DESIGN AND SOFTWARE IMPLEMENTATION OF LIBRARY FUNCTIONS FOR ELECTRONIC CIRCUIT SIMULATION

A THESIS
SUBMITTED TO THE DEPARTMENT OF
ELECTRICAL AND ELECTRONICS ENGINEERING
AND THE INSTITUTE OF ENGINEERING AND SCIENCES
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
MASTER OF SCIENCE

By
Mustafa Nazım YAZGAN
September 1994

DESIGN AND SOFTWARE IMPLEMENTATION OF LIBRARY FUNCTIONS FOR ELECTRONIC CIRCUIT SIMULATION

A THESIS

SUBMITTED TO THE DEPARTMENT OF ELECTRICAL AND

ELECTRONICS ENGINEERING

AND THE INSTITUTE OF ENGINEERING AND SCIENCES

OF BILKENT UNIVERSITY

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

FOR THE DEGREE OF

MASTER OF SCIENCE

By

Mustafa Nazım Yazgan

1994

Mustafa Nazım Yazgan

tarafından Değişlanmıştır.

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. M. Ali Tan (Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Prof. Dr. Abdullah Atalar (Co-Supervisor)

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. Ömer Morgül

I certify that I have read this thesis and that in my opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Asst. Prof. Dr. David Davenport

Approved for the Institute of Engineering and Sciences:

Prof. Dr. Mehmet Baray
Director of Institute of Engineering and Sciences

ABSTRACT

DESIGN AND SOFTWARE IMPLEMENTATION OF LIBRARY FUNCTIONS FOR ELECTRONIC CIRCUIT SIMULATION

Mustafa Nazım Yazgan
M.S. in Electrical and Electronics Engineering
Supervisor: Assoc. Prof. Dr. M. Ali Tan
Co-Supervisor: Prof. Dr. Abdullah Atalar
1994

In this thesis, the software implementation of "SIMLIB", a library of electronic circuit simulation, is described. The building blocks of a circuit simulator, namely, input parser, types of analyses, methods and matrix solvers are discussed. The algorithms and some special techniques employed in this library, as well as ways of modifying the program are explained. SIMLIB has become a good environment for the researchers to try and develop new ideas on circuit simulation.

Keywords : Computer-Aided Design, Electronic circuit simulation, SPICE, AWE, AC analysis, DC analysis, Transient analysis, Newton-Raphson iteration, Trapezoidal approximation, Matrices, C++ programming language.

ÖZET

ELEKTRONİK DEVRE SİMÜLASYONU İÇİN YAZILIM TASARIMI VE GERÇEKLENMESİ

Mustafa Nazım Yazgan
Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticileri: Doç Dr. M. Ali Tan
Prof. Dr. Abdullah Atalar
1994

Bu tezde, bilgisayar destekli tasarım yardımıyla geliştirilen, "SIMLIB" adını verdiğimiz bir elektronik devre simülasyon paketi anlatılmıştır. Öncelikle devre simülasyonunun temel parçaları olan girdi okunuşu, analiz türleri, metodlar ve matris çozücüler hakkında kısa bilgiler verilmiş, ardından hazırlanan bu pakette kullanılan algoritmalar, özel teknikler açıklanmıştır. Bu paket programa eklemelerin ve yeniliklerin nasıl yapılacağı belirtilmiştir. SIMLIB, devre simülasyonu ile ilgelenen araştımacıların yeni fikirlerini denemeleri ve uygulamaları için oldukça faydalı bir ortam oluşturmuştur.

Anahtar kelimeler : Bilgisayar destekli tasarım, Elektronik devre simülasyonu, SPICE, AWE, AC analiz, DC analiz, Geçici durum analizi, Newton-Raphson iterasyonu, Trapezoid yaklaşım, Matrisler, C++ programlama dili.

ACKNOWLEDGEMENT

I would like to express my deepest gratitude to Dr. M. Ali Tan and Dr. Abdullah Atalar for their supervision, guidance, suggestions and encouragement throughout the development of this thesis.

I would like to thank Ogan Ocalı for his collaboration and invaluable help on the implementation of this work. I would also like to thank Satılmış Topçu and Mustafa Çelik, members of the CAD group at Bilkent University, and all of my friends for their moral support.

I like to acknowledge the financial supports of TÜBİTAK.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

In 1958, the fabrication of integrated circuits (IC's) containing transistors and their interconnections on a single substrate was realized. The advances in IC technology are still continuing from that time on. By time, the capacity of IC's has increased to contain millions of transistors on a very small area of silicon.

The fabrication process of IC's is an expensive procedure. Therefore, the analysis of the designed IC before its implementation plays an important role. If the analysis step is not taken into account, some fatal bugs and errors in the design stage will end up with a costly, but useless chip.

The traditional way of analyzing simple circuits is the hand calculation. However, the complexity of the integrated circuits has reached such a level that the hand calculation of the entire circuit is almost impossible. A detailed verification by hand calculations of even a small part of a complex IC design takes an excessive amount of time. An economic solution to this problem is computer-aided analysis. At this step, the concept of "simulation" of electronic circuits on computers arises.

Circuit simulation provides various advantages. Circuit modifications and corrections can be made before implementation. Eliminating such errors in an early design stage may lead to considerable savings in design cost and time. The second important point is that, extensive simulation capabilities gives the designer a deeper insight into the circuit behavior.

The circuit simulation techniques introduced in 1950s were used to analyze circuits with tens of transistors. Today, even though basically the same

techniques are used, the capacity of the simulators have increased to simulate circuits with tens of thousands of transistors. The advances in desktop workstations and parallel machines have forced researchers to adapt circuit simulators to the latest hardware developments. Nevertheless, accuracy and speed are two opposing factors in circuit simulation. Therefore, the need for novel approaches and new methods continues.

An industry standard for electronic circuit simulation is SPICE [1]. SPICE combines Modified Nodal Analysis (MNA) [2] with Trapezoidal Integration (in its time domain transient mode) and Newton-Raphson (NR) iteration [3], [4]. The combination of algorithms that forms the core of SPICE has been in use, with minor modifications, since mid-1960s. There have been many attempts to improve upon SPICE, but no simulator has managed to replace its industry standard status.

Need For New Circuit Simulators

Everyday, new ideas and new methods are introduced to analyze circuits and systems. Also, the technological developments force simulators to cover new devices and simulate larger circuits. The point is to make simulators work faster, more efficiently, accurately and adaptively.

Some of the new methods used in the graduate studies at Bilkent University can be given as follows:
Asymptotic Waveform Evaluation (AWE) technique was introduced by Lawrence Pillage and Ronald Rohrer from Carnegie Mellon University, [5] to analyze the transient response of RLC circuits, by matching the initial boundary conditions and a number of moments of the actual circuit to that of a reduced order of poles. Some of the simulation studies in Bilkent have been based on this concept.
Piece-wise Linear Asymptotic Waveform Evaluation (PLAWE) is a simulator package that employs AWE method, and uses piece-wise linear models of nonlinear circuit elements. [6] The former implementation was realized by a group of graduate students at Bilkent University. That has been followed by a detailed research on PLAWE by Satılmış Topçu, as a Ph.D. study at the same university [7].
Nonlinear AWE (NOWE) method is proposed by Ogan Ocalı, as a nonlinear version of AWE method, that covers analysis of circuits having nonlinear elements [8].
Generalized AWE (GAWE) is another extension of AWE method which uses multi point moment matching. This is the Ph.D. subject of Mustafa Çelik [9].

The method is especially employed for AC analysis of electronic circuit at high frequencies.

As it is seen that there has been a number of new studies on circuit simulation at Bilkent University, which have to be investigated immediately to see whether they are worth studying or not. This is the point that has become the motivation of this work. Therefore, it is decided to implement a new circuit simulation tool.

The main goal is to create an easy-to-modify and extendible library which can be used to develop new circuit simulators. The program which is implemented in this Master's Thesis, consists a library of circuit analysis, that is called as *SIMLIB*. Furthermore, any type of devices, analyses, methods can easily be added to this library. The style of programming is also a concept that has to be taken into account, so that further work on the program could be carried out easily.

As the result of this study, a complete set of library is implemented. A simulator based on these modules is written. The modules are, input parser, sparse matrix setup and solver, Newton-Raphson iteration, DC circuit solver, DC sweep analysis, AC analysis, transient analysis and some circuit element libraries, which will be explained in the following chapters.

The program is implemented in C++ [10] on Sun [1] Workstations running under UNIX [2] operating system.

C++ is a version of C, a very common programming language. It is designed to
- be a better C
- support data abstraction
- support object-oriented programming (OOP).

The styles of programming typically used in C are "procedural programming" and "modular programming". C++ is a "better C", because C++ provides better support for these styles of programming than C does, while C++ does not lose any generality and efficiency and remains almost completely a superset of C.

It is apparent that OOP will be the programming style of the future because of its advantages over most of the classical languages. Therefore, it has been decided to implement the libraries in C++.

[1] SUN is a trademark of Sun Microsystems.
[2] UNIX is a trademark of AT&T Bell Laboratories.

In Chapter 2, a brief discussion on some basic building blocks of a circuit simulator, that are implemented in SIMLIB is presented to explain what a circuit simulator does conceptually. Moreover, Chapter 3 leads the new programmers to the details of the program. In this chapter, the implementations of input parser, analyses and AWE method are shown in an algorithmic way. In the same chapter, the implemented parts of SIMLIB are listed and adding a new type of analysis and a new element to this library is given with an example. This part of the thesis, together with the appendices is prepared to serve as a manual for SIMLIB.

Chapter 2

BASIC CONCEPTS OF A SIMULATOR

In this chapter, three basic parts of an electronic circuit simulator are covered briefly. In the first section, the interpretation of the circuit declaration (in other words Input Parser) is explained. In the second section, some types of analysis that are applied to electronic circuits are introduced. Some methods employed in these analyses are also mentioned. In the third section, formulation of the circuit in matrix form is discussed.

2.1 Input Parser

Generally, an electronic circuit is described in text format so that it can be viewed and typed by human beings. However, this style of declaration has to be interpreted by the simulation program to carry out the operations faster and easily using its internal database, this job is done by the *parser*. Furthermore, the input parser has to make error detection and correction to prevent from the erroneous results, and warn the user for wrong declarations.

Inputs and Outputs

The input that is accepted by SIMLIB is a SPICE-like circuit definition as a text file. The style of the definitions are explained in Appendix A, SIMLIB Manual. The output is the global variable "circuits", which is an internal structure consisting the circuit definition, devices, models, analyses to be performed,

output formats, etc. (see Appendix B)

Two main functions of the input parser are:

First Pass Counts the number of each device, puts the node names in the node table, puts the voltage source names (or current-controlled elements) in the voltage source table (table structure is explained in Appendix D), checks the number of parameters of the device declarations and reports errors.

Second Pass Converts the node names and voltage source names into index numbers using the tables, checks the parameters, initializes the values of some devices. The index numbers will be used as pointers to the elements of matrix that will be formed during the analysis parts.

The program flow chart of the input parser is shown in Fig. 2.1.

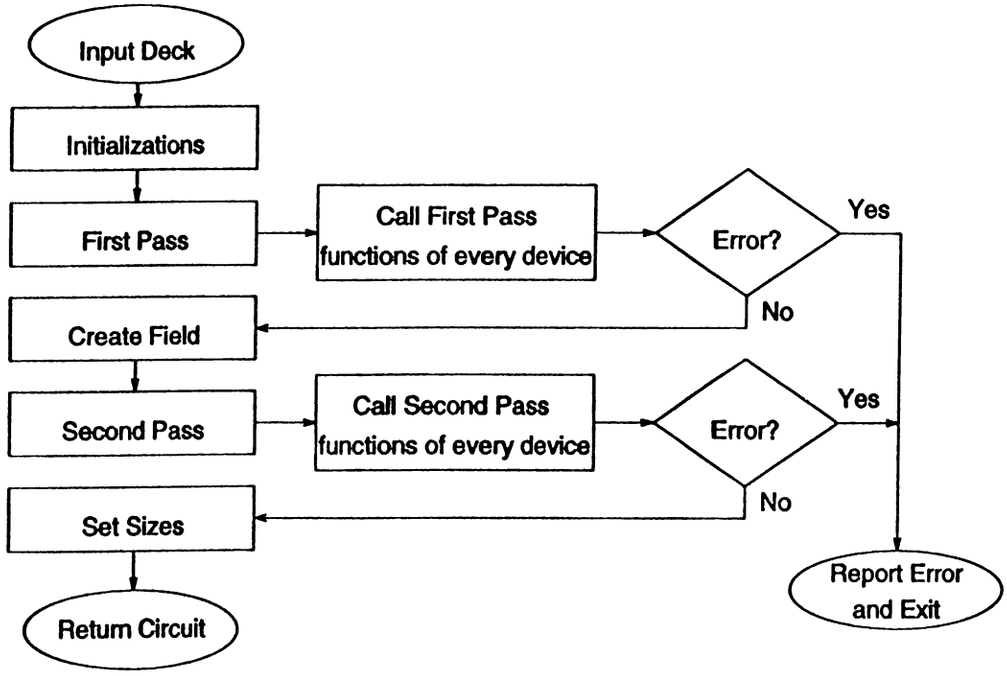


Figure 2.1: Input Parser Flow Chart

Modularity of the input parser

Every device is responsible for checking errors and setting up its field in the memory. Therefore, if addition of a new device or a new type of option card is required, its new first pass, second pass, and field creation functions should be created. Adding a new device and type of analysis are explained in Chapter 3, Section 5 in detail.

Errors handled

Undefined cards and devices are reported. Specific input errors belonging to a device are handled in its corresponding functions, such as, unexpected number of parameters and typing errors of numbers.

2.2 Types of Analyses

The main part of a simulator is the analysis part. The operations that will be performed on the circuit is defined in this part. There are a number of analysis types that are mainly applied to electronic circuits. Also, the new methods have been introduced in some of these analyses. However, since the aim in this work is not to cover all the existing analyses and methods, it is tried to build an extendible library, which can be used to develop any kind of analyses. Nevertheless, most common ones are implemented. They are considered in this section.

The types of analyses are explained in brief, on the following simple example. The manual method is employed for this section.

DC Analysis

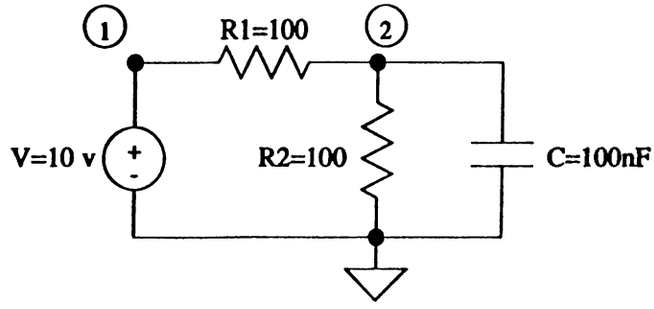


Figure 2.2: A Simple RC Circuit

The circuit in Fig. 2.2 consists only two resistors, a capacitor and a voltage source. Therefore, it is a linear, dynamic circuit. First of all, consider the DC operating point. The capacitor can be discarded in this case and writing down the KCL and KVL equations, the unknowns —current through the voltage source, and the node voltages— are found as follows:

$$i_V - i_{R1} = 0$$

$$i_{R1} - i_{R2} = 0$$

$$V_{R1} = e_1 - e_2 = i_{R1} \cdot R1$$

$$V_{R2} = e_2 = i_{R2} \cdot R2$$

$$\Rightarrow i_V = 50mA,\ e_1 = 10V,\ e_2 = 5V$$

This is the DC operating point solution of the linear circuit.

