BUSTLE, A NEW CIRCUIT SIMULATION TOOL

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
M. Murat ALAYBEYİ
July 1990

BUSTLE, A NEW CIRCUIT SIMULATION TOOL

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

M. Murat Alaybeyi

By
M. Murat Alaybeyi
July 1990

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(Principal Advisor)

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. Mehmet Ali Tan

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. Ayhan Altıntaş

Approved for the Institute of Engineering and Sciences:

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

to my father

ABSTRACT

BUSTLE, A NEW CIRCUIT SIMULATION TOOL

M. Murat Alaybeyi
M.S. in Electrical and Electronics Engineering
Supervisor: Prof. Dr. Abdullah Atalar
July 1990

A new circuit simulation tool, BUSTLE, using *Asymptotic Waveform Evaluation (AWE)* technique and *Piece-wise Linear (PWL)* models, is implemented. The results are very promising, especially for large circuits.

This piece of work, in cooperation with [1], explains the techniques used in the simulator BUSTLE, such as

- efficient LU decomposition of the sparse matrices,
- using derivative and integral moments in order to get rid of the instability problem,
- combining the PWL approach with AWE in transient analysis,

and illustrates some simulation results.

ÖZET

BUSTLE, YENİ BİR DEVRE SİMÜLATÖRÜ

M. Murat Alaybeyi
Elektrik ve Elektronik Mühendisliği Bölümü Yüksek Lisans
Tez Yöneticisi: Prof. Dr. Abdullah Atalar
Temmuz 1990

BUSTLE, asimptotik eğri tahmini ve parçalı doğrusal yaklaşımını kullanan yeni bir devre çözümleme programıdır. Elde edilen sonuçlar, özellikle büyük devreler için oldukça olumludur.

Bu çalışma, [1] ile birlikte, BUSTLE'de kullanılınan bazı metodları açıklamaktadır. Bunlar arasında

- seyrek elemanlı matrislerin etkili üçgensel ayrıştırılması,
- asimptotik eğri tahmini metodunun kararsızlık problemini çözmek için türev ve entegral momentlerin birlikte eşleştirilmeleri,
- zamanda geçici inceleme için asimptotik eğri tahmini ve parçalı doğrusal yaklaşım tekniklerinin birleştirilmesi

sayılabilir. Son olarak bazı simülasyon sonuçları verilmiştir.

ACKNOWLEDGMENT

My first debt is to Prof. Abdullah Atalar who has taught me the meaning of the word "research", and provided perfect dosage of criticism and unfailing support throughout the project. I am thankful for his proper combination of justice and mercy on deadlines to ensure the completion of this thesis. I explicitly want to acknowledge the support of Assoc. Prof. Mehmet Ali Tan for his critical guidance and useful advice. In addition I am grateful to Prof. Ronald Rohrer of Carnegie Mellon University not only for his encouragement and suggestions but also for initiating the research.

I owe debts of gratitude, too, to Cemal Tamer Dikmen and Satılmış Topçu who are members of CAD group in Bilkent University, for their co-operation in the project. This thesis would not have existed without their contributions. No brief statement can adequately acknowledge the many kinds of help that I have received from Cemal both in the project and in the course of preparing this thesis, even the "acknowledgment" part.

A special note of thanks is due to another contributor, Prof. Erol Sezer for his stimulating insights, and to Assoc. Prof. Ayhan Altıntaş for his complementing study. I am indebted as well to Mujtaba Fidaul Haq for his co-operation in forming the LU package, and to Gözde Bozdağı, Gökhun Tanyer and Mustafa Karaman for their helps in shaping the project in its early stages. I feel obliged to mention Ogan Ocalı and his nice discussion as well as his great pleasure in the exchange of ideas often on the way home from the office at midnight.

Finally I owe a measure of gratitude to my respective institution, Bilkent University and to the sponsor of the project NATO-SFS.

Contents

List of Figures

List of Tables

Chapter 1

INTRODUCTION

Simulation plays an important role in the design of integrated circuits. Using simulation, a designer can determine the functionality and the performance of a design before the expensive and time-consuming step of manufacture. BUSTLE, (**B**ilkent **U**niversity **S**imulation **T**ool for **L**inearized **E**nvironment), is a circuit level simulator determining the analog waveforms for the branch voltages and currents and the node voltages.

Circuit simulation tools, with different accuracy and speed, are used in circuit analysis and design. The accuracy and speed requirements vary depending on the type and size of the circuit, and the aims of the user. There is a trade-off between the accuracy and the speed of the simulation. This trade-off assumes the most important role when the number of elements increases prohibitively as is the case in VLSI circuits.

The extensive computations and thus very long simulation time are mainly due to the complex nonlinear characteristic of devices and the large number of iterations for computing the transient response in timing analysis. Most of the circuit simulators employ various numerical and iterative methods (e.g. Newton Raphson) to find the operating points of nonlinear circuits and numerical integration methods (Forward Euler, Backward Euler, Trapezoidal, etc.) to compute the transient response of energy storage elements.

Aspects of stability, convergence and hence completion of the job in a successful manner are all important issues for circuit simulators. Moreover the models of new devices resulting from the emerging technology must be easily put into a simulator. Otherwise, the simulator may become obsolete in a short time. The simulator should have provisions such that even the user has the capability of doing this integration.

BUSTLE is developed, with the motivation of the above facts. The first aim is to complete the simulation successfully without any problem such as convergence. BUSTLE employs Asymptotic Waveform Evaluation (AWE) technique instead of numerical integration methods, in order to compute the response of energy storage elements.

Piece-wise linear (PWL) representation is used to characterize the nonlinear elements. The main reason to choose the PWL characterization is to avoid solving nonlinear equations and to deal with a set of linear equations to decrease the time complexity and guarantee the convergence in DC Analysis. AWE is mainly for linear(ized) circuits. Therefore, the use of PWL approximation makes the utilization of AWE easy and efficient for nonlinear devices.

Another important advantage facilitated by PWL approximation is that the user can define his own device models for nonlinear devices easily. Subsequently, it provides a considerable flexibility to the user for choosing his own model and determining the trade-off between speed and accuracy in the simulation. This also renders the simulator independent of the trends of technology.

Consequently, PWL approach brings the advantages of guaranteed DC convergence, efficient usage of AWE, and user defined modeling property.

While using AWE, transient analysis is the part that calls for most of the attention and care. Since the approximate poles and the residues are found for any output of the circuit, only a simple plotting routine does the AC analysis. Note that it is also possible to give an approximate proper rational transfer function of a system in terms of s. The sensitivity analysis using AWE technique may also be performed with a little additional cost [5][20].

BUSTLE is the result of a co-operative study of the CAD group in the Department of Electrical and Electronics Engineering of Bilkent University. The whole of the program is written in *C*, using the *UNIX* operating system and some programming tools (i.e. Lex, Yacc) and SUN 3/110 Workstations. This thesis is a complementary work with [1]. Some important points are not examined in detail but referred to [1].

The following chapter, DC analysis, discusses piece-wise linear modeling, and determination of the operating points using PWL approach. Chapter 3 is about the implementation of LU decomposition of sparse matrices on the computer. This chapter mainly deals with the data structures and algorithms used. Chapter 4 consists of a short description of AWE technique, and the moment matching algorithm used in the implementation. A detailed examination of the concepts of this chapter can be found in [1]. Then merging PWL with AWE for the transient analysis is discussed in the last section.

Chapter 5 is a collection of some simulation results. Some other examples can be found in [1]. This chapter illustrates the advantages that is brought by BUSTLE. The results are compared with those of SPICE. Chapter 6 is a guide for users, and finally the conclusions and the future work can be found in Chapter 7.

Chapter 2

DC ANALYSIS

The solutions to a circuit with DC inputs are called *operating points.* The term *DC analysis* refers to the determination of the operating points. The behavior of non-linear circuits are quite different from that of linear circuits. Though there may be no solution or multiple solutions for a non-linear circuit, there is always a unique solution for a linear circuit. BUSTLE uses piece-wise linear (PWL) techniques to determine the operating points of a circuit. Therefore, a nonlinear network is replaced by a piecewise-linear network with a corresponding simplification of the problem. Consequently, solution of nonlinear algebraic systems of equations are reduced to the solution of a set of linear systems of equations :

$$\mathbf{M}\,\mathbf{x} \;=\; \mathbf{b} \tag{2.1}$$

where **M** is the matrix that describes the resistive network, and **b** is the source vector.

2.1 Determination of the Operating Points

The general methods used to determine the operating points are mesh analysis, nodal analysis, and the tableau analysis. BUSTLE uses tableau analysis which is a completely general analysis method and works for all linear circuits. This method avoids some restrictions caused by nodal and mesh analysis and conceptually it is simpler than the others. Tableau analysis consists of writing out the complete list of linearly independent KCL equations, linearly independent KVL equations, and the branch equations. KCL can be expressed as

$$\mathbf{A}\,\mathbf{i}_b \;=\; \mathbf{0} \tag{2.2}$$

whereas the KVL is given by

$$\mathbf{v}_b - \mathbf{A}^T \mathbf{v}_n = \mathbf{0} \tag{2.3}$$

where **A** is the *reduced incidence matrix* [24], $\mathbf{i}_b$ is the vector containing branch currents, $\mathbf{v}_b$ and $\mathbf{v}_n$ are the branch and node voltages respectively.

Branch constitutive equations can be written as

$$\mathbf{G}\,\mathbf{v}_b + \mathbf{R}\,\mathbf{i}_b = \mathbf{w} \tag{2.4}$$

where **w** is the vector including the independent current and voltage sources, as well as the influence of initial conditions on capacitors and inductors. This vector also includes the equivalent sources due to linearization of nonlinear elements. Equations (2.2)-(2.4) can be put into one matrix equation.

$$\underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & -\mathbf{A}^T \\ \mathbf{0} & \mathbf{A} & \mathbf{0} \\ \mathbf{G} & \mathbf{R} & \mathbf{0} \end{bmatrix}}_{\mathbf{M}} \underbrace{\begin{bmatrix} \mathbf{v}_b \\ \mathbf{i}_b \\ \mathbf{v}_n \end{bmatrix}}_{\mathbf{x}} = \underbrace{\begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{w} \end{bmatrix}}_{\mathbf{b}} \tag{2.5}$$

Listing the tableau equations, none of the variables is eliminated so all three vectors $\mathbf{v}_b$, $\mathbf{i}_b$, and $\mathbf{v}_n$ are present as variables. Since we must have as many equations as there are variables, it is clear that the price we pay for the increased generality is that the tableau analysis involves many more equations than node analysis does. In computer-aided circuit analysis, however, this objection turns out to be an advantage because the matrix associated with tableau analysis is extremely sparse which brings the benefits of highly efficient numerical algorithms. The algorithms and the data structure used to solve this sparse systems of linear equations are described in Chapter 3.

