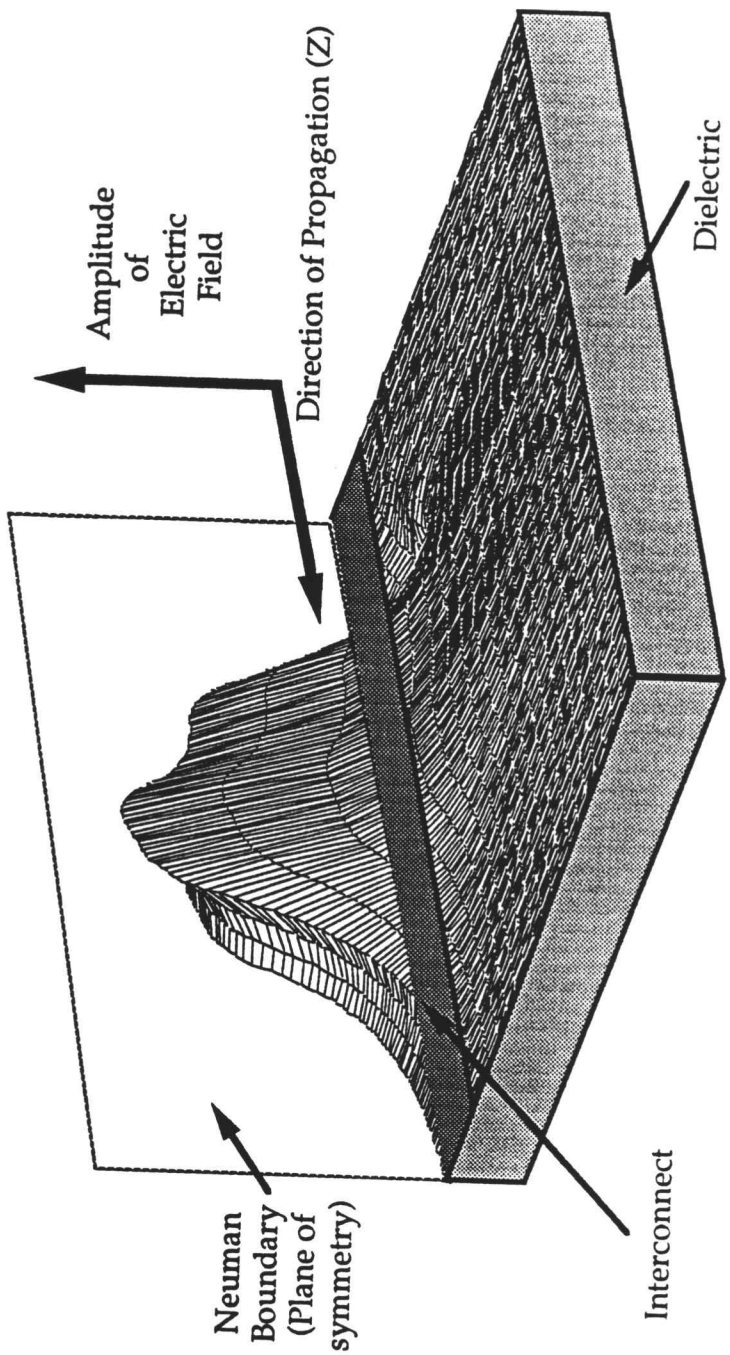


Fig. 7 Electric field under dielectric interface and corresponding interconnect structure

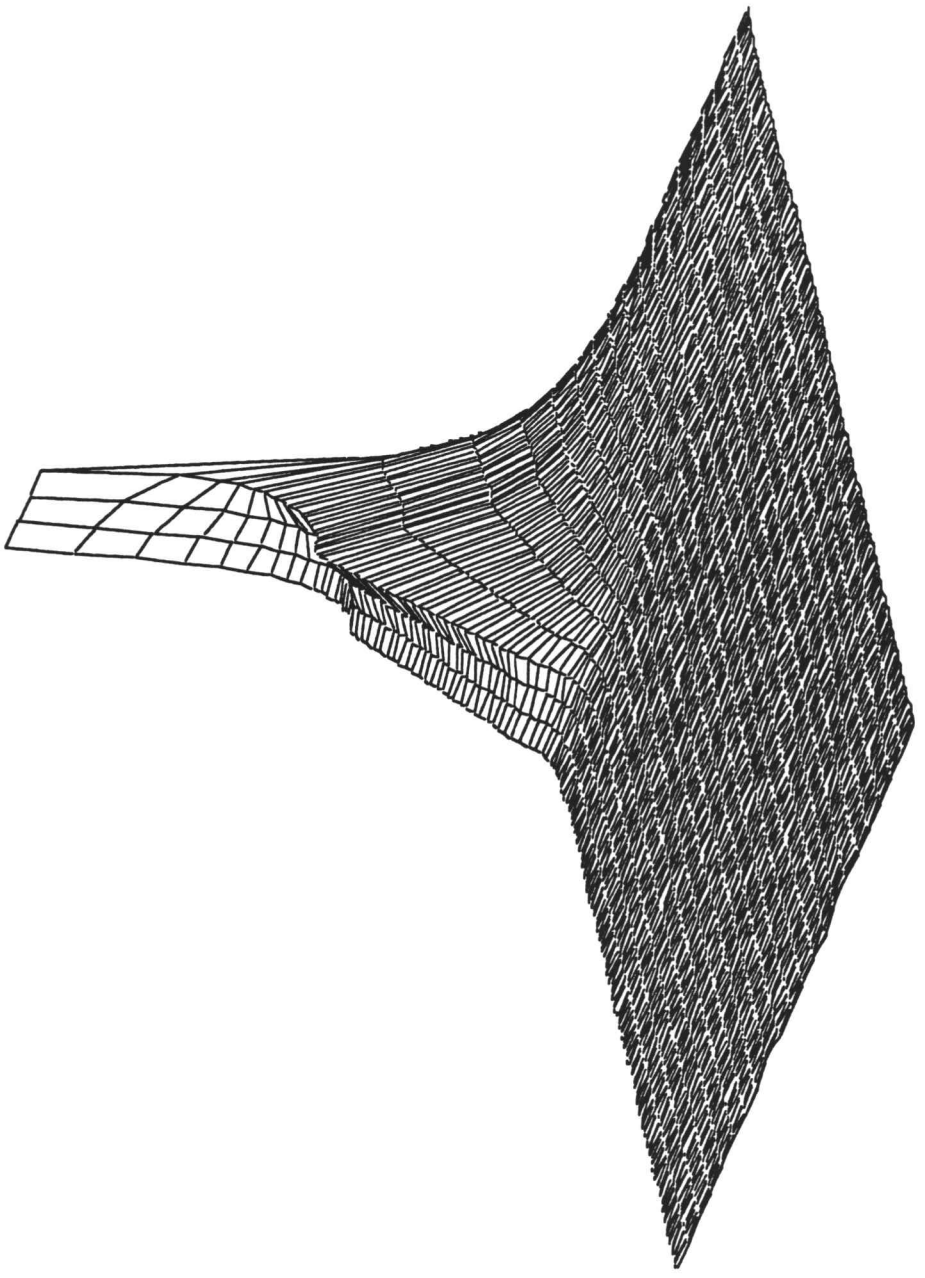


Fig. 8a Electric field under dielectric interface for a smooth interconnect (σ=1e6) at 200 time steps

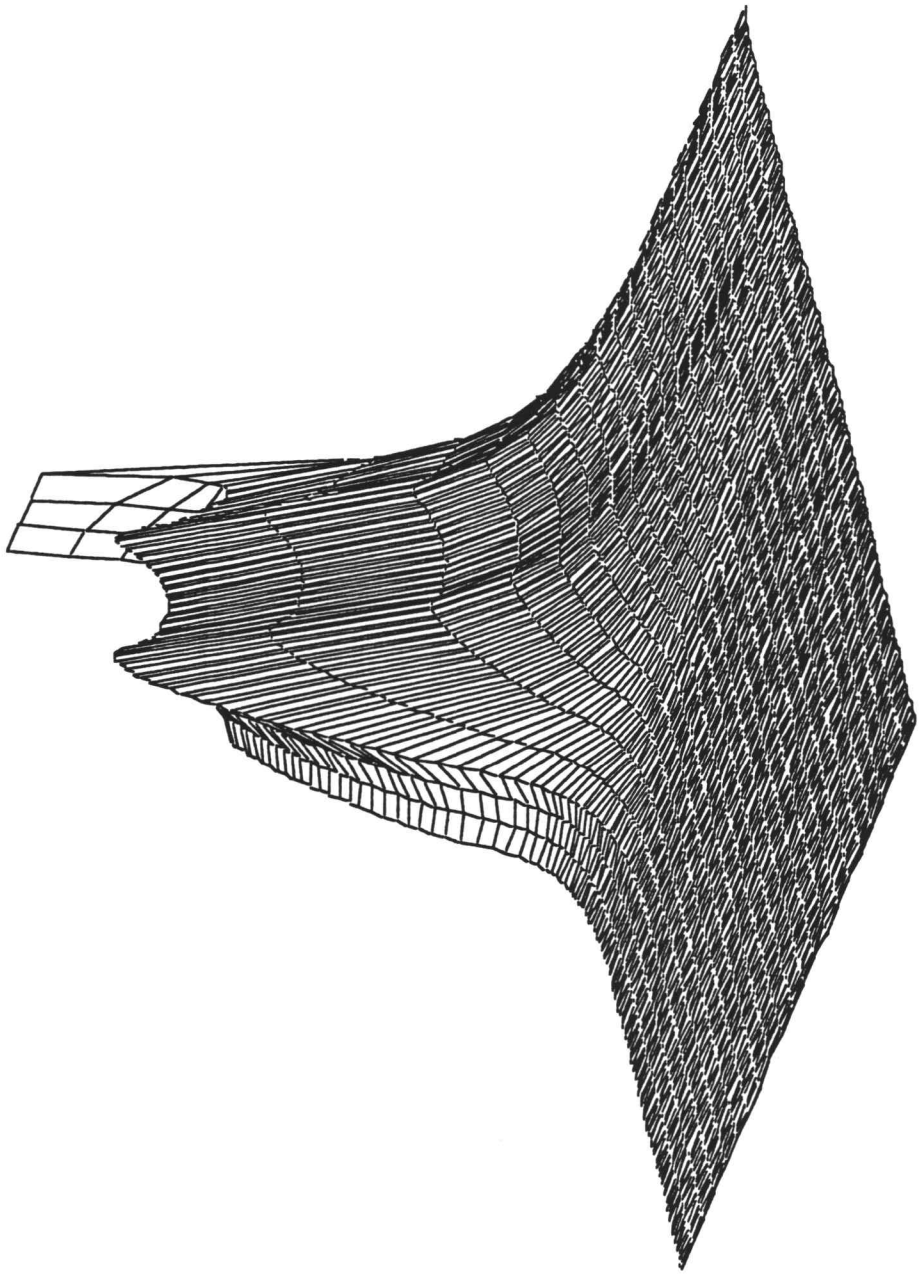


Fig. 8b Electric field under dielectric interface for a smooth interconnect (σ=1e6) at 300 time steps

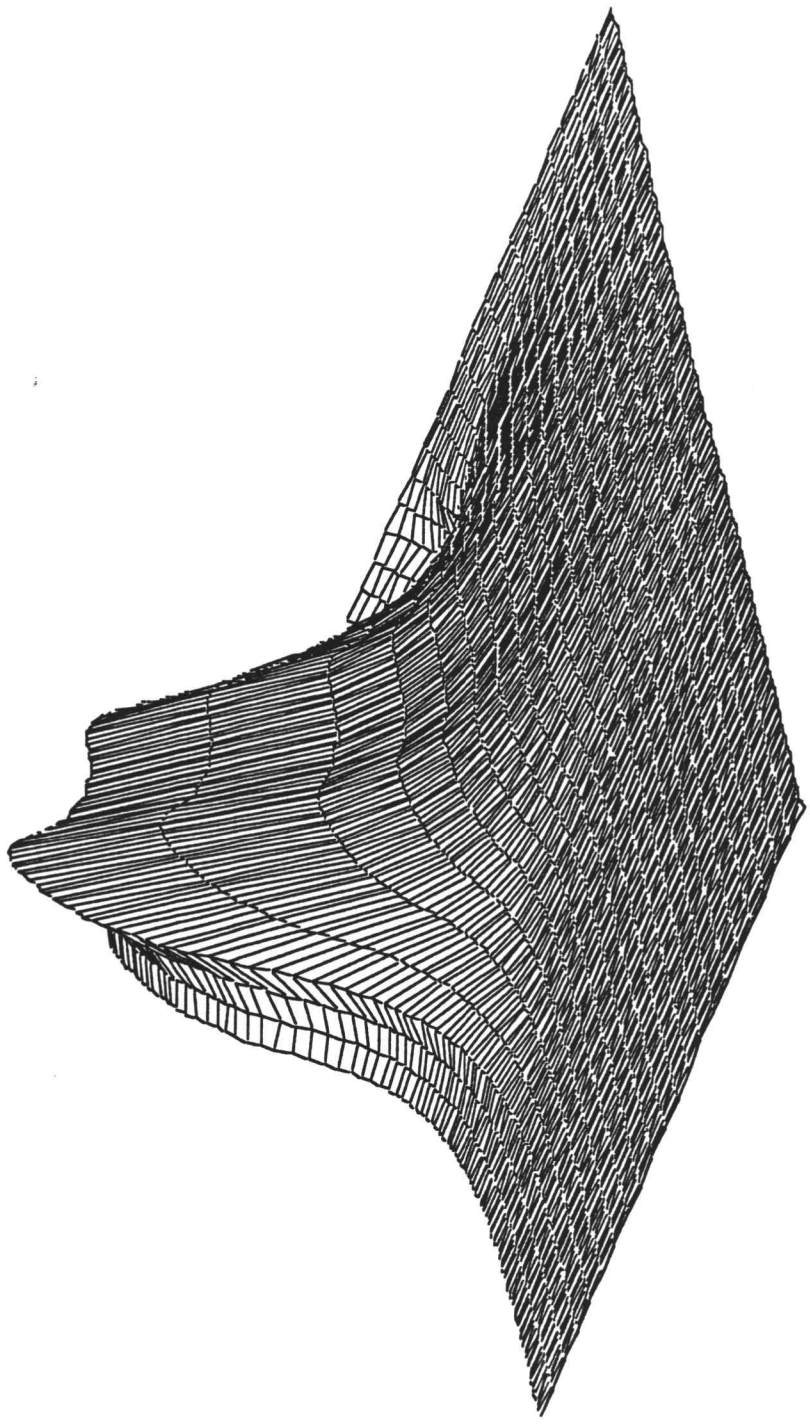


Fig. 8c Electric field under dielectric interface for a smooth interconnect (σ=1e6) at 400 time steps

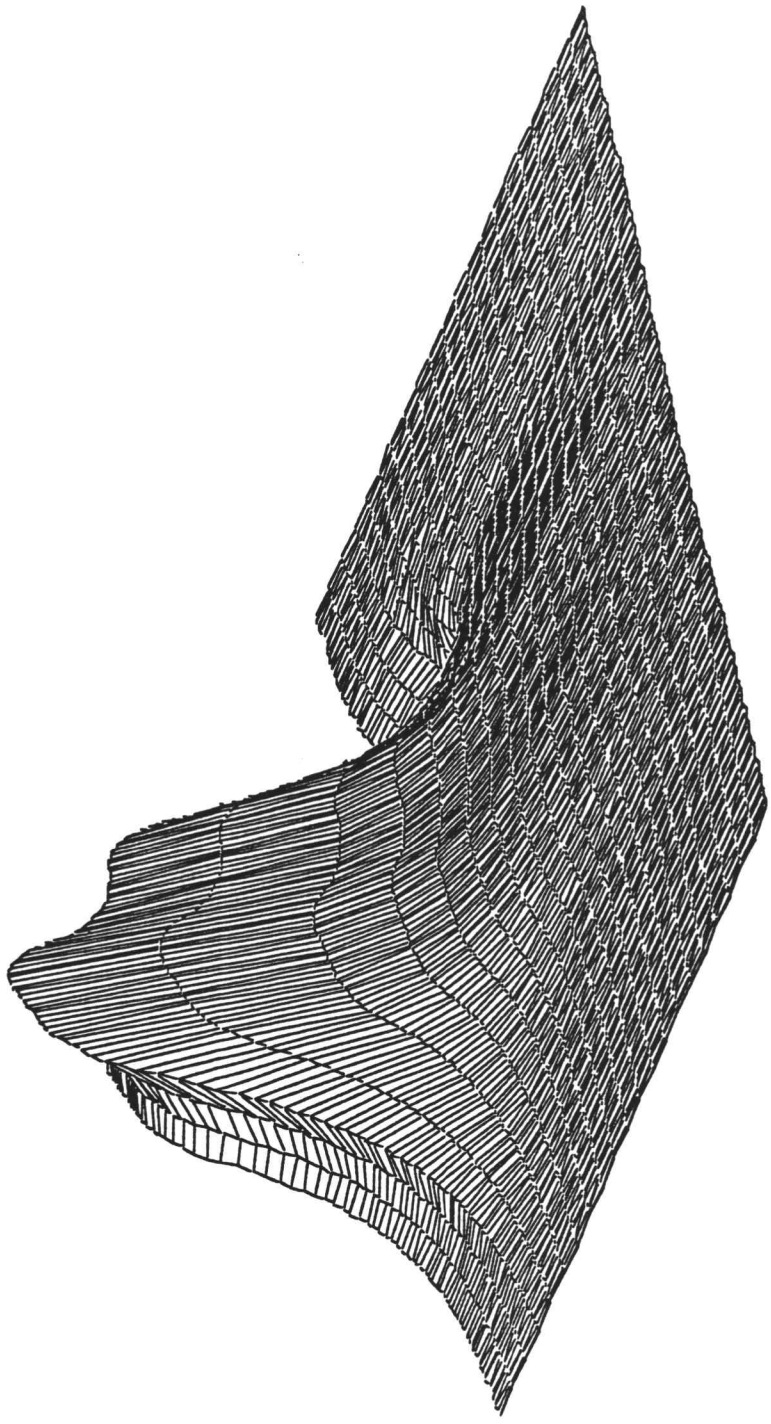


Fig. 8d Electric field under dielectric interface for a smooth interconnect (σ=1e6) at 500 time steps