AC Analysis

AC analysis can also be called as the frequency response of the circuit. This time, the capacitor is replaced by a frequency dependent complex valued resistor. And the value of the voltage source is replaced by 1 (that is, the magnitude). The equivalent circuit is shown in Fig. 2.3. At this step, the same type of operations as in the DC analysis part will be carried out on the given range of frequencies. These operations are complex due to the value of the capacitor–and in general, dynamic elements. Then the AC analysis of the circuit is completed.

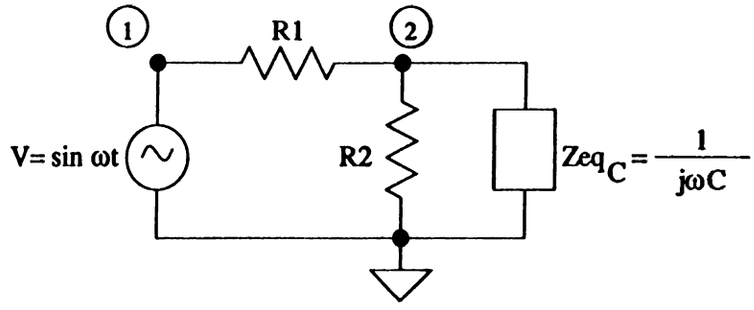


Figure 2.3: AC Equivalent of RC Circuit

Transient Analysis

In this type of analysis, the response of the circuit is examined as the function of time. In this case, the dynamic elements, energy storage elements have to be taken into account. This analysis is generally used to compute the delay of the circuits, and how they respond to different inputs (impulse, step, ramp, pulse).

The transient behavior of the above example is calculated exactly as follows (initial voltage on capacitor is assumed to be zero):

$$V_C(t) = V_V \cdot \frac{R_2}{R_1 + R_2} \cdot (1 - e^{-t/R_2C})$$

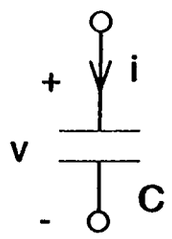
where V_V is the value of the voltage source.

However, the solution is not that easy for larger circuits. The exact solution can not be formulated in most of the cases. Then, some approximations, which are used to linearize the circuit, have to be done. In this work, the trapezoidal approximation is used.

Trapezoidal Approximation

In the transient analysis of the dynamic devices, like capacitors and inductors, integration approximations have to be used to express them as a set of linear equations.

Consider a capacitor as an example:



$$i = C\frac{dv}{dt}$$

Figure 2.4: Capacitor

The capacitor voltage in terms of the integral of the capacitor current is:

$$v(t+\Delta t) = v(t) + \frac{1}{C}\int_{t}^{(t+\Delta t)} i(\tau)d\tau$$

One can consider approximating the integral equation in three possible ways:

$$\int_{t}^{(t+\Delta t)} i(\tau)d\tau \approx \begin{cases} \Delta t \cdot i(t) & \text{Forward Euler (FE)} \\ \Delta t \cdot i(t+\Delta t) & \text{Backward Euler (BE)} \\ \frac{\Delta t}{2}[i(t)+i(t+\Delta t)] & \text{Trapezoidal (TR)} \end{cases}$$

In this work, the trapezoidal approximation method is used to update capacitor voltages and inductor currents. And the equations for the capacitor forms to be:

$$v(t+\Delta t) \approx v(t) + \frac{\Delta t}{2C}i(t) + \frac{\Delta t}{2C}i(t+\Delta t)$$

This equation shows that a capacitor at any instant can be replaced with its Thevènin or Norton equivalent. In SIMLIB, Norton Equivalent is preferred as depicted in Fig. 2.5.

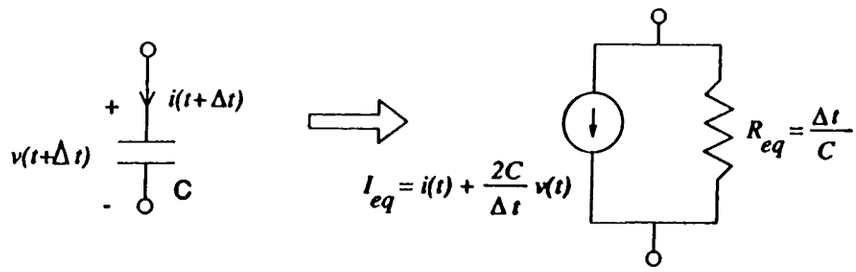


Figure 2.5: Norton equivalent of a capacitor using Trapezoidal Approximation

Note that, the three types of approximations are one step integration approximations. Moreover, higher order integration can also be considered, which employs several preceding time-points to predict the value at the next step better. These complex methods consume excess amount of time, hence they are not used in most of the circuit simulators.

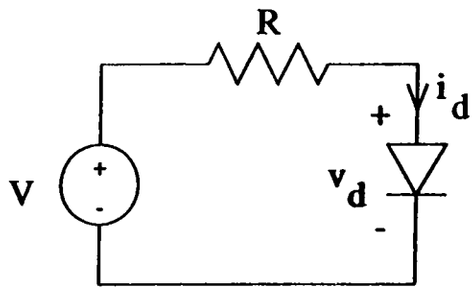


Figure 2.6: Simple Diode Circuit

Nonlinear Case

Consider the diode circuit example as shown in Fig. 2.6, a manual approach to find the current through the diode would be as follows:

$$i_d = \frac{V - v_d}{R}$$

$$i_d = I_s(e^{v_d/V_T} - 1)$$

Graphically, it is seen that the solution can be found by load line method [11]. Computationally, some iterations have to be made to reach the solution. An iteration simply consists of assigning a guess for the solution, initially. Then solving the equations above, the error —the difference between the guess and the result— is obtained. According to the error, the guess is revised. Same steps are repeated until the error is sufficiently small. One common way to realize this type of iteration is the Newton-Raphson algorithm which will be pointed out later in this section.

To handle these operations on the computer, nonlinear devices have to be linearized. At every step of iteration, using the operating point of the nonlinear devices computed at the previous step, their linear models are replaced. For example, the Norton equivalent of the diode is replaced in the previous circuit. It is depicted in Fig. 2.7.

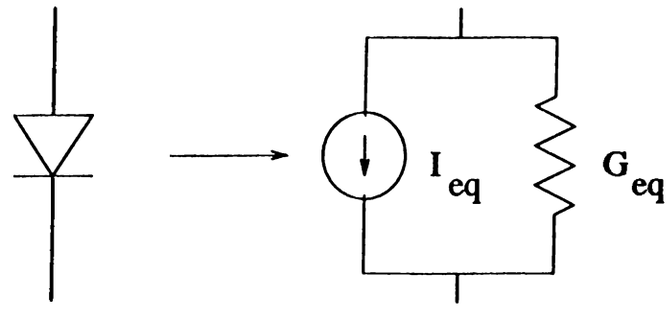


Figure 2.7: Linearization of Diode

$$\text{where} \quad G_{eq} = I_s \cdot exp(\frac{V_{op}}{V_t})/V_t$$

$$\text{and} \quad I_{eq} = I_s[exp(\frac{V_{op}}{V_t}) - 1] - G_{eq} \cdot V_{op}$$

V_{op} is the DC operating point of diode.

As can be viewed from the previous analysis types, every circuit is converted to its equivalent linear form. Therefore, the solution is obtained by performing linear DC analysis. All analyses can be based on the solution of the linear equivalent of the circuit.

Newton-Raphson Algorithm

Newton-Raphson iteration is the most important part of DC and Transient analyses. The Newton-Raphson algorithm seeks to solve the nonlinear equation

$$f(x) = 0$$

iteratively by successive solution of a set of linearized approximations to this equation.

The flow diagram of the Newton-Raphson (NR) algorithm is shown in Fig. 2.8. Every nonlinear device is searched in every step of NR iteration, and linearized according to the solution coming from the last step. As an example, linearization of diode is described in the previous part and shown in Fig. 2.7.

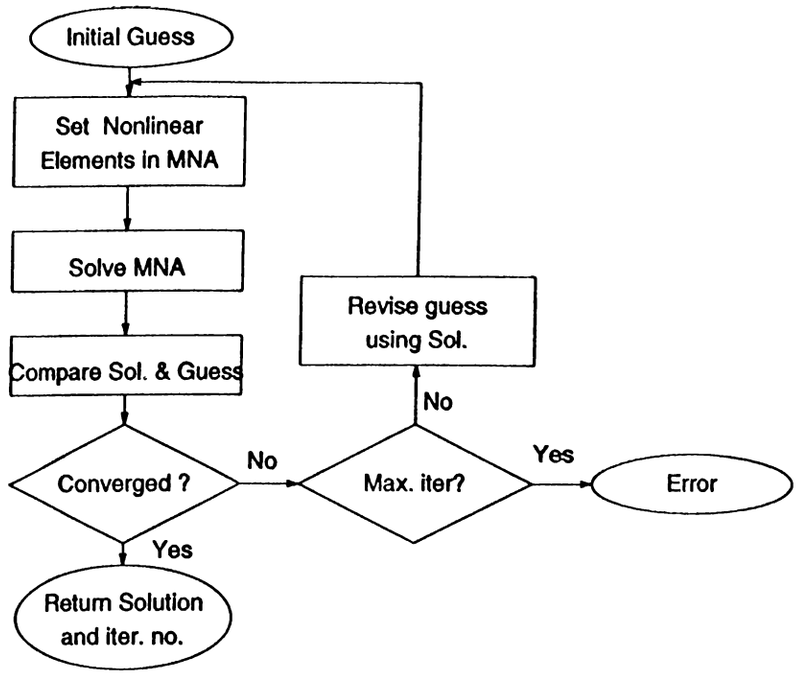


Figure 2.8: Newton-Raphson Algorithm Flow Diagram

Then, the equations are formed according to the new circuit —which is linear—, and solved until two successive solutions are sufficiently close.

Newton-Raphson iteration may not always converge. Accordingly, there has been various modifications on the pure form to help convergence. One way that is tried at Bilkent University is using piece-wise linear (PWL) models [12]. The NR algorithm may fail if it can not visit the right set of linear regions of the elements that the real solution lies. In PLAWE, a possibility of changing region randomly is included, to assure convergence. Eventually, the right set of regions where the solution lies will be found, even if there is a chain of wrong solutions.

A special treatment about diodes is taken into consideration to help convergence in this work. Diode model is modified as depicted in Fig. 2.9. To avoid numerical errors and lots of iterations —which are a result of the exponential behavior—, the function is considered as a line that is tangent to the ideal one at the point where $v_d = 1.0$, for $v_d > 1.0$. It is shown as a dotted line in Fig. 2.9. Such modifications on the models of some nonlinear devices must be done to help Newton-Raphson algorithm to find the solution easier.

The flow diagram of nonlinear version of DC Analysis is shown in Fig. 2.10. The loop is also called Newton-Raphson iteration.

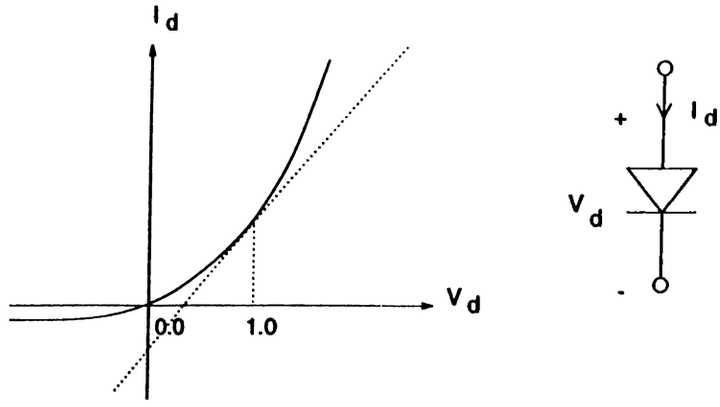


Figure 2.9: Diode Model

Another type of analysis is DC sweep analysis. It is performed to examine the effect of a source. The value of an independent source is swept in a range, then the solution is found as a function of this varying input. This calls DC analysis part with the new value of the input at every step. The flow diagram is shown in Fig. 2.11.

The flow diagrams of AC analysis and nonlinear version of Transient analysis are shown in Figs. 2.12 and 2.13, respectively.