2.2 Modeling of Nonlinear Devices

Modeling is the process by which the electrical properties of a non-linear device is represented by means of mathematical equations or tables. Physical device models usually involve many complicated equations. Typical timing studies have shown that the major part of the computational effort in network analysis is spent in evaluating these complicated relationships. Further, most analysis methods also require derivatives of the model equations, which is a cumbersome and error-prone task for the designer. The iterative methods, such as Newton-Raphson, to solve the nonlinear equations do not guarantee the convergence. In

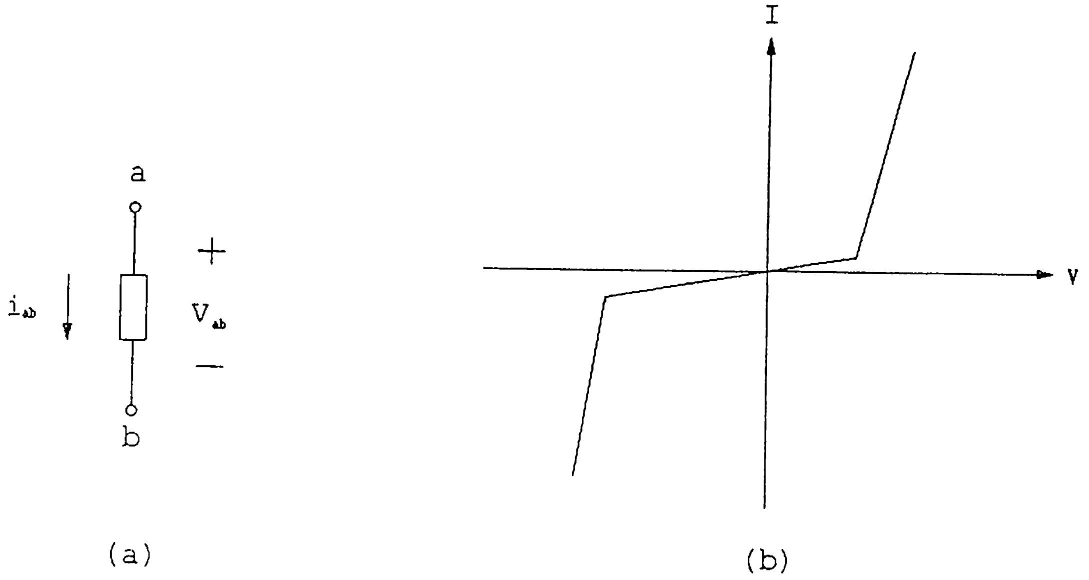


Figure 2.1: (a) Representation of a two-terminal nonlinear device; (b) i-v characteristics of a two-terminal nonlinear device.

order to avoid these problems, piece-wise linear (PWL) representation is used in BUSTLE for the modeling of nonlinear devices. Table models are employed to describe two and three terminal nonlinear devices.

2.2.1 Modeling of two-terminal nonlinear devices

Consider a two terminal element as shown in Fig. 2.1.a. If the voltage v across the element, and the current i which enters the element satisfies $f(v,i) = 0$ for every time instant, it is called a *resistor*. Probably the most familiar circuit element is the two terminal linear resistor. Linear resistor is a special case of the resistor and Ohm's law states that, at all times

$$f(v,i) = v - R\,i = i - G\,v = 0 \tag{2.6}$$

where the constant R is the *resistance* and the constant G is the *conductance*. Equation 2.6 can be represented in either the i-v plane or the v-i plane.

The i-v characteristics of two-terminal nonlinear resistors is approximated with a piece-wise linear model, passing through some sample i-v values which are given by the user. BUSTLE assumes that the i-v characteristics is linear between these points, so that the characteristics becomes a combination of linear segments. By using these values, the resistance R_l or the conductance G_l, and the equivalent source w_l due to linearization of a two-terminal nonlinear device can be calculated very easily, where l is the segment number. The branch equation of the lth segment of an element can be written as

$$\begin{aligned} i &= i_l^0 + G_l v \\ v &= v_l^0 + R_l i \end{aligned} \tag{2.7}$$

and, in general,

$$\mathbf{G}_l\,\mathbf{v}_b + \mathbf{R}_l\,\mathbf{i}_b = \mathbf{w}_l + \mathbf{w} \tag{2.8}$$

The tableau equations can be written as follows:

$$\begin{bmatrix} \mathrm{i} & 0 & -\mathbf{A}^T \\ 0 & \mathbf{A} & 0 \\ \mathbf{G}_l & \mathbf{R}_l & 0 \end{bmatrix} \begin{bmatrix} \mathbf{v}_b \\ \mathbf{i}_b \\ \mathbf{v}_n \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ \mathbf{w}_l \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \mathbf{w} \end{bmatrix} \tag{2.9}$$

or in a more compact form

$$\mathbf{M}_l\,\mathbf{x}_l = \mathbf{w}_l + \mathbf{w} \tag{2.10}$$

The subscript l denotes the *segment* in which the network operates. The right-hand-side vectors denote the equivalent sources due to linearization and the independent sources respectively; they are written separately for clarity.

2.2.2 Modeling of three-terminal nonlinear devices

Three-terminal nonlinear devices are represented as a combination of two 2-terminal devices placed between the three nodes as shown in Figure 2.2. The parasitic capacitors are also included in the device model. The values of these capacitors are given in the model card.

The characteristics of a three-terminal nonlinear device is defined by two branch equations and a number of boundaries for each region. These two branch equations define a 2-dimensional hyperplane in the 4-dimensional space which describes the characteristics of the nonlinear device where the boundaries describe the region at which these equations are valid. The two branch equations for the three-terminal nonlinear devices are of the general form:

$$a_1v_1 + a_2v_2 + a_3i_1 + a_4i_2 + a_5 = 0 \tag{2.11}$$

and at most three of the coefficients a_1, a_2, a_3, a_4 can be nonzero. This does not impose any restriction on the generality since one of them can be eliminated using the other plane equation. The boundaries are described by the inequalities of the following form,

$$a_1v_1 + a_2v_2 + a_3i_1 + a_4i_2 + a_5 \geq 0 \tag{2.12}$$

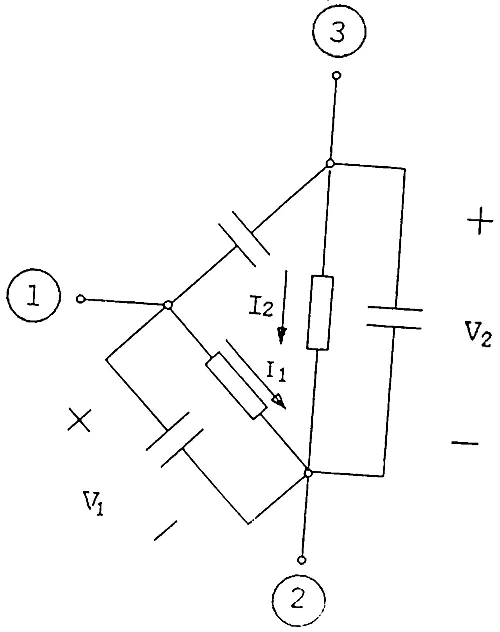


Figure 2.2: Representation of a three-terminal nonlinear device.

and at most two of the coefficients a_1, a_2, a_3, a_4 can be nonzero since two of the variables can be eliminated, using the two branch equations of the region. Each nonlinear device must contain a segment that satisfies the origin (both the equations and the boundaries) in order to have a valid solution when all of the independent sources are killed. This is required in order to start the DC analysis which will be described in the next section. Also this rule does not impose any restriction on the generality, since any device which does not fit this rule can be modeled with a PWL device satisfying this rule and an independent current source in parallel.

2.3 The Algorithm Used

For DC solution, all inductors and capacitors are replaced by independent sources. Given a valid solution x_0 for an arbitrary source vector y_0, satisfying the boundaries of the region R_0,

$$\mathrm{M}_0\,\mathrm{x}_0 = \mathrm{w}_0 + \mathrm{y}_0 \tag{2.13}$$

we would like to find the solution x and the region R_f for a given source vector y

$$\mathrm{M}_f\,\mathrm{x} = \mathrm{w}_f + \mathrm{y}. \tag{2.14}$$

The algorithm used for the DC Analysis has been derived from the *Katzenelson's algorithm* [9, 10] which guarantees the convergence in the DC Analysis [8].

The modified version of the Katzenelson's algorithm, used to find the solution $\mathbf{x}$ and the final operating region set R_f, is as follows:

1. Set $i = 0$.
2. Solve $\mathbf{x}$ from
   $\mathbf{M}_i \mathbf{x} = \mathbf{w}_i + \mathbf{y}$.
3. If $\mathbf{x}$ satisfies the boundaries of R_i then TERMINATE, else GOTO 4.
4. Let λ be the ratio of the distance from $\mathbf{x}_i$ to the first region boundary crossed when traversing from $\mathbf{x}_i$ to $\mathbf{x}$, to the distance from $\mathbf{x}_i$ to $\mathbf{x}$.
5. Compute $\mathbf{x}_{i+1}$ as
   $\mathbf{x}_{i+1} = \mathbf{x}_i + \lambda(\mathbf{x} - \mathbf{x}_i)$.
6. Set R_{i+1} to the neighbor region of R_i separated from it with the first crossed boundary.
7. Increment i and GOTO 2.

Note that for the first DC analysis, $\mathbf{x}_0$ and $\mathbf{y}_0$ can be selected as $\mathbf{0}$, since by definition, every nonlinear element is modeled to have a passive resistive segment satisfying the origin, and $\mathbf{0}$ is the solution when all the independent sources are killed. It is obviously a poor starting point, but that is the only valid solution known initially. Afterwards the lastly found operating points are chosen as $\mathbf{x}_0$, $\mathbf{y}_0$, and R_0. We are expected to do less computation starting from the last solution, since it is more probable that the old solution is closer to the new one than the origin ($\mathbf{0}$).

Chapter 3

LU DECOMPOSITION

The major part of the computation time of the circuit simulation tools is spent on finding the solutions of sets of linear equations. BUSTLE uses LU decomposition and Forward and Backward Substitutions to solve these equations. This section is a detailed explanation of the methods used by BUSTLE to solve linear sets of equations efficiently.

The simulator uses piecewise linear (PWL) approach in DC analysis which is performed several times throughout the program. In DC analysis *Sparse Tableau Analysis* is used which in the end , boils down to finding the solution of the matrix equation $\mathbf{Mx} = \mathbf{b}$. The matrix $\mathbf{M}$ is usually a very sparse matrix for large circuits, having only 0.1% or fewer non-zero elements on the average. DC analysis generally performs a set of LU decompositions in order to find the operating segments of the PWL devices. Most of the LU decompositions performed by the simulator are this type. There is also another LU decomposition following each DC analysis in order to find the steady state solutions of the circuit.

Solving a set of linear equations on the computer efficiently is a problem which has been investigated for a long time and is also being investigated now [14, 13, 11]. The problem gets even more complicated when the set of equations form a sparse matrix. The problem springs from the fact that it is not straightforward to harvest the advantages of the sparsity of the matrix that can be translated to profitable gains in memory consumption, computational speed and accuracy.

3.1 The Function of LU Decomposition

This part of the program is a package of C functions used to quickly and accurately solve large sparse and real systems of linear equations which can be expressed as $\mathbf{Mx} = \mathbf{b}$. The package is optimized for speed using a good strategy of pivot selection and is able to perform numerical pivoting to avoid numerical inaccuracy in the solution. What it actually does is,

- It decomposes $\mathbf{M}$ into two triangular matrices such that $\mathbf{M} = \mathbf{LU}$, where $\mathbf{L}$ is a lower triangular matrix and $\mathbf{U}$ is an upper triangular matrix.
- then it solves the $\mathbf{y}$ vector from $\mathbf{Ly} = \mathbf{b}$ by forward substitution which is a very easy task. Next $\mathbf{Ux} = \mathbf{y}$ is solved by a backward substitution where $\mathbf{y}$ is found from the forward substitution. This whole process is called *Forward and Backward Substitution (FBS)*. Once the LU decomposition is performed, the $\mathbf{x}$ vector can be solved for the same $\mathbf{M}$ matrix and different $\mathbf{b}$ vectors using only FBS.

3.2 Implementation of the Program Taking Various Aspects Under Consideration

3.2.1 Criteria of Efficiency

Efficiency of the program is characterized by the following points

Computation time : Since the speed of the whole simulator crucially depends on the computation time of both the LU factorization and FBS (especially for large circuits), they must be executed as fast as possible.

Minimization of the computation time is one of the major concerns for us while developing the program. We have been extremely cautious to cut off any redundancy in the computation to make the program as fast as possible.

