NASA-CR-193619

P 182

IMPLEMENTATION OF INTERCONNECT SIMULATION TOOLS IN SPICE

by

H. Satsangi
J. E. Schutt-Aine

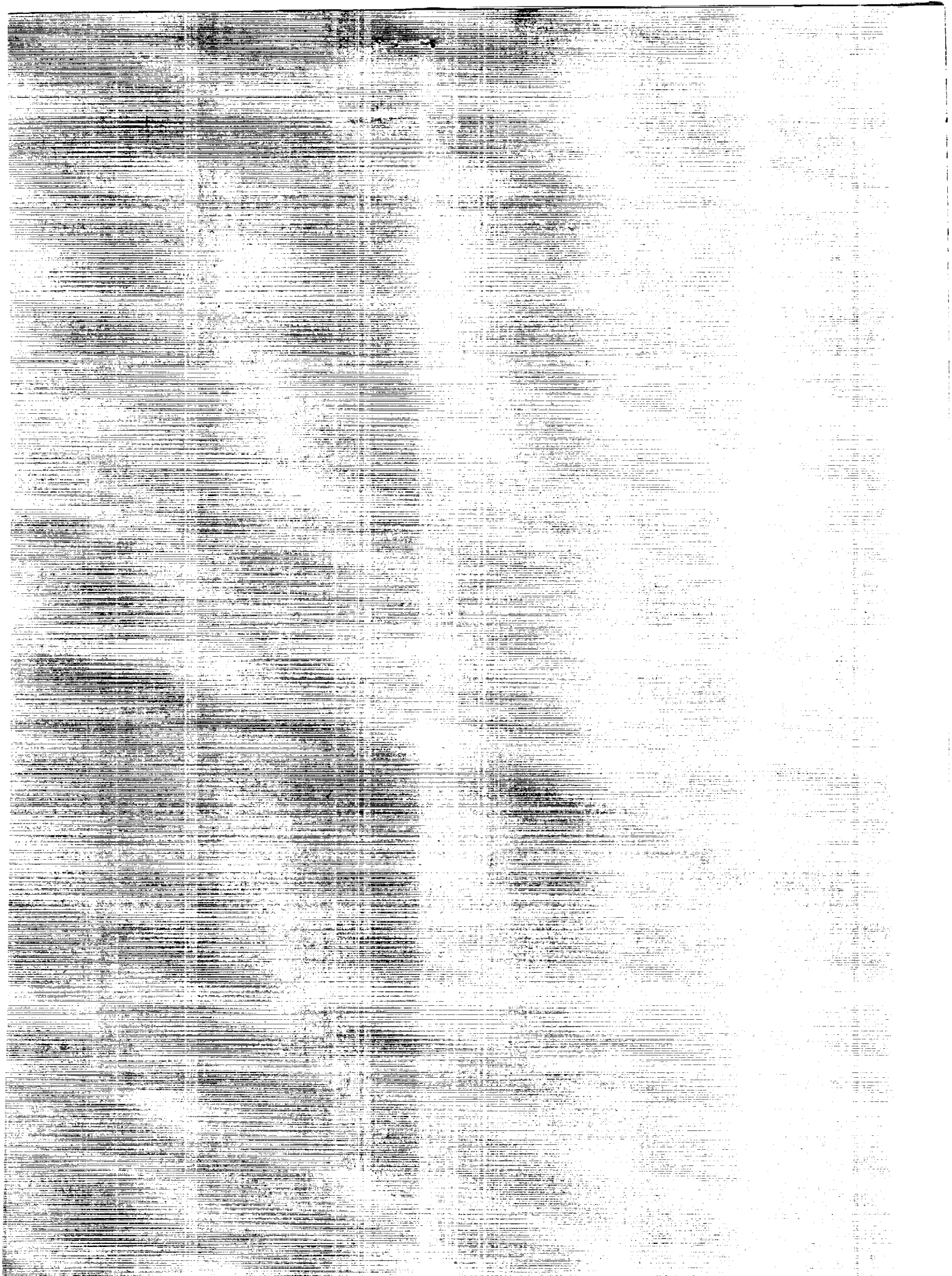
Department of Electrical and Computer Engineering
University of Illinois
1406 W. Green Street
Urbana, IL 61801

Technical Report
August 1993

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
NASA-Ames Research Center
Moffett Field, CA 94035-1000

Grant No. NASA NAG2-823



UILU-ENG-93-2559

Electromagnetic Communication Laboratory Report No. 93-4

IMPLEMENTATION OF INTERCONNECT SIMULATION TOOLS IN SPICE

by

H. Satsangi
J. E. Schutt-Aine

Department of Electrical and Computer Engineering
University of Illinois
1406 W. Green Street
Urbana, IL 61801

Technical Report
August 1993

Prepared for

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION
NASA-Ames Research Center
Moffett Field, CA 94035-1000

Grant No. NASA NAG2-823

Electromagnetic Communication Laboratory
Department of Electrical and Computer Engineering
Engineering Experiment Station
University of Illinois at Urbana-Champaign
Urbana, Illinois 61801

ABSTRACT

Accurate computer simulation of high speed digital computer circuits and communication circuits requires a multimode approach to simulate both the devices and the interconnects between devices. Classical circuit analysis algorithms (lumped parameter) are needed for circuit devices and the network formed by the interconnected devices. The interconnects, however, have to be modeled as transmission lines which incorporate electromagnetic field analysis.

An approach to writing a multimode simulator is to take an existing software package which performs either lumped parameter analysis or field analysis and add the missing type of analysis routines to the package. In this work a traditionally lumped parameter simulator, SPICE, is modified so that it will perform lossy transmission line analysis using a difference model approach.

Modifying SPICE3E2 or any other large software package is not a trivial task. An understanding of the programming conventions used, simulation software, and simulation algorithms is required. This thesis was written to clarify the procedure for installing a device into SPICE3E2. The installation of three devices is documented and the installations of the first two provide a foundation for installation of the lossy line which is the third device. The details of discussions are specific to SPICE, but the concepts will be helpful when performing installations into other circuit analysis packages.

TABLE OF CONTENTS

page

1. INTRODUCTION 1
1.1. Background 1
1.2. Purpose 2
1.3. Contents 2
1.4. SPICE3E2 Comments 3

2. GENERAL OPERATION OF A STAMP ORIENTED SIMULATOR 4
2.1. General Circuit Simulator Flow Chart 4
2.2. Circuit Description 6
2.3. Construction of Circuit Matrix 7
2.3.1. Linear resistor stamp 10
2.3.2. Independent voltage source stamp 11
2.4. Matrix Solution Techniques and Sparsity 15
2.5. Output of Results 16
2.6. Notes on Transient and Nonlinear Analysis 16
2.7. Summary 20

3. DERIVATION OF TRANSMISSION LINE STAMPS 21
3.1. Number of Device Nodes and Stamp Size 21
3.1.1. One-port devices 22
3.1.2. Two-port stamps 23
3.2. Voltage Source/Impedance Stamp for the Lossless Transmission Line 23
3.3. Current Source/Admittance Stamp for a Transmission Line 26
3.3.1. Independent current source stamp 26
3.3.2. Lossless transmission line stamp 27
3.3.3. Transient analysis of a lossy transmission line 29
3.4. Stamp Comparison 29
3.5. Summary 30

4. OVERVIEW OF ASPECTS OF SPICE3E2 RELEVANT TO DEVICE INSTALLATION 31
4.1. Organization and Conventions in SPICE3E2 Relevant to Device Installation 31
4.1.1. Packages 31
4.1.2. Interpackage communication 32
4.1.3. Package function naming conventions 33
4.1.4. Devices, models, and instances 34
4.1.5. The CKT data structure 35
4.1.6. Summary of relevant organizations and conventions 36
4.2. SPICE3E2 Directory Structure 37
4.3. Loading a Device Stamp into the Circuit Matrix in Spice3e2 39
4.3.1. Storage of instance specific data 39
4.3.2. Insertion of a device stamp into the circuit matrix 41
4.4. Loading of Device Data from the Input File 44
4.5. Summary 45

5. DEVICE INSTALLATION STRATEGY 47
5.1. General Approach 47
5.2. Specific Strategy 48
5.3. Twelve-Step Plan 50
5.4. Summary 51

6. INSTALLATION OF A NEGATIVE RESISTOR 52
6.1. Description of the Negative Resistor Stamp 52
6.2. Description of the Negative Resistor Input File Line 53
6.3. Details of the Twelve-Device Installation Steps for a Negative Resistor 54
6.3.1. Create negative resistor directory 54
6.3.2. Copy files of ordinary resistor 54
6.3.3. Change names in copied device files 55
6.3.4. Change names in copied header files 56
6.3.5. Change names in copied parser file 56
6.3.6. Modify parser header file 56
6.3.7. Modify main parsing routine 57
6.3.8. Modify simulator files 57
6.3.9. Modify files used by make 57
6.3.10. Verify establishment of communication with the main code 59
6.3.11. Modify operation of copied device code 59
6.3.12. Check new device operation 60
6.4. Summary 60

7. INSTALLATION OF A LOSSLESS TRANSMISSION LINE MODEL FOR TRANSIENT ANALYSIS 61
7.1. Lossless Transmission Line Models Revisited 61
7.2. Referencing Previous Values 62
7.2.1. The delay table 62
7.2.2. Delay table management 63
7.2.3. Interpolation 63
7.2.4. Initial conditions and the delay table 64
7.3. Voltage Source/Impedance Model vs. Current Source/Admittance Model 64
7.4. Summary of Steps 1 Through 10 of the Installation 66
7.5. Changes Comprising Step 11 68
7.5.1. Modifications to ntradefs.h 68
7.5.2. Modification to ntrasetup.c 68
7.5.3. Modifications to ntraask.c 69
7.5.4. Modification of ntraload.c 69
7.5.5. Modifications to ntraacct.c 70
7.5.6. Modifications to ntratrunc.c 70
7.6. Summary 71

8. INSTALLATION OF A LOSSY TRANSMISSION LINE MODEL FOR TRANSIENT ANALYSIS 72
8.1. Difference Between the Lossy and Lossless Line Stamps 72
8.2. Requirements of the Source Function 73
8.3. Modification Strategy 74
8.4. Conversion to Lossy Line Model Part I 74
8.4.1. Insert parameter fields into device data structure 75
8.4.2. Modify ntraitf.h 75
8.4.3. Modify ntraparam.c 75
8.4.4. Modify ntrasetup.c 76
8.4.5. Modify ntraload.c 76
8.4.6. Modify ntraacct.c 78
8.4.7. Modify ntratrunc.c 78
8.4.8. Update spice3e2/src/lib/dev/ntra directory, and modify files used by make 79
8.4.9. Verify the model gives the correct results for the specific line used 79
8.5. Conversion to Lossy Line Model Part II 79
8.5.1. Write the function fileread 80
8.5.2. Modify ntra.c to include a new parameter 80
8.5.3. Modify NTRAparam to call fileread 80
8.5.4. Modify ntradefs.h 80
8.5.5. Verify the changes worked 81

8.6. Transient Analysis Run of the Lossy Line 81
8.6.1. Formulate circuit 81
8.6.2. Place line specifications in a file 82
8.6.3. Run vdmdiff 83
8.6.4. Write SPICE input file 84
8.6.5. Perform analysis 84
8.7. Summary 85

9. CONCLUSIONS 87
9.1. Direct Current and Alternating Current Analysis 87
9.1.1. dc analysis 87
9.1.2. ac analysis 88
9.2. Modifications to Increase Manageability 88
9.2.1. Modularizing NTRAload 89
9.2.2. NTRAloadLHS 89
9.2.3. NTRAdcLoad 90
9.2.4. NTRAloadUIC 90
9.2.5. NTRAloadUdc 90
9.2.6. NTRAinitDelTab 90
9.2.7. NTRAcalcRHS 91
9.2.8. NTRAgetInterpExcit1 91
9.2.9. CKTgetSol 91
9.2.10. NTRAloadRHS 92
9.2.11. CKTgetMode 92
9.3. Summary 92

APPENDIX A. RESISTOR CODE 93
A.1. Device Specific Files 93
A.1.1. Contents of nresload.c before renaming 93
A.1.2. Contents of nresload.c after renaming 94
A.1.3. Contents of nres.c before renaming 95
A.1.4. Contents of nres.c after renaming 96
A.2. Device Header Files 97
A.2.1. nresdefs.h after renaming 97
A.2.2. nresext.h after renaming 99
A.2.3. nresitf.h after renaming 100
A.3. INP2N 102
A.3.1. Contents of inp2n.c before renaming 102
A.3.2. Contents of inp2n.c after renaming 105
A.4. Parser Header File 107
A.5. INPpas2 108
A.6. Main Parsing Routine 109

A.6.1. Contents of bconf.c after modification....109
A.6.2. Contents of subckt.c after modification....110
A.7. Files Used by Make....111
A.7.1. Contents of makedefs before modification....111
A.7.2. Contents of msc51.bat before modification....112
A.7.3. Excerpt from defaults after modification....112
A.7.4. Excerpt from response.lib after modication....113
A.8. NRESload....113
A.8.1. Excerpt from nresload.c before modification....113
A.8.2. Excerpt from nresload.c after modification....114

APPENDIX B. LOSSLESS TRANSMISSION LINE CODE....115
B.1. Device Data Structure....115
B.1.1. Contents of ntradefs.h before modification....115
B.1.2. Contents of ntradefs.h after modification....118
B.2. NTRAsetup....120
B.2.1. Contents of ntrasetup.c before modification....120
B.2.2. Contents of ntrasetup.c after modification....122
B.3. NTRAask....123
B.4. NTRAload....124
B.4.1. Contents of ntraload.c before modification....124
B.4.2. Contents of ntraload.c after modification....126
B.5. NTRAacct....129
B.5.1. Contents of ntraacct.c before modification....129
B.5.2. Contents of ntraaccct.c after modification....130
B.6. NTRAtrunc....132
B.6.1. Contents of ntratrunc.c before modification....132
B.6.2. Contents of ntratrunc.c after modification....133

APPENDIX C. LOSSY TRANSMISSION LINE CODE....135
C.1. Function Decleration and Argument Description of G....135
C.2. Contents of an Example Difference Parameters File....138
C.3. Header Files....139
C.3.1. Contents of ntradefs.h after modification....139
C.3.2. Contents of ntraitf.h....142
C.4. NTRAparam....142
C.5. NTRAsetup....143
C.6. NTRAload....147
C.6.1. Contents of ntraload.c before modification....147
C.6.2. Contents of ntraload.c after modification....150
C.7. NTRAacct....154
C.7.1. Contents of ntraacct.c before modification....155

C.7.2. Contents of ntraacct.c after modification.....155
C.8. NTRAtrunc.....156
C.8.1. Contents of ntratrunc.c before modification.....156
C.8.2. Contents of ntratrunc.c after modification.....157
C.9. Listing of the Function fileread.....158
C.10. IFparm Table.....160
C.11. NTRAparam.....161

APPENDIX D. FUTURE CODE MODIFICATIONS.....163
D.1. Complete listing of NTRAload.....163
D.2. Modified NTRAload.....167
D.3. NTRAcalcRHS.....169
D.4. NTRAgetInterpExcit1.....169
D.5. NTRAgetDelTabIndGrtr.....170

REFERENCES.....172

1. INTRODUCTION

1.1. Background

Computer simulation of circuits has been used for years to test and verify circuit designs at the gate level or transistor level. Interconnections were originally modeled as lumped parameter shorts. As device speeds became faster and integrated circuits became smaller and denser, interconnects could no longer be modeled as lumped parameter shorts. Instead, the interconnects are now modeled as transmission lines to take into account cross-talk noise, wave-form distortion, and signal attenuation effects [1].

Simulating circuits at the transistor level which contain interconnects requires that a circuit simulator have the capability of handling ordinary circuit analysis and electromagnetic analysis. Simulators which handle both types (or three types if gate level is included) are termed multimode.

One approach used in writing a multimode simulator is to utilize a standard circuit analysis package which works for one type of analysis and to add in components which perform the other analysis types. In this work, transmission line models have been inserted into a traditionally lumped parameter simulator. The insertion of devices into an existing circuit analysis package is not a trivial task. Accurate and detailed documentation of the analysis package is required along with an understanding of the general algorithm driving the circuit analysis program. An understanding of the programming conventions used in large software packages is also required.

1.2. Purpose

This thesis describes the insertion of a lossy transmission line model for transient analysis into the SPICE3E2 circuit analysis program. A lossless and lossy transmission line model already exist in SPICE3E2 (see [2] for SPICE3E2 lossy line background). A good introduction to transmission line analysis is found in [3] and much more detail is found in [4]. The concern of this thesis is not to compare the SPICE indigenous models with the models to be installed, but to detail the installation procedure for the step invariant difference model of the lossy transmission line. The thesis has two main purposes. The first is to document the modifications made to SPICE3E2 in order to incorporate a new lossy line model, and the second is to elucidate the device installation procedure with respect to SPICE3E2. The installation of a device into SPICE3E2 was more difficult than anticipated since the documentation which comes standard with SPICE3E2 (see [5] - [11]) is not up to date with the release and assumes that the reader has a level of familiarity with the functioning of a circuit simulator code. The documentation which comes standard with SPICE3E2 is the documentation for SPICE3C, and in the device installation section there is an omission of one important step.

This thesis can by itself be used as a "how to guide" with respect to device installation. It is, however, not a replacement for the SPICE3E2 manuals in any other way, but can be used in conjunction with the SPICE3E2 manuals to gain a better understanding of the functioning of the SPICE3E2 circuit simulator.

1.3. Contents

Two purposes of the thesis were mentioned in the previous section. The second purpose was to serve as a device installation tutorial with respect to SPICE3E2. To this end the thesis is organized in the following fashion: circuit simulator information and device installation information. Chapter 2 contains general information on circuit analysis programs such as the general circuit analysis program algorithm and techniques for implementing each step of the algorithm. Chapter 3 is an extension of Chapter 2 and is specifically concerned with transmission lines. Chapter 4 discusses the SPICE3E2 circuit analysis program and, in particular, the background information necessary with respect to device installation. Chapter 5 gives an overview of the device installation process, and

Chapters 6 through 8 are step by step examples of device installations. Chapter 6 details the installation procedure for a contrived device to illustrate the bookkeeping portion common to all device installations. Chapter 7 lays the foundation for the lossy line by detailing the installation steps for a lossless transmission line model for transient analysis. Chapter 8 details the installation of the lossy transmission line model for transient analysis into SPICE3E2 by converting the routines of the lossless transmission line installed in Chapter 7. Chapter 9 is the concluding chapter and it discusses and suggests some extensions and modifications to SPICE3E2, which would increase the manageability of the code.

1.4. SPICE3E2 Comments

The source code for SPICE3E2 is available from UC Berkeley free of charge. The program is written in the C programming language [12]. In order to make modifications to the source code, the programmer must be proficient in the C language. The source was modified when installing the lossy transmission line. Listings of the source code involved in or modified during device installation are given in the appendices. To conserve space, only relevant portions of the code listings are shown, and in many cases the header files associated with a function were not shown. In all cases, a copyright statement associated with each code listing was not shown. Most of the code that will be shown was originally written by T. Quarles and the copyright statement associated with the code is shown below.

```
/**********
Copyright 1990 Regents of the University of California.  All rights reserved.
Author: 1985 Thomas L. Quarles
**********/
```

2. GENERAL OPERATION OF A STAMP ORIENTED SIMULATOR

This chapter discusses the basics of circuit simulator operation. The general algorithm and each of its components are examined. The discussion is within the framework of linear device analysis and confined to resistors and independent voltage sources. The examples in this chapter apply to SPICE3E2, but the concepts are general. Implementation details and low level algorithms have been deferred to Chapters 4, 6, 7, and 8.

2.1. General Circuit Simulator Flow Chart

A general flow chart indicating the major steps in the execution of a circuit analysis program is shown in Figure 2.1. First, the circuit and simulation descriptions are obtained from user input. The information from the user is organized in a form most suitable for the steps to follow. Once the input information is appropriately organized, a matrix equation

$$\mathbf{A}\mathbf{x} = \mathbf{b}, \tag{2.1}$$

where A is an n x n matrix, is constructed. The matrix equation is solved for **x**, and the results are made available. Depending on the type of analysis, steps Construct Circuit Matrix and Solve Matrix may be executed multiple times before all or any of the results are available.

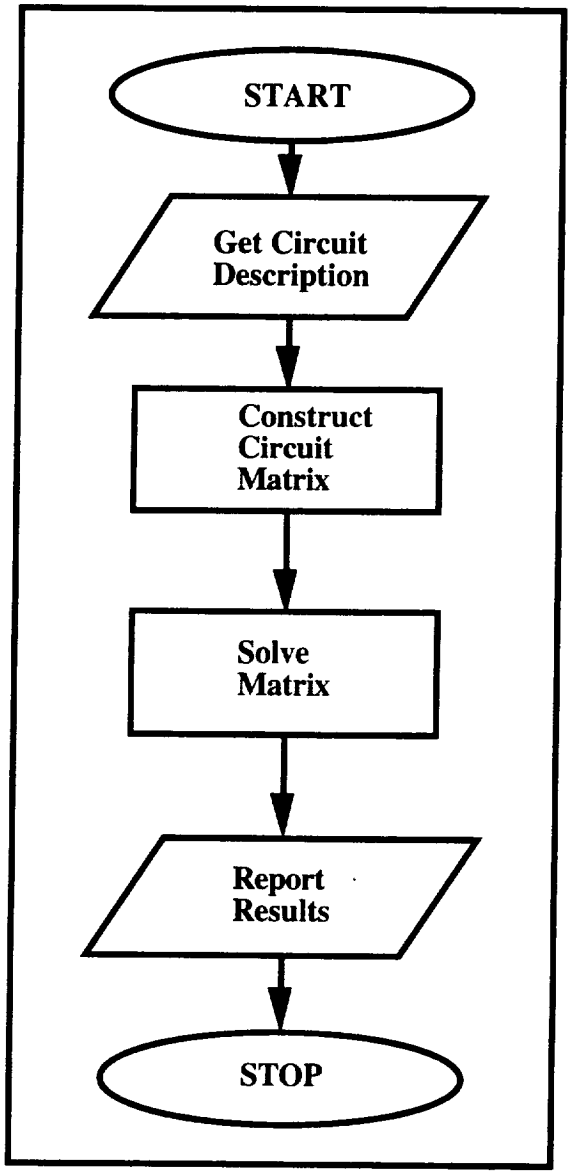


Figure 2.1. Circuit analysis algorithm in flow chart form.

The details of the algorithms and programming differ from simulator to simulator, but the overall steps are the same. In the following sections, each of the steps is examined, and strategies or algorithms are presented. The most important step with respect to this thesis is the second, Construct Circuit Matrix. The modifications that will be discussed in the subsequent chapters primarily concern this second step.

2.2. Circuit Description

Analysis of a circuit commences after the circuit and simulation information are provided by the user. User input is often obtained through a graphical interface. Alternatively, a text file may be filled by the user, listing the devices, device parameters, interconnections of the devices, and simulation specifications. Regardless of the manner used to communicate circuit and simulation descriptions, the information is stored in simulator data structures for later referral by the matrix construction circuit simulation routines. The format of the data structure containing the circuit information is the subject of this section.

Data structure formats are dictated by the algorithms and routines accessing the structure. Matrix construction algorithms, discussed further in the following section, access and, therefore, define the format of the circuit description structures.

Equations for a circuit may be written using a node by node application of Kirchoff's Current Law (KCL). A node by node approach results in a data structure comprised of a linked list containing elements associated with each node. Among the fields of the list element would be fields containing node identification information, fields associated with all elements branching from the node, and fields containing neighbor node information. Considering a device connected between nodes a and b, the node by node algorithm references the device once at node a and once at node b. Therefore, the matrix fill routines must be accessed twice for each device in the circuit.

A more popular algorithm for filling a circuit matrix accesses the matrix fill routines only once per device. This method employs a device by device approach, and is discussed in more detail in the following section. The device by device routine requires a linked list similar to that shown in Figure 2.2. In the data structure of Figure 2.2 link elements include the device identification, a pointer to a device parameter list, a pointer to a list of nodes connected to the device, a forward pointer, and a backward pointer as a minimum. Other fields, needed for overhead, are not shown.

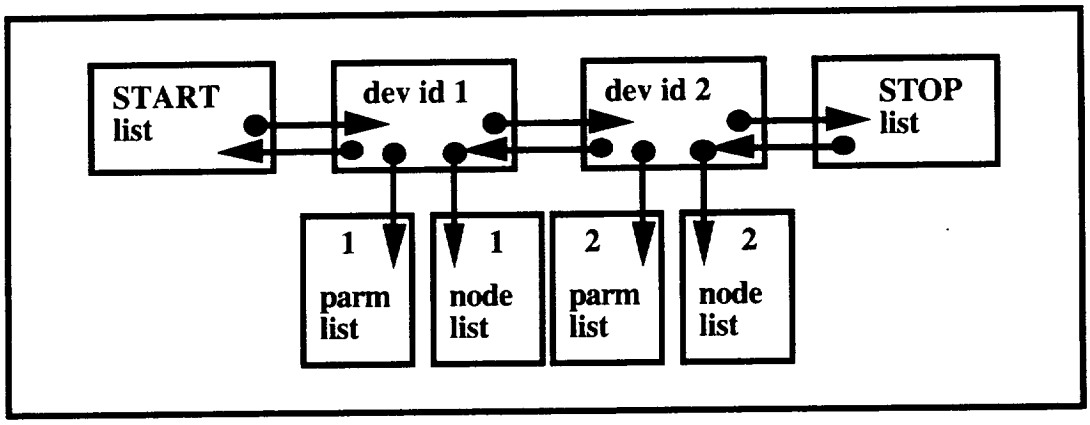


Figure 2.2. Linked list structure used with device by device matrix fill algorithm for storing circuit description.

2.3. Construction of Circuit Matrix

Equations describing a circuit can be written by employing KCL or Kirchoff's Voltage Law (KVL) analysis [13]. KCL is used in circuit simulators. Nodal analysis of the circuit given in Figure 2.3 yields Equations (2.2) - (2.9).

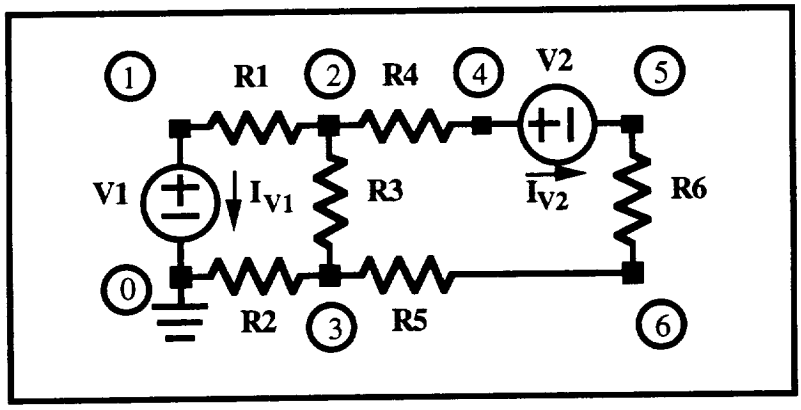


Figure 2.3. Example Circuit 1. Nodes are labeled by encircled numbers.

$$\frac{V_1 - V_2}{R1} + I_{V1} = 0 \tag{2.2}$$

$$\frac{V_2 - V_1}{R1} + \frac{V_2 - V_3}{R3} + \frac{V_2 - V_4}{R4} = 0 \tag{2.3}$$

$$\frac{V_3 - V_2}{R3} + \frac{V_3}{R2} + \frac{V_3 - V_6}{R5} = 0 \tag{2.4}$$

$$\frac{V_4 - V_2}{R4} + I_{V2} = 0 \tag{2.5}$$

$$\frac{V_5 - V_6}{R6} - I_{V2} = 0 \tag{2.6}$$

$$\frac{V_6 - V_5}{R6} + \frac{V_6 - V_3}{R5} = 0 \tag{2.7}$$

$$V_1 = V1 \tag{2.8}$$

$$V_4 - V_5 = V2 \tag{2.9}$$

Equations (2.2) - (2.9) can be rewritten in the following form:

$$\left(\frac{1}{R1}\right)V_1 + \left(\frac{-1}{R2}\right)V_2 + (1)I_{V1} = 0 \tag{2.10}$$

$$\left(\frac{1}{R1} + \frac{1}{R3} + \frac{1}{R4}\right)V_2 + \left(\frac{-1}{R1}\right)V_1 + \left(\frac{-1}{R3}\right)V_3 + \left(\frac{-1}{R4}\right)V_4 = 0 \tag{2.11}$$

$$\left(\frac{1}{R3} + \frac{1}{R2} + \frac{1}{R5}\right)V_3 + \left(\frac{-1}{R3}\right)V_2 + \left(\frac{-1}{R5}\right)V_6 = 0 \tag{2.12}$$

$$\left(\frac{1}{R4}\right)V_4 + \left(\frac{-1}{R4}\right)V_2 + (1)I_{V2} = 0 \tag{2.13}$$

$$\left(\frac{1}{R6}\right)V_5 + \left(\frac{-1}{R6}\right)V_6 + (-1)I_{V2} = 0 \tag{2.14}$$

$$\left(\frac{1}{R6}+\frac{1}{R5}\right)V_6+\left(\frac{-1}{R6}\right)V_5+\left(\frac{-1}{R5}\right)V_3=0 \tag{2.15}$$

$$(1)V_1 = V1 \tag{2.16}$$

$$(1)V_4+(-1)V_5 = V2 \tag{2.17}$$

The equations can be solved systematically by putting them into a matrix of the form of Equation (2.1) with **x** = **v**, and **b** = **s**, where **v** is the voltage and current vector, or the values of the voltages at all the different nodes and currents in particular branches, and **s** is the source vector or the values of independent sources in the circuit.

The resulting matrix is shown in Figure 2.4. This matrix can be solved for the node voltages V_1 to V_6 and the branch currents I_{V1} and I_{V2}. Any other branch currents can be obtained by using the node voltages and component values. All of which are known after solving the matrix.

$$\begin{bmatrix}
\left(\frac{1}{R1}\right) & \left(\frac{-1}{R2}\right) & & & & & (1) & \\
\left(\frac{-1}{R1}\right) & \left(\frac{1}{R1}+\frac{1}{R3}+\frac{1}{R4}\right) & \left(\frac{-1}{R3}\right) & \left(\frac{-1}{R4}\right) & & & & \\
 & \left(\frac{-1}{R3}\right) & \left(\frac{1}{R3}+\frac{1}{R2}+\frac{1}{R5}\right) & & & \left(\frac{-1}{R5}\right) & & \\
 & \left(\frac{-1}{R4}\right) & & \left(\frac{1}{R4}\right) & & & (1) & \\
 & & & & \left(\frac{1}{R6}\right) & \left(\frac{-1}{R6}\right) & (-1) & \\
 & & \left(\frac{-1}{R5}\right) & & \left(\frac{-1}{R6}\right) & \left(\frac{1}{R5}+\frac{1}{R6}\right) & & \\
(1) & & & & & & & \\
 & & & (1) & (-1) & & &
\end{bmatrix}
\begin{bmatrix} V_1 \\ V_2 \\ V_3 \\ V_4 \\ V_5 \\ V_6 \\ I_{V1} \\ I_{V2} \end{bmatrix}
=
\begin{bmatrix} \\ \\ \\ \\ \\ \\ V1 \\ V2 \end{bmatrix}$$

Figure 2.4. Matrix for Example Circuit 1.

In a circuit analysis program the construction of a matrix, such as shown in Figure 2.4, follows the conversion of user input into a circuit representation (ref. Figure 2.1). A popular method to construct the matrix employed by circuit simulators, including SPICE, takes advantage of the fact that each device in the circuit makes an independent contribution to the circuit matrix. The pattern representing this contribution is termed a stamp.

2.3.1. Linear resistor stamp

A linear resistor contributes to the circuit equations in the same manner for dc, ac, and transient analysis. The stamp derived in this section is valid for all linear resistors obeying Equation (2.18) [14].

$$v = iR \tag{2.18}$$

Refer to nodes 2, 3, and the resistor R3 between the two nodes from the circuit of Figure 2.3. It is seen from the matrix (Figure 2.4) that R3 is present only in the rows corresponding to nodes 2 and 3. R3 is not included in the KCL equation at nodes other than 2 and 3 since it is connected to only nodes 2 and 3 and is not a control for a dependent device. KCL at a node takes the form of Equation (2.19).

$$i_1 + i_2 + i_3 + \ldots + i_n = 0 \tag{2.19}$$

The numbers 1, 2, 3, and n are indices for the branches connected to the node of interest. All of the currents are assumed to be going out of the node. At node 2, i_1 will be assigned the current going through R1, i_2 the current going through R4, and i_3 the current going through R3. The equation for current i_3 is shown below, assuming resistor R3 has value R3.

$$i_3 = \frac{V_2 - V_3}{R3} \tag{2.20}$$

The branch current, i_3, is the only expression involving R3 in the KCL equation written at node 2. Applying KCL at node 3 with i_3 assigned to the current flowing through R3, and out of node 3, as opposed to out of node 2, yields

$$i_3 = \frac{V_3 - V_2}{R3}. \tag{2.21}$$

Equations (2.20) and (2.21) are the only two expressions including R3. Equation (2.20) is part of the KCL at node 2, and (2.21) part of the KCL at node 3. Therefore, R3 affects the KCL equations at nodes 2 and 3 only, as shown by the pattern or stamp in Figure 2.5.

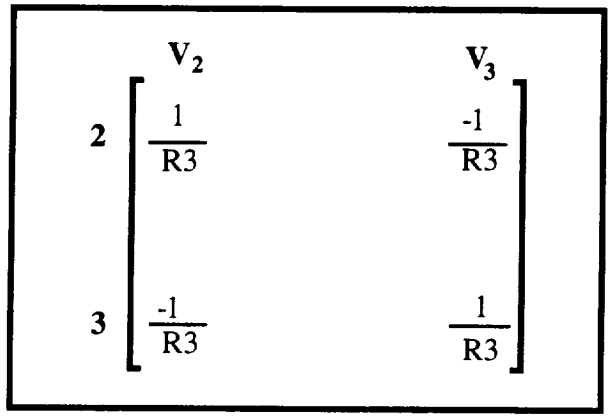


Figure 2.5. Stamp for resistor R3 from Example Circuit 1.

Examining the circuit matrix (Figure 2.4) will show that the rest of row 2 can be constructed by superposing the stamps for the remaining resistors attached to node 2. The stamp for R1 looks like Figure 2.5 except the rows involved are 2 and 1, the columns are V_2 and V_1 and the resistor value is R2. Similarly, resistor R4 has a stamp of the form shown in Figure 2.5 except that all of the 3's are replaced by 4's. The stamps for R1 and R4 also affect rows 1 and 4. A general stamp for a resistor of value R is shown in Figure 2.6. In the figure, a and b refer to arbitrary rows which are not necessarily adjacent.

2.3.2. Independent voltage source stamp

The stamp for an independent voltage source is derived in this section. The derived stamp is valid for dc, ac, and transient analyses. When writing KCL, the variables are usually the voltages at the nodes used in expressions for the currents at the node of interest, and equations are written at every node. This approach produces n equations, one at each of the n nodes, with n unknowns, the n unknown node voltages. In the case of voltage

sources, the current through the voltage source is not expressible in terms of node voltages and instead must be declared as a variable.

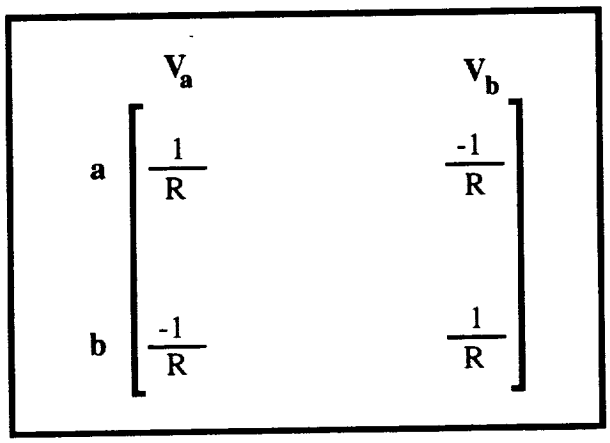


Figure 2.6. General stamp for a linear resistor of value R connected between two nodes a and b.

Examine the voltage source V2 connected between nodes 4 and 5 of the circuit in Figure 2.3. When writing KCL at node 4, the current through the voltage source is written as I_{v2}. This is seen as +1 at matrix position (4, I_{v2}) in Figure 2.7. When considered from node 5, the current is $-I_{v2}$. This produces the -1 at matrix position (5, I_{v2}). The current through the voltage source contributes an extra variable to the matrix. The system of equations becomes n equations in n+1 unknowns. Equation n+1 is obtained by using the value of the voltage source. The equation is

$$V_4 - V_5 = V2. \tag{2.22}$$

Equation (2.22) appears as a +1 and a -1 at matrix positions (V2, V4) and (V2, V5), respectively, and a V2 at matrix position (V2, sources). Therefore, the pattern that voltage source V2 contributes is shown in Figure 2.7. The general stamp of an independent voltage source is shown in Figure 2.8. where V corresponds to the value of the voltage source which is connected between nodes a and b, with a as the positive node.

The circuit matrix of Figure 2.4 can be constructed by combining the stamps of the various resistors and voltage sources involved. Whenever an element of a stamp occupies the same position in the matrix as an element of another stamp, they are combined by

addition. Notice that for R2 and V1 the section of the stamp relating to node 0 (row 0 and column V_0) is not present in the matrix, because R2 and V1 are connections to ground. The voltage at node 0 is already known to be zero and the rows and columns associated with node 0 can be eliminated.

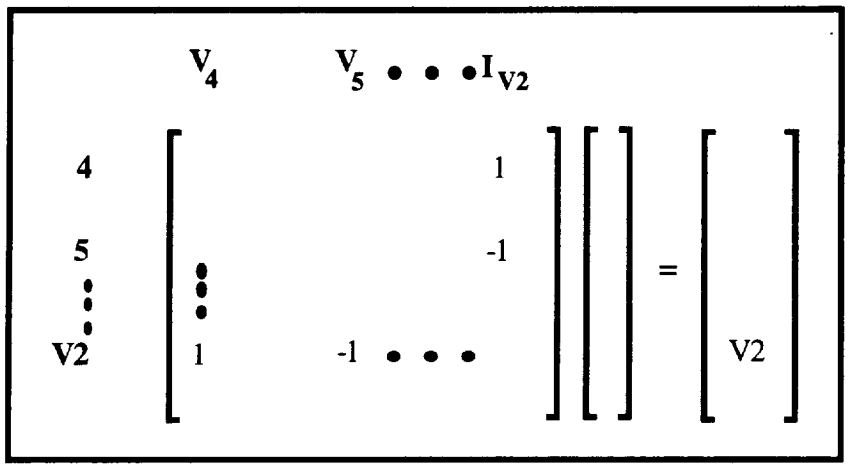


Figure 2.7. Specific voltage source stamp for V2 of Example Circuit 1.

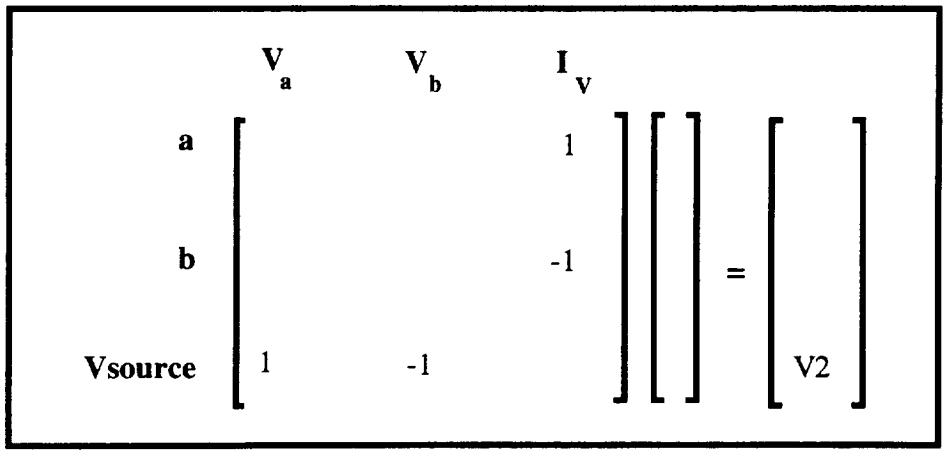


Figure 2.8. General stamp for a voltage source connected between nodes a and b, with a as the positive node.

The basic algorithm employing stamps for filling a matrix is shown as a flow chart in Figure 2.9. A problem may seem to exist with respect to the stamp approach since the matrix size is not known a-priori. The capability in particular programming languages to dynamically allocate memory allows the matrix to start out with a size of zero and grow

dynamically in cell size or have existing entries appended with the insertion of each device stamp. Before entering the stamp of a device into the circuit matrix the row and column entries that the device will affect are checked for previous allocation. If space in the matrix has already been allocated for an entry, then the stamp information is added to the entry. If space has not been allocated in the matrix, then space is allocated for the entry and the stamp information is used to fill the entry.