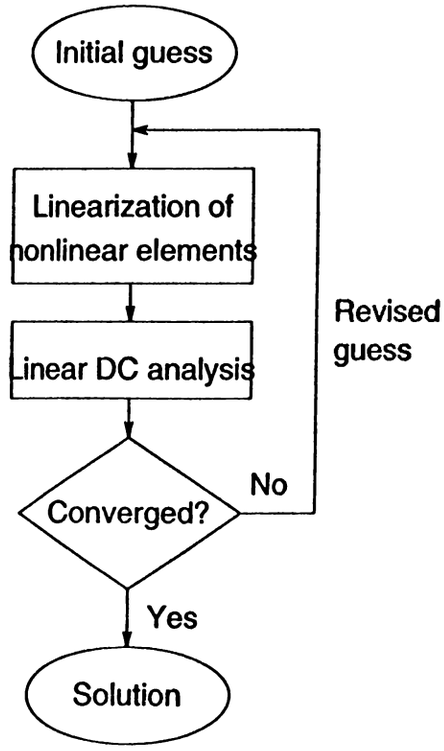


Figure 2.10: Nonlinear DC Analysis

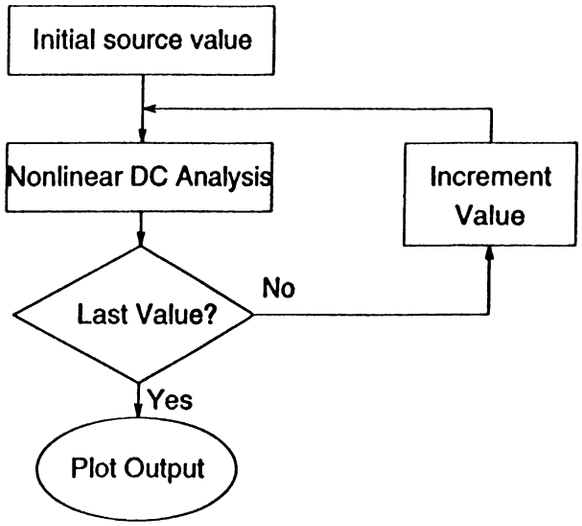


Figure 2.11: DC Sweep Analysis

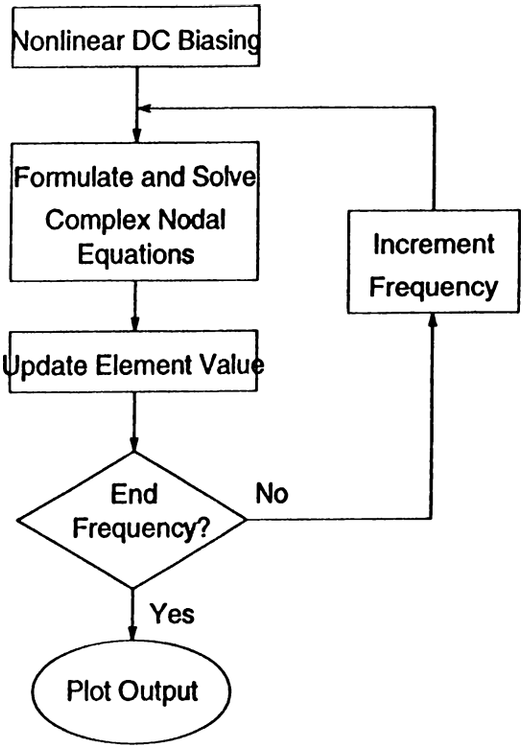


Figure 2.12: AC Analysis

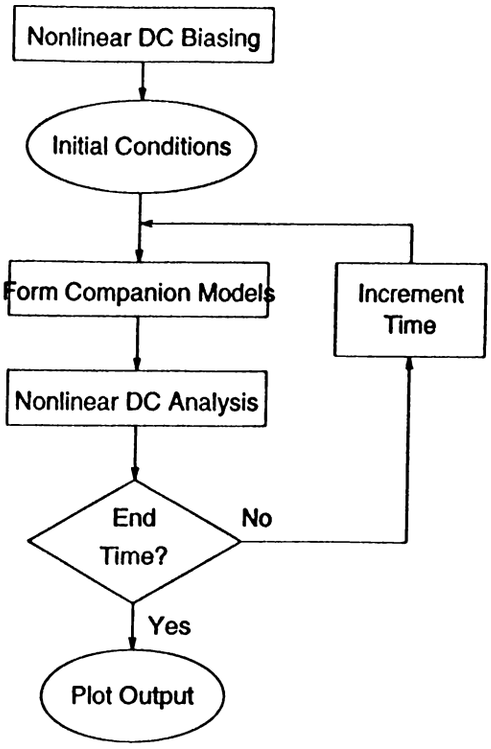


Figure 2.13: Transient Analysis

The method Asymptotic Waveform Evaluation (AWE) can be employed in AC and Transient analyses. This is an approximation of response of linear circuits by a reduced order of polynomial rational. It will be mentioned in Chapter 3, in detail.

2.3 Matrix Setup

As it has already been remarked in the previous section, every type of analysis can be handled by replacing the circuit with its DC equivalent and performing linear DC analysis. That is, the core of the simulator is the solution of set of linear equations. This approach leads us to matrix concept, as linear equations can have a matrix form.

The equations of the sample circuit in Fig. 2.2 can be rewritten in terms of the unknowns, e_1, e_2 and i_V as follows:

$$\begin{aligned} G1 \cdot e_1 \; + \quad & -G1 \cdot e_2 \; + \; -i_V \; = \; 0 \\ -G1 \cdot e_1 \; + \; & (G1 + G2) \cdot e_2 \quad\quad = \; 0 \\ e_1 \quad\quad & \quad\quad = \; V \end{aligned}$$

It is apparent that these equations can be formed as a matrix equation :

$$\begin{bmatrix} G1 & -G1 & -1 \\ -G1 & (G1+G2) & 0 \\ 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ i_V \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ V \end{bmatrix}$$

Solving this matrix equation yields the results.

Now concentrate on how a matrix belonging to a certain analysis is created.

Types of matrix formulation

There are two common formulations. One is Nodal Analysis and the other is *Sparse Tableau Analysis* (STA). [11] There is also Loop Analysis, which is not generally used in circuit simulators. Most common usage of Nodal Analysis is its modified version, which will be referred as Modified Nodal Analysis (MNA). [2]

In PLAWE [6], STA formulation is used. STA is easier to setup, but it produces larger sized matrices and requires more memory space than MNA, which also decreases the efficiency of the solvers, especially for circuits having lots of elements.

Modified Nodal Analysis

Modified Nodal Analysis is used to formulate a circuit as a form of matrix equation. As the name suggests, it is the generalization of Nodal Analysis, which can only cope with voltage controlled elements. MNA was introduced to deal with problems associated with any other type of elements, such as voltage sources and current-controlled circuit elements. It includes KCL and Branch Constitutive Relations. So far, many types of matrix formulations are used in the circuit simulators, but MNA has become the most widely used one. An industry standard, SPICE also uses MNA.

The matrix equation formed by using MNA can be partitioned as:

$$\begin{bmatrix} Y_n & E \\ F & D \end{bmatrix} \begin{bmatrix} e \\ i \end{bmatrix} = \begin{bmatrix} J \\ K \end{bmatrix}$$

where Y_n is the nodal admittance matrix, the contribution of each element is inserted into this matrix, as long as currents can be expressed as explicit functions of voltages.

E is the incidence matrix of the voltage sources and current-controlled elements of the circuit. F and D (can be called as Branch Constitutive Relations) contain the contributions of the circuit elements which are not voltage controlled.

The vectors J and K represent the contributions of the excitation sources, i.e., independent current and voltage sources.

Setting up MNA Matrix

The main advantage of dealing with matrix type of equations is that the contributions of all the individual elements can be inserted directly at appropriate places in the matrix. Therefore, setup functions of every device can be written independently. These setup functions of the elements and devices are called "stencils". In Fig. 2.14, the contribution of a resistor in MNA matrix is depicted. The stencils implemented in this work are given in Appendix C.

In every step, setting up MNA and destroying it after finding the solution takes a lot of time. Keeping the old MNA and adding it only the differences in the values of the devices saves time. Therefore, every device has an initial setup part that forms the MNA at the first step and next-step parts that introduces the changes in their values. For example, the resistors are only visited at the first step, since their values do not change throughout the analysis. However, this method of matrix setup may cause numerical errors if the entries of matrix are near zero or the matrix is ill-conditioned.

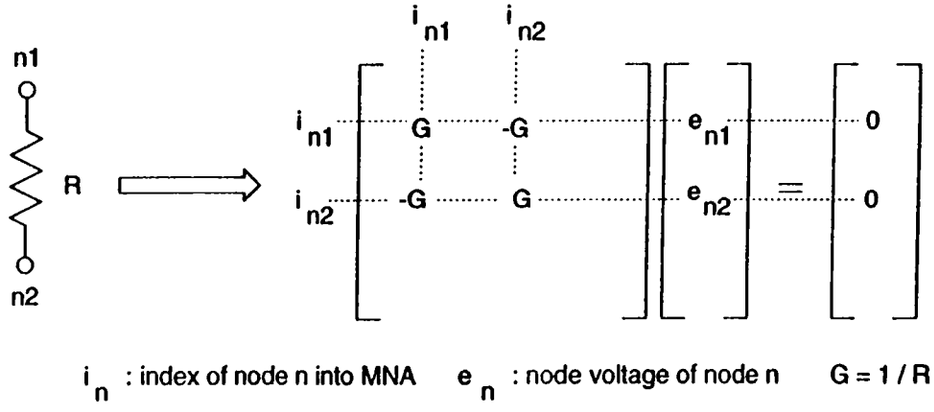


Figure 2.14: Stencil of a Resistor

Since every analysis introduces a different method, different functions are written to handle them. Therefore, an analysis should visit appropriate setup functions to build up the MNA matrix. However, for example, in the DC sweep analysis, since capacitors have no effect on the result, no function is

implemented in DC, for capacitors. Adding a new analysis means writing these functions (stencils) for every device and calling them according to the method employed. Addition of a new analysis type is described in Chapter 3, Section 5.

2.3.1 Matrix Solvers

In SIMLIB, real and complex matrix solvers are used. Using the complex library of C++, converting a real matrix package into a complex package is very simple, therefore doing operations on complex numbers is the same as the real numbers, which is an advantage of C++. First a sparse matrix package and a non-sparse matrix package is written and then their complex versions are created.

Sparse Matrices

The resulting matrix of a Nodal Equation set of an electronic circuit, has in general lots of zero-valued elements, because every node is not connected to every other node. Such matrices are called Sparse Matrices. Moreover, for much larger circuits, the MNA or STA matrices are much more sparse. Even the percentage of the zero-valued elements reaches to 99%. Sparse Matrix Packages take advantage of this property; reduce storage requirements; and reduce the number of floating point operations entailed for LU-factorization and FBS. [3]

Non-sparse Matrix Solvers

In the moment analysis part,(explained in Chapter 3, AWE method) non-sparse matrices are used. Since the matrices are non-sparse, using sparse techniques introduces lots of overheads. Therefore, creating all the elements of the matrix and performing operations on it is a better way. It doesn't have to use indexing methods, managing memory is simpler.

All the matrix packages used for this work are explained in detail in Appendix D.

Chapter 3

ALGORITHMS AND IMPLEMENTATION

In this chapter, the algorithms of the input parser, analyses and AWE technique are listed and explained so as to introduce how the program works. In the fourth section, the implemented parts are mentioned. Finally, modification of the existing library is explained with two examples in the last section.

3.1 Algorithm of Input Parser

1 Initialize the "circuit" [1] structure. That is, open field, set the variables to their initial values.

2 Start with the first line of the input file consisting the circuit definition for performing the following step.

3 Check the first character on the current line. If it is valid, call the "first_pass" function of the corresponding class, else print the error message and exit.

Note : These functions are written according to the style of definition of each device. The node names are inserted into node tables, (table structure is explained in Appendix D) to be able to convert them into index numbers in the 7^{th} step. And errors in the declarations are reported.

[1] The reader may look at the Appendices B and C for the database and matrix setup functions to check the quoted words.

4 Read the next line and repeat step 3 until end of file is reached.

5 Create field in the memory for the devices declared by calling "create_field" functions of them. The size, that is the number of the devices are counted in step 3.

6 Go back to the first line of the input deck.

7 Check the first character on the current line. If it is valid, call the "second_pass" function of the corresponding class, else print the error message and exit. In these functions the parameters are read and inserted into their corresponding fields.

8 Compute the size of circuit and matrix according to the analysis type to be applied. The size is the sum of number of nodes, voltage sources and current-controlled elements.

9 Return the "circuit" structure which contains device classes and the option cards and type of analysis that will be performed.

3.2 Algorithms of Analyses

Assuming that the input parser is called, the data is ready in the "circuits" structure.

DC sweep analysis

1 The source -whose DC value will be swept in a range- is set to the initial value.

2 The solution array is initialized to zero. The solution array is the vector of unknowns of the MNA function.

3 Call "set_MNA" functions[2] (stencils) of each device, except capacitors and nonlinear devices.

4 Call Newton-Raphson iteration function, which is outlined as follows:

4.1 Add the linearized models of the nonlinear devices into MNA by calling "set_MNA" functions of them. (These functions require operating point, which is extracted from the solution array.)

[2] All of the MNA set up functions are given in Appendix C.

4.2 Solve the resulting matrix equation.

4.3 Compare the old solution array with the new result. If the difference is not sufficiently small, then assign solution array the new result and goto step 4.1.

5 Print out the desired entries of the solution array. This is defined in the .print card of the input deck (See Appendix A), and the "circuits" structure consists this information.

6 Increment the source value. If last value is not reached, then goto step 4.

7 Print out the statistics. (Solution time, number of iterations, etc.)

AC Analysis

In AC analysis, the Matrices are complex, so the functions for this analysis uses complex sparse matrix package.

1 Set frequency to the start value.

2 Call "set_CMNA_incond" functions of each device. The complex MNA matrix will be formed.

3 Solve the complex matrix equation.

4 Print out the desired entries of the solution array, in the given format (magnitude, phase, real or complex part).

5 Increment frequency. If it is the last value, then stop.

6 Call "set_CMNA" functions of each device. Goto step 3.

7 Print out the statistics.

Transient Analysis

1 Specify whether the circuit contains nonlinear elements or not.

2 Call "set_MNA_incond" functions of each device.

3 If the circuit is linear, solve MNA equations, else call Newton-Raphson to get the solution. This yields the initial condition.

4 Print out the desired entries of the solution array.

5 Destroy MNA, and call "set_MNA" functions of each device, except dynamic elements. MNA is created again since the size is changed because of the initial conditions of capacitors and inductors has introduced a larger circuit at the second step. This time the circuit is smaller.

6 Increment time.

7 Call "set_MNA" functions of dynamic elements. Norton equivalents of capacitors and inductors are inserted according to the trapezoidal approximation.

8 If the circuit is linear, solve MNA equations, else call Newton-Raphson to get the solution.

9 Print out the desired entries of the solution array.

10 If the end of analysis time isn't reached, then goto step 6.

11 Print out the statistics.

Some example circuits and print-outs are given in Appendix F.

3.3 Algorithm of AWE Method

AWE (Asymptotic Waveform Evaluation) technique is a new approach which is used in timing analysis of RLC interconnect trees. It approximates time response of a state variable by less number of exponentials, and frequency response by a lower order polynomial rational. This method finds moments of the actual circuit (integral or derivative) and matches them to the polynomial. New revised versions of AWE are used to find nonlinear circuit solutions, and it is adapted to frequency analysis.

For the theoretical details of AWE see the references [5], [13], [14], [15],[16], [17],[18],[19],[8]. Now, concentrate on the computational algorithm to simulate a circuit using AWE :

1 2q-1 moments are needed. These could be integral moments, derivative moments or a combination of them.

The moments are found as follows:

Integral moments : First, find the DC operating point of the circuit (open circuit capacitors, short circuit inductors). Call the capacitor voltages and inductor currents $m_{-1}^{\mathbf{st}}$ moments. (A superscript C is used for capacitor moments and L is used for inductor moments.) Then

1. Set independent sources to zero.

2. Replace capacitors with current sources of value $C \cdot m_l^C$

3. Replace inductors with voltage sources of value $L \cdot m_l^L$

4. Compute the resulting circuit. The voltages across capacitors are m_{l+1}^C, and the currents through the inductors are m_{l+1}^L.

5. Return to step **2.**, and find the next moment until l reaches the last index of integral moments to be calculated.

Derivative moments : This time replace capacitors and inductors with voltage and current sources respectively. The values are set to their initial conditions. Call the solution (capacitor currents and inductor voltages) $m_{-2}^{\mathbf{nd}}$ moment.

1. Set independent sources to zero.

2. Replace capacitors with voltage sources of value m_k^C/C

3. Replace inductors with current sources of value m_k^L/L

4. Compute the resulting circuit. The currents through capacitors are m_{k-1}^C, and the voltages across the inductors are m_{k-1}^L.

5. Return to step **2.**, and find the next moment until k reaches the last index of derivative moments to be calculated.

Note that the index of the first integral moment is 0 and the others follows as 1, 2, 3, ..., the index of the first derivative moment is -2 the others follows as -3, -4, -5, ... m_{-1} is the DC solution of the circuit. So it is neither called integral nor derivative moment.