Accuracy : Since various routines of the simulator uses the LU factorization and FBS quite frequently, the error in computing them has an immense effect on the overall accuracy of the simulator. It should be mentioned here that algorithms used to compute LU decomposition and FBS invariably causes an error which is cumulative in nature, though the error

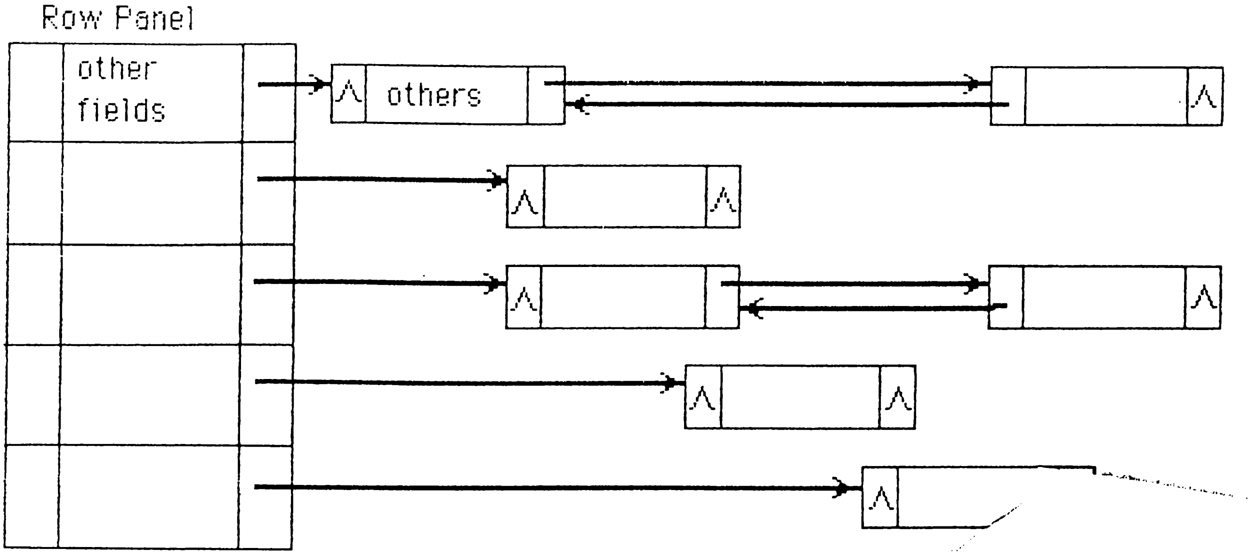


Figure 3.1: A possible one dimensional data structure for the M matrix

incurred by the simulator itself may or may not be cumulative. Various methods like numerical pivoting, minimization of fill-in's etc. which will be described later, are used in order to make as little compromise with the accuracy of the program as possible. If necessary, the program has the capability of doing iterative refinements on the solution.

In order to increase the accuracy one has to make frequency scaling [4], which is not related with LU decomposition.

Memory Consumption : It is quite reasonable to expect that owing to the sparse nature of the matrix **M**, an efficient and compact use of the memory is possible. But minimizing the consumption of the memory may not optimize the overall performance of the program. In this program though we have been careful not to make waste of memory allocation, we did not seek optimizing the program from the memory consumption standpoint.

3.2.2 Possible Data Structures and the Data Structure Actually Used

There can be several ways to store the sparse matrix in an efficient form. Two of the feasible data structures are illustrated in Fig. 3.1 and Fig. 3.2, while the data structure we have actually used is shown in Fig. 3.3.

In the data structure we have used, the rows and columns are stored as

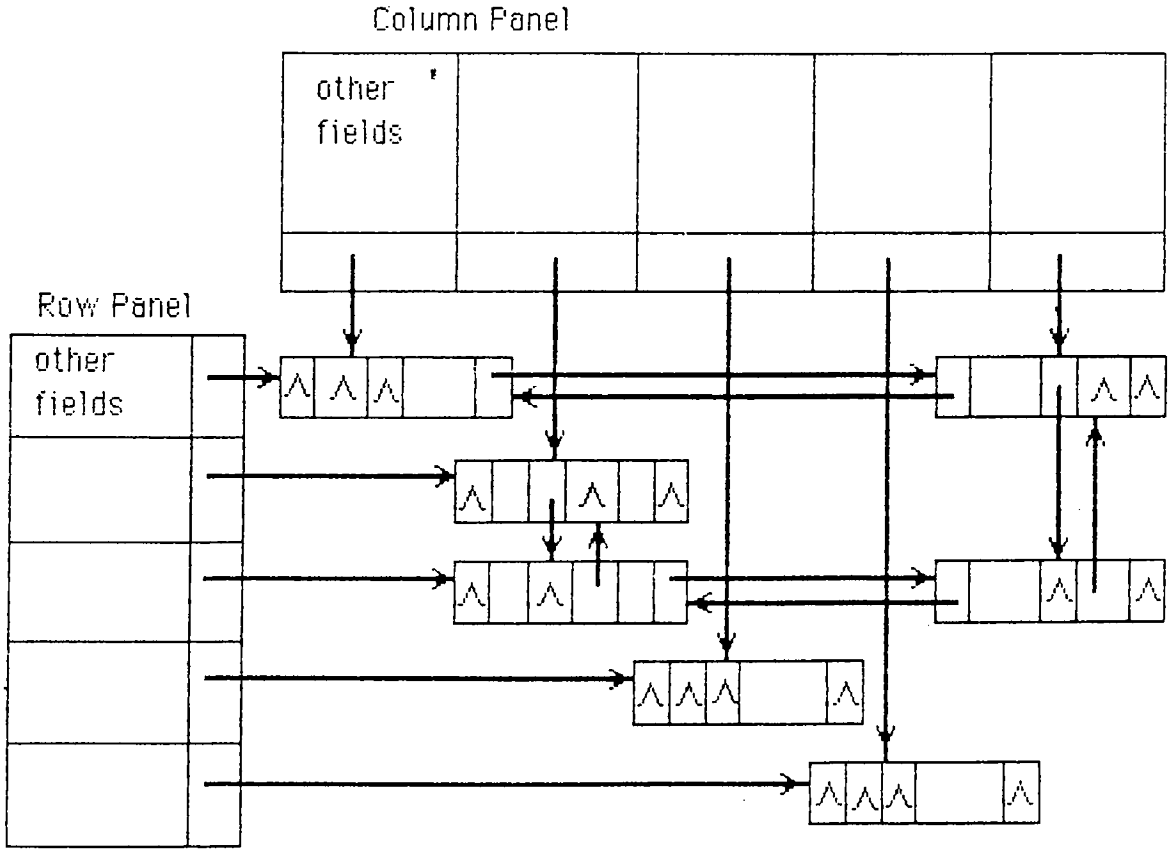


Figure 3.2: A possible two dimensional data structure for the M matrix

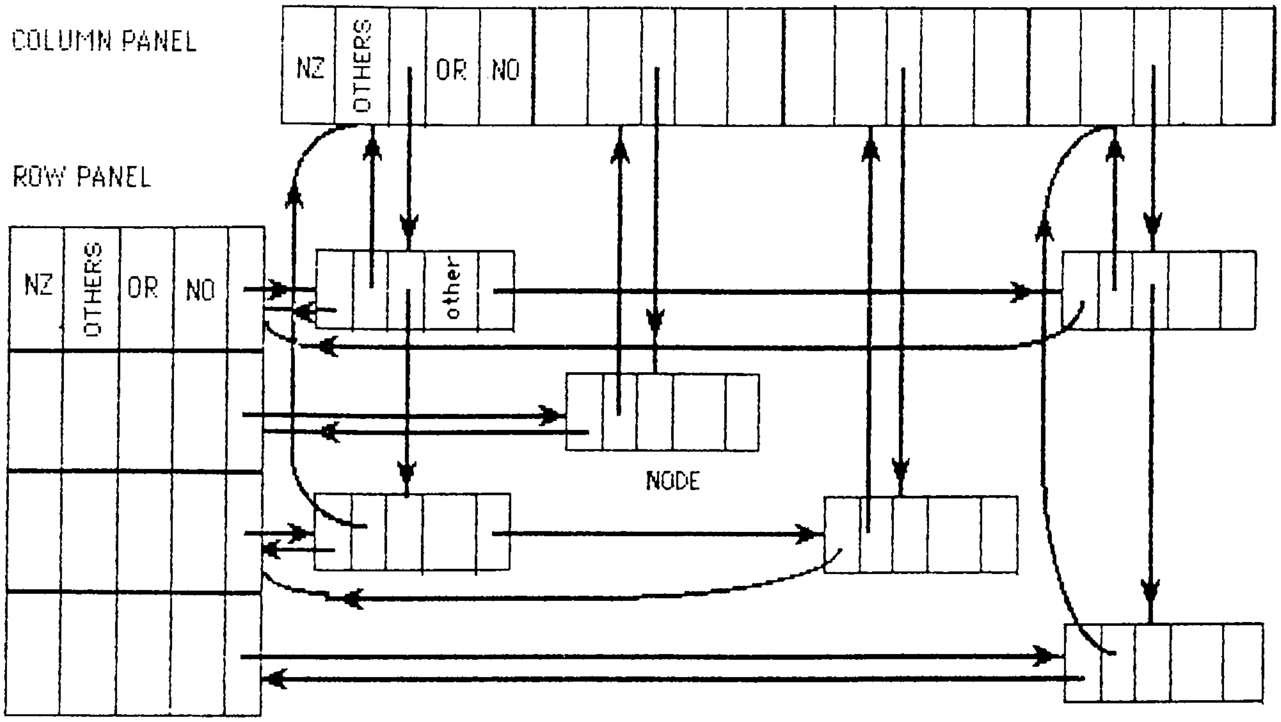


Figure 3.3: Representation of a 4 × 4 sparse matrix having only 6 elements, according to the structure we used.

arrays of structures. Each element of row or column panel has five fields. They are shown in detail in Fig. 3.4, and their function is described below.

or : It is the original row or column number which remains unchanged throughout the program.

no : It indicates the order of selection of a row or column as the pivot row or column.

begin : This is a pointer field pointing to the first element of the row or column.

max : This pointer field points to the largest element in magnitude of the row or column.

NZ : The number of nonzero elements in a row is known as NZUR and the NZ field of a row stores the NZUR of that row. Whereas, the number of nonzero elements in a column is known as NZLC and the NZ field of a column panel stores it.

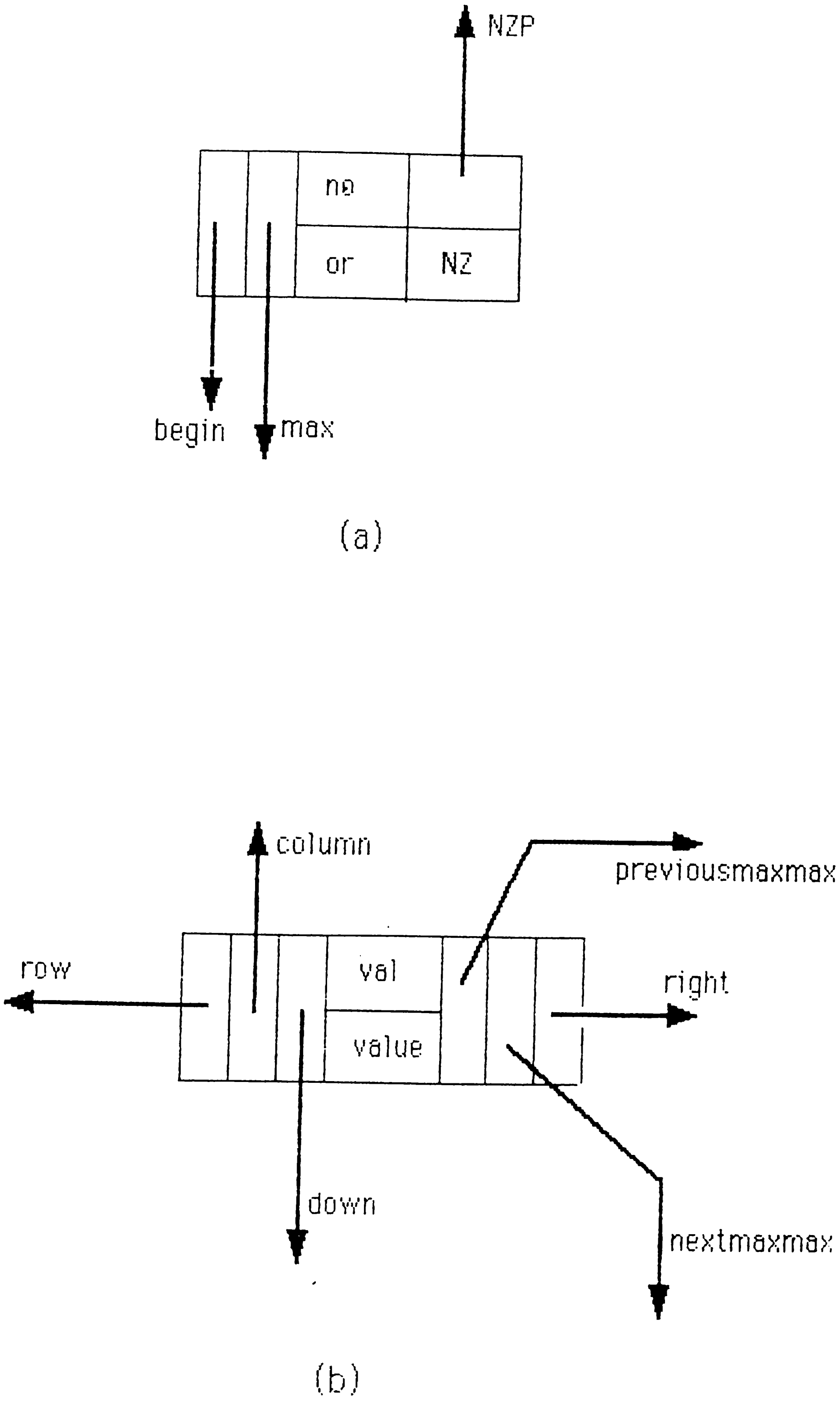


Figure 3.4: Structure of (a) a panel, and (b) a node

On the other hand, the nonzero elements of the matrix are stored in "nodes" which are connected both to the panels and to each other. The fields of a node are illustrated in Fig. 3.4.b and are described below.