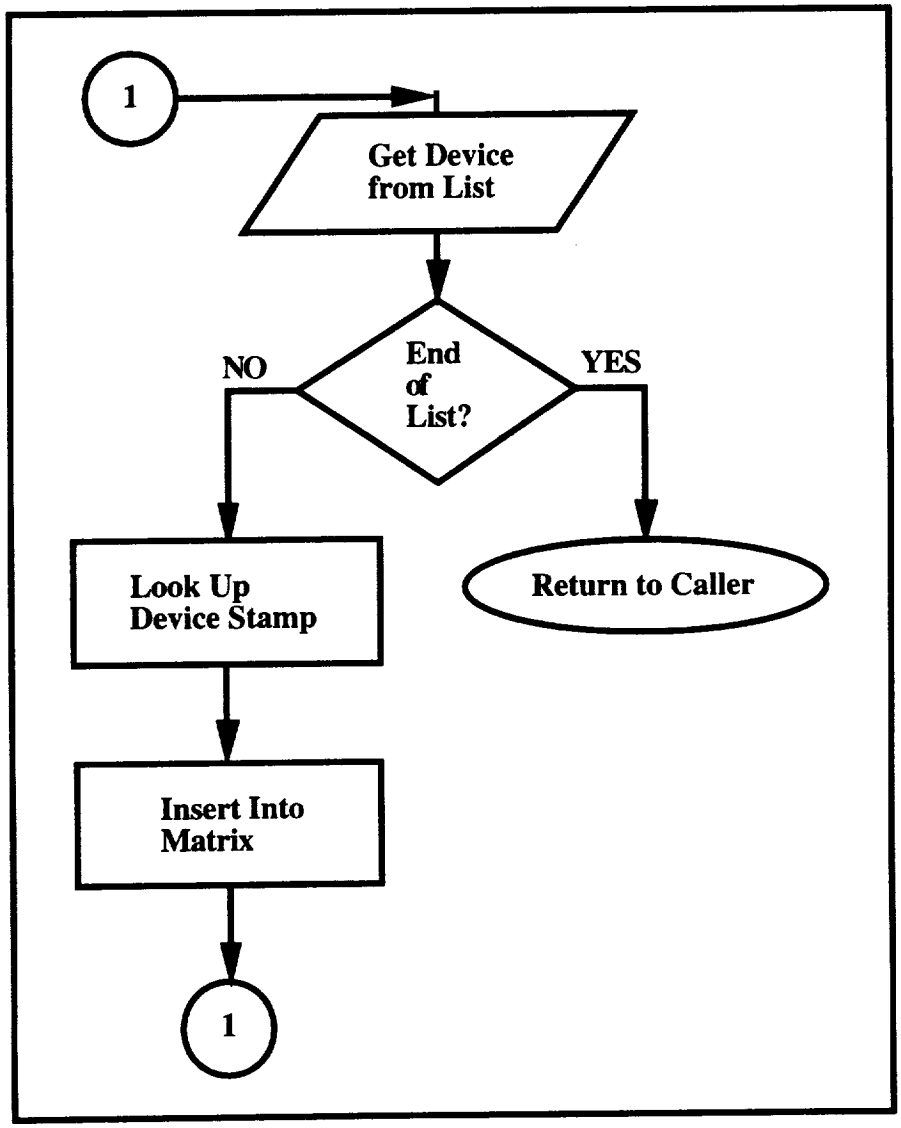


Figure 2.9. Flow chart for device by device matrix fill algorithm.

2.4. Matrix Solution Techniques and Sparsity

Many techniques, both direct and iterative, are available for solving matrices. Whether a specific circuit matrix is solved just once, the direct technique, or is solved multiple times until the solution is deemed valid, the iterative technique, depends on the type of analysis desired. Direct techniques are used for operating point analysis or dc analysis. Iterative techniques are used for nonlinear device analysis. In transient analysis (see Section 2.6) the matrix is solved either directly or iteratively at each time step, depending on the type of devices in the circuit.

The direct matrix solving technique of Gaussian elimination, or a permutation of Gaussian elimination, such as LU factorization, is often used [14]. The details and complexity of matrix solving routines vary with the structure of the matrix. The characteristic of matrix sparsity is discussed in the remainder of this section.

The majority of the entries in the matrix of Figure 2.4 are zeroes. The characteristic of having many empty or zero entries in a matrix is termed sparsity. Special matrix solving techniques exist to take advantage of and retain sparsity of a matrix during the solution process [15].

Rows of a circuit matrix represent nodes at which KCL or device equations are being written, and each of the columns represent either a node voltage or a branch current involved. The number of entries per row related to a node is proportional to the number of branches emanating from the node. For circuits with many elements, where the nodes are not highly interconnected and the device equations involve few variables, the matrix representing the circuit will be sparse. Many simulators, including SPICE3E2, have a sparse matrix package which exploits and perpetuates the sparsity of the matrix.

In contrast to dense matrices, ordinarily stored in two-dimensional arrays, linked list structures are employed for sparse matrices. There is no need to waste memory on storing an entry of zero. When referencing the structure, if a list element corresponding to a particular matrix location does not exist, then the matrix entry is assumed to be zero.

2.5. Output of Results

Once the matrix has been solved, the results are available in output data structures. Routines which interface from the simulator to the output package access the data structure and present the simulation results to the output package structures.

2.6. Notes on Transient and Nonlinear Analysis

The details of the third step of Figure 2.1, Solve Matrix, vary with different types of analysis. The general flow chart for a transient analysis is shown in Figures 2.10 and 2.11. At the beginning of transient analysis, the current through inductors and transmission lines, and the voltage across capacitors and p/n junctions must be known. If this initial condition or operating point data is not specified by the user, a dc analysis is performed to obtain the state of the circuit before transient analysis proceeds.

Transient analysis starts at an initial time *Time_start* and finishes at a time *Time_stop*. In the figures, *Time* is the variable keeping track of the simulation time. The matrix is set up and solved at a finite number of time points in between the start and stop times, therefore, discretizing the continuum between Time_start and Time_stop. Consecutive time points at which the solution is calculated are separated by fixed or variable time steps. In the fixed scheme the time step is set a-priori and does not change. In the variable method the appropriate step is calculated at the present time point t_i to reach the next time point t_{i+1}. The significance of time step size will be discussed later in this section.

Once Time increases beyond Time_stop the simulation ends. At times less than Time_stop, Increment Time appropriately updates the simulation time by either the fixed or variable methods, and control is passed to Fill Matrix. The process is repeated until completion. The first step of Figure 2.11 is Fill Matrix, with the subheading Update Stamps. The stamps presented in Section 2.3 were uncomplicated, depending only on known device parameters which remained constant. In contrast, many devices have stamps which depend on parameters which change from time point to time point and, hence, rec-

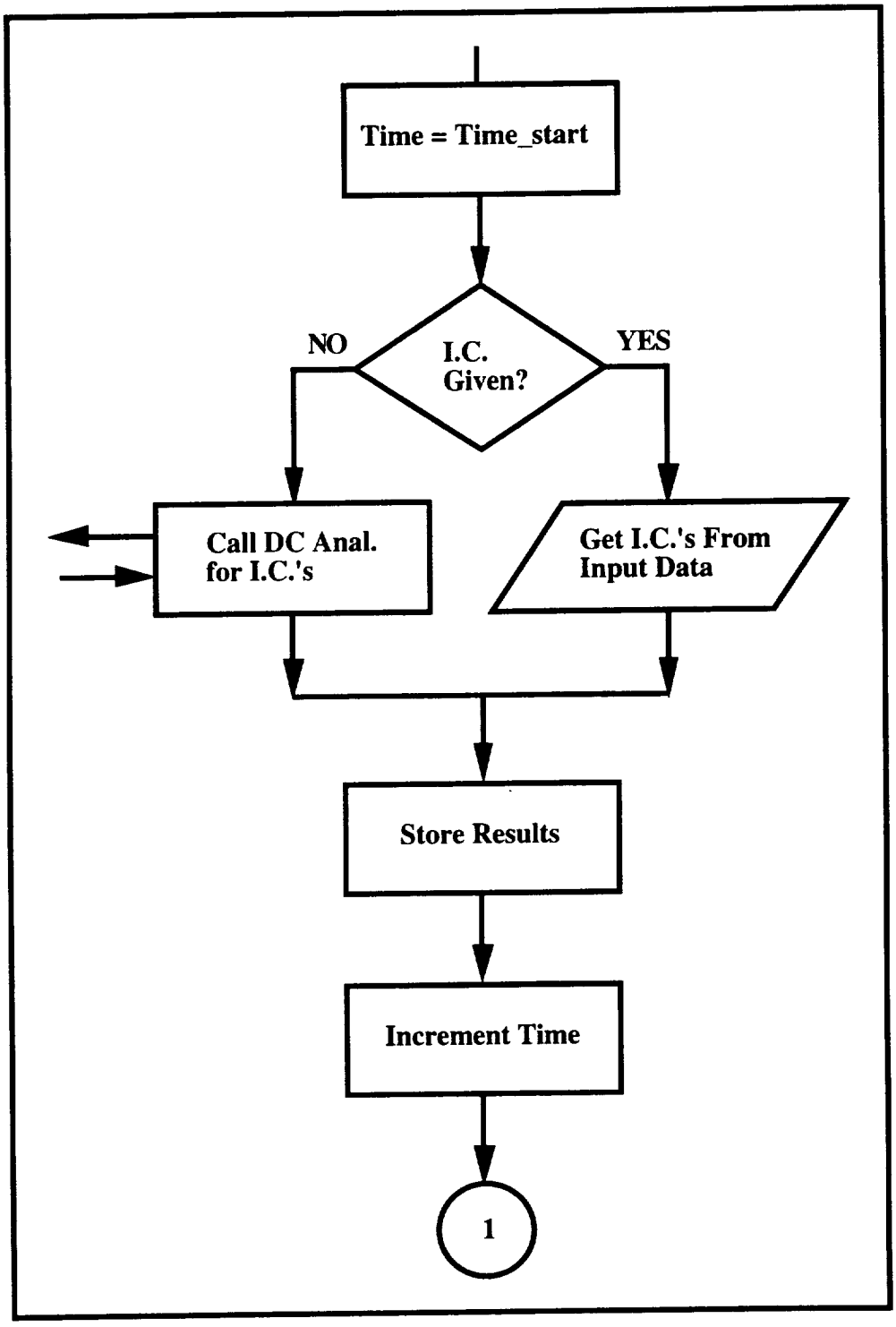


Figure 2.10. Flow chart for transient analysis, part I.

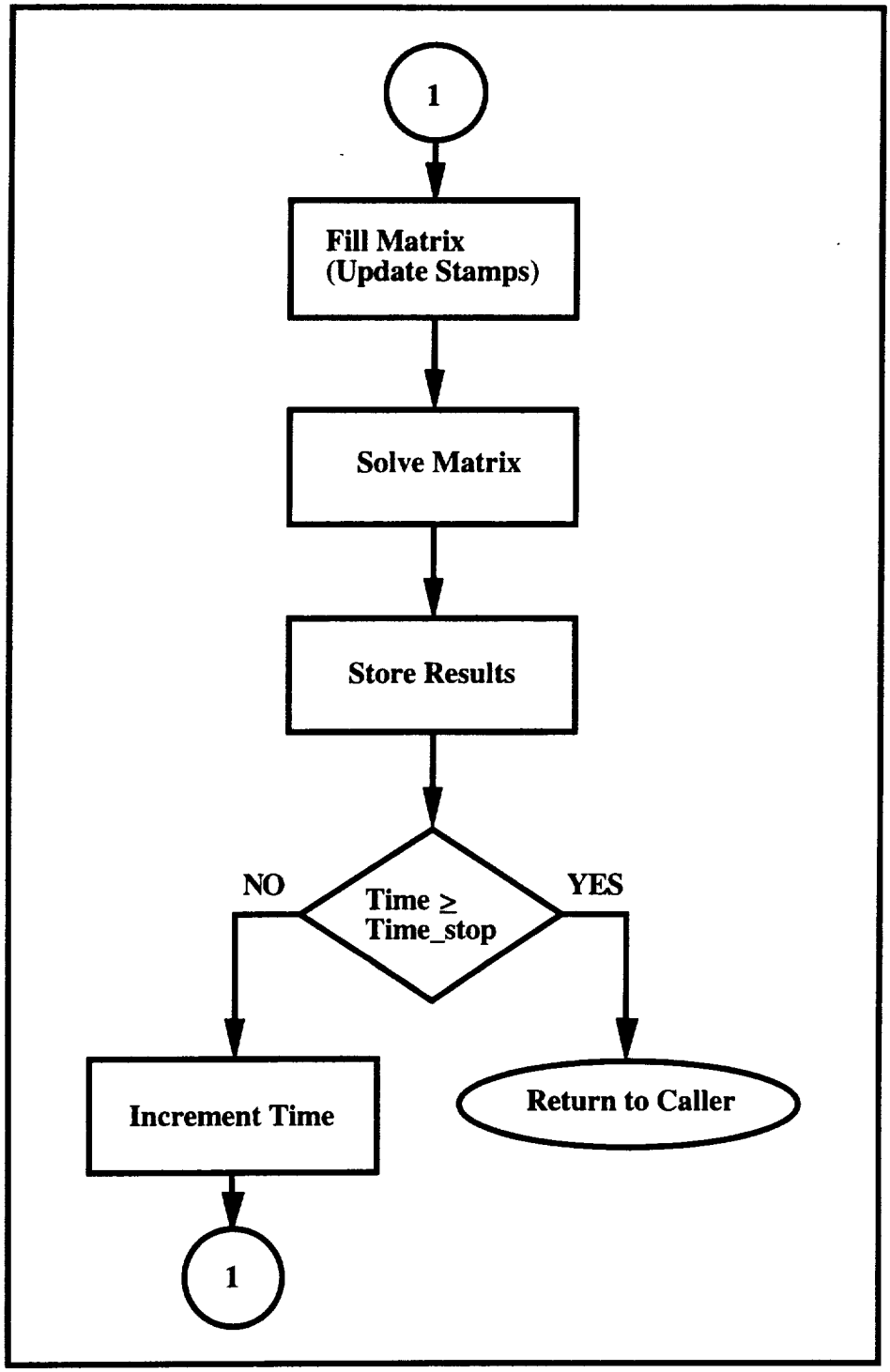


Figure 2.11. Flow chart for transient analysis, part II.

uire updating before each matrix fill. For example, a linear capacitor has a transient analysis stamp which requires information about the voltage across it at time t_{i-1} for a matrix fill at time t_i. At the first iteration of Figure 2.11, the stamp is constructed for the capacitor based on the initial condition values. Once the matrix is solved, the voltages at the nodes of the capacitor and, therefore, the voltage across the capacitor are available and will be used in computing the capacitor stamp at the next time step.

The transient analysis formulae for devices are a result of the discretization of the differential equation representing the voltage/current relationship for the device. The resulting formulae are referred to as linear multistep formulae [17], which have stability requirements, dictating the time step size. The requirements change with the activity of the circuit. Larger time steps can be taken when the voltage or current are not changing rapidly in the circuit. Smaller steps are needed when the circuit is rapidly changing. Variable time steps have the advantage of reducing the number of matrix evaluations by utilizing larger time steps where possible. Variable time steps, however, are more difficult to program and entail extra overhead in the main code. In contrast, fixed time steps require much less overhead and are significantly easier to program, but lead to unnecessary extra matrix evaluations. See [6], [7], and [14] - [17] for details.

This section has centered on linear devices which require the matrix to be solved once at a particular time point. If any of the devices are nonlinear the Solve Matrix step of Figure 2.11 is not a simple matrix evaluation, but multiple evaluations. See [15] for further discussion.

2.7. Summary

A circuit simulator has four basic operations: obtain the circuit description, construct the circuit matrix, solve the matrix, and report the results. Operations two and three may be executed multiple times depending on the type of analysis. Once the circuit description is defined by the user and stored in linked list data structures by the simulator, the matrix is constructed. A popular method of matrix construction in circuit simulators is the stamp method, in which the matrix is filled device by device based on the contributions each device makes to a circuit matrix. The stamp for a resistor connected between two nodes a and b is shown in Figure 2.6. Once the matrix is filled, LU factorization can be used to solve the matrix. Transient analysis occurs over an interval. The matrix is constructed and solved at a finite number of time points in the interval. Some stamps may depend on values from a previous time point. This requires updating stamps before filling a matrix. More information on computer analysis of circuits is found in [14] - [17].

3. DERIVATION OF TRANSMISSION LINE STAMPS

Two stamps used in transient analysis of a lossless transmission line are examined. Section 3.1 supplies the background on the number of device nodes vs. stamp size to set the foundation for comparing the two stamps. Sections 3.2 and 3.3 describe the stamps, and Section 3.4 compares them.

3.1. Number of Device Nodes and Stamp Size

In this section the relationship between device nodes and stamp size is examined. Every circuit element has an associated model. Complex devices have models comprised of the models of simpler devices. All models have external nodes, the nodes the device shares with the rest of the circuit, and possible internal nodes and branch currents which only the device uses (if there are no other circuit elements being controlled by the internal values). Internal nodes and branch currents are a result of the joining of simple models to construct a more complex model. There is an important difference with respect to stamp size between external nodes and internal nodes and internal branch currents.

Stamp size depends on both external and internal nodes and internal currents. Larger stamps result in larger circuit matrices, which take longer to solve, and more memory to store. It is desirable to make device stamps as small as possible to obtain smaller circuit matrices resulting in shorter solution times and better memory utilization.

Stamps can be made smaller by eliminating rows and columns. Row and column elimination translates into eliminating variables (nodes or currents). It is impossible to eliminate the rows and columns related to the external nodes since this would disconnect a device from the circuit. The only choice is eliminating internal nodes or currents. This can sometimes be done by rewriting a device model, and will be demonstrated in Section 3.3.

3.1.1. One-port devices

A one-port device (see Figure 3.1) has two nodes that connect it with the outside circuit. A stamp for such a device will contain at least a row and column for each node. Therefore, for a one-port device the stamp is at least 2x2. This is the case for the resistor (see Figure 2.6). The voltage source (see Figure 2.8) required an extra row and column making the matrix 3x3.

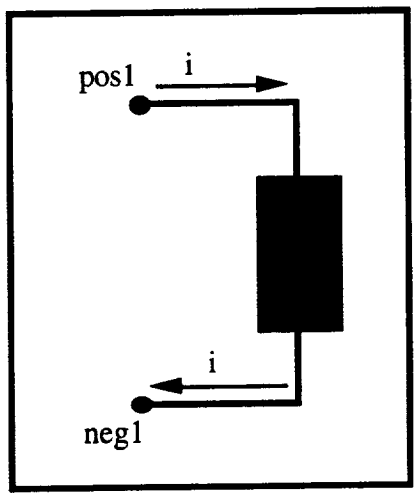


Figure 3.1. One-port device.

If there are no controlled devices depending on the current through the voltage source, the third row (Vsource) and column I_V in Figure 2.8 will not be used in the overall circuit matrix by other devices. The third row and column are necessary to handle the current through the voltage source only. The first two rows and columns in Figure 2.8, which relate to the external nodes of the device, are used in the overall circuit matrix by the other devices. The voltage source will share the external nodes as points of connection in the circuit (see Figure 2.4). Therefore, a voltage source will share two rows and columns with other devices, and will add an extra row and column to the circuit matrix for exclusive use.

In the remainder of this thesis the rows and columns in a stamp representing the internal nodes and internal currents of a device will be referred to as extra or added on, because the rows and columns are only used by the device to which they are internal, and

increase the stamp size for the device beyond the size indicated by the external nodes of the device.

3.1.2. Two-port stamps

A diagram representing a two-port is shown in Figure 3.2. A two-port has four external nodes, and the stamp will be at least 4x4.

Depending on the specific two-port device there may be internal nodes or branch currents that will increase the stamp size beyond 4x4. Assuming no controlled elements these internal nodes and currents will not be used by any other device of the overall circuit.

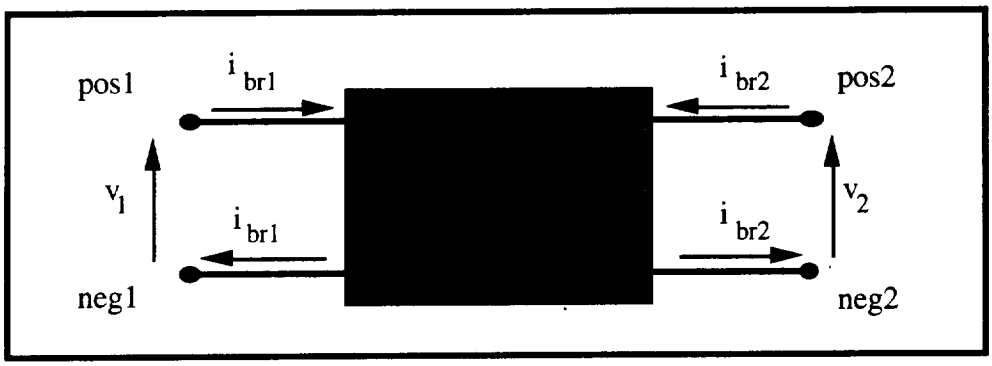


Figure 3.2. Two-port device.

3.2. Voltage Source/Impedance Stamp for the Lossless Transmission Line

A valid model for a lossless transmission line is shown in Figure 3.3 and Equations (3.1) and (3.2). In the equations τ is the transmission line delay. This is the time for a signal to travel once from one end of the line to the other. The voltage source/impedance model is used in the lossless transmission line module that comes with SPICE3E2.. The nodes in the figure have been named using the node naming convention employed in SPICE3E2. All subsequent circuit diagrams and stamps presented will utilize this naming convention [5].

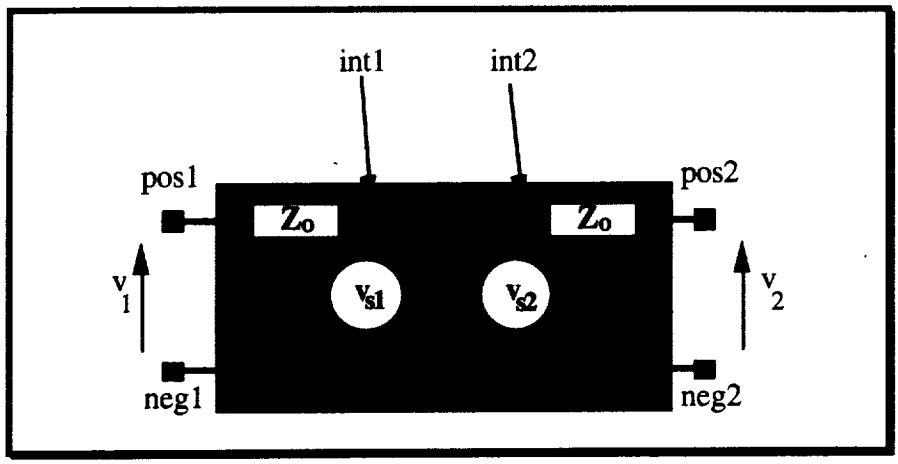


Figure 3.3. Voltage source/impedance model.

$$v_{s1}(t) = 2v_2(t-\tau) - v_{s2}(t-\tau) \tag{3.1}$$

$$v_{s2}(t) = 2v_1(t-\tau) - v_{s1}(t-\tau) \tag{3.2}$$

In Chapter 2 it was pointed out that the matrix in Figure 2.4 can be constructed by superposing stamps for the resistors and the voltage sources. Similarly, impedance or admittance stamps and voltage source stamps can be used to construct the stamp of the model shown in Figure 3.3. The stamp of the Figure 3.3 schematic is shown in Figure 3.4.

Note that neg1 and neg2 should not be assumed to be grounded. Also, Equations (3.1) and (3.2) have no bearing on the form of the stamp, only on the values of V_{s1} and V_{s2} which appear on the right-hand side of the equality. The independent voltage source equation is

$$v_{int1} - v_{neg1} = v_{s1} \tag{3.3}$$

This is the same as Equation (2.22) with the nodes and source renamed. Equation (3.3) is useful when deriving the form of the stamp. Equations (3.1) and (3.2) are useful when filling the stamp with numerical values.

The row and column naming in Figure 3.4 are consistent with SPICE3E2; therefore, the row label ibr1 still marks a voltage source equation of the form of Equation

(3.3). This stamp was formed using Y_o; the admittance is used as opposed to $1/Z_o$ as was done in Chapter 2. Y_o is used since in the SPICE3E2 code that will be discussed in Chapter 4, and Chapters 6 through 9, the stamps are filled using admittance values as opposed to the reciprocal of the impedance.

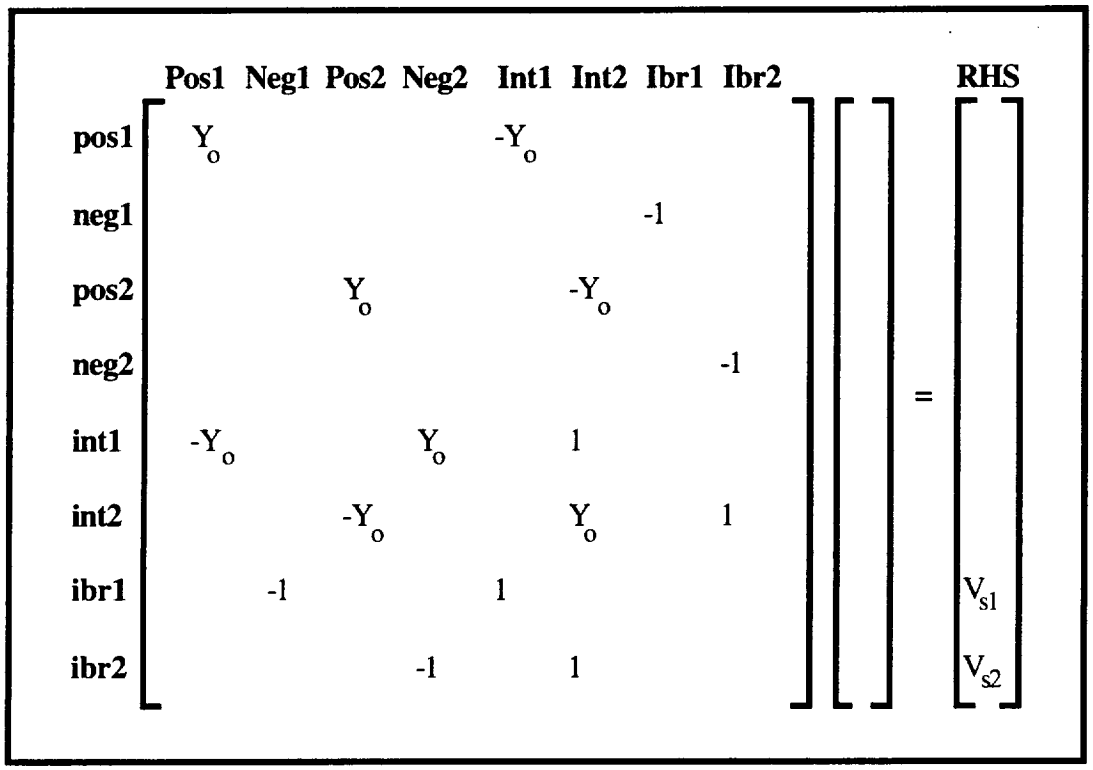


Figure 3.4. Voltage source/impedance stamp for a lossless transmission line.

The voltage source/impedance stamp is an 8x8 matrix with 18 nonzero entries. The size and number of nonzero entries are due to the voltage source connected in series with the impedance. The currents through the voltage source contribute the rows and columns Ibr1 and Ibr2. The series connection contributes the internal nodes Int1 and Int2.

Both the internal nodes and the rows and columns associated with ibr can be eliminated by converting to a current source in parallel with an impedance which is commonly called the current source/admittance model. This model is the subject of the next section.

3.3. Current Source/Admittance Stamp for a Transmission Line

Before the stamp for the current source/admittance model is discussed in Section 3.3.2, the stamp for a current source is derived in Section 3.3.1.

3.3.1. Independent current source stamp

In Figure 3.5, i_{in} and i_{out} represent the current from outside the source coming into the top node and going out of the bottom node. The currents i_{in} and i_{out} are due to the rest of the circuit, and are not being contributed by the current source i_{s1}. The current i_{s1} feeding into the top node and out of the bottom node is the contribution from the current source. The position of i_{s1} in the nodal equations for the top and bottom nodes determines the stamp for the independent current source.

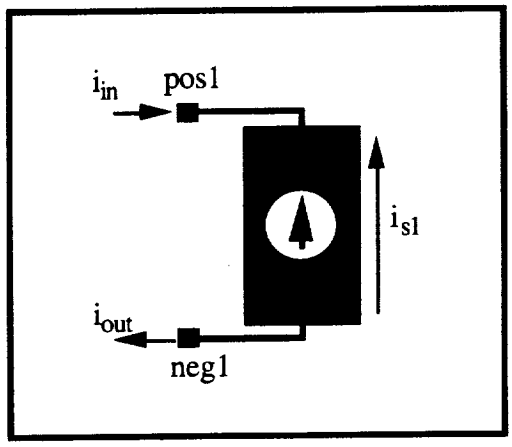


Figure 3.5. Current source diagram used in stamp derivation.

KCL at the top node yields

$$-i_{in} - i_{s1} = 0. \tag{3.4}$$

Transferring i_{s1}, since it is known, to the right-hand side yields

$$-i_{in} = i_{s1}. \tag{3.5}$$

Similarly, KCL at the bottom node results in

$$i_{in} + i_{s1} = 0 \tag{3.6}$$

or

$$i_{in} = -i_{s1}. \tag{3.7}$$

Therefore, the stamp for an independent current source of value I connected between nodes a and b and sending current into node a is as shown in Figure 3.6. The circuit matrix will consist of a certain number of rows and columns and has a right-hand side vector. It can be seen from the current source stamp that the current source does not create any new rows or columns in the circuit matrix. The value of the current source is added appropriately to the right-hand side vector. A current source does not add any new rows or columns, but fills an entry in an already existing right-hand side vector.

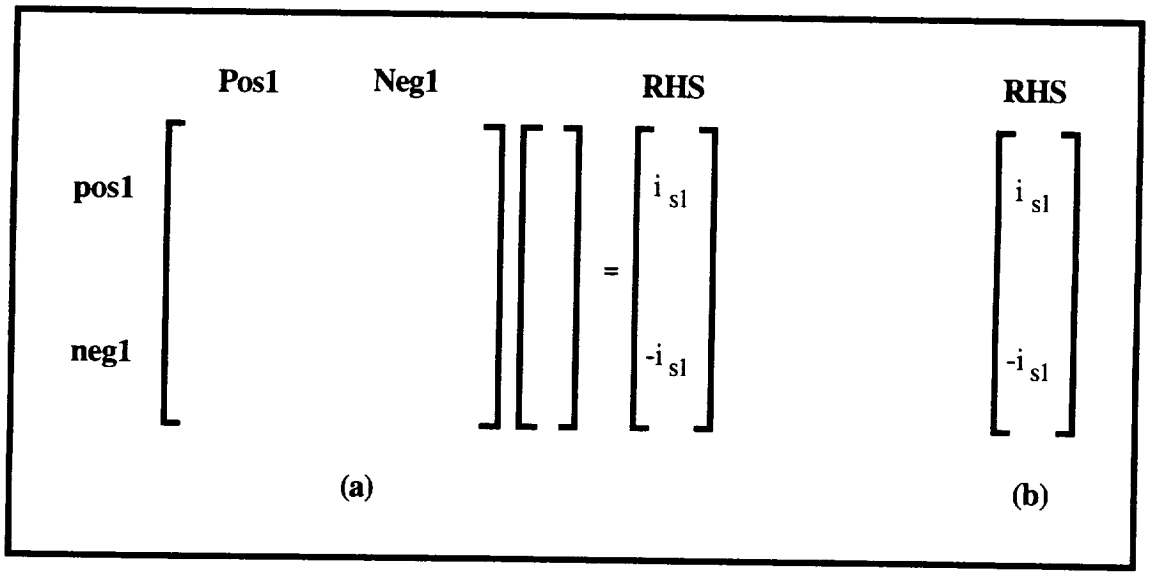


Figure 3.6. (a) Stamp for an independent current source. (b) Conventional representation of independent current source stamp.

3.3.2. Lossless transmission line stamp

The current source/admittance model for a lossless transmission line is given in Figure 3.7 and Equations (3.8) through (3.9), where Y_o is the characteristic admittance, τ

is the delay of the line, and i_{s1} and i_{s2} are the currents from the source on the left and right respectively. These can be derived from the voltage source/impedance model by finding the Norton equivalent circuit.

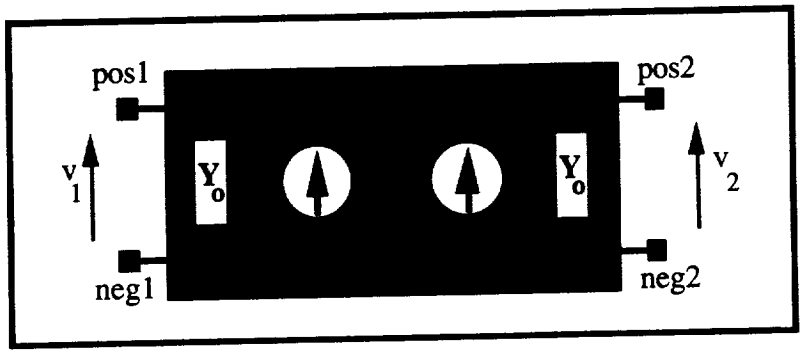


Figure 3.7. Current source/admittance model.

$$i_{s1}(t) = 2Y_o v_2(t-\tau) - i_{s2}(t-\tau) \tag{3.8}$$

$$i_{s2}(t) = 2Y_o v_1(t-\tau) - i_{s1}(t-\tau) \tag{3.9}$$

The stamp for the model in Figure 3.7 is shown in Figure 3.8. This stamp was derived by using the current source stamp and the impedance stamp. Notice that only rows and columns associated with the external nodes remain. The current source/admittance stamp requires a Y_o, i_{s1}, and i_{s2}. Y_o is the characteristic admittance of the line and remains constant through the transient analysis. The two currents i_{s1} and i_{s2} are computed as given in Equations (3.8) and (3.9). The equations indicate that the currents i_{s1} and i_{s2} require port voltage and source current from one line delay prior to the present time. The details of storing values at the present time to be referred to one line delay later are found in Chapter 6.

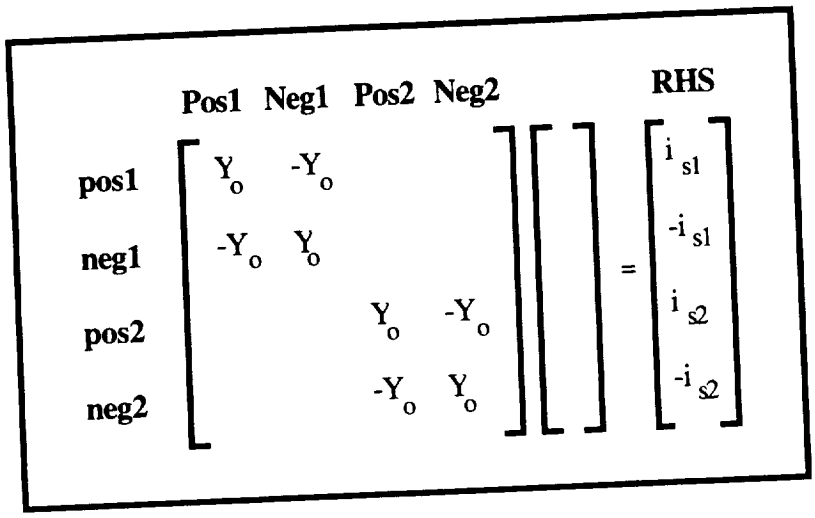


Figure 3.8. Current source/admittance stamp for a lossless transmission line.

3.3.3. Transient analysis of a lossy transmission line

The lossy transmission line model is very similar to the lossless model. The following discussion will mention only the current source/admittance stamp, but also applies in analogous fashion to the voltage source/impedance model. The stamp for the lossy model is as shown in Figure 3.8, but the expressions for i_{s1} and i_{s2} are not as given in Equations (3.8) and (3.9).

In this work, once the framework for a lossy line was established by implementing a current source/admittance model for the lossless line, the values for i_{s1}, and i_{s2} were supplied by a subroutine developed by D. Kuznetsov. See Kuznetsov and Schutt-Aine for details on models of lossy and lossless single- and multi-conductor lines [4].

3.4. Stamp Comparison

The voltage source/impedance stamp is shown in Figure 3.4 and the current source is shown in Figure 3.8. Notice that the current source/admittance stamp is 4x4, four times smaller than the voltage source/impedance stamp. The overall circuit matrix will be smaller when using the current source/admittance stamp and, therefore, will take less time to solve.

In addition to matrix size, the current source/admittance matrix has only 12 non-zero elements as opposed to the 18 of the voltage source/impedance stamp. With fewer nonzero elements, the current source/admittance stamp will have fewer memory references when being loaded into the overall circuit matrix and, therefore, will be loaded faster.

3.5. Summary

A device stamp will be at least n x n, not including the right-hand side vector, where n is the number of nodes. Extra nodes can sometimes be eliminated by rewriting the equations representing the device. The voltage source/impedance stamp for transient analysis of the transmission line is larger and has more nonzero entries than for the current source/admittance stamp. Transient analysis for a lossy transmission line and lossless line differ only in the expressions for the independent sources.

4. OVERVIEW OF ASPECTS OF SPICE3E2 RELEVANT TO DEVICE INSTALLATION

The purpose of this chapter is to supply the necessary background information on SPICE3E2 requires in device installation. All of the details of SPICE3E2 code operation are not presented. Complementary information is found in [6] and [7].

4.1. Organization and Conventions in SPICE3E2 Relevant to Device Installation

This section examines the programming conversions used in SPICE3E2, the organization of data structures, and defines terms used in the SPICE3E2 documentation, such as device, model , and instance.

4.1.1. Packages

The SPICE3E2 distribution consists of several different directories containing different portions of the code. For example, a directory exists containing all of the overall simulator routines, another containing only sparse matrix routines, another containing parsing routines (routines which interpret SPICE3E2 input files and commands), and further directories containing other functions specific to a particular simulator operation (see Section 4.2.). A group of related routines in SPICE3E2 is referred to as a package. The routines occupy a single directory dedicated to the package.

The routines of a package are organized into separate files. The contents of the directories are files containing functions. Usually, but *not always*, there is an organization of one function to a file. An important point is placing functions in

separate files, and grouping like function files into distinct directories only lends organization to a code, and does not provide a method to attain function hiding [18]. All functions can still call all other functions.

4.1.2. Interpackage communication

The code for SPICE3E2 possesses pseudo-function hiding attained via a convention restricting the manner in which functions from a particular package are called [18]. Calls from a function to functions within the same package are unrestricted; however, outer package functions must be called by using function pointers in an interface data structure. This restriction is up to the programmers to comply with. If calls between packages directly call a function as opposed to using the pointers in the interface, the compiler will not flag this as an error.

In the convention there are no restrictions placed on calls made within the package. These calls may be done by name or any other manner chosen by the programmer. Calls to functions in a particular package from outside the package, however, are done by accessing a data structure containing function pointers to package functions. Inclusion or exclusion of a function from the data structure will determine whether or not the function is callable from outside the package and, therefore, whether or not the function is private.

As an example, consider two packages A and B. As shown in Figure 4.1, A consists of several functions, and B is shown to consist of four functions. Bcreate, Bload, and Bdestroy are all pointed to by the interface data structure for package B; therefore, any function in package A with pointer to the interface data structure for package B can access the three functions, but can not access the function Binterpolate. Once again, this is only because the programmer adopts the approach of making all outer package calls via an interface structure. If a direct call by function name to Binterpolate existed in package A, there is nothing to stop a call with the proper syntax from executing.

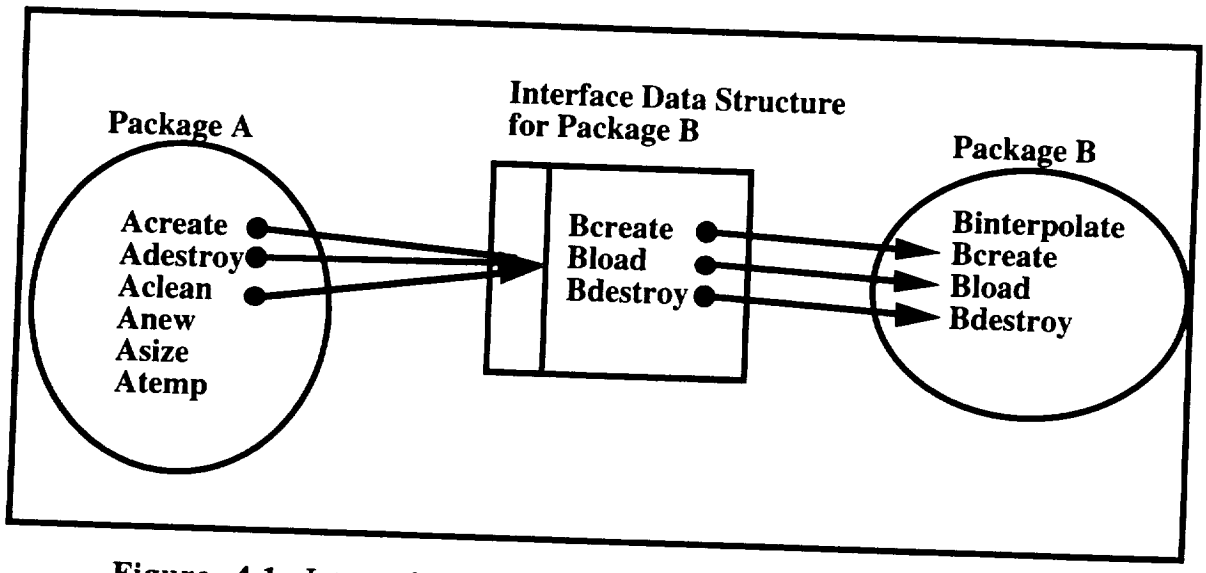


Figure 4.1. Interpackage function calls in SPICE3E2 are performed by accessing a data structure with pointers to the other functions of the other package.