2 Using these moments form the matrix equation as, (supposing there are n integral and m derivative moments, $(m+n+1)$ should be even)

$$\begin{bmatrix} -m_{-n-1} & -m_{-n-2} & -m_{-n-3} & \cdots & (*)m_{(q-1)} \\ -m_{-n-2} & -m_{-n-3} & -m_{-n-4} & \cdots & (*)m_q \\ -m_{-n-3} & -m_{-n-4} & -m_{-n-5} & \cdots & (*)m_{(q+1)} \\ \vdots & \vdots & \vdots & & \vdots \\ (*)m_{(q-1)} & (*)m_q & (*)m_{(q+1)} & \cdots & m_{m-1} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{(q-1)} \end{bmatrix} = \begin{bmatrix} (*)m_q \\ (*)m_{(q+1)} \\ (*)m_{(q+2)} \\ \vdots \\ m_{m-1} \end{bmatrix}$$

$$\text{where } q = \frac{m+n+1}{2} \text{ and } (*) = \begin{cases} -1 & if\ index < -1 \\ 1 & otherwise \end{cases}$$

The solution is the coefficients of polynomial,

$$C(\lambda) = a_0 + a_1\lambda + a_2\lambda^2 + \cdots + a_{q-1}\lambda^{q-1} + \lambda^q$$

3 The reciprocal of the roots of this polynomial gives us the approximate poles of the state variable x, where x can be represented as,

$$x(t) = \sum_{j=1}^{q} k_j e^{p_j t}$$

p_j are the poles, and k_j are the corresponding residues.

4 Obtaining the Vandermonde matrix we can find the residues of the poles,

$$\nu \mathrm{k} = \mathrm{m_l}$$

$$\nu = \begin{bmatrix} \lambda_1^{-n} & \lambda_2^{-n} & \cdots & \lambda_q^{-n} \\ \lambda_1^{-n+1} & \lambda_2^{-n+1} & \cdots & \lambda_q^{-n+1} \\ \vdots & \vdots & & \vdots \\ \lambda_1^{q-1} & \lambda_2^{q-1} & \cdots & \lambda_q^{q-1} \end{bmatrix} \text{ and } \mathbf{m_l} = \begin{bmatrix} -m_{-n-1} \\ -m_{-n-2} \\ \vdots \\ (*)m_{q-1} \end{bmatrix}$$

where λ_i 's are the roots of $C(\lambda)$, and m_l is formed by the first lower-order q moments.

5 The solution of the matrix equation in the fourth step yields the residues. Then the approximation for $x(t)$ is found.

For this method, a complex polynomial root finder and a moment class is implemented. The flow diagrams are given in Figs. 3.1 and 3.2.

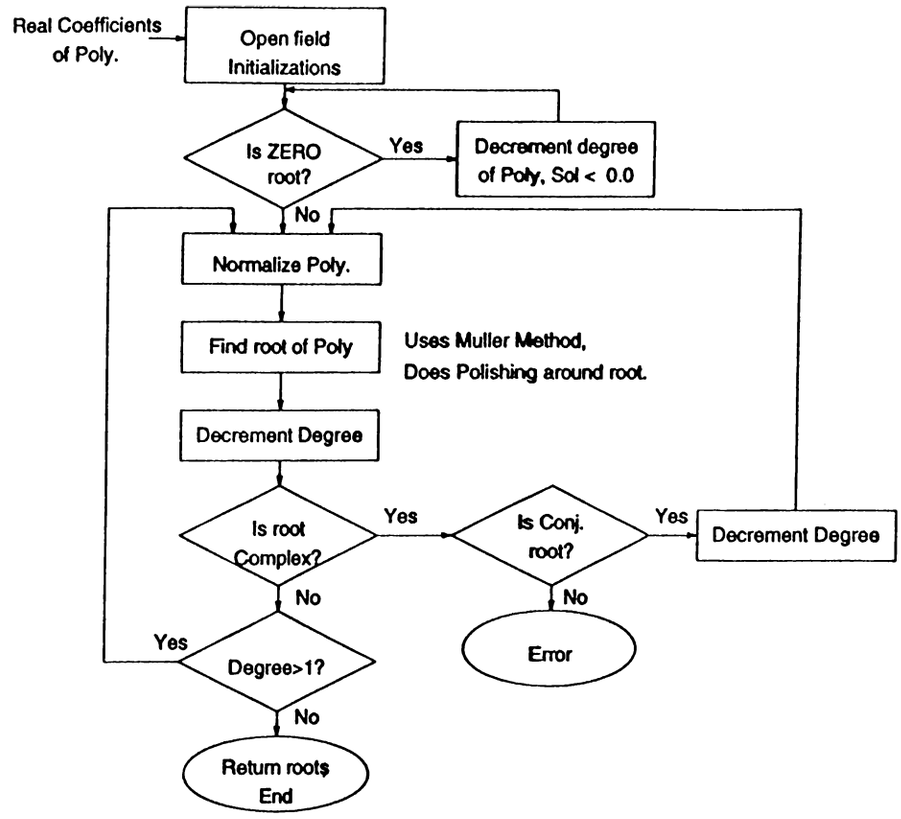


Figure 3.1: Diagram of the Polynomial Root Finder

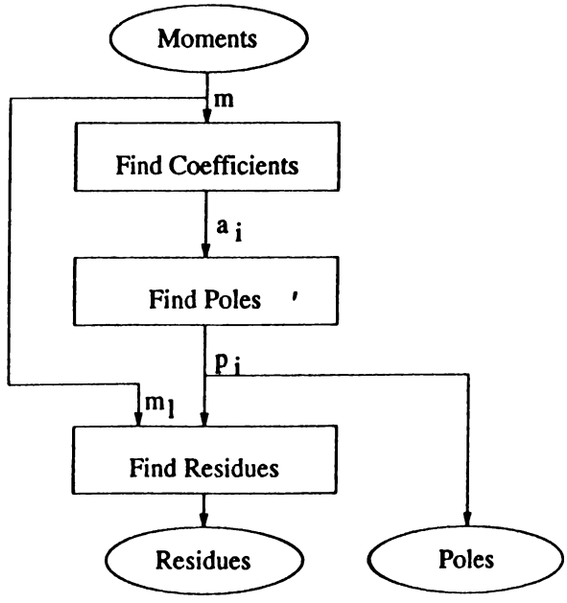


Figure 3.2: Flow diagram of Moment Matching

3.4 Implementation

SIMLIB is a collection of library functions for electronic circuit simulation. As it can be viewed from the chart in Fig. 3.3, some of the circuit elements and their analysis functions are implemented. One can fill the blanks or add a new row, which means adding a new device, or add a new column, which means adding a new analysis type. This is explained step by step in the next section.

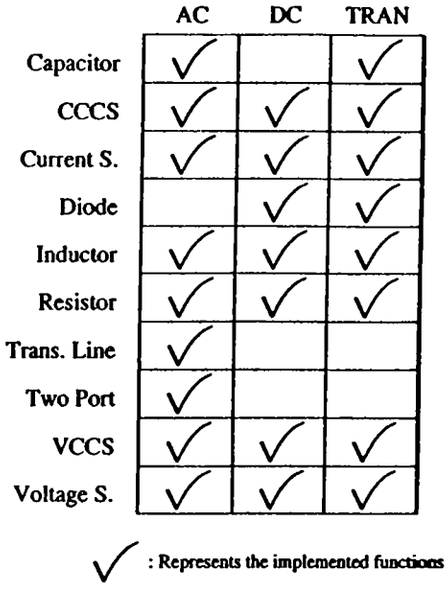


Figure 3.3: Analysis vs. Elements

Besides, some routines that control the flow of the program are implemented. One of these routines checks the "jobs" structure, which is embedded in "circuits" structure [3] and calls the corresponding functions. That is AC, DC or transient analyses. Then in these analyses the stencils of the devices are called.

Also matrix packages which is used to form a matrix equation and to solve this are implemented. The table package which is used for indexing in the input parser part is written. Both packages are implemented by Ogan Ocalı. The usage of them are given in Appendix D.

The remaining functions and their usage are given in Appendix E. And some of the source listings are added.

[3] Details of the database is given in Appendix B.

3.5 Modification

One of the requirements of this work is to create an easy-to-modify library. This property will be described with an example in this part. Examining Appendix B and E, directory structure and variable declarations will be helpful before going into details of the program.

3.5.1 Adding a New Device

In this part, a "capacitor" as an example to a new device is described.

1. Write "capacitor" class.

   (a) Write the include file of the class, which declares the variables and the functions related to the element. Add it to the file "simlibhome/include/twoterm.h".

```
class cap {

  int *n1,*n2,              // node indices
      count;                // used as internal pointer
  double *val,              // value of the capacitor
         *ic,               // initial condition
         *Ieqp,*Geqp;       // equivalent I,G at the previous step

 public:

  int capno;                // total number of capacitors in the class

  cap() ;                   // construction
  ~cap() ;                  // destruction
  void first_pass(incirc *curcirc);   //  Input
  void create_field();                //  Parser
  void second_pass(incirc *curcirc);  //  Functions


  void set_MNA(double *sol,double dT);  //  Transient an. stencil func.
  void set_CMNA(double freq);           //  AC analysis stencil func.

  void list_all();
};
```

   (b) Write the class functions in a file "simlibhome/source/devdir/capacitor.C".

```
#include "includes.h"
/******
     capacitor class definitions
*******/
```

```
  cap::cap()    // initial declaration
    {
      capno=0;
      count=0;
    }

  cap::~cap()    // destruction function
    {
//  cout << "Cap. destroyed!";
    }

  void cap::first_pass(incirc *curcirc)
/****
usage : Cxxxxxx node1 node2 value <ic=number>
no. of parameters of capacitor should be at least 4.
global variables;
word_num : no of words on a line,
word_list[i] : the i'th word on the line.
node1 and node2 are added to the ''nodetable''.
capno is incremented at every call.
****/
    {
      if (word_num<4) {
        sprintf(mess,"Error in capacitor %s !\n",word_list[0]);
        errormess();
      } else {
        curcirc->nodetable.putsym(word_list[1]);
        curcirc->nodetable.putsym(word_list[2]);
        capno++;
      }
    }

  void cap::create_field()
    {
       n1 = new int[capno];
       n2 = new int[capno];
       val = new double[capno];
       ic = new double[capno];
       Ieqp = new double[capno];
       Geqp = new double[capno];
    }

  void cap::second_pass(incirc *curcirc)
    {
      int i=count;
//  using the sym table ''nodetable'' insert values n1,n2.
//  n1 and n2 are index number which will be used to point to an MNA entry.
      n1[i] = get_index(curcirc,word_list[1]);
      n2[i] = get_index(curcirc,word_list[2]);
// read_num returns 0 if there's an error in typing of the value,
// or converts the string into a double.

      if(!read_num(word_list[3],val[i])) {
        cout<<"Error in " <<word_list[0]<<" value!\n";
```

```
        exit(1);
      }
 // if there's an error in i.c. value it is taken as 0.
      ic[i]=0.0;
      if (!strcmp(word_list[4],"ic") && (word_list[5][0]=='='))
        if(!read_num(word_list[6],ic[i]))
          ic[i]=0.0;

      Geqp[i] = Ieqp[i] = 0.0;
      count++;
    }

/**** This function is used for transient analysis *****/
 void cap::set_MNA(double *sol,double dT)
    {
      double Vop,Geq,Ieq,Iadd,Gadd;

      for (int i=0 ; i<capno; i++)
        {
          Geq = (2.0*val[i])/dT;

          if ((Geqp[i] == 0.0) && (Ieqp[i] ==0.0)) { // At the first step
            Iadd = Ieq =  Ieqp[i] =  Geq * ic[i] ;   // these will be
            Gadd = Geqp[i] = Geq ;                   // evaluated according to the ic value
          } else {
            if (n1[i]>-1)                            // For the next steps
              Vop = sol[n1[i]];                      // Ieq and Geq will be
            if (n2[i]>-1)                            // calculated using Vop
              Vop -= sol[n2[i]];                     // found at previous step
            Iadd =  Vop*(Geq+Geqp[i])-2*Ieqp[i];
            Ieqp[i] += Iadd ;
            Gadd = Geq - Geqp[i];
            Geqp[i] = Geq;
          }
          if (Gadd!=0.0) {                           // Stencils, see appendix C
            if (n1[i]>-1)
              ins_M(M,n1[i],n1[i],Gadd);
            if (n2[i]>-1)
              ins_M(M,n2[i],n2[i],Gadd);
            if ((n1[i]>-1) && (n2[i]>-1)) {
              ins_M(M,n1[i],n2[i],-Gadd);
              ins_M(M,n2[i],n1[i],-Gadd);
            }
          }
         if (n1[i]>-1)
            ins_b(b,n1[i],Iadd);
          if (n2[i]>-1)
            ins_b(b,n2[i],-Iadd);
        }
    }

/****** AC analysis part ********/
 void cap::set_CMNA(double f)
{
  complex Gadd;
```

```
    for (int i=0; i<capno; i++) {

      Gadd = complex(0,2*PI*f*val[i]);      // AC equivalent impediance of cap.


      ins_M(CM,n1[i],n1[i],Gadd);           // Stencil of cap.
      ins_M(CM,n2[i],n2[i],Gadd);

      ins_M(CM,n1[i],n2[i],-Gadd);
      ins_M(CM,n2[i],n1[i],-Gadd);

    }
  }

  /******* listing of all capacitors in the class *******/
    void cap::list_all()
      {
        for (int i=0; i<capno; i++)
          cout<<"cap["<<i<<"] = "<<n1[i]<<","<<n2[i]<<",val="<<val[i]<<"\n";
      }
```

So the capacitor class implementation is completed.

2. Add the capacitor class to the "circuit" structure, which is explained in Appendix B. It is declared in "simlibhome/include/inp.h".

```
    .
    .
    .
typedef struct inckt {

  char *subname;

  res *resistor;
  vs *volt;
  .
  .
  .
  cap *capacitor;  // should be added.
  .
  .
  .

} incirc;
```

3. The input parser functions should be called from the input parser module. It is the file "simlibhome/source/input/device.C".

```
  .
void first_pass(incirc *curcirc)
{


  .
  curcirc->capacitor = new cap;   // should be added.

  .
  .
    while (parse_file(infile)) {
    lineno++;

    switch (word_list[0][0]) {

    case 'c' :                                  //
    case 'C' :                                  //
      curcirc->capacitor->first_pass(curcirc);  // should be added.
      break;                                    //
  .
  .
  .
}

 void create_fields()
{


  .
  temp->capacitor->create_field();   //  should be added.
}

void second_pass(incirc *curcirc)
{



    while (parse_file(infile)) {
    lineno++;

    switch (word_list[0][0]) {

    case 'c' :                                //  should be added.
    case 'C' :
      curcirc->capacitor->second_pass(curcirc);
      break;
```

At this step capacitor can be accepted as an element in the input deck.