row panel pointer : This is a pointer field pointing to the specific row in the row panel which it belongs.

column panel pointer : This is a pointer field pointing to the specific column in the column panel which it belongs.

right : This pointer points to the node on the right, if there is any.

down : This pointer points to the node below, if there is any.

val, value : They contain the value of the nonzero elements and they are described in section 3.3.3 in more detail.

nextmaxmax, previousmaxmax : These fields are used to link the pivot candidates to each other, and also to mark the elements which are not pivot candidates.

In order to solve a matrix equation using LU decomposition we must perform the following operations n times where n is the order of the M matrix.

1. Pivot selection
2. Row and column interchange
3. Normalizing the pivot row and zeroing the elements under pivot
4. Updatings

And then FBS is performed as many times as required by the simulator.

We are going to calculate the time complexity of the above items for all of the structures mentioned above, then show that the structure we have used performs best.

1. **Pivot Selection** : The complexity of the calculations made for pivot selection highly depends on the pivot selection algorithm. But it is easily seen that the data structure we have used eases the work of pivot selection for any algorithm.

2. **Row and Column Interchange :** As soon as a pivot is selected we have to interchange the pivot row and column with the i'th row and column, where i is the order the pivot is selected. In the data structure we have used, a row or column interchange has time complexity of $O(1)$ as there is no physical row or column interchange in this structure. Whenever an element is chosen as the pivot, the *no* field of the row panel of that element is numbered in the order it is selected. For example, if an element say, of the fourth row, is chosen as the pivot at the beginning; instead of interchanging the first and the fourth row, the number field of the fourth row-panel is set to "1". The column interchange is also performed in a similar way. In fact this numbering operation is used for marking a row or column when it is selected, so that in the later operations they are skipped and they are not subjected to any further execution. Although we do not perform any physical interchange, the algorithm behaves as if the interchanges are done physically.

   But this is not the case with the data structures in Fig. 3.1 and Fig. 3.2. The first structure has $O(1)$ time complexity for row interchange, whereas the time complexity of column interchange is $O(nk)$, where k is the average number of nonzero elements in a row or column, and n is the order of the matrix. Although k is usually a small number, the performance gets worse as n increases. For the second structure it can be easily shown that the time complexity of both the row and column interchange is $O(k^2)$.

3. **Normalizing the pivot row and zeroing the elements under the pivot :** In fact the factorization can be done without the other items but this item is the fundamental job during the LU decomposition. There are two major things to consider in this part. They are

   Floating point operations (flops): Operations like normalizing the pivot row and multiplications and subtractions in the subsequent rows involve flops which can not be avoided. The computation time to perform floating point operations are the same for all the structures, as long as the same algorithm is employed in the pivot selection. It can be shown that the time complexity is $O(nk^2)$. These operations are generally more time consuming than the non-floating point operations which are mentioned below.

   Non-floating point operations : Operations like visiting a node and checking if it is marked or not are non-floating point operations. These operations must be performed in addition to flops to LU factorize the matrix and they vary from structure to structure.

4. **Updatings** : In order to make a proper selection of pivots we have to store certain information about the matrix such as NZUR's of the rows. This information may change at any step of the LU factorization and hence must be updated whenever undergoes a change. These updatings are highly dependent on the data structure, and the algorithm employed. For the data structure we have employed, the updating procedures will be explained later.

3.2.3 Methods Generally Used to Increase the Efficiency

The methods generally used to achieve efficiency are as follows:

Numerical Pivoting

The element by which a row is normalized is known as the pivot element. The selection of the pivot regarding the numerical values of the elements in the matrix is called numerical pivoting. In order not to lose much from accuracy, several type of pivoting strategies can be employed. The most widely used of these strategies are partial pivoting and complete pivoting which are well known pivoting strategies and can be found in any elementary book on numerical analysis [12, 11]. Hence, they are not discussed here.

Minimization of Number of *Fill-ins*

Let's suppose a_{ii} is chosen as the pivot, and the pivot row is normalized. While zeroing the pivot column, a_{ji} is multiplied with a_{ik} and subtracted from a_{jk}, when $j > i$ and $k > i$. If a_{jk} was 0 previously, it was not stored in the memory. But now a new non-zero value has occurred at a_{jk}; this is called a *fill-in*. A fill-in causes the following problems.

1. A new memory location to be allocated for it which is not very desirable.
2. An increase in the time complexity, due to a few extra non-flops.
3. A drop in the accuracy because of an increase in flops.

4. New fill-ins , which is the most important of the problems, as these new fill-ins, in turn, cause the trouble mentioned in above items and a host of new fill-ins.

So it is clear from the problems mentioned above that we should try to minimize the number of fill-ins. An estimate on the number of possible fill-ins a pivot can cause is $(NZUR-1)\times(NZLC-1)$ of pivot row and pivot column, as this is the total number of multiplications that should be performed. These products are then subtracted from a_{jk}, which because of the sparsity of the matrix has a great probability of being zero.

Hence, $(NZUR-1)\times(NZLC-1)$ is a proper estimate on the possible number of fill-in's a pivot will create. So, the smaller the product, the lesser the number of fill-ins. As can be seen, if either NZUR or NZLC is 1, there is no fill-in. More over the elements in the pivot row(column) are not included in the calculations anymore, which causes a decrease in NZUR(NZLC). All these are very desirable. So it is a good strategy to select the pivot from a row or column whose NZ field is 1, even though the element is not large in magnitude.

3.2.4 Our Strategy for Pivot Selection

Our pivot selecting strategy is a combination of classical numerical pivoting strategies and minimum fill-in strategy. The algorithm used for LU decomposition is described below.

1. A pivot must be the maximum element of both its row and column, in our jargon, we say a pivot must be *maxmax* (this strict rule is modified in *multimaxmax* strategy which will be described later). Note that there is at least one maxmax element : the maximum element in the matrix. So there is always at least one pivot candidate which is necessary for the continuation of the job.
2. The elements satisfying the above condition are found, and then among these pivot candidates the one which has a minimum value of the product $(NZUR-1)\times(NZLC-1)$ is selected to be the pivot.
3. Floating point operations are performed according to the selected pivot above. Then we discard the elements in the pivot row and column from the further calculations.

4. Because of the operations and discardings there will be changes among the pivot candidates selected according to the first rule. Therefore we must handle these changes by updating maxmax'es.
5. Because of the discardings in "3", there will be changes in NZUR and NZLC's of some rows and columns. We have to register these changes and update the list which is composed of maxmax elements sorted in the ascending order of the products $(NZUR - 1) \times (NZLC - 1)$.

3.3 Some of the Tricks Used in the Software

3.3.1 Memory Allocation

The package uses dynamic memory allocation, because of the large variety of the input matrices in dimension and sparsity. The memory allocation performed by the system may be very slow if it is not used judiciously. For example, for a specific type of data structure of size m, allocating memory space to n variables of this structure type separately, that is using malloc(m) n times, is much slower than allocating them altogether by malloc($n * m$). For our M matrix, we know the number of nonzero elements from the very beginning. Hence, we can utilize this situation by allocating all the required memory space right at the beginning. In this subsection we will mention some of the procedures which use tricks like this in the memory allocation.

Duplicate : At the beginning of the program, the M matrix is set up in the memory. Since we will have to deal with the matrix for a lot of times, we duplicate the matrix to a different memory location and perform the LU factorization on this copy of the matrix. In the very beginning of the program chunks of memory of equal size are allocated for both the original and the copy of the matrix. However the actual size of memory allocated to each of them is a bit larger than the number of nonzero elements in the matrix, which we call the "tolerance". The tolerance is introduced to take care of the changes in the stencils of the original matrix which may cause an increase in the number of nonzero elements in the matrix, requiring extra memory space.

After the initial set up of the original matrix, memory spaces are allocated for copies of the list-header and panels in a similar way. The only remaining task to be done at this stage of duplication is to copy the web of pointers of the original setup to the duplicate copy. The original setup is full of pointers as was shown if Fig. 3.3, Fig. 3.4 and Fig. 3.5. So if not done cleverly, it can take a long time to duplicate the pointers to their respective position in the duplicate copy. The strategy we have used to copy the pointers which are in fact addresses of memory locations, resembles the method of relative addressing. First of all, the offset, M->beginning-of-nodes – $Mcopy$->beginning-of-nodes, is calculated. Then, this offset is added to the memory address written in the pointer field of the nodes of the original setup and copied to the respective pointer fields of the duplicate. This works if all nodes of the original setup and the duplicate setup are allocated in the same page and hence has a constant relative address difference between them. The same relative copying method is used to copy the panel pointers.

stalloc (allocation for stencils) : In order to use the advantage of the information that all of the nodes use same amount of memory, the memory allocation is handled by the program. If a memory space for a new node is requested, then since we know how much memory is necessary, there is no need to call a system function. Instead the package routines uses stalloc or falloc.

A few elements of the M matrix can change values or even disappear and a few new elements can appear whenever the stencils of the matrix is changed at the beginning of each LU factorization in a particular simulation. The function *stfree* pushes the address of the freed nodes into a stack and the function *stalloc* allocates memory space for a new element by popping an empty node address from the stack. The tolerance nodes which are allocated at the very beginning of the of the setup are also pushed into the stack.

If, in case, the stack happens to be empty, i.e. there is no free space left, we allocate a new space of memory which has STACKPAGE number of more space than previous one and duplicate all the elements $(NZUR - 1) \times (NZLC - 1)$ to the new one. Now, we have STACKPAGE amount of new free space.

falloc (fill-in allocation) : During LU factorization of $Mcopy$, we will need new space for fill-in nodes. *falloc* is used in order to provide these new nodes. When the first fill-in occurs, a page of size FILL-IN-PAGE is created, using the system's allocation command. The pointer indicating the beginning of the page is then pushed into a stack. The page is then used to store subsequent fill-ins until if is full whence a new page is created and the pointer showing its beginning address is pushed into the stack.

When a stencil or stencils of *Mcopy* is altered by some other routine of the simulator, the *Mcopy* is LU factorized from the beginning. Hence the data in the "FILL-IN-PAGE"s become unnecessary and are virtually freed by the procedure *ffree*. What actually happens is instead of being freed physically, they are overwritten as new fill-ins occur. This technique serves our purpose better as nearly the same number of fill-ins occur for the handful of stencil changes. This procedure saves us the time of freeing memory spaces and reallocating them.