The interpackage communication in SPICE3E2 works in the above described manner, and is an example of object orientation in ordinary C. Using the inter-package communication convention requires knowledge of the predesigned interface structure. When a new function has been introduced in a package and should be visible to the functions in other packages, the appropriate interfaces have to be modified.

A tempting short cut is to avoid the extra overhead of dealing with outer package function calls through function pointers in a data structure, but this is necessary when maintaining a code as large as SPICE3E2. Organizing a code in some object oriented manner increases the size limit of manageable code [18].

4.1.3. Package function naming conventions

All functions comprising SPICE3E2 (except main) belong to a package. The naming convention for a function is *PACKAGE_NAMEfunctionName*, where

PACKAGE_NAME is in all caps and has the same name as the directory containing the package. The first word in the function name is all lower case with all following words having only the first letter of the word capitalized. The naming of variables in data structures follows a similar convention of *DATA_STRUCTURE_ NAMEvariableName*. DATA_STRUCTURE_NAME is the name of the structure and if associated with a particular package has the name of the package. The conventions on lower case and uppercase for the variable name are the same as those for a function name.

4.1.4. Devices, models, and instances

Devices in SPICE3E2 have models and instances associated with them. A device is a circuit element category such as capacitor, BJT, or resistor. A BJT device will be used as an example. A circuit may contain 2 pnp and 3 npn BJTs for a total of 5 BJTs. The type of BJT is taken into account by models. A BJT has two models associated with it, a pnp and an npn. Consider a circuit in which there are no pnp BJTs and three npn BJTs. This circuit will result in zero instances of the pnp model and three instances of the npn model of the BJT device. Model data separate a pnp from an npn. Instance data identify particular pnps and npns.

The data structure used in SPICE3E2 to hold the information for a particular device is a linked list of model data structures with each list element containing model specific data, therefore, separating an npn from a pnp model, and a pointer to a linked list of instance data structures of the particular model. The data structure containing information on the BJTs in the example (zero pnps and three npns) is shown in Figure 4.2.

Even for devices with only one type of model, a model/instance organization exists to ensure future expansion to multiple models. A resistor has only one model. The resistor device data structure for a circuit containing only one resistor is shown in Figure 4.3.

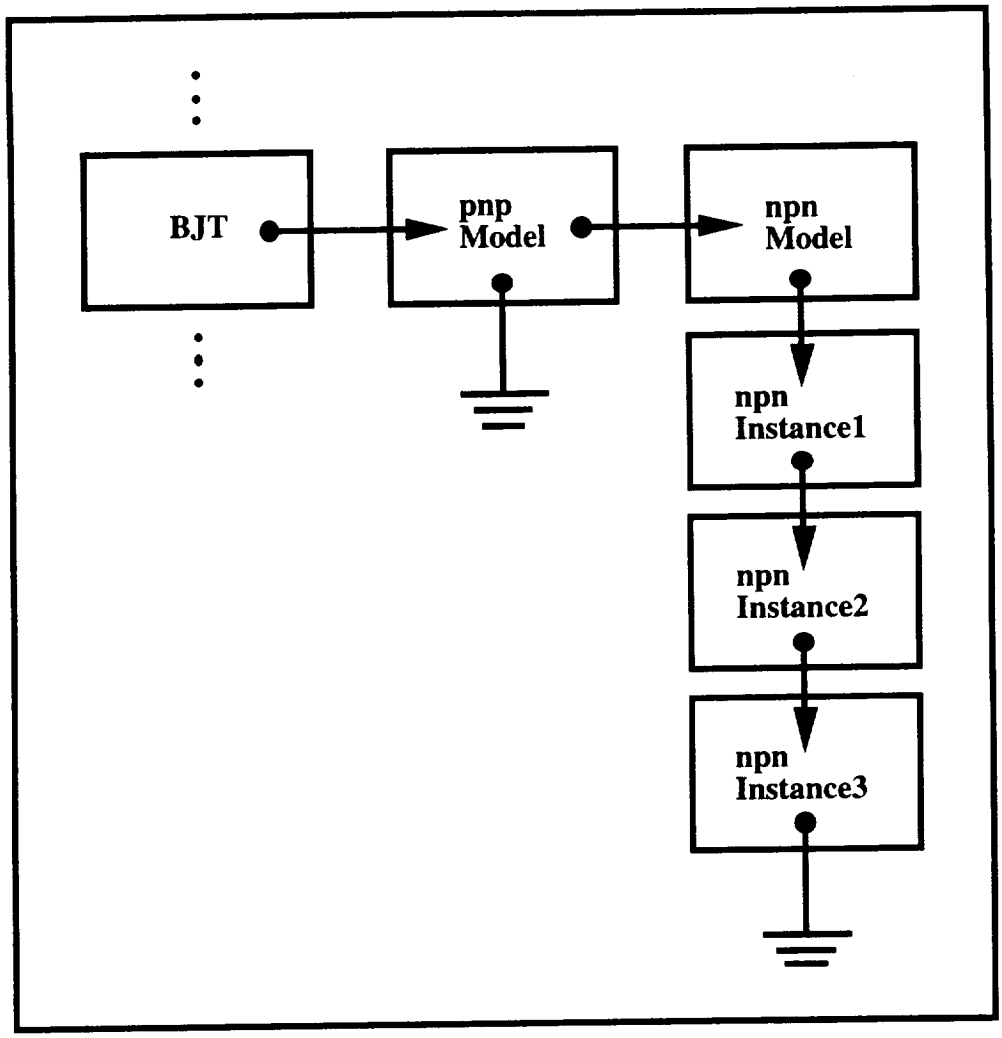


Figure 4.2. BJT device structure.

4.1.5. The CKT data structure

There are several data structures in SPICE3E2. Three of these structures are encountered in device installation. The interface structure and the device structure have already been discussed in Sections 4.1.2 and 4.1.4. The final structure to consider is the CKT structure. This structure is large and is described in more detail in [6]. The CKT structure is a main data structure of the simulator, and contains information on the simulator. Simulation time, pointers to the model data structures, and pointers to the circuit matrix are a few examples of the fields in the CKT data structure. Any functions requiring information on the circuit are passed the CKT structure. All of the device routines are sent the CKT structure when called.

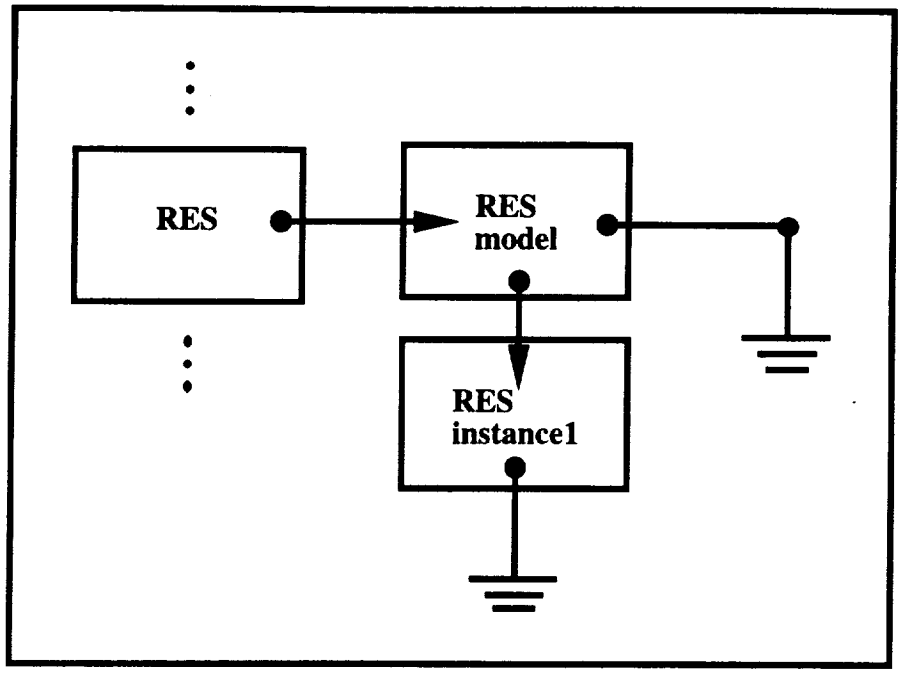


Figure 4.3. Resistor device data structure for a circuit with only one resistor.

4.1.6. Summary of relevant organizations and conventions

SPICE3E2 is a large code. The functions comprising SPICE3E2 are placed in files, usually one function per file. These files are organized into directories. Each directory containing source code of related routines defines a package. The related routines in a package perform a step in the circuit analysis algorithm. To assist in maintaining a code the size of SPICE3, a convention enforcing function hiding is used for calling functions outside of a package. This form of function hiding works only if the convention is used.

Finally, the information relating to all of the occurrences of a particular device in the circuit for analysis is a data structure consisting of linked model elements with each model element having a pointer to the beginning of a list of linked instances elements. A pointer to the beginning of the linked list of models for each type of device in the circuit is found in the CKT structure.

4.2. SPICE3E2 Directory Structure

The directory map for SPICE3E2 is shown in Figure 4.4. All directories are shown. Notice that there are no more directories below the directories in spice3e2/src/lib/dev.

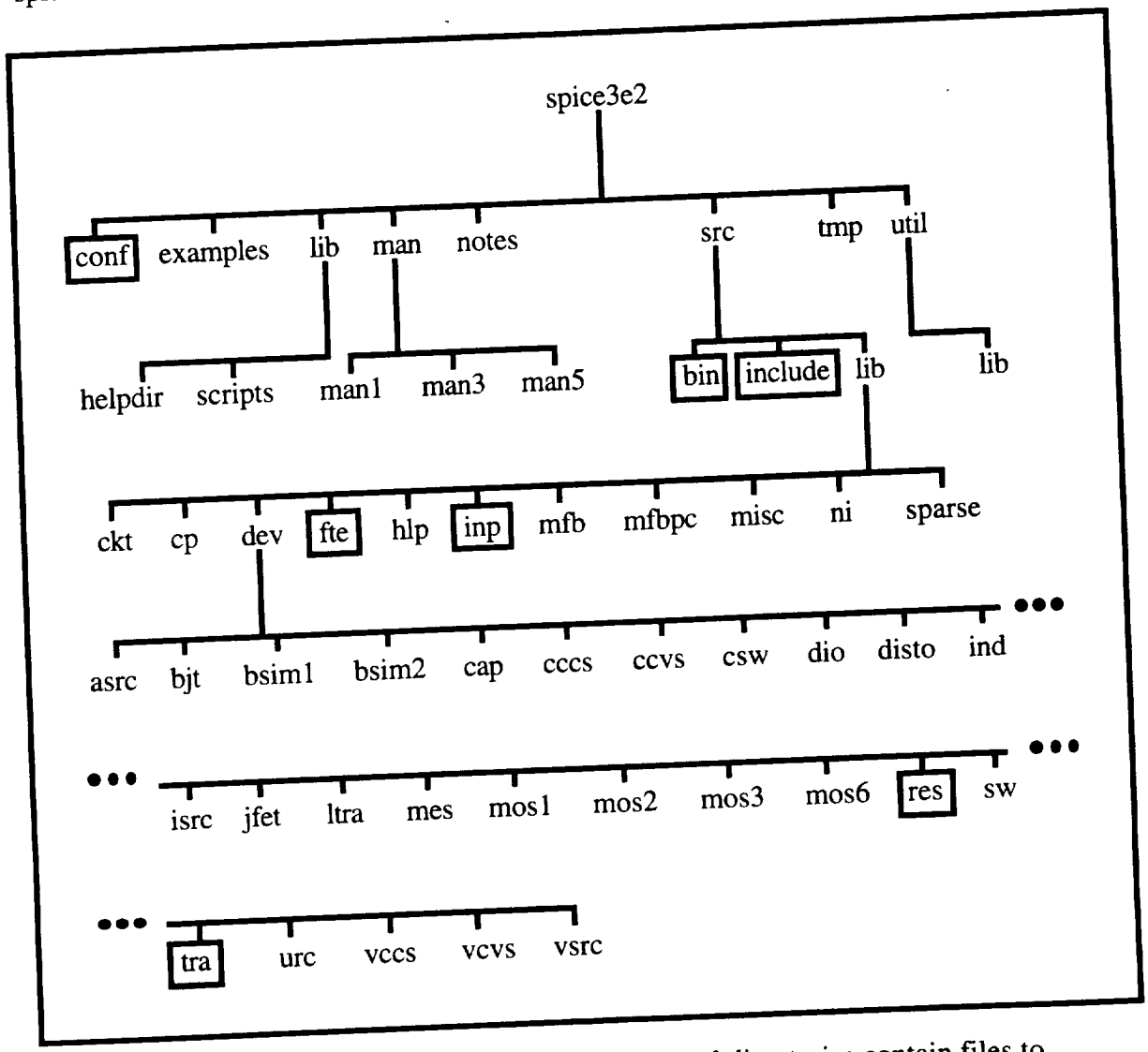


Figure 4.4. Spice directory structure. Boxed directories contain files to be modified during a device installation.

The makefile for SPICE3E2 resides in spice3e2/util and a configuration file which it uses is in spice3e2/conf. The configuration file is called defaults. All source code is in spice3e2/src. The function main (in file main.c), some configuration files, and the executable are in spice3e2/src/lib/bin. All the headers for files in the distribution are in

spice3e2/src/include. The directory spice3e2/src/lib contains the bulk of the source code with the overall circuit routine package in spice3e2/src/lib/ckt, and math routines in spice3e2/src/lib/cp (complex), ni (numerical integration), and sparse (sparse matrix). SPICE3E2 graphics package are found in spice3e2/src/lib/mfb (for workstations) and mfbpc (for PC's). The front-end package, or nutmeg routines, are in spice3e2/src/lib/fte, while the parsing package is in spice3e2/src/lib/inp. The directory hlp in the spice3e2/src/lib directory contains help facility routines, and any functions which defied categorization are in spice3e2/src/lib/misc. Finally, the device routines are in spice3e2/src/lib/dev. There is a directory for each device supported by SPICE3E2. These subdirectories of spice3e2/src/lib/dev contain routines for calculating stamp entries, inserting stamps, updating stamps, and other functions involved with the specific device stamp and the specific model and instance data structure for the device. The routines in the dev subdirectories are called by routines in the circuit package when a function has to be performed. For example, at the point when the circuit matrix is to be filled, a general function exists in spice3e2/src/lib/ckt which calls routines below in spice3e2/src/lib/dev/nnnnn, where nnnnn is the directory associated with a device having a stamp loaded into the circuit matrix.

Installing a new device means creating functions to handle the insertion of a new stamp, routines to send information to the stamp filling routines, new data structures for the stamp filling routines, and interface structures for the device. All of the routines and data structures are not in the same place. Installing a device is independent of many SPICE packages, but is more complex than just creating device routines in the spice3e2/src/lib/dev/new_device.

The boxed directories in Figure 4.4 indicate directories containing files to be modified in the device installation process. The directory spice3e2/conf contains a file which has to be modified so that new files will be compiled. The spice3e2/src/bin directory contains some configuration files to be modified, spice3e2/src/include contains the interface data structures and device data structures along with other important structure definitions. There is an obscure change to make in spice3e2/src/lib/fte. The directory spice3e2/src/lib/inp requires modification and additions so that the input file line for a new device can be parsed. Finally, a directory for the new device has to be created in spice3e2/src/lib/dev, and filled with routines for the new device (see Chapters 5 and 6).

4.3. Loading a Device Stamp into the Circuit Matrix in Spice3e2

There is one routine responsible for loading the dc and transient analysis stamps into the circuit matrix for a particular type of device. The name of the stamp loading function is XXXload, and is found in file xxxload.c in the spice3e2/src/lib/dev/xxx. As an example of this naming convention, the dc and transient load function for a BJT is the BJTload found in spice3e2/src/lib/dev/bjt/bjtload.c.

The load function, when called, will load the device stamp into the circuit matrix for each instance of each model for the device. In the analysis of a circuit containing 3 pnp and 5 npn BJTs a call to BJTload will insert the stamps for each of the eight BJTs into the circuit matrix.

The load function for each type of device in the circuit is called when it is time to load the circuit matrix. In a circuit consisting of a voltage source and resistors, only RESload and VSRCload are called. The function that calls the DEVload routines is in the spice3e2/src/lib/ckt directory, and is called CKTload. The function CKTload is called when it is time to build the circuit matrix. CKTload has access to an array of device interface structures. There is one element in the array for each type of active device. This gives CKTload access to the load function, as well as other device specific functions for each device. If an array element is null, then the circuit does not contain the device associated with that array element, and the load function for the device will not be called.

4.3.1. Storage of instance specific data

When the load function for a particullar device is called, a pointer to the corresponding device data is passed to the load function from CKTload. This pointer contains the beginning address of the model data structure. In this manner, the load function starts out with the first instance of the first model of the device, performs the loading and moves on to the next instance of the same model or the next model as appropriate.

The manner in which a load function traverses the device structure is illustrated in the code excerpt from the RESload function shown in Figure 4.5. Notice that the pointer to

the resistor device data has been passed in as the pointer to the first resistor model even though the CKT structure itself has also been passed. The device data pointer is passed separately since it is code less duplication to have CKTload, the single calling function, passing the pointer than it would be to have each device's load function extract the pointer. The trouble of extracting the resistor device data pointer has been left to the calling function.

In the function shown in Figure 4.5 the register variable *model* points to the first (and only) resistor model, and the variable *here* will point to the instance data being loaded. The first *for* loop advances along the linked list of models in the device structure, and the second *for* loop sets *here* to the first instance of the model type and later advances the *here* pointer to the next instance of the model until NULL is reached.

Figure 4.6 will be referred to in conjunction with the code in Figure 4.5 in the following example. Figure 4.6 shows a diagram of the resistor device data for a circuit containing only two resistors along with code executed when traversing the structure. The *here* pointer points to the first instance by accessing the beginning of the instance list from a pointer in the model structures indicated by the line, here = model->RESinstances. The inner for loop of Figure 4.5 ends its first run through and the stamp for instance 1 is loaded into the circuit matrix. After the first iteration *here* is advanced from instance 1 to instance 2 by accessing the pointer here->RESnextInstance. During the second iteration the stamp for the second resistor is loaded. Once control returns to the top of the inner for loop here->RESnextInstance is accessed and a NULL is encountered. Next, model->RESnextModel is accessed and a NULL is hit, and RESload is exited.

```
/*ARGSUSED*/
int
RESload(inModel,ckt)
   GENmodel *inModel;
   CKTcircuit *ckt;
      /* actually load the current resistance value into the
       * sparse matrix previously provided
       */
{
   register RESmodel *model = (RESmodel *)inModel;
   register RESinstance *here;

   /*  loop through all the resistor models */
   for( ; model != NULL; model = model->RESnextModel ) {

      /* loop through all the instances of the model */
      for (here = model->RESinstances; here != NULL ;
            here=here->RESnextInstance) {


.
.
.
            FILL STAMP
.
.
.
      }
   }
   return(OK);
}
```

Figure 4.5. Excerpt from RESload (spice3e2/src/lib/dev/res/resload.c) to illustrate advancing along the device structure.

4.3.2. Insertion of a device stamp into the circuit matrix

A stamp has a finite number of nonzero entries. These entries contribute to the overall circuit matrix as indicated in Chapter 2. In the instance structure a pointer exists for each nonzero stamp entry. The pointers are initialized to point to the specific place in the circuit matrix where the stamp entry is to go. These fast matrix loading pointers, or fast matrix pointers, for a resistor are listed below.

RESposPosptr
RESnegNegptr
RESposNegptr
RESnegPosptr.

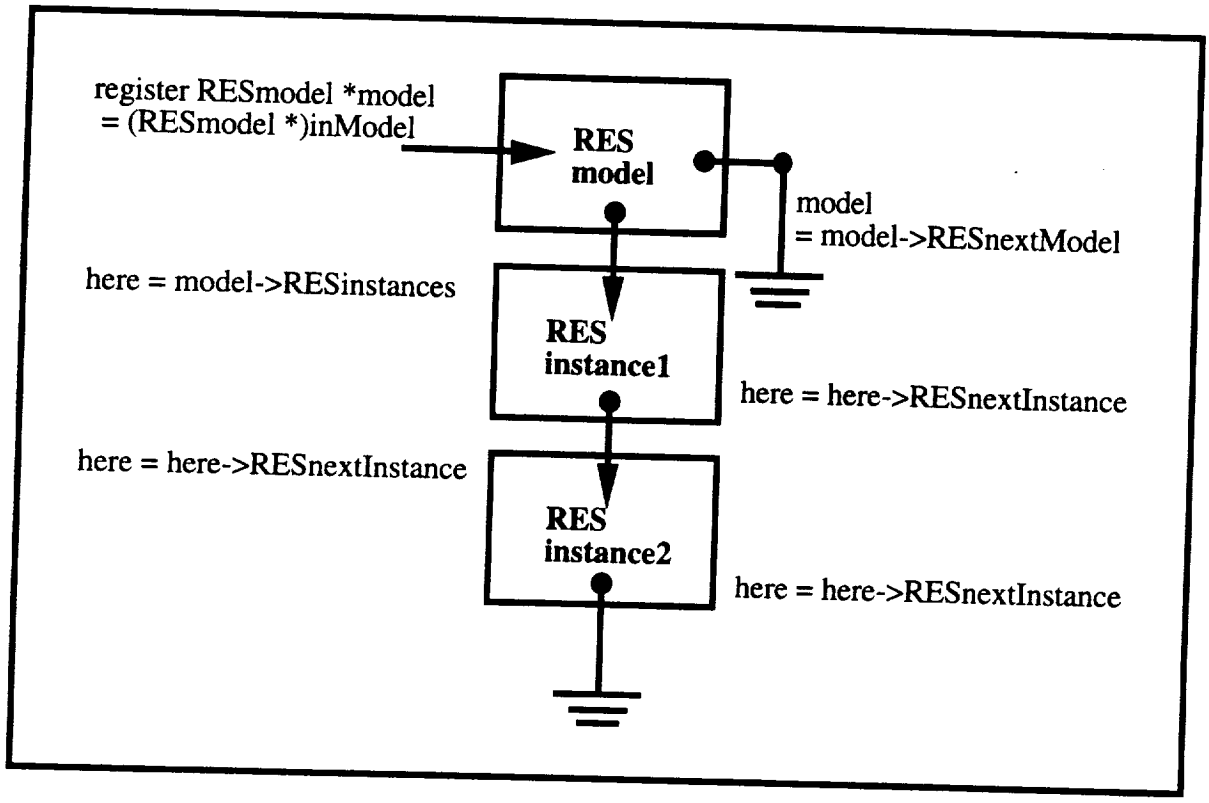


Figure 4.6. Assigning of the variables here and model while stepping through the instance list of two resistors (circuit has two resistors).

These pointers are initialized to circuit matrix positions in the function RESsetup. This function is shown in Figure 4.7. Once the fast matrix pointers are initialized they can be used in the code fragment as shown in Figure 4.8. This fragment shows the conductance of the resistor being inserted into the circuit matrix via the fast pointers.

```
.
.
.
int
RESsetup(matrix,inModel,ckt,state)
    register SMPmatrix *matrix;
    GENmodel *inModel;
    CKTcircuit*ckt;
    int *state;
        /* load the resistor structure with those pointers needed later
         * for fast matrix loading
         */
{
    register RESmodel *model = (RESmodel *)inModel;
    register RESinstance *here;

    /*  loop through all the resistor models */
    for( ; model != NULL; model = model->RESnextModel ) {

        /* loop through all the instances of the model */
        for (here = model->RESinstances; here != NULL ;
                here=here->RESnextInstance) {

/* macro to make elements with built in test for out of memory */
#define TSTALLOC(ptr,first,second) \
if((here->ptr = SMPmakeElt(matrix,here->first,here->second))==
                                                (double *)NULL){\
    return(E_NOMEM);\
}

            TSTALLOC(RESposPosptr, RESposNode, RESposNode);
            TSTALLOC(RESnegNegptr, RESnegNode, RESnegNode);
            TSTALLOC(RESposNegptr, RESposNode, RESnegNode);
            TSTALLOC(RESnegPosptr, RESnegNode, RESposNode);
        }
    }
    return(OK);
}
```

Figure 4.7. RESsetup excerpt, (spice3e2/src/lib/dev/res/ressetup.c), to illustrate advancing along the device structure.

```
.
.
.
int
RESload(inModel,ckt)
.
.
.
 /* FILL STAMP */
        *(here->RESposPosptr) += here->RESconduct;
        *(here->RESnegNegptr) += here->RESconduct;
        *(here->RESposNegptr) -= here->RESconduct;
        *(here->RESnegPosptr) -= here->RESconduct;
      }
   }
   return(OK);
}
```

Figure 4.8. Excerpt from RESload showing the stamp filling portion.

4.4. Loading of Device Data from the Input File

Section 4.3 examined the stamp loading process. The function CKTload calls the load functions for the devices comprising the circuit. Each load function loads the circuit matrix with the stamp for every instance of every model of the device based on parameter values in the instance structures of the device being loaded.

Some of the data present in the instance structures originally resided in the input file associated with the circuit being analyzed. The data were copied into the instance structure for a device from the input file by parsing the input, placing the input in appropriate tables, and retrieving the input data from the tables when needing to fill the fields of an instance structure associated with a particular device. This is illustrated in Figure 4.9.

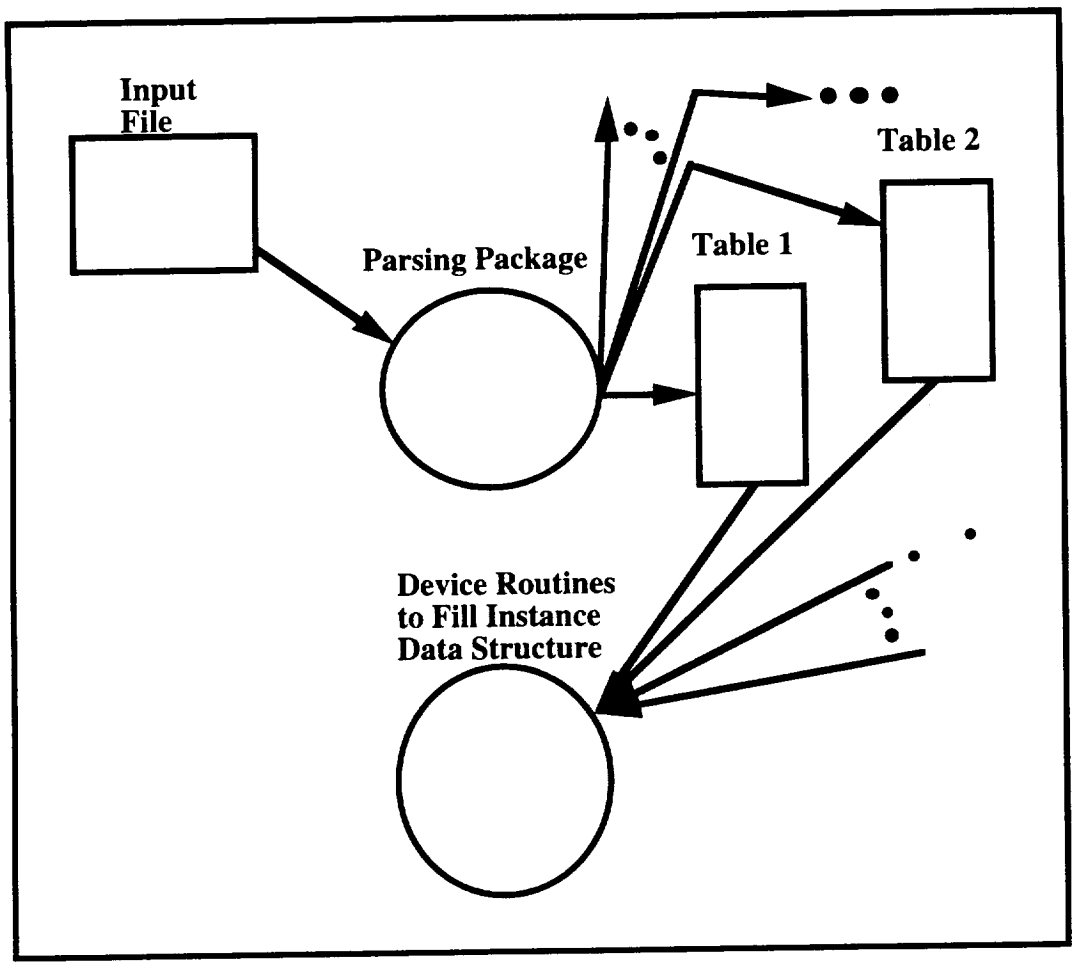


Figure 4.9. Travels of data from input file to device instance data structure.

Each device package contains a function called XXXparam where XXX is the prefix associated with a particular device package. XXXparam loads fields of the instance data structure for a device with appropriate values from the parameter tables. A data structure exists for each device which lists the valid parameters for the device. This data structure is useful when referring to entries in the parameter tables.

4.5. Summary

Files in SPICE3E2 that work together to perform a major simulator function are organized into packages. Packages communicate through data structures. Each device type in SPICE3E2 has an associated device data structure consisting of a linked list of model data structures with each model element having a pointer to the beginning of a linked list of

instance data structures. The circuit matrix is loaded by traversing the device data structure and accessing data for the individual stamps stored in the instance elements. The instance structure is loaded by device specific routines having access to the tables filled by the parsing routines.

5. DEVICE INSTALLATION STRATEGY

This chapter examines the general strategy for device installation into the SPICE3E2 circuit analysis program. The sections move from a general approach towards a more specific twelve-step plan.

5.1. General Approach

Each new device requires routines which handle parsing, updating stamp entries, loading the stamp into the circuit matrix, and other overhead associated with the device stamp. These functions obtain circuit data by accessing the CKT data structure, and are called by higher functions through pointers in an interface data structure. This description leads to a categorization of the device specific routines based upon the work which they perform. Device specific functions are involved in the performance of calculations and manipulations to load the unique stamp for the device into the circuit matrix and are also involved in communication with the rest of SPICE3E2. Therefore, in a broad sense, installing a new device module into SPICE3E2 entails insuring that the new module can communicate with the rest of the simulator and that the routines of the new module actually fill the circuit matrix correctly. This is illustrated in Figure 5.1.

Correct communication is established by reinstalling an already available SPICE3E2 device most similar to the new device. The routines of the old device are renamed. For example, all resistor functions and data structures may be reproduced under the name "nres" to supply a module for a negative resistor. No stamp specifics would be changed. Compiling and testing SPICE3E2 should result in no difference between the old device (res) and the reinstalled renamed version of the old device (nres). The changes to be made after the communication step will convert the reinstalled copy into an actual new device.

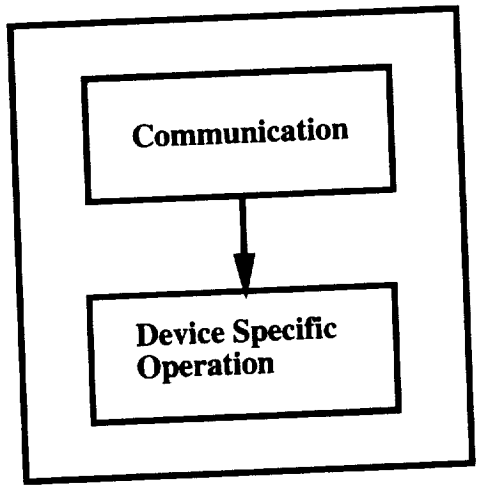


Figure 5.1. Two-step approach for device installation into SPICE3E2.

The approach outlined in this section is a safe methodology since the testing of the second step requires the first step to be complete. The first step should be checked before commencing with the second. If the first step is not verified, debugging the second step becomes very difficult. The source of a bug could be an error in the code dealing with device function behavior or an error in the code dealing with communication, and it is very difficult to discern which. For this reason, it is better to make changes to the device routines beyond mere function and data name changes only after the success of the communication modifications has been determined.

5.2. Specific Strategy

The general two steps of Section 5.1 can be broken down into further steps as shown in Figure 5.2.

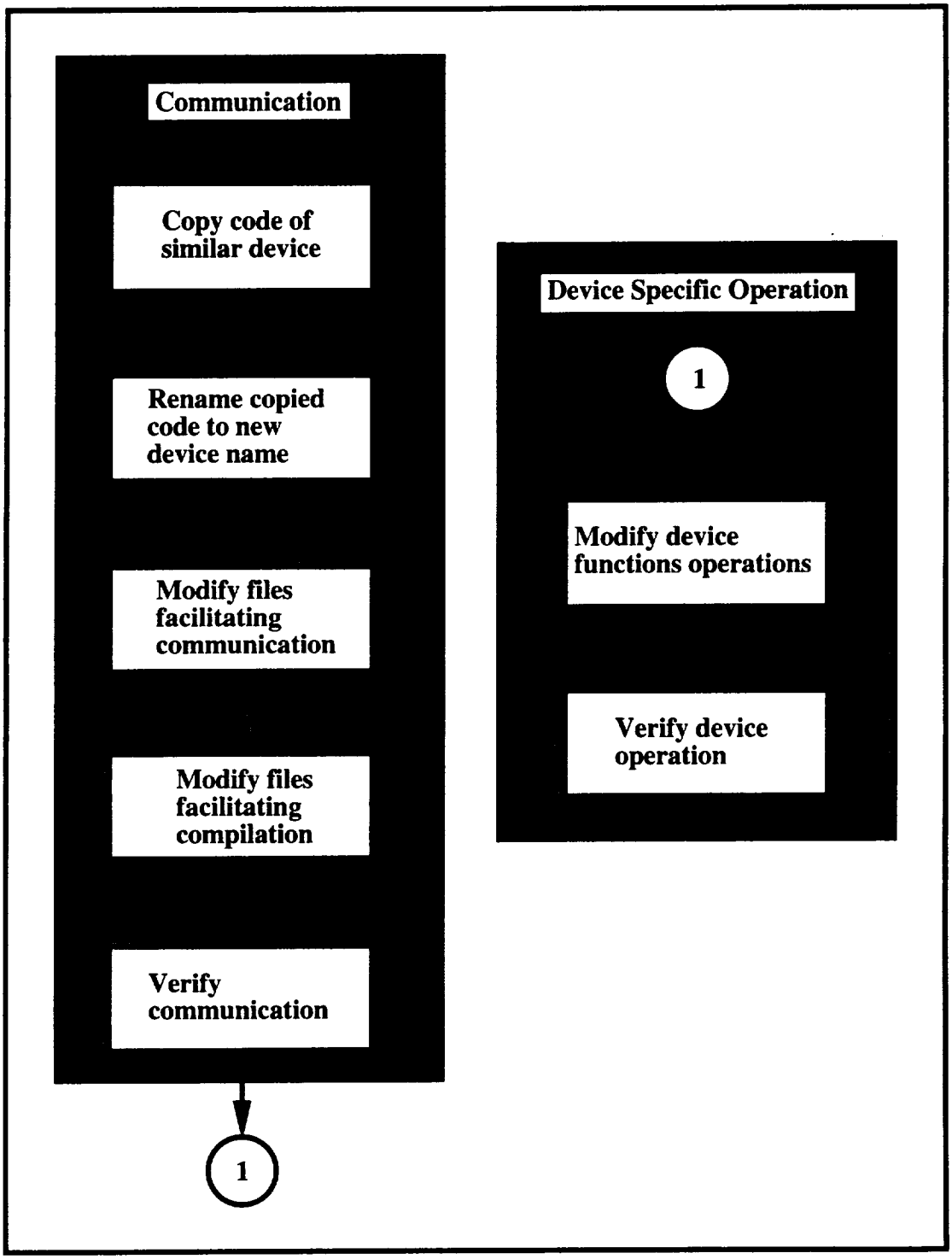


Figure 5.2. Steps comprising the communication and device specific operation steps of the general approach to device installation in SPICE3E2.

As previously mentioned, the first step in device installation is to copy device specific code from an already existing device. The names of variables, functions, and headers in the copied files, as well as the names of the files themselves, are changed to indicate association with the new device. Note, only the names and not the functionality of the routines have been changed. Next, the files and data structures responsible for inter-package communication in SPICE3E2 are updated allowing the simulator to access the new routines. Verifying that the new device behaves like the older device from which files were copied can not be done without recompiling SPICE. The UNIX make command is used to compile SPICE3E2. There are files used by the utility *make* in various directories, and these files require modification before a successful compile may occur.

The two steps after verification of communication comprising the device specific modifications are the changes to be made to the device specific routines to obtain new device behavior, and confirmation that the new device behavior is achieved.

5.3. Twelve-Step Plan

The twelve-step plan for device installation is shown in list form in Figure 5.3. It is a further breakdown of the device installation strategy. Details about these steps are discussed in Chapters 6 through 8 via three examples of device installation.

1. Create a directory for the device specific routines of the new device.
2. Copy the files associated with the SPICE3E2 device which is most similar to the device to install.
3. Change the names of the data structures, functions, headers in the copied device files.
4. Change the names of the data structure variables, functions, headers, and macro definitions in the copied header files.
5. Change the names of data structures, functions, and headers in copied parser file.
6. Modify the parser header file.

7. Modify the main parsing routine.
8. Modify the simulator files.
9. Modify the files used by *make*.
10. Check for successful establishment of communication.
11. Modify the operation of the device specific code.
12. Check for correct new device operation.

5.4. Summary

The strategy for device installation was viewed at different levels of detail. A device in SPICE3E2 participates in communication with the rest of the simulator and also fills the circuit matrix with the device stamp. The overall approach is to reinstall the most similar existing device under the new device name to establish correct communication before proceeding to change the operations of the copied device routines to achieve the desired new device behavior.

6. INSTALLATION OF A NEGATIVE RESISTOR

This chapter details the twelve-step plan introduced in Chapter 5 for a negative resistor. The steps 1 through 10 and step 12 are common to all device installations. This chapter uses the negative resistor as a simple example of establishing communication between a device package and the rest of SPICE3E2. The device is described in Sections 6.1 and 6.2. Following the two introductory sections are the details of the installation in Section 6.3. The source code referred to in this chapter is found in Appendix A.

6.1. Description of the Negative Resistor Stamp

A negative resistor is a one-port device having the IV relationship shown in (6.1). Current flows from the negative node to the positive node in a negative resistor. Figure 3.1 has been repeated as Figure 6.1 for ease of reference in the following discussion.

$$v = -Ri \tag{6.1}$$

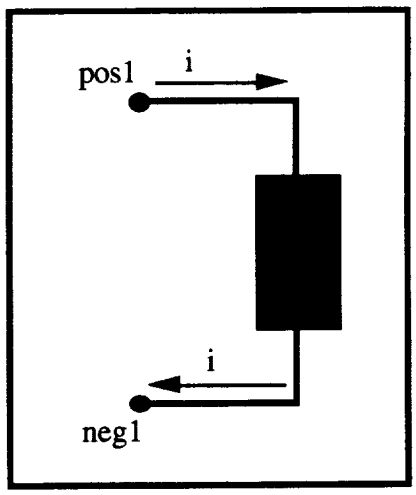


Figure 6.1. One-port device.

The contribution to KCL at the positive node is

$$\frac{-1}{R}(V_{pos} - V_{neg}) \tag{6.2}$$

The stamp for a negative resistor is as shown in Figure 6.2. It will be shown in the device installation procedure for the negative resistor that only the device code dealing with stamp filling has to be modified after communication with SPICE is established .

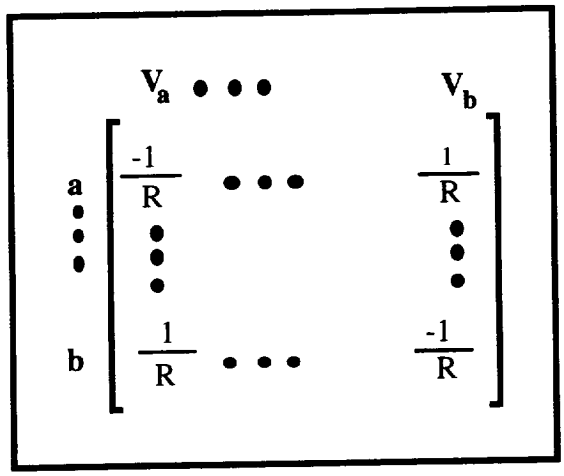


Figure 6.2. Stamp for a negative resistor.

6.2. Description of the Negative Resistor Input File Line

The description of a negative resistor device is not complete until the user interface, or the manner in which a user can include a negative resistor in a circuit analysis, is described. This means defining an input file line for the negative resistor or, in SPICE terminology, defining the negative resistor card. The card for a negative resistor has been chosen to be *Nxxxxx node1 node2 value* in this example. The xxxxx indicates the usual SPICE flexibility in naming of a device, node1 and node2 are the numeric labels of the positive and negative nodes, respectively, and *value* is the value in ohms of the negative resistor. Note that NRES1 5 0 4 represents a negative resistor connected from node 5 to ground of value 4 Ω. The I/V relationship for this particular device is

$$v = -4i \tag{6.3}$$

6.3. Details of the Twelve-Device Installation Steps for a Negative Resistor

This section consists of twelve subsections, one for each of the twelve-steps.

6.3.1. Create negative resistor directory

There are over 20 files associated with the negative resistor. Most of these files contain device specific functions. The convention in SPICE3E2 is to store files containing device specific functions in a subdirectory of spice3e2/src/lib/dev dedicated to the device. Since the negative resistor was new to SPICE3E2, a directory did not exist for it and had to be created. The directory that was created is spice3e2/src/lib/dev/nres.

6.3.2. Copy files of ordinary resistor

Files for the ordinary resistor exist in three places, as shown in Figure 6.3, spice3e2/src/lib/dev/res contains device files, spice3e2/src/include contains header files, and spice3e2/src/lib/inp contains the parser file.