4. The transient analysis function is "simlibhome/source/analyses/tran.C". This module calls "set_MNA_dyn" function, which calls the transient analysis stencil functions of all the dynamic elements. "set_MNA_dyn" is in the file "simlibhome/source/methods/mna.C". "set_MNA" function of the class capacitor should be added in here.

```
void set_MNA_dyn(double *sol)
{
  .
  .
  .
  circ->parent->capacitor->set_MNA(sol,circ->job.step);  // should be added.

}
```

5. The AC analysis function is "simlibhome/source/analyses/ac.C". This module calls "set_CMNA_incond" and "set_CMNA" functions, which are the complex version of set_MNA. These functions are in the file "simlibhome/source/methods/mna.C".

```
void set_CMNA_incond()
{
  .
  .
  .
  circ->parent->capacitor->set_CMNA(circ->job.start); // should be added.
// initial value
}
void set_CMNA(double f)
{


  .
  circ->parent->capacitor->set_CMNA(circ->job.step);  // should be added.
// next step additive value
}
```

After the completion of the steps above, capacitor is added to the simulator for transient and AC analyses. Note that it is easier to copy one of the existing device classes and make the changes on this file.

3.5.2 Adding a New Analysis

DC analysis will be explained as an example for a new analysis.

1. To parse the analysis declaration in the input deck a "read_dc" function is added to the class "jobs". Include file is "simlibhome/include/jobs.h".

```
class jobs {
 public:
  .

  .
 void read_dc();
};
```

And write the function in file "simlibhome/source/input/jobs.C".

```
void jobs::read_dc()
{
/***
usage: .dc vs_name start_value stop_value step_value
***/
  if (word_num<5) {
    strcpy(mess,"Error in .dc card! (check number of words)\n");
    errormess();
  }

  dcok=1;  // dcok flag is set TRUE,
           // means DC analysis will be performed
  strcpy(vsname,word_list[1]);
  if (!(read_num(word_list[2],start) &&
        read_num(word_list[3],stop) &&
        read_num(word_list[4],step))) {
    strcpy(mess,"Error in .dc card! (check numbers)\n");
    errormess();
  }
}
```

Note that vsname, start, stop and step are public variables of jobs. See Appendix E for details.

2. Reading the analysis card should be added into the "first_pass" function of the input parser .

```
void first_pass(incirc *curcirc)
{
  .

  .
case '.' :

    if (!strcmp(word_list[0],".dc")) {
        circ->job.read_dc();
```

```
        break;
      }


}
```

3. The MNA setup functions (stencils) should be added to the device classes that are supported in the new analysis. See Appendix C for information about stencils.
   For example, "set_MNA" function for a resistor will be written as :

```
void res::set_MNA()
  {
    if (resno<1) return;

    int ni,nj;
    double ival;

    for (int i=0; i<resno; i++) {
      ni = n1[i];        // node indices
      nj = n2[i];

      ival = 1.0/val[i];     //  G = 1/R

      ins_M(M,ni,ni,ival);   // stencil for resistor
      ins_M(M,nj,nj,ival);
      ins_M(M,ni,nj,-ival);
      ins_M(M,nj,ni,-ival);

    }
  }
```

4. Write the "set_MNA" function in "simlibhome/methods/mna.C" which calls the stencil functions of the devices.

```
void set_MNA()
{
  int size;

  size = circ->totalsize;  //get the size of the MNA matrix
// This size is set in the input parser

  M = spmcreate(size,size);
  b = new double[size];
  for (int i=0;i<size;i++) b[i] = 0.0;

  circ->parent->resistor->set_MNA();
  circ->parent->volt->set_MNA();
```

```
    circ->parent->VCCS->set_MNA();
    .


}
```

5. Then write the analysis function which calls these MNA setup functions. Analysis function of DC part is "simlibhome/source/analyses/dc.C"

```
#include "includes.h"
#include "mna.h"


void do_dcsweep()
{
  double *sol,*sol1,*sold,*soli;
  int vsindex,iterno=1,totaliter=0,itershow=circ->job.showiter;
  double tim,vval;

// Get the index of the voltage source whose
// value will be swept

  if ((vsindex = circ->parent->vstable(circ->job.vsname))<0) {
    cerr<<"Error in .dc card vs is not defined!\n";
    exit(1);
  }

  sol = new double[circ->totalsize];
  soli = new double[circ->totalsize];
  sold = new double[circ->totalsize];
  for(int j=0;j<circ->totalsize;j++) sol[j]=sold[j]=soli[j]=0.0;

// place of vs in MNA is no. of nodes + vsindex
  vsindex +=  circ->parent->nodetable.nofsym() ;
  circ->job.outnames[strlen(circ->job.outnames)] = '\0';
  cout<<" Voltage("<<circ->job.vsname<<")  "<<circ->job.outnames;

  tim = clock();

  set_MNA();
  j=0;
  for (vval = circ->job.start; vval <= circ->job.stop;vval += circ->job.step)
    {
      j++;
      b[vsindex] = vval ;
      for(j=0;j<circ->totalsize;j++) soli[j]=sol[j];
      sol1 = NewtonRaphson(sol,iterno);
      sol = new double[circ->totalsize];
      for(j=0;j<circ->totalsize;j++) {
sol[j]=sol1[j];
      }
      totaliter += iterno;  // counts the iter no.
      output(vval,sol);
```

```
      if (itershow)
cout<<iterno;
      cout<<"\n";
    }
  cout<<"\n total iteration = "<<totaliter;
  tim = clock() - tim;
  cout<<"\ndc sweep time="<<tim/1000000<<" sec.\n";

}
```

"NewtonRaphson" function is implemented in file "simlibhome/methods/nr.C". This funtion calls "set_MNA_nonlin" function, which calls the MNA setup functions of nonlinear elements. Of course, such special functions should be designed and added by the programmer according to the methods used.

Chapter 4

CONCLUSION

In this thesis, some library functions to develop electronic circuit simulators are implemented. The collection of these functions is called as SIMLIB. The main property of SIMLIB is that modifications such as addition of new analysis types and methods can be applied easily. Besides, although this implementetion covers only some circuit elements, any device can be added to the simulation library with a little effort. To describe such additions and modifications, an example is given.

SIMLIB consists of an input parser, sparse and non-sparse matrix packages, a symbol table package, AC, DC, transient analyses, moment matching by using the AWE technique and some device classes including most commonly used ones, linear elements, capacitor, inductor, diode, transmission line, etc.

The programs are written in C++, which is an object-oriented programming language. The object-oriented feature has provided to write the program in a modular way. This modularity enables other programmers to get familiar with the program easily.

Since the aim was to build a development tool rather than to write a very fast or exteremely accurate simulator, simple models of the existing devices have been used, and the implementation has been restricted to a limited set of devices. As a result, the library can be easily modified and extended to include other devices.

The future work should concentrate on constructing new simulators in which some novel methods are examined. Applying new methods and extending the types of analyses, including new devices to SIMLIB, modifying the modules to make the program work faster and more accurately are some of the issues left to be investigated.

Appendix A

SIMLIB MANUAL

It is almost the same as SPICE input deck. There are some differences which the users should be careful about. All the nodes can be either numeric or alphanumeric. There's no value checks for the time being.

A.1 Elements

In the following, the formats of declaration of elements are listed.

1. Resistors

   Rxxxxxx [node1] [node2] [value]

2. Voltage Sources

   Vxxxxxx [node+] [node-] [value]

3. Current Sources

   Ixxxxxx [node+] [node-] [value]

4. Capacitors

   Cxxxxxx [node+] [node-] [value]

5. Inductors

   Lxxxxxx [node+] [node-] [value]

6. Voltage Controlled Current Sources

   Gxxxxxx [node+] [node-] [controlling_node+] [controlling_node+] [value]

7. Voltage Controlled Voltage Sources

   Exxxxxx [node+] [node-] [controlling_node+] [controlling_node+] [value]

8. Current Controlled Current Sources

   Fxxxxxx [node+] [node-] [V_name] [value]

9. Current Controlled Voltage Sources

   Hxxxxxx [node+] [node-] [V_name] [value]

10. Diodes

    Dxxxxxx [node+] [node-] <is=[is_value]> <vt=[vt_value]>

11. Transmission lines

    Txxxxxx [node1] [node2] [node3] [node4] Z0=[value] F=[value] NL=[value]

    or

    Txxxxxx [node1] [node2] [node3] [node4] C=[value] D=[value] L=[value]

12. Two-ports

    Yxxxxxx [node1] [node2] [node3] [node4] (y11) (y12) (y21) (y22)

    Where yxx are the elements of admittance matrix ;

$$Y = \begin{pmatrix} y11 & y12 \\ y21 & y22 \end{pmatrix}$$

    In two port representation (yxx) will be written as :

$$((p_0\ p_1\ p_2 \ldots p_n)(q_0\ q_1\ q_2 \ldots q_m))$$

    where

$$yxx = \frac{(p_0 + p_1 \cdot w + p_2 \cdot w^2 + \ldots + p_n \cdot w^n)}{(q_0 + q_1 \cdot w + q_2 \cdot w^2 + \ldots + q_n \cdot w^m)}$$

    (Note that a '+' sign at the beginning will add the current line to the previous line.)

A.2 Analyses

1. DC analysis

   Usage : .dc [V_name] [start_value] [stop_value] [step_value]

2. Transient analysis

   Usage : .tran [step_time] [stop_time]

3. AC analysis

   Usage : .ac [start_freq] [stop_freq] [step_freq]

A.3 Output formatting

1. Printing :

   Usage : .print v(node) i(V_name)

   (if only v or i is written magnitudes of the values are printed.
   other options :
   vm,im : magnitude ;
   vp,ip : phase ;
   vr,ir : real part ;
   vi,ii : imagimary part .)

2. Options :

   Usage : .options <option_list>

   (list : lists the circuit (to see how the input deck is parsed).
   printmb : prints the MNA matrix and the source vector in every step for tracing what happens during the analyses (not recommended for large circuits.
   iterno : Prints the number of iterations at every calculation point.)

Appendix B

DATABASE

B.1 Global Declarations

The most important global variables are the circuit structure and the matrix structure. See Appendix E, Section 2 for a detailed list.

B.1.1 Circuit

The circuit structure contains [1]:

- Device classes (includes every data and function related to)
- Lookup Tables (which are defined as classes) for indexing
- Models for the devices
- Jobs to be performed, style of the output
- Subcircuits

B.1.2 Devices

Devices are implemented as classes having required variables and functions. A device typically contains :

[1]The circuit structure declaration is in file "simlibhome/include/inp.h".

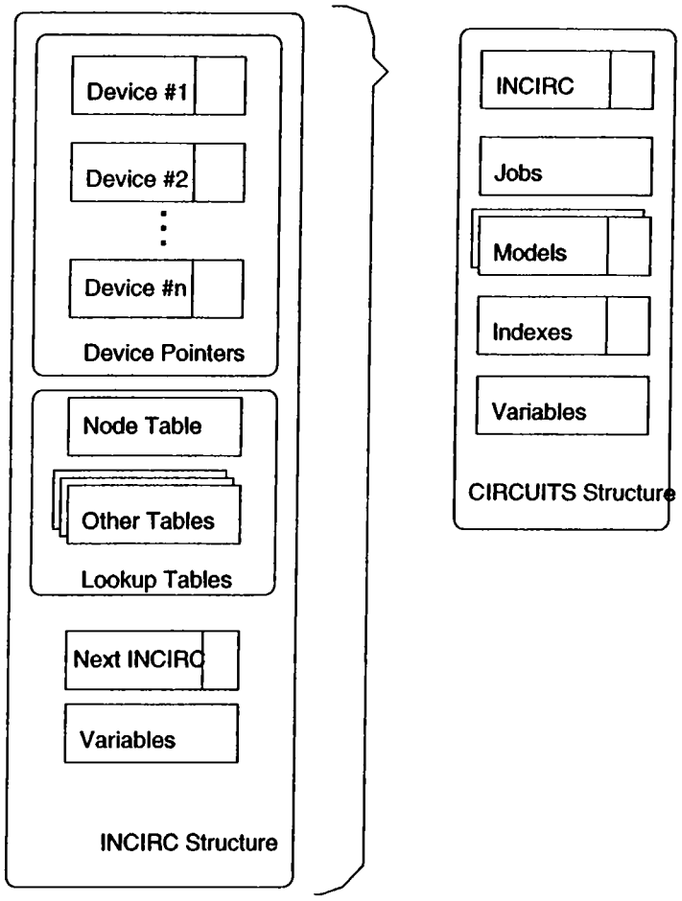


Figure B.1: CIRCUITS Structure

- Its occurance number in the subcircuit
- Node numbers used as the indexes into the MNA matrix
- Device values (e.g. resistence value for a resistor)
- Construction and destruction functions
- Functions for input parsing and circuit creation
- Stencils for MNA formulation (for each type of analysis)
- A listing function

Examples for two-terminal devices:

- Capacitor
- Diode
- VCCS
- CCCS
- Inductor
- Voltage Source
- Current Source
- Resistor

Examples for four-terminal devices :

- Transmission Line
- Two port

B.2 Jobs

Jobs is the class responsible for reading the cards of an input deck. It sets the variables indicating whether a DC, AC or Transient analysis will be performed. It reads the model cards and also the option cards. It contains the format of the output style.

B.3 Subcircuit

This is held by the recursive structure of the "circuit". In every **circ** variable there's a field of pointer pointing to another **circ** structure.

B.4 Matrix

There are four Matrix Packages used in this program:

- Real Sparse Matrix Package
- Complex Sparse Matrix Package
- Real Non-Sparse Matrix Package
- Complex Non-Sparse Matrix Package

All of them are implemented as classes in C++. In Appendix D the usage and the properties are explained.

Appendix C

STENCILS

In this part, the stencil functions of the elements are listed.

Representation:

The following matrix equation is tried to be constructed.

$$\mathbf{M} \cdot \mathbf{x} = \mathbf{b}$$

where $\mathbf{M}$ is an nxn matrix and $\mathbf{x}$ and $\mathbf{b}$ are vectors of length n.

n_i is the index of node i of the element for the MNA matrix.
$M[n_i, n_j] += value$ means adding $value$ into the entry of matrix $\mathbf{M}$.
$b[n_i] += value$ means adding $value$ into the n_i, n_j entry of matrix $\mathbf{M}$.

1. Resistor of value R

$$M[n_1, n_1] += G$$

$$M[n_1, n_2] += -G$$

$$M[n_2, n_1] += -G$$

$$M[n_2, n_2] += G$$

$$G = \frac{1}{R}$$

This is used at the initial set up of every analysis.

2. Independent Current Source of value I

$$b[n_1] += -I$$

$$b[n_2] += I$$

Used for all analyses.

3. Independent Voltage Source of value V

n_v is the index of voltage source.

$$M[n_1, n_v] += 1$$

$$M[n_v, n_1] += 1$$

$$M[n_2, n_v] += -1$$

$$M[n_v, n_2] += -1$$

$$b[n_v] += V$$

Used for all analyses.

4. Current Controlled Current Source of value β

n_{cv} is the index of voltage source whose current is the controlling variable.

$$M[n_1, n_{cv}] += \beta$$

$$M[n_2, n_{cv}] += -\beta$$

5. Voltage Controlled Current Source of value g

n_{c1} and n_{c2} are indices of nodes of controlling voltage.

$$M[n_1, n_c1] += g$$

$$M[n_1, n_c2] += -g$$

$$M[n_2, n_c1] += -g$$

$$M[n_2, n_c2] += g$$

6. Capacitor of value C

Transient Analysis

For the first step:

$$G_{eq} = \frac{2 \cdot C}{\Delta t}$$
$$I_{eq} = G_{eq} \cdot V_i$$

G_{eq} is inserted in MNA, for only once, as in the case of resistors. Currents are treated as shown in the current sources.