3.3.2 Listheaders and Multiplication Table

In order to implement the efficiency increasing strategies explained in section 3.2.4, a separate data structure is constructed, Fig. 3.5, in addition to the structure which stores the nonzero elements of the matrix. A part of this structure is *listheaders* which in fact, is an array of nodes whose nextmaxmax fields point to possible pivot candidates (maxmax'es) in an ordered manner. The first element of the array points to maxmax'es whose $(NZUR-1)\times(NZLC-1) = 0$, the second element points to maxmax'es whose $(NZUR-1)\times(NZLC-1) = 1$ and so on.

The listheader array facilitates the task of selecting a pivot quite elegantly. We always select the pivot such that it is the maxmax pointed by the non-NULL nextmaxmax pointer of the topmost element of the listheader array. It should be mentioned that with a high probability the nextmaxmax pointers of the upper listheaders will point at some maxmax'es after the updatings have been performed at each operation. Using the data structure, Addmaxmax, which inserts a maxmax to the list pointed by the proper listheader, is a very fast and simple function.

The two way linked list structure if the maxmax elements provides simplicity at the updatings. For example a deletemaxmax operation consists of only two pointer changes. The nextmaxmax fields of the non-maxmax elements are equated to a specific address, called ABYSS, so that maxmax elements can be easily identified.

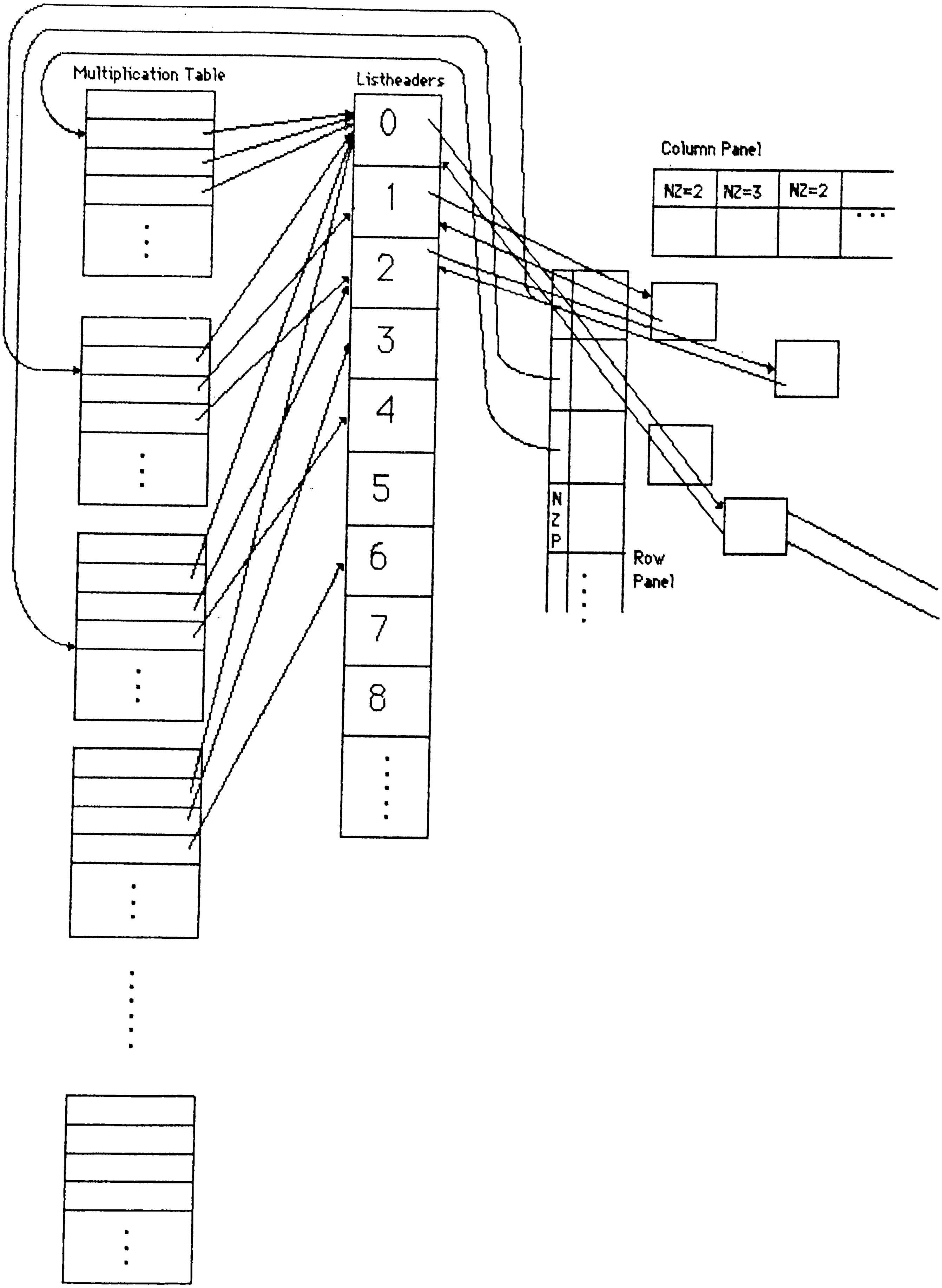


Figure 3.5: The structure of Listheaders and Multiplication Table

3.3.3 Representing the Values with Two Fields

In the sparse tableau analysis, we know that most of the nonzero elements are 1's or -1's. There is no need for a floating point multiplication with these elements. the result is obvious, the number which is multiplied with these elements either stay the same or change sign. In order to speed up using this property we use two fields to store a value. If the value is 1 or -1 we store it in the integer field, *val*. Otherwise we put a zero into the *val* field and the number is stored in the double (floating point) field, *value*.

3.3.4 Multiple Pivot Candidates

We introduce multiple pivot candidates strategy in order to reduce the bad effects of the strict rule, "a pivot must be a maxmax". Because of this strict rule we may be burdened with a lot of unnecessary fill-ins. For example, an element of numeric value 10, may be selected as a pivot candidate although an element of numeric value 9.5 at the same row has a lower $(NZUR - 1) \times (NZLC - 1)$. In order to prevent such situations, we can allow more than one element in a row or column, to be pivot candidates, which we call *multimaxmax* strategy. We can normalize every row and afterwards every column, with the maximum element of that row/column, then defining a threshold, select the elements greater then this threshold as pivot candidates. Another way to do this is to define a fast function, may be a function of the power bits of the floating point number, which is a rough measure of the magnitude and then assume all of the elements which has the maximum rough magnitude to be pivot candidates.

Chapter 4

Asymptotic Waveform Evaluation

Asymptotic Waveform Evaluation (AWE) is a recently proposed technique for the approximate pole zero representation of linear time invariant circuits [2] [3]. It is, in fact, a form of Padé approximation [16]. AWE uses the differential state equations

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{4.1}$$

to find an approximation for the state variables. We know that, the homogeneous solution of 4.1 is of the form

$$x_i(t) = \sum_{l=1}^{q} k_l e^{p_l t} \tag{4.2}$$

where q is the order of the circuit. k_l and p_l are the residues and poles respectively.

AWE finds an approximation to 4.2 such that

$$x_i(t) = \sum_{l=1}^{q'} k'_l e^{p'_l t} \tag{4.3}$$

where q' is the order of approximation which is smaller than q (in most cases $q' \ll q$), and k'_l and p'_l are the dominant approximate residues and poles respectively. They are calculated from the moments of the circuit. An important restriction with AWE is that it may produce unstable poles even though the circuit is stable, which is also a major problem for Padé approximation [16][15]. This problem is overcome by combining differential and integral moments.

The computation of derivative and integral moments, and the calculation of poles and residues using the combination of derivative and integral moments is described in [1].

4.1 Using the Combination of Derivative and Integral Moments to Obtain Stable Approximations

Using AWE, we may find unstable approximations for stable circuits. This is because we are trying to approximate a higher order system with a lower order one. And the moments may give inconsistent information for that lower order system. For example, assume that the original waveform has a negative initial condition and goes to zero at steady state after a large positive overshoot (i.e., the area under the original response is positive). If we use the integral moment for matching, the first order approximation can not find a stable solution. But if we use the first derivative moment approximation or a second order approximation (derivative or integral or a combination of both), we can find a stable approximation. We can also conclude with right half-plane (RHP) poles using only derivative moments. So it is a good idea to use an appropriate combination of integral and derivative moments for the calculation of poles and residues of every state variable independently. Here is the algorithm used for this purpose.

- Necessary moments are computed according to the order of approximation.
- For every state variable DO

```
(i) Compute the poles using the proper moments (initially start from
    all integral moments if the user does not redefine this parameter).
(ii) If there is any right-hand-plane pole then
       If there are no integral moments used then
         increase the order of approximation by 1,
       else
         replace the highest order integral moment with the next
         derivative moment, and go to (i) to compute the poles.
(iii) Compute the residues.
```

So, the number of derivative moments used in the approximation is increased one by one until a stable approximation is found or all of the integral moments are replaced with the derivative ones. If a stable approximation can not be found then the order of approximation is increased by one. And we are trying to avoid large orders, due to numerical inaccuracy and increased number of operations which requires long time for the approximation. But we have observed for a large number of examples that a stable and good approximation can be found before the 5'th order. If a stable approximation can not be found up to a certain order, (which never occurred for all the examples we tried) the order of approximation is not increased anymore, but the first derivative is used to approximate the response (Forward Euler) to shift in time. After that a new AWE is made with different initial conditions. Note that the dominant approximate poles and residues depend on the initial conditions. Note that, using this algorithm, the order of approximation and the number of derivative moments used in the approximation may not be the same for different state variables. And it is not necessary to approximate all of the states with the same order for transient analysis as far as you find a good approximation for that state variable.

4.2 Transient Analysis Using AWE

In the beginning of the transient analysis, a dc analysis is performed in order to find the operating points. For this dc analysis the capacitors with user defined initial values are replaced with voltage sources of the same value, while the other capacitors are assumed to be open circuits. Similar things are performed for the inductors. Another dc analysis follows in order to find the steady state values. Then we can approximate the state variables with asymptotic waveform evaluation technique, using the initial conditions, steady state values and the linearized circuit itself.

Using AWE we obtain approximate analytic expressions for capacitor voltages and inductor currents. These expressions are valid on the time axis as long as PWL elements satisfy the boundaries of the set of current operating regions, R_i. In order to find voltages and currents of each device, these expressions are evaluated at certain time instants and using these values as sources, the circuit is solved by a mere substitution (FBS). As we progress over time with steps the nonlinear devices in the circuit may change their segments. If this occurs, we must know the time when one piece-wise linear device, at least, changes its segment. As soon as we realize a segment change, we go back over time and search for the time of segment change. The capacitor voltages and inductor currents at the time instant of segment change, are the initial conditions for the next AWE. The same thing happens when there is an input change at time t_0. We evaluate the approximate expressions found for energy storage elements and solve the circuit at time t_0^- by a mere substitution. A new DC analysis is done at time t_0^+ using the new source vector, and a new AWE is performed for $t > t_0^-$. For DC analysis, we can use the previous solution and segments (instead of **0** vector) as initial valid solution. This saves a lot of computation.

The selection of the time step in transient analysis is a quite critical issue for the time efficiency standpoint. If the time step is chosen too small, then too many unnecessary computations must be performed. This may even cause the simulation not to terminate in a reasonable time. Conversely too large time steps may cause large errors if there exist high frequency poles with large residues. Another drawback of the large time step is that we may skip an overshoot of the waveform which may possibly cause a segment change. Therefore, the time step used in transient analysis is dynamically calculated after each FBS. In this calculation, we consider primarily the rate of change of the most rapidly changing exponential. There are some parameters used in the calculation of the time step. For example the calculated time step is divided by the SAFETY factor which can be set in the .OPTIONS card. TSLOOPLIMIT is another parameter which is useful if the internal time step turns out to be too small. In a PWL circuit if there is no segment change over a TSLOOPLIMIT time step period, then the time step is multiplied with TSMULT at every time step until a segment change occurs. This causes an exponential increase of the internal time step and prevents an infinite loop due to an error in the calculation of the time step. BUSTLE has some other parameters for the transient analysis which will not be mentioned.