The device files, found in the dev/res directory, were copied into the nres directory created in Section 6.1.1. The names of the device files begin with the prefix res except for the files makedefs, msc51.bat, and response.lib. All of the files copied into dev/nres beginning with prefix *res* were renamed to begin with prefix *nres*. For example, *resload.c* was renamed to *nresload.c*. The files makedefs, msc51.bat, and response.lib were not and should not be renamed.

The resistor header files begin with the prefix res, and were copied to similarly named files with prefix nres within the same directory spice3e2/src/lib/include. The result was the files nresdefs.h, nresext.h, and nresitf.h.

Finally, there has to be a specific function to parse the input file line of the negative resistor, and for this purpose inp2r.c was copied to inp2n.c within the directory spice3e2/src/lib/inp.

spice3e2/src/lib/dev/res

makedefs*	resdel.c*	resmdel.c*	response.lib*	ressload.c*
msc51.bat*	resdest.c*	resmpar.c*	respzld.c*	ressprt.c*
res.c*	resload.c*	resnoise.c*	ressacl.c*	ressset.c*
resask.c*	resmask.c*	resparam.c*	ressetup.c*	restemp.c*

spice3e2/src/include

resdefs.h	resext.h	resitf.h

spice3e2/src/lib/inp

inp2r.c

Figure 6.3. Files associated with the ordinary resistor listed under location.

6.3.3. Change names in copied device files

The device files copied into spice3e2/src/lib/dev/nres in Section 6.1.2 are appropriately named to indicate association with the negative resistor, but the contents of the files are still identical to those of the ordinary resistor. The first step in changing the contents of these files to obtain negative resistor behavior is to rename the functions, variables, and data structures found within. This changing of names will create a distinction between resistor and negative resistor functions, and will avoid function redeclaration errors and allow the device function code to compile. After renaming, however, the functions are identical procedure wise, and negative resistor functions will behave exactly like ordinary resistor functions. Changing the *behavior* of the device functions is discussed in Section 6.1.11.

An example of the typical changes to be made are shown in Appendix A under Section A.1. A typical device function before modification is shown in Section A.1.1 with the code that will be modified shown in boxes. A listing of the function after modification is supplied in Section A.1.2. Most of the files in the nres directory require the type of changes shown in the example of Section A.1.1 and Section A.1.2. The file nres.c is

slightly different and the places requiring changes in it are shown in Section A.1.3. The renamed version of nres.c is shown in Section A.1.4.

The modifications to the files makedefs, msc51.bat, and response.lib will not be discussed in this subsection, but will be deferred to 6.1.9. The renaming of the three files will be discussed in Section 6.1.9.

6.3.4. Change names in copied header files

There are further name changes of the kind performed in the device functions of spice3e2/src/lib/dev/nres. The changes are in the copied header files and copied parser file. The changes made to the copied header files nresdefs.h, nresext.h, and nresitf.h located in spice3e2/src/include are very similar to those made in Section 6.1.3 and, therefore, will not be discussed; however, the modified code is shown in Section A.2 in Appendix A. The following Section 6.3.5 discusses modification to the copied parser file.

6.3.5. Change names in copied parser file

The copied parser file is inp2n.c and resides in the directory spice3e2/src/lib/inp. The renaming of the function contents is similar to that discussed in the preceding two sections, but can be a little tricky since in certain places only the letter R(r) is being changed to N(n) as opposed to the prefix RES being changed to NRES, and not all instances of the letter R(r) are changed. For this reason the unmodified version of inp2n.c is shown in Section A.3.1 with the places for modification indicated by the boxes. Section A.3.2 shows the modified version of inp2n.c. Notice that tab->defRmod was changed to tab->defNmod, but ptemp.rValue is not changed, because ptemp is a variable of type IFvalue and rValue is a field in that structure which is to hold a real value. The r in rValue is not connected with resistance.

6.3.6. Modify parser header file

The header file for the parser, inpdefs.h, is in the spice3e2/src/include directory with all of the other header files in SPICE3E2. Additions were made to inpdefs.h to include the new parser function INP2N in the data structures contained in inpdefs.h. Excerpts from the modified versions of inpdefs.h are shown in Section A.4 with the lines

of interest boxed in. In the first section, the line defNmod is already included in inpdefs.h; therefore, the first section did not require modification. In the second section, the line *void INP2N(GENERIC*,INPtables*,card*);* has been added. In the third section, the line *void INP2N();* has been included.

6.3.7. Modify main parsing routine

Chapter 4 described the parsing of a device card. The function that searched the input file and called the appropriate device specific parsing function is INPpas2, found in spice3e2/src/lib/inp/inppas2.c. The calls to the device specific parsing functions are in a switch statement. Section A.5 shows parts of this switch statement in the excerpts from the modified version of INPpas2. The code which was added to the function in order to modify it is boxed. When the first character read from a line is an 'N' or 'n' the boxed code is executed.

6.3.8. Modify simulator files

The simulator files that were modified are bconf.c, cconf.c, and config.c located in spice3e2/src/bin and subckt.c located in spice3e2/src/lib/fte. The changes to be made in the files are very similar to each other. The files bconf.c and cconf.c are very much alike and require identical changes. Three sections from modified versions of bconf.c are shown in A.6.1, with the added code boxed in. The file config.c differs from bconf.c in that the first section of bconf.c shown in A.6.1 is not included in config.c. The two following sections are included in config.c, however, and require the same changes as in bconf.c and cconf.c.

An excerpt from subckt.c is shown in A.6.2. The line added is *case 'n': return (2);* and is shown in the box in the listing for subckt.c. This line specifies the number of nodes of the device and is the reason for the *return (2)* in the added line. If the new device had four nodes, then a 4 would be returned.

6.3.9. Modify files used by make

The changes discussed to this point, if done correctly, will perform ordinary resistor analysis when the negative resistor is called since the names of and the contents of copied functions have been changed, but not the workings of the functions. The line

NRES2 5 0 7 will result in an ordinary resistor from node five to ground with a value of 7Ω. The functions called to perform the analysis will be those of the negative resistor, but these will perform exactly as the counterpart ordinary resistor functions . Before testing to see if this is indeed the result, the code had to be compiled. SPICE3E2 is compiled using the make utility of UNIX [19]. The make utility uses some files in various directories; these files are makedefs, msc51.bat, and response.lib, and the file *defaults* in the spice3e2/conf directory. The instances of the files makedefs, msc51.bat, and response.lib to modify are the copied ones in the new directory spice3e2/src/lib/dev/nres, and those in the spice3e2/src/lib/inp directory.

The contents of the file makedefs in the nres directory before modification are shown in Section A.7.1. There are three areas for change. The section CFILES should contain the names of the nres function files as opposed to the res function files. The COBJS section should contain nres objects as opposed to res objects. Finally, the MODULE should be nres.

The contents of msc51.bat of the nres directory are shown in Section A.7.2 and, as in makedefs, the prefixes should be changed to nres. Care was taken that not all instances of res in msc51.bat be changed to nres. There is only one instance of res that should not be changed and it occurs at the bottom of the file in the line *lib ..\.\dev1.lib @response.lib.*

The changes made in spice3e2/src/lib/dev/nres/response.lib are only prefix changes from res to nres, similar to msc51.bat. Therefore, this particular response.lib is not shown, but an excerpt from a response.lib for another directory is shown in Section A.7.4.

The files makedefs, msc51.bat, and response.lib in the spice3e2/src/lib/inp directory required modification to include inp2n.c in the compilation. This meant adding *inp2n.c* to the list of CFILES in makedefs and *inp2n.o* to the list of COBJS in makedefs. In msc51.bat the line *cl /I..\.\.\include /c inp2n.c >> ..\.\.\msc.out* was added. The addition to response.lib in the inp directory was *+inp2n.obj&*. The line *+inp2n.obj&* was inserted into the list contained in response.lib as shown in Section A.7.4.

Part of the function of the file defaults is to indicate to the make utility whether or not to compile the code for a device. Section A.7.3 contains an excerpt from the file defaults after modification with the addition *nres* boxed in. This includes the negative resistor functions into the compilation process.

The code was recompiled by typing *util/build mips* from within the spice3e2 directory under the Ultrix operating system. If SPICE3E2 is installed under a different operating system, the Spice3e2 Installation Guide can be consulted. The guide is found in the file spice3e2/readme.

Compilation may be done after appropriate steps in the installation process to check the changes. In this case, the files used by make should be updated before compilation, and parts of step 9 should be implemented earlier.

6.3.10. Verify establishment of communication with the main code

This step was performed by constructing a simple input circuit which included a negative resistor, performing a SPICE run on it, and checking that the negative resistor behaved like an ordinary resistor with current flowing through it from positive to negative.

If there are any errors, correcting them and repeating this step will make debugging the overall installation easier. Once this step is successful, communication has been established between the new package (nres) and the rest of SPICE3E2. The remaining steps tailor the operation of nres to that of a negative resistor and verify successful negative resistor operation.

6.3.11. Modify operation of copied device code

The changes made up to this point were name changes, or the additions of names to files to facilitate proper compilation or proper communication with the rest of the code. The next changes converted the nres package from a mere renamed duplicate of the res package to a unique package to handle the negative resistor. This step is the largest for most other devices, but in the contrived case of a negative resistor, only one function has to be changed. The function that was changed is the NRESload function, which loads the stamp into the matrix. The stamp for an ordinary resistor can be converted to that of a negative resistor and vice-versa by negating the entries of the respective stamp. An unmodified version of NRESload is listed in Section A.8.1. The boxed code loads the resistor stamp into the matrix by adding the resistor admittance to the appropriate matrix positions.

Negating the admittance will result in the negative resistor stamp being loaded, and the nres package will perform analysis for a negative resistor. The modified code is listed in Section A.8.2

6.3.12. Check new device operation

Verifying installation of the negative resistor was similar to step 10, except this time it was checked to see if the current ran from negative to positive through the negative resistor.

6.4. Summary

The installation of a negative resistor was described to illustrate the steps in the installation of a device into SPICE3E2, particularly the steps involved with establishing communication between the new package and the rest of the code which are common to all device installations. The negative resistor is a contrived practice case and was very similar to an ordinary resistor. The only function to be changed after establishing communication was the stamp loading function. In the installation of a nonhypothetical device, modification of copied device code operation will be more substantial as shown in Chapters 7 and 8.

7. INSTALLATION OF A LOSSLESS TRANSMISSION LINE MODEL FOR TRANSIENT ANALYSIS

A lossless transmission line model already exists in SPICE3E2. The reasons for installing a new lossless model are primarily a more efficient stamp, and to set the foundation for a lossy model. As mentioned in Chapter 3, the lossless line model already in SPICE3E2 uses a voltage source/impedance stamp. This chapter details the installation of a lossless transmission line model which utilizes the more efficient current source/admittance stamp. The installation of the lossless model sets the foundation for the installation of the lossy model of Chapter 8. As with the negative resistor installation example, the most similar device will be chosen as the foundation device for the lossless transmission line. The files of the existing lossless line model will be copied to form a basis for the new model.

The first two sections recap information from Chapter 3 and provide information on the code used to manipulate a crucial data structure of the line model. Section 7.3 describes some of the differences in coding between the voltage source/impedance and current source/admittance models. Section 7.4 summarizes the first ten steps in the device installation procedure for the lossless model being installed. The fifth section gives details on step eleven of the installation procedure. This chapter covers the installation of code for transient analysis of only the lossless line. Stamps for ac or transient analysis will not be examined. The installation procedure for ac and transient analysis is very similar to the presentation to follow for the transient analysis case.

7.1. Lossless Transmission Line Models Revisited

Chapter 3 introduced the voltage source/impedance model and current source/admittance models for transient analysis of a lossless transmission line. In both models, the impedance element remained a constant equal to the characteristic value for the

line. The source terms, however, varied at every time step. The voltage source expressions (Equations (3.1) and (3.2)) are printed again as (7.1) and (7.2), and those for the current source model ((3.8) and (3.9)) are shown as (7.3) and (7.4). In either model values from τ, or one-line delay, in the past are referenced.

$$v_{s1}(t) = 2v_2(t-\tau) - v_{s2}(t-\tau) \tag{7.1}$$

$$v_{s2}(t) = 2v_1(t-\tau) - v_{s1}(t-\tau) \tag{7.2}$$

$$i_{s1}(t) = 2Y_o v_2(t-\tau) - i_{s2}(t-\tau) \tag{7.3}$$

$$i_{s2}(t) = 2Y_o v_2(t-\tau) - i_{s1}(t-\tau) \tag{7.4}$$

7.2. Referencing Previous Values

The lossless model installed in SPICE3E2 is the voltage source/impedance model and a data structure is used to store values that will be referenced later in the simulation. This data structure is a simple one-dimensional array called the delay table. A pointer to it exists in the transmission line instance data structure. The same structure and code associated with the maintenance of the structure will be used for the current source model to be installed. The following three subsections describe the delay table. The information may be useful in future modifications or in understanding how equations of the form of Equations (7.1) to (7.4) are supported in the transmission line package.

7.2.1. The delay table

The delay table is a one-dimensional array. Space in the delay table is allocated three elements at a time. The first element holds the time associated with the data to be stored in the following two array positions. As an example (see Figure 7.1), if the simulation time is 2.5, then a 2.5 is stored in the first of the three elements. The second of the three array positions is used to store the information that will be required by the first source. In the case of the voltage source/impedance model at time 2.5, this means that the value of $2V_2(2.5) - V_{s2}(2.5)$ is stored in the second position. The third matrix position holds the information that will be needed by the second source, with the present simulation time being 2.5; the value stored is $2V_1(2.5) - V_{s1}(2.5)$.

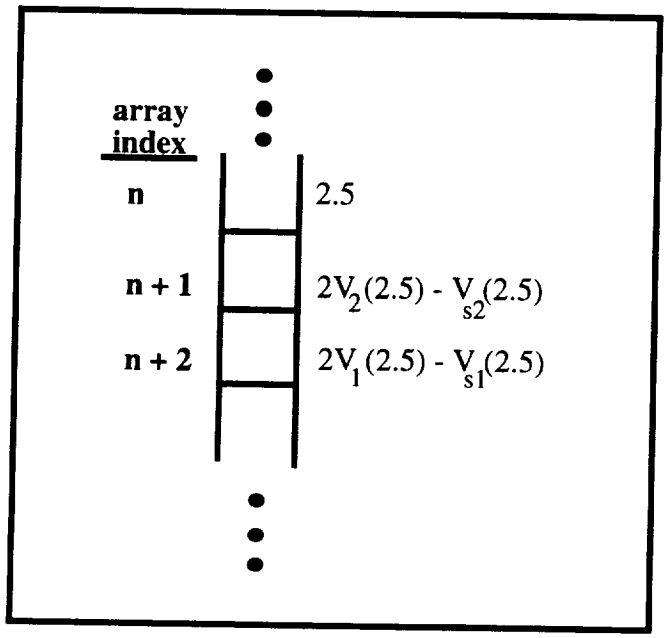


Figure 7.1. Delay table example.

7.2.2. Delay table management

Predicting the number of entries in and, therefore, the size of the delay table is not possible since SPICE3E2 uses variable time steps. The delay table is allocated with a set size at first and reallocated as needed. As the simulation proceeds, certain values in the delay table become too old, the time associated with the data is earlier than the t-τ, where t is the present simulation time (see Figure 7.2). More specifically, two values just before t-τ *will be used*, but any earlier than these two will not be referred to again. The table is checked after every matrix solving for values which are too old. The entries which are too old are discarded and the array rearranged. The code concerned with checking and rearranging the delay table is in spice3e2/src/lib/dev/ntra/ntraacct.c (see Appendix B, Section B.5.1).

7.2.3. Interpolation

Figure 7.2 also illustrates the fact that with respect to some present time t data from exactly τ in the past may not be available. Since variable time steps are used, a value from exactly t-τ away is usually not available, and second-order interpolation is used to obtain the value from τ away in all cases. The code that performs the interpolation is part of the

function NTRAload found in spice3e2/src/lib/dev/ntra/ntraload.c (see Section B.4.1). The boxed code finds the three values to use in the interpolation, stopping with the first value greater than t-τ. Here->NTRAinput1 and here->NTRAinput2 (see Section B.4.1) are set equal to the interpolated values.

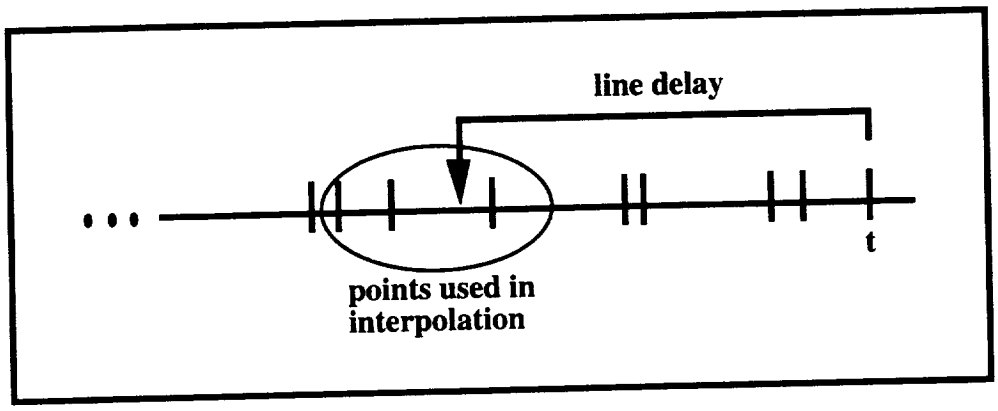


Figure 7.2. Time line.

7.2.4. Initial conditions and the delay table

A special case has to be instituted for the delay table. The run following the establishing of initial conditions will try to reference some value from τ away. Most often this will be some negative value from before the simulation started, or when the initial conditions are assumed to be valid. This situation is handled by loading the initial value of the sources as the first two sets of entries from the initial condition run. Close examination of the code above the final boxed section in the listing of B.4.1 will reveal that these three values of time (the times of the initial two entries and the entry following the initial condition run) make it possible to handle the situation of a negative value of t-τ since the counter i is not incremented if the time associated with the latest entry is greater than t-τ. The loop is exited and values t1 through t3 are available with t3 the latest and t1 the earliest.

7.3. Voltage Source/Impedance Model vs. Current Source/Admittance Model

The main difference between the two models is the resulting stamps. The stamps derived in Chapter 3 for the two models are shown again as Figures 7.3 and 7.4.

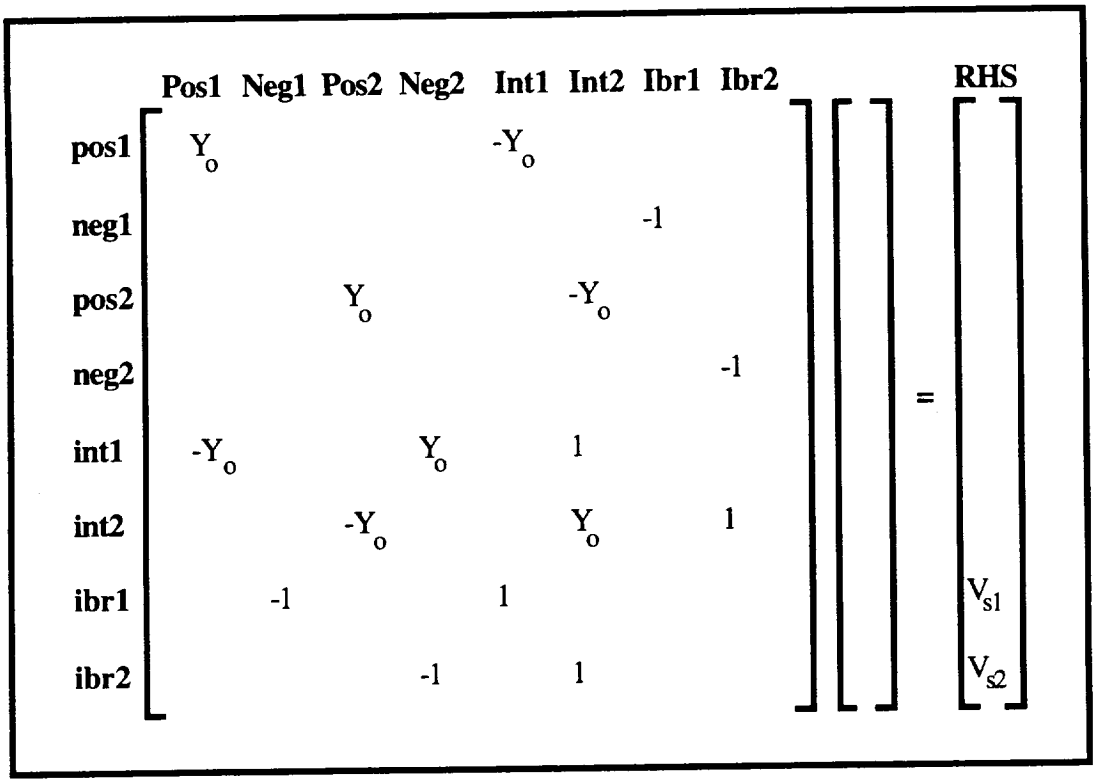


Figure 7.3. Voltage source/impedance stamp.

The stamp for the voltage source/impedance model has four rows and columns which are not present in the current source/admittance model. The instance data structure associated with the current source/admittance model will not contain fields for the internal nodes and device equations of the opposing model. The data structure of the current source admittance model will also not contain pointers to the sparse matrix incorporating the internal node and branch equation fields. Section B.1.1 shows the instance data structure for the voltage source/impedance model with the fields to be removed or replaced in the process of converting to the current source/admittance model boxed in. Changing the device data structure for the lossless transmission line will require changing any functions which refer to the old removed or changed fields. The only functions that refer to fields within the instance data structure are the device functions. The other changes to make to the device functions when converting from the voltage source/impedance model to the current source/admittance model result from the different source terms (see Equations 7.1 - 7.4).

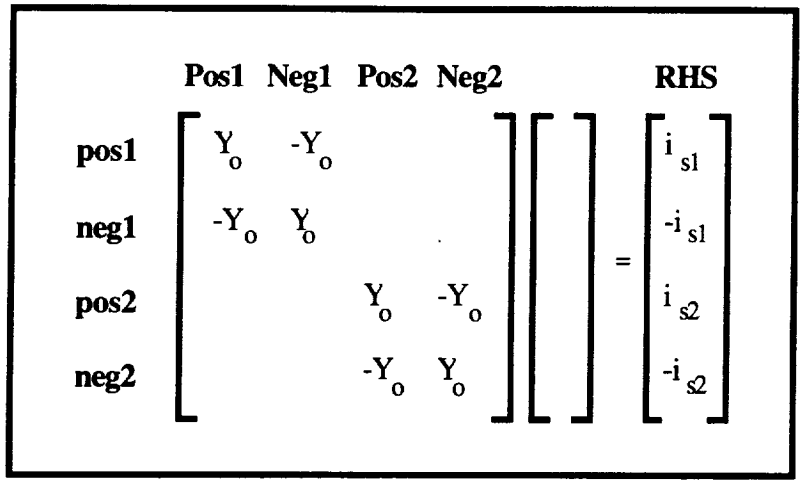


Figure 7.4. Current source/impedance stamp.

7.4. Summary of Steps 1 Through 10 of the Installation

This section summarizes steps 1 through 10 of the device installation for the lossless transmission line. These steps are very similar to those performed for the negative resistor of Chapter 6.

Since the lossless model which comes with SPICE3E2 will be kept, a new installation is performed for the current source admittance model of the lossless line. This means that existing lossless line code will not be modified, but copied to serve as a foundation for the new lossless model. In Chapter 8, the files of the current source/admittance model will be modified to convert to a more general lossy model.

The input file line, which will include the new lossless line in a circuit, is *Nxxxx posnode1 negnode1 posnode2 negnode2 z0=value <td=value> <f=freq <nl=nrmlen>> <ic=v1, i1, v2, i2>*. This is exactly the same as the line for the voltage source/impedance model, except that an N is used instead of a T [9] This means that the parsing function for the new line INP2N in file spice3e2/src/lib/inp/inp2n.c will be an exact copy of INP2T in inp2t.c, except the names in the code will have to be changed. The function INPpas2 must call INP2N when an N is read. The code in INPpas2 to call INP2N was added during the negative resistor installation and only the comments have to be changed.

An N will incorporate the new line into a circuit and NTRA will be the prefix of the functions and data structures of the new lossless model. The prefix for the lossless model already in SPICE3E2 is TRA. Since an N is being used for the new model the code associated with the negative resistor was written over. The negative resistor was an example case only and not intended for use; therefore, the letter N is used again for the new lossless transmission line.

The following list briefly describes steps 1 through 10 of the device installation for the current source/admittance model for transient analysis of the lossless transmission line. All the steps were carried out as listed unless otherwise stated.

1. Create directory spice3e2/src/lib/dev/ntra.

2. a. Copy all files in spice3e2/src/lib/dev/tra to the directory spice3e2/src/lib/dev/ntra and change the names of the files to begin with ntra as opposed to tra.

   b. Copy tradefs.h, traext.h, and traitf.h in the directory spice3e2/src/include to ntradefs.h, ntraext.h, and ntraitf.h in the same directory.

   c. Copy inp2t.c to inp2n.c in the spice3e2/src/lib/inp directory.

3. Change the names in the copied device files to represent association with the new model. All instances of TRA (tra) go to NTRA (ntra) and T's to N's.

4. Change the names in the copied header files.

5. Change the names in the copied parser file spice3e2/src/lib/inp/inp2n.c.

6. Modify the parser header file. This step was omitted since the changes to make are exactly the same as was done for the negative resistor.

7. Modify the main parsing routine to include a call to INP2N. Only comments have to be changed since this change was made during the installation of the negative resistor.

8. Modify the simulator files bconf.c, cconf.c, config.c, of the spice3e2/src/bin directory, and subckt.c of the spice3e2/src/lib/fte directory.

9. a. Modify the files used by make in the spice3e2/src/lib/dev/ntra directory and the spice3e2/src/lib/inp directory. The changes in the inp directory were skipped since they were already performed for the negative resistor installation.

   b. Include ntra in the list of device modules to be compiled in the file spice3e2/conf/defaults. Make sure to remove nres from the compilation list.

c. Compile using make.

10. Submit a run which includes the new transmission line. Look for results in agreement with the voltage source/impedance model or with some other precalculated results.

7.5. Changes Comprising Step 11

This chapter details the eleventh step of the twelve-step installation plan. The list that follows is a breakdown of the files to modify in the eleventh step. The subsections will detail each of the steps. The files in the following list are all found in spice3e2/src/lib/dev/ntra.

1. ntradefs.h
2. ntrasetup.c
3. ntraask.c
4. ntraload.c
5. ntraacct.c
6. ntratrunc.c

7.5.1. Modifications to ntradefs.h

Section 7.3 discussed most of the changes to the device data structure found in ntradefs.h. The file ntradefs.h also contains the device model data structure which did not require modification. Section B.1.1. contains the listing for ntradefs.h before modification. As previously stated, the first two blocks are removed since the new stamp does not include internal nodes or branch equations. The second, third, and fourth boxed-in sections are all pointers to the sparse matrix and are changed since the internal nodes and branch equations have been removed. The boxed-in section under device parameters is deleted since the contents of it are constants used to refer to the internal node and branch equation fields which were removed. The modified version of ntradefs.h is listed in Section B.1.2. Notice the changes in the pointers to the sparse matrix.

7.5.2. Modification to ntrasetup.c

A listing of NTRAsetup before modification is in Section B.2.1. The code in the first four boxes is eliminated in the modified version since it references instance data

structure fields which have been deleted in Section 7.5.1. The last three boxes contain code which initializes the pointers to the sparse matrix. Since the sparse matrix pointers were modified in Section 7.5.1, the code in the last three boxes of the premodified NTRAsetup listing will be replaced with code to initialize the new sparse matrix pointers. The modified version of NTRAsetup is listed in Section B.2.2. Notice the changes to the sparse matrix pointer allocation section.

7.5.3. Modifications to ntraask.c

Excerpts from the NTRAask function are found in Section B.3. The two boxed sections contain references to parameters and instance data structure fields which no longer exist (see Section 7.4.1), and the code within the boxes is deleted.

7.5.4. Modification of ntraload.c

The file spice3e2/src/lib/dev/ntra/ntraload.c contains NTRAload, the function which loads the lossless transmission line stamp for dc and transient analyses. Section B.4.1 shows excerpts from NTRAload before modification.

The first two boxed sections contain code which enters the voltage source/impedance stamp into the matrix except for the right-hand side vector. The right-hand side of the stamp contains source values (see Figure 7.4) which have to be calculated. The code in the first two boxes was replaced by code which enters the current source/admittance stamp into the matrix.

The next box contains code to be executed during a dc run. The dc analysis stamp and code will look different for a current source/admittance model. The dc model will not be discussed since the primary concern is the transient model. The ability to specify initial conditions of a circuit will be used to avoid doing a dc analysis to obtain the operating point. After the transient analysis code is working correctly, the dc analysis stamp for the current source/admittance model may be installed using the same techniques being demonstrated in the examples of transient analysis stamp installation.

The fourth box of code will only be executed if initial conditions are being used. This code had to be modified so that based on the initial conditions the initial value of current sources can be set as opposed to voltage sources.

The fifth box contains code which is only executed if a dc analysis was done to find the initial conditions. It was not modified since the dc stamp for the current source/admittance model has to be installed. As long as the transient stamp installation is tested with initial conditions specified by the user, this code will not be executed. It will be commented out to prevent accidental execution. The final box contains the loading of the right-hand side components of the device stamp. This code will be changed.

None of the code discussed so far in this section computes the right-hand side for a regular transient analysis iteration as opposed to the special case of the very first iteration in the transient analysis. The reason is that any other runs will use interpolation on values of the delay table to find the appropriate source value as explained in Section 7.2.3. The loading of a present value into the delay table for future reference is explained in the next section. The modified version of NTRAload is shown in Section B.4.2.

7.5.5. Modifications to ntraacct.c

The boxed sections of the NTRAacct listing in Section B.5.1 show the code involved with loading the delay table. As explained in Section 7.2.1, the values of Equations (7.1) and (7.2) are stored. This was changed to store the values of Equations (7.3) and (7.4) The modified code is shown in Section B.5.2.

7.5.6. Modifications to ntratrunc.c

Excerpts from the unmodified version of the function NTRAtrunc are listed in B.6.1. The purpose of NTRAtrunc is to set break points based on the changing of the source (see [7]). This is another piece of code which utilizes fields which no longer exist after the modifications have been made to the instance data structure. In the unmodified version, the v variables v1, v2, v3... are set equal to the voltage source values of the model (see B.5.1 also). Notice that in the modified version of NTRAtrunc listed in Section B.6.2 v1, v2, v3,... now represent the value of a current source as opposed to a voltage source (see Section B.5.2 also). The variables v1, v2, v3,... were not renamed in order to avoid

an extra source of error during the modification of NTRAtrunc, but may be changed if a more representative name is desired.

7.6. Summary

The lossless transmission line model installed in this chapter has a different stamp, a current source/admittance stamp, from the standard lossless line model of SPICE3E2. The lossless model already in SPICE3E2 was not modified to use a different stamp, instead a lossless line model using the current source/admittance stamp was treated as a new device and installed using the resident SPICE3E2 lossless model as a basis for the new line model.

The lossless line model of this chapter will be used as the starting point for the installation of a lossy line model in the next chapter. The device code for the new lossless model will be modified to obtain the lossy line performance.

8. INSTALLATION OF A LOSSY TRANSMISSION LINE MODEL FOR TRANSIENT ANALYSIS

This chapter details the installation of a lossy transmission line. The first step is the installation of a simpler lossless model. This was shown in Chapter 7. The modifications to the lossless line code of Chapter 7 which convert the lossless functions into functions that are a part of the lossy line module are discussed. Most of the changes made are a result of the argument requirements of a function which will return source values for the current source/admittance model. After the difference between the lossy and lossless line models is restated (see also Chapter 3), the function which returns source values is discussed. The modification process can be split into two parts. Parts one and two are described and code modification examples associated with each part are discussed. All of the code examples referred to in this chapter are in Appendix C.

8.1. Difference Between the Lossy and Lossless Line Stamps

The lossy model uses the same current source/admittance model as the lossless model (see Section 3.3.2). The stamp looks the same as shown in Figure 7.4. The difference between the two stamps is the setting of the source values in the right-hand side vector. The lossless model uses Equations (7.3) and (7.4), but the lossy model uses different expressions. More can be read about the expressions for the sources in the report by D. Kuznetsov and J. Schutt-Aine [4].

As far as the installation process of the lossy line is concerned, a subroutine written by Kuznetsov will be used to supply the source values. The function characteristics and required parameters will be discussed in the following section.

8.2. Requirements of the Source Function

The function which returns the value of the current sources for the lossy model is called G and is located in spice3e2/src/lib/dev/ntra/vdmmodel.c along with some utility functions. It is a parallel, step invariant difference model.

The function requires several difference parameters to calculate source values. These parameters are referred to as difference parameters and will vary with line specifications. Another piece of code supplied by Kuznetsov, called vdmdiff, located in spice3e2/src/bin, will produce a file of difference parameters associated with a specified line. Most of the contents of the parameter file are read in to be passed to the function G, but some are read in and used in calculations elsewhere. See Section C.2 for a listing and organization of the parameter file. The function declaration of G is shown in Section C.1, along with a short explanation of the arguments to be passed to it.

Some additional points about the arguments of G are presented in this paragraph. The variable xyold is the value of the voltage at port 1 or port 2 depending on whether L=1 or 2 from the previous time iteration. This value is also the most recently available solution at the ports. The values of the current source depend on values from one line delay away as in the lossless model. The values of Equations (8.1) and (8.2) from a line delay before the present time are set to xw. Equation (8.1) is used when L=1, and 8.2 when L=2. The value of xw during the previous iteration is xwold. Isp is allocated on the heap and is passed to the function.

$$2(v_2(t-\tau)Y_{o2} - G_2(t-\tau)) + I_{s2}(t-\tau) \tag{8.1}$$

$$2(v_1(t-\tau)Y_{o1} - G_1(t-\tau)) + I_{s1}(t-\tau) \tag{8.2}$$

Since the function G requires difference parameters to be passed to it, the parameters will be read into the instance data structure and passed as needed. In this manner the difference parameters for the device are stored where the rest of the device data are stored, and since G will be called an arbitrary number of times, accessing a data structure will be more efficient than reading from the parameter file on every call. The new fields introduced into the instance data structure required that several modifications be made to the code. The strategy used when making the modifications is described in the following section.

8.3. Modification Strategy

The lossy line is included in a circuit by using the following syntax: *Nxxx posnode1 negnode1 posnode2 negnode2 filename*. This is similar to the manner in which a lossless line was called, except that all of the parameters following *negnode2* have been substituted by *filename*, the name of the difference parameters file. The modifications associated with insertion of the lossy line model, therefore, deal not only with the proper calling of the function G, but also with the proper passing of the new input file line.

Parsing changes are saved for part two. In part one the parameters are hardwired into the instance data structure and are used in calling G. Upon successful completion of part one, the second part is undertaken. Part two involves changing a data structure so that the new syntax is recognized and writing a function to read the difference parameters into the instance structure. The list of changes forming part one is at the beginning of Section 8.4 and the list for part two is at the start of Section 8.5.

8.4. Conversion to Lossy Line Model Part I

The following is a list of steps for part one of the modification procedure. The following subsections discuss each of the steps.

1. Insert fields into the instance structure to store the difference parameters.
2. Modify ntraitf.h.
3. Modify ntraparam.c.
4. Modify ntrasetup.c.
5. Modify ntraload.c.
6. Modify ntraacct.c.
7. Modify ntratrunc.c.

8. Update spice3e2/src/lib/dev/ntra directory and modify files used by make.
9. Verify that the model gives the correct results for the specific line used.

8.4.1. Insert parameter fields into device data structure

Section C.3.1 shows the listing of spice3e2/src/include/ntradefs.h after modification. The fields in the box are the new fields, one for every parameter in the line difference parameter file produced by vdmdiff, and a few used in providing storage for the results of calculations associated with arguments passed to G. The new fields do not have the prefix NTRA in front of them; this was only to distinguish them from the fields already present while modification attempts were in progress. The field prefixes have not been changed at the time of this writing, but will be changed in order to stay consistent with the naming conventions in SPICE3E2..

8.4.2. Modify ntraitf.h

This header file, found in the same directory as ntradefs.h, requires only a small change as is indicated by the boxed code in the listing of Section C.3.2. The listing is of code after modification and the modification is to make sure that the name field indicates that the device is a *lossy* as opposed to a *lossless* transmission line.

8.4.3. Modify ntraparam.c

The listing of C.4 shows a modified version of spice3e2/src/lib/dev/ntra/ntraparam.c. The changes made to the code of functions NTRAparam are indicated in the figure by the boxes. The necessity for these changes will be evident in the next section where the parameter fields added to the device structure will be set to specific values based on a particular transmission line. The parameters of impedance and time delay are given in the parameter file. Difference parameter fields were created to hold these values when read from the parameter file. There is no need to specify them on the input file line. The code that is commented out will prevent program execution from stopping because characteristic impedance and line delay were not specified on the input file line.

8.4.4. Modify ntrasetup.c

The function NTRAsetup in spice3e2/src/lib/dev/ntra/ntrasetup.c is where the parameter values for a specific transmission line case will be hard wired in. NTRAsetup was chosen over NTRAparam since all of the parameters can be loaded in one call to the function. The modification to NTRAsetup is only temporary, and will be undone in part two, where a function to read in the data and set the value of the fields in the device structure will be employed. The boxed code in C.5 shows the setting of the various device parameter fields to the specific values. The parameter values are for a line of length 0.675m, distributed inductance 5.39×10^{-7} H/m, distributed capacitance 3.9×10^{-11} F/m, distributed resistance 1.25×10^{2} Ω/m, skin resistance 0.0 $\Omega/(Hz^{-0.5})$, and distributed conductance of 0.0 S/m.

A listing of the parameter file for the given line is in C.2. The numbers up to the first line of text are all difference parameters to be loaded into the device structure. The text after the numbers explains the format of the file and the transmission line for which the numbers are valid. A short description of each of the parameters is given in the file format explanation. Notice that Mwf, Mwb, My1, and My2 all indicate the size of the arrays required to hold the data for awf and fcwf, awb and fcwb, ay1 and fcy1, and ay2 and fcy2, respectively.

8.4.5. Modify ntraload.c

The function NTRAload in spice3e2/src/lib/dev/ntra/ntraload.c before modification is listed in C.6.1, and after modification is listed in C.6.2. The boxes of C.6.1 show places in the code that will be modified.

The first box highlights the variables internal to the function. Some more internal variables are added for use as temporary variables. When examining the listing in C.6.2 not all of the variables may be used; this is the result of deletions and other modifications made some time after the insertion of the variables.

The second box is around code associated with loading the left-hand side of the device stamp. The field NTRAconduct is not used for the lossy line. Examination of the parameter file listing C.2 will show Ay1 and Ay2 to be the characteristic

admittance of the line. Since both values will equal each other only Ay1 is used, as shown in Section C.6.2.

The following box is around code which loads the source values for the very first transient analysis run with the restriction that the initial conditions have been supplied by the user. A valid expression for the excitation argument required by G, which is equivalent to the expressions in Equations (8.1) and (8.2), is shown in the boxed code in Section C.6.1. Therefore, the expressions are left the same except that NTRAconduct is replaced by Ay2 and Ay1. The expressions are set equal to the temporary variables xw1 and xw2. The variable xwold is set equal to xw1 and then G is called for the particular source value. Variable xwold is set to xw1, since this is the initial run and all existing conditions are assumed to have existed without disturbance long before the present iteration. Notice the arguments passed to G in the call shown in Section C.6.2. The first call has a first argument of 1 since the value of the first source is being requested. The arguments here->ay1, here->fcy1, here->awb, and here->fcwb are pointers to the beginning of arrays as indicated in Section C.2, here->My1, here->Awb, and here->Mwb are also difference parameters being passed in the appropriate fields. The value being passed for the present time step is ckt->CKTtime. This is the value of the simulation time step and since there have been no previous runs and the simulation starts at time zero, ckt->CKTtime can be passed as the length of the time step. At the end of the lossy line listing, here->oldtime is set to the value of ckt->CKTtime, and during the next iteration the time step length can be set by calculating ckt->CKTtime - here->oldtime. The voltage at port one or the input port is given by here->NTRAinitVolt1 the user-defined initial port voltage. The time step size from a delay away is also set to ckt->CKTtime. The call to G for the second source is analogous to the first. Notice that the values of here->xwold1 and here->xwold2 are set for use in the next run.

The next boxed code in Section C.6.1 deals with all of the runs following the initial one. The code in the box loads the values of the excitations based on interpolation on values stored in the delay table. In the lossy case the values of the sources are set by calls to G. The code up to the boxes does not change since, as will be seen in the listing for NTRAacct, the delay table will be used to store values of excitations, and the value of excitation from one line delay away (xw1 and xw2) will have to be found using interpolation just as the values for the sources were being found in the lossless case. The remainder of the modifications can be followed in Section C.6.2. After xw1 is set, Ty is calculated as mentioned in the previous paragraph. Tw is the time step size previous to the

time from one delay away and therefore is set to t3 - t2. The value of xyold is obtained by consulting the solution of the matrix formed during the previous iteration which is the most recent solution available for the voltages at the positive and negative nodes. The pointer to the matrix solutions is ckt->CKTrhsOld, and specific values can be referenced by indexing the address space it points to appropriately. After xyold is calculated, all of the parameters required by G are available and G is called to return the value for source one. After the call to G, the value of here>xwold1 is rewritten by xw1, and xwold1 is now updated for the next run. It is important to update xwold1 only after G has been called; otherwise, xw1 will be entered as the value for xw1 *and* xwold1, and the subroutine will have incorrect arguments passed to it. The only parameters which have to be calculated for the call associated with the second source are xw2 and xyold. After the call to G for the second source, the value of here->xwold2 is updated. Finally, just before entering the newly calculated values of here->NTRAinpu1(2) into the right-hand side of the stamp, the *oldtime* field of the instance data structure is updated.