For the other steps:

$$I_{eq} = I_p + 2 \cdot G_{eq} \cdot V_{op}$$

where Δt is the time step, V_i is the initial voltage, I_p is the equivalent current at the last step , V_{op} is the voltage across capacitor at the last step.

AC Analysis

$$G_{eq} = 2\pi j \cdot f \cdot C$$

where f is the frequency.

7. Inductor of value L

Transient Analysis

In this case, Thèvenin equivalent is replaced. Therefore, an index for the access node and another index for the voltage source is needed. Call them n_3 and n_v, respectively.

For the first step:

$$G_{eq} = \frac{\Delta t}{2 \cdot L}$$
$$V_{eq} = I_i / G_{eq}$$

For the other steps:

$$V_{eq} = V_{op} + I_p / G_{eq}$$

where Δt is the time step, I_i is the initial current, I_p is the current through the voltage source computed at the last step , V_{op} is the voltage across inductor at the last step.

Then the stencils will be as that of the resistor for G_{eq} between nodes n_2 and n_3, and as that of the voltage source for V_{eq} between nodes n_1 and n_3.

AC Analysis

$$G_{eq} = \frac{1}{2\pi j \cdot f \cdot L}$$

where f is the frequency.

8. Diode

DC and Transient Analysis

In the first step:

$$G_{eq} = I_s \cdot e^{V_{op}/V_t}/V_t$$

$$I_{eq} = I_s \cdot (e^{V_{op}/V_t} - 1) - G_{eq} \cdot V_{op}$$

$$M[n_1, n_1] += G_{eq}$$

$$M[n_1, n_2] += -G_{eq}$$

$$M[n_2, n_1] += -G_{eq}$$

$$M[n_2, n_2] += G_{eq}$$

$$b[n_1] += -I_{eq}$$

$$b[n_2] += I_{eq}$$

In the other steps the differences in G_{eq} and I_{eq} are calculated and inserted into the above entries.

9. Transmission Line

AC Analysis

The following equation will be inserted into matrix M.

$$Av + Bi = 0$$

explicitly,

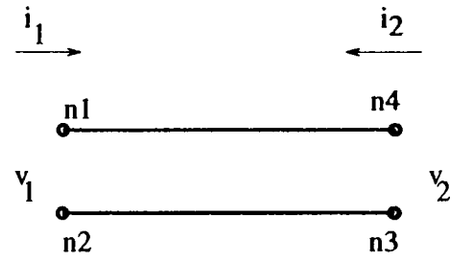


Figure C.1: A Transmission Line

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = 0$$

$$a_{11} = E_1$$

$$a_{12} = -1$$

$$a_{21} = E_2/Z0$$

$$a_{22} = 0$$

$$b_{11} = Z0 \cdot E_2$$

$$b_{12} = 0$$

$$b_{21} = E_1$$

$$b_{22} = 1$$

where,

$$E_1 = cosh(\gamma D)$$

$$E_2 = -sinh(\gamma D)$$

$$\gamma = j2\pi f\sqrt{LC}$$

$$\gamma D = j2\pi fNL/F$$

since

$$\sqrt{LC}D = NL/F$$

Either Z0 (characteristic impedance), F (frequency of the transmission line) and NL (normalized electrical length) or L (equivalent series inductance), C (equivalent shunt capacitance) and D (time delay) parameters may be given.

First of all a conversion is made, if L, C, D are given:

$$Z0 = \sqrt{L/C}$$

$$NL/F = \sqrt{LC} \cdot D$$

Then the stencil of the transmission line will be as (n_{i1} and n_{i2} are indexes of the additional current variables):

$$M[n_{i1}, n_1] = a_{11}$$

$$M[n_{i1}, n_2] = -a_{11}$$

$$M[n_{i1}, n_3] = a_{12}$$

$$M[n_{i1}, n_4] = -a_{12}$$

$$M[n_{i2}, n_1] = a_{21}$$

$$M[n_{i2}, n_2] = -a_{21}$$

$$M[n_{i2}, n_3] = a_{22}$$

$$M[n_{i2}, n_4] = -a_{22}$$

$$M[n_{i1}, n_{i1}] = b_{11}$$

$$M[n_{i1}, n_{i2}] = b_{12}$$

$$M[n_{i2}, n_{i1}] = b_{21}$$

$$M[n_{i2}, n_{i2}] = b_{22}$$

Note that since the other devices have no effect on the entries of the transmission line, "=" sign is used. The following assignment is done only at the first step.

$$M[n_1, n_{i1}] = 1$$

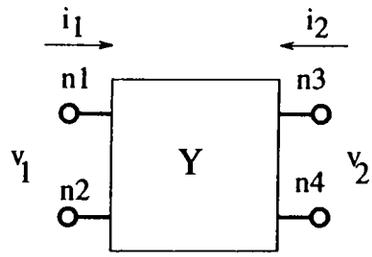


Figure C.2: A Two-port

$$M[n_2, n_{i1}] = -1$$

$$M[n_3, n_{i2}] = 1$$

$$M[n_4, n_{i2}] = -1$$

10. Two-port [1]

AC Analysis

The equation for two-port is:

$$\begin{bmatrix} y_{11} & y_{12} \\ y_{21} & y_{22} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} + \begin{bmatrix} -1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} i_1 \\ i_2 \end{bmatrix} = 0$$

The stencils are as follows(n_{i1} and n_{i2} are indexes of the additional current variables):

$$M[n_{i1}, n_1] = y_{11}$$

$$M[n_{i1}, n_2] = -y_{11}$$

$$M[n_{i1}, n_3] = y_{12}$$

$$M[n_{i1}, n_4] = -y_{12}$$

$$M[n_{i2}, n_1] = y_{21}$$

$$M[n_{i2}, n_2] = -y_{21}$$

$$M[n_{i2}, n_3] = y_{22}$$

$$M[n_{i2}, n_4] = -y_{22}$$

[1]The reader should check the declaration of two-port in SIMLIB Manual.

Note that since the other devices have no effect on the entries of the two-port, "=" sign is used. The following assignment is done only at the first step.

$$M[n_{i1}, n_{i1}] = -1$$

$$M[n_{i1}, n_{i2}] = 0$$

$$M[n_{i2}, n_{i1}] = 0$$

$$M[n_{i2}, n_{i2}] = -1$$

$$M[n_1, n_{i1}] = 1$$

$$M[n_2, n_{i1}] = -1$$

$$M[n_3, n_{i2}] = 1$$

$$M[n_4, n_{i2}] = -1$$

Appendix D

MATRIX and TABLE PACKAGES

D.1 Matrix packages

The directory of the matrix packages is "simlibhome/source/matdir/".

spm.C : real sparse matrix package.

cspm.C : complex sparse matrix package.

realnsp.C : real non-sparse matrix package.

comnsp.C : complex non-sparse matrix package.

The functions of *cspm* are :

```
Cspm *Cspmcreate(int n, int m);
void spmdestroy(Cspm *sa);
complex getelem(Cspm *in,int k, int l);
int  putelem(Cspm *in,int k,int l, complex x);
int  addelem(Cspm *in,int k,int l, complex x);
void spmprint(Cspm *sa);
complex *mspmsolve(Cspm *in,complex *b);
```

Cspm is the complex sparse matrix structure. Creation function is *Cspm-create* function. *putelem* and *addelem* funtions are used to form the matrix.

mspmsolve takes an nxn complex matrix (**M**) and nx1 complex vector (**b**) and returns the vector that is the result of $\mathbf{x} = \mathbf{M}^{-1}\mathbf{b}$.

spm is the same as cspm, but its variable type is *spm*.

The functions of realnsp are:

```
Rmat R,R1,R2;  (declation of matrices)
double r;
int i,j;       (indices)
operators defined :
                    R[i][j] = r;   (assignment of an entry
                                    of the matrix R)
                    R=R1;
                    R=R1+R2;       (addition of matrices)
                    R=R1-R2;       (subtraction of matrices)
                    R=r*R1; R=R1*r; R=R1*R2; (multiplication)
                    R=!R1;         (transpose)
                    R=R1<R2;       (R = inv(R1) * R2)
                    R=R1|R2;       (hconcat)
                    R=Id(n,m); (Identity matrix with size nxn
                               and diogonal value m (if not
                               given its value is 1))
                    R=Id(R1);  (Identity matrix with size of R1)
                    R=R1/R2;   (R = R1*inv(R2))
                    print(R);
  +=,-=,*=,/=,<=,|=   operators are also defined.
```

comnsp is the same as realnsp, but its variable type is *Cmat*.

D.2 Table Package

The file is "simlibhome/source/misc/table.C". The following is the include file of table.C.

```
class symtab
   {
      int n,nup,std;
```

```
        char **names;

public:
        symtab();
        ~symtab();
        int  nofsym();
        void deletesym(int i);
        void cleanreplicas();
        void sortnames();
        int  index(char *name);
        int  operator()(char* name);
        void printnames();
        char *operator[](int k);
        int  putsym(char *name);
};
```

For example, for a declaration as,

symtab st;

st.putsym(symbol_name); puts symbol *symbol_name* in table *st*, returns the number of symbols.

st.index(symbol_name); or st(symbol_name); returns the index of symbol *symbol_name* in table *st*, uses sorting and cleaning replicas.

st[i]; returns the name of the symbol at the i^{th} entry.

st.nofsym(); returns the number of symbols in table *st*.

The other functions are self-explanatory.

Appendix E

LISTINGS

E.1 Simlib Directory Structure

```
simlibhome/.

total 8
drwxr-xr-x  2 yazgan       512 Oct 24 12:52 bin/
drwxr-xr-x  2 yazgan       512 Oct 24 12:55 examples/
drwxr-xr-x  2 yazgan       512 Oct 24 13:34 include/
drwxr-xr-x  2 yazgan      1024 Oct 24 13:32 lib/
drwxr-xr-x  2 yazgan       512 Oct 23 11:15 man/
drwxr-xr-x  2 yazgan       512 Oct 23 11:16 manual/
drwxr-xr-x  8 yazgan      1024 Oct 24 13:32 source/
drwxr-xr-x  2 yazgan       512 Oct 24 14:15 tutorial/

bin:
total 520
-rwxr-xr-x  1 yazgan    328078 Oct 24 12:53 simgcc*   (compiled by g++)
-rwxr-xr-x  1 yazgan    180224 Oct 23 18:35 simlib*   (compiled by CC (Saber))

examples:
total 6
-rw-r--r--  1 yazgan       212 Oct 24 13:04 acdene
-rw-r--r--  1 yazgan        99 Oct 23 19:01 dcdene
-rw-r--r--  1 yazgan       133 Oct 23 19:30 ind

include:
```

```
total 26
-rw-r--r--  1 yazgan      1047 Oct 24 13:30 comnsp.h
-rw-r--r--  1 yazgan      1582 Oct 23 11:19 cspm.h
-rw-r--r--  1 yazgan       985 Oct 23 16:50 fourterm.h
-rw-r--r--  1 yazgan       827 Oct 23 15:47 globals.h
-rw-r--r--  1 yazgan       351 Oct 23 15:57 includes.h
-rw-r--r--  1 yazgan      1381 Oct 23 15:37 inp.h
-rw-r--r--  1 yazgan       467 Oct 23 11:19 insert.h
-rw-r--r--  1 yazgan       323 Oct 23 15:47 jobs.h
-rw-r--r--  1 yazgan       389 Oct 23 11:19 mna.h
-rw-r--r--  1 yazgan       100 Oct 23 14:57 models.h
-rw-r--r--  1 yazgan       499 Oct 24 13:34 moment.h
-rw-r--r--  1 yazgan       245 Oct 23 11:19 myread.h
-rw-r--r--  1 yazgan      1124 Oct 24 13:30 realnsp.h
-r--r--r--  1 yazgan      1454 Oct 23 11:19 spm.h
-rw-r--r--  1 yazgan      2502 Oct 23 11:19 subckt.h
-r--r--r--  1 yazgan       485 Oct 23 11:19 table.h
-rw-r--r--  1 yazgan      2516 Oct 23 16:50 twoterm.h

lib:
total 758
-rw-r--r--  1 yazgan      3648 Oct 23 18:46 ac.o
-rw-r--r--  1 yazgan      4672 Oct 23 18:46 capacitor.o
-rw-r--r--  1 yazgan      5224 Oct 23 18:46 cccs.o
-rw-r--r--  1 yazgan      4648 Oct 23 18:46 cs.o
-rw-r--r--  1 yazgan     27144 Oct 23 18:46 cspm.o
-rw-r--r--  1 yazgan      4188 Oct 23 18:46 dc.o
-rw-r--r--  1 yazgan     14388 Oct 23 18:46 device.o
-rw-r--r--  1 yazgan      5392 Oct 23 18:46 diode.o
-rw-r--r--  1 yazgan      5272 Oct 23 18:46 dojob.o
-rw-r--r--  1 yazgan      5796 Oct 23 18:46 inductor.o
-rw-r--r--  1 yazgan      2612 Oct 23 18:46 insert.o
-rw-r--r--  1 yazgan      5808 Oct 23 18:46 jobs.o
-rw-r--r--  1 yazgan      8724 Oct 23 18:47 liban.a
-rw-r--r--  1 yazgan     90030 Oct 23 18:47 libdev.a
-rw-r--r--  1 yazgan     21570 Oct 23 18:47 libinp.a
-rw-r--r--  1 yazgan      9450 Oct 23 18:47 libmet.a
-rw-r--r--  1 yazgan     18114 Oct 23 18:47 libmisc.a
-rw-r--r--  1 yazgan    142606 Oct 24 12:48 libsimgcc.a
-rw-r--r--  1 yazgan    200838 Oct 23 18:47 libsimlib.a
-rw-r--r--  1 yazgan     53334 Oct 23 18:47 libsp.a
-rw-r--r--  1 yazgan      5268 Oct 23 18:46 mna.o
-rw-r--r--  1 yazgan      4856 Oct 23 18:46 myread.o
-rw-r--r--  1 yazgan      3132 Oct 23 18:46 nr.o
```

```
-rw-r--r--  1 yazgan      1636 Oct 23 11:18 readdene.o
-rw-r--r--  1 yazgan      5224 Oct 23 18:46 resistor.o
-rw-r--r--  1 yazgan      2908 Oct 23 18:46 simmain.o
-rw-r--r--  1 yazgan     20768 Oct 23 18:46 spm.o
-rw-r--r--  1 yazgan      4224 Oct 23 18:46 subckt.o
-rw-r--r--  1 yazgan      3020 Oct 23 18:46 table.o
-rw-r--r--  1 yazgan     15200 Oct 23 18:46 transline.o
-rw-r--r--  1 yazgan     20812 Oct 23 18:46 twoport.o
-rw-r--r--  1 yazgan      5844 Oct 23 18:46 vccs.o
-rw-r--r--  1 yazgan      6588 Oct 23 18:46 vs.o

man:
total 0

manual:
total 0

source:
total 73
-rw-r--r--  1 yazgan     29209 Oct 24 12:46 Makefile
-rw-r--r--  1 yazgan      3903 Oct 23 14:02 Makefile.gcc
drwxr-xr-x  2 yazgan       512 Oct 23 19:45 analyses/
-rw-r--r--  1 yazgan      4702 Oct 23 19:46 center.prj (prj file for xobjectcenter)
drwxr-xr-x  2 yazgan       512 Oct 23 18:50 devdir/
drwxr-xr-x  2 yazgan       512 Oct 23 18:05 input/
drwxr-xr-x  2 yazgan       512 Oct 24 13:34 matdir/
drwxr-xr-x  2 yazgan       512 Oct 23 18:50 methods/
drwxr-xr-x  2 yazgan       512 Oct 24 12:53 misc/

source/analyses:
total 8
-rw-r--r--  1 yazgan      1334 Oct 23 17:24 ac.C
-rw-r--r--  1 yazgan      1387 Oct 23 17:25 dc.C
-rw-r--r--  1 yazgan      1237 Oct 23 19:45 tran.C

source/devdir:
total 42
-rw-r--r--  1 yazgan      2951 Oct 23 16:54 capacitor.C
-rw-r--r--  1 yazgan      2607 Oct 23 16:55 cccs.C
-rw-r--r--  1 yazgan      1984 Oct 23 16:55 cs.C
-rw-r--r--  1 yazgan      2394 Oct 23 16:56 diode.C
-rw-r--r--  1 yazgan      3651 Oct 23 16:59 inductor.C
-rw-r--r--  1 yazgan      2412 Oct 23 16:59 resistor.C
-rw-r--r--  1 yazgan      2523 Oct 23 17:31 subckt.C
```

```
-rw-r--r--  1 yazgan      5694 Oct 23 17:00 transline.C
-rw-r--r--  1 yazgan      7174 Oct 23 17:02 twoport.C
-rw-r--r--  1 yazgan      2776 Oct 23 17:02 vccs.C
-rw-r--r--  1 yazgan      3250 Oct 23 17:02 vs.C

source/input:
total 12
-rw-r--r--  1 yazgan      8603 Oct 23 15:59 device.C
-rw-r--r--  1 yazgan      3045 Oct 23 15:59 jobs.C

source/matdir:
total 89
-rw-r--r--  1 yazgan     11247 Oct 24 13:30 comnsp.C
-rwxr-xr-x  1 yazgan     28285 Oct 23 11:17 cspm.C*
-rw-r--r--  1 yazgan      2588 Oct 24 13:33 moment.C
-rw-r--r--  1 yazgan     12382 Oct 24 13:30 realnsp.C
-rw-r--r--  1 yazgan      5391 Oct 24 13:34 roots.C
-rw-r--r--  1 yazgan     27687 Oct 23 11:17 spm.C

source/methods:
total 6
-rw-r--r--  1 yazgan      3699 Oct 23 17:20 mna.C
-rw-r--r--  1 yazgan      1270 Oct 23 17:22 nr.C

source/misc:
total 13
-rw-r--r--  1 yazgan      1975 Oct 23 17:42 dojob.C
-rw-r--r--  1 yazgan       975 Oct 23 17:09 insert.C
-rw-r--r--  1 yazgan      2093 Oct 23 17:11 myread.C
-rw-r--r--  1 yazgan       616 Oct 24 12:53 simmain.C
-rw-r--r--  1 yazgan      4223 Oct 23 17:13 table.C

tutorial:
total 17
-rw-r--r--  1 yazgan      5500 Oct 24 14:13 readme
```