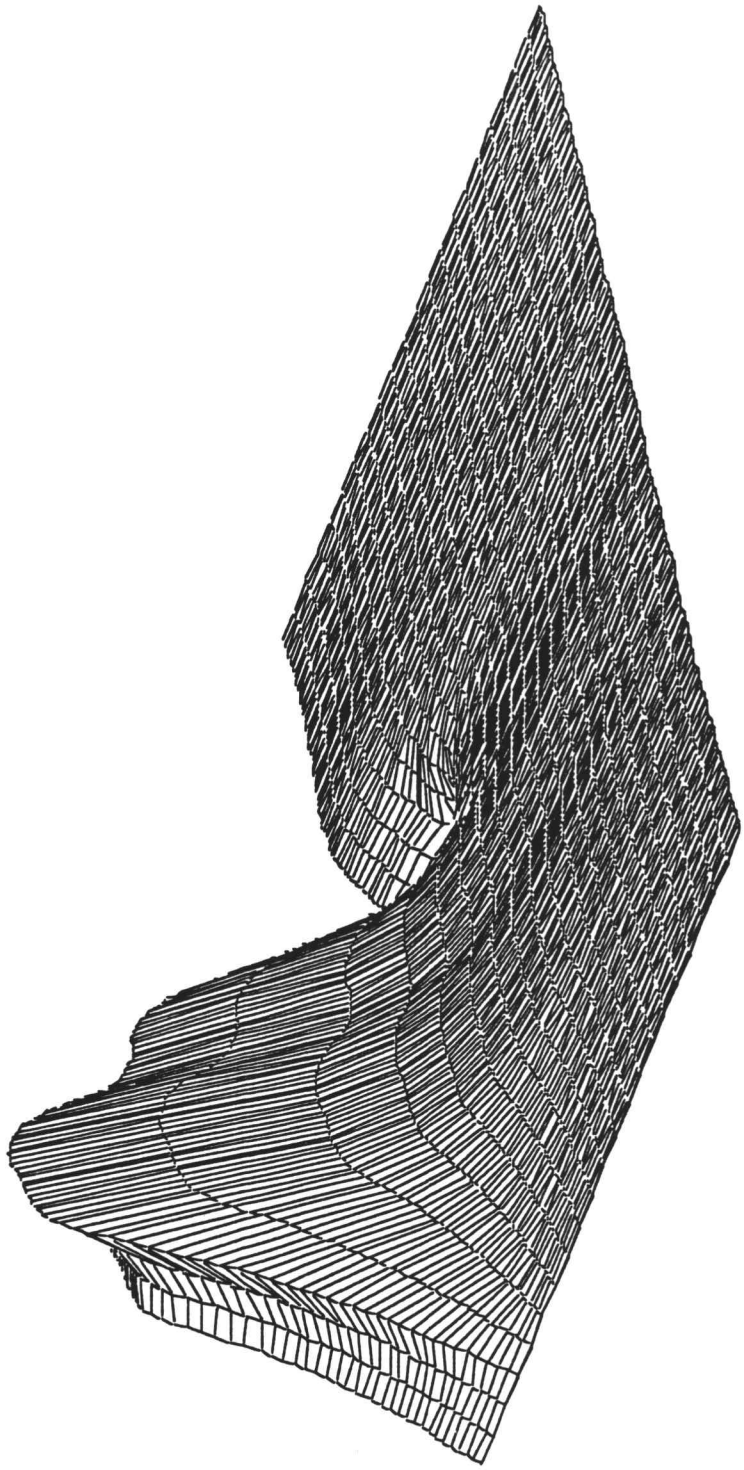


Fig. 8e Electric field under dielectric interface for a smooth interconnect (σ=1e6) at 600 time steps

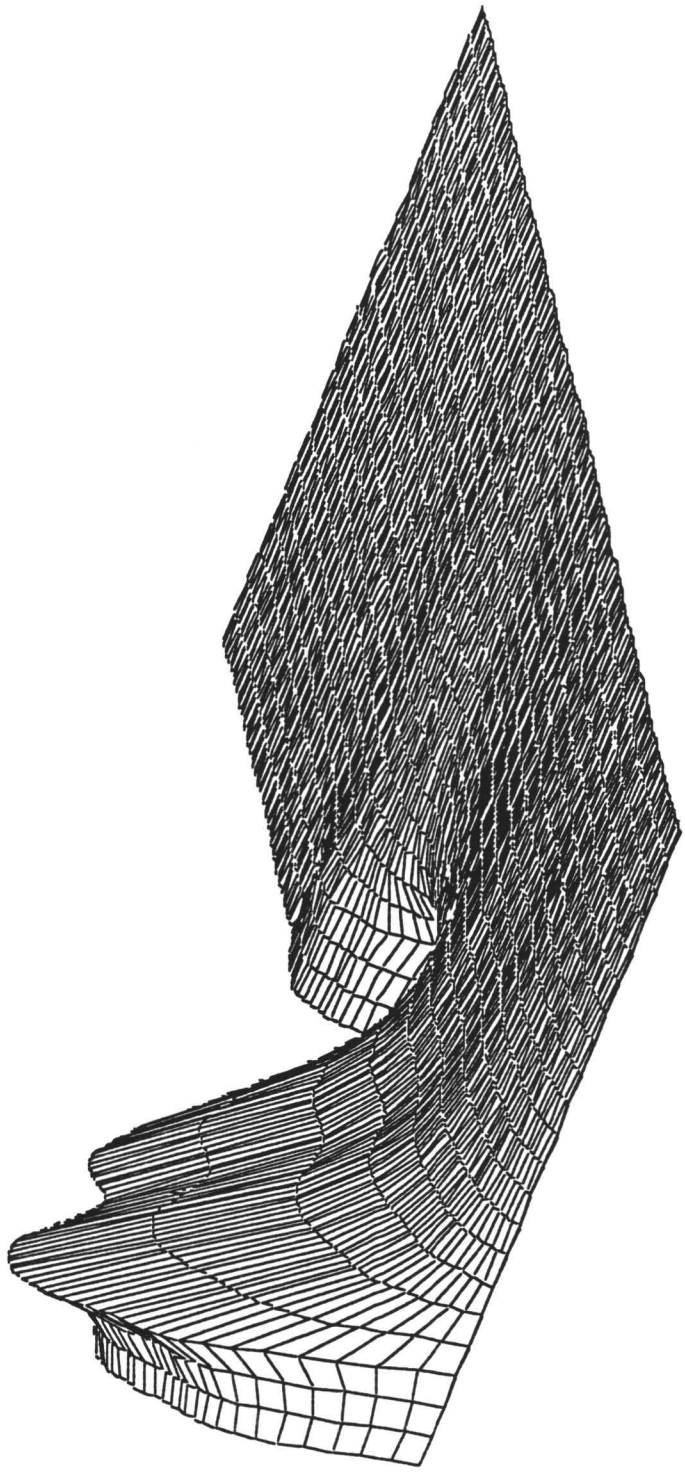


Fig. 8f Electric field under dielectric interface for a smooth interconnect (σ=1e6) at 700 time steps

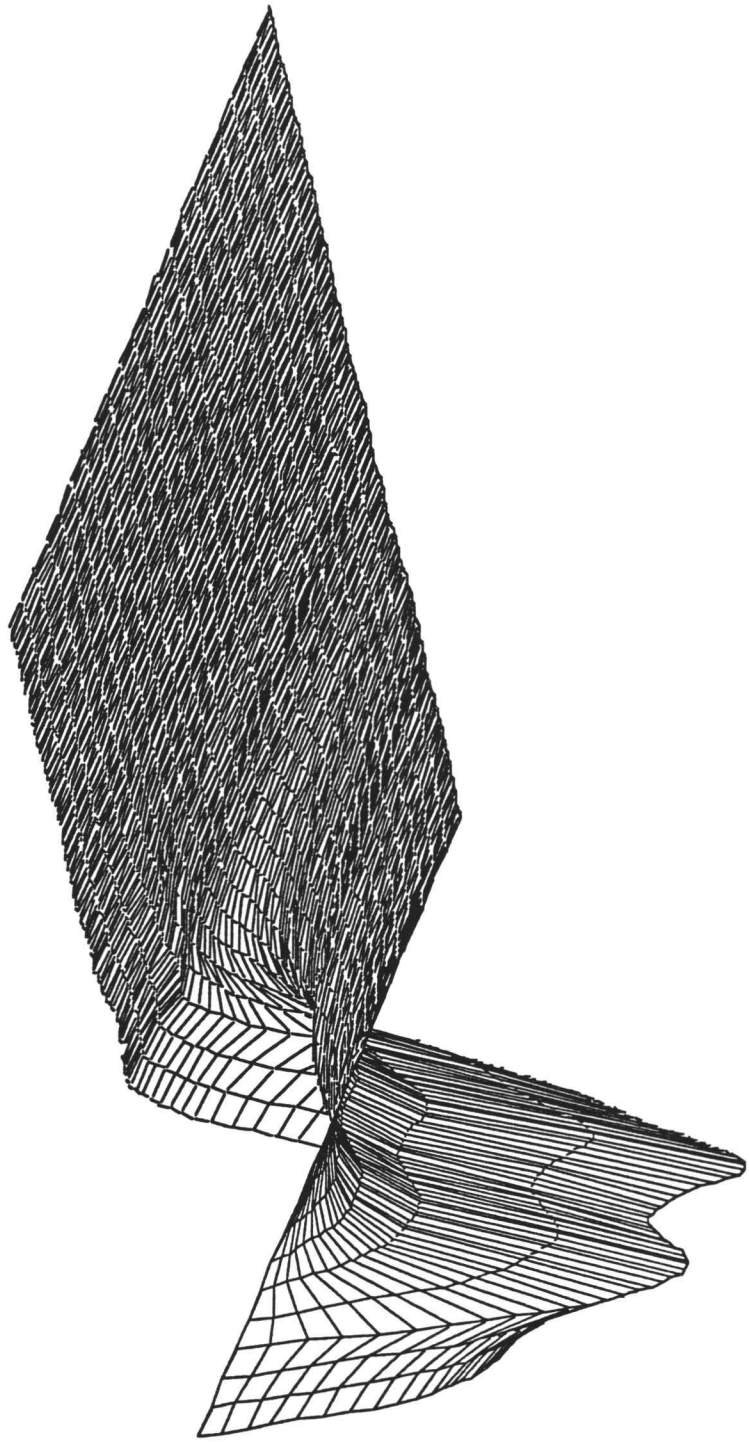


Fig. 8g Electric field under dielectric interface for a smooth interconnect (σ=1e6) at 800 time steps

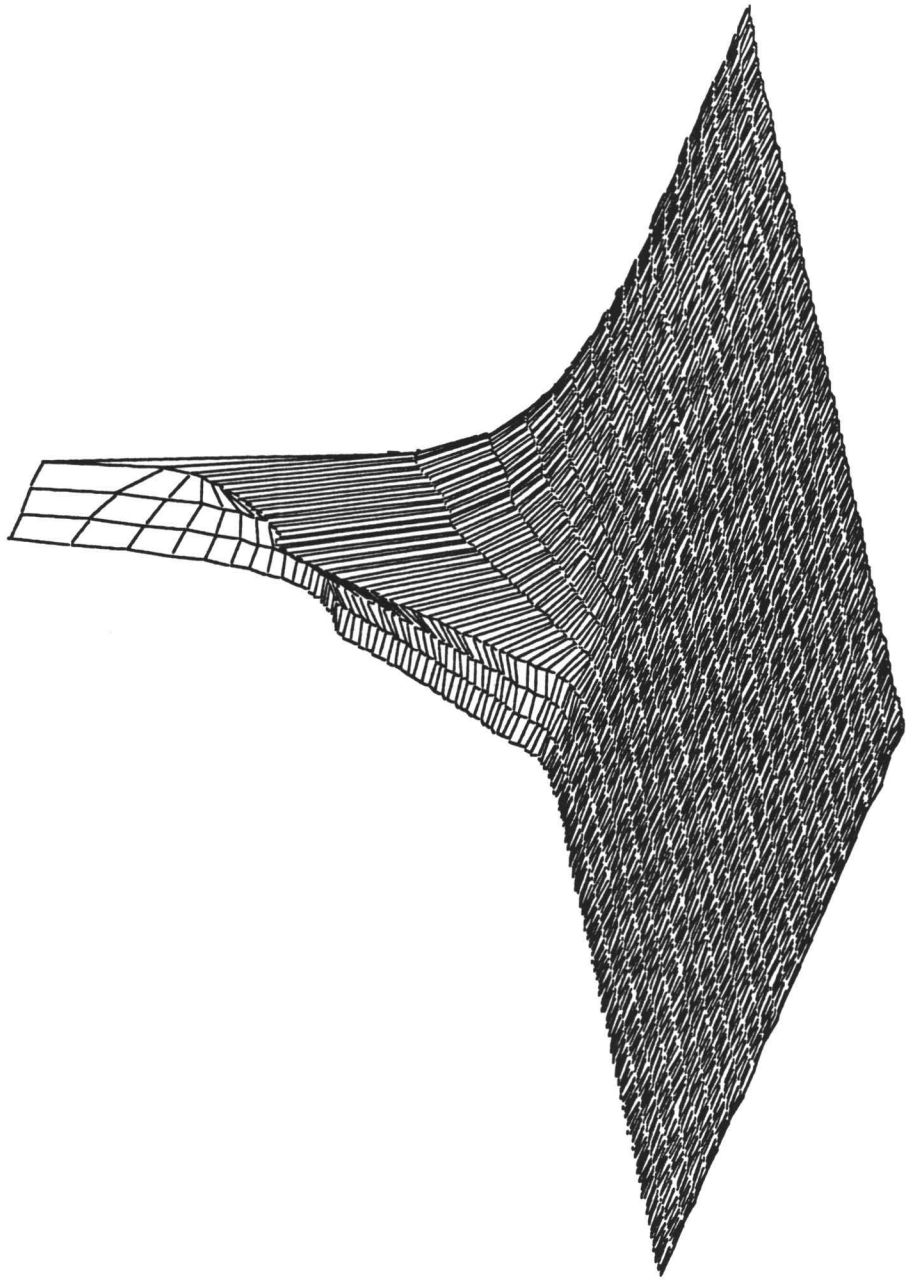


Fig. 9a Electric field under dielectric interface for a rough (rgh=.3μm) interconnect (σ=1e6) at 200 time steps

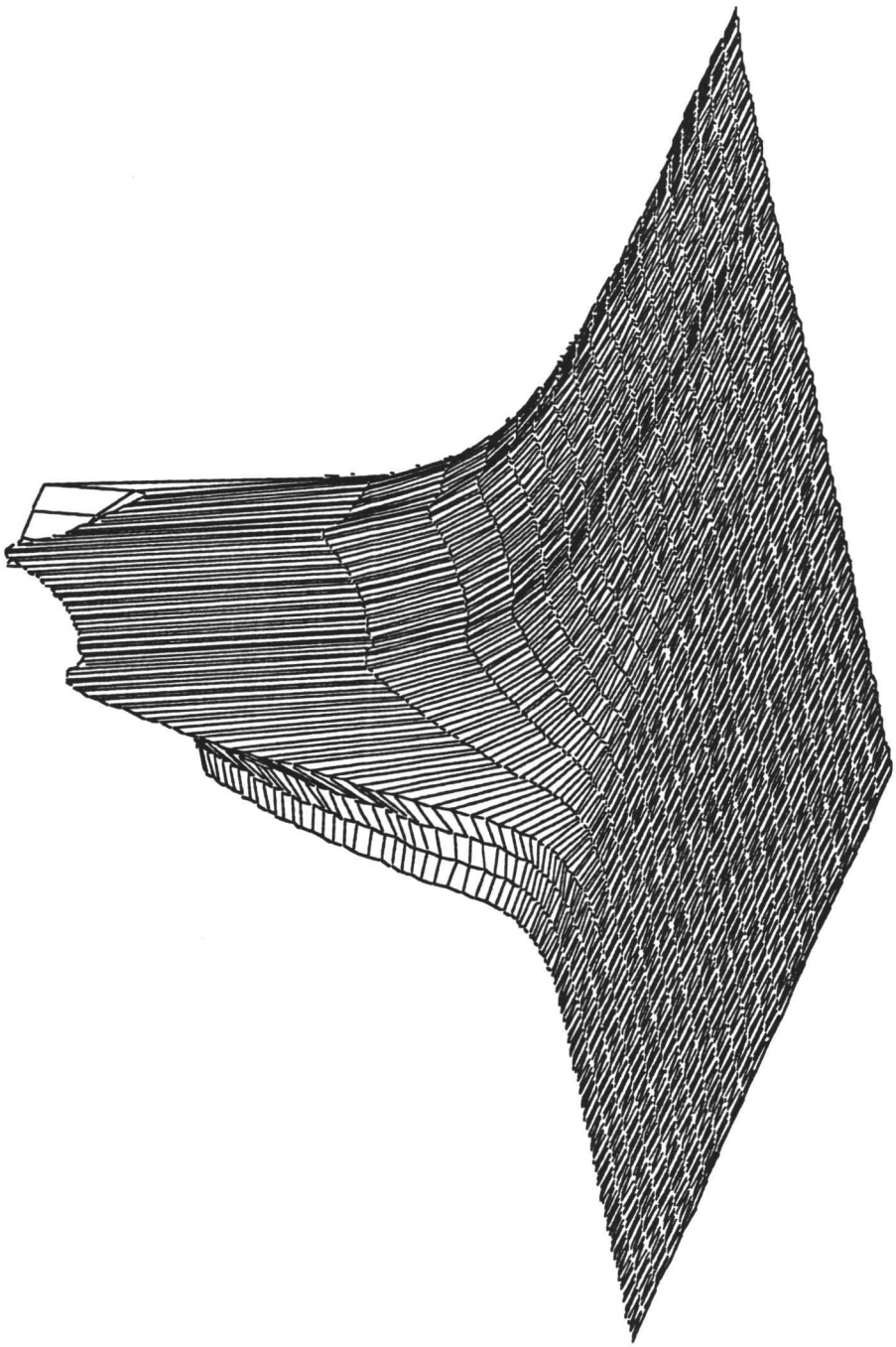


Fig. 9b Electric field under dielectric interface for a rough (rgh=.3μm) interconnect (σ=1e6) at 300 time steps

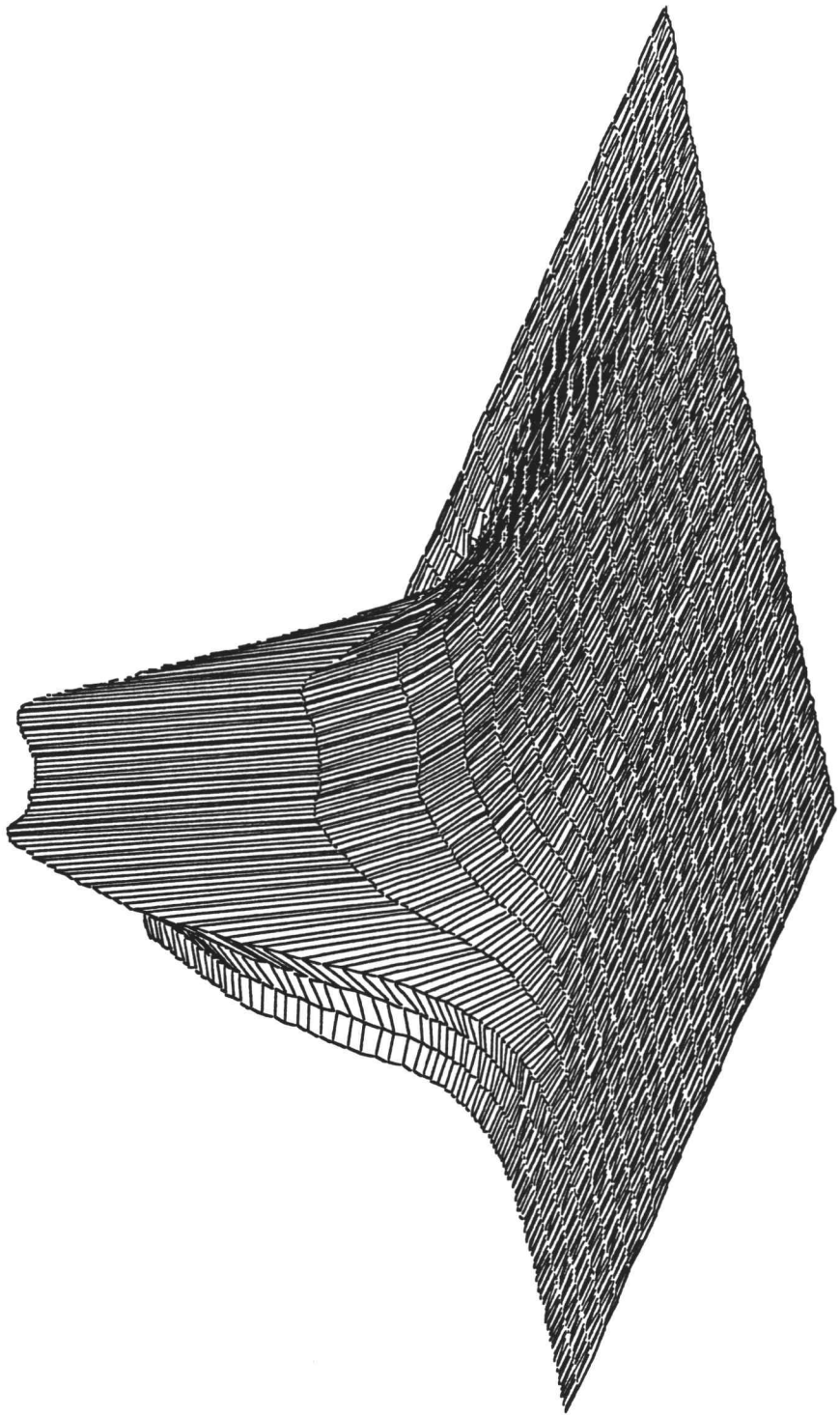


Fig. 9c Electric field under dielectric interface for a rough (rgh=.3μm) interconnect (σ=1e6) at 400 time steps