8.4.6. Modify ntraacct.c

The function NTRAacct located in spice3e2/src/lib/dev/ntra/ntraacct.c is where entries are made into the delay table. The listing of NTRAacct before modification is in Section C.7.1, with the code which makes entries into the delay table boxed in. The listing should be modified to enter the present values of xw1 and xw2, as given by Equations (8.1) and (8.2). Once these are entered into the delay table, they can be referred to later in the interpolation section of NTRAload. The modified code is shown in C.7.2.

8.4.7. Modify ntratrunc.c

Breakpoints are set in NTRAtrunc and since the excitation equations changed to the form given in (8.1) and (8.2) the expressions in NTRAtrunc, which load the values of excitation at the present iteration, must be modified accordingly. The code to modify is shown in the box in the listing of Section C.8.1, and the modified code is listed in Section C.8.2.

8.4.8. Update spice3e2/src/lib/dev/ntra directory, and modify files used by make

The files to include in the ntra directory are vdmmodel.c, which contains the difference model function G, and some files containing math support for the routines in vdmmodel.c, vdmmodel.h, vdmmath.c, and complex.c. The files vdmmodel.h and vdmmath.c are included at the beginning of vdmmodel.c, and complex.c is included in vdmmath.c. Therefore, when vdmmodel.c is compiled so are the other files listed in this section. The file vdmmodel.c is compiled by including it in the listing of source files and object files in makedefs, msc51.bat, and response.lib.

8.4.9. Verify the model gives the correct results for the specific line used

This is the final self-explanatory step of part one. Once the above modifications were made, an input case was run and compared with available data provided on runs of the same difference model in an environment other than SPICE. The code was debugged until agreement between the two solutions was reached.

8.5. Conversion to Lossy Line Model Part II

Section 8.4 described modifications to achieve analysis capability for the case of one specific transmission line. Only one line could be analyzed since the difference parameters for a particular line were hard coded in NTRAsetup (see 8.4.4). In part two a function will be written that will read in data from the difference parameter file and set the appropriate fields in the instance data structure. In addition, an extra input file line parameter will be added that will allow the specification of the name of the difference parameters file on the input file line itself.

The steps comprising part two are the following:

1. Write the function fileread.
2. Modify ntra.c to include a new parameter.
3. Modify NTRAparam to call fileread.
4. Modify ntradefs.h.

5. Verify the changes worked.

8.5.1. Write the function fileread

A listing of the function fileread contained in spice3e2/src/lib/dev/ntra/fileread.c is in Section C.9. The code is self-explanatory, comprised mainly of memory allocation for the arrays and fscanf functions to read in the data. Later the code can be improved by having the fileread function return a flag to indicate success or failure and then letting SPICE3E2 handle the exiting from program execution as opposed to exiting in fileread itself.

8.5.2. Modify ntra.c to include a new parameter

The way a new parameter for a device is allowed on the input file line is to include the name of the parameter in the interface parameter, or IFparm, table contained in spice3e2/src/lib/dev/ntra/ntra.c. There are no changes required on the parser end of the code. The reason for this is that when the parser reads in a parameter on the input file line the IFparm table is searched and the parameter value set. Therefore, when *filename = a.spr* is read, a.spr is stored as the value of filename. The listing in C.10 shows a modified ntra.c with the modification made boxed in.

8.5.3. Modify NTRAparam to call fileread

The modified version of NTRAparam with the call to fileread shown is listed in C.11. When NTRAparam is called with NTRA_PARAM_FILE_NAME, the appropriate code is executed in the switch statement, and the value of the filename is retrieved and set equal to the filename field of the device instance structure. The addition of a filename field is one of two changes that will be discussed in the following section. The code on NTRAsetup, entered in Section 8.4.4 was removed at this stage.

8.5.4. Modify ntradefs.h

Two changes were made inside ntradefs.h. The first was the addition of a field to hold the name of the difference parameter file. The chosen field name sticks to the

SPICE3E2 naming convention. The name of the field is NTRAfileName. The second modification is to assign a numerical value to the constant NTRA_PARAM_FILE_NAME. This allows proper functioning of the case statement in NTRAparam and of the IFparm table.

8.5.5. Verify the changes worked

The verification of part one assures the calculation portion of the lossy line model. Part two was verified by using the syntax for the new lossy line in the input file and confirming that the instance structure was loaded correctly.

8.6. Transient Analysis Run of the Lossy Line

This section discusses a circuit analysis run incorporating the lossy line model for which this chapter details the installation process. The steps in formulating and carrying out the analysis are listed below. As in previous chapters the following subsections discuss each of the steps.

1. Formulate circuit.
2. Place line specifications in a file.
3. Run vdmdiff to obtain the difference parameters.
4. Write SPICE input file.
5. Perform analysis.

8.6.1. Formulate circuit

A circuit for which a lossy line run was performed is shown in Figure 8.1. This particular circuit is simple and primarily tests the operation of the lossy line module. The voltage pulse supplied by the source is shown in Figure 8.2. Notice that the pulse will not repeat for 100 ns due to the period specified.

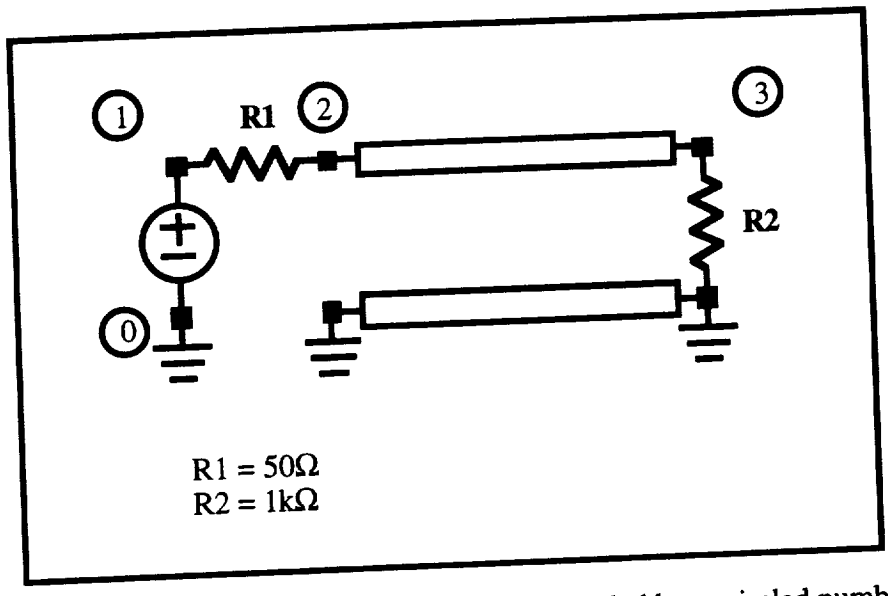


Figure 8.1. Example Circuit 2. Nodes are labeled by encircled numbers.

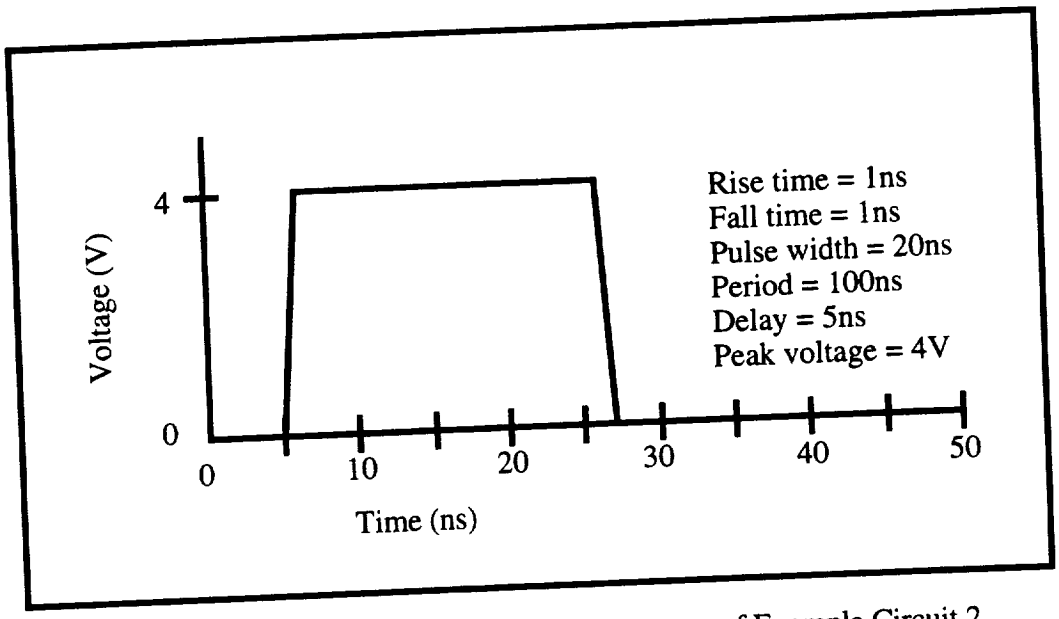


Figure 8.2. Voltage signal generated at source of Example Circuit 2.

8.6.2. Place line specifications in a file

The program vdmdiff supplies the file of difference parameters for transient analysis. Before vdmdiff can supply difference parameters for a transmission line, it requires information about the transmission line. There are two ways to supply the information. In the first method, line specifications are listed in a particular file which vdmdiff will read. In the second method, vdmdiff is supplied the name of a new file.

When a presently nonexistent filename is supplied, vdmdiff will poll the user for the required values and will save the line specifications to the given new filename. Once the line specifications are given, vdmdiff stores the difference parameters in the file a.spr. The contents of a line specification file of the title line1.lin is shown in Figure 8.3.

```
line1.lin              output file
6.7500000e-01  m,          line length
5.3900000e-07  H/m,        distributed inductance
3.9000000e-11  F/m,        distributed capacitance
1.2500000e+02  Ohm/m,      distributed resistance
0.0000000e+00  Ohm/(Hz)^1/2,  skin resistance
0.0000000e+00  S/m,        distributed conductance
7                      order of the approximation
```

Figure 8.3. Transmission line specifications file, line1.lin.

8.6.3. Run vdmdiff

The screen output from a run of vdmdiff is shown in Figure 8.4. Notice that the input file is that given in Section 8.6.2 and the output file name is a.spr.

```
Enter line parameters file name <a.lin>: output file
6.7500000e-01  m,          line length
5.3900000e-07  H/m,        distributed inductance
3.9000000e-11  F/m,        distributed capacitance
1.2500000e+02  Ohm/m,      distributed resistance
0.0000000e+00  Ohm/(Hz)^1/2,  skin resistance
0.0000000e+00  S/m,        distributed conductance
7                      order of the approximation

Do you want to change anything (y/n) ? <n>
Approximating propagation function...

Approximating characteristic admittance...

Job completed.  Results saved into file: n
```

Figure 8.4. Screen dump of a vdmdiff run.

8.6.4. Write SPICE input file

The circuit was defined in Section 8.6.1 and the difference parameter file name was given in Section 8.6.3. With this information known, a SPICE input file can be constructed. The input file, line1_test.in, constructed for the circuit of this section is shown in Figure 8.5. Notice that the lossy transmission line syntax contains only the difference parameter filename and the specification of the initial conditions after the node specification.

```
tx-line transient analysis test circ
v1 1 0      pulse(0 4 5ns 1ns 1ns 20ns 100ns)
r1 1 2      50
r2 3 0       1k
n1 2 0 3 0 filename=a.spr ic=0, 0, 0, 0
.tran 0.1ns 50ns 0.0 0.5ns uic
.end
```

Figure 8.5. SPICE input file, found in spice3e2/src/bin/line1_test.in, for transient analysis of Example Circuit 2.

8.6.5. Perform analysis

Analyzing the circuit is as simple as typing *spice3 line1_test.in* from within the spice3e2/src/bin directory, and typing run at the SPICE prompt. If terminal specifications have been correctly set, SPICE3E2 is capable of displaying the simulation results graphically [10]. The simulation results can also be dumped to a file and displayed using any of the various plotting packages. The voltages at nodes 2 and 3 of the circuit, or the voltages at the input and output ports of the transmission line, are shown in Figures 8.6 and 8.7.

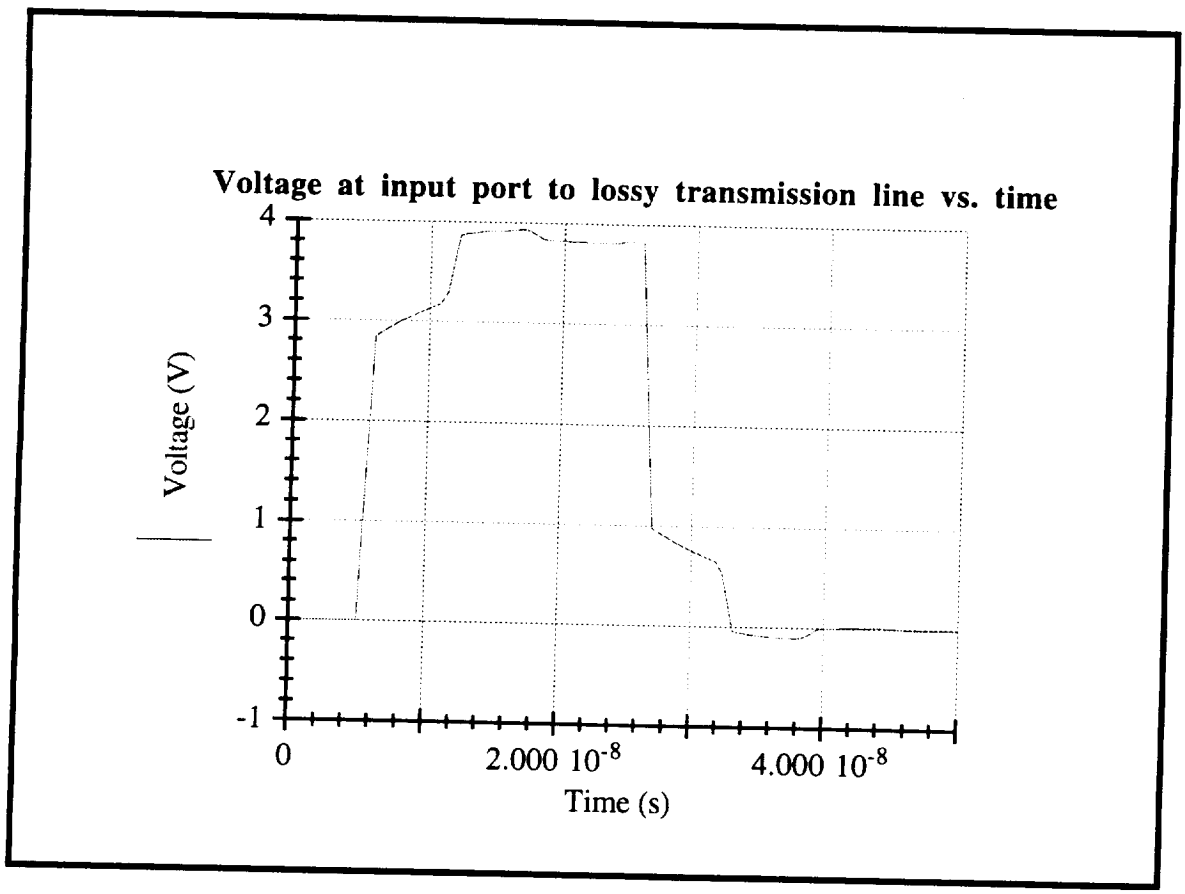


Figure 8.6. Voltage vs. time at input port of lossy transmission line of Example Circuit 2.

8.7. Summary

The installation of a lossy line model to perform transient analysis was detailed in this chapter. The differences between lossy and lossless lines were described as the difference in expressions for the source values appearing on the right-hand side of the stamp. The function which evaluates the source expressions for the lossy model installed in this chapter is called G. It is located in spice3e2/src/lib/dev/ntra/vdmmodel.c. The function G is a parallel, step invariant difference model, and requires several difference parameters. These parameters can be computed for a particular transmission line using vdmdiff found in spice3e2/src/bin. The difference parameters file is read and the values of the parameters are stored in the device specific instance data structure for use in future calls to G. The changes required by incorporating the difference model into the existing lossless line module (the lossless module installed in Chapter 7) can be separated into two parts. The first part deals with calling the function G correctly and incorporating the new

parameters associated with G into the device functions. This is done by hardwiring the difference parameters for one line into the instance structure and making the appropriate changes to device functions. After successful completion of part one, part two is performed, dealing with parsing the new lossy line input file syntax and reading the difference parameters into the instance structure.

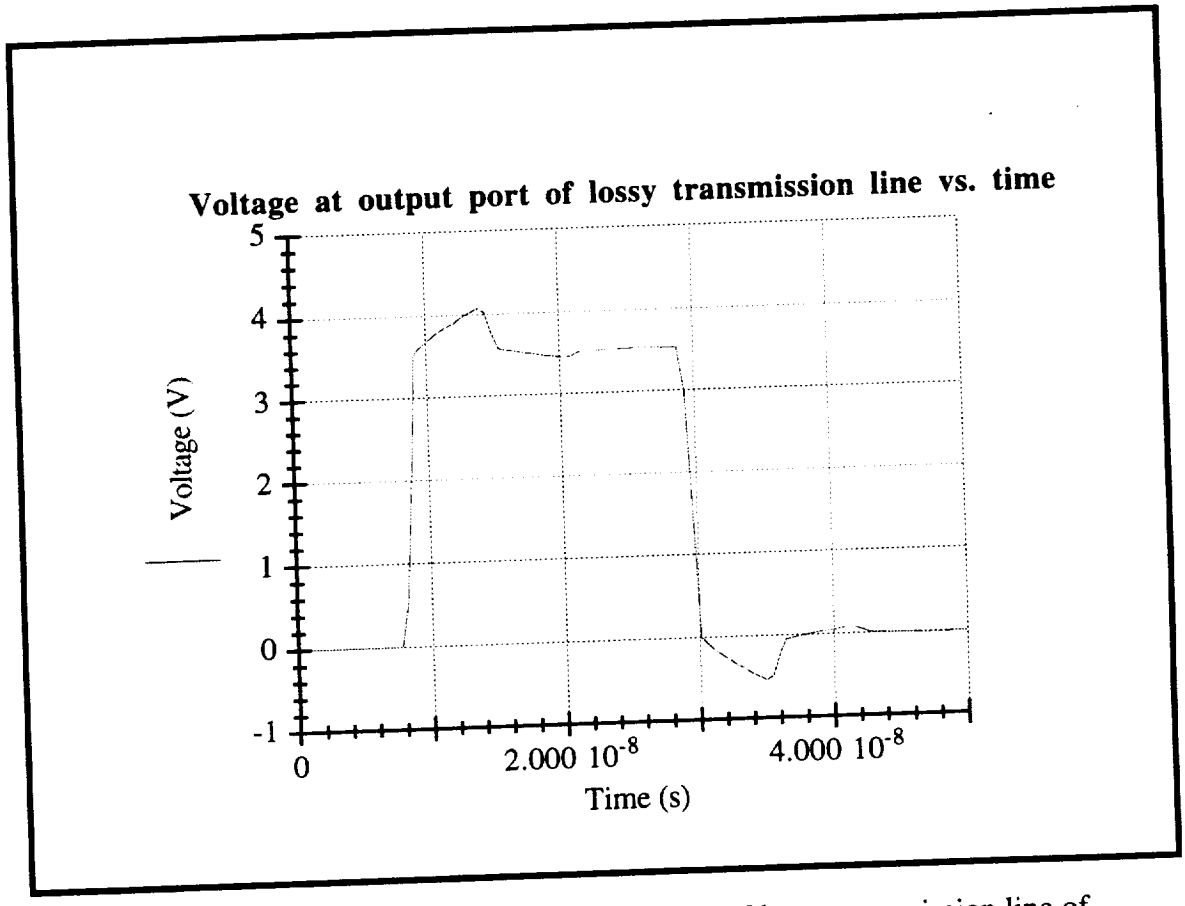


Figure 8.7. Voltage vs. time at output port of lossy transmission line of Example Circuit 2.

9. CONCLUSIONS

The previous chapters stepped through the process of installation into SPICE3E2 for a negative resistor, lossless transmission line transient analysis model and a lossy transmission line transient analysis model. After performing device installations, some further modifications to the code came to mind which would increase the manageability of and therefore extend the life of the source code. The device package changes may be performed on other device packages and not only on the new lossy line package, ntra, which is used as the example package. Before detailing the modifications to increase the manageability of the code, other modification suggestions will be discussed.

9.1. Direct Current and Alternating Current Analysis

There are two more changes to make to the functions in the new lossy line module, ntra, to complete the installation process. The changes involve inserting code to handle the loading of the stamps for dc operating point analysis and ac analysis.

9.1.1. dc analysis

The place to insert the dc stamp is in the NTRAload function, and the precise location in the function is indicated by the third box in the listing of Section B.4.1. The fifth box of the same listing contains code associated with dc analysis, in that it is executed during the very first run of the transient analysis if the initial conditions were obtained by dc operating point analysis. The listing in Section C.6.2 will show that the two sections associated with dc analysis are commented out in the code for the NTRAload function. The steps remaining are to formulate a dc analysis stamp for the lossy transmission line and to insert it into the position for the dc stamp in NTRAload and uncomment and modify the code which is in the fifth box of Section B.4.1.

9.1.2. ac analysis

The function to load the matrix for ac analysis of the lossy transmission line is located in the file spice3e2/src/lib/dev/ntra/ntraacld.c. This file contains the function NTRAacLoad, but the function itself is empty. There is nothing between the opening and closing braces. This is where code is to be written to load the ac stamp into the circuit matrix. The NTRAload function can be used as an example.

9.2. Modifications to Increase Manageability

The major modules of SPICE3E2, such as the parser module and device modules, have good interfacing through which to communicate. An example of this is the relative ease with which a new input line parameter was added in Chapter 8. One of the difficulties in device installation is that interfacing and modularity do not extend into the device packages. In contrast, the input parser package has simpler and more modular functions, and therefore is better organized and easier to read and modify. Another difficulty is that the programmer involved with modifying the code has to be concerned with directly accessing storage data structures. This can be awkward when the storage structure is not straightforward. For example, a programmer interested in modifying the transmission line code has to know how the delay table functions in order to store and retrieve from it. Using the delay table requires use of awkward expressions such as **(here->NTRAdelays + (3*(i-2))+1)*, as in the interpolation section of NTRAload (a complete and up-to-date listing is found in Section D.1). There is no existing function which will simply enter a value in to the delay table at the next entry or which will retrieve a value from the delay table. Another point to consider is that all of the functions in the device package should communicate through the device data structure. This means rewriting the function G so that it extracts the difference parameters directly from the device data structure as opposed to having them passed in the function call.

The modifications will, in summary, make the code more modular, standardize the interface among the routines, and make the code more object oriented by treating data as an entity which can be stored to and retrieved from without consideration of the structure of

the storage data structure. As an example, the function NTRAload will be rewritten using the above modifications.

9.2.1. Modularizing NTRAload

A complete listing of NTRAload is found in Section D.1. The listing is roughly 4 1/3 pages long. The function can be viewed to consist of a few major sections. The first deals with loading the general stamp for transient analysis into the matrix. Second, the dc stamp is loaded if a dc analysis is to be performed. Third, if this is the first iteration of the transient analysis, the values of the sources are calculated from the given initial conditions. Fourth, if this is the first iteration of the transient analysis and initial conditions were not given, the values of the sources are calculated from the results of the dc operating point analysis, again only if the present iteration is the first. Fifth, if the iteration is not the first, the source values are calculated based on interpolation of entries of the delay table. Finally, sixth, the source values are loaded into the right-hand side of the circuit matrix. The rewritten version of NTRAload with a new function added for each major section is listed in Section D.2.2. The modified function listing is only 1 1/2 pages, and a fair amount of this is commenting.

The functions comprising NTRAload are NTRAloadLHS, NTRAdcLoad, NTRAloadUIC, NTRAloadUdc, NTRAinitDelTab, NTRAcalcRHS, NTRAloadRHS, and ckt->CKTgetMode. The following subsections describe the functions. Since the purpose of this chapter is to describe the types of changes to be made and not to list all of the changes to arrive at a more organized package, only one of the functions is explored in depth. This function is NTRAcalcRHS, and the functions comprising it will be described along with functions even farther down the hierarchy. There are two subsections following the section on NTRAcalcRHS which are not directly about the other functions comprising NTRAload.

9.2.2. NTRAloadLHS

This function performs performs matrix loading using the fast matrix pointers in the device structure, such as the code under the comment MOST OF THE STAMP FILLED HERE, in D.1.1. Creating and using this function help to organize the NTRAload function.

9.2.3. NTRAdcLoad

This function uses the fast matrix pointers associated with the stamp of dc analysis to load the dc stamp into the circuit matrix. The code that it replaces is under the comment STAMP FILL FOR DC ANALYSIS.

9.2.4. NTRAloadUIC

NTRAlaodUIC loads the circuit matrix with the user-defined initial conditions and will look like the corresponding piece of code in NTRAload, found in Section D.1 under the comment, USE THE INITIAL CONDITIONS SUPPLIED INSTEAD OF THE DC ANALYSIS VALUES. The new code would be similar except that the call to G will be much cleaner since G would be rewritten to extract the difference parameters from the device data structure.

9.2.5. NTRAloadUdc

The function NTRAloadUdc is similar to the loadUIC function except that the results from the initial transient run should be used. This code would again be much like the corresponding version in NTRAload. The code in the NTRAload listing under USE THE DC VALUES AS START has not been modified for the lossy transmission line; therefore, the lines in the listing would not be the code to appear in loadUdc. Even if the code in the listing of NTRAload were updated, the code in loadUdc would not appear the same since the values from a previous solution will be extracted in a different fashion as will be pointed out in Section 9.2.9.

9.2.6. NTRAinitDelTab

NTRAinitDelTab stands for initialize the delay table. As will be seen in following functions, specific functions will exist to interface to the delay table and so the code making entries into the delay table will look different than the listing right below SET UP DELAY TABLE in D.1. The first two sets of entries will, however, be initialized in the same manner.

9.2.7. NTRAcalcRHS

This function which is listed in C.3 obtains the interpolated values of excitation from the delay table, calls G, and does the necessary bookkeeping for the next set of calls to G (which is the next time that NTRAcalcRHS is called). The fields NTRAxyold, NTRAxw1, and NTRAxw2 have been added to the device data structure in order to call G with the instance structure as an interface. The function NTRAgetInterpExcit1 returns a value as opposed to setting NTRAxw1(2) inside itself. This is purely a matter of taste; the interfacing for this function was chosen as such because it is more natural to have a get function return a value. NTRAgetInterpExcit is the subject of the next subsection.

9.2.8. NTRAgetInterpExcit1

This function, like the counterpart function NTRAgetInterpExcit2, obtains an interpolated value of the excitation function associated with a source (1 or 2). The second-order interpolation is performed much like it is in the original version of NTRAload, except that the manner in which values are retrieved from the delay table is different. The function NTRAgetDelTabIndGrtr returns the index of the delay table entry, which is the first value greater than the time argument passed to it in a search from smallest value to largest value. The contents of NTRAgetDelTabIndGrtr is a for loop very much like that shown below the comment FIND INTERPOLATED VALUES in the listing of C.1. NTRAgetDelTabIndGrtr is listed in Section C.5.

9.2.9. CKTgetSol

The function CKTgetSol used in NTRAcalcRHS is used to treat the circuit matrix in the same fashion as the delay table. CKTgetSol uses the index passed to it to retrieve the correct value from the solution vector. The code for CKTgetSol would be implemented as *return *(ckt->CKTrhsOld + index_passed)*. This is a one-line function and since ckt->CKTgetSol would be called in several device packages, the total function call overhead associated with CKTgetSol will be high, and may cause significant slowdown of the code. One way to combat this is to create a macro that returns the solutions of the previous iteration. It could be called CKT_GET_SOL and declared as follows: *#define*

*CKT_GET_SOL(a) *(ckt->CKTrhsOld + a).* The use of a macro would still allow the data to appear as an object, the modifications' programmer would not have to deal with pointers to the circuit matrix, and there would be no function call since the compiler would expand CKT_GET_SOL before the actual code was compiled.

9.2.10. NTRAloadRHS

This function loads the values of the sources found by the previous function into the right-hand side of the circuit matrix. Once again, a function to insert values into the matrix is needed. This function could also be written as a macro.

9.2.11. CKTgetMode

This function is used in the modified version of NTRAload and returns a flag which can be used to decide what the circuit analysis mode is.

9.3. Summary

The previous chapters detailed the installation process of three different device models into SPICE3E2. After the installation of the lossy line, some ideas were formulated on making the source code more manageable. These ideas of increasing modularity, increasing object orientation, and standardizing the interface in the code were presented in this chapter. An example rewrite of the NTRAload function illustrated these concepts.

APPENDIX A.
RESISTOR CODE

This appendix contains code and code fragments from unmodified and modified codes connected with the installation of a negative resistor.

A.1. Device Specific Files

This section contains excerpts from the files nresload.c and nres.c before and after the name change modifications discussed in Section 6.1.3. The files are in the directory spice3e2/src/lib/dev/nres.

A.1.1. Contents of nresload.c before renaming

```
#include "spice.h"
#include <stdio.h>
#include "cktdefs.h"
#include "resdefs.h"
#include "sperror.h"
#include "suffix.h"

 /*ARGSUSED*/
 int
 RESload(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
      /* actually load the current resistance value into the
       * sparse matrix previously provided
       */
 {
```

```
    register RESmodel *model = (RESmodel *)inModel;
    register RESinstance *here;

    /*  loop through all the resistor models */
    for( ; model != NULL; model = model->RESnextModel ) {

      /* loop through all the instances of the model */
      for (here = model->RESinstances; here != NULL ;
            here=here->RESnextInstance) {

        *(here->RESposPosptr) += here->RESconduct;
        *(here->RESnegNegptr) += here->RESconduct;
        *(here->RESposNegptr) -= here->RESconduct;
        *(here->RESnegPosptr) -= here->RESconduct;
      }
    }
    return(OK);
}
```

A.1.2. Contents of nresload.c after renaming

```
#include "spice.h"
#include <stdio.h>
#include "cktdefs.h"
#include "nresdefs.h"
#include "sperror.h"
#include "suffix.h"


/*ARGSUSED*/
int
NRESload(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
      /* actually load the current nresistance value into the
       * sparse matrix previously provided
       */
{
    register NRESmodel *model = (NRESmodel *)inModel;
    register NRESinstance *here;

    /*  loop through all the nresistor models */
    for( ; model != NULL; model = model->NRESnextModel ) {

      /* loop through all the instances of the model */
      for (here = model->NRESinstances; here != NULL ;
            here=here->NRESnextInstance) {

        *(here->NRESposPosptr) += here->NRESconduct;
        *(here->NRESnegNegptr) += here->NRESconduct;
```

```
            *(here->NRESposNegptr) -= here->NREStconduct;
            *(here->NRESnegPosptr) -= here->NREStconduct;
        }
    }
    return(OK);
}
```

A.1.3. Contents of nres.c before renaming

```
#include "spice.h"
#include <stdio.h>
#include "resdefs.h"
#include "devdefs.h"
#include "ifsim.h"
#include "suffix.h"


IFparm RESpTable[] = { /* parameters */
 IOP( "resistance", RES_RESIST, IF_REAL,"Resistance"),
 IOP( "w",          RES_WIDTH,  IF_REAL,"Width"),
 IOP( "l",          RES_LENGTH, IF_REAL,"Length"),
 IOP( "c",          RES_CURRENT,IF_REAL,"Current"),
 IOP( "p",          RES_POWER,  IF_REAL,"Power"),
 IP("sens_resist", RES_RESIST_SENS, IF_FLAG,
     "flag to request sensitivity WRT resistance"),
 OP("sens_dc",  RES_QUEST_SENS_DC,  IF_REAL,    "dc sensitivity "),
 OP("sens_real",RES_QUEST_SENS_REAL,IF_REAL,
     "dc sensitivity and real part of ac sensitivity"),
 OP("sens_imag",RES_QUEST_SENS_IMAG,IF_REAL,
     "dc sensitivity and imag part of ac sensitivity"),
 OP("sens_mag", RES_QUEST_SENS_MAG, IF_REAL,    "ac sensitivity of magnitude"),
 OP("sens_ph",  RES_QUEST_SENS_PH,  IF_REAL,    "ac sensitivity of phase"),
 OP("sens_cplx",RES_QUEST_SENS_CPLX,IF_COMPLEX,    "ac sensitivity"),
 IOP("temp",        RES_TEMP,   IF_REAL,"Instance operating temperature"),
} ;
IFparm RESmPTable[] = { /* model parameters */
```

```
IOP( "tc1",   RES_MOD_TC1,     IF_REAL,"First order temp. coefficient"),
IOP( "tc2",   RES_MOD_TC2,     IF_REAL,"Second order temp. coefficient"),
IOP( "rsh",   RES_MOD_RSH,     IF_REAL,"Sheet resistance"),
IOP( "defw",  RES_MOD_DEFWIDTH, IF_REAL,"Default device width"),
IP( "r",      RES_MOD_R,       IF_FLAG,"Device is a resistor model"),
IOP( "narrow", RES_MOD_NARROW,  IF_REAL,"Narrowing of resistor"),
IOP("tnom",   RES_MOD_TNOM,    IF_REAL,"Parameter measurement temperature"),
};
char *RESnames[] = {
    "R+",
    "R-"
};

int RESnSize = NUMELEMS(RESnames);
int RESpTSize = NUMELEMS(RESpTable);
int RESmPTSize = NUMELEMS(RESmPTable);
int RESiSize = sizeof(RESinstance);
int RESmSize = sizeof(RESmodel);
```

A.1.4. Contents of nres.c after renaming

```
#include "spice.h"
#include <stdio.h>
#include "nresdefs.h"
#include "devdefs.h"
#include "ifsim.h"
#include "suffix.h"


IFparm NRESpTable[] = { /* parameters */
 IOP( "nresistance", NRES_RESIST, IF_REAL,"Nresistance"),
 IOP( "w",         NRES_WIDTH,  IF_REAL,"Width"),
 IOP( "l",         NRES_LENGTH, IF_REAL,"Length"),
 IOP( "c",         NRES_CURRENT,IF_REAL,"Current"),
 IOP( "p",         NRES_POWER,  IF_REAL,"Power"),
 IP("sens_resist", NRES_RESIST_SENS, IF_FLAG,
      "flag to request sensitivity WRT resistance"),
 OP("sens_dc",  NRES_QUEST_SENS_DC,  IF_REAL,    "dc sensitivity "),
 OP("sens_real",NRES_QUEST_SENS_REAL,IF_REAL,
      "dc sensitivity and real part of ac sensitivity"),
 OP("sens_imag",NRES_QUEST_SENS_IMAG,IF_REAL,
      "dc sensitivity and imag part of ac sensitivity"),
```

```
OP("sens_mag", NRES_QUEST_SENS_MAG, IF_REAL,   "ac sensitivity of
magnitude"),
OP("sens_ph",  NRES_QUEST_SENS_PH,  IF_REAL,   "ac sensitivity of
phase"),
OP("sens_cplx",NRES_QUEST_SENS_CPLX,IF_COMPLEX,   "ac
sensitivity"),
IOP("temp",      NRES_TEMP,  IF_REAL,"Instance operating temperature"),
} ;
IFparm NRESmPTable[] = { /* model parameters */
IOP( "tc1",   NRES_MOD_TC1,     IF_REAL,"First order temp. coefficient"),
IOP( "tc2",   NRES_MOD_TC2,     IF_REAL,"Second order temp. coefficient"),
IOP( "rsh",   NRES_MOD_RSH,     IF_REAL,"Sheet resistance"),
IOP( "defw",  NRES_MOD_DEFWIDTH, IF_REAL,"Default device width"),
IP( "r",    NRES_MOD_R,       IF_FLAG,"Device is an nresistor model"),
IOP( "narrow", NRES_MOD_NARROW,  IF_REAL,"Narrowing of nresistor"),
IOP("tnom",   NRES_MOD_TNOM,    IF_REAL,"Parameter measurement
temperature"),
};

char *NRESnames[] = {
   "N+",
   "N-"
};

int     NRESnSize = NUMELEMS(NRESnames);
int     NRESpTSize = NUMELEMS(NRESpTable);
int     NRESmPTSize = NUMELEMS(NRESmPTable);
int     NRESiSize = sizeof(NRESinstance);
int     NRESmSize = sizeof(NRESmodel);
```

A.2. Device Header Files

The device header files for the negative resistor are nresdefs.h, nresext.h, and nresitf.h. These are located in the directory spice3e2/src/include. Since all of the code in Sections A.2.1, A.2.2, and A.2.3 have been renamed, the variables and function references found within indicate association with the negative resistor.

A.2.1. nresdefs.h after renaming

```
#ifndef NRES
#define NRES

#include "ifsim.h"
#include "cktdefs.h"
#include "gendefs.h"
#include "complex.h"
#include "noisedef.h"
```

```
        /* definitions used to describe nresistors */


    /* information used to describe a single instance */

    typedef struct sNRESinstance {
       struct sNRESmodel *NRESmodPtr;   /* backpointer to model */
       struct sNRESinstance *NRESnextInstance;  /* pointer to next instance of
                                    * current model*/

       IFuid NRESname;  /* pointer to character string naming this instance */

       int NRESposNode; /* number of positive node of nresistor */
       int NRESnegNode; /* number of negative node of nresistor */

       double NREStemp;    /* temperature at which this nresistor operates */
       double NRESconduct;  /* conductance at current analysis temperature */
       double NRESresist;   /* nresistance at temperature Tnom */
       double NRESwidth;    /* width of the nresistor */
       double NRESlength;   /* length of the nresistor */
       double *NRESposPosptr;   /* pointer to sparse matrix diagonal at
                         * (positive,positive) */
       double *NRESnegNegptr;   /* pointer to sparse matrix diagonal at
                         * (negative,negative) */
       double *NRESposNegptr;   /* pointer to sparse matrix offdiagonal at
                         * (positive,negative) */
       double *NRESnegPosptr;   /* pointer to sparse matrix offdiagonal at
                         * (negative,positive) */
       unsigned NRESresGiven : 1;  /* flag to indicate nresistance was specified */
       unsigned NRESwidthGiven  : 1;   /* flag to indicate width given */
       unsigned NRESlengthGiven : 1;   /* flag to indicate length given */
       unsigned NREStempGiven   : 1;   /* indicates temperature specified */
       int   NRESsenParmNo;       /* parameter # for sensitivity use;
               set equal to  0 if not a design parameter*/
    #ifndef NONOISE
      double NRESnVar[NSTATVARS];
    #else /* NONOISE */
            double *NRESnVar;
    #endif /* NONOISE */

    } NRESinstance ;


    /* per model data */

    typedef struct sNRESmodel {      /* model structure for a nresistor */
       int NRESmodType; /* type index of this device type */
       struct sNRESmodel *NRESnextModel; /* pointer to next possible model in
                            * linked list */
       NRESinstance * NRESinstances; /* pointer to list of instances that have this
                            * model */
       IFuid NRESmodName;      /* pointer to character string naming this model */
```

```
    double NREStnom;       /* temperature at which nresistance measured */
    double NREStempCoeff1;  /* first temperature coefficient of nresistors */
    double NREStempCoeff2;  /* second temperature coefficient of nresistors */
    double NRESsheetRes;    /* sheet resistance of devices in ohms/square */
    double NRESdefWidth;    /* default width of an nresistor */
    double NRESnarrow;      /* amount by which device is narrower than drawn */
    unsigned NREStnomGiven: 1;  /* flag to indicate nominal temp. was given */
    unsigned NREStc1Given : 1;  /* flag to indicate tc1 was specified */
    unsigned NREStc2Given : 1;  /* flag to indicate tc2 was specified */
    unsigned NRESsheetResGiven  :1; /* flag to indicate sheet resistance given*/
    unsigned NRESdefWidthGiven  :1; /* flag to indicate default width given */
    unsigned NRESnarrowGiven    :1; /* flag to indicate narrow effect given */
} NRESmodel;

/* device parameters */
#define NRES_RESIST 1
#define NRES_WIDTH 2
#define NRES_LENGTH 3
#define NRES_CONDUCT 4
#define NRES_RESIST_SENS 5
#define NRES_CURRENT 6
#define NRES_POWER 7
#define NRES_TEMP 8

/* model parameters */
#define NRES_MOD_TC1 101
#define NRES_MOD_TC2 102
#define NRES_MOD_RSH 103
#define NRES_MOD_DEFWIDTH 104
#define NRES_MOD_NARROW 105
#define NRES_MOD_R 106
#define NRES_MOD_TNOM 107

/* device questions */
#define NRES_QUEST_SENS_REAL    201
#define NRES_QUEST_SENS_IMAG    202
#define NRES_QUEST_SENS_MAG     203
#define NRES_QUEST_SENS_PH      204
#define NRES_QUEST_SENS_CPLX    205
#define NRES_QUEST_SENS_DC      206

/* model questions */

#include "nresext.h"

#endif /*NRES*/
```

A.2.2. nresext.h after renaming

```
#ifdef __STDC__
extern int NRESask(CKTcircuit*,GENinstance*,int,IFvalue*,IFvalue*);
extern int NRESdelete(GENmodel*,IFuid,GENinstance**);
```

```
extern void NRESdestroy(GENmodel**);
extern int NRESload(GENmodel*,CKTcircuit*);
extern int NRESmodAsk(CKTcircuit*,GENmodel*,int,IFvalue*);
extern int NRESmDelete(GENmodel**,IFuid,GENmodel*);
extern int NRESmParam(int,IFvalue*,GENmodel*);
extern int NRESparam(int,IFvalue*,GENinstance*,IFvalue*);
extern int NRESpzLoad(GENmodel*,CKTcircuit*,SPcomplex*);
extern int NRESsAcLoad(GENmodel*,CKTcircuit*);
extern int NRESsLoad(GENmodel*,CKTcircuit*);
extern int NRESsSetup(SENstruct*,GENmodel*);
extern void NRESsPrint(GENmodel*,CKTcircuit*);
extern int NRESsetup(SMPmatrix*,GENmodel*,CKTcircuit*,int*);
extern int NREStemp(GENmodel*,CKTcircuit*);
extern int NRESnoise(int,int,GENmodel*,CKTcircuit*,Ndata*,double*);
#else /* stdc */
extern int NRESask();
extern int NRESdelete();
extern void NRESdestroy();
extern int NRESload();
extern int NRESmodAsk();
extern int NRESmDelete();
extern int NRESmParam();
extern int NRESparam();
extern int NRESpzLoad();
extern int NRESsAcLoad();
extern int NRESsLoad();
extern int NRESsSetup();
extern void NRESsPrint();
extern int NRESsetup();
extern int NREStemp();
extern int NRESnoise();
#endif /* stdc */
```

A.2.3. nresitf.h after renaming

```
#ifdef DEV_nres

#ifndef DEV_NRES
#define DEV_NRES

#include "nresext.h"
extern IFparm NRESpTable[ ];
extern IFparm NRESmPTable[ ];
extern char *NRESnames[ ];
extern int NRESpTSize;
extern int NRESmPTSize;
extern int NRESnSize;
extern int NRESiSize;
extern int NRESmSize;

SPICEdev NRESinfo = {
    {
```

```
        "Nresistor",
        "Simple linear negative resistor",

        &NRESnSize,
        &NRESnSize,
        NRESnames,

        &NRESpTSize,
        NRESpTable,

        &NRESmPTSize,
        NRESmPTable,
      },

      NRESparam,
      NRESmParam,
      NRESload,
      NRESsetup,
      NRESsetup,
      NREStemp,
      NULL,
      NULL,
      NRESload,  /* ac load and normal load are identical */
      NULL,
      NRESdestroy,
#ifdef DELETES
      NRESmDelete,
      NRESdelete,
#else /* DELETES */
      NULL,
      NULL,
#endif /* DELETES */
      NULL,
      NRESask,
      NULL,
#ifdef AN_pz
      NRESpzLoad,
#else /* AN_pz */
      NULL,
#endif /* AN_pz */
      NULL,
#ifdef AN_sense
      NRESsSetup,
      NRESsLoad,
      NULL,
      NRESsAcLoad,
      NRESsPrint,
      NULL,
#else /* AN_sense */
      NULL,
      NULL,
      NULL,
      NULL,
      NULL,
```

```
    NULL,
#endif /* AN_sense */
    NULL, /* Disto */
#ifdef AN_noise
    NRESnoise,
#else   /* AN_noise */
    NULL,
#endif  /* AN_noise */

    &NRESiSize,
    &NRESmSize

};

#endif
#endif
```

A.3. INP2N

This section contains code for the unmodified and modified versions of the function INP2N found in the file spice3e2/src/lib/inp/inp2n.c.