E.2 Global variables and functions

```
#define ISDEF 1e-15
#define VTDEF 0.025875

#define MAXCOLNO 130 // max column for input file

extern  char mess[80];
extern  void errormess();
extern  int get_index(incirc *curcirc,char *indexname);
extern  int lineno;
extern  circuits *circ;
extern  spm *M;
extern  double *b;
extern  Cspm *CM;
extern  complex *cb;


extern void output(double ilk,double *sol);

extern void output(double ilk,complex *sol);

extern void printMb();

extern void do_tran();

extern void do_dcsweep();

extern void do_ac();

extern void do_job();

extern double *NewtonRaphson(double *x,int &iterno);

extern double *do_dc(double *sol,double *sold,int &iterno);

extern  void errormess();

extern  int get_index(incirc *curcirc,char *indexname);
```

E.3 Two terminal device classes

```
typedef struct  pulsedef {      // Spice compatible  pulse definition
  double v1,v2,td,tr,tf,pw,per;
} pulse;


class res {

  int resno,count;
  int *n1,*n2;
  double *val;

 public:

  res() ;
  ~res() ;
  void first_pass(incirc *curcirc);
  void create_field();
  void second_pass(incirc *curcirc);
  void set_MNA();
  void set_CMNA();
  void list_all();
};


class vs {

  int vsno,count;
  int *n1,*n2;
  double *val;
  int *index;
  pulse *vpulse;

 public:

  vs() ;
  ~vs() ;
  void first_pass(incirc *curcirc) ;
  void create_field();
  void second_pass(incirc *curcirc);
  void set_MNA();
  void set_CMNA();
  void list_all();
```

```
 };


 class cs {

   int csno,count;
   int *n1,*n2;
   double *val;
   pulse *cpulse;

  public:

   cs() ;
   ~cs() ;
   void first_pass(incirc *curcirc) ;
   void create_field();
   void second_pass(incirc *curcirc);
   void set_MNA();
   void set_MNA(Cspm *M , complex *b);
   void list_all();
};


class vccs {

  int vccsno,count;
  int *n1,*n2,*nc1,*nc2;
  double *val;

 public:

  vccs() ;
  ~vccs() ;
  void first_pass(incirc *curcirc) ;
  void create_field();
  void second_pass(incirc *curcirc);
  void set_MNA();
  void set_CMNA();
  void list_all();
};



class cccs {
```

```
    int cccsno,count;
    int *n1,*n2,*cont_index;
    double *val;

  public:

    cccs() ;
    ~cccs() ;
    void first_pass(incirc *curcirc) ;
    void create_field();
    void second_pass(incirc *curcirc);
    void set_MNA();
    void set_CMNA();
    void list_all();
};
class cap {

   int *n1,*n2,count;
   double *val,*ic,*Ieqp,*Geqp;

 public:

   int capno;

   cap() ;
   ~cap() ;
   void first_pass(incirc *curcirc);
   void create_field();
   void second_pass(incirc *curcirc);
   void init_MNA();
   void set_MNA(double *sol,double dT);
   void set_CMNA(double freq);
   void list_all();
};


class ind {

  int *n1,*n2,*n3,count;
  double *val,*ic,*Geqp;
  int *index;

 public:
```

```
    int indno;

    ind() ;
    ~ind() ;
    void first_pass(incirc *curcirc);
    void create_field();
    void second_pass(incirc *curcirc);
    void init_MNA(spm *M , double *b);
    void init_MNA(Cspm *M , complex *b);
    void set_MNA(spm *M , double *b,double *sol,double dT);
    void set_init_CMNA(double freq);
    void set_CMNA(double freq);
    void list_all();
};


class diod {

   int *n1,*n2,count;
   double *is,*vt,*Ieqp,*Geqp;

 public:

   int diodno;

   diod() ;
   ~diod() ;
   void first_pass(incirc *curcirc);
   void create_field();
   void second_pass(incirc *curcirc);
   void set_MNA(double *sol);
   void list_all();
};
```

E.4 Four terminal device classes

```
class tline {
```

```
    int *n1,*n2,*n3,*n4,*index1,*index2,count;
    double *Z0,*F,*NL,*C,*L,*D;
  public:

    int tlineno;

    tline() ;
    ~tline() ;
    void first_pass(incirc *curcirc);
    void create_field();
    void second_pass(incirc *curcirc);
    void init_MNA(spm *M , double *b);
    void init_MNA(Cspm *M , complex *b);
    void set_init_CMNA(double freq);
    void set_CMNA(double freq);
    void list_all();
};

class twoport {

  int *n1,*n2,*n3,*n4,*index1,*index2,count;
  polrat *a11,*a12,*a21,*a22;

 public:

  int twoportno;

  twoport() ;
  ~twoport() ;
  void first_pass(incirc *curcirc);
  void create_field();
  void second_pass(incirc *curcirc);
  void init_MNA(spm *M , double *b);
  void init_MNA(Cspm *M , complex *b);
  void set_init_CMNA(double freq);
  void set_CMNA(double freq);
  void list_all();
};
```

E.5 Moment class

```
extern complex *roots(int degree, double *coeff);

class Moment {
  Rmat Int;
  Rmat Der;
  int first,last;

public :
  Moment(int in, int dn, double *I, double *D);
  Moment(int in, int dn);
  Moment(Rmat I,Rmat D);
  ~Moment();

  double& operator[](int n);

  friend Rmat coefficients(Moment &in);

  friend  Rmat  momlow(Moment &in);

  friend void print(Moment &in);
  friend Cmat residues(Moment &in,Cmat &pol);
};


extern Cmat poles(Rmat &in);
```

E.6 Jobs class

```
class jobs {


 public:

  char outnames[100];
```

```
    double start,stop,step;
    char vsname[20];
    int list,tranok,acok,dcok,aweok,printmb,showiter;
    int *outindex,*outform,outputno;

    jobs();
    ~jobs();
    void read_tran();
    void read_ac();
    void read_dc();
    void read_output();
    void read_opt();
    void read_model();
};
```

E.7 MNA setup functions

```
extern void ins_M(spm *M,int row,int col,double val);
extern void ins_b(double *b,int row, double val);
extern void ins_M(Cspm *M,int row,int col,complex val);
extern void ins_b(complex *b,int row, complex val);
extern void put_M(spm *M,int row,int col,double val);
extern void put_b(double *b,int row, double val);
extern void put_M(Cspm *M,int row,int col,complex val);
extern void put_b(complex *b,int row, complex val);
extern int MNAindex(int i,indbuf *buff);


extern void set_CMNA_incond();

extern void set_CMNA(double f);


extern void set_MNA();
extern void set_MNA_nonlin(incirc *curcirc,double *sol);
extern void set_MNA_dyn(double *sol);
extern void set_MNA_incond();
```

Appendix F

EXAMPLES

F.1 DC sweep analysis of a BJT inverter

The circuit given in Fig. F.1 is a three stage BJT inverter. The input deck is as follows. Instead of BJTs, their Ebers-Moll model is inserted.

```
***Three stage BJT inverter
vs 1 0 0
vcc 8 0 5
Rb1 1 5 10000
Rc1 8 2 1000
Rb2 2 6 10000
Rc2 8 3 1000
Rb3 3 7 10000
Rc3 8 4 1000

dbc1 5 9
dbe1 5 10
dbc2 6 11
dbe2 6 12
dbc3 7 13
dbe3 7 14

vbc1 9 2 0
vbe1 10 0 0
vbc2 11 3 0
```

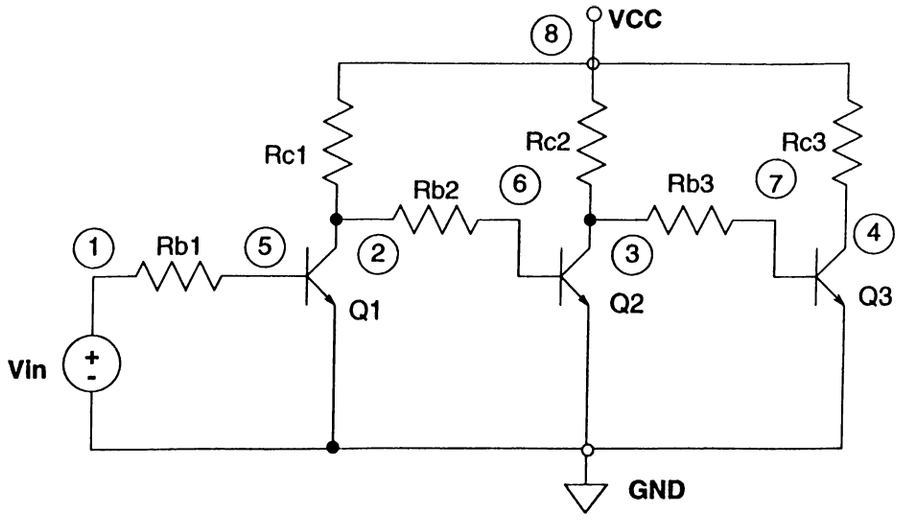


Figure F.1: Three stage BJT inverter

```
vbe2 12 0 0
vbc3 13 4 0
vbe3 14 0 0

fcb1 2 5 vbe1 0.99
feb1 0 5 vbc1 0.66
fcb2 3 6 vbe2 0.99
feb2 0 6  vbc2 0.66
fcb3 4 7 vbe3 0.99
feb3 0 7 vbc3 0.66

.dc vs 0 5 0.05
.print v(2) v(3) v(4)
.options iterno
.end
```

The input voltage is swept from 0 to 5 volts. The output waveforms at the collectors of the BJTs are shown and compared with the SPICE outputs in Fig. F.2. SPICE and SIMLIB results are indistinguishable.