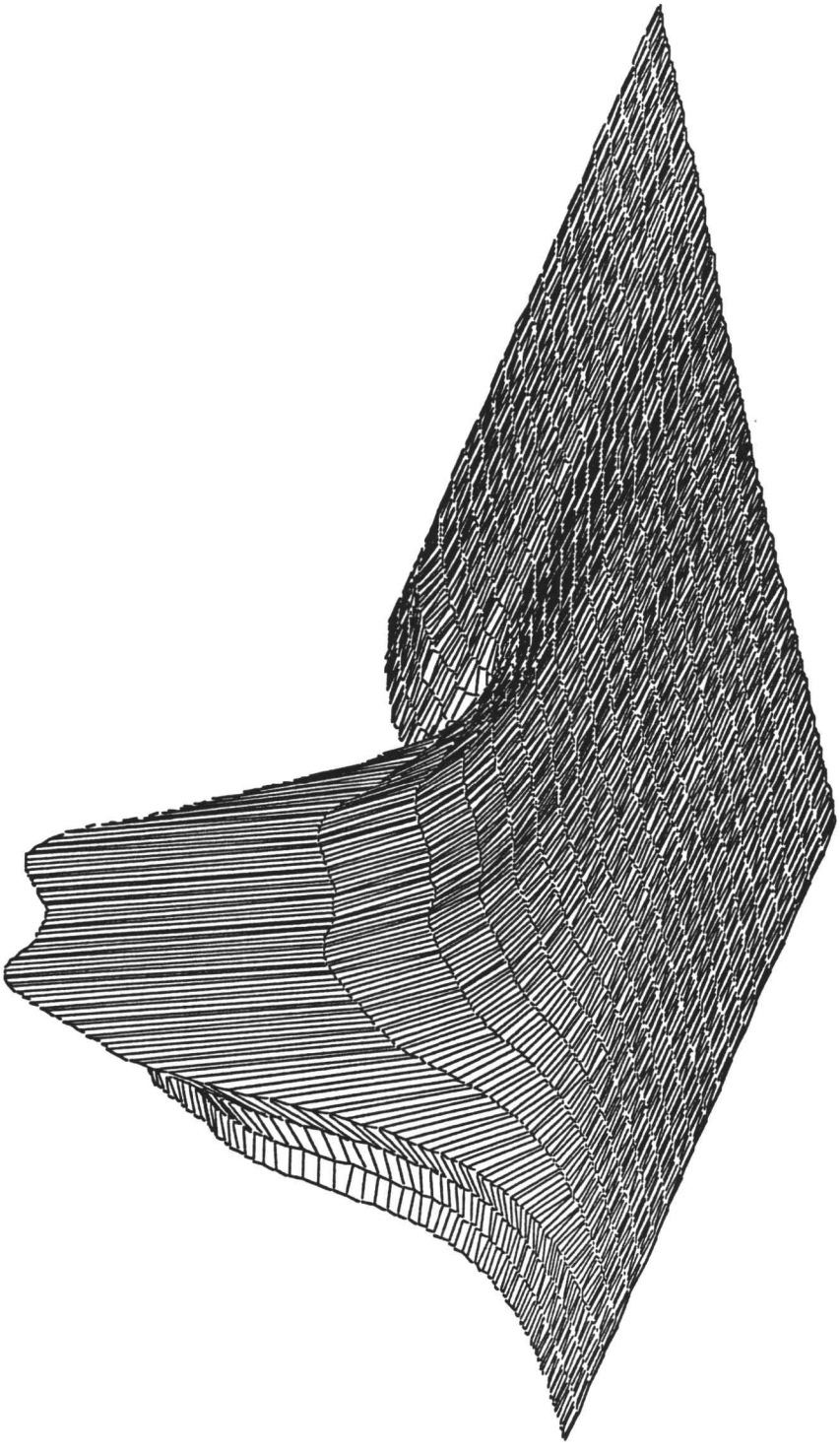


Fig. 9d Electric field under dielectric interface for a rough (rgh=.3μm) interconnect (σ=1e6) at 500 time steps

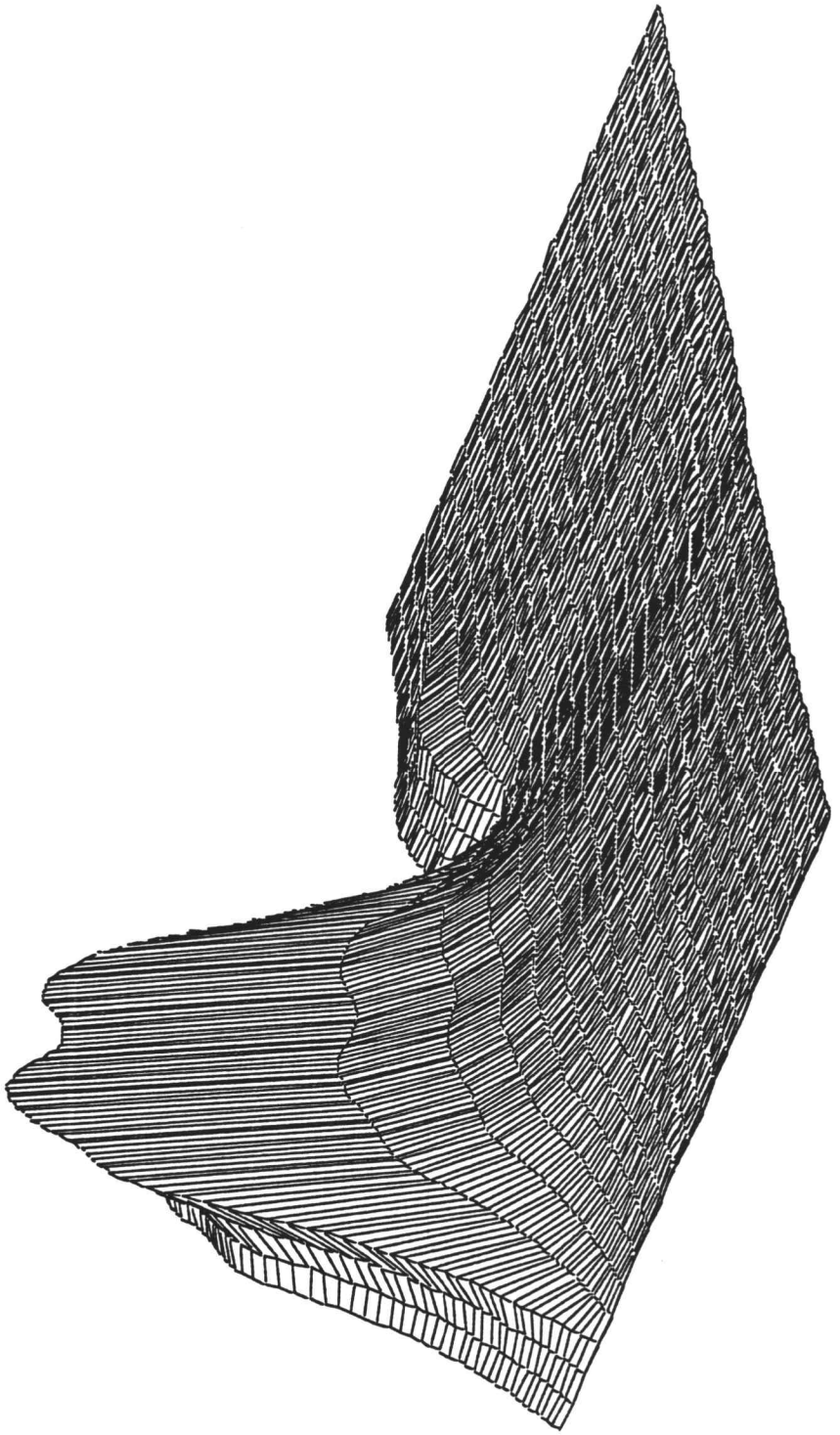


Fig. 9e Electric field under dielectric interface for a rough (rgh=.3μm) interconnect (σ=1e6) at 600 time steps

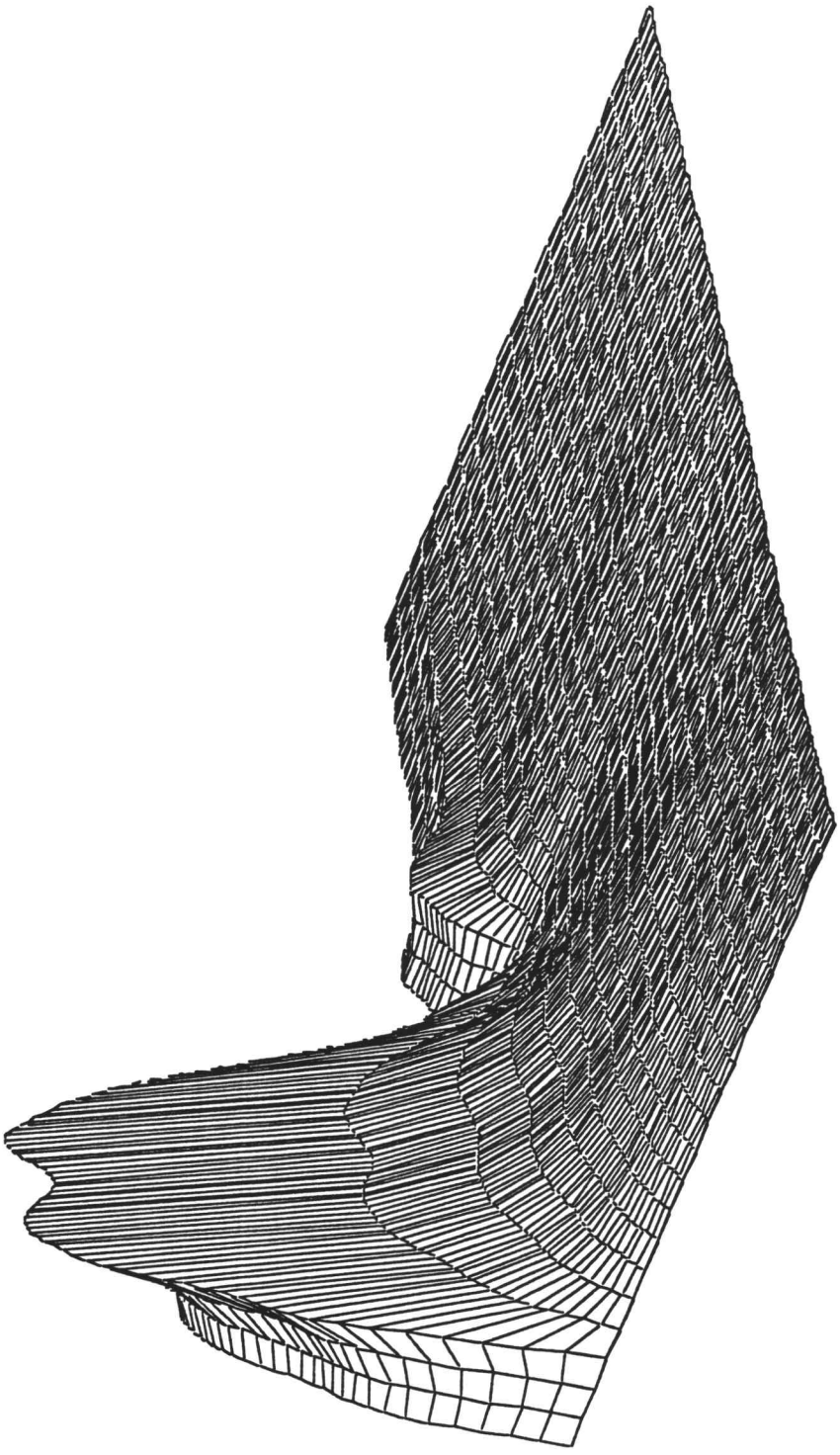


Fig. 9f Electric field under dielectric interface for a rough (rgh=.3μm) interconnect (σ=1e6) at 700 time steps

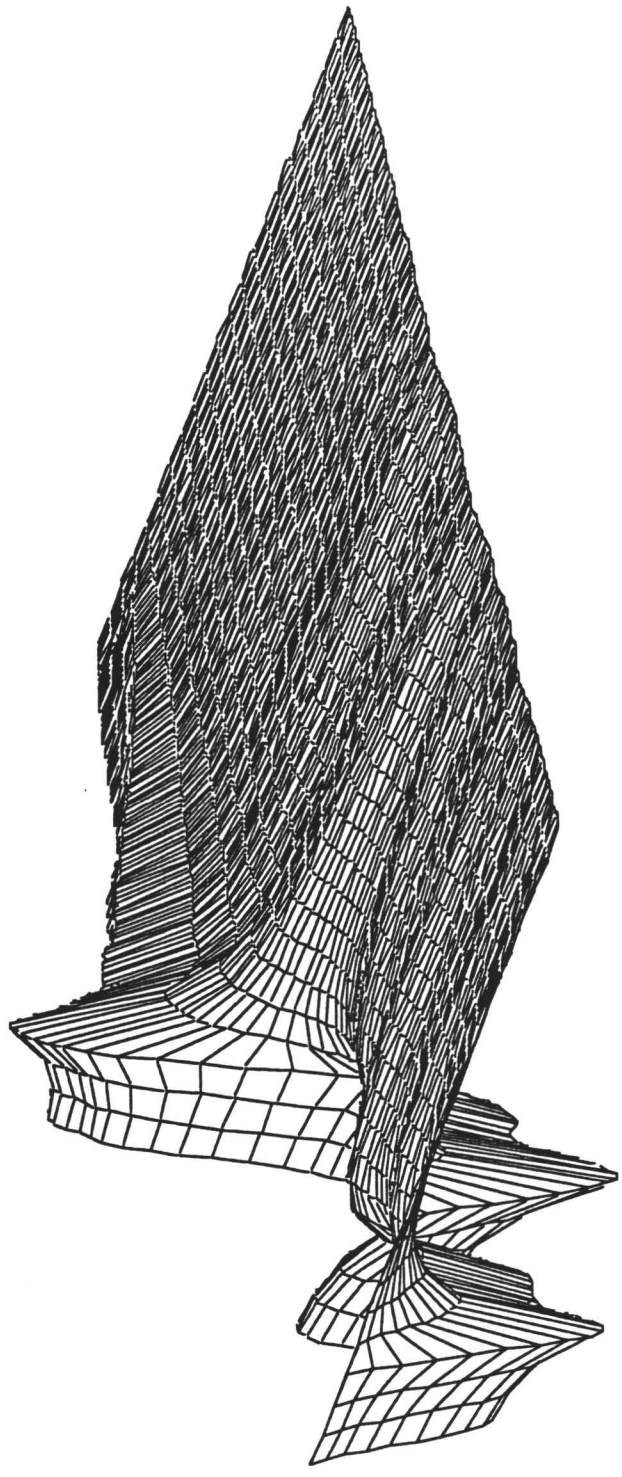


Fig. 9g Electric field under dielectric interface for a rough (rgh=.3μm) interconnect (σ=1e6) at 800 time steps

For many of the figures it is difficult to assess the attenuation of the electric field due to the presence of dispersion. Due to the non-uniform distribution of frequency components in the wave form, the amplitude of the pulse is not a good indicator of the attenuation constant. Initially an inverse, or reverse dispersion may occur allowing the pulse amplitude to potentially increase as the low frequency components of the pulse catch up with the high frequency components. In a microstrip structure, which is similar to the interconnect structure being investigated, the effective dielectric constant increases with frequency. The velocity of a frequency component is inversely proportional to the square root of the effective dielectric constant, therefore the velocity goes down with increasing frequency.

Effects of scaling

Figure 10 shows the electric field under the center of the 0.5x0.5μm interconnect and the same interconnect scaled by two and four. All three wave forms are 10μm from the input plane. The higher losses and dispersion of the smallest interconnect are easily seen in the decreased amplitude of the wave form. The early arrival of the wave form for the smallest interconnect is most likely due to the close proximity of the input plane, but further analysis need to be performed before a conclusion can be drawn.