A.3.1. Contents of inp2n.c before renaming

```
#include "spice.h"
#include <stdio.h>
#include "ifsim.h"
#include "inpdefs.h"
#include "inpmacs.h"
#include "fteext.h"
#include "suffix.h"
void
INP2R(ckt,tab,current)
    GENERIC *ckt;
    INPtables *tab;
    card *current;
```

```
{
/* parse a resistor card */
/* Rname <node> <node> [<val>][<mname>][w=<val>][l=<val>] */

int mytype; /* the type we determine resistors are */
int type;  /* the type the model says it is */
char *line; /* the part of the current line left to parse */
char *name; /* the resistor's name */
char *model;   /* the name of the resistor's model */
char *nname1;  /* the first node's name */
char *nname2;  /* the second node's name */
GENERIC *node1; /* the first node's node pointer */
GENERIC *node2; /* the second node's node pointer */
double val;    /* temp to held resistance */
int error;     /* error code temporary */
int error1;    /* secondary error code temporary */
INPmodel *thismodel;   /* pointer to model structure describing our model */
GENERIC *mdfast;   /* pointer to the actual model */
GENERIC *fast;  /* pointer to the actual instance */

IFvalue ptemp;  /* a value structure to package resistance into */
char *nname2;  /* the second node's name */
GENERIC *node1; /* the first node's node pointer */
GENERIC *node2; /* the second node's node pointer */
double val;    /* temp to held resistance */
int error;     /* error code temporary */
int error1;    /* secondary error code temporary */
INPmodel *thismodel;   /* pointer to model structure describing our model */
GENERIC *mdfast;   /* pointer to the actual model */
GENERIC *fast;  /* pointer to the actual instance */
IFvalue ptemp;  /* a value structure to package resistance into */
int waslead;   /* flag to indicate that funny unlabeled number was found */
double leadval; /* actual value of unlabeled number */
IFuid uid;     /* uid for default model */

  mytype = INPtypelook("Resistor");
  if(mytype < 0 ) {
    LITERR("Device type Resistor not supported by this binary\n")
    return;
  }
  line = current->line;
  INPgetTok(&line,&name,1);
  INPinsert(&name,tab);
  INPgetTok(&line,&nname1,1);
  INPtermInsert(ckt,&nname1,tab,&node1);
  INPgetTok(&line,&nname2,1);
  INPtermInsert(ckt,&nname2,tab,&node2);
  val = INPevaluate(&line,&error1,1);
```

```
/* either not a number -> model, or
 * follows a number, so must be a model name
 * -> MUST be a model name (or null)
 */
INPgetTok(&line,&model,1);
if(*model) { /* token isn't null */
   INPinsert(&model,tab);
   thismodel = (INPmodel *)NULL;
   current->error = INPgetMod(ckt,model,&thismodel,tab);
   if(thismodel != NULL) {
      if(mytype != thismodel->INPmodType) {
         LITERR("incorrect model type")
         return;
      }
      mdfast = thismodel->INPmodfast;
      type = thismodel->INPmodType;
   } else {
      type = mytype;
      if(!tab->defRmod) {
         /* create default R model */
         IFnewUid(ckt,&uid,(IFuid)NULL,"R",UID_MODEL,(GENERIC**)NULL);
         IFC(newModel, (ckt,type,&(tab->defRmod),uid))
      }
      mdfast = tab->defRmod;
   }
   IFC(newInstance,(ckt,mdfast,&fast,name))
} else {
   type = mytype;
   if(!tab->defRmod) {
      /* create default R model */
      IFnewUid(ckt,&uid,(IFuid)NULL,"R",UID_MODEL,(GENERIC**)NULL);
      IFC(newModel, (ckt,type,&(tab->defRmod),uid))
   }
   IFC(newInstance,(ckt,tab->defRmod,&fast,name))
}
if(error1 == 0) { /* got a resistance above */
   ptemp.rValue = val;
   GCA(INPpName,("resistance",&ptemp,ckt,type,fast))
}
IFC(bindNode,(ckt,fast,1,node1))
IFC(bindNode,(ckt,fast,2,node2))
PARSECALL((&line,ckt,type,fast,&leadval,&waslead,tab))
if(waslead) {
   ptemp.rValue = leadval;
   GCA(INPpName,("resistance",&ptemp,ckt,type,fast))
}
return;
}
```

A.3.2. Contents of inp2n.c after renaming

```
#include "spice.h"
#include <stdio.h>
#include "ifsim.h"
#include "inpdefs.h"
#include "inpmacs.h"
#include "fteext.h"
#include "suffix.h"

void
INP2N(ckt,tab,current)
  GENERIC *ckt;
  INPtables *tab;
  card *current;

{
/* parse a negative resistor card */
/* Nname <node> <node> [<val>][<mname>][w=<val>][l=<val>] */

int mytype; /* the type we determine nresistors are */
int type;  /* the type the model says it is */
char *line; /* the part of the current line left to parse */
char *name; /* the nresistor's name */
char *model;   /* the name of the nresistor's model */
char *nname1;  /* the first node's name */
char *nname2;  /* the second node's name */
GENERIC *node1; /* the first node's node pointer */
GENERIC *node2; /* the second node's node pointer */
double val;    /* temp to held nresistance */
int error;     /* error code temporary */
int error1;    /* secondary error code temporary */
INPmodel *thismodel;   /* pointer to model structure describing our model */
GENERIC *mdfast;   /* pointer to the actual model */
GENERIC *fast;  /* pointer to the actual instance */
IFvalue ptemp;  /* a value structure to package nresistance into */
int waslead;   /* flag to indicate that funny unlabeled number was found */
double leadval; /* actual value of unlabeled number */
IFuid uid;     /* uid for default model */

   mytype = INPtypelook("Nresistor");
   if(mytype < 0 ) {
     LITERR("Device type Nresistor not supported by this binary\n")
     return;
   }
   line = current->line;
   INPgetTok(&line,&name,1);
   INPinsert(&name,tab);
   INPgetTok(&line,&nname1,1);
   INPtermInsert(ckt,&nname1,tab,&node1);
   INPgetTok(&line,&nname2,1);
   INPtermInsert(ckt,&nname2,tab,&node2);
   val = INPevaluate(&line,&error1,1);
```

```
    /* either not a number -> model, or
     * follows a number, so must be a model name
     * -> MUST be a model name (or null)
     */
    INPgetTok(&line,&model,1);
    if(*model) { /* token isn't null */
      INPinsert(&model,tab);
      thismodel = (INPmodel *)NULL;
      current->error = INPgetMod(ckt,model,&thismodel,tab);
      if(thismodel != NULL) {
        if(mytype != thismodel->INPmodType) {
          LITERR("incorrect model type")
          return;
        }
        mdfast = thismodel->INPmodfast;
        type = thismodel->INPmodType;
      } else {
        type = mytype;
        if(!tab->defNmod) {
          /* create default N model */

IFnewUid(ckt,&uid,(IFuid)NULL,"N",UID_MODEL,(GENERIC**)NULL);
          IFC(newModel, (ckt,type,&(tab->defNmod),uid))
        }
        mdfast = tab->defNmod;
      }
      IFC(newInstance,(ckt,mdfast,&fast,name))
    } else {
      type = mytype;
      if(!tab->defNmod) {
        /* create default N model */

IFnewUid(ckt,&uid,(IFuid)NULL,"N",UID_MODEL,(GENERIC**)NULL);
        IFC(newModel, (ckt,type,&(tab->defNmod),uid))
      }
      IFC(newInstance,(ckt,tab->defNmod,&fast,name))
    }
    if(error1 == 0) { /* got a nresistance above */
      ptemp.rValue = val;
      GCA(INPpName,("nresistance",&ptemp,ckt,type,fast))
    }

    IFC(bindNode,(ckt,fast,1,node1))
    IFC(bindNode,(ckt,fast,2,node2))
    PARSECALL((&line,ckt,type,fast,&leadval,&waslead,tab))
    if(waslead) {
      ptemp.rValue = leadval;
      GCA(INPpName,("nresistance",&ptemp,ckt,type,fast))
    }
    return;
}
```

A.4. Parser Header File

The listing below consists of excerpts from the file inpdefs.h in the spice3e2/src/lib/include directory.

```
.
.
.
   GENERIC *defJmod;
   GENERIC *defKmod;
   GENERIC *defLmod;
   GENERIC *defMmod;
   GENERIC *defNmod;
   GENERIC *defOmod;
   GENERIC *defPmod;
   GENERIC *defQmod;
   GENERIC *defRmod;
.
.
.
  void INP2L(GENERIC*,INPtables*,card*);
  void INP2M(GENERIC*,INPtables*,card*);
  void INP2N(GENERIC*,INPtables*,card*);
  void INP2O(GENERIC*,INPtables*,card*);
  void INP2Q(GENERIC*,INPtables*,card*,GENERIC*);
  void INP2R(GENERIC*,INPtables*,card*);
  void INP2S(GENERIC*,INPtables*,card*);
  void INP2T(GENERIC*,INPtables*,card*);
  void INP2U(GENERIC*,INPtables*,card*);
.
.
.
  void INP2M();
  void INP2N();
  void INP2O();
  void INP2Q();
  void INP2R();
  void INP2S();
  void INP2T();
  void INP2U();
.
.
.
```

A.5. INPpas2

In this section excerpts from the function INPpas2, found in file spice3e2/src/lib/inp/inppas2.c are listed. The listing is for INPpas2 after modification.

```
.
.
.
        c = *(current->line);
        c = islower(c) ? toupper(c) : c;

    switch(c) {
.
.
.


      case 'R':   /* Rname <node> <node> [<val>][<mname>][w=<val>][l=<val>] */
        INP2R(ckt,tab,current);
        break;
      case 'N':   /* Nname <node> <node> [<val>][<mname>][w=<val>][l=<val>] */
        INP2N(ckt,tab,current);
        break;
      case 'C':   /* Cname <node> <node> <val> [IC=<val>] */
        INP2C(ckt,tab,current);
        break;
      case 'L':   /* Lname <node> <node> <val> [IC=<val>] */
        INP2L(ckt,tab,current);
        break;
      case 'G':   /* Gname <node> <node> <node> <node> <val> */
        INP2G(ckt,tab,current);
        break;
.
.
.


          default:
            /* the un-implemented device */
            LITERR(" unknown device type - error \n")
        break;
      }
    }
.
.
.
```

A.6. Main Parsing Routine

This section contains partial listings for two simulator files spice3e2/src/lib/bin/bconf.c and spice3e2/src/lib/fte/subckt.c. after modification.

A.6.1. Contents of bconf.c after modification

```
.
.
.
 #define DEV_dio
 #define DEV_ind
 #define DEV_isrc
 #define DEV_mos1
 #define DEV_mos2
 #define DEV_res
 #define DEV_nres
 #define DEV_vccs
.
.
.
 #include "isrcitf.h"
 #include "mos1itf.h"
 #include "mos6itf.h"
 #include "resitf.h"
 #include "nresitf.h"
 #include "switf.h"
 #include "vccsitf.h"
 #include "vcvsitf.h"
 #include "vsrcitf.h"

.
.
.

 #ifdef DEV_mos6
      &MOS6info,
 #endif
 #ifdef DEV_res
      &RESinfo,
 #endif
 #ifdef DEV_nres
      &NRESinfo,
 #endif
 #ifdef DEV_sw
      &SWinfo,
 #endif
.
.
```

A.6.2. Contents of subckt.c after modification

```
.
.
.

int
inp_numnodes(c)
   char c;
{
   if (isupper(c))
      c = tolower(c);
   switch (c) {
      case ' ':
      case '\t':
      case '.':
      case 'x':
      case '*':
      return (0);

      case 'b': return (2);
      case 'c': return (2);
      case 'd': return (2);
      case 'e': return (4);
      case 'f': return (2);
      case 'g': return (4);
      case 'h': return (2);
      case 'i': return (2);
      case 'j': return (3);
      case 'k': return (0);
      case 'l': return (2);
      case 'm': return (4);
      case 'o': return (4);
      case 'q': return (4);
      case 'r': return (2);
      case 'n': return (2);
      case 's': return (4);
      case 't': return (4);

      case 'u': return (3);
      case 'v': return (2);
      case 'w': return (3);
      case 'z': return (3);

      default:
```

```
            fprintf(cp_err, "Warning: unknown dev type ->subckt.c: %c\n", c);
            return (2);
        }
    }
```

A.7. Files Used by Make

The listings in this sections are examples of the files utilized by the UNIX make command when compiling SPICE3E2. Section A.7.1 contains the listing of spice3e2/src/lib/dev/nres/makedefs before modifcation. Section A.7.2 contains the listing of spice3e2/src/lib/dev/nres/msc51.bat. Section A.7.3 contains an excerpt from spice3e2/conf/defaults, and the final subsection contains an excerpt from the file spice3e2/src/lib/inp/response.lib.

A.7.1. Contents of makedefs before modification

```
CFILES      = res.c resask.c resdel.c resdest.c resload.c resmask.c \
              resmdel.c resmpar.c resnoise.c resparam.c respzld.c \
              ressacl.c ressetup.c ressload.c ressprt.c ressset.c \
              restemp.c

COBJS       = res.o resask.o resdel.o resdest.o resload.o resmask.o \
              resmdel.o resmpar.o resnoise.o resparam.o respzld.o \
              ressacl.o ressetup.o ressload.o ressprt.o ressset.o \
              restemp.o

MODULE          = res
LIBRARY         = dev
MODULE_TARGET   = $(OBJLIB_DIR)/$(MODULE)

NUMBER          = 1
```

A.7.2. Contents of msc51.bat before modification

```
cl /I..\..\..\include /c res.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resask.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resdel.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resdest.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resload.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resmask.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resmdel.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resmpar.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resnoise.c >> ..\..\..\msc.out
cl /I..\..\..\include /c resparam.c >> ..\..\..\msc.out
cl /I..\..\..\include /c respzld.c >> ..\..\..\msc.out
cl /I..\..\..\include /c ressacl.c >> ..\..\..\msc.out
cl /I..\..\..\include /c ressetup.c >> ..\..\..\msc.out
cl /I..\..\..\include /c ressload.c >> ..\..\..\msc.out
cl /I..\..\..\include /c ressprt.c >> ..\..\..\msc.out
cl /I..\..\..\include /c ressset.c >> ..\..\..\msc.out
cl /I..\..\..\include /c restemp.c >> ..\..\..\msc.out
lib ..\..\dev1.lib @response.lib
```

A.7.3. Excerpt from defaults after modification

```
.
.
.
DEVICES = asrc bjt cap cccs ccvs csw dio ind isrc jfet ltra \
          mes mos1 mos2 mos3 mos6 res nres sw tra urc vccs \
          vcvs vsrc
.
.
.
```

A.7.4. Excerpt from response.lib after modication

```
.
.
.
 +inp2k.obj&
 +inp2l.obj&
 +inp2m.obj&
 +inp2n.obj&
 +inp2o.obj&
 +inp2q.obj&
 +inp2r.obj&
 +inp2s.obj&

.
.
.
```

A.8. NRESload

This section revisits the code for the function NRESload found in spice3e2/src/lib/dev/nres/nresload.c. Section A.8.1 is the code as last seen renamed. The code to be changed is boxed. The function modified to load the stamp of a negative resistor is shown in Section A.8.2.

A.8.1. Excerpt from nresload.c before modification

```
.
.
.
/*ARGSUSED*/
int
NRESload(inModel,ckt)
   GENmodel *inModel;
   CKTcircuit *ckt;
      /* actually load the current nresistance value into the
```

```
     * sparse matrix previously provided
     */
{
  register NRESmodel *model = (NRESmodel *)inModel;
  register NRESinstance *here;
  /*  loop through all the nresistor models */
  for( ; model != NULL; model = model->NRESnextModel ) {

    /* loop through all the instances of the model */
    for (here = model->NRESinstances; here != NULL ;
        here=here->NRESnextInstance) {

      *(here->NRESposPosptr) += here->NRESconduct;
      *(here->NRESnegNegptr) += here->NRESconduct;
      *(here->NRESposNegptr) -= here->NRESconduct;
      *(here->NRESnegPosptr) -= here->NRESconduct;
    }
  }
  return(OK);
}
```

A.8.2. Excerpt from nresload.c after modification

```
.
.
.
  /*  loop through all the nresistor models */
  for( ; model != NULL; model = model->NRESnextModel ) {
    /* loop through all the instances of the model */
    for (here = model->NRESinstances; here != NULL ;
        here=here->NRESnextInstance) {
      *(here->NRESposPosptr) -= here->NRESconduct;
      *(here->NRESnegNegptr) -= here->NRESconduct;
      *(here->NRESposNegptr) += here->NRESconduct;
      *(here->NRESnegPosptr) += here->NRESconduct;
    }
  }
  return(OK);
}
```

APPENDIX B.
LOSSLESS TRANSMISSION LINE CODE

This appendix contains code and code fragments from unmodified and modified versions of a data structure and functions connected with the lossless transmission line. The modified versions have been adapted to the current source/admittance model of a lossless transmission line as opposed to the voltage source/impedance model of the unmodified version. Whenever three vertical dots appear in the listings, some part of the listing for the file or function has been omitted. This has been done to conserve space by excluding header files and parts of the function which are irrelevant to the discussions referring to this appendix.

B.1. Device Data Structure

This section contains excerpts from the file ntradefs.h before and after current source/admittance model modification. The file is located in the directory spice3e2/src/include.

B.1.1. Contents of ntradefs.h before modification

.
.
.

```
typedef struct sNTRAinstance {
   struct sNTRAmodel *NTRAmodPtr;   /* backpointer to model */
   struct sNTRAinstance *NTRAnextInstance;  /* pointer to next instance of
                              * current model*/
   IFuid NTRAname;     /* pointer to character string naming this instance */
```

```
int NTRAposNode1;   /* number of positive node of end 1 of t. line */
int NTRAnegNode1;   /* number of negative node of end 1 of t. line */
int NTRAposNode2;   /* number of positive node of end 2 of t. line */
int NTRAnegNode2;   /* number of negative node of end 2 of t. line */
int NTRAintNode1;   /* number of internal node of end 1 of t. line */
int NTRAintNode2;   /* number of internal node of end 2 of t. line */
double NTRAimped;   /* impedance - input */
double NTRAconduct; /* conductance - calculated */
double NTRAtd;      /* propagation delay */
double NTRAnl;      /* normalized length */
double NTRAf;       /* frequency at which nl is measured */
double NTRAinput1;  /* accumulated excitation for port 1 */
double NTRAinput2;  /* accumulated excitation for port 2 */
double NTRAinitVolt1;   /* initial condition:  voltage on port 1 */
double NTRAinitCur1;    /* initial condition:  current at port 1 */
double NTRAinitVolt2;   /* initial condition:  voltage on port 2 */
double NTRAinitCur2;    /* initial condition:  current at port 2 */
double NTRAreltol;      /* relative deriv. tol. for breakpoint setting */
double NTRAabstol;      /* absolute deriv. tol. for breakpoint setting */
double *NTRAdelays;     /* delayed values of excitation */
int NTRAsizeDelay;      /* size of active delayed table */
int NTRAallocDelay;     /* allocated size of delayed table */
int NTRAbrEq1;      /* number of branch equation for end 1 of t. line */
int NTRAbrEq2;      /* number of branch equation for end 2 of t. line */
double *NTRAibr1Ibr2Ptr;    /* pointer to sparse matrix */
double *NTRAibr1Int1Ptr;    /* pointer to sparse matrix */
double *NTRAibr1Neg1Ptr;    /* pointer to sparse matrix */
double *NTRAibr1Neg2Ptr;    /* pointer to sparse matrix */
double *NTRAibr1Pos2Ptr;    /* pointer to sparse matrix */
double *NTRAibr2Ibr1Ptr;    /* pointer to sparse matrix */
double *NTRAibr2Int2Ptr;    /* pointer to sparse matrix */
double *NTRAibr2Neg1Ptr;    /* pointer to sparse matrix */
double *NTRAibr2Neg2Ptr;    /* pointer to sparse matrix */
double *NTRAibr2Pos1Ptr;    /* pointer to sparse matrix */
double *NTRAint1Ibr1Ptr;    /* pointer to sparse matrix */
double *NTRAint1Int1Ptr;    /* pointer to sparse matrix */
double *NTRAint1Pos1Ptr;    /* pointer to sparse matrix */
double *NTRAint2Ibr2Ptr;    /* pointer to sparse matrix */
double *NTRAint2Int2Ptr;    /* pointer to sparse matrix */
double *NTRAint2Pos2Ptr;    /* pointer to sparse matrix */
double *NTRAneg1Ibr1Ptr;    /* pointer to sparse matrix */
double *NTRAneg2Ibr2Ptr;    /* pointer to sparse matrix */
double *NTRApos1Int1Ptr;    /* pointer to sparse matrix */
double *NTRApos1Pos1Ptr;    /* pointer to sparse matrix */
double *NTRApos2Int2Ptr;    /* pointer to sparse matrix */
double *NTRApos2Pos2Ptr;    /* pointer to sparse matrix */
```

```
    unsigned NTRAimpedGiven : 1; /* flag to indicate impedence was specified */
    unsigned NTRAtdGiven : 1;   /* flag to indicate delay was specified */
    unsigned NTRAnlGiven : 1;   /* flag to indicate norm length was specified */
    unsigned NTRAfGiven : 1;    /* flag to indicate freq was specified */
    unsigned NTRAicV1Given : 1; /* flag to ind. init. voltage at port 1 given */
    unsigned NTRAicC1Given : 1; /* flag to ind. init. current at port 1 given */
    unsigned NTRAicV2Given : 1; /* flag to ind. init. voltage at port 2 given */
    unsigned NTRAicC2Given : 1; /* flag to ind. init. current at port 2 given */
    unsigned NTRAreltolGiven:1; /* flag to ind. relative deriv. tol. given */
    unsigned NTRAabstolGiven:1; /* flag to ind. absolute deriv. tol. given */
} NTRAinstance ;


/* per model data */
typedef struct sNTRAmodel {       /* model structure for a ntransmission lines */
    int NTRAmodType; /* type index of this device type */
    struct sNTRAmodel *NTRAnextModel; /* pointer to next possible model in
                                * linked list */
    NTRAinstance * NTRAinstances; /* pointer to list of instances that have this
                                * model */
    IFuid NTRAmodName;       /* pointer to character string naming this model */
} NTRAmodel;
/* device parameters */
#define NTRA_Z0 1
#define NTRA_TD 2
#define NTRA_NL 3
#define NTRA_FREQ 4
#define NTRA_V1 5
#define NTRA_I1 6
#define NTRA_V2 7
#define NTRA_I2 8
#define NTRA_IC 9
#define NTRA_RELTOL 10
#define NTRA_ABSTOL 11
#define NTRA_POS_NODE1 12
#define NTRA_NEG_NODE1 13
#define NTRA_POS_NODE2 14
 #define NTRA_NEG_NODE2 15
 #define NTRA_INPUT1 16
 #define NTRA_INPUT2 17
 #define NTRA_DELAY 18
 #define NTRA_BR_EQ1 19
 #define NTRA_BR_EQ2 20
 #define NTRA_INT_NODE1 21
 #define NTRA_INT_NODE2 22
.
.
.
```

B.1.2. Contents of ntradefs.h after modification

```
.
.
.

/* information used to describe a single instance */

typedef struct sNTRAinstance {
   struct sNTRAmodel *NTRAmodPtr;   /* backpointer to model */
   struct sNTRAinstance *NTRAnextInstance;  /* pointer to next instance of
                                * current model*/
   IFuid NTRAname;     /* pointer to character string naming this instance */

   int NTRAposNode1;   /* number of positive node of end 1 of t. line */
   int NTRAnegNode1;   /* number of negative node of end 1 of t. line */
   int NTRAposNode2;   /* number of positive node of end 2 of t. line */
   int NTRAnegNode2;   /* number of negative node of end 2 of t. line */

   double NTRAimped;   /* impedance - input */
   double NTRAconduct; /* conductance - calculated */
   double NTRAtd;      /* propagation delay */
   double NTRAnl;      /* normalized length */
   double NTRAf;       /* frequency at which nl is measured */

   double NTRAinput1;  /* accumulated excitation for port 1 */
   double NTRAinput2;  /* accumulated excitation for port 2 */
   double NTRAinput1Old;  /* prev val of accumulated excitation for port 1 */
   double NTRAinput2Old;  /* prev val of accumulated excitation for port 2 */

   double NTRAinitVolt1;   /* initial condition:  voltage on port 1 */
   double NTRAinitCur1;    /* initial condition:  current at port 1 */
   double NTRAinitVolt2;   /* initial condition:  voltage on port 2 */
   double NTRAinitCur2;    /* initial condition:  current at port 2 */

   double NTRAreltol;     /* relative deriv. tol. for breakpoint setting */
   double NTRAabstol;      /* absolute deriv. tol. for breakpoint setting */

   double *NTRAdelays;    /* delayed values of excitation */
   int NTRAsizeDelay;     /* size of active delayed table */
   int NTRAallocDelay;    /* allocated size of delayed table */

                          /* FOR USE WITH STAMP FILLING */
   double *NTRApos1Pos1Ptr;   /* pointer to sparse matrix */
   double *NTRApos1Neg1Ptr;   /* pointer to sparse matrix */
   double *NTRAneg1Pos1Ptr;   /* pointer to sparse matrix */
   double *NTRAneg1Neg1Ptr;   /* pointer to sparse matrix */
   double *NTRApos2Pos2Ptr;   /* pointer to sparse matrix */
   double *NTRApos2Neg2Ptr;   /* pointer to sparse matrix */
   double *NTRAneg2Pos2Ptr;   /* pointer to sparse matrix */
   double *NTRAneg2Neg2Ptr;   /* pointer to sparse matrix */

   unsigned NTRAimpedGiven : 1; /* flag to indicate impedence was specified */
```

```
    unsigned NTRAtdGiven : 1;   /* flag to indicate delay was specified */
    unsigned NTRAnlGiven : 1;   /* flag to indicate norm length was specified */
    unsigned NTRAfGiven : 1;    /* flag to indicate freq was specified */
    unsigned NTRAicV1Given : 1; /* flag to ind. init. voltage at port 1 given */
    unsigned NTRAicC1Given : 1; /* flag to ind. init. current at port 1 given */
    unsigned NTRAicV2Given : 1; /* flag to ind. init. voltage at port 2 given */
    unsigned NTRAicC2Given : 1; /* flag to ind. init. current at port 2 given */
    unsigned NTRAreltolGiven:1; /* flag to ind. relative deriv. tol. given */
    unsigned NTRAabstolGiven:1; /* flag to ind. absolute deriv. tol. given */

} NTRAinstance ;


/* per model data */

typedef struct sNTRAmodel {       /* model structure for an ntransmission lines */
    int NTRAmodType; /* type index of this device type */
    struct sNTRAmodel *NTRAnextModel; /* pointer to next possible model in
                                       * linked list */
    NTRAinstance * NTRAinstances; /* pointer to list of instances that have
                                     this model */

    IFuid NTRAmodName;       /* pointer to character string naming this model*/

} NTRAmodel;


/* device parameters */
#define NTRA_Z0 1
#define NTRA_TD 2
#define NTRA_NL 3
#define NTRA_FREQ 4
#define NTRA_V1 5
#define NTRA_I1 6
#define NTRA_V2 7
#define NTRA_I2 8
#define NTRA_IC 9
#define NTRA_RELTOL 10
#define NTRA_ABSTOL 11
#define NTRA_POS_NODE1 12
#define NTRA_NEG_NODE1 13
#define NTRA_POS_NODE2 14
#define NTRA_NEG_NODE2 15
#define NTRA_INPUT1 16
#define NTRA_INPUT2 17
#define NTRA_DELAY 18
.
.
.
```

B.2. NTRAsetup

This section contains the listings of the function NTRAsetup found in file spice3e2/src/lib/dev/ntra/ntrasetup.c before and after modification to operate as part of the current source/admittance model.

B.2.1. Contents of ntrasetup.c before modification

```
.
.
.
int
NTRAsetup(matrix,inModel,ckt,state)
    register SMPmatrix *matrix;
    GENmodel *inModel;
    register CKTcircuit *ckt;
    int *state;
        /* load the ntransmission line structure with those pointers needed later
         * for fast matrix loading
         */
{
    register NTRAmodel *model = (NTRAmodel *)inModel;
    register NTRAinstance *here;
    int error;
    CKTnode *tmp;

    /*  loop through all the ntransmission line models */
    for( ; model != NULL; model = model->NTRAnextModel ) {
        /* loop through all the instances of the model */
        for (here = model->NTRAinstances; here != NULL ;
                here=here->NTRAnextInstance) {

            if(here->NTRAbrEq1==0) {
                error = CKTmkVolt(ckt,&tmp,here->NTRAname,"i1");
                if(error) return(error);
                here->NTRAbrEq1 = tmp->number;
            }

            if(here->NTRAbrEq2==0) {
                error = CKTmkVolt(ckt,&tmp,here->NTRAname,"i2");
                if(error) return(error);
                here->NTRAbrEq2 = tmp->number;
            }
```

```
if(here->NTRAintNode1==0) {
   error = CKTmkVolt(ckt,&tmp,here->NTRAname,"int1");
   if(error) return(error);
   here->NTRAintNode1 = tmp->number;
}
```

```
if(here->NTRAintNode2==0) {
   error = CKTmkVolt(ckt,&tmp,here->NTRAname,"int2");
   if(error) return(error);
   here->NTRAintNode2 = tmp->number;
}
```

```
/* allocate the delay table */
here->NTRAdelays = (double *)MALLOC(15*sizeof(double));
here->NTRAallocDelay = 4;
```

```
/* macro to make elements with built in test for out of memory */
#define TSTALLOC(ptr,first,second) \
if((here->ptr = SMPmakeElt(matrix,here->first,here->second))==(double *)NULL){\
  return(E_NOMEM);\
}
```

```
TSTALLOC(NTRAibr1Ibr2Ptr, NTRAbrEq1, NTRAbrEq2)
TSTALLOC(NTRAibr1Int1Ptr, NTRAbrEq1, NTRAintNode1)
TSTALLOC(NTRAibr1Neg1Ptr, NTRAbrEq1, NTRAnegNode1)
TSTALLOC(NTRAibr1Neg2Ptr, NTRAbrEq1, NTRAnegNode2)
TSTALLOC(NTRAibr1Pos2Ptr, NTRAbrEq1, NTRAposNode2)
TSTALLOC(NTRAibr2Ibr1Ptr, NTRAbrEq2, NTRAbrEq1)
TSTALLOC(NTRAibr2Int2Ptr, NTRAbrEq2, NTRAintNode2)
TSTALLOC(NTRAibr2Neg1Ptr, NTRAbrEq2, NTRAnegNode1)
```

```
TSTALLOC(NTRAibr2Neg2Ptr, NTRAbrEq2, NTRAnegNode2)
TSTALLOC(NTRAibr2Pos1Ptr, NTRAbrEq2, NTRAposNode1)
TSTALLOC(NTRAint1Ibr1Ptr, NTRAintNode1, NTRAbrEq1)
TSTALLOC(NTRAint1Int1Ptr, NTRAintNode1, NTRAintNode1)
TSTALLOC(NTRAint1Pos1Ptr, NTRAintNode1, NTRAposNode1)
TSTALLOC(NTRAint2Ibr2Ptr, NTRAintNode2, NTRAbrEq2)
TSTALLOC(NTRAint2Int2Ptr, NTRAintNode2, NTRAintNode2)
```

```
TSTALLOC(NTRAint2Pos2Ptr, NTRAintNode2, NTRAposNode2)
TSTALLOC(NTRAneg1Ibr1Ptr, NTRAnegNode1, NTRAbrEq1)
TSTALLOC(NTRAneg2Ibr2Ptr, NTRAnegNode2, NTRAbrEq2)
TSTALLOC(NTRApos1Int1Ptr, NTRAposNode1, NTRAintNode1)
TSTALLOC(NTRApos1Pos1Ptr, NTRAposNode1, NTRAposNode1)
TSTALLOC(NTRApos2Int2Ptr, NTRAposNode2, NTRAintNode2)
TSTALLOC(NTRApos2Pos2Ptr, NTRAposNode2, NTRAposNode2)
```

.
.
.

B.2.2. Contents of ntrasetup.c after modification

```
.
.
.

int
NTRAsetup(matrix,inModel,ckt,state)
   register SMPmatrix *matrix;
   GENmodel *inModel;
   register CKTcircuit *ckt;
   int *state;
      /* load the transmission line structure with those pointers needed later
       * for fast matrix loading
       */
{
   FILE *data;
   register NTRAmodel *model = (NTRAmodel *)inModel;
   register NTRAinstance *here;
   int error;
   CKTnode *tmp;

   /*  loop through all the transmission line models */
   for( ; model != NULL; model = model->NTRAnextModel ) {

      /* loop through all the instances of the model */
      for (here = model->NTRAinstances; here != NULL ;
            here=here->NTRAnextInstance) {


         /* allocate the delay table */
         here->NTRAdelays = (double *)MALLOC(15*sizeof(double));
         here->NTRAallocDelay = 4;

/* macro to make elements with built in test for out of memory */
#define TSTALLOC(ptr,first,second) \
if((here->ptr = SMPmakeElt(matrix,here->first,here->second))==(double
*)NULL){\
  return(E_NOMEM);\
}

         TSTALLOC(NTRApos1Pos1Ptr, NTRAposNode1, NTRAposNode1)
         TSTALLOC(NTRApos1Neg1Ptr, NTRAposNode1, NTRAnegNode1)
         TSTALLOC(NTRAneg1Pos1Ptr, NTRAnegNode1, NTRAposNode1)
         TSTALLOC(NTRAneg1Neg1Ptr, NTRAnegNode1, NTRAnegNode1)

         TSTALLOC(NTRApos2Pos2Ptr, NTRAposNode2, NTRAposNode2)
         TSTALLOC(NTRApos2Neg2Ptr, NTRAposNode2, NTRAnegNode2)
         TSTALLOC(NTRAneg2Pos2Ptr, NTRAnegNode2, NTRAposNode2)
         TSTALLOC(NTRAneg2Neg2Ptr, NTRAnegNode2, NTRAnegNode2)

.
.
```

B.3. NTRAask

This section lists the contents of spice3e2/src/lib/dev/ntra/ntraask.c before the current source/admittance model changes. The listing after the modifications is not shown.

```
.
.
.

int
NTRAask(ckt,inst,which,value,select)
    CKTcircuit *ckt;
    GENinstance *inst;
    int which;
    IFvalue *value;
    IFvalue *select;
{
    NTRAinstance *here = (NTRAinstance *)inst;
    int temp;

    switch(which) {
      case NTRA_POS_NODE1:
        value->iValue = here->NTRAposNode1;
        return (OK);
      case NTRA_NEG_NODE1:
        value->iValue = here->NTRAnegNode1;
        return (OK);
      case NTRA_POS_NODE2:
        value->iValue = here->NTRAposNode2;
        return (OK);
      case NTRA_NEG_NODE2:
        value->iValue = here->NTRAnegNode2;
        return (OK);
      case NTRA_INT_NODE1:
        value->iValue = here->NTRAintNode1;
        return (OK);
      case NTRA_INT_NODE2:
        value->iValue = here->NTRAintNode2;
        return (OK);
.
.
.
```

```
case NTRA_BR_EQ1:
  value->rValue = here->NTRAbrEq1;
  return (OK);
case NTRA_BR_EQ2:
  value->rValue = here->NTRAbrEq2;
  return (OK);
```

```
.
.
.
        }
        return (OK);
      default:
        return (E_BADPARM);
    }
    /* NOTREACHED */
}
```

B.4. NTRAload

The following two subsections contain excerpts from the NTRAload function found in spice3e2/src/lib/dev/ntra/ntraload.c. The first listing is before the function has been modified to handle the current source/admittance stamp. The second listing is after the modification.