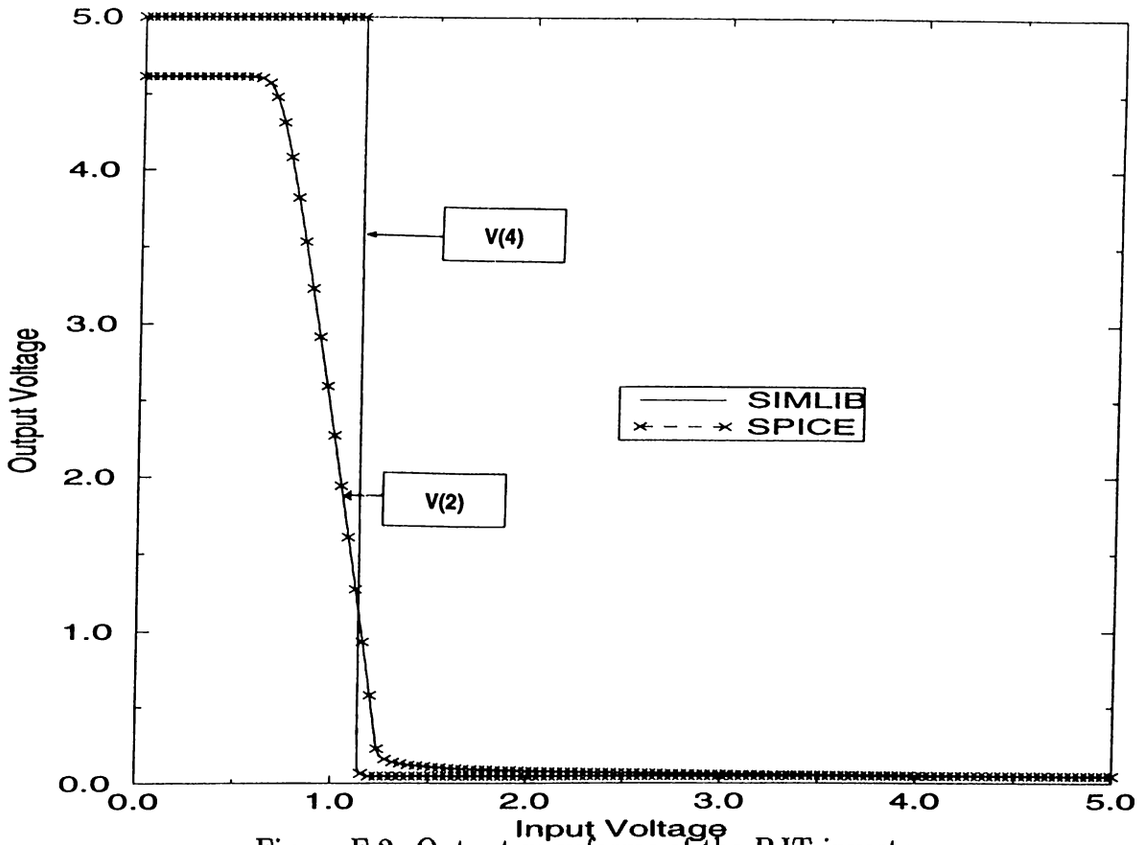


Figure F.2: Output waveforms of the BJT inverter

F.2 AC analysis of a transmission line circuit

For AC analysis, an example circuit having dynamic elements and a transmission line is given. The circuit is shown in Fig. F.3. The input deck is as follows.

```
***AC example
r1 0 2 100
l1 2 1 1e-9
c1 2 1 1e-12
vs 1 0 1
g1 0 3 2 0 0.01
r2 3t 0 100
t1  3 0 3t 0 Z0=50  F=1732Meg NL=.069444

.ac 1G 10G 100MEG
.print vm(2) vp(2) vm(3t) vp(3t)
*.options  list iterno printmb
.end
```

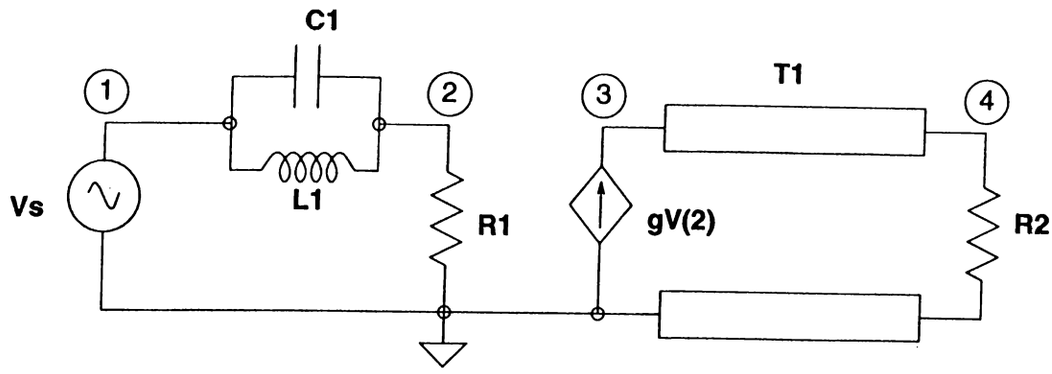


Figure F.3: Example circuit for AC analysis

The frequency range is 1 Gigahertz to 10 Gigahertz. The magnitude and phase of the output at node 4 is given in Fig. F.4. SPICE and SIMLIB results are indistinguishable.

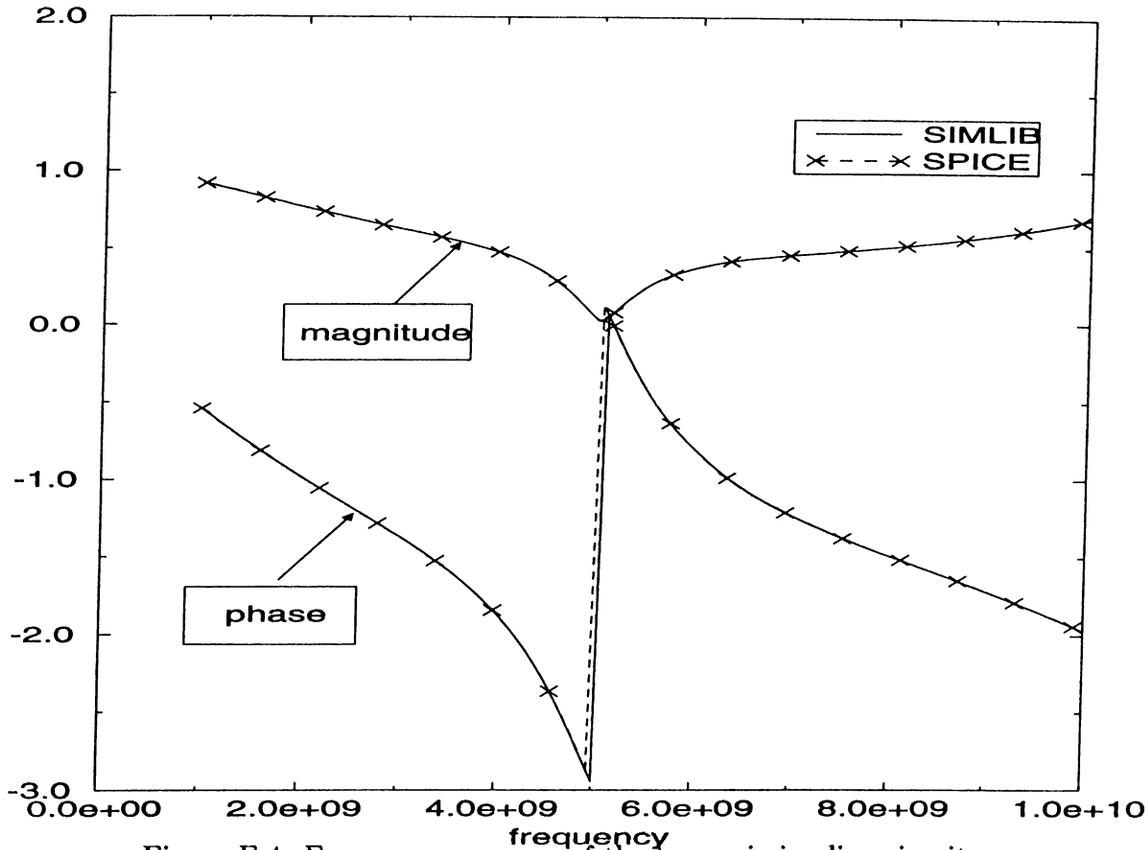


Figure F.4: Frequency response of the transmission line circuit

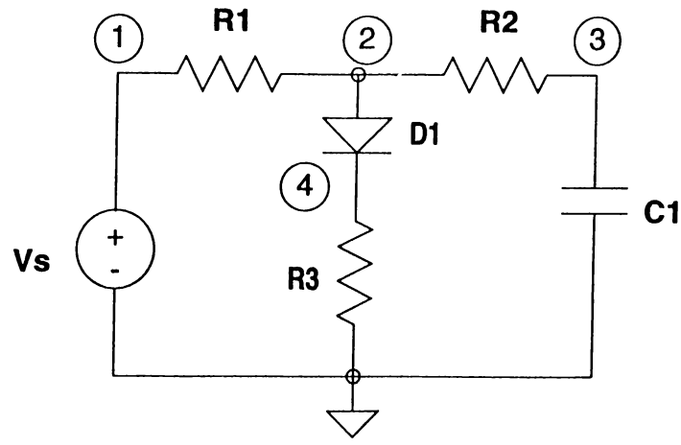


Figure F.5: Example circuit for transient analysis

F.3 Transient analysis of a nonlinear dynamic circuit

An example circuit having a diode as a nonlinear element and a capacitor is given in Fig. F.5. And the transient response of the circuit is shown in Fig. F.6. Again, SPICE and SIMLIB results are indistinguishable. The input deck is as follows:

```
***transient example
vs 1 0 10
r1 1 2 500
d1 2 4
r3 4 0 100
r2 2 3 500
c1 3 0 1e-5 ic=-5

.tran 0.0001 0.01
.print v(2) i(vs) v(3)
.options list
.end
```

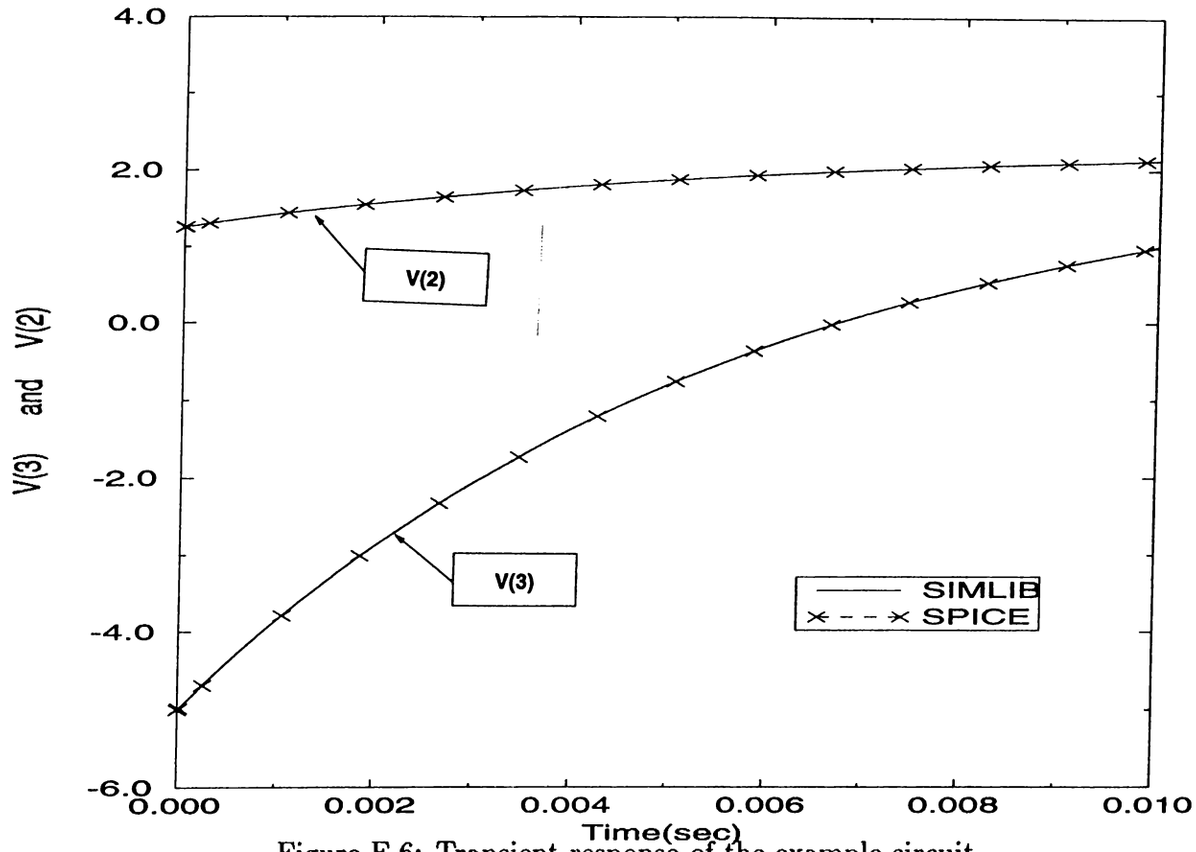


Figure F.6: Transient response of the example circuit

REFERENCES

[1] L. W. Nagel. "SPICE2: A computer program to simulate semiconductor circuits,". Technical Report Memo ERL-M520, University of California, Berkeley, May 1975.

[2] C. W. Ho, A. E. Ruehli, and P. A. Brennan "The Modified Nodal Approach to network analysis," *IEEE Trans. Circuit Theory*, vol. CAS-22, pp. 504–509, June 1975.

[3] Leon. O. Chua and P.-M. Lin. *Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques.* Prentice-Hall, Englewood Cliffs, NJ, 1975.

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

[5] L. T. Pillage and R. A. Rohrer "Asymptotic Waveform Evaluation for Timing Analysis," *IEEE Trans. on Computer-Aided Design*, vol. 9, pp. 352–366, April 1990.

[6] C. T. Dikmen, M. M. Alaybeyi, S. Topcu, A. Atalar, E. Sezer, M. A. Tan, and R. A. Rohrer "Piecewise linear asymptotic waveform evaluation for transient simulation of electronic circuits," in *Proc. of the Int. Symp. on Circuits and Systems 1991 (ISCAS '91)*, pp. 854–857, Singapore, June 1991.

[7] S. Topçu. *PLAWE : A piecewise linear circuit simulator using asymptotic waveform evaluation.* PhD thesis, Bilkent University, 1994.

[8] O. Ocalı, M. A. Tan, and A. Atalar. "A new method for nonlinear circuit simulation in time domain : NOWE,". Submitted for publication.

[9] M. Çelik. *Fast algorithms for linear and nonlinear microwave circuit simulation.* PhD thesis, Bilkent University, 1994.

[10] B. Stroustrup. *The C++ programming language.* Addison-Wesley Publishing Company, 1991.

[11] Leon. O. Chua, Charles A. Desoer, and Ernest S. Kuh. *Linear and Nonlinear Circuits.* McGraw-Hill, New York, NY, 1987.

[12] S. Topçu, O. Ocalı, A. Atalar, and M.A. Tan, "A Novel Algorithm for DC Analysis of Piecewise Linear Circuits: POPCORN," *IEEE Trans. Circuits Syst.-I*, Accepted for publication.

[13] P. K. Chan "Comments on Asymptotic waveform evaluation for timing analysis," *IEEE Trans. on Computer-Aided Design*, vol. 10, pp. 1078–1079, August 1991.

[14] T. K. Tang, M. S. Nakhla, and J. R. Griffith "Analysis of Lossy Multiconductor Transmission Lines using the Asymptotic Waveform Evaluation Technique," *IEEE Trans. on Microwave Theory Tech.*, vol. 39, pp. 2107–2116, December 1991.

[15] John Y. Lee, Xiaoli Huang, and Ronald A. Rohrer "Pole and Zero Sensitivity Calculation in Asymptotic Waveform Evaluation," *IEEE Trans. on Computer-Aided Design*, vol. 11, pp. 586–597, May 1992.

[16] M. Celik, O. Ocali, M.A. Tan, and A. Atalar. "Pole-zero computation in microwave circuits using multipoint Padé approximation,". submitted for publication.

[17] X. Huang, V. Raghavan, and R. A. Rohrer "AWEsim: A Program for the Efficient Analysis of Linear(ized) Circuits," in *Tech. Digest IEEE Trans. Computer-Aided Design*, pp. 534–537, November 1990.

[18] E. Chiprout and M. Nakhla. *Asymptotic Waveform Evaluation and Moment Matching for Interconnect Analysis.* Kluwer Academic Publishers, Boston, 1993.

[19] V. Raghavan and R.A. Rohrer "AWESpice: A general tool for the efficient and accurate simulation of interconnect problems," in *Proc. Design Automation Conference*, June 1992.