As a result of dynamic selection of the time step, the simulator spends more effort when there are rapid voltage or current changes, and skips quickly in the time axis if there are slow changes. This provides an event driven feature to the simulator.

Chapter 5

RESULTS

In this chapter, some simulation results of BUSTLE will be presented. Some other results can be found in [1]. The results and computation times of BUSTLE is compared with those of SPICE 2G.6. All of the measurements are made at SUN 3/110 workstations. As it is mentioned before, BUSTLE leaves the accuracy speed trade-off to the user by giving him/her a number of options. The minimum order of approximation, the *minorder*, which can be given a value in the .OPTIONS card, determines the number of moments matched in AWE. This parameter is an important parameter for the accuracy of the approximation. For example, it can be selected as 1 for a digital CMOS circuit, but this would not be sufficient for an RLC circuit which has an oscillatory response. The minimum order and also the number of derivatives that will be matched initially can be determined by the user. There are also some other parameters that can be set by the user to improve the accuracy or the speed. Another important feature is that, user can define his own models (or use one from the library) for nonlinear devices. This provides a capability to keep pace with the emerging technology, also user can control the accuracy speed trade-off by choosing the number of segments used for modeling.

EXAMPLES

1) The first example circuit, Fig. 5.1, which is taken from [3], demonstrates the usage of derivative and integral moments together, to get stable approximations. Using basic AWE method [2], we end up with RHP poles for C2 and L3 for a second order approximation, which is not mentioned in [3].

However, using the method described in section 4.1, we can find stable approximations for all of the state variables. As it can be seen from the Table 5.1, after using the first derivative moment, a stable approximation can be found for

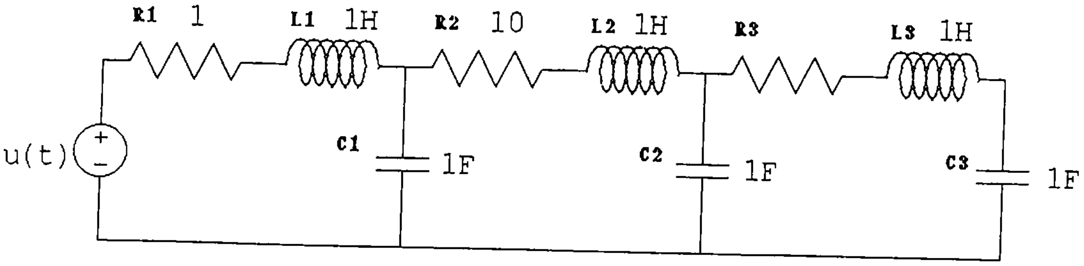


Figure 5.1: Pillage's 6th order RLC circuit

2.0	2.1	3.0	3.1
-1.206e-2	-2.015e-2	**5.053e−1**	-8.935e-3
3.012e−2	-1.653e+1	-1.269e-1	-2.803e+0
		-8.915e-3	-1.965e-1

4.0	5.0	Actual
-5.556e-1 + j8.965e-1	-1.029e-1	-5.556e-1 + j8.965e-1
-5.556e-1 - j8.965e-1	-1.330e-1	-5.556e-1 - j8.965e-1
-8.914e-3	-8.914e-3	-8.915e-3
-1.023e-1	-5.556e-1 + j8.965e-1	-1.029e-1
	-5.556e-1 - j8.965e-1	-9.797e+0
		-9.998e+1

Table 5.1: Approximate Poles for response at C2 and the actual poles of the circuit.

2.0	2.1	2.2	2.3	3.0
-1.022e-2	-4.301e-9	**0.000e+0**	**0.000e+0**	-8.914e-3
4.033e−2	**4.301e−9**	-5.404e+17	-1.001e-1	-7.975e-1
				-1.047e-1

Table 5.2: Trials of BUSTLE to find a second order approximation and conclusion with a third order approximation.

minimum order	computation time
1	1.52 sec.
2	1.72 sec.
3	1.80 sec.
4	2.00 sec.
5	3.13 sec.

Table 5.3: Total execution time list for different minorder requirements

C2, whereas for L3, we can not get rid of RHP poles using a second order approximation even all possible combinations of derivative and integral moments are tried. In this case the order of approximation is increased automatically, and a stable approximation in the 3'rd order (Table 5.2) is found. But the remaining states are approximated with second order which is the minimum required order. Note that a better approximation for L3 is performed which satisfies the accuracy requirements of the user.

The step responses of 3 different nodes of the circuit in Fig. 5.1, computed for different minorder requirements, can be seen in Fig. 5.2. The execution time list is given in Table 5.3. The computation time of BUSTLE is pretty good. Additionally, for this circuit, BUSTLE finds the results analytically in about 30% of the total execution times listed in the table, and 70% of the execution time passes during the evaluation of the exponentials at certain time instants and simple substitutions.

2) This example is a 200'th order RLC ladder circuit (Fig. 5.3). Since the circuit is too large, the whole of it is not drawn. But the circuit is a repetition of the ladder. All of the capacitors are $10\mu f$ and all of the inductors are 100mh. The initial voltages of all of the capacitors are given as 0, and the initial inductor currents increments by 1 ma after every 5 inductor, so that the initial currents of the first 5 inductors are 0 ma, the second 5 are 1 ma, and the last 5 are 19 ma. The voltage waveforms of the nodes 15, 101 and 201, are drawn in Fig. 5.4, for different minorder values. The timings and the *normalized rms differences* [1] from SPICE are listed in Table 5.4.

3) The third example, Fig. 5.5, is a full wave rectifier. The diodes are

---

[1]
$$Normalized\ rms\ difference = \sqrt{\frac{\int_{t_{start}}^{t_{stop}} (x_1(t) - x_2(t))^2\, dt}{V_{max}^2\,(t_{stop} - t_{start})}}$$

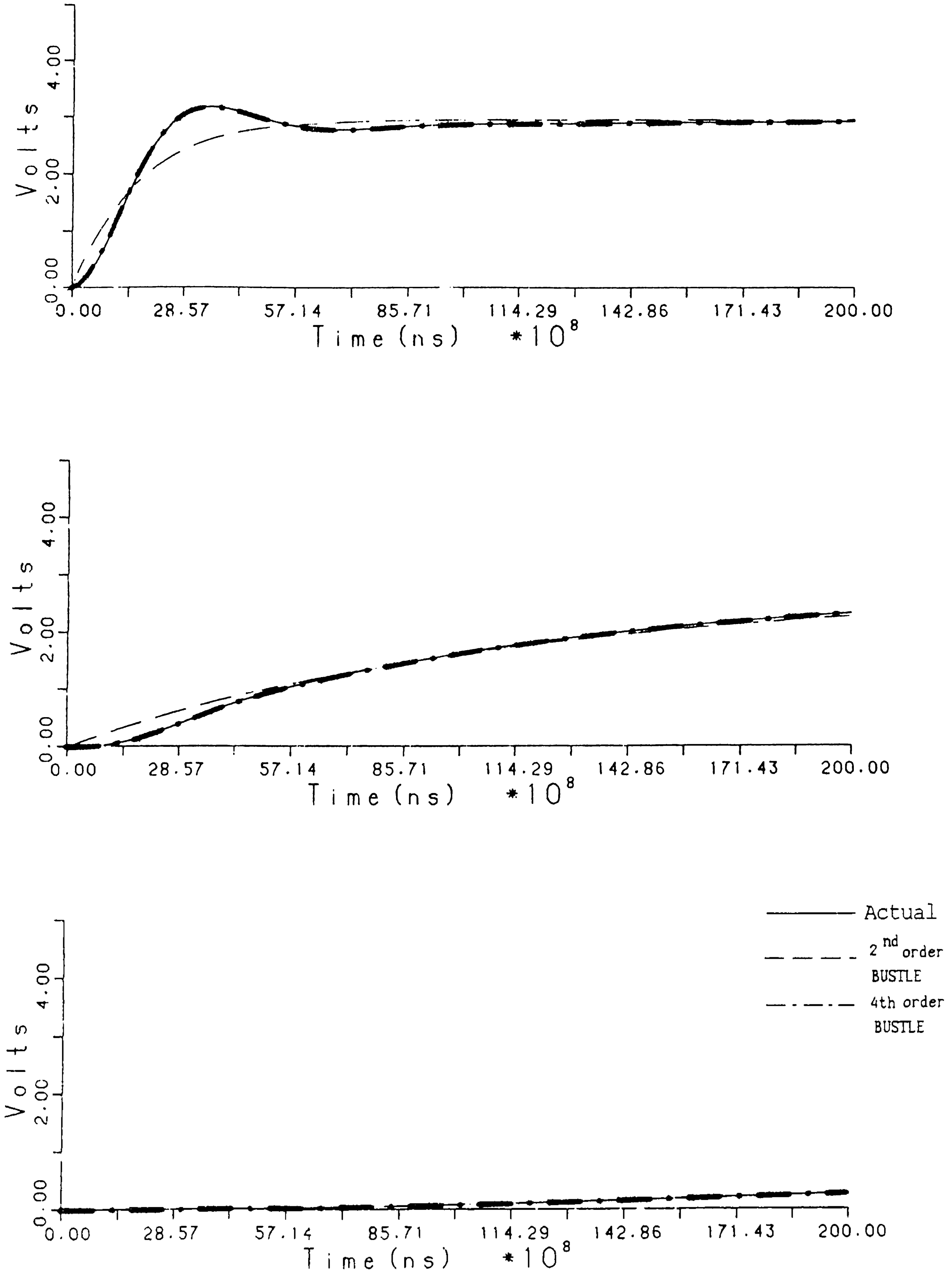


Figure 5.2: Output waveforms of the capacitor voltages, for the circuit in Fig. 5.1, computed using different minorder values.

minimum order	computation time	normalized rms. dif. with SPICE		
		v(15)	v(101)	v(201)
1	23.72 sec.	8.9%	4.8%	5.6%
2	27.40 sec.	4.5%	2.8%	4.7%
3	31.70 sec.	4.6%	1.5%	1.5%
4	41.03 sec.	3.8%	1.1%	0.6%
spice	174.30 sec.	-	-	-

Table 5.4: Total execution time list of the 200'th order RLC ladder for different minorder requirements, and the normalized rms difference from SPICE.

PROCEDURE NAME	NUMBER OF CALLS	CPU SECONDS	PERCENTAGE
INPUT READING	1	1.42	4.5
CAPLOOP	1	0.10	0.3
LU-C	1	0.07	0.2
SET UP	1	0.17	0.5
CONVERT MODELS	1	0.00	0.0
TRAN	1	29.92	94.4
FIND-DELTA-TIME	100	0.00	0.0
DC	2	1.97	6.2
SOLVE-MXB	3	2.73	8.6
LU	5	2.97	9.4
FBS	129	12.02	37.9
AWE	1	7.85	24.8
COMPLEXEQNSOLVER	417	3.27	10.3
ROOTFIND	0	0.00	0.0
FIND-2-ROOTS	0	0.00	0.0
FIND-3-ROOTS	217	0.50	1.6
FIND-4-ROOTS	0	0.00	0.0
TOTAL	1	31.70	100.0

Table 5.5: The CPU time for some major functions of BUSTLE, measured in the simulation of the 200'th order RLC circuit (minorder=3).