B.4.1. Contents of ntraload.c before modification

```
.
.
.

int
NTRAload(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
        /* actually load the current values into the
         * sparse matrix previously provided
         */
{
```

```
register NTRAmodel *model = (NTRAmodel *)inModel;
register NTRAinstance *here;
double t1,t2,t3;
double f1,f2,f3;
register int i;
/*  loop through all the ntransmission line models */
for( ; model != NULL; model = model->NTRAnextModel ) {

   /* loop through all the instances of the model */
   for (here = model->NTRAinstances; here != NULL ;
         here=here->NTRAnextInstance) {

      *(here->NTRApos1Pos1Ptr) += here->NTRAconduct;
      *(here->NTRApos1Int1Ptr) -= here->NTRAconduct;
      *(here->NTRAneg1Ibr1Ptr) -= 1;
      *(here->NTRApos2Pos2Ptr) += here->NTRAconduct;
      *(here->NTRAneg2Ibr2Ptr) -= 1;
      *(here->NTRAint1Pos1Ptr) -= here->NTRAconduct;
      *(here->NTRAint1Int1Ptr) += here->NTRAconduct;
      *(here->NTRAint1Ibr1Ptr) += 1;

      *(here->NTRAint2Int2Ptr) += here->NTRAconduct;
      *(here->NTRAint2Ibr2Ptr) += 1;
      *(here->NTRAibr1Neg1Ptr) -= 1;
      *(here->NTRAibr1Int1Ptr) += 1;
      *(here->NTRAibr2Neg2Ptr) -= 1;
      *(here->NTRAibr2Int2Ptr) += 1;
      *(here->NTRApos2Int2Ptr) -= here->NTRAconduct;
      *(here->NTRAint2Pos2Ptr) -= here->NTRAconduct;

      if(ckt->CKTmode & MODEDC) {
         *(here->NTRAibr1Pos2Ptr) -= 1;
         *(here->NTRAibr1Neg2Ptr) += 1;
         *(here->NTRAibr1Ibr2Ptr) -=(1-ckt->CKTgmin)*here->NTRAimped;
         *(here->NTRAibr2Pos1Ptr) -= 1;
         *(here->NTRAibr2Neg1Ptr) += 1;
         *(here->NTRAibr2Ibr1Ptr) -= (1-ckt->CKTgmin)*here->NTRAimped;
      } else {
         if (ckt->CKTmode & MODEINITTRAN) {
            if(ckt->CKTmode & MODEUIC) {
               here->NTRAinput1 = here->NTRAinitVolt2 + here->NTRAinitCur2
                  * here->NTRAimped;
               here->NTRAinput2 = here->NTRAinitVolt1 + here->NTRAinitCur1
                  * here->NTRAimped;
            } else {
```

```
                here->NTRAinput1 =
                   ( *(ckt->CKTrhsOld+here->NTRAposNode2)
                   - *(ckt->CKTrhsOld+here->NTRAnegNode2) )
                   + ( *(ckt->CKTrhsOld+here->NTRAbrEq2)
                      *here->NTRAimped);
                here->NTRAinput2 =
                   ( *(ckt->CKTrhsOld+here->NTRAposNode1)
                   - *(ckt->CKTrhsOld+here->NTRAnegNode1) )
                   + ( *(ckt->CKTrhsOld+here->NTRAbrEq1)
                      *here->NTRAimped);
              }
              *(here->NTRAdelays  ) = -2*here->NTRAtd;
              *(here->NTRAdelays +3) = -here->NTRAtd;
              *(here->NTRAdelays+6) = 0;
              *(here->NTRAdelays+1) = *(here->NTRAdelays +4) =
                    *(here->NTRAdelays+7) = here->NTRAinput1;
              *(here->NTRAdelays+2) = *(here->NTRAdelays +5) =
                    *(here->NTRAdelays+8) = here->NTRAinput2;
              here->NTRAsizeDelay = 2;
            } else {
              if(ckt->CKTmode & MODEINITPRED) {
.
.
.
              }
            }
            *(ckt->CKTrhs + here->NTRAbrEq1) += here->NTRAinput1;
            *(ckt->CKTrhs + here->NTRAbrEq2) += here->NTRAinput2;
          }
        }
      }
      return(OK);
}
```

B.4.2. Contents of ntraload.c after modification

```
.
.
.

int
NTRAload(inModel,ckt)
   GENmodel *inModel;
   CKTcircuit *ckt;
      /* actually load the current values into the
       * sparse matrix previously provided
       */
{
```

```
/* Variables declared for inside the function */
   register NTRAmodel *model = (NTRAmodel *)inModel;
   register NTRAinstance *here;
   double t1,t2,t3;
   double f1,f2,f3;
   register int i;


   /*  loop through all the NTRAnsmission line models */
   for( ; model != NULL; model = model->NTRAnextModel ) {

      /* loop through all the instances of the model */
      for (here = model->NTRAinstances; here != NULL ;
           here=here->NTRAnextInstance) {

/* MOST OF THE STAMP FILLED HERE */
         *(here->NTRApos1Pos1Ptr) += here->NTRAconduct;
         *(here->NTRApos1Neg1Ptr) -= here->NTRAconduct;
         *(here->NTRAneg1Pos1Ptr) -= here->NTRAconduct;
         *(here->NTRAneg1Neg1Ptr) += here->NTRAconduct;

         *(here->NTRApos2Pos2Ptr) += here->NTRAconduct;
         *(here->NTRApos2Neg2Ptr) -= here->NTRAconduct;
         *(here->NTRAneg2Pos2Ptr) -= here->NTRAconduct;
         *(here->NTRAneg2Neg2Ptr) += here->NTRAconduct;


/* STAMP FILL FOR DC ANALYSIS */
/* This section is to be left commented out until the stamp fill for the transient
analysis
   stamp filling and solving for the line is determined as functioning correctly.
Until then
   use initial conditions specification from the input file.  Once transient analysis is
   working this section is the place to modify the stamp for DC analysis.
*/
        if(ckt->CKTmode & MODEDC) {
/*
            *(here->NTRAibr1Pos2Ptr) -= 1;
           *(here->NTRAibr1Neg2Ptr) += 1;
            *(here->NTRAibr1Ibr2Ptr) -= (1-ckt->CKTgmin)*here->NTRAimped;
            *(here->NTRAibr2Pos1Ptr) -= 1;
            *(here->NTRAibr2Neg1Ptr) += 1;
            *(here->NTRAibr2Ibr1Ptr) -= (1-ckt->CKTgmin)*here->NTRAimped;
*/
         } else {
           if (ckt->CKTmode & MODEINITTRAN) {
/* THE INITIAL TRANSIENT RUN */
              if(ckt->CKTmode & MODEUIC) {
/* USE THE INITIAL CONDITIONS SUPPLIED INSTEAD OF THE DC
ANALYSIS VALUES */

                 here->NTRAinput1 = here->NTRAconduct* here->NTRAinitVolt2
                 + here->NTRAinitCur2;
```

```
                here->NTRAinput2 = here->NTRAconduct * here->NTRAinitVolt1
                + here->NTRAinitCur1;

            } else {
/* COMMENTED OUT TILL DEBUGGING FINISHED */
/* USES THE DC VALUES AS START */
/*
                here->NTRAinput1 = here->NTRAinput2Old
                + here->NTRAconduct *(*(ckt->CKTrhsOld
                + here->NTRAposNode2) - *(ckt->CKTrhsOld
                + here->NTRAnegNode2) );


                here->NTRAinput2 = here->NTRAinput1Old
                + here->NTRAconduct *( *(ckt->CKTrhsOld
                + here->NTRAposNode1) - *(ckt->CKTrhsOld
                + here->NTRAnegNode1) );
*/

            }

/* SET UP THE DELAY TABLE */
            *(here->NTRAdelays  ) = -2*here->NTRAtd;
            *(here->NTRAdelays +3) = -here->NTRAtd;
            *(here->NTRAdelays+6) = 0;
            *(here->NTRAdelays+1) = *(here->NTRAdelays +4) =
                 *(here->NTRAdelays+7) = here->NTRAinput1;
            *(here->NTRAdelays+2) = *(here->NTRAdelays +5) =
                 *(here->NTRAdelays+8) = here->NTRAinput2;
            here->NTRAsizeDelay = 2;


        } else {

/* FIND INTERPOLATED VALUES */

.
.
.

            }
        }
/* FILL THE RIGHT HAND SIDE */
        *(ckt->CKTrhs + here->NTRAposNode1) += here->NTRAinput1;
        *(ckt->CKTrhs + here->NTRAnegNode1) -= here->NTRAinput1;
        *(ckt->CKTrhs + here->NTRAposNode2) += here->NTRAinput2;
        *(ckt->CKTrhs + here->NTRAnegNode2) -= here->NTRAinput2;
        here->NTRAinput1Old = here->NTRAinput1;
        here->NTRAinput2Old = here->NTRAinput2;

        }
    }
  }
  return(OK);
```

```
}
```

B.5. NTRAacct

The following two subsections contain excerpts from the NTRAacct function found in spice3e2/src/lib/dev/ntra/ntraacct.c. The first listing is before the function has been modified to handle the current source/admittance stamp. The second listing is after the modification.

B.5.1. Contents of ntraacct.c before modification

```
.
.
.

int
NTRAaccept(ckt,inModel)
    register CKTcircuit *ckt;
    GENmodel *inModel;
{
    register NTRAmodel *model = (NTRAmodel *)inModel;
    register NTRAinstance *here;
    register int i=0,j;
    double v1,v2,v3,v4;
    double v5,v6,d1,d2,d3,d4;
    double *from,*to;
    int error;

    /*  loop through all the ntransmission line models */
    for( ; model != NULL; model = model->NTRAnextModel ) {

      /* loop through all the instances of the model */
      for (here = model->NTRAinstances; here != NULL ;
            here=here->NTRAnextInstance) {
        if( (ckt->CKTtime - here->NTRAtd) > *(here->NTRAdelays+6)) {
          /* shift! */
          for(i=2;i<here->NTRAsizeDelay &&
              (ckt->CKTtime - here->NTRAtd > *(here->NTRAdelays+3*i));i++)
                { /* loop does it all */ ; }
          i -= 2;
          for(j=i;j<=here->NTRAsizeDelay;j++) {
            from = here->NTRAdelays + 3*j;
```

```
                to = here->NTRAdelays + 3*(j-i);
                *(to) = *(from);
                *(to+1) = *(from+1);
                *(to+2) = *(from+2);
              }
              here->NTRAsizeDelay -= i;
            }
            if(ckt->CKTtime - *(here->NTRAdelays+3*here->NTRAsizeDelay) >
                ckt->CKTminBreak) {
              if(here->NTRAallocDelay <= here->NTRAsizeDelay) {
                /* need to grab some more space */
                here->NTRAallocDelay += 5;
                here->NTRAdelays = (double *)REALLOC((char *)here->NTRAdelays,
                  (here->NTRAallocDelay+1)*3*sizeof(double));
              }
              here->NTRAsizeDelay ++;
              to = (here->NTRAdelays  +3*here->NTRAsizeDelay);
              *to = ckt->CKTtime;
              to = (here->NTRAdelays+1+3*here->NTRAsizeDelay);
              *to = ( *(ckt->CKTrhsOld + here->NTRAposNode2)
                 -*(ckt->CKTrhsOld + here->NTRAnegNode2))
                + *(ckt->CKTrhsOld + here->NTRAbrEq2)*
                  here->NTRAimped;
              *(here->NTRAdelays+2+3*here->NTRAsizeDelay) =
                ( *(ckt->CKTrhsOld + here->NTRAposNode1)
                 -*(ckt->CKTrhsOld + here->NTRAnegNode1))
                + *(ckt->CKTrhsOld + here->NTRAbrEq1)*
                  here->NTRAimped;
.
.
.
          }
        }
        return(OK);
      }
```

B.5.2. Contents of ntraaccct.c after modification

```
.
.
.

int
NTRAaccept(ckt,inModel)
    register CKTcircuit *ckt;
    GENmodel *inModel;
{
```

```
register NTRAmodel *model = (NTRAmodel *)inModel;
register NTRAinstance *here;
register int i=0,j;
double v1,v2,v3,v4;
double v5,v6,d1,d2,d3,d4;
double *from,*to;
int error;


/*  loop through all the NTRAnsmission line models */
for( ; model != NULL; model = model->NTRAnextModel ) {

  /* loop through all the instances of the model */
  for (here = model->NTRAinstances; here != NULL ;
       here=here->NTRAnextInstance) {
    if( (ckt->CKTtime - here->NTRAtd) > *(here->NTRAdelays+6)) {
      /* shift! */
      for(i=2;i<here->NTRAsizeDelay &&
        (ckt->CKTtime - here->NTRAtd > *(here->NTRAdelays+3*i));i++)
          { /* loop does it all */ ; }
      i -= 2;
      for(j=i;j<=here->NTRAsizeDelay;j++) {
        from = here->NTRAdelays + 3*j;
        to = here->NTRAdelays + 3*(j-i);
        *(to) = *(from);
        *(to+1) = *(from+1);
        *(to+2) = *(from+2);
      }
      here->NTRAsizeDelay -= i;
    }

    if(ckt->CKTtime - *(here->NTRAdelays+3*here->NTRAsizeDelay) >
        ckt->CKTminBreak) {
      if(here->NTRAallocDelay <= here->NTRAsizeDelay) {
        /* need to grab some more space */
        here->NTRAallocDelay += 5;
        here->NTRAdelays = (double *)REALLOC(
        (char *)here->NTRAdelays, (here->NTRAallocDelay+1)
        *3*sizeof(double));
      }
      here->NTRAsizeDelay ++;

/* Inserting present data into delay table */
      to = (here->NTRAdelays  +3*here->NTRAsizeDelay);
      *to = ckt->CKTtime;

      to = (here->NTRAdelays+1+3*here->NTRAsizeDelay);
      *to = 2*here->NTRAconduct * ( *(ckt->CKTrhsOld
      + here->NTRAposNode2) - *(ckt->CKTrhsOld
      + here->NTRAnegNode2) ) - here->NTRAinput2;

      to = (here->NTRAdelays+2+3*here->NTRAsizeDelay);
      *to = 2*here->NTRAconduct * ( *(ckt->CKTrhsOld
      + here->NTRAposNode1) - *(ckt->CKTrhsOld
```

```
        + here->NTRAnegNode1) ) - here->NTRAinput1;
.
.
.
```

B.6. NTRAtrunc

The following two subsections contain excerpts from the NTRAtrunc function found in spice3e2/src/lib/dev/ntra/ntratrunc.c. The first listing is before the function has been modified to handle the current source model. The second listing is after the modification.

B.6.1. Contents of ntratrunc.c before modification

```
.
.
.

int
NTRAtrunc(inModel,ckt,timeStep)
  GENmodel *inModel;
  register CKTcircuit *ckt;
  double *timeStep;

{
  register NTRAmodel *model = (NTRAmodel *)inModel;
  register NTRAinstance *here;
  double v1,v2,v3,v4;
  double v5,v6,d1,d2,d3,d4;
  double tmp;
  /*  loop through all the ntransmission line models */
  for( ; model != NULL; model = model->NTRAnextModel ) {

    /* loop through all the instances of the model */
    for (here = model->NTRAinstances; here != NULL ;
        here=here->NTRAnextInstance) {
      v1 = ( *(ckt->CKTrhsOld + here->NTRAposNode2)
        - *(ckt->CKTrhsOld + here->NTRAnegNode2))
        + *(ckt->CKTrhsOld + here->NTRAbrEq2) *
          here->NTRAimped;
```

```
        v2 = *(here->NTRAdelays+1+3*(here->NTRAsizeDelay));
        v3 = *(here->NTRAdelays+1+3*(here->NTRAsizeDelay-1));
        v4 = ( *(ckt->CKTrhsOld + here->NTRAposNode1)
           - *(ckt->CKTrhsOld + here->NTRAnegNode1))
           + *(ckt->CKTrhsOld + here->NTRAbrEq1) *
             here->NTRAimped;
        v5 = *(here->NTRAdelays+2+3*(here->NTRAsizeDelay));
        v6 = *(here->NTRAdelays+2+3*(here->NTRAsizeDelay-1));
        d1 = (v1-v2)/ckt->CKTdeltaOld[1];
        d2 = (v2-v3)/ckt->CKTdeltaOld[2];
        d3 = (v4-v5)/ckt->CKTdeltaOld[1];
        d4 = (v5-v6)/ckt->CKTdeltaOld[2];
.
.
.
```

B.6.2. Contents of ntratrunc.c after modification

```
.
.
.

int
NTRAtrunc(inModel,ckt,timeStep)
   GENmodel *inModel;
   register CKTcircuit *ckt;
   double *timeStep;

{
   register NTRAmodel *model = (NTRAmodel *)inModel;
   register NTRAinstance *here;
    double v1,v2,v3,v4;
    double v5,v6,d1,d2,d3,d4;
   double tmp;

   /*  loop through all the NTRAnsmission line models */
   for( ; model != NULL; model = model->NTRAnextModel ) {

      /* loop through all the instances of the model */
      for (here = model->NTRAinstances; here != NULL ;
            here=here->NTRAnextInstance) {
         v1 = ( *(ckt->CKTrhsOld + here->NTRAposNode2)
            - *(ckt->CKTrhsOld + here->NTRAnegNode2))
            + here->NTRAinput2 * here->NTRAimped;
         v2 = *(here->NTRAdelays+1+3*(here->NTRAsizeDelay));
         v3 = *(here->NTRAdelays+1+3*(here->NTRAsizeDelay-1));
         v4 = ( *(ckt->CKTrhsOld + here->NTRAposNode1)
            - *(ckt->CKTrhsOld + here->NTRAnegNode1))
            + here->NTRAinput1 * here->NTRAimped;
         v5 = *(here->NTRAdelays+2+3*(here->NTRAsizeDelay));
         v6 = *(here->NTRAdelays+2+3*(here->NTRAsizeDelay-1));
```

```
d1 = (v1-v2)/ckt->CKTdeltaOld[1];
d2 = (v2-v3)/ckt->CKTdeltaOld[2];
d3 = (v4-v5)/ckt->CKTdeltaOld[1];
d4 = (v5-v6)/ckt->CKTdeltaOld[2];
.
.
.
```

APPENDIX C.
LOSSY TRANSMISSION LINE CODE

This appendix contains code and code fragments from unmodified and modified versions of a data structure and functions connected with the lossy transmission line. Whenever three vertical dots appear in the listings, some part of the listing for the file or function has been omitted. This has been done to conserve space by excluding header files and parts of the function which are irrelevant to the discussions referring to this appendix.

C.1. Function Decleration and Argument Description of G

The following lists the function prototype for G with short explanations of the arguments of G. The function is found in the file spice3e2/src/lib/dev/ntra/vdmmodel.c.

The function prototype is:

```
double G(   int L, double *ay, double *fcy, int My, double Ty,
            double xyold, double Aw, double *aw, double *fcw,
            int Mw, double Tw, double xw, double xwold, double *Isp    )
```

The following is a short listing and explanation of the input variables. The variables supplied in the input file (see the Input File section) are marked as such.

L — The index of the current source for which the subroutine is doing calculations (1 or 2). This is determined by the calling program.

*ay — A pointer to a set of difference approximation parameters for the characteristic admittance. There are My(1 or 2) of these. [ay1 when L=1 and ay2 when L=2], *supplied* see NOTE.

*fcw — See *ay. [fcw1 when L=1 and fcw2 when L=2], *supplied* see NOTE.

My — The difference model order for characteristic admittance. This is also the number of ay(1 or 2) and fcw(1 or 2) values, and comes

before ay and fcw in the input file and can be used to tell a read-in program how many values to read-in or how much space to allocate.
[My1 when L=1 and My2 when L=2], *supplied.*

Ty The current value of the time step. Ty = t(n) - t(n-1) see Figure C.1.

xyold The old value of port voltage xy at t(n-1). V1 if L=1 and V2 if L=2 see Figure C.1.

Aw The final value of the propagation function.
[Awb if L=1 and Awf if L=2] *supplied.*

*aw A pointer to a set of difference approximation parameters for the propagation function. [awb if L=1 and awf if L=2] *supplied* see NOTE.

*fcw See *aw, [fcwb if L=1 and fcwf if L=2], *supplied* see NOTE.

Mw The difference model order for the propagation function.
[Myb if L=1 and Myf if L=2] *supplied.*

Tw The time step used during the analysis one transmission line delay ago.
Tw = t(n-m) - t(n-m-1) see Fgure 2.

xw The delayed value of the excitation for the propagation function.
(2 * (V2(t-tau)* Yo2 - G2(t-tau)) + Is2(t-tau)) if L=1
(2 * (V1(t-tau)* Yo1 - G1(t-tau)) + Is1(t-tau)) if L=2
tau = the line delay,
xw is found by second order interpolation involving the value of xwe at t(n-m), t(n-m-1), t(n-m-2), where t(n-m) is the first time point greater than t(n)-tau. xwe at a particular time looks just like xw except the values of voltage and currents aren't taken from t-tau but right at the present t.
$xwe(t) = 2*(v_L(t)*Y_{oL} - G_L(t)) + Is_L(t)$ where L = 1 or 2 as appropriate, see Figure 2.

xwold The value of xw calculated at the time step just before the present one (xw at t(n-1) see Figure 2)

*Isp Part of the current source current that originated from the current source in the frequency domain. This is a pointer the calling program passes to G and uses in the calculation of xwe.

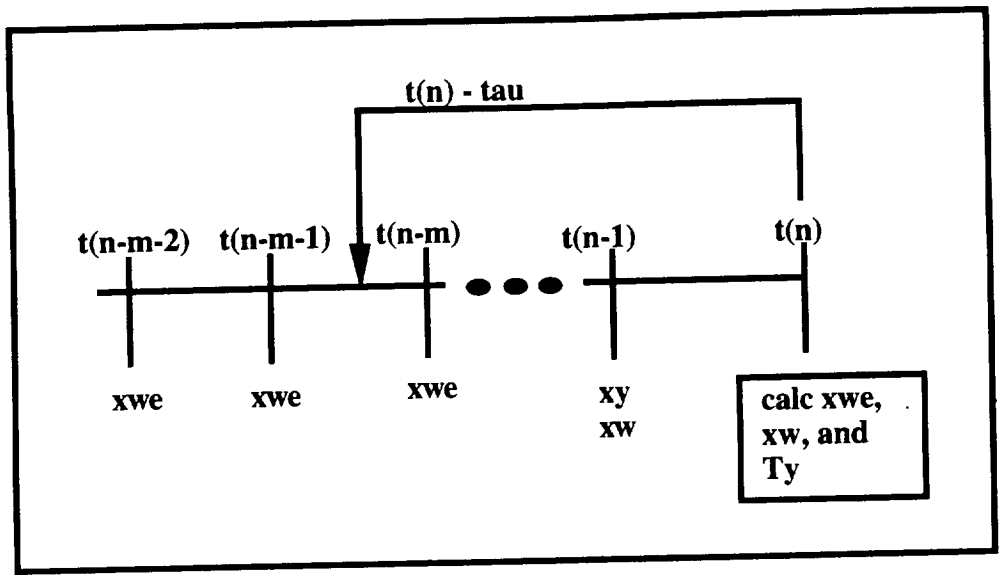


Figure C.1. Time line illustrating history of arguments to G.

The following is a description of the labels in Figure C.1:

t( ) This represents a time point where the argument inside is an index used to refer to the time point. A time step is the difference between two consecutive time points.

n The index of the present time step, n = 0, 1, 2,...

tau The line delay.

t(n) The present time point at which the stamp is being used.
t(n-1) The time point just before the present one.
t(n-m) The very first time point that is greater than t(n)-tau, $m+2 \leq n$.
t(n-m-1) The time point just preceding t(n-m).
t(n-m-2) The time point just preceding t(n-m-1).

NOTE: In the input file there is a list of these values. The values can be read into an array or into memory accessed by a pointer. The subroutine requires the beginning address. (see Input File section).

C.2. Contents of an Example Difference Parameters File

This section contains a listing of the contents of the difference parameter file created by the vdmdiff program. Inside the parameter file the specification for the line associated with the parameters is given. The function vdmdiff is found in spice3e2/src/bin and was written by D. Kuznetsov.

```
6
4.2409111e-02     2.5867665e+07
4.3788458e-02     1.3135440e+07
6.7126054e-02     7.0383468e+06
6.2640246e-02     2.5156424e+06
4.3410412e-02     7.1643273e+05
4.2151584e-02     8.4929645e+04
6.9847406e-01     9.9999992e-01     3.0947848e-09
6
4.2409111e-02     2.5867665e+07
4.3788458e-02     1.3135440e+07
6.7126054e-02     7.0383468e+06
6.2640246e-02     2.5156424e+06
4.3410412e-02     7.1643273e+05
4.2151584e-02     8.4929645e+04
6.9847406e-01     9.9999992e-01     3.0947848e-09
5
-3.5948372e-03    3.2832546e+07
-6.1014303e-04    1.7430986e+07
-1.8665157e-03    1.2688930e+07
-1.2594786e-03    3.8003358e+06
-1.1752701e-03    4.6826466e+05
 8.5062457e-03    8.9442719e-10
5
-3.5948372e-03    3.2832546e+07
-6.1014303e-04    1.7430986e+07
-1.8665157e-03    1.2688930e+07
-1.2594786e-03    3.8003358e+06
-1.1752701e-03    4.6826466e+05
 8.5062457e-03    8.9442719e-10
```

This file contains the Difference Approximation parameters for the line propagation function and characteristic admittance.

The distributed line parameters are:

```
a.lin              parameters file
l = 6.7500000e-01  m,       line length
L = 5.3900000e-07  H/m,     distributed inductance
C = 3.9000000e-11  F/m,     distributed capacitance
R = 1.2500000e+02  Ohm/m,   distributed resistance
```

```
Rs = 0.0000000e+00          Ohm/(Hz)^1/2,  skin resistance
G  = 1.0000000e-16  S/m,        distributed conductance
```

The format of the file is:

```
Mwf           order of the approx-tion for the forward propagation function;
awf[1] fcwf[1]    |
   .          | the Difference Approximation parameters
   .          | for the forward propagation function;
   .          |
awf[Mwf] fcwf[Mwf] |
 Awf Bwf tauf    fin. & init. val-s of forw. prop. func. and forw. prop. delay;
Mwb           order of the approx-tion for the backward propagation function;
awb[1] fcwb[1]    |
   .          | the Difference Approximation parameters
   .          | for the backward propagation function;
   .          |
awb[Mwb] fcwb[Mwb] |
Awb Bwb taub    fin. & init. val-s of backw. prop. func. and backw. prop. delay;
My1           order of the appr. for the near-end characteristic admittance;
ay1[1] fcy1[1]    |
   .          | the Difference Approximation parameters
   .          | for the near-end characteristic admittance;
   .          |
ay1[My1] fcy1[My1] |
 Ay1 By1       finial and initial values of the near-end char. admitt.;
My2           order of the appr. for the far-end characteristic admittance;
ay2[1] fcy1[1]    |
   .          | the Difference Approximation parameters
   .          | for the far-end characteristic admittance;
   .          |
ay2[My2] fcy2[My2] |
 Ay2 By2       finial and initial values of the far-end char. admitt.
```

C.3. Header Files

This section contains excerpts from the header files that are modified in converting from the lossless model to the lossy model. The header files are found in the directory spice3e2/src/include with the other SPICE3E2 headers.

C.3.1. Contents of ntradefs.h after modification

```
.
.
.
```

```
typedef struct sNTRAinstance {
    struct sNTRAmodel *NTRAmodPtr;    /* backpointer to model */
    struct sNTRAinstance *NTRAnextInstance;  /* pointer to next instance of
                                * current model*/
    IFuid NTRAname;     /* pointer to character string naming this instance */

    int NTRAposNode1;   /* number of positive node of end 1 of t. line */
    int NTRAnegNode1;   /* number of negative node of end 1 of t. line */
    int NTRAposNode2;   /* number of positive node of end 2 of t. line */
    int NTRAnegNode2;   /* number of negative node of end 2 of t. line */

    double NTRAimped;   /* impedance - input */
    double NTRAconduct; /* conductance - calculated */
    double NTRAtd;      /* propagation delay */
    double NTRAnl;      /* normalized length */
    double NTRAf;       /* frequency at which nl is measured */

    double NTRAinput1;  /* accumulated excitation for port 1 */
    double NTRAinput2;  /* accumulated excitation for port 2 */
    double NTRAinput1_old;  /* prev val of accumulated excitation for port 1 */
    double NTRAinput2_old;  /* prev val of accumulated excitation for port 2 */

    double NTRAinitVolt1;   /* initial condition:  voltage on port 1 */
    double NTRAinitCur1;    /* initial condition:  current at port 1 */
    double NTRAinitVolt2;   /* initial condition:  voltage on port 2 */
    double NTRAinitCur2;    /* initial condition:  current at port 2 */
    double NTRAreltol;      /* relative deriv. tol. for breakpoint setting */
    double NTRAabstol;      /* absolute deriv. tol. for breakpoint setting */

    double *NTRAdelays;     /* delayed values of excitation */
    int NTRAsizeDelay;      /* size of active delayed table */
    int NTRAallocDelay;     /* allocated size of delayed table */

    double *NTRAibr1Ibr1Ptr;    /* pointer to sparse matrix */
    double *NTRAibr1Neg1Ptr;    /* pointer to sparse matrix */
    double *NTRAibr1Pos1Ptr;    /* pointer to sparse matrix */
    double *NTRAibr2Ibr2Ptr;    /* pointer to sparse matrix */
    double *NTRAibr2Neg2Ptr;    /* pointer to sparse matrix */
    double *NTRAibr2Pos2Ptr;    /* pointer to sparse matrix */

                                /* FOR USE WITH STAMP FILLING */
    double *NTRApos1Pos1Ptr;    /* pointer to sparse matrix */
    double *NTRApos1Neg1Ptr;    /* pointer to sparse matrix */
    double *NTRAneg1Pos1Ptr;    /* pointer to sparse matrix */
    double *NTRAneg1Neg1Ptr;    /* pointer to sparse matrix */
    double *NTRApos2Pos2Ptr;    /* pointer to sparse matrix */
    double *NTRApos2Neg2Ptr;    /* pointer to sparse matrix */
    double *NTRAneg2Pos2Ptr;    /* pointer to sparse matrix */
    double *NTRAneg2Neg2Ptr;    /* pointer to sparse matrix */

    unsigned NTRAimpedGiven : 1; /* flag to indicate impedence was specified */
    unsigned NTRAtdGiven : 1;   /* flag to indicate delay was specified */
    unsigned NTRAnlGiven : 1;   /* flag to indicate norm length was specified */
    unsigned NTRAfGiven : 1;    /* flag to indicate freq was specified */
```

```
    unsigned NTRAicV1Given : 1;  /* flag to ind. init. voltage at port 1 given */
    unsigned NTRAicC1Given : 1;  /* flag to ind. init. current at port 1 given */
    unsigned NTRAicV2Given : 1;  /* flag to ind. init. voltage at port 2 given */
    unsigned NTRAicC2Given : 1;  /* flag to ind. init. current at port 2 given */
    unsigned NTRAreltolGiven:1;  /* flag to ind. relative deriv. tol. given */
    unsigned NTRAabstolGiven:1;  /* flag to ind. absolute deriv. tol. given */

    double *awf;   /* Parameters used in difference model */
    double *fcwf;
    double *awb;
    double *fcwb;
    double *ay1;
    double *fcy1;
    double *ay2;
    double *fcy2;
    double Awf;

    double Bwf;
    double tauf;
    double Awb;
    double Bwb;
    double taub;
    double Ay1;
    double By1;
    double Ay2;
    double By2;

    double Is1;
    double Is2;

    double xwold1;
    double xwold2;

    double oldtime;

    int Mwf;
    int Mwb;
    int My1;
    int My2;

} NTRAinstance ;

.
.
.
```

C.3.2. Contents of ntraitf.h

```
.
.
.

SPICEdev NTRAinfo = {
  {
    "Ntranline",
    "Lossy transmission line",

    &NTRAnSize,
    &NTRAnSize,
    NTRAnames,

    &NTRApTSize,
    NTRApTable,

    0/*&NTRAmPTSize,
    NULL/*NTRAmPTable/**/,
  },
.
.
.
```

C.4. NTRAparam

Excerpts from spice3e2/src/lib/dev/ntra/ntraparam.c, which contains the NTRAparam function, are shown in this section. The listing is of NTRAparam after it has been modified for lossy functionality.

```
.
.
.

int
NTRAparam(param,value,inst,select)
  int param;
  IFvalue *value;
  GENinstance *inst;
  IFvalue *select;
{
  NTRAinstance *here = (NTRAinstance *)inst;

  switch(param) {
    case NTRA_RELTOL:
```

```
            here->NTRAreltol = value->rValue;
            here->NTRAreltolGiven = TRUE;
            break;
        case NTRA_ABSTOL:

            here->NTRAabstol = value->rValue;
            here->NTRAabstolGiven = TRUE;
            break;
        case NTRA_Z0:
/*              here->NTRAimped = value->rValue;
*/
            here->NTRAimpedGiven = TRUE;
            break;
        case NTRA_TD:
/*              here->NTRAtd = value->rValue;
*/
            here->NTRAtdGiven = TRUE;
            break;
        case NTRA_NL:
            here->NTRAnl= value->rValue;
            here->NTRAnlGiven = TRUE;
            break;
.
.
.
```

C.5. NTRAsetup

Shown below are excerpts from spice3e2/src/lib/dev/ntra/ntrasetup.c, which is the file that contains the NTRAsetup function. The modifications made to convert NTRAsetup to a function of the lossy line module are shown in the boxes.

```
.
.
.
int
NTRAsetup(matrix,inModel,ckt,state)
    register SMPmatrix *matrix;
    GENmodel *inModel;
    register CKTcircuit *ckt;
    int *state;
        /* load the transmission line structure with those pointers needed later
         * for fast matrix loading
         */
{
    FILE *data;
```

```
register NTRAmodel *model = (NTRAmodel *)inModel;
register NTRAinstance *here;
int error;
CKTnode *tmp;

/*  loop through all the transmission line models */
for( ; model != NULL; model = model->NTRAnextModel ) {

   /* loop through all the instances of the model */
   for (here = model->NTRAinstances; here != NULL ;
        here=here->NTRAnextInstance) {

/* ======Difference paraneters for lossy line being loaded ====== */
```

```
here->Mwf = 6;
here->awf = (double *)MALLOC(here->Mwf*sizeof(double));
*(here->awf + 0) = 4.2409111e-02;
*(here->awf + 1) = 4.3788458e-02;
*(here->awf + 2) = 6.7126054e-02;
*(here->awf + 3) = 6.2640246e-02;
*(here->awf + 4) = 4.3410412e-02;
*(here->awf + 5) = 4.2151584e-02;
```

```
here->fcwf = (double *)MALLOC(here->Mwf*sizeof(double));
*(here->fcwf + 0) = 2.5867665e7;
*(here->fcwf + 1) = 1.3135440e7;
*(here->fcwf + 2) = 7.0383468e6;
*(here->fcwf + 3) = 2.5156424e6;
*(here->fcwf + 4) = 7.1643273e5;
*(here->fcwf + 5) = 8.4929645e4;
here->Awf = 6.9847406e-1;
here->Bwf = 9.9999992e-1;
here->tauf = 3.0947848e-9;
```

```
here->Mwb = 6;
here->awb = (double *)MALLOC(here->Mwb*sizeof(double));
*(here->awb + 0) = 4.2409111e-02;
*(here->awb + 1) = 4.3788458e-02;
*(here->awb + 2) = 6.7126054e-02;
*(here->awb + 3) = 6.2640246e-02;
*(here->awb + 4) = 4.3410412e-02;
*(here->awb + 5) = 4.2151584e-02;
```

```
here->fcwb = (double *)MALLOC(here->Mwb*sizeof(double));
*(here->fcwb + 0) = 2.5867665e7;
*(here->fcwb + 1) = 1.3135440e7;
*(here->fcwb + 2) = 7.0383468e6;
*(here->fcwb + 3) = 2.5156424e6;
*(here->fcwb + 4) = 7.1643273e5;
*(here->fcwb + 5) = 8.4929645e4;
here->Awb = 6.9847406e-1;
here->Bwb = 9.9999992e-1;
here->taub = 3.0947848e-9;
```

```
here->My1 = 5;
here->ay1 = (double *)MALLOC(here->My1*sizeof(double));
*(here->ay1 + 0) = -3.5948372e-03;
*(here->ay1 + 1) = -6.1014303e-04;
*(here->ay1 + 2) = -1.8665157e-03;
*(here->ay1 + 3) = -1.2594786e-03;
*(here->ay1 + 4) = -1.1752701e-03;
```

```
here->fcy1 = (double *)MALLOC(here->My1*sizeof(double));
*(here->fcy1 + 0) = 3.2832546e7;
*(here->fcy1 + 1) = 1.7430986e7;
*(here->fcy1 + 2) = 1.2688930e7;
*(here->fcy1 + 3) = 3.8003358e6;
*(here->fcy1 + 4) = 4.6826466e5;
here->Ay1 = 8.5062457e-03;
here->By1 = 8.9442719e-10;
```

```
here->My2 = 5;
here->ay2 = (double *)MALLOC(here->My2*sizeof(double));
*(here->ay2 + 0) = -3.5948372e-03;
*(here->ay2 + 1) = -6.1014303e-04;
*(here->ay2 + 2) = -1.8665157e-03;
*(here->ay2 + 3) = -1.2594786e-03;
*(here->ay2 + 4) = -1.1752701e-03;
```

```
        here->fcy2 = (double *)MALLOC(here->My2*sizeof(double));
        *(here->fcy2 + 0) = 3.2832546e7;
        *(here->fcy2 + 1) = 1.7430986e7;
        *(here->fcy2 + 2) = 1.2688930e7;
        *(here->fcy2 + 3) = 3.8003358e6;
        *(here->fcy2 + 4) = 4.6826466e5;
        here->Ay2 = 8.5062457e-03;
        here->By2 = 8.9442719e-10;
/* ========================================== */

        /* allocate the delay table */
        here->NTRAdelays = (double *)MALLOC(15*sizeof(double));
        here->NTRAallocDelay = 4;

/* macro to make elements with built in test for out of memory */
#define TSTALLOC(ptr,first,second) \
if((here->ptr = SMPmakeElt(matrix,here->first,here->second))==(double
*)NULL){\
  return(E_NOMEM);\
}

        TSTALLOC(NTRApos1Pos1Ptr, NTRAposNode1, NTRAposNode1)
        TSTALLOC(NTRApos1Neg1Ptr, NTRAposNode1, NTRAnegNode1)
        TSTALLOC(NTRAneg1Pos1Ptr, NTRAnegNode1, NTRAposNode1)
        TSTALLOC(NTRAneg1Neg1Ptr, NTRAnegNode1, NTRAnegNode1)

        TSTALLOC(NTRApos2Pos2Ptr, NTRAposNode2, NTRAposNode2)
        TSTALLOC(NTRApos2Neg2Ptr, NTRAposNode2, NTRAnegNode2)
        TSTALLOC(NTRAneg2Pos2Ptr, NTRAnegNode2, NTRAposNode2)
        TSTALLOC(NTRAneg2Neg2Ptr, NTRAnegNode2, NTRAnegNode2)

        if(!here->NTRAnlGiven) {
          here->NTRAnl = .25;
        }
        if(!here->NTRAfGiven) {
          here->NTRAf = 1e9;
        }
        if(!here->NTRAreltolGiven) {
          here->NTRAreltol = 1;
        }
        if(!here->NTRAabstolGiven) {
          here->NTRAabstol = 1;
        }
        if(!here->NTRAimpedGiven) {
/*          (*(SPfrontEnd->IFerror))(ERR_FATAL,
              "%s: ntrans z0 must be given ->ntrasetup.c",
              &(here->NTRAname));
            return(E_BADPARM);
*/
        }
        if(!here->NTRAparamfileGiven) {
```

```
            (*(SPfrontEnd->IFerror))(ERR_FATAL,
                "%s: ntrans filename must be given ->ntrasetup.c",
                &(here->NTRAname));
            return(E_BADPARM);
          }


        }
      }
      return(OK);
    }
```

C.6. NTRAload

Two excerpts are provided of the NTRAload function, found in spice3e2/src/lib/dev/ntra/ntraload.c. The listing in Section C.6.1 shows the function before modification and the sections of code that require modification to convert NTRAload to a lossy line function. Section C.6.2 lists the modified version of NTRAload.