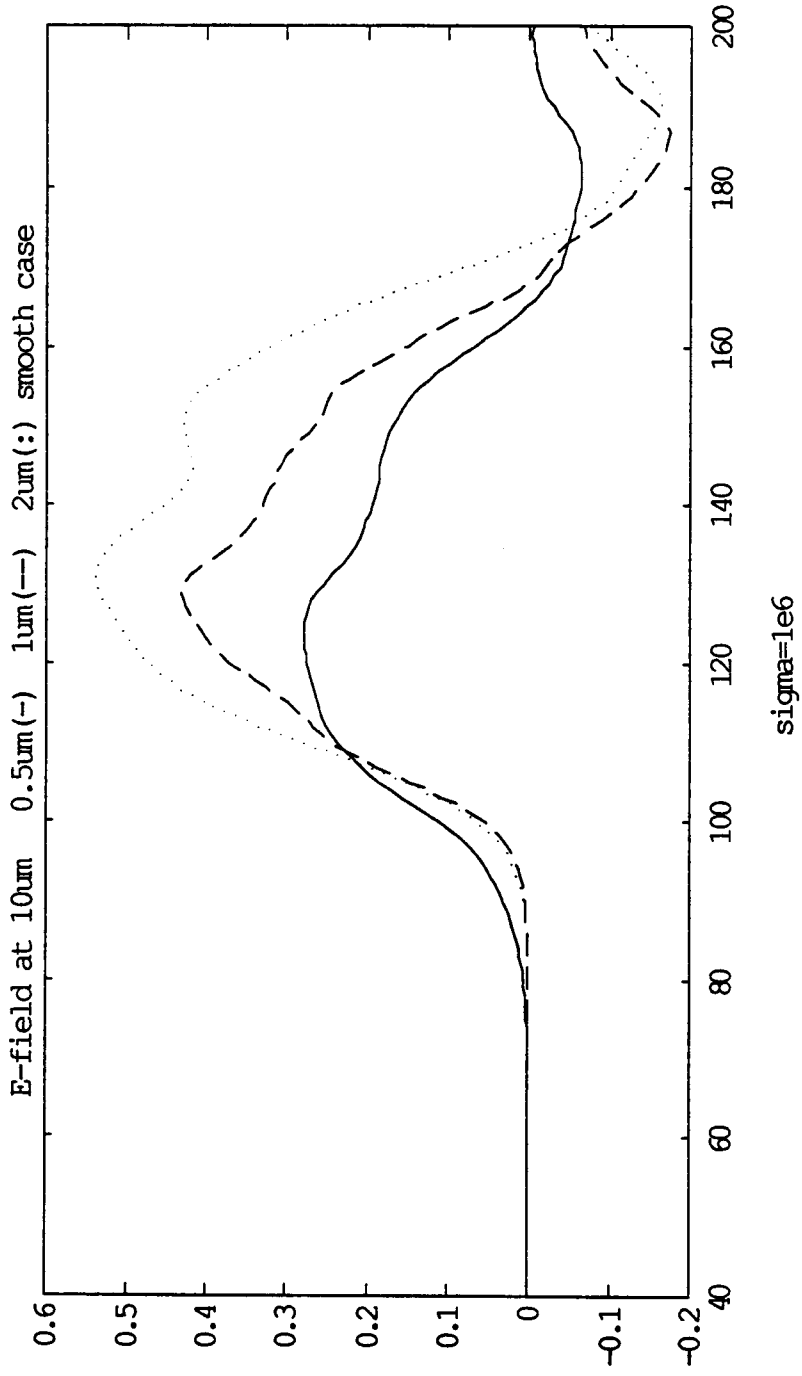


Fig. 10 Electric field under center of a smooth interconnect for various scalings.

Smooth versus rough case

Figures (11-13) show the deviation between a perfectly smooth interconnect and an interconnect with 0.3[μm] of periodic surface roughness. Each of the curves in the figure represent the vertical electric field component under the center of the interconnect strip, at a fixed position from the input plane. The vertical axis in the relative electric field strength and the horizontal axis is the number of time steps that have elapsed. The rapid drop in the amplitude of the electric field as it moves away from the input plane is due to the redistribution of the field structure into the edge dominate field pattern and due to the D.C. magnetic field transient discussed earlier. After the initial amplitude drops, the remaining amplitude losses occur due to dispersion, finite conductivity, and the presence of the surface roughness. From figures (11-13) it can be seen that after the initial amplitude drops have occurred the rough case has a lower velocity, more dispersion, and higher losses.

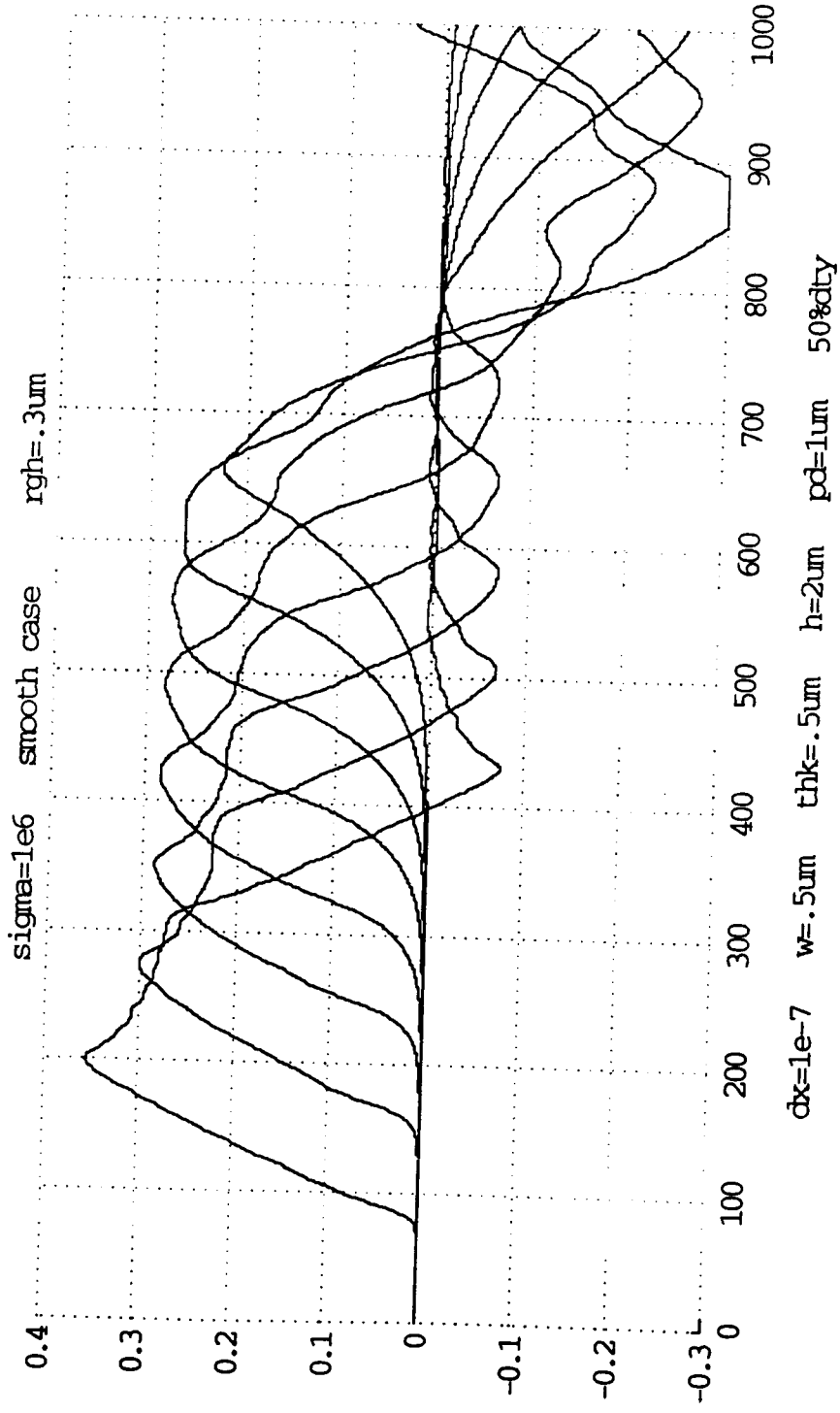


Fig. 11 Electric field under center of a smooth interconnect (σ=1e6)

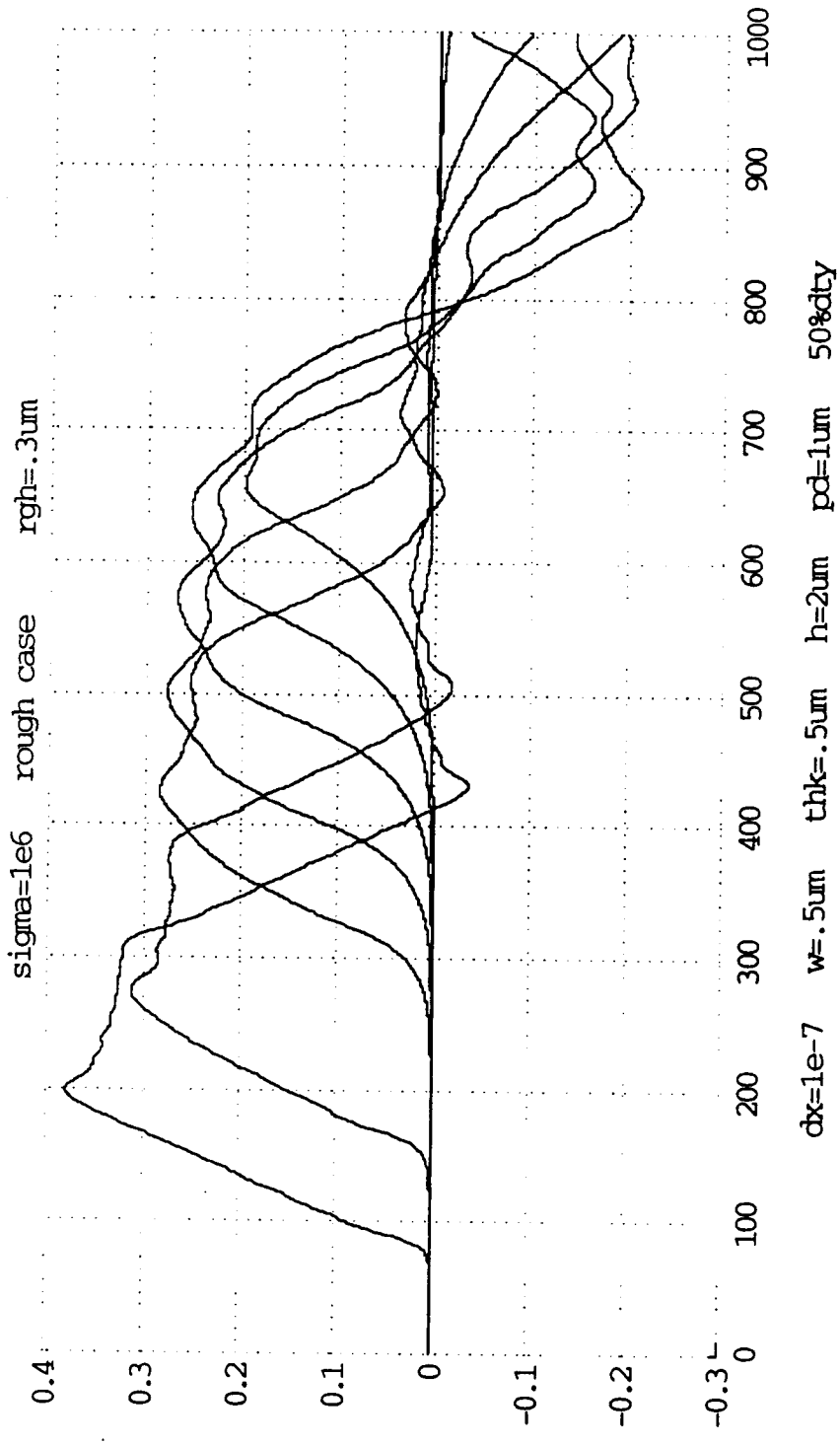


Fig. 12 Electric field under center of a rough interconnect (σ=1e6)

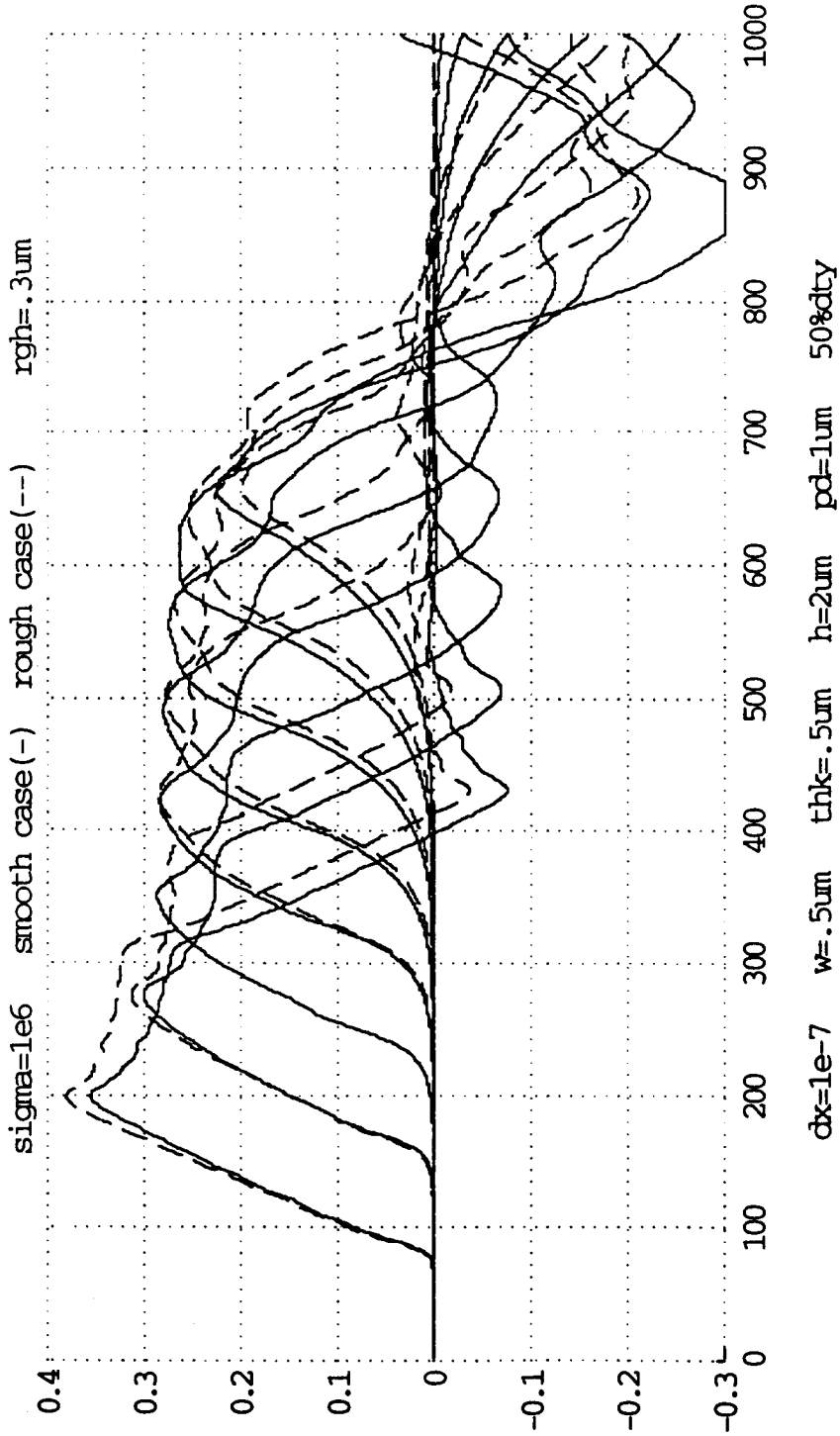


Fig. 13 Electric field under center of a smooth and rough interconnect (σ=1e6)

Effects of varying the conductivity

Figure 10 shows the effect of varying the conductivity on the vertical electric field component for the perfectly smooth interconnect structure. Figure 11 shows the same conductivity changes, but with 0.3[μm] surface roughness present on the interconnect structure. In figure 11 the 1e3 and 1e5 conductivities vary radically from the rest of the test cases. The reasons for these deviations are not clear at this time and warrant further investigation. Some possible explanations are resonance in the structure, filtering effects of the structure, or inverse dispersion occurring. Figures 12 and 13 are the electric field under the interconnect center at various distances (20μm, 40μm, 60μm, 80μm, 100μm, 120μm, 140μm, and 160μm) from the input of the two deviant conductivities.

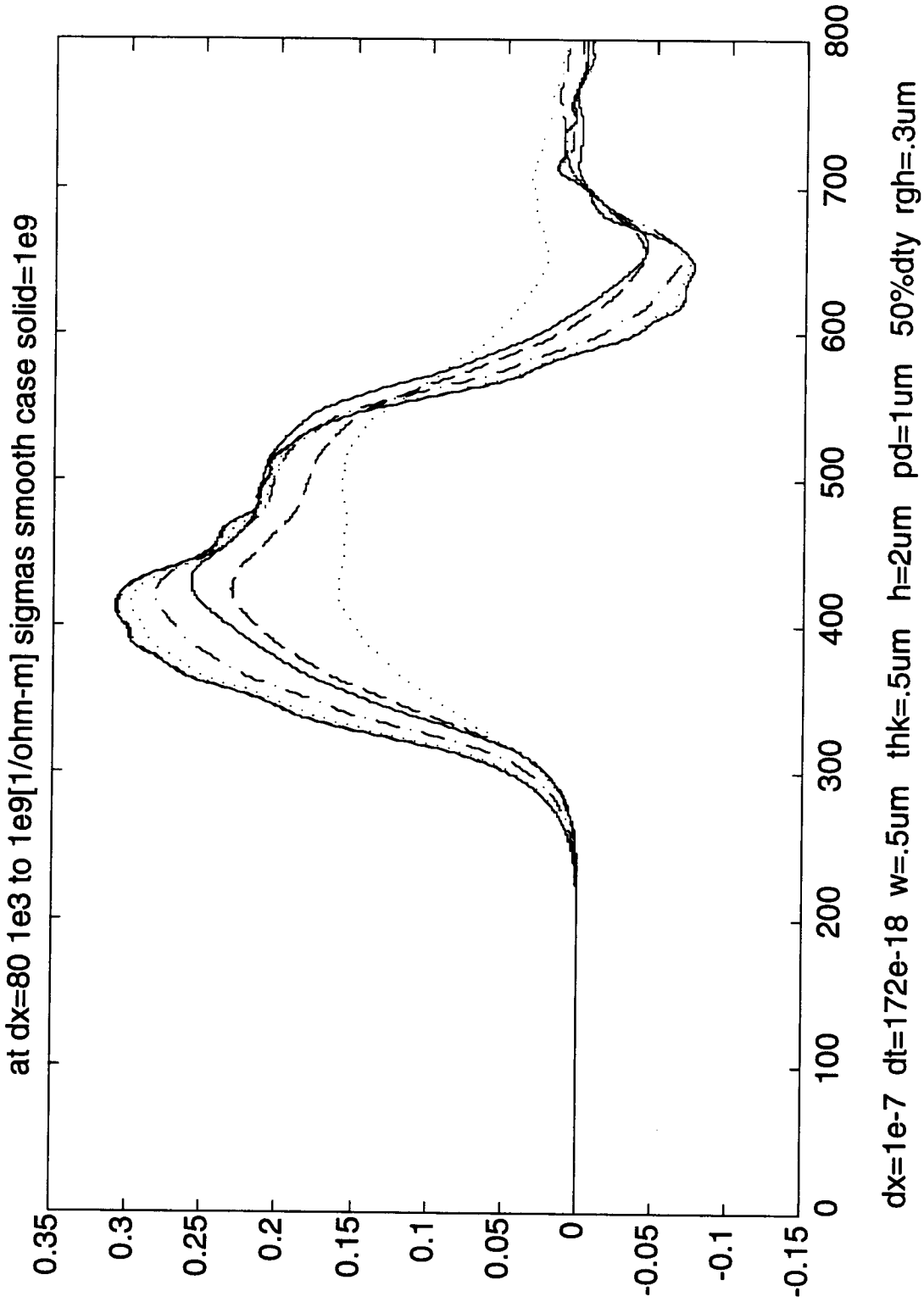


Figure 14 Effect of various conductivities on a perfectly smooth interconnect at a fixed distance (8μm) from the input plane as a function of time.