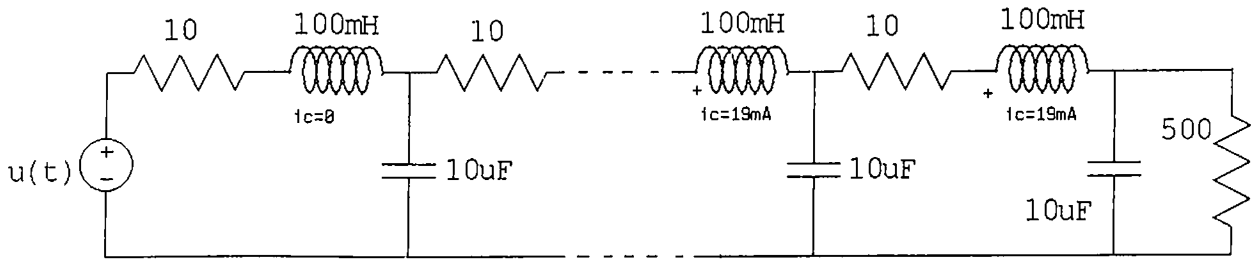


Figure 5.3: An 200'th order RLC ladder circuit

Simulator	computation time	normalized rms. dif. with SPICE (1000pt)
BUSTLE	4.60 sec.	1.4%
SPICE (100pt)	8.05 sec.	6.1%
SPICE (1000pt)	15.11 sec.	-

Table 5.6: Total execution time list of the full-wave rectifier circuit, and the normalized rms differences with 1000pt SPICE waveform.

modeled with two segments, one representing the OFF region, where the other is the ON region with $V_0 = 0.7v$. Transient analysis is performed with SPICE and BUSTLE with a square wave input. The voltage waveform on the load can be seen in Fig. 5.6. SPICE1 is the SPICE simulation using 100 time steps in transient analysis where SPICE2 uses 1000 time steps. It is surprising that the two waveforms are different. Transient analysis, using 100 and 1000 points, are also performed with BUSTLE, but the results do not differ if the time step is changed since the response is known analytically, and evaluated at time steps. The result of BUSTLE is very close to SPICE2, as it is seen from Fig. 5.6. The *normalized rms of the difference* between BUSTLE and SPICE2 is 1.4% where it is 6.1% between SPICE1 and SPICE2. As far as the execution times are concerned, it can be easily noticed in Table 5.6 that BUSTLE is again faster than both of the SPICE simulations. Consequently the selection of time step does not effect the simulation result of BUSTLE, but significantly changes the results of SPICE. We also made simulations of this circuit using a diode model with 12 segments that is extracted from SPICE. But the results of both simulations are almost identical, the normalized rms difference between the two simulations is 0.3%. Obviously two segments are good enough to model a diode, there is no need to use a more complex model.

4) The fourth example, Fig. 5.7, is a CMOS NOR gate with 4 MOSFETS, two NMOS and two PMOS. Both type of transistors are modeled simply with

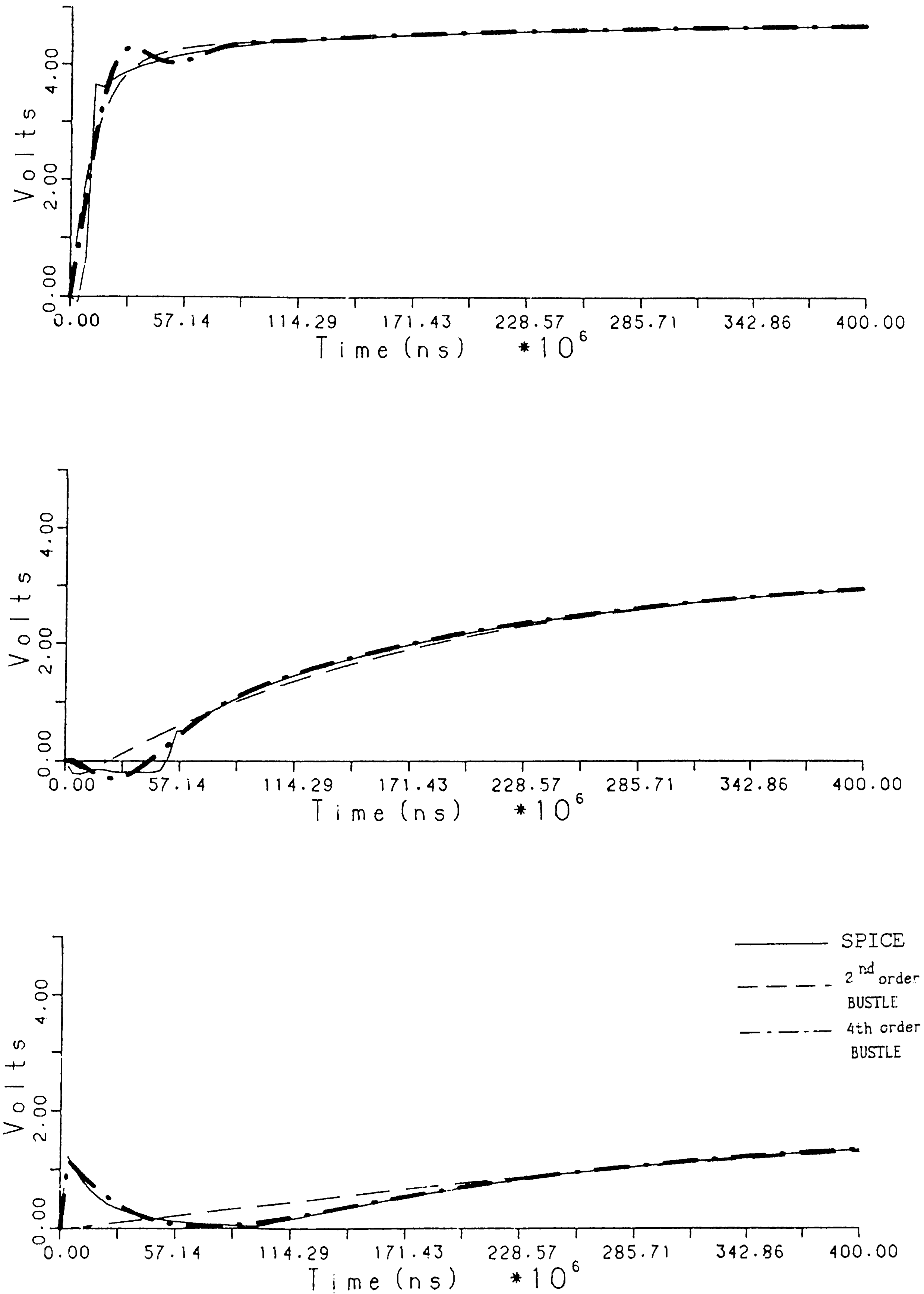


Figure 5.4: Output waveforms of the node voltages 15, 101 and 201(the voltage on the 500 ohm resistor), for the 200'th order RLC circuit, computed using BUSTLE with different minorder values, and SPICE

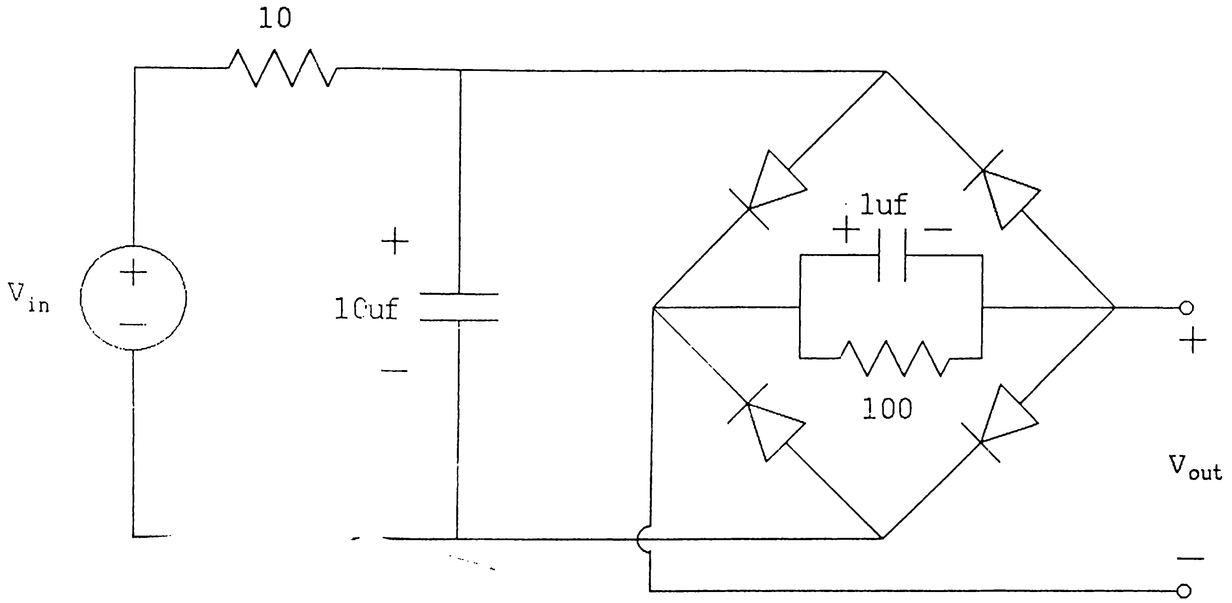


Figure 5.5: Full-wave rectifier circuit

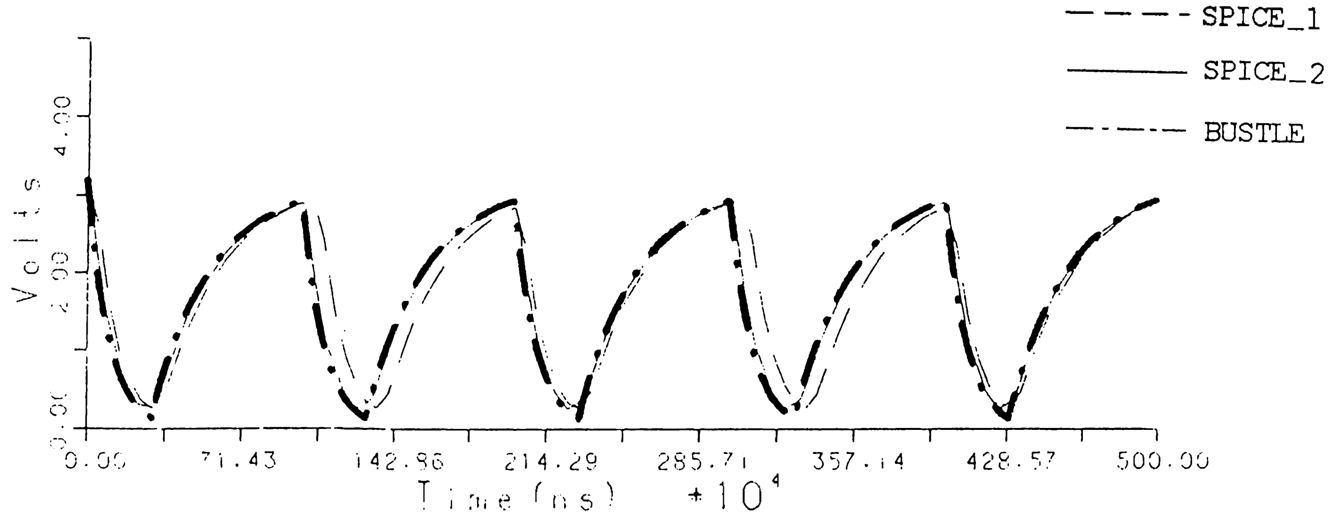


Figure 5.6: Output waveforms of BUSTLE and SPICE for the full-wave rectifier circuit