C.6.1. Contents of ntraload.c before modification

```
.
.
.

int
NTRAload(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
        /* actually load the current values into the
         * sparse matrix previously provided
         */
{
/* Variables declared for inside the function */
    register NTRAmodel *model = (NTRAmodel *)inModel;
    register NTRAinstance *here;
    double t1,t2,t3;
    double f1,f2,f3;
    register int i;


    /*  loop through all the NTRAnsmission line models */
    for( ; model != NULL; model = model->NTRAnextModel ) {

        /* loop through all the instances of the model */
        for (here = model->NTRAinstances; here != NULL ;
```

```
        here=here->NTRAnextInstance) {

/* MOST OF THE STAMP FILLED HERE */
        *(here->NTRApos1Pos1Ptr) += here->NTRAconduct;
        *(here->NTRApos1Neg1Ptr) -= here->NTRAconduct;
        *(here->NTRAneg1Pos1Ptr) -= here->NTRAconduct;
        *(here->NTRAneg1Neg1Ptr) += here->NTRAconduct;

        *(here->NTRApos2Pos2Ptr) += here->NTRAconduct;
        *(here->NTRApos2Neg2Ptr) -= here->NTRAconduct;
        *(here->NTRAneg2Pos2Ptr) -= here->NTRAconduct;
        *(here->NTRAneg2Neg2Ptr) += here->NTRAconduct;


/* STAMP FILL FOR DC ANALYSIS */
/* This section is to be left commented out until the stamp fill for the transient
analysis
   stamp filling and solving for the line is determined as functioning correctly.
Until then
   use initial conditions specification from the input file. Once transient analysis is
   working this section is the place to modify the stamp for DC analysis.
*/
        if(ckt->CKTmode & MODEDC) {
/*

           *(here->NTRAibr1Pos2Ptr) -= 1;
           *(here->NTRAibr1Neg2Ptr) += 1;
           *(here->NTRAibr1Ibr2Ptr) -= (1-ckt->CKTgmin)*here->NTRAimped;
           *(here->NTRAibr2Pos1Ptr) -= 1;
           *(here->NTRAibr2Neg1Ptr) += 1;
           *(here->NTRAibr2Ibr1Ptr) -= (1-ckt->CKTgmin)*here->NTRAimped;
*/
        } else {
          if (ckt->CKTmode & MODEINITTRAN) {
/* THE INITIAL TRANSIENT RUN */
            if(ckt->CKTmode & MODEUIC) {
/* USE THE INITIAL CONDITIONS SUPPLIED INSTEAD OF THE DC
ANALYSIS VALUES */

              here->NTRAinput1 = here->NTRAconduct*
              here->NTRAinitVolt2 + here->NTRAinitCur2;

              here->NTRAinput2 = here->NTRAconduct *
              here->NTRAinitVolt1 + here->NTRAinitCur1;

            } else {
/* COMMENTED OUT TILL DEBUGGING FINISHED */
/* USES THE DC VALUES AS START */
/*
              here->NTRAinput1 = here->NTRAinput2Old
              + here->NTRAconduct * (*(ckt->CKTrhsOld
```

```
                + here->NTRAposNode2) - *(ckt->CKTrhsOld
                + here->NTRAnegNode2) );


                here->NTRAinput2 = here->NTRAinput1Old
                + here->NTRAconduct * ( *(ckt->CKTrhsOld
                + here->NTRAposNode1) - *(ckt->CKTrhsOld
                + here->NTRAnegNode1) );
*/

            }

/* SET UP THE DELAY TABLE */
            *(here->NTRAdelays  ) = -2*here->NTRAtd;
            *(here->NTRAdelays +3) = -here->NTRAtd;
            *(here->NTRAdelays+6) = 0;
            *(here->NTRAdelays+1) = *(here->NTRAdelays +4) =
                *(here->NTRAdelays+7) = here->NTRAinput1;
            *(here->NTRAdelays+2) = *(here->NTRAdelays +5) =
                *(here->NTRAdelays+8) = here->NTRAinput2;
            here->NTRAsizeDelay = 2;


        } else {

/* FIND INTERPOLATED VALUES */
            if(ckt->CKTmode & MODEINITPRED) {

              for(i=2;(i<here->NTRAsizeDelay) &&
                 (*(here->NTRAdelays +3*i) <=
                 (ckt->CKTtime-here->NTRAtd));i++) {;/*loop does it*/}
              t1 = *(here->NTRAdelays + (3*(i-2)));
              t2 = *(here->NTRAdelays + (3*(i-1)));
              t3 = *(here->NTRAdelays + (3*(i  )));

              if( (t2-t1)==0  || (t3-t2) == 0) continue;
              f1 = (ckt->CKTtime - here->NTRAtd - t2) *
                 (ckt->CKTtime - here->NTRAtd - t3) ;
              f2 = (ckt->CKTtime - here->NTRAtd - t1) *
                 (ckt->CKTtime - here->NTRAtd - t3) ;
              f3 = (ckt->CKTtime - here->NTRAtd - t1) *
                 (ckt->CKTtime - here->NTRAtd - t2) ;
              if((t2-t1)==0) { /* should never happen, but don't want
                          * to divide by zero, EVER... */
                 f1=0;
                 f2=0;
              } else {
                 f1 /= (t1-t2);
                 f2 /= (t2-t1);
              }
              if((t3-t2)==0) { /* should never happen, but don't want
                          * to divide by zero, EVER... */
                 f2=0;
                 f3=0;
```

```
                } else {
                  f2 /= (t2-t3);
                  f3 /= (t2-t3);
                }
                if((t3-t1)==0) { /* should never happen, but don't want
                                  * to divide by zero, EVER... */
                  f1=0;
                  f2=0;
                } else {
                  f1 /= (t1-t3);
                  f3 /= (t1-t3);
                }
.
.
.
                here->NTRAinput1 = f1 * *(here->NTRAdelays + (3*(i-2))+1)
                                + f2 * *(here->NTRAdelays + (3*(i-1))+1)
                                + f3 * *(here->NTRAdelays + (3*(i  ))+1);
                here->NTRAinput2 = f1 * *(here->NTRAdelays + (3*(i-2))+2)
                                + f2 * *(here->NTRAdelays + (3*(i-1))+2)
                                + f3 * *(here->NTRAdelays + (3*(i  ))+2);

              }
            }
/* FILL THE RIGHT HAND SIDE */
          *(ckt->CKTrhs + here->NTRAposNode1) += here->NTRAinput1;
          *(ckt->CKTrhs + here->NTRAnegNode1) -= here->NTRAinput1;
          *(ckt->CKTrhs + here->NTRAposNode2) += here->NTRAinput2;
          *(ckt->CKTrhs + here->NTRAnegNode2) -= here->NTRAinput2;
           here->NTRAinput1Old = here->NTRAinput1;
           here->NTRAinput2Old = here->NTRAinput2;

          }
        }
      }
      return(OK);
}
```

C.6.2. Contents of ntraload.c after modification

```
.
.
.
int
NTRAload(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
        /* actually load the current values into the
         * sparse matrix previously provided
```

```
    */
{
/* Variables declared for inside the function */
  register NTRAmodel *model = (NTRAmodel *)inModel;
  register NTRAinstance *here;
  double t1,t2,t3;
  double f1,f2,f3;
  register int i;

/* extra vars */
  int L;
  double xw1, xw2;
  double Ty;
  double delay;
  double xyold;
  double Tw;
  double xw;
  double xwold;
  double zero;
/****************/

  /*  loop through all the NTRAnsmission line models */
  for( ; model != NULL; model = model->NTRAnextModel ) {

    /* loop through all the instances of the model */
    for (here = model->NTRAinstances; here != NULL ;
          here=here->NTRAnextInstance) {

/* MOST OF THE STAMP FILLED HERE */
        *(here->NTRApos1Pos1Ptr) += here->Ay1;
        *(here->NTRApos1Neg1Ptr) -= here->Ay1;
        *(here->NTRAneg1Pos1Ptr) -= here->Ay1;
        *(here->NTRAneg1Neg1Ptr) += here->Ay1;

        *(here->NTRApos2Pos2Ptr) += here->Ay1;
        *(here->NTRApos2Neg2Ptr) -= here->Ay1;
        *(here->NTRAneg2Pos2Ptr) -= here->Ay1;
        *(here->NTRAneg2Neg2Ptr) += here->Ay1;


/* STAMP FILL FOR DC ANALYSIS */
        if(ckt->CKTmode & MODEDC) {
/*            *(here->NTRAibr1Pos2Ptr) -= 1; */
/*            *(here->NTRAibr1Neg2Ptr) += 1; */
/*            *(here->NTRAibr1Ibr2Ptr) -= (1-ckt->CKTgmin)*here-
>NTRAimped;*/
/*            *(here->NTRAibr2Pos1Ptr) -= 1; */
/*            *(here->NTRAibr2Neg1Ptr) += 1; */
/*            *(here->NTRAibr2Ibr1Ptr) -= (1-ckt->CKTgmin)*here-
>NTRAimped;*/
        } else {
/* NOT DOING A DC ANALYSIS MATRIX FILL */
          if (ckt->CKTmode & MODEINITTRAN) {
/* THE INITIAL TRANSIENT RUN */
```

```
            if(ckt->CKTmode & MODEUIC) {
/* USE THE INITIAL CONDITIONS SUPPLIED INSTEAD OF THE DC
ANALYSIS VALUES */

                xw1 = here->Ay2* here->NTRAinitVolt2
                + here->NTRAinitCur2;

                xwold = xw1;

                here->NTRAinput1 = G(1,here->ay1,here->fcy1,here->My1,
                ckt->CKTtime,here->NTRAinitVolt1,here->Awb,
                here->awb,here->fcwb,here->Mwb,ckt->CKTtime,xw1,
                xwold,&(here->Is1));

                here->xwold1 = xw1;


                xw2 = here->Ay1 * here->NTRAinitVolt1
                + here->NTRAinitCur1;

                xwold = xw2;

                here->NTRAinput2 = G(2,here->ay2,here->fcy2,here->My2,
                ckt->CKTtime,here->NTRAinitVolt2,here->Awf,
                here->awf,here->fcwf,here->Mwf,ckt->CKTtime,xw2,
                xwold,&(here->Is2));

                here->xwold2 = xw2;


            } else {
/* COMMENTED OUT TILL DEBUGGING FINISHED */
/* USE THE DC VALUES AS START */
/*
                here->NTRAinput1 = *(ckt->CKTrhsOld + here->NTRAbrEq2)
                + here->NTRAconduct * ( *(ckt->CKTrhsOld
                + here->NTRAposNode2) - *(ckt->CKTrhsOld
                + here->NTRAnegNode2) );


                here->NTRAinput2 = *(ckt->CKTrhsOld + here->NTRAbrEq1)
                + here->NTRAconduct * ( *(ckt->CKTrhsOld
                + here->NTRAposNode1) - *(ckt->CKTrhsOld
                + here->NTRAnegNode1) );
*/

            }

/* SET UP THE DELAY TABLE */
            *(here->NTRAdelays  ) = -2*here->NTRAtd;
            *(here->NTRAdelays +3) = -here->NTRAtd;
            *(here->NTRAdelays+6) = 0;
            *(here->NTRAdelays+1) = *(here->NTRAdelays +4) =
                *(here->NTRAdelays+7) = here->NTRAinput1;
```

```
                *(here->NTRAdelays+2) = *(here->NTRAdelays +5) =
                    *(here->NTRAdelays+8) = here->NTRAinput2;
                here->NTRAsizeDelay = 2;


            } else {

/* FIND INTERPOLATED VALUES */
                if(ckt->CKTmode & MODEINITPRED) {
                  for(i=2;(i<here->NTRAsizeDelay) &&
                     (*(here->NTRAdelays +3*i) <=
                     (ckt->CKTtime-here->NTRAtd));i++) {;/*loop does it*/}
                  t1 = *(here->NTRAdelays + (3*(i-2)));
                  t2 = *(here->NTRAdelays + (3*(i-1)));
                  t3 = *(here->NTRAdelays + (3*(i  )));
                  if( (t2-t1)==0  || (t3-t2) == 0) continue;
                  f1 = (ckt->CKTtime - here->NTRAtd - t2) *
                      (ckt->CKTtime - here->NTRAtd - t3) ;
                  f2 = (ckt->CKTtime - here->NTRAtd - t1) *
                      (ckt->CKTtime - here->NTRAtd - t3) ;
                  f3 = (ckt->CKTtime - here->NTRAtd - t1) *
                      (ckt->CKTtime - here->NTRAtd - t2) ;
                  if((t2-t1)==0) { /* should never happen, but don't want
                                * to divide by zero, EVER... */
                    f1=0;
                    f2=0;
                  } else {
                    f1 /= (t1-t2);
                    f2 /= (t2-t1);
                  }
                  if((t3-t2)==0) { /* should never happen, but don't want
                                * to divide by zero, EVER... */
                    f2=0;
                    f3=0;
                  } else {
                    f2 /= (t2-t3);
                    f3 /= (t2-t3);
                  }
                  if((t3-t1)==0) { /* should never happen, but don't want
                                * to divide by zero, EVER... */
                    f1=0;
                    f2=0;
                  } else {
                    f1 /= (t1-t3);
                    f3 /= (t1-t3);
                  }
.
.
.
                  xw1 = f1 * *(here->NTRAdelays + (3*(i-2))+1)
                            + f2 * *(here->NTRAdelays + (3*(i-1))+1)
                            + f3 * *(here->NTRAdelays + (3*(i  ))+1);

                  Ty = ckt->CKTtime - here->oldtime;
```

```
                Tw = t3 - t2;


                xyold = *(ckt->CKTrhsOld + here->NTRAposNode1)
                - *(ckt->CKTrhsOld + here->NTRAnegNode1);

                here->NTRAinput1 = G(1,here->ay1,here->fcy1,here->My1,
                Ty,xyold,here->Awb,here->awb,here->fcwb,
                here->Mwb,Tw,xw1,here->xwold1,&(here->Is1));

                here->xwold1 = xw1;


                xw2 = f1 * *(here->NTRAdelays + (3*(i-2))+2)
                          + f2 * *(here->NTRAdelays + (3*(i-1))+2)
                          + f3 * *(here->NTRAdelays + (3*(i  ))+2);

                xyold = *(ckt->CKTrhsOld + here->NTRAposNode2)
                - *(ckt->CKTrhsOld + here->NTRAnegNode2);


                here->NTRAinput2 = G(2,here->ay2,here->fcy2,here->My2,
                Ty,xyold,here->Awf,here->awf,here->fcwf,
                here->Mwf,Tw,xw2,here->xwold2,&(here->Is2));

                here->xwold2 = xw2;

            }
          }
/* FILL THE RIGHT HAND SIDE */
        here->oldtime = ckt->CKTtime;
        *(ckt->CKTrhs + here->NTRAposNode1) += here->NTRAinput1;
        *(ckt->CKTrhs + here->NTRAnegNode1) -= here->NTRAinput1;
        *(ckt->CKTrhs + here->NTRAposNode2) += here->NTRAinput2;
        *(ckt->CKTrhs + here->NTRAnegNode2) -= here->NTRAinput2;

        }
      }
    }
   return(OK);
}
```

C.7. NTRAacct

The following two subsections contain excerpts from spice3e2/src/lib/dev/ntra/ntraacct.c of the function NTRAacct. The first subsection shows a listing before conversion to a function of the lossy package with code to be modified boxed in. The second section shows the modified version.

C.7.1. Contents of ntraacct.c before modification

```
.
.
.

int
NTRAaccept(ckt,inModel)
    register CKTcircuit *ckt;
    GENmodel *inModel;
{
    register NTRAmodel *model = (NTRAmodel *)inModel;
    register NTRAinstance *here;
.
.
.

/* Inserting data into table */
            to = (here->NTRAdelays  +3*here->NTRAsizeDelay);
            *to = ckt->CKTtime;
```

```
            to = (here->NTRAdelays+1+3*here->NTRAsizeDelay);
            *to = 2*here->NTRAconduct * ( *(ckt->CKTrhsOld +
                  here->NTRAposNode2) - *(ckt->CKTrhsOld +
                  here->NTRAnegNode2) ) - here->NTRAinput2;

            to = (here->NTRAdelays+2+3*here->NTRAsizeDelay);
            *to = 2*here->NTRAconduct * ( *(ckt->CKTrhsOld +
                  here->NTRAposNode1) - *(ckt->CKTrhsOld +
                  here->NTRAnegNode1) ) - here->NTRAinput1;
```

```
.
.
.
```

C.7.2. Contents of ntraacct.c after modification

```
.
.
.
int
NTRAaccept(ckt,inModel)
    register CKTcircuit *ckt;
    GENmodel *inModel;
{
    register NTRAmodel *model = (NTRAmodel *)inModel;
    register NTRAinstance *here;
.
```

```
.
.

/* Inserting data into table */
            to = (here->NTRAdelays  +3*here->NTRAsizeDelay);
            *to = ckt->CKTtime;

            to = (here->NTRAdelays+1+3*here->NTRAsizeDelay);

            *to = 2*(here->Ay2 * ( *(ckt->CKTrhsOld + here->NTRAposNode2) -
*(ckt->CKTrhsOld + here->NTRAnegNode2) ) - here->NTRAinput2 ) + here-
>Is2;

            to = (here->NTRAdelays+2+3*here->NTRAsizeDelay);

            *to = 2*(here->Ay1 * ( *(ckt->CKTrhsOld + here->NTRAposNode1) -
*(ckt->CKTrhsOld + here->NTRAnegNode1) ) - here->NTRAinput1 ) + here-
>Is1;
.
.
.
```

C.8. NTRAtrunc

This section contains excerpts from the NTRAtrunc function found in spice3e2/src/lib/dev/ntra/ntratrunc.c. The first section contains the listing of NTRAtrunc as a lossless line function and the second section contains the listing of NTRAtrunc as a lossy line function.

C.8.1. Contents of ntratrunc.c before modification

```
.
.
.

int
NTRAtrunc(inModel,ckt,timeStep)
   GENmodel *inModel;
   register CKTcircuit *ckt;
   double *timeStep;

{
   register NTRAmodel *model = (NTRAmodel *)inModel;
   register NTRAinstance *here;
    double v1,v2,v3,v4;
    double v5,v6,d1,d2,d3,d4;
   double tmp;
```

```
/*  loop through all the NTRAnsmission line models */
for( ; model != NULL; model = model->NTRAnextModel ) {

  /* loop through all the instances of the model */
  for (here = model->NTRAinstances; here != NULL ;
       here=here->NTRAnextInstance) {
    v1 = ( *(ckt->CKTrhsOld + here->NTRAposNode2)
      - *(ckt->CKTrhsOld + here->NTRAnegNode2))
      + here->NTRAinput2 * here->NTRAimped;
    v2 = *(here->NTRAdelays+1+3*(here->NTRAsizeDelay));
    v3 = *(here->NTRAdelays+1+3*(here->NTRAsizeDelay-1));
    v4 = ( *(ckt->CKTrhsOld + here->NTRAposNode1)
      - *(ckt->CKTrhsOld + here->NTRAnegNode1))
      + here->NTRAinput1 * here->NTRAimped;
    v5 = *(here->NTRAdelays+2+3*(here->NTRAsizeDelay));
    v6 = *(here->NTRAdelays+2+3*(here->NTRAsizeDelay-1));
    d1 = (v1-v2)/ckt->CKTdeltaOld[1];
    d2 = (v2-v3)/ckt->CKTdeltaOld[2];
    d3 = (v4-v5)/ckt->CKTdeltaOld[1];
    d4 = (v5-v6)/ckt->CKTdeltaOld[2];
.
.
.
```

C.8.2. Contents of ntratrunc.c after modification

```
.
.
.
int
NTRAtrunc(inModel,ckt,timeStep)
  GENmodel *inModel;
  register CKTcircuit *ckt;
  double *timeStep;

{
  register NTRAmodel *model = (NTRAmodel *)inModel;
  register NTRAinstance *here;
  double v1,v2,v3,v4;
  double v5,v6,d1,d2,d3,d4;
  double tmp;

  /*  loop through all the NTRAnsmission line models */
  for( ; model != NULL; model = model->NTRAnextModel ) {

    /* loop through all the instances of the model */
    for (here = model->NTRAinstances; here != NULL ;
         here=here->NTRAnextInstance) {
      v1 = 2*(here->Ay2 * ( *(ckt->CKTrhsOld + here->NTRAposNode2)
```

```
        - *(ckt->CKTrhsOld + here->NTRAnegNode2) ) -
        here->NTRAinput2 ) + here->Is2;
      v2 = *(here->NTRAdelays+1+3*(here->NTRAsizeDelay));
      v3 = *(here->NTRAdelays+1+3*(here->NTRAsizeDelay-1));
      v4 = 2*(here->Ay1 * ( *(ckt->CKTrhsOld + here->NTRAposNode1)
        - *(ckt->CKTrhsOld + here->NTRAnegNode1) ) -
        here->NTRAinput1 ) + here->Is1;
      v5 = *(here->NTRAdelays+2+3*(here->NTRAsizeDelay));
      v6 = *(here->NTRAdelays+2+3*(here->NTRAsizeDelay-1));
.
.
.
```

C.9. Listing of the Function fileread

The function fileread is found in spice3e2/src/lib/dev/ntra/fileread.c and the function reads in parameters from the file of difference parameters and sets the appropriate fields of the device specific data structures.

```
.
.
.
/*
  *******======>>>>>>>> IMPORTANT <<<<<<<<======********
  LATER change all exits to returns for cleaner exiting of the program
  i.e., let SPICE handle the fact that the simulation can not proceed.
*/

int fileread( NTRAinstance *here ) {
  FILE *data;
  int i;
  double test;

  if ( (data = fopen( here->NTRAfileName, "r")) == NULL ) {
    printf( "***ERROR: fileread: could not access the difference parameters file
-exiting...\n\n" );
    return (0);
  }


  fscanf (data, "%d", &(here->Mwf) );
  if ( !( here->awf= malloc( (here->Mwf)*sizeof(double)) ) ) {
    printf ("Out of memory.\n");
    exit(1);
  }
  if ( !( here->fcwf= malloc( (here->Mwf)*sizeof(double)) ) ) {
    printf ("Out of memory.\n");
    exit(1);
```

```
}
for(i=0; i< here->Mwf; ++i) {
 fscanf(data, "%le %le",(here->awf)+i,(here->fcwf)+i);
}


fscanf(data, "%le %le %le", &(here->Awf), &(here->Bwf), &(here->tauf) ) ;
fscanf(data, "%d", &(here->Mwb) );
if ( !( here->awb= malloc( (here->Mwb)*sizeof(double)) ) ) {
 printf ("Out of memory.\n");
 exit(1);
}
if ( !( here->fcwb= malloc( (here->Mwb)*sizeof(double)) ) ) {
 printf ("Out of memory.\n");
 exit(1);
}
for(i=0; i< here->Mwb; ++i) {
 fscanf(data, "%le %le",(here->awb)+i,(here->fcwb)+i);
}
fscanf(data, "%le %le %le", &(here->Awb), &(here->Bwb), &(here->taub)
        );


fscanf( data, "%d", &(here->My1) );
if ( !( here->ay1= malloc( (here->My1)*sizeof(double)) ) ) {
 printf ("Out of memory.\n");
 exit(1);
}
if ( !( here->fcy1= malloc( (here->My1)*sizeof(double)) ) ) {
 printf ("Out of memory.\n");
 exit(1);
}
for(i=0;i< here->My1;++i) {
 fscanf(data, "%le %le",(here->ay1)+i,(here->fcy1)+i);
}
fscanf(data, "%le %le", &(here->Ay1), &(here->By1) );


fscanf( data, "%d", &(here->My2) );
if ( !( here->ay2= malloc( (here->My2)*sizeof(double)) ) ) {
 printf ("Out of memory.\n");
 exit(1);
}
if ( !( here->fcy2= malloc( (here->My2)*sizeof(double)) ) ) {
 printf ("Out of memory.\n");
 exit(1);
}
for( i=0; i< here->My2; ++i ) {
 fscanf(data, "%le %le",(here->ay2)+i,(here->fcy2)+i);
}
fscanf(data, "%le %le", &(here->Ay2), &(here->By2) );
return( 1 );
}
```

C.10. IFparm Table

This section contains excerpts from the file spice3e2/src/lib/dev/ntra/ntra.c. The file ntra.c contains the interface parameter table. The following listing shows the parameter table after modification.

```
.
.
.
IFparm NTRApTable[] = { /* parameters */
 IOP( "z0", NTRA_Z0,  IF_REAL   , "Characteristic impedance"),
 IOP( "zo", NTRA_Z0,  IF_REAL   , "Characteristic impedance"),
 IOP( "f",  NTRA_FREQ, IF_REAL   , "Frequency"),
 IOP( "td", NTRA_TD,  IF_REAL   , "Transmission delay"),
 IOP( "nl", NTRA_NL,  IF_REAL   , "Normalized length at frequency given"),
 IOP( "v1", NTRA_V1,  IF_REAL   , "Initial voltage at end 1"),
 IOP( "v2", NTRA_V2,  IF_REAL   , "Initial voltage at end 2"),
 IOP( "i1", NTRA_I1,  IF_REAL   , "Initial current at end 1"),
 IOP( "i2", NTRA_I2,  IF_REAL   , "Initial current at end 2"),
 IOP( "filename", NTRA_PARAM_FILE_NAME,  IF_STRING,
       "Line parameter file name"),
 IP("ic", NTRA_IC,  IF_REALVEC,"Initial condition vector:v1,i1,v2,i2"),
 OP( "rel", NTRA_RELTOL,  IF_REAL   , "Rel. rate of change of deriv. for
bkpt"),
 OP( "abs", NTRA_ABSTOL,  IF_REAL   , "Abs. rate of change of deriv. for
bkpt"),
 OP( "pos_node1", NTRA_POS_NODE1,IF_INTEGER,"Positive node of end 1
of t. line"),
 OP( "neg_node1", NTRA_NEG_NODE1,IF_INTEGER,"Negative node of end 1
of t. line"),
 OP( "pos_node2", NTRA_POS_NODE2,IF_INTEGER,"Positive node of end 2
of t. line"),
 OP( "neg_node2", NTRA_NEG_NODE2,IF_INTEGER,"Negative node of end 2
of t. line"),
 OP( "delays",NTRA_DELAY, IF_REALVEC, "Delayed values of excitation")
 };

/*static IFparm NTRAmPTable[] = { /* model parameters */
/*};/**/

char *NTRAnames[] = {
    "P1+",
    "P1-",
    "P2+",
    "P2-"
};

int     NTRAnSize = NUMELEMS(NTRAnames);
int     NTRApTSize = NUMELEMS(NTRApTable);
```

```
int     NTRAmPTSize = 0;
int     NTRAiSize = sizeof(NTRAinstance);
int     NTRAmSize = sizeof(NTRAmodel);
```

C.11.NTRAparam

This section lists an excerpt from spice3e2/src/li/dev/ntra/ntraparam.c which contains the NTRAparam function. The listing shows the modified version of NTRAparam, making it a part of the lossy line package.

```
.
.
.
/* ARGSUSED */
int
NTRAparam(param,value,inst,select)
    int param;
    IFvalue *value;
    GENinstance *inst;
    IFvalue *select;
{
    NTRAinstance *here = (NTRAinstance *)inst;

    switch(param) {
      case NTRA_RELTOL:
        here->NTRAreltol = value->rValue;
        here->NTRAreltolGiven = TRUE;
        break;

      case NTRA_ABSTOL:
        here->NTRAabstol = value->rValue;
        here->NTRAabstolGiven = TRUE;
        break;

      case NTRA_Z0:
/*          here->NTRAimped = value->rValue;
*/
          here->NTRAimpedGiven = TRUE;
          break;
```

```
.
.
.
    case NTRA_PARAM_FILE_NAME:
      here->NTRAfileName = value->sValue;
      if ( fileread( here ) ) {
          here->NTRAparamfileGiven = TRUE;
       here->NTRAtd = here->tauf;      /* tauf = taub */
       here->NTRAtdGiven = TRUE;
      }
      break;
.
.
.
```

APPENDIX D.
FUTURE CODE MODIFICATIONS

This appendix contains listings of source code which is discussed in Chapter 9. The code is as yet unimplemented and is only an example of the types of modifications which can be made to SPICE3E2 in order to make the program more easy to maintain and modify.

D.1. Complete listing of NTRAload

The following is a complete listing of the function NTRAload found in spice3e2/src/lib/ntralaod.c, except for the exclusion of diagnostic print statements and the copyright notice.

```
/*
 */

#include "spice.h"
#include <stdio.h>
#include "util.h"
#include "cktdefs.h"
#include "ntradefs.h"
#include "trandefs.h"
#include "sperror.h"
#include "suffix.h"
#include "vdmmodel.h"

#define HMAX 1.0e-10

/*ARGSUSED*/
int
NTRAload(inModel,ckt)
   GENmodel *inModel;
   CKTcircuit *ckt;
      /* actually load the current values into the
       * sparse matrix previously provided
       */
{
```

```
/* Variables declared for inside the function */
   register NTRAmodel *model = (NTRAmodel *)inModel;
   register NTRAinstance *here;
   double t1,t2,t3;
   double f1,f2,f3;
   register int i;

/* extra vars */
   int L;
   double xw1, xw2;
   double Ty;
   double delay;
   double xyold;
   double Tw;
   double xw;
   double xwold;
   double zero;
/****************/

   /*  loop through all the NTRAnsmission line models */
   for( ; model != NULL; model = model->NTRAnextModel ) {

     /* loop through all the instances of the model */
     for (here = model->NTRAinstances; here != NULL ;
          here=here->NTRAnextInstance) {

/* MOST OF THE STAMP FILLED HERE */
        *(here->NTRApos1Pos1Ptr) += here->Ay1;
        *(here->NTRApos1Neg1Ptr) -= here->Ay1;
        *(here->NTRAneg1Pos1Ptr) -= here->Ay1;
        *(here->NTRAneg1Neg1Ptr) += here->Ay1;

        *(here->NTRApos2Pos2Ptr) += here->Ay1;
        *(here->NTRApos2Neg2Ptr) -= here->Ay1;
        *(here->NTRAneg2Pos2Ptr) -= here->Ay1;
        *(here->NTRAneg2Neg2Ptr) += here->Ay1;


/* STAMP FILL FOR DC ANALYSIS */
       if(ckt->CKTmode & MODEDC) {
/*           *(here->NTRAibr1Pos2Ptr) -= 1; */
/*           *(here->NTRAibr1Neg2Ptr) += 1; */
/*           *(here->NTRAibr1Ibr2Ptr) -= (1-ckt->CKTgmin)*here-
>NTRAimped;*/
/*           *(here->NTRAibr2Pos1Ptr) -= 1; */
/*           *(here->NTRAibr2Neg1Ptr) += 1; */
/*           *(here->NTRAibr2Ibr1Ptr) -= (1-ckt->CKTgmin)*
              here->NTRAimped;*/
       } else {
/* NOT DOING A DC ANALYSIS MATRIX FILL */
         if (ckt->CKTmode & MODEINITTRAN) {
/* THE INITIAL TRANSIENT RUN */
           if(ckt->CKTmode & MODEUIC) {
```

```
/* USE THE INITIAL CONDITIONS SUPPLIED INSTEAD OF THE DC
ANALYSIS VALUES */

                xw1 = here->Ay2* here->NTRAinitVolt2
                        + here->NTRAinitCur2;

                xwold = xw1;
                here->NTRAinput1 = G(1,here->ay1,here->fcy1,here->My1,
                ckt->CKTtime,here->NTRAinitVolt1,here->Awb,
                here->awb,here->fcwb,here->Mwb,ckt->CKTtime,xw1,
                xwold,&(here->Is1));
                here->xwold1 = xw1;


                xw2 = here->Ay1 * here->NTRAinitVolt1
                        + here->NTRAinitCur1;

                xwold = xw2;
                here->NTRAinput2 = G(2,here->ay2,here->fcy2,here->My2,
                ckt->CKTtime,here->NTRAinitVolt2,here->Awf,
                here->awf,here->fcwf,here->Mwf,ckt->CKTtime,xw2,
                xwold,&(here->Is2));
                here->xwold2 = xw2;


            } else {
/* COMMENTED OUT TILL DEBUGGING FINISHED */
/* USE THE DC VALUES AS START */
/*
                here->NTRAinput1 = *(ckt->CKTrhsOld + here->NTRAbrEq2)
                + here->NTRAconduct * ( *(ckt->CKTrhsOld
                + here->NTRAposNode2) - *(ckt->CKTrhsOld
                + here->NTRAnegNode2) );


                here->NTRAinput2 = *(ckt->CKTrhsOld + here->NTRAbrEq1)
                + here->NTRAconduct * ( *(ckt->CKTrhsOld
                + here->NTRAposNode1) - *(ckt->CKTrhsOld
                + here->NTRAnegNode1) );
*/

            }

/* SET UP THE DELAY TABLE */
            *(here->NTRAdelays  ) = -2*here->NTRAtd;
            *(here->NTRAdelays +3) = -here->NTRAtd;
            *(here->NTRAdelays+6) = 0;
            *(here->NTRAdelays+1) = *(here->NTRAdelays +4) =
                *(here->NTRAdelays+7) = here->NTRAinput1;
            *(here->NTRAdelays+2) = *(here->NTRAdelays +5) =
                *(here->NTRAdelays+8) = here->NTRAinput2;
            here->NTRAsizeDelay = 2;
```

```
        } else {

/* FIND INTERPOLATED VALUES */
            if(ckt->CKTmode & MODEINITPRED) {
              for(i=2;(i<here->NTRAsizeDelay) &&
                 (*(here->NTRAdelays +3*i) <=
                 (ckt->CKTtime-here->NTRAtd));i++) {;/*loop does it*/}
              t1 = *(here->NTRAdelays + (3*(i-2)));
              t2 = *(here->NTRAdelays + (3*(i-1)));
              t3 = *(here->NTRAdelays + (3*(i  )));
              if( (t2-t1)==0  || (t3-t2) == 0) continue;
              f1 = (ckt->CKTtime - here->NTRAtd - t2) *
                 (ckt->CKTtime - here->NTRAtd - t3) ;
              f2 = (ckt->CKTtime - here->NTRAtd - t1) *
                 (ckt->CKTtime - here->NTRAtd - t3) ;
              f3 = (ckt->CKTtime - here->NTRAtd - t1) *
                 (ckt->CKTtime - here->NTRAtd - t2) ;
              if((t2-t1)==0) { /* should never happen, but don't want
                          * to divide by zero, EVER... */
                f1=0;
                f2=0;
              } else {
                f1 /= (t1-t2);
                f2 /= (t2-t1);
              }
              if((t3-t2)==0) { /* should never happen, but don't want
                          * to divide by zero, EVER... */
                f2=0;
                f3=0;
              } else {
                f2 /= (t2-t3);
                f3 /= (t2-t3);
              }
              if((t3-t1)==0) { /* should never happen, but don't want
                          * to divide by zero, EVER... */
                f1=0;
                f2=0;
              } else {
                f1 /= (t1-t3);
                f3 /= (t1-t3);
              }
              xw1 = f1 * *(here->NTRAdelays + (3*(i-2))+1)
                        + f2 * *(here->NTRAdelays + (3*(i-1))+1)
                        + f3 * *(here->NTRAdelays + (3*(i  ))+1);

              Ty = ckt->CKTtime - here->oldtime;
              Tw = t3 - t2;


              xyold = *(ckt->CKTrhsOld + here->NTRAposNode1)
                        - *(ckt->CKTrhsOld + here->NTRAnegNode1);

              here->NTRAinput1 = G(1,here->ay1,here->fcy1,here->My1,
              Ty,xyold,here->Awb,here->awb,here->fcwb,
```

```
                here->Mwb,Tw,xw1,here->xwold1,&(here->Is1));
                here->xwold1 = xw1;


                xw2 = f1 * *(here->NTRAdelays + (3*(i-2))+2)
                          + f2 * *(here->NTRAdelays + (3*(i-1))+2)
                          + f3 * *(here->NTRAdelays + (3*(i  ))+2);

                xyold = *(ckt->CKTrhsOld + here->NTRAposNode2)
                          - *(ckt->CKTrhsOld + here->NTRAnegNode2);


                here->NTRAinput2 = G(2,here->ay2,here->fcy2,here->My2,
                Ty,xyold,here->Awf,here->awf,here->fcwf,
                here->Mwf,Tw,xw2,here->xwold2,&(here->Is2));
                here->xwold2 = xw2;

              }
            }
/* FILL THE RIGHT HAND SIDE */
        here->oldtime = ckt->CKTtime;
        *(ckt->CKTrhs + here->NTRAposNode1) += here->NTRAinput1;
        *(ckt->CKTrhs + here->NTRAnegNode1) -= here->NTRAinput1;
        *(ckt->CKTrhs + here->NTRAposNode2) += here->NTRAinput2;
        *(ckt->CKTrhs + here->NTRAnegNode2) -= here->NTRAinput2;

        }
      }
    }
    return(OK);
}
```

D.2. Modified NTRAload

The following listing is of a modified and as yet unimplemented version of the NTRAload function.

```
#include "ntraload.h"

int
NTRAload(inModel,ckt)
    GENmodel *inModel;
    CKTcircuit *ckt;
      /* actually load the current values into the
       * sparse matrix previously provided
       */
{
    /* Variables declared for inside the function */
    register NTRAmodel *model = (NTRAmodel *)inModel;
```

```
    register NTRAinstance *here;

    /*  loop through all the NTRAnsmission line models */
    for( ; model != NULL; model = model->NTRAnextModel ) {

        /* loop through all the instances of the model */
        for ( here = model->NTRAinstances; here != NULL ;
              here=here->NTRAnextInstance ) {

            /* MOST OF THE STAMP FILLED HERE */
            NTRAloadLHS( here );


            if( ckt->CKTgetMode( ) == SPICE_DC ) {
                /* STAMP FILL FOR DC ANALYSIS */
                NTRAdcLoad( here );

            } else {
                /* NOT DOING A DC ANALYSIS MATRIX FILL */

                if ( ckt->CKTgetMode( ) == SPICE_INITTRAN ) {
                    /* THE INITIAL TRANSIENT RUN */

                    if( ckt->CKTgetMode( ) == SPICE_UIC ) {
                        /* USE THE INITIAL CONDITIONS
                        SUPPLIED INSTEAD OF THE DC
                        ANALYSIS VALUES */
                        NTRAloadUIC( here );

                    } else {
                        /* USE THE DC VALUES AS A START */
                        NTRAloadUDC( here );

                    }

                    /* SET UP THE DELAY TABLE */
                    NTRAinitDelTab( here );

                } else {
                    /* LOAD THE SOURCE VALUES */
                    NTRAcalcRHS( here );

                }
            }
                /* FILL THE RIGHT HAND SIDE */
                NTRAloadRHS( here );

            }
        }
    }
    return(OK);
}
```

D.3. NTRAcalcRHS

The following listing is of a function called NTRAcalcRHS. The function is not yet implemented.

```
#include "ntracalcrhs.h"

void NTRAcalcRHS( NTRAinstance *here ) {

        here->NTRAxw1 = NTRAgetInterpExcit1( );
        here->NTRAxyold = ckt->CKTgetSol( here->NTRAposNode1 )
                        - ckt->CKTgetSol( here->NTRAnegNode1 );
        here->NTRAinput1 = G( 1, here );
        here->NTRAxwold1 = here->NTRAxw1;

        here->NTRAxw2 = NTRAgetInterpExcit2( );
        here->NTRAxyold = ckt->CKTgetSol( here->NTRAposNode2 )
                        - ckt->CKTgetSol( here->NTRAnegNode2 );
        here->NTRAinput2 = G( 2, here );
        here->NTRAxwold2 = here->NTRAxw2;

        here->oldtime = ckt->CKTtime;

}
```

D.4. NTRAgetInterpExcit1

The following listing is of a function called NTRAInterpExcit1. The function is not yet implemented.

```
#include "ntragetintexcit1.h"

double NTRAgetInterpExcit1( NTRAinstance *here ) {

        int i;
        double t1, t2, t3;
        double f1, f2, f3;
        double excit1;

        i = NTRAgetDelTabIndGrtr( ckt->CKTtime - here->NTRAtd );
        t1 = NTRAgetDelTabTime( i-2 );
        t2 = NTRAgetDelTabTime( i-1 );
        t3 = NTRAgetDelTabTime( i );

        if( (t2-t1)==0  || (t3-t2) == 0) continue;
```

```
        f1 = (ckt->CKTtime - here->NTRAtd - t2) *
             (ckt->CKTtime - here->NTRAtd - t3) ;
        f2 = (ckt->CKTtime - here->NTRAtd - t1) *
             (ckt->CKTtime - here->NTRAtd - t3) ;
        f3 = (ckt->CKTtime - here->NTRAtd - t1) *
             (ckt->CKTtime - here->NTRAtd - t2) ;

        if((t2-t1)==0) { /* should never happen, but don't want
                          * to divide by zero, EVER... */
                f1=0;
                f2=0;
        } else {
                f1 /= (t1-t2);
                f2 /= (t2-t1);
        }
        if((t3-t2)==0) { /* should never happen, but don't want
                          * to divide by zero, EVER... */
                f2=0;
                f3=0;
        } else {
                f2 /= (t2-t3);
                f3 /= (t2-t3);
        }
        if((t3-t1)==0) { /* should never happen, but don't want
                          * to divide by zero, EVER... */
                f1=0;
                f2=0;
        } else {
                f1 /= (t1-t3);
                f3 /= (t1-t3);
        }

        excit1 = f1 * NTRAgetDelTabExc1( i-2 )
                   + f2 * NTRAgetDelTabExc1( i-1 )
                   + f3 * NTRAgetDelTabExc1( i );

        return excit1;

}
```

D.5. NTRAgetDelTabIndGrtr

The following listing is of a function called NTRAgetDelTabIndGrtr. The function is not yet implemented.

```
#include "ntragetdtindg.h"

int NTRAgetDelTabIndGrtr( double time ) {
```

```
        int i;

        for( i=2; ( i < NTRAgetDelTabSize( ) &&
             NTRAgetDelTabTime( i ) <= time ); i++ ) {;
                /* loop to determine the index of the time value just greater
                   than the value of time */
        }

        return i;

}
```

REFERENCES

[1] J.E. Schutte-Aine and R. Mittra, *Modeling and Simulation of High-Speed Digital Circuit Interconnections.* Urbana, Illinois: Technical Report No. 88-2, Electromagnetic Communication Laboratory, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, April 1988.

[2] J. S. Roychowdry and D. O. Pederson, "Efficient transient simulation of lossy interconnect," 28th ACM/IEEE Design Automation Conference, Paper 42.1, pp 740-745, 1991.

[3] C. Warren, "Realizing a transmission line model," IEEE MICRO, vol. , no. pp 76-79, June 1990.

[4] D.B. Kuznetsov and J. E. Schutt-Aine, *Transmission Line Modeling and Transient Simulation.* Urbana, Illinois: Technical Report no. 92-4, Electromagnetic Communication Laboratory, Department of Electrical and Computer Engineering, University of Illinois at Urbana-Champaign, December 1992.

[5] T. L. Quarles, *Adding Devices to SPICE3.* Memorandum No. UCB/ERL M89/45, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, April 1989.

[6] T. L. Quarles, *The Spice3 Implementation Guide.* Memorandum No. UCB/ERL M89/44, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, April 1989.

[7] T. L. Quarles, *Analysis of Performance and Convergence Issues for Circuit Simulation.* Memorandum No. UCB/ERL M89/42, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, April 1989.

[8] W. Christopher, J. Hsu, and T. l. Quarles, *A Short Introduction to SPICE3.* CAD Group U.C. Berkeley, May 1989.

[9] B. Johnson, T. L. Quarles, A. R. Newton, D. O. Pederson, and A. Sangiovanni-Vincentelli, *SPICE3 Version 3e User's Manual.* Department of Electrical Engineering and Computer Sciences, University of California, Berkeley, CA 94720, April 1991.

[10] J. Hsu, *Nutneg Programmer's Guide.* document with SPICE3 Release C1, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, April 1989.

[11] T. L. Quarles, *Benchmark Circuits: Results for Spice3.* Memorandum No. UCB/ERL M89/47, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, April 1989.

[12] H. Schildt, *C the Complete Reference, 2nd ed.* Berkeley, CA: Osborne McGraw-Hill, 1990.

[13] S. I. Pearson and G. J. Maler, *Introductory Circuit Analysis.* New York: John Wiley and Sons, Inc., 1960.

[14] J. Vlach and K. Singhal, *Computer Methods for Circuit Analysis and Design.* New York: Van Nostrand Reinhold Co., 1983.

[15] L. O. Chua and P. Lin, *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques.* Englewood Cliffs, CA: Prentice-Hall, 1975.

[16] A. E. Ruehli, *Circuit Analysis, Simulation and Design, Part I.* North-Holland (Elseviers Science Pub. Co.), 1987.

[17] W. J. McCalla, *Fundamentals of Computer-Aided Circuit Simulation.* Boston: Kluwar Academic Publishers, 1988.

[18] B. Stroustrup, *The C++ Programming Language 2nd Ed.* Reading, MA: Addison-Wesley, 1990.

[19] M. G. Sobell, *A Practical Guide to Unix System V.* Redwood City, CA: The Benjamin/Cummings Publishing Company, Inc., 1991.