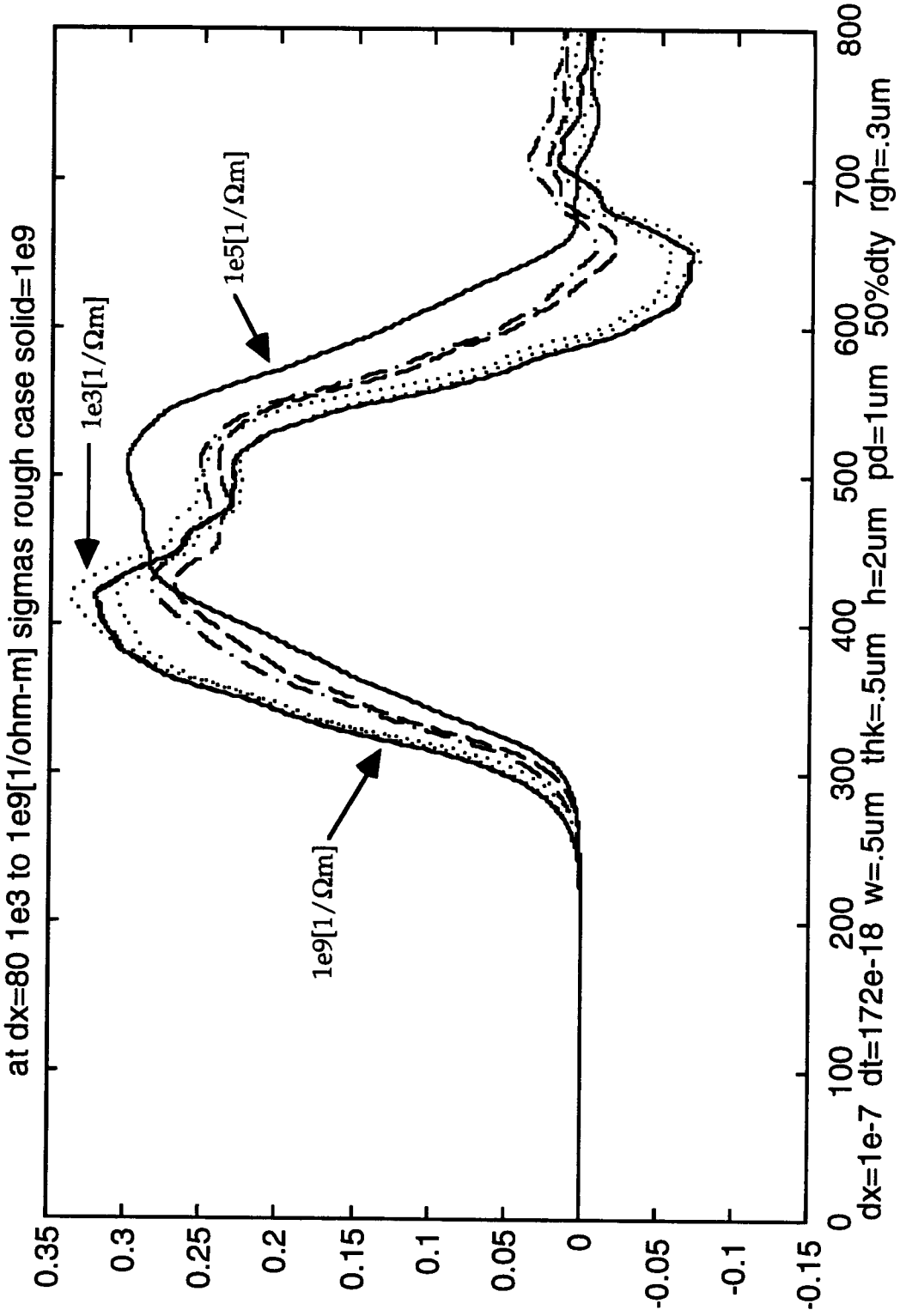


Figure 15 Effect of various conductivities on a rough (0.3μm) interconnect at a fixed distance (8μm) from the input plane as a function of time.

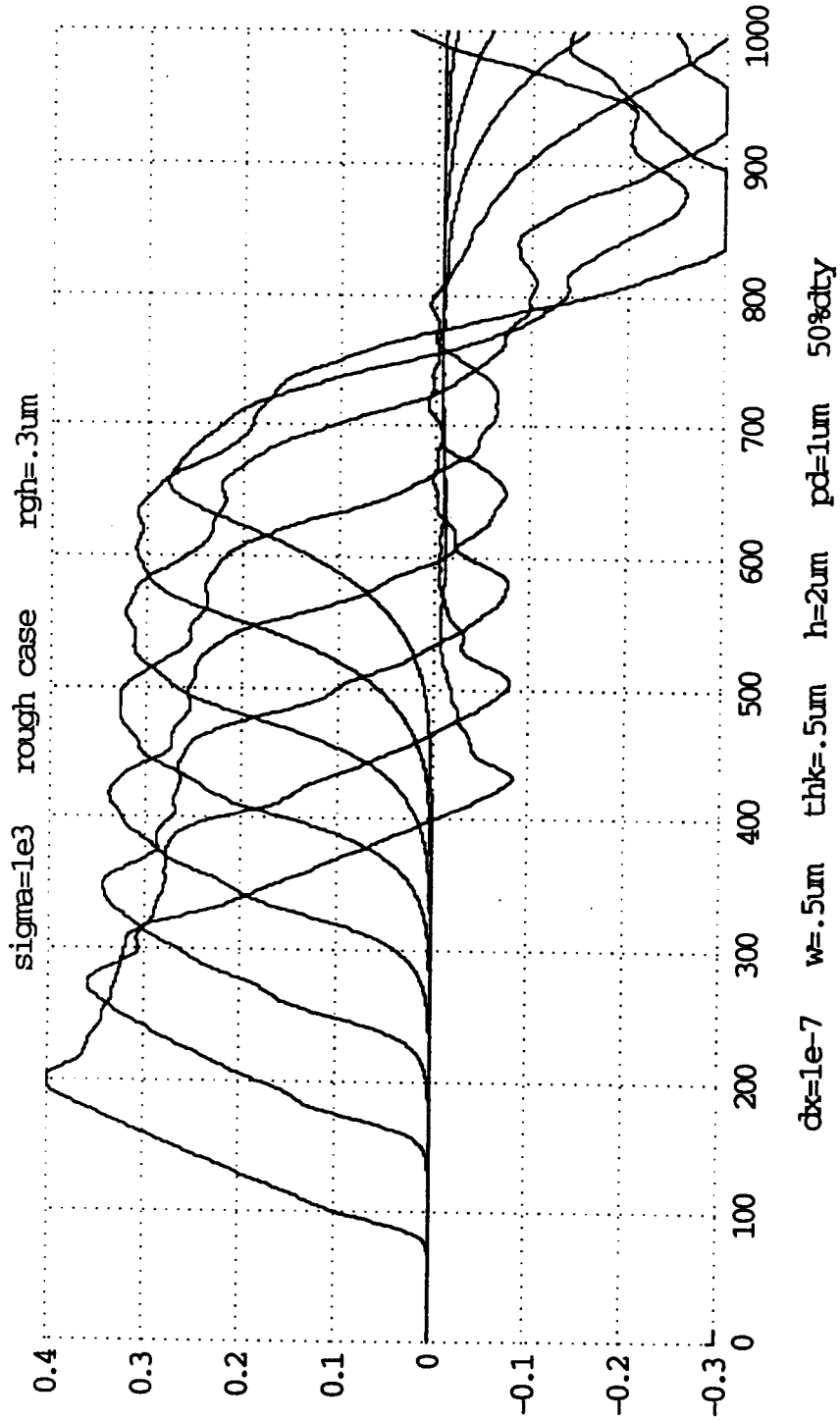


Fig. 16 Electric field under center of rough interconnect (σ=1e3)

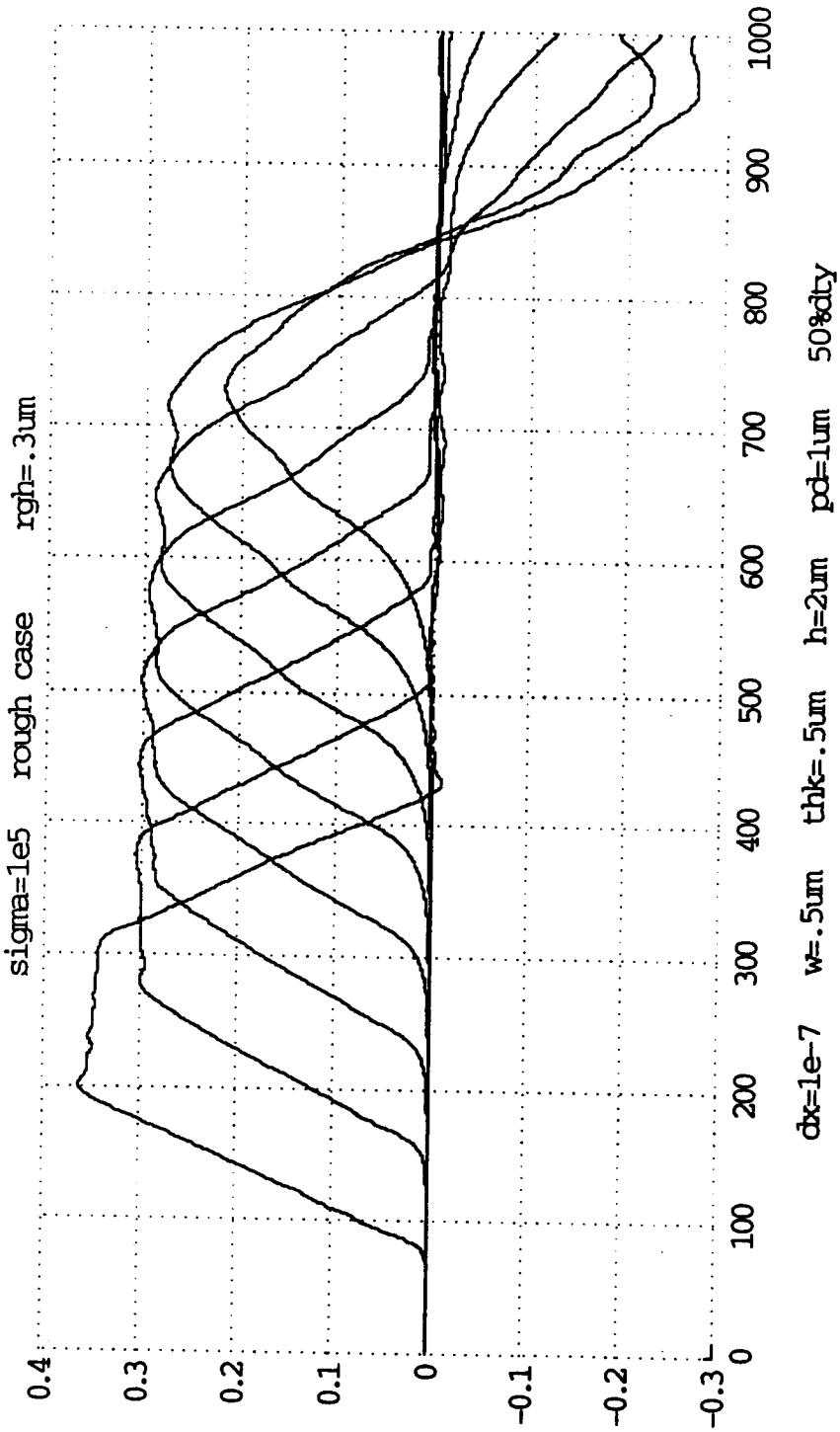


Fig. 17 Electric field under center of rough interconnect (σ=1e3)

Effects of surface roughness amplitude

Figure 18 shows the attenuation of the electric field as a function of surface roughness. As the amplitude of the surface roughness increases, the attenuation increases and the pulse delay increases. This agrees with the expected results. The effects of the loss due to roughness would be more pronounced if the pulse was allowed to propagate further. However computer memory is limited and thus the simulation space was small.

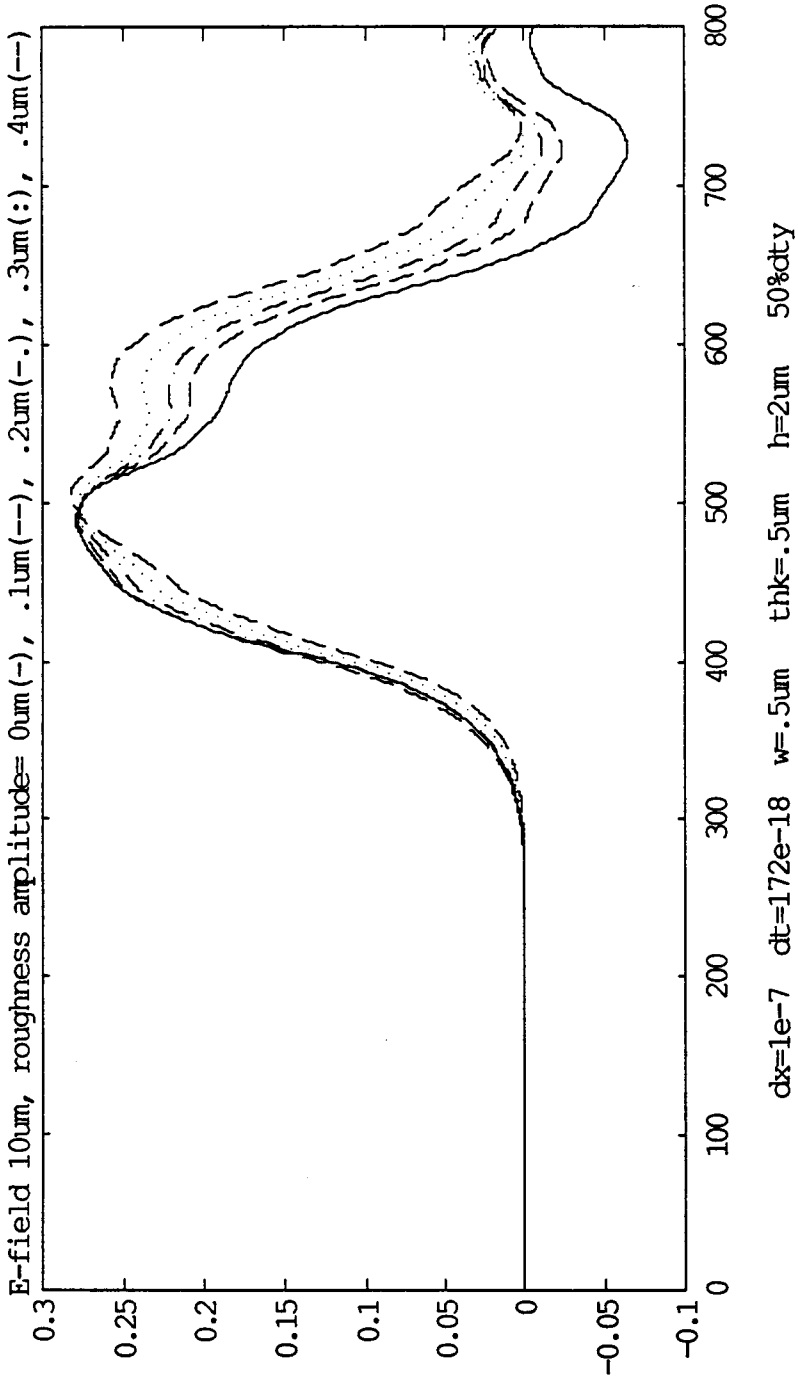


Figure 18 Electric field at a fixed distance (10μm) from the input plane as a function of time for various surface roughnesses.

Effects of surface roughness period

Figure 19 shows the effect of varying the period of the surface roughness for a set conductivity (1e6[1/Ω-m]) and a set surface roughness (0.3μm). From Figure 19 it is seen that the period of the roughness increases on the attenuation of the electric field, but not as significantly as the effect of the roughness amplitude shown earlier. Figure 19 shows that the smaller the period of surface roughness, the slower the propagation velocity becomes and the more dispersion that occurs.

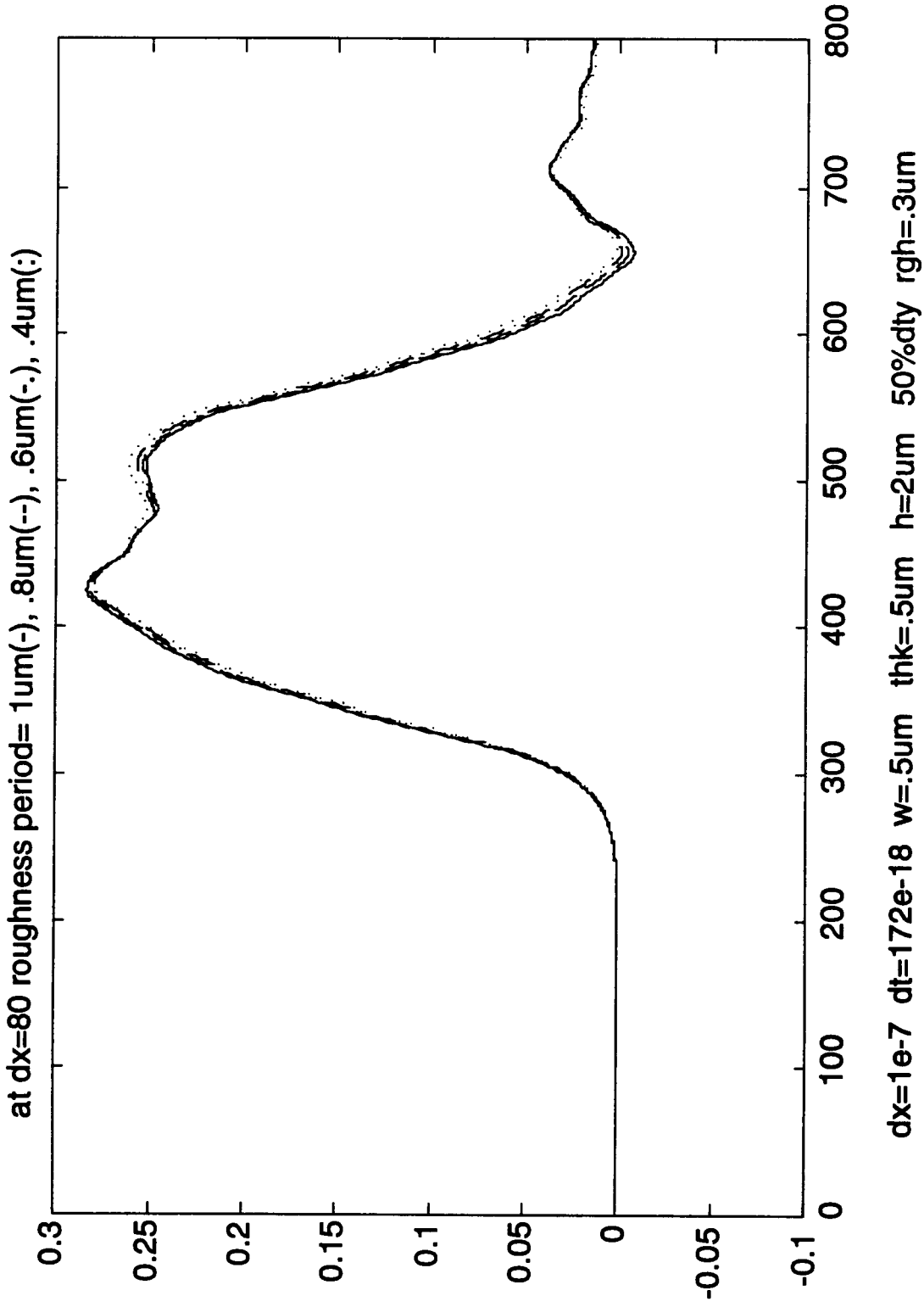


Fig. 19 Electric field at a fixed distance (8μm) from the input plane as a function of time for various periods of surface roughnesses.