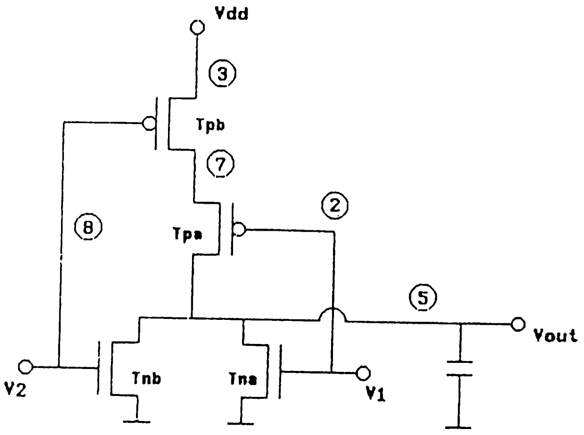


Figure 5.7: CMOS NOR circuit

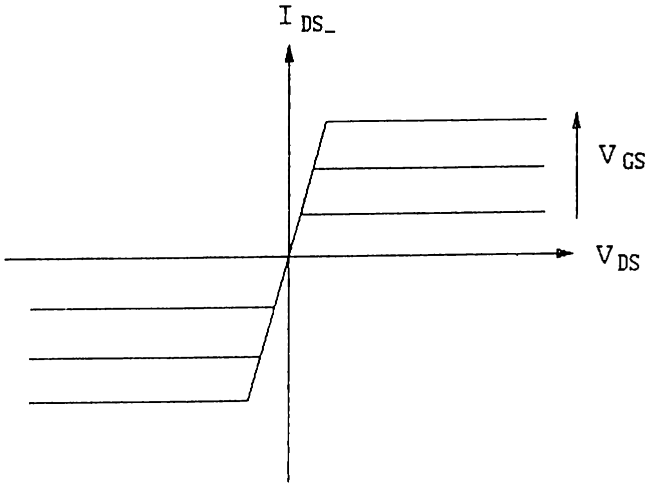


Figure 5.8: Piece-wise Linear NMOS model used, in the simulation of the CMOS NOR gate

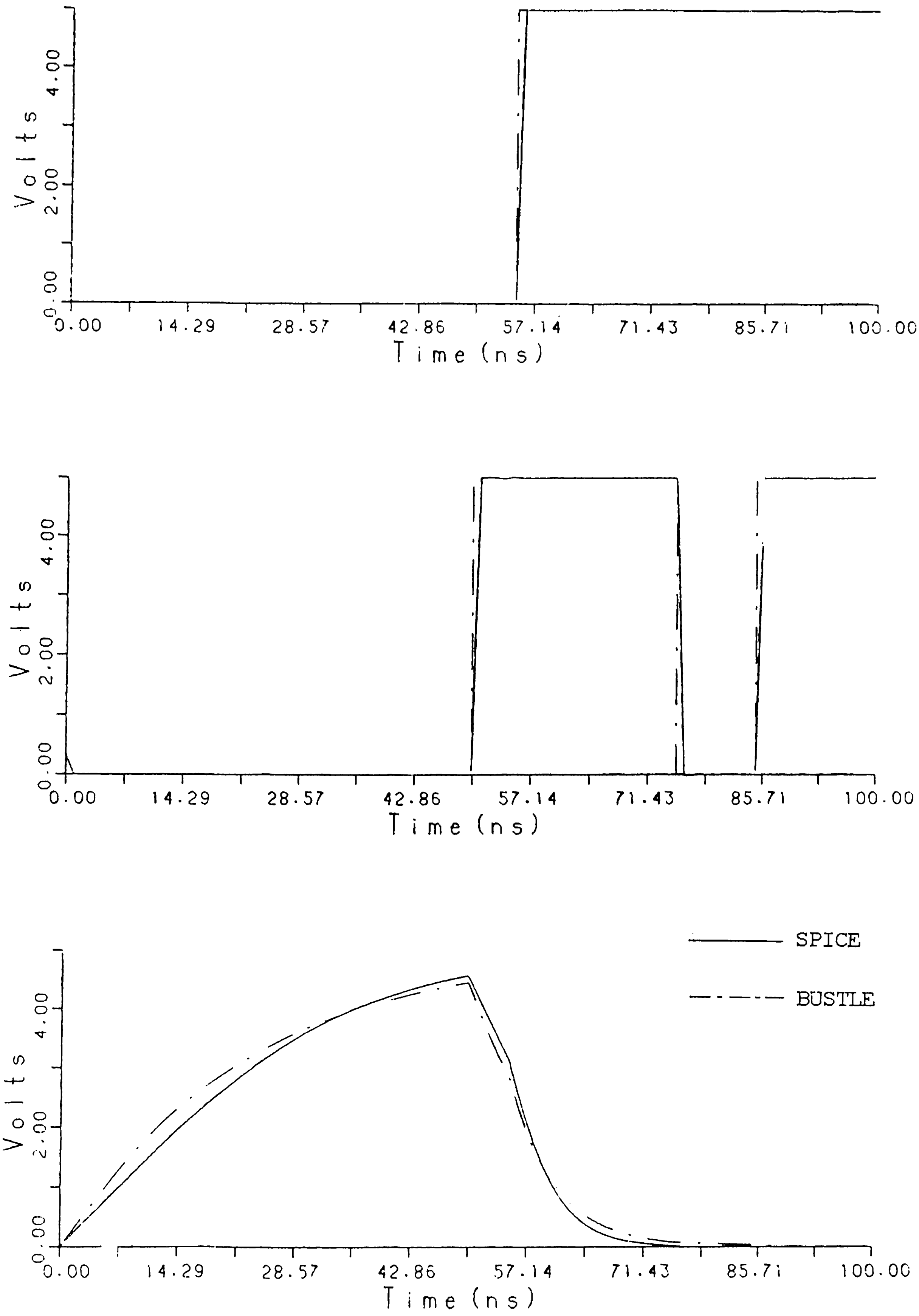


Figure 5.9: The output waveform of the CMOS NOR gate

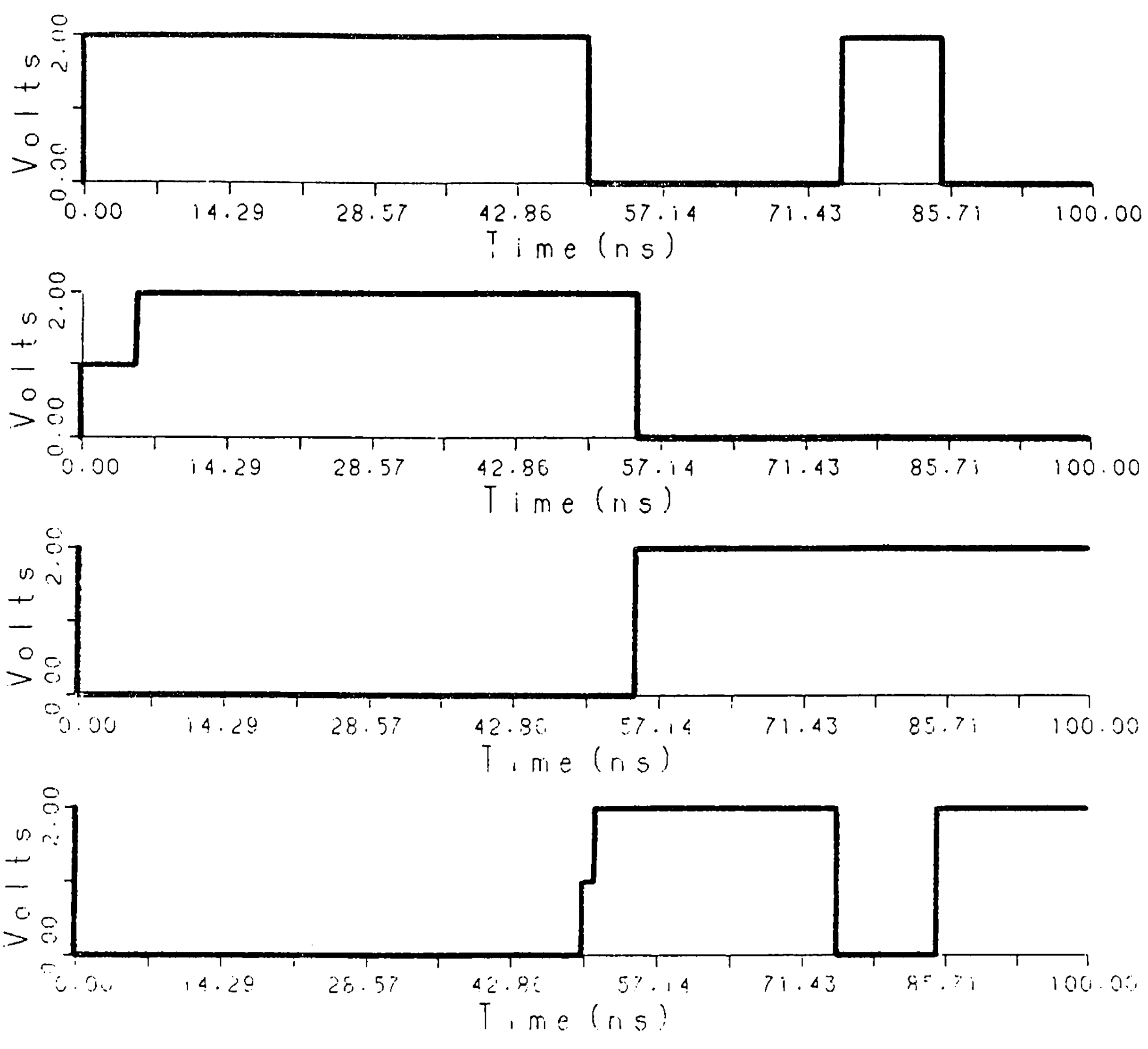


Figure 5.10: The operating regions of MOSFETs in the simulation of the CMOS NOR gate

4 PWL regions, cutoff, linear, saturation and reverse saturation, Fig. 5.8. The NOR gate is loaded with a 2pf capacitor. A transient analysis is performed with BUSTLE and SPICE. The first two graphs in Fig. 5.9 are the inputs. The difference between the inputs is not an error due to BUSTLE. It is a result of the algorithm employed by SPICE which can not change an input between two time steps. This causes a normalized rms difference of 7% for the first and 12% for the second waveform.

There is little difference between the two output waveforms, although the charging and discharging of the capacitor is very slow which is a disadvantage for simple modeling. The normalized rms difference is 3.3%. The execution time is again shorter than that of the both SPICE results.

A new facility of BUSTLE is that user can examine the operating segments of PWL devices, which he would like to observe, easily by adding a "seg" command in the print card. The operating segments of the MOSFETS of the NOR gate can be seen in Fig. 5.9. The 0 level denotes that the transistor is in the CUT OFF region, and level 1 and 2 are for SATURATION and LINEAR regions respectively. If we examine Fig. 5.10 we can see that at $t = 0^-$ PMOS'es are in CUT-OFF, NMOS'es are in LINEAR. At $t = 0^+$ all of the transistors change their operating regions : Tpa to LINEAR, Tpb to SATURATION and NMOS'es to CUT-OFF. Then at $t =$ 5.5nsec. Tpb goes into LINEAR and this goes on.

This facility is a lot of help to the user in the analysis of a circuit, because it is easier to understand the PWL models which have been conventional for nonlinear elements. For example a designer generally thinks the diode as a device which is ON or OFF, instead of an exponential characteristic. We believe that BUSTLE is highly educational since the solution style is very similar to the manual solution style.

5) This example is composed of diodes and MOSFETS, as shown in Fig. 5.11. This is a simple Flip-Flop, i. e. a bi-stable multivibrator. At first Q (node 3) is high and Q' (node 4) is low. Then Q' is forced to be high, by charging C1 through D2. The circuit stays in this stable state for a while, then its again forced to changed its state by the other source-diode combination. Fig. 5.12 is the output of BUSTLE for the input file given in chapter 6. The plotting routine is designed for a SUN workstation, and can be called from BUSTLE using the PLOT card. One can easily examine the output waveforms using this plotting function. Moving the mouse and clicking the button, the time and voltage values of any point can be learned. The first and third waveforms

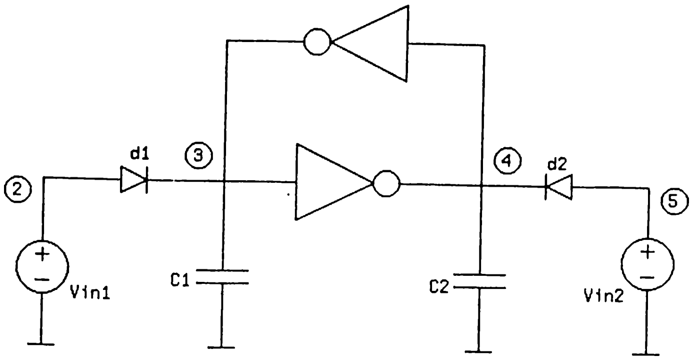