CONCLUSIONS

The finite-difference time-domain simulations of the effects of surface roughness on the properties such as delay and distortion of typical time domain voltage waveforms in VLSI interconnects has been presented. The effects of finite conductivity, amplitude of the surface roughness, and period of the surface roughness on the propagation of an electromagnetic pulse down a VLSI interconnect were determined. Both the delay and distortion associated with the interconnect were strong functions of the conductivity and the amplitude of the surface roughness. The effect of varying the period of the surface roughness on the delay and distortion was not as strong, but still present.

This work presented the first step in investigating the effects of VLSI interconnect surface roughness, finite conductivities, and finite thicknesses and demonstrated that the FD-TD method is well suited to this analysis. The FD-TD simulator written for this investigation is ultimately limited by the available computer memory and the imagination of the programmer.

FUTURE WORK

The results from the FD-TD simulations have provided several useful insights into the effects of surface roughness and finite conductivity on the propagation characteristics of VLSI interconnects. The potential of the FD-TD technique has not been exhausted yet, and leaves plenty of room for future work on this problem. To extract quantitative parameters from the FD-TD simulations requires that the time domain results be transformed into the frequency domain. Several barriers must be over come before this can be done. Super-absorbing boundary conditions must be incorporated into the simulation to eliminate the boundary reflections. The presence of even small reflections in the simulation time domain output causes large and ill-defined errors when converted to the frequency domain. Additionally a simple Fourier transform of the time domain yields very little usable frequency domain data due to the extremely short sample period and sampling step size. The short sample period in the time domain converts into a large step between frequency divisions in the frequency domain. Recently Bi presented a method of using the Discrete Fourier Transform, desampling, decimation filters, and the multiple signal classification method to achieve "super-resolution" in the frequency domain which may eliminate this problem [21]. By applying super-absorbing boundary conditions and "super-resolution" techniques it should be possible to extract quantitative parameters from the FD-TD simulations including effective dielectric constant, propagation constant, attenuation constant, and line impedance as a function of frequency.

Future work may also include analyzing non-periodic roughness and interconnect structures which are not uniform. Time or field varying material properties can also be incorporated into the simulation to model non-linear systems possibly including active components. The flexibility of the FD-TD method leaves many possibilities for future work.

BIBLIOGRAPHY

[1] A.E. Sanderson, "Effect of Surface Roughness on Propagation of the TEM Mode", Advances in Microwaves, Vol.7, Academic Press, New York, 1971

[2] R.H. Dennard, "Design of Ion-Implated MOSFET's with Very Small Physical Dimensions", IEEE J. Solid-State Circuits, vol. 9, p.256-267, Oct. , 1974.

[3] K.C. Saraswat, "Effects of scaling of Interconnections on the Time Delay of VLSI Circuits", IEEE Trans. on Elec. Dev., vol. ED-29, no. 4, Apr., 1982.

[4]D.M. Pozar, Microwave Engineering,Reading, Massachusetts, Addison-Wesley Pub. Co.,1990.

[5] K. S. Yee, "Numerical solution of initial boundary value problems involving Maxwell's equations in isotropic media", IEEE Trans. Ants. Prop., vol. AP-14, pp. 302-307, May 1966.

[6] M. De Pourcq, "Field and power-density calculations in closed microwave systems by three-dimensional finite differences", IEE Proceedings. Part H, Ants. Prop., vol. 132, pp.360-368, Oct. 1985.

[7] X. Zhang, "Calculations of the dispersive characteristics of microstrips by the time-domain finite difference method", IEEE Trans. MTT, vol. 36, N0. 2, pp.263-267, Feb. 1988.

[8] J. Fang, "A super-absorbing boundary algorithm for solving electromagnetic problems by the time-domain finite-difference method", Joint AP-S, URSI-B Session 27, pp. 472-474,

[9] Gerrit Mur, "Absorbing Boundry Conditions for the Finite-Difference Approximation of the Time-Domain Elecromagnetic-Field Equations", IEEE Trans. Electromagnetic Compatibility, vol. EMC-23, No. 4, pp.377-382, Nov. 1981.

[10] J.G. Maloney, G.S. Smith, "The Efficient Modeling of Thin Material Sheets in the Finite-Difference Time-Domain (FDTD Method", IEEE Trans. Ants. Prop., vol. 40, No. 3, pp 323-330, March 1992.

[11] P.A.Tirkas, K.R.Demarest, "Modeling of Thin Dielectric Structures Using the Finite-Difference Time-Domain Technique",IEEE Trans. Ants. Prop., vol. 39, No. 9, pp 1338-1344, Sept. 1991.

[12] X. Zhang, "Time-Domain Finite Difference Approach to the Calculation of the Frequency-Dependent Characteristics of Microstrip Discontinuities", IEEE Trans. MTT, vol. 36, N0. 2, pp.1775-1787, Dec. 1988.

[13] G, Liang,"Full-Wave Analysis of Coplanar Waveguide and Slotline Using the Time-Domain Finite-Difference Method",IEEE Trans. MTT, vol. 37, N0. 12, pp.1949-1957, Dec. 1989

[14] D, Sheen, "Application of the Three-Dimensional Finite-Difference Time-Domain Method to the Analysis of the Planar Microstrip Circuits",IEEE Trans. MTT, vol. 38, N0. 7 pp.849-857, Jul. 1990.

[15] N.G. Einspruch, VLSI Electronics, Microstructure Science, Vol. 15,Academic Press Inc., Orlando, FL, 1987

[16]Y. Pauleau, "Interconnect Materials for VLSI Circuits", Solid State Tech.,pg. 62,Feb., 1987

[17] E.C. Jordan, Reference Data for Engineers: Radio, Electronics, Computer, and Communications, Howard W. Sams and Co. Inc., Indianapolis, IN,pp. 4-15,1985

[18] S.S. Cohen in, VLSI Electronics, Microstructure Science,ed. N.G. Einspruch, Vol. 15,Academic Press Inc., Orlando, FL, pp.17, 1987

[19] J.M. Ziman, Electrons and Phonons Oxford Univ. Press, London,1962

[20] M.S.P. Lucas,"Title not availible at this time.", J. Appl. Physics, Vol. 36,pp.1632,1965

[21] Z. Bi, "Fast Finite-Difference Time-Domain Analysis of Resonators Using Digital Filtering and Spectrum Estimation Techniques",IEEE Trans. MTT,Vol. 40, No. 8, pp. 1611-1619, Aug. 1992

Appendix: Program listing

This appendix includes the C code for the FD-TD simulator. The program is broken ito several parts:

ahead.c the main header file declaring varibles, constants, and functions.

fd-tda.c the main loop of the program which allocates memory and initiallizes the maticies.

fdnodefunc.c The engine of the simulator. This is where the parameters are calculated and manipulated.

ecases.c stores the electric functions for the various cell types. Basically this is just a large switch statement.

hcases.c stores the magnetic functions for the various cell types.

inout.c collection of functions to write data to a file

ahead.h an extern version of ahead.c

Note this version is setup to insert roughness onto an interconnect structure.

Program: Ahead.c

```
/*****************************************************************
  Header file for Finite Difference program
  includes data structures and global varibles
                Maynard Falconer 6/92
*****************************************************************/

#include <stdio.h>
#include <math.h>

#define Acceleration 1.85
#define PI 3.14159265359
#define EO 8.854187818e-12
#define UO 1.25663706e-6

/******Data  Structures**********/



typedef struct{
        double Ex,Ey,Ez,Hx,Hy,Hz;
        int typ;
        } Node;

/****** Function declarations ******/

Node ***makematrix(     Node ***m,int nrh,int nch,int ndh);
void freematrix();
void nrerror(char error_text[]);
void initmatrix1(int nrh,int nch,int ndh);
void init2matrix1(int nrh,int nch,int ndh);
void  do_fin_diff();
void do_E(int i,int j,int k);
void do_H(int i,int j,int k);
void fwrtEx();
void fwrtE();
void fwrtCt(int Z, double Sig2);

/*****Global  Varibles**************************/

Node ***grid;                           /**3-D matrix current time**/
Node ***gridO;                          /**3-D matrix old time**/
```

```
Node ***gridtemp;                    /**temp for swapping old and
new**/
double dx;                           /**grid spacing**/
double dt;                           /**time increment**/
double to;                           /**input signal info**/
int SIZEI,SIZEJ,SIZEK;
char filename[20]="txCC0000";
char filenameEx[20]="txCC0000Ex";
char filenameCt[20]="txCC0000Ct";
int delta;                  /***current number of time steps taken***/
double CAa,CAb,CBa,CBb,CCa,CCb;
int ht,w,thk;                        /** ht is height, w is width, thk is
thickness**/
int pd,dty,rgh;             /**pd=period,dty=uptime,rgh=roughness**/
int nn;                     /** number of periods in SIZEK **/
int DELTAMAX;
double risetime, holdtime,falltime;
double *INPT;
double *OUTPT20;
double *OUTPT40;
double *OUTPT60;
double *OUTPT80;
double *OUTPT100;
double *OUTPT120;
double *OUTPT140;
double *OUTPT160;
/**double *OUTPT180; */
```

Program: fd-tda.c

```
/*************************************************
* written by Maynard Falconer 3/93
*
*************************************************/


#include "ahead.c"
#include "inout.c"

main()
{


        SIZEI=80;                       /*** X-component Vertical ***/
        SIZEJ=24;                       /**** Y-component Width  ****/
        SIZEK=179;                      /** Z-component Length **/
        dx=(1.0e-7);                    /**** grid spacing size****/
        dt=(sqrt(EO*UO)*dx*0.515);      /***time  increment***/
        to=70;                          /**input wave**/
        ht=20;                          /**distance between strip and
gnd**/
        w=3;                        /** 2*w-1=width of strip**/
        thk=5;                          /*** thickness of strip ***/
        rgh=3;                          /**roughness**/
        pd=10;                              /**period of roughness**/
        dty=5;                          /**uptime of roughness**/
        risetime=0.1e-13;
        holdtime=0.1e-13;
        falltime=0.1e-13;

        DELTAMAX=1001;
        delta=0;

        /** number of periods in SIZEK**/
        nn=(SIZEK+1)%pd;
        if (nn=0){
                nn=(SIZEK+1)/pd;
        }else{
                SIZEK=SIZEK+nn;
                nn=(SIZEK+1)/pd;
        }
```

```
        INPT=(double *)calloc(sizeof(double),DELTAMAX+1);
        OUTPT20=(double *)calloc(sizeof(double),DELTAMAX+1);
        OUTPT40=(double *)calloc(sizeof(double),DELTAMAX+1);
        OUTPT60=(double *)calloc(sizeof(double),DELTAMAX+1);
        OUTPT80=(double *)calloc(sizeof(double),DELTAMAX+1);
       OUTPT100=(double *)calloc(sizeof(double),DELTAMAX+1);
       OUTPT120=(double *)calloc(sizeof(double),DELTAMAX+1);
       OUTPT140=(double *)calloc(sizeof(double),DELTAMAX+1);
       OUTPT160=(double *)calloc(sizeof(double),DELTAMAX+1);
/*      OUTPT180=(double *)calloc(sizeof(double),DELTAMAX+1); */




        grid = makematrix(grid,SIZEI,SIZEJ,SIZEK);
        gridO = makematrix(gridO,SIZEI,SIZEJ,SIZEK);

        initmatrix1(SIZEI,SIZEJ,SIZEK);
/*      init2matrix1(SIZEI,SIZEJ,SIZEK);
*/

        do_fin_diff(grid,gridO);


        fwrtEx();


}


/*******************************************************
This routine allocates memory for a nrl*ncl*ndl matrix
written by M Falconer
        adapted from dmatrix in 'Numerical Recipes in C'
*******************************************************/

Node  ***makematrix(m,nih,njh,nkh)
Node  ***m;
int  nih,njh,nkh;
{
        int i,j;

        m=(Node ***) calloc(sizeof(Node**),(nih+1));
        if (!m) printf("allocation failure 1 in matrix construction\n");
```

```
        for (i=0;i<=nih;i++){
                m[i]=(Node **)calloc(sizeof(Node*),(njh+1));
                if (!m[i]) printf("allocation failure 2 in matrix
construction\n");
        }
        for (i=0;i<=nih;i++){
                for(j=0;j<=njh;j++){
                        m[i][j]=(Node*)calloc(sizeof(Node),(nkh+1));
                        if (!m[i][j]) {
                        printf("allocation failure 3 in matrix construction\n");
                        }
                }
        }
        return m;
}

/*****************************************************
This routine frees up grids' memory for reallocation
                                        M Falconer 6/92
*****************************************************/
void freematrix()
{
        int i,j;
        for (i=SIZEI;i>=0;i--){
                for(j=SIZEJ;j>=0;j--){
                        free(grid[i][j]);
                }
        }
        for (i=SIZEI;i>=0;i--){
                free(*grid[i]);
        }
        free(**grid);

        return;
}


/*************************************************
 error reporting routine from
        'Numerical Recipies in C'
   ************************************************/
void nrerror(error_text)
char error_text[];
```

```
{
        void exit();
        printf("Run-time error.....\n");
        printf("%s\n",error_text);
        printf("Now exiting to System....\n");
        exit(1);
}



/*************************************************************************
***/
/*************************************************************************
***/
/*************************************************************************
***/
/*************************************************************************
***/


void initmatrix1(int nih,int njh,int nkh)
        {
        int i,j,k;

        /****clear matrix*****/
        for(i=0;i<=nih;i++){
                for(j=0;j<=njh;j++){
                        for(k=0;k<=nkh;k++){
                                grid[i][j][k].Ex=0.0;
                                grid[i][j][k].Ey=0.0;
                                grid[i][j][k].Ez=0.0;
                                grid[i][j][k].Hx=0.0;
                                grid[i][j][k].Hy=0.0;
                                grid[i][j][k].Hz=0.0;
                                gridO[i][j][k].Ex=0.0;
                                gridO[i][j][k].Ey=0.0;
                                gridO[i][j][k].Ez=0.0;
                                gridO[i][j][k].Hx=0.0;
                                gridO[i][j][k].Hy=0.0;
                                gridO[i][j][k].Hz=0.0;
                                grid[i][j][k].typ=-99;
                                gridO[i][j][k].typ=-99;
```

```
                }
            }
        }

/** set walls ***/
        /** xz y=0 Neuman boundry this is a permanent wall**/
        /** xz y=SIZE Dirichlet boundry**/
        for(i=1;i<=(nih-1);i++){
            for(k=1;k<=(nkh-1);k++){
                grid[i][0][k].typ=103;
                gridO[i][0][k].typ=103;
                grid[i][njh][k].typ=4;
                gridO[i][njh][k].typ=4;
            }
        }

        /** xy z=0 Dirichlet boundry **/
        /** xy z=SIZE Dirichlet boundry**/
        for(i=1;i<=(nih-1);i++){
            for(j=1;j<=(njh-1);j++){
                grid[i][j][0].typ=1;
                gridO[i][j][0].typ=1;
                grid[i][j][nkh].typ=2;
                gridO[i][j][nkh].typ=2;
            }
        }

        /** yz x=0 metal boundry this is a permanent wall**/
        /** yz x=SIZE Dirichlet boundry**/
        for(j=1;j<=(njh-1);j++){
            for(k=1;k<=(nkh-1);k++){
                grid[0][j][k].typ=5;
                gridO[0][j][k].typ=5;
                grid[nih][j][k].typ=6;
                gridO[nih][j][k].typ=6;
            }
        }

        /*** seams in z direction ***/
        for(k=1;k<=(nkh-1);k++){
                grid[0][0][k].typ=101;
                gridO[0][0][k].typ=101;
                grid[0][njh][k].typ=13;
                gridO[0][njh][k].typ=13;
                grid[nih][0][k].typ=101;
```

```
                    gridO[nih][0][k].typ=101;
                    grid[nih][njh][k].typ=14;
                    gridO[nih][njh][k].typ=14;
}

/*** seams in y direction ***/
for(j=1;j<=(njh-1);j++){
                    grid[0][j][0].typ=15;
                    gridO[0][j][0].typ=15;
                    grid[0][j][nkh].typ=17;
                    gridO[0][j][nkh].typ=17;
                    grid[nih][j][0].typ=16;
                    gridO[nih][j][0].typ=16;
                    grid[nih][j][nkh].typ=18;
                    gridO[nih][j][nkh].typ=18;
}

/*** seams in x direction ***/
for(i=1;i<=(nih-1);i++){
                    grid[i][0][0].typ=101;
                    gridO[i][0][0].typ=101;
                    grid[i][0][nkh].typ=2;
                    gridO[i][0][nkh].typ=2;
                    grid[i][njh][0].typ=108;
                    gridO[i][njh][0].typ=108;
                    grid[i][njh][nkh].typ=10;
                    gridO[i][njh][nkh].typ=10;
}

/*** corners **/
grid[0][0][0].typ=101;
gridO[0][0][0].typ=101;
grid[0][0][nkh].typ=101;
gridO[0][0][nkh].typ=101;
grid[0][njh][0].typ=21;
gridO[0][njh][0].typ=21;
grid[0][njh][nkh].typ=22;
gridO[0][njh][nkh].typ=22;
grid[nih][0][0].typ=101;
gridO[nih][0][0].typ=101;
grid[nih][0][nkh].typ=101;
gridO[nih][0][nkh].typ=101;
grid[nih][njh][0].typ=25;
gridO[nih][njh][0].typ=25;
grid[nih][njh][nkh].typ=26;
```

```
        gridO[nih][njh][nkh].typ=26;


        /*** fill in regular cells ***/
        for(i=1;i<=(nih-1);i++){
                for(j=1;j<=(njh-1);j++){
                        for(k=1;k<=(nkh-1);k++){
                                grid[i][j][k].typ=0;
                                gridO[i][j][k].typ=0;
                        }
                }
        }



        /** input plane **/
        for(i=1;i<=(ht-1);i++){
                for(j=1;j<=w+1;j++){
                        grid[i][j][0].typ=506;
                        gridO[i][j][0].typ=506;
                }
        }
        return;
}


/***********************************************
 This routine enters a center fired pulse instead of an edge
  fired pulse.  The pulse is roughly trapazoidal with finite
   risetime, falltime, and the side is rolled off (i.e. there are
   no abrupt field changes, so ringing is avoided)
***********************************************/
void init2matrix1(int nih,int njh,int nkh)
        {
        int i,j,k,mid;
      int ht1,ht2,fall,rf;
        char c;

/** make front wall completely metal**/
        /** xy z=0 Dirichlet boundry **/
        /** xy z=SIZE Dirichlet boundry**/
        for(i=1;i<=(nih-1);i++){
                for(j=1;j<=(njh-1);j++){
                                grid[i][j][0].typ=1;
                                gridO[i][j][0].typ=1;
```

```
                }
        }

 /** insert Ex and Hy components **/

       mid=(int)SIZEK/2;
         ht1=(int)(holdtime/(2*dt));
         ht2=mid+ht1;
         ht1=mid-ht1;
         if(ht2>=(SIZEK-2)){
                ht2=SIZEK-2;
         }
         if(ht1<=2){
                ht1=2;
         }

         fall= (int)((double)(falltime/dt));
         fall= fall+ht2;
         if(fall>=(SIZEK-1)){
                fall=SIZEK-1;
         }
         rf=(int)(risetime/dt);
         rf= (ht1 - rf);
         if(rf<=1){
                rf=1;
         }

         for(i=1;i<(ht);i++){
                for(j=1;j<=w;j++){
                        for(k=ht1; k<=ht2;k++){
                                gridO[i][j][k].Ex=(dt/(UO*dx))*1.0;

         gridO[i][j][k].Hx=grid[i][j][k].Ex*sqrt(13.0)/(120.0*PI);
                        }
                        for(k=(ht2+1); k<=fall;k++){
                                gridO[i][j][k].Ex=(dt/(UO*dx))*(fall-k)*1.0/(fall-
ht2+1)*1.0;

         gridO[i][j][k].Hx=(grid[i][j][k].Ex)*sqrt(13.0)/(120.0*PI);
                        }
                        for(k=rf; k<ht1;k++){
                                gridO[i][j][k].Ex=(dt/(UO*dx))*(k-rf)*1.0/(ht1-
rf*1.0);

         gridO[i][j][k].Hx=(grid[i][j][k].Ex)*sqrt(13.0)/(120.0*PI);
```

```
                }


        }
    }

    for(i=1;i<ht;i++){
        for(j=w+1;j<w+4;j++){
            for(k=1;k<SIZEK;k++){
            gridO[i][j][k].Ex=gridO[i][w][k].Ex*(w+4-j)/4.0;
            gridO[i][j][k].Hy=(grid[i][j][k].Ex)*sqrt(13.0)/(120.0*PI);
            }
        }
    }

    return;
}
```

Program fdnodefunc.c

```
#include "ahead.h"


/*************************************************************
Loop to preform finite difference-time domain on a matrix

                                    Maynard Falconer 3/93
*************************************************************/
void do_fin_diff()
        {
        Node ***gridtemp;
        char c;
        int i,j,k,N;
        double er1,er2,sig1,sig2,er3,sig3;
        c=0;
        er1=1.0;
        sig1=0.0;
        er2=4.0;
        sig2=0.0;
        er3=1.0;
        sig3=1e3;    /*5.813e4;*/
        CAa=1.0;
        CAb=(dt*dt/(UO*EO*dx*dx));
        CBa=1.0;
        CBb=(dt*dt/(UO*EO*dx*dx));
        CCa=1.0;
        CCb=(dt*dt/(UO*EO*dx*dx));



        /** check typ **/
        for(i=0;i<=SIZEI;i++){
                for(j=0;j<=SIZEJ;j++){
                        for(k=0;k<=SIZEK;k++){
                                if(grid[i][j][k].typ==-99){
                                        printf("uninitiated cell
%d,%d,%d\n",i,j,k);
                                        c=getchar();
                                }
```

```
                        if(gridO[i][j][k].typ==-99){
                                printf("uninitiated cell
%d,%d,%d\n",i,j,k);
                                c=getchar();
                        }
                }
        }
}

/*      fwrtE(); */

/** This is the main loop **/
        delta=1;
        while (delta<=DELTAMAX){

/** H cases are straight forward and done here **/
                for(i=SIZEI;i>=0;i--){
                        for(j=SIZEJ;j>=0;j--){
                                for(k=SIZEK;k>=0;k--){

                                        do_H(i,j,k);
                                }
                        }
                }

/** Substrate region **/
                CAa=(2*EO-dt*sig2/er2)/(2*EO+dt*sig2/er2);
                CAb=(dt*dt/(UO*EO*dx*dx))/(er2+dt*sig2/(2*EO));
                CBa=(2*EO-dt*sig2/er2)/(2*EO+dt*sig2/er2);
                CBb=(dt*dt/(UO*EO*dx*dx))/(er2+dt*sig2/(2*EO));
                CCa=(2*EO-dt*sig2/er2)/(2*EO+dt*sig2/er2);
                CCb=(dt*dt/(UO*EO*dx*dx))/(er2+dt*sig2/(2*EO));
                for(i=(ht-1);i>=0;i--){
                        for(j=SIZEJ;j>=0;j--){
                                for(k=SIZEK;k>=0;k--){
                                        do_E(i,j,k);
                                }
                        }
                }

/** boundry interface cases **/
                /** between substrate and air **/
                CAa=(2*EO-dt*sig2/er2)/(2*EO+dt*sig2/er2);
                CAb=(dt*dt/(UO*EO*dx*dx))/(er2+dt*sig2/(2*EO));
```

```
            CBa=(2*EO-
dt*(sig2+sig1)/(er2+er1))/(2*EO+dt*(sig2+sig1)/(er2+er1));

        CBb=(dt*dt/(UO*EO*dx*dx))/((er2+er1)/2+dt*(sig2+sig1)/(4*EO));
            CCa=(2*EO-
dt*(sig2+sig1)/(er2+er1))/(2*EO+dt*(sig2+sig1)/(er2+er1));

        CCb=(dt*dt/(UO*EO*dx*dx))/((er2+er1)/2+dt*(sig2+sig1)/(4*EO));
            i=ht;
            for(j=SIZEJ;j>=(w+2);j--){
                for(k=SIZEK;k>=0;k--){
                    do_E(i,j,k);
                }
            }

            /** between substrate and strip **/
            CAa=(2*EO-dt*sig2/er2)/(2*EO+dt*sig2/er2);
            CAb=(dt*dt/(UO*EO*dx*dx))/(er2+dt*sig2/(2*EO));
            CBa=(2*EO-
dt*(sig2+sig3)/(er2+er3))/(2*EO+dt*(sig2+sig3)/(er2+er3));

        CBb=(dt*dt/(UO*EO*dx*dx))/((er2+er3)/2+dt*(sig2+sig3)/(4*EO));
            CCa=(2*EO-
dt*(sig2+sig3)/(er2+er3))/(2*EO+dt*(sig2+sig3)/(er2+er3));

        CCb=(dt*dt/(UO*EO*dx*dx))/((er2+er3)/2+dt*(sig2+sig3)/(4*EO));
            i=ht;
            for(j=w;j>=0;j--){
                for(k=SIZEK;k>=0;k--){
                    do_E(i,j,k);
                }
            }

            /** between substrate and air and strip**/
            CAa=(2*EO-
dt*(sig2+sig3)/(er2+er3))/(2*EO+dt*(sig2+sig3)/(er2+er3));

        CAb=(dt*dt/(UO*EO*dx*dx))/((er2+er3)/2+dt*(sig2+sig3)/(4*EO));
            CBa=(2*EO-
dt*(sig2+sig1)/(er2+er1))/(2*EO+dt*(sig2+sig1)/(er2+er1));

        CBb=(dt*dt/(UO*EO*dx*dx))/((er2+er1)/2+dt*(sig2+sig1)/(4*EO));
            CCa=(2*EO-
dt*(sig2+sig1+sig3)/(er2+er1+er3))/(2*EO+dt*(sig2+sig1+sig3)/(er2+er1+er3
));
```

```
        CCb=(dt*dt/(UO*EO*dx*dx))/((er2+er1+er3)/3+dt*(sig2+sig1+sig3)/(
6*EO));
                i=ht;
                j=w+1;
                for(k=SIZEK;k>=0;k--){
                        do_E(i,j,k);
                }

/** Air region **/
                CAa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CAb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
                CBa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CBb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
                CCa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CCb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
                for(i=SIZEI;i>=(ht+thk+rgh+2);i--){
                        for(j=SIZEJ;j>=0;j--){
                                for(k=SIZEK;k>=0;k--){
                                        do_E(i,j,k);
                                }
                        }
                }

/** fill in metal strip**/
                CAa=(2*EO-dt*sig3/er3)/(2*EO+dt*sig3/er3);
                CAb=dt*dt/(UO*EO*dx*dx)/(er3+dt*sig3/(2*EO));
                CBa=(2*EO-dt*sig3/er3)/(2*EO+dt*sig3/er3);
                CBb=dt*dt/(UO*EO*dx*dx)/(er3+dt*sig3/(2*EO));
                CCa=(2*EO-dt*sig3/er3)/(2*EO+dt*sig3/er3);
                CCb=dt*dt/(UO*EO*dx*dx)/(er3+dt*sig3/(2*EO));
                for(i=(ht+thk-rgh+1);i>(ht+1);i--){
                        for(j=w;j>=0;j--){
                                for(k=0;k<=SIZEK;k++){
                                        do_E(i,j,k);
                                }
                        }
                }

/** Air on face of metal strip **/
                CAa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CAb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
                CBa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
```

```
            CBb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
            CCa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

      CCb=(dt*dt/(UO*EO*dx*dx))/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
            j=w+1;
            for(i=(ht+1);i<(ht+thk-rgh+1);i++){
                  for(k=SIZEK;k>=0;k--){
                        do_E(i,j,k);
                  }
            }

/** air between strip and wall in Y direction***/
            CAa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
            CAb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
            CBa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
            CBb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
            CCa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
            CCb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
            for(i=(ht+thk+rgh+2);i>=(ht+1);i--){
                  for(j=SIZEJ;j>=w+2;j--){
                        for(k=SIZEK;k>=0;k--){
                              do_E(i,j,k);
                        }
                  }
            }


/*****************************************************************/
/*********deal with roughness*********/
for(N=0;N<nn;N++){


/** air strip interfaces in YZ plane**/
            CAa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
            CAb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
            CBa=(2*EO-
dt*(sig1+sig3)/(er1+er3))/(2*EO+dt*(sig1+sig3)/(er1+er3));

      CBb=(dt*dt/(UO*EO*dx*dx))/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
            CCa=(2*EO-
dt*(sig1+sig3)/(er1+er3))/(2*EO+dt*(sig1+sig3)/(er1+er3));

      CCb=(dt*dt/(UO*EO*dx*dx))/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
            i=ht+thk-rgh+1;
```

```
                for(j=w;j>=0;j--){
                        for(k=(N*pd+1);k<((N+1)*pd-dty-1);k++){
                                do_E(i,j,k);
                        }
                }
                i=ht+thk+rgh+1;
                for(j=w;j>=0;j--){
                        for(k=((N+1)*pd-dty);k<((N+1)*pd);k++){
                                do_E(i,j,k);
                        }
                }

/** Air strip interfaces in XY plane **/

                CAa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CAb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
                CBa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CBb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
                CCa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CCb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
        /** frontsides in metal **/
                k=((N+1)*pd-dty);
                for(i=(ht+thk-rgh+1);i<=(ht+thk+rgh);i++){
                        for(j=w;j>=0;j--){
                                do_E(i,j,k);
                        }
                }
        /** backsides out of metal **/
                k=(N*pd);
                for(i=(ht+thk-rgh+2);i<=(ht+thk+rgh);i++){
                        for(j=w;j>=0;j--){
                                do_E(i,j,k);
                        }
                }
/** corners/edges **/
        /**backside in bottom corner**/
                CAa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CAb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
```

```
            CBa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

      CBb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
            CCa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

      CCb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
            k=(N*pd);
            i=(ht+thk-rgh+1);
            for(j=w;j>=0;j--){
                  do_E(i,j,k);
            }

      /**backside in top corner**/
            CAa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
            CAb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
            CBa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

      CBb=(dt*dt/(UO*EO*dx*dx))/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
            CCa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
            CCb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
            k=(N*pd);
            i=(ht+thk+rgh+1);
            for(j=w;j>=0;j--){
                  do_E(i,j,k);
            }

/** fill in metal bump**/
            CAa=(2*EO-dt*sig3/er3)/(2*EO+dt*sig3/er3);
            CAb=dt*dt/(UO*EO*dx*dx)/(er3+dt*sig3/(2*EO));
            CBa=(2*EO-dt*sig3/er3)/(2*EO+dt*sig3/er3);
            CBb=dt*dt/(UO*EO*dx*dx)/(er3+dt*sig3/(2*EO));
            CCa=(2*EO-dt*sig3/er3)/(2*EO+dt*sig3/er3);
            CCb=dt*dt/(UO*EO*dx*dx)/(er3+dt*sig3/(2*EO));
            for(i=(ht+thk-rgh+1);i<(ht+thk+rgh+1);i++){
                  for(j=w;j>=0;j--){
                        for(k=((N+1)*pd-dty+1);k<((N+1)*pd);k++){
                              do_E(i,j,k);
                        }
                  }
            }
/** fill in air gap**/
            CAa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
```

```
                CAb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
                CBa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CBb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
                CCa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CCb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
                for(i=(ht+thk-rgh+2);i<=(ht+thk+rgh+1);i++){
                        for(j=w+1;j>=0;j--){
                                for(k=((N)*pd+1);k<((N+1)*pd-dty);k++){
                                        do_E(i,j,k);
                                }
                        }
                }


/** between air and strip edge **/
        /**top edge**/
                CAa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CAb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
                CBa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CBb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
                CCa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CCb=(dt*dt/(UO*EO*dx*dx))/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
                i=ht+thk-rgh+1;
                j=w+1;
                for(k=(N*pd+1);k<((N+1)*pd-dty);k++){
                                do_E(i,j,k);
                }
                i=ht+thk+rgh;
                j=w+1;
                for(k=((N+1)*pd-dty);k<((N+1)*pd-1);k++){
                                do_E(i,j,k);
                }

        /**back edge**/
                CAa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CAb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
                CBa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CBb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
                CCa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CCb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
                k=N*pd;
```

```
                j=w+1;
                for(i=(ht+thk-rgh+2);i<=(ht+thk+rgh);i++){
                        do_E(i,j,k);
                }
        /**get backedge corner**/
                CCa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CCb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
                i=ht+thk-rgh+1;
                        do_E(i,j,k);
        /**get backedge top corner**/
                CAa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CAb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
                CCa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CCb=(dt*dt/(UO*EO*dx*dx))/(er1+dt*sig1/(2*EO));
                i=ht+thk+rgh+1;
                        do_E(i,j,k);

        /** fill in metal bump faces**/
                CAa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CAb=dt*dt/(UO*EO*dx*dx)/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
                CBa=(2*EO-dt*sig1/er1)/(2*EO+dt*sig1/er1);
                CBb=dt*dt/(UO*EO*dx*dx)/(er1+dt*sig1/(2*EO));
                CCa=(2*EO-dt*(sig1+sig3)/(er1+er3))
/(2*EO+dt*(sig1+sig3)/(er1+er3));

        CCb=(dt*dt/(UO*EO*dx*dx))/((er1+er3)/2+dt*(sig1+sig3)/(4*EO));
                j=w+1;
                for(i=(ht+thk-rgh+1);i<(ht+thk+rgh);i++){
                        for(k=((N+1)*pd-dty);k<=((N+1)*pd-1);k++){
                                do_E(i,j,k);
                        }
                }


} /**end of for N loop **/


        INPT[delta]=grid[ht-1][1][0].Ex*(UO*dx/dt);
        OUTPT20[delta]=grid[ht-1][1][20].Ex*(UO*dx/dt);
        OUTPT40[delta]=grid[ht-1][1][40].Ex*(UO*dx/dt);
        OUTPT60[delta]=grid[ht-1][1][60].Ex*(UO*dx/dt);
```

```
        OUTPT80[delta]=grid[ht-1][1][80].Ex*(UO*dx/dt);
      OUTPT100[delta]=grid[ht-1][1][100].Ex*(UO*dx/dt);
      OUTPT120[delta]=grid[ht-1][1][120].Ex*(UO*dx/dt);
      OUTPT140[delta]=grid[ht-1][1][140].Ex*(UO*dx/dt);
      OUTPT160[delta]=grid[ht-1][1][160].Ex*(UO*dx/dt);
/*    OUTPT180[delta]=grid[ht-1][1][180].Ex*(UO*dx/dt); */


      if(0==delta%50){
            fwrtE();
      }

/*    if(delta==300){
            fwrtCt(20,sig3);
      }
*/
      /***swap grids**/
      gridtemp=gridO;
      gridO=grid;
      grid=gridtemp;



      delta=delta+1;
      }

      /***swap grids**/
      gridtemp=gridO;
      gridO=grid;
      grid=gridtemp;

      return;
}
```

Program: ecases.c

```
#include "ahead.h"

void do_E(int i,int j,int k)
{

if(grid[i][j][k].typ<100){

switch(grid[i][j][k].typ){

        case 0:
                /** type 0 is a standard nonboundry-homogenous case**/

                grid[i][j][k].Ex=grid[i][j][k].Hz-grid[i][j-1][k].Hz;
                grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
                grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

                grid[i][j][k].Ey=grid[i][j][k].Hx-grid[i][j][k-1].Hx;
                grid[i][j][k].Ey+=grid[i-1][j][k].Hz-grid[i][j][k].Hz;
                grid[i][j][k].Ey=(CBb*grid[i][j][k].Ey)+(CBa*gridO[i][j][k].Ey);

                grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
                grid[i][j][k].Ez+=grid[i][j-1][k].Hx-grid[i][j][k].Hx;
                grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

                break;

        case 1:
                /** type 1 is xy z=0 dirichlet E **/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
                grid[i][j][k].Ez+=grid[i][j-1][k].Hx-grid[i][j][k].Hx;
                grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

                break;

case 2:case 4:case 6:case 8:case 9:case 10:case 12:case 13:case 14:case 16:case 17:
case 18:case 20:case 21:case 22:case 23:case 24:case 25:case 26:case 7:case 11:
```

```
case 15:case 19:
            /** type buffer dirichlet E**/


            grid[i][j][k].Ex=0.0;

            grid[i][j][k].Ey=0.0;

            grid[i][j][k].Ez=0.0;

            break;

      case 3:
            /** type 3 is xz y=0 dirichlet E**/

            grid[i][j][k].Ex=0.0;

            grid[i][j][k].Ey=grid[i][j][k].Hx-grid[i][j][k-1].Hx;
            grid[i][j][k].Ey+=grid[i-1][j][k].Hz-grid[i][j][k].Hz;
            grid[i][j][k].Ey=(CBb*grid[i][j][k].Ey)+(CBa*gridO[i][j][k].Ey);

            grid[i][j][k].Ez=0.0;

            break;

      case 5:
            /** type 5 is yz x=0 **/

            grid[i][j][k].Ex=grid[i][j][k].Hz-grid[i][j-1][k].Hz;
            grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
            grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

            grid[i][j][k].Ey=0.0;

            grid[i][j][k].Ez=0.0;

            break;

      default:
                  /**UNKNOWN TYPE**/
                  printf("error unknown type!!!%d\n",grid[i][j][k].typ);
                  break;

      }
```

```
}else{
if(grid[i][j][k].typ<200){

switch(grid[i][j][k].typ){

        case 101: case 103: case 105: case 107: case 108: case 109: case 111: case
112:
        case 113: case 115: case 116: case 117: case 119: case 120: case 121: case
122:
        case 123: case 124: case 125: case 126:
                /** type 101 is Neuman E center of cell buffer**/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=0.0;

                break;

        case 102:
                /** type 102 is xy z=SIZE Neuman E center of cell**/

                grid[i][j][k].Ex=grid[i][j][k].Hz-grid[i][j-1][k].Hz;
                grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
                grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

                grid[i][j][k].Ey=grid[i][j][k].Hx-grid[i][j][k-1].Hx;
                grid[i][j][k].Ey+=grid[i-1][j][k].Hz-grid[i][j][k].Hz;
                grid[i][j][k].Ey=(CBb*grid[i][j][k].Ey)+(CBa*gridO[i][j][k].Ey);

                grid[i][j][k].Ez=0.0;

                break;

        case 104:
                /** type 104 is xz y=SIZE Neuman E center of cell**/

                grid[i][j][k].Ex=grid[i][j][k].Hz-grid[i][j-1][k].Hz;
                grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
                grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
```

```
                grid[i][j][k].Ez+=grid[i][j-1][k].Hx-grid[i][j][k].Hx;
                grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

                break;

        case 106:
                /** type 106 is yz x=SIZE Neuman E center of cell**/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=grid[i][j][k].Hx-grid[i][j][k-1].Hx;
                grid[i][j][k].Ey+=grid[i-1][j][k].Hz-grid[i][j][k].Hz;
                grid[i][j][k].Ey=(CBb*grid[i][j][k].Ey)+(CBa*gridO[i][j][k].Ey);

                grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
                grid[i][j][k].Ez+=grid[i][j-1][k].Hx-grid[i][j][k].Hx;
                grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

                break;

        case 110:
                /** type 110 is xy z=SIZE xz y=SIZE Neuman E center of
cell**/

                grid[i][j][k].Ex=grid[i][j][k].Hz-grid[i][j-1][k].Hz;
                grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
                grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=0.0;

                break;

        case 114:
                /** type 114 is yz x=SIZE xz y=SIZE Neuman E center of
cell**/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
                grid[i][j][k].Ez+=grid[i][j-1][k].Hx-grid[i][j][k].Hx;
                grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);
```

```
                break;

        case 118:
                /** type 118 is xy z=SIZE yz x=SIZE Neuman E center of
cell**/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=grid[i][j][k].Hx-grid[i][j][k-1].Hx;
                grid[i][j][k].Ey+=grid[i-1][j][k].Hz-grid[i][j][k].Hz;
                grid[i][j][k].Ey=(CBb*grid[i][j][k].Ey)+(CBa*gridO[i][j][k].Ey);

                grid[i][j][k].Ez=0.0;

                break;

        default:
                        /**UNKNOWN TYPE**/
                        printf("error unknown type!!!%d\n",grid[i][j][k].typ);
                        break;

        } /*** end of switch staement ***/

}else{

switch(grid[i][j][k].typ){

        case 203:
                /** type 203 is xz y=0 Neuman boundry uncentered **/

                grid[i][j][k].Ex=2*grid[i][j][k].Hz;
                grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
                grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

                grid[i][j][k].Ey=grid[i][j][k].Hx-grid[i][j][k-1].Hx;
                grid[i][j][k].Ey+=grid[i-1][j][k].Hz-grid[i][j][k].Hz;
                grid[i][j][k].Ey=(CBb*grid[i][j][k].Ey)+(CBa*gridO[i][j][k].Ey);

                grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
                grid[i][j][k].Ez-=2*grid[i][j][k].Hx;
                grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

                break;
```

```
case 501:
        /** type 501 Neuman xz y=SIZE dirichlet yz x=o ***/

        grid[i][j][k].Ex=grid[i][j][k].Hz-grid[i][j-1][k].Hz;
        grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
        grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

        grid[i][j][k].Ey=0.0;

        grid[i][j][k].Ez=0.0;

        break;

case 502:
        /** type 502 Neuman xy z=SIZE dirichlet yz x=o ***/

        grid[i][j][k].Ex=grid[i][j][k].Hz-grid[i][j-1][k].Hz;
        grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
        grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

        grid[i][j][k].Ey=0.0;

        grid[i][j][k].Ez=0.0;

        break;

case 503:
        /** type 503 Neuman xz y=SIZE dirichlet xy z=0 ***/

        grid[i][j][k].Ex=0.0;

        grid[i][j][k].Ey=0.0;

        grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
        grid[i][j][k].Ez+=grid[i][j-1][k].Hx-grid[i][j][k].Hx;
        grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

        break;

case 504:
        /** type 504 is a metal strip end piece**/
        /** only ez=0 **/

        grid[i][j][k].Ex=grid[i][j][k].Hz-grid[i][j-1][k].Hz;
        grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
```

```
                grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

                grid[i][j][k].Ey=grid[i][j][k].Hx-grid[i][j][k-1].Hx;
                grid[i][j][k].Ey+=grid[i-1][j][k].Hz-grid[i][j][k].Hz;
                grid[i][j][k].Ey=(CBb*grid[i][j][k].Ey)+(CBa*gridO[i][j][k].Ey);

                grid[i][j][k].Ez=0.0;

                break;

        case 505:
                /** input plane Neuman case **/

                grid[i][j][k].Ex=(dt/(UO*dx))*exp((-1)*((delta-to)*(delta-
to))/(to*to));

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=0.0;

        /*      printf("\t\t Ex(%d,%d,%d)=%6.4e\r",i,j,k,grid[i][j][k].Ex);
                */
                break;

        case 506:
                /** input plane Dirichlet case **/

        if(delta<3){
                grid[i][j][k].Ex=0.0;
        }
     if((delta>234) && (delta<351)){
        grid[i][j][k].Ex=(dt/(UO*dx))*((351.0-delta)/(116.0));
     }

        if(delta>=3 && delta<119){
                grid[i][j][k].Ex=(dt/(UO*dx))*(delta-3.0)/(116.0);
        }
     if(delta>=119 && delta<=234){
          grid[i][j][k].Ex=(dt/(UO*dx))*1.0;
     }
        if(delta>=351){
                grid[i][j][k].Ex=0.0;
        }

                grid[i][j][k].Ey=0.0;
```

```
        grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
        grid[i][j][k].Ez+=grid[i][j-1][k].Hx-grid[i][j][k].Hx;
        grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

        break;

case 507:
        /** Neuman yz x=size Dirichlet xy z=0 **/

        grid[i][j][k].Ex=0.0;

        grid[i][j][k].Ey=0.0;

        grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
        grid[i][j][k].Ez+=grid[i][j-1][k].Hx-grid[i][j][k].Hx;
        grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

        break;

case 508:
        /** corner Neuman xz y=size Dirichlet xy z=0 yz x=0 **/

        grid[i][j][k].Ex=0.0;

        grid[i][j][k].Ey=0.0;

        grid[i][j][k].Ez=0.0;

        break;

case 509:
        /** type 509 is xy z=0 dirichlet E & Ez=0 **/

        grid[i][j][k].Ex=0.0;

        grid[i][j][k].Ey=0.0;

        grid[i][j][k].Ez=0.0;

        break;

case 510:
        /** type 510 is xz y=0 Neuman UC yz x=0 Dirichlet **/
```

```
                grid[i][j][k].Ex=2*grid[i][j][k].Hz;
                grid[i][j][k].Ex+=grid[i][j][k-1].Hy-grid[i][j][k].Hy;
                grid[i][j][k].Ex=(CAb*grid[i][j][k].Ex)+(CAa*gridO[i][j][k].Ex);

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=0.0;

                break;

                case 511:
                /** type 511 is xz y=0 Neuman UC yz x=Size Dirichlet
buffer!**/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=0.0;

                break;

                case 512:
                /** type 512 is xz y=0 Neuman UC xy z=0 Dirichlet **/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=grid[i][j][k].Hy-grid[i-1][j][k].Hy;
                grid[i][j][k].Ez-=2*grid[i][j][k].Hx;
                grid[i][j][k].Ez=(CCb*grid[i][j][k].Ez)+(CCa*gridO[i][j][k].Ez);

                break;

        case 513:
                /** type 513 is xz y=0 Neuman UC xy z=Size Dirichlet
buffer!**/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=0.0;
```

```
                break;

        case 514:
                /** type 514 is xz y=0 Neuman UC xy z=0 & yz x=0 dirichlet**/

                grid[i][j][k].Ex=0.0;

                grid[i][j][k].Ey=0.0;

                grid[i][j][k].Ez=0.0;

                break;

        default:
                        /**UNKNOWN TYPE**/
                        printf("error unknown type!!!%d\n",grid[i][j][k].typ);
                        break;

        } /** end of switch statement **/
}       /** innner else statement **/
} /** outer else statement **/

        return;
}
```

Program: hcases.c

```
#include "ahead.h"

void do_H(int i,int j,int k)
{

if(grid[i][j][k].typ<100){

switch(grid[i][j][k].typ){

        case 0:
                /** type 0 is a standard nonboundry-homogenous case**/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 1:
                /** type 1 is xy z=0 dirichlet E **/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;

case 2:case 4:case 6:case 8:case 9:case 10:case 12:case 13:case 14:case 16:case 17:
case 18:case 19:case 20:case 21:case 22:case 23:case 24:case 25:case 26:
```

```
                /** type buffer dirichlet E**/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=0.0;

                break;

        case 3:
                /** type 3 is xz y=0 dirichlet E**/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 5:
                /** type 5 is yz x=0 **/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 7:
                /** type 7 is xy z=0  xz y=0  dirichlet E**/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
```

```
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=0.0;

                break;

        case 11:
                /** type 11 is xz y=0 yz x=0 dirichlet E**/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 15:
                /** type 15 is xy z=0 yz x=0 dirichlet E **/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;

        default:
                        /**UNKNOWN TYPE**/
                printf("error unknown type!!!%d\n",grid[i][j][k].typ);
                        break;

        }

}else{
if(grid[i][j][k].typ<200){

switch(grid[i][j][k].typ){
```

```
        case 101: case 103: case 105: case 107: case 108: case 109: case 111: case
112:
        case 113: case 115: case 116: case 117: case 119: case 120: case 121: case
122:
        case 123: case 124: case 125:case 110:case 114:case 118:case 126:
                /** type 101 is Neuman E center of cell buffer**/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=0.0;

                break;


        case 102:
                /** type 102 is xy z=SIZE Neuman E center of cell**/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 104:
                /** type 104 is xz y=SIZE Neuman E center of cell**/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;

        case 106:
                /** type 106 is yz x=SIZE Neuman E center of cell**/
```

```
                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=0.0;

                break;

        default:
                        /**UNKNOWN TYPE**/
                        printf("error unknown type!!!%d\n",grid[i][j][k].typ);
                        break;

        } /*** end of switch staement ***/

}else{

switch(grid[i][j][k].typ){

        case 203:
                /** type 203 is xz y=0 Neuman boundry uncentered **/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 501:
                /** type 501 Neuman xz y=SIZE dirichlet yz x=o ***/

                grid[i][j][k].Hx=0.0;
```

```
                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;

        case 502:
                /** type 502 Neuman xy z=SIZE dirichlet yz x=o ***/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 503:
                /** type 503 Neuman xz y=SIZE dirichlet xy z=0 ***/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;


        case 504:
                /** type 504 is a metal strip end piece**/
                /** only ez=0 **/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
```

```
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 505:
                /** input plane Neuman type **/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 506:
                /** input plane Dirichlet type **/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

        case 507:
                /** Neuman yz x=size Dirichlet xy z=0 **/
```

```
                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=0.0;

                break;

        case 508:
                /** corner Neuman xz y=size Dirichlet xy z=0 yz x=0 **/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;

        case 509:
                /** type 509 is xy z=0 dirichlet E &Ez=0 **/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;

        case 510:
                /** type 510 is xz y=0 Neuman UC yz x=0 Dirichlet **/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
```

```
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=gridO[i][j][k].Hz+gridO[i][j+1][k].Ex-
gridO[i][j][k].Ex;
                grid[i][j][k].Hz+=gridO[i][j][k].Ey-gridO[i+1][j][k].Ey;

                break;

                case 511:
                /** type 511 is xz y=0 Neuman UC yz x=Size Dirichlet
buffer!**/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=0.0;

                break;

                case 512:
                /** type 512 is xz y=0 Neuman UC xy z=0 Dirichlet **/

                grid[i][j][k].Hx=gridO[i][j][k].Hx+gridO[i][j][k+1].Ey-
gridO[i][j][k].Ey;
                grid[i][j][k].Hx+=gridO[i][j][k].Ez-gridO[i][j+1][k].Ez;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;

                case 513:
                /** type 513 is xz y=0 Neuman UC xy z=Size Dirichlet buffer!
**/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=0.0;

                grid[i][j][k].Hz=0.0;
```

```
                break;

        case 514:
                /** type 514 is xz y=0 Neuman UC xy z=0 & yz x=0 dirichlet**/

                grid[i][j][k].Hx=0.0;

                grid[i][j][k].Hy=gridO[i][j][k].Hy+gridO[i+1][j][k].Ez-
gridO[i][j][k].Ez;
                grid[i][j][k].Hy+=gridO[i][j][k].Ex-gridO[i][j][k+1].Ex;

                grid[i][j][k].Hz=0.0;

                break;

        default:
                        /**UNKNOWN TYPE**/
                        printf("error unknown type!!!%d\n",grid[i][j][k].typ);
                        break;

        } /** end of switch statement **/
}       /** innner else statement **/
} /** outer else statement **/

        return;
}
```

Program: inout.c

```
/************************************************
 Functions to store data into a file
                          Maynard Falconer 3/93
  ************************************************/

void fwrtEx()
{
        int j;
        FILE *fp;

        fp=fopen(filename,"w");
        rewind(fp);
        for(j=1;j<=DELTAMAX;j++){
                fprintf(fp,"%20.18e\n",INPT[j]);
        }
        fflush(fp);
        fclose(fp);

        filename[6]='2';
        fp=fopen(filename,"w");
        rewind(fp);
        for(j=1;j<=DELTAMAX;j++){
                fprintf(fp,"%20.18e\n",OUTPT20[j]);
        }
        fflush(fp);
        fclose(fp);

        filename[6]='4';
        fp=fopen(filename,"w");
        rewind(fp);
        for(j=1;j<=DELTAMAX;j++){
                fprintf(fp,"%20.18e\n",OUTPT40[j]);
        }
        fflush(fp);
        fclose(fp);

        filename[6]='6';
        fp=fopen(filename,"w");
        rewind(fp);
        for(j=1;j<=DELTAMAX;j++){
                fprintf(fp,"%20.18e\n",OUTPT60[j]);
        }
        fflush(fp);
```

```
    fclose(fp);

    filename[6]='8';
    fp=fopen(filename,"w");
    rewind(fp);
    for(j=1;j<=DELTAMAX;j++){
            fprintf(fp,"%20.18e\n",OUTPT80[j]);
    }
    fflush(fp);
    fclose(fp);

    filename[5]='1';
  filename[6]='0';
   fp=fopen(filename,"w");
 rewind(fp);
  for(j=1;j<=DELTAMAX;j++){
        fprintf(fp,"%20.18e\n",OUTPT100[j]);
}
 fflush(fp);
fclose(fp);

 filename[6]='2';
    filename[5]='1';
  fp=fopen(filename,"w");
 rewind(fp);
  for(j=1;j<=DELTAMAX;j++){
        fprintf(fp,"%20.18e\n",OUTPT120[j]);
}
 fflush(fp);
fclose(fp);

 filename[6]='4';
    filename[5]='1';
  fp=fopen(filename,"w");
 rewind(fp);
  for(j=1;j<=DELTAMAX;j++){
        fprintf(fp,"%20.18e\n",OUTPT140[j]);
}
 fflush(fp);
fclose(fp);

 filename[6]='6';
    filename[5]='1';
  fp=fopen(filename,"w");
 rewind(fp);
```

```
        for(j=1;j<=DELTAMAX;j++){
            fprintf(fp,"%20.18e\n",OUTPT160[j]);
        }
        fflush(fp);
        fclose(fp);
/*
        filename[6]='8';
         filename[5]='1';
        fp=fopen(filename,"w");
        rewind(fp);
        for(j=1;j<=DELTAMAX;j++){
            fprintf(fp,"%20.18e\n",OUTPT180[j]);
        }
        fflush(fp);
        fclose(fp);

*/
        return;
}
void fwrtE()
{
        int h,i,j,k,temp;
        FILE *fp;

        temp=delta;
        h=temp/1000;
        temp=temp%1000;
        i=temp/100;
        temp=temp%100;
        j=temp/10;
        k=temp%10;
        filenameEx[4]=h+'0';
        filenameEx[5]=i+'0';
        filenameEx[6]=j+'0';
        filenameEx[7]=k+'0';
        fp=fopen(filenameEx,"w");
        rewind(fp);
        for(k=0;k<=SIZEK;k++){
                for(j=1;j<=SIZEJ;j++){
                        fprintf(fp,"%20.18e ",(UO*dx/dt)*gridO[(ht-1)][j][k].Ex);
                }
                fprintf(fp,"\n");
        }
        fclose(fp);
        return;
```

```
}
void fwrtCt( int Z,  double Sig2)
{
        int n,h,i,j,k,temp;
        FILE *fp;

        for (n=Z; n<Z+3*pd; n++){
                temp=n;
                h=temp/1000;
                temp=temp%1000;
                i=temp/100;
                temp=temp%100;
                j=temp/10;
                k=temp%10;
                filenameCt[4]=h+'0';
                filenameCt[5]=i+'0';
                filenameCt[6]=j+'0';
                filenameCt[7]=k+'0';
                fp=fopen(filenameCt,"w");
                rewind(fp);
                for(j=ht-1;j<=ht+rgh+2;j++){
                        for(i=1;i<=w+2;i++){
                        fprintf(fp,"%20.18e ",(UO*dx/dt)*gridO[i][j][n].Ex);
                        }
                        fprintf(fp,"\n");
                }
                fclose(fp);
        }
        return;
}
```

Program: ahead.h

```
/*****************************************************************
  Header file for Finite Difference program
  includes data structures and global varibles
              Maynard Falconer 3/93
*****************************************************************/


#include <stdio.h>
#include <math.h>

#define Acceleration 1.85
#define PI 3.14159265359
#define EO 8.854187818e-12
#define UO 1.25663706e-6

/******Data  Structures**********/

typedef struct{
        double Ex,Ey,Ez,Hx,Hy,Hz;
        int typ;
        } Node;



/*****Global  Varibles*************************/

extern Node ***grid;                          /**3-D matrix current
time**/
extern Node ***gridO;                         /**3-D matrix old time**/
extern Node ***gridtemp;                      /**temp for swapping old
and new**/
extern double dx;                             /**grid spacing**/
extern double dt;                             /**time increment**/
extern double to;                             /**input signal info**/
extern int SIZEI,SIZEJ,SIZEK;
extern char filename[20];
extern char filenameEx[20];
extern char filenameCt[20];
extern int delta;                             /***current number of time
steps taken***/
extern double CAa,CAb,CBa,CBb,CCa,CCb;
extern int ht,w,thk;
extern int  pd,dty,rgh;
```

```
extern int nn;

extern int DELTAMAX;
extern double risetime,holdtime, falltime;
extern double *INPT;
extern double *OUTPT20;
extern double *OUTPT40;
extern double *OUTPT60;
extern double *OUTPT80;
extern double *OUTPT100;
extern double *OUTPT120;
extern double *OUTPT140;
extern double *OUTPT160;
/***extern double *OUTPT180;  **/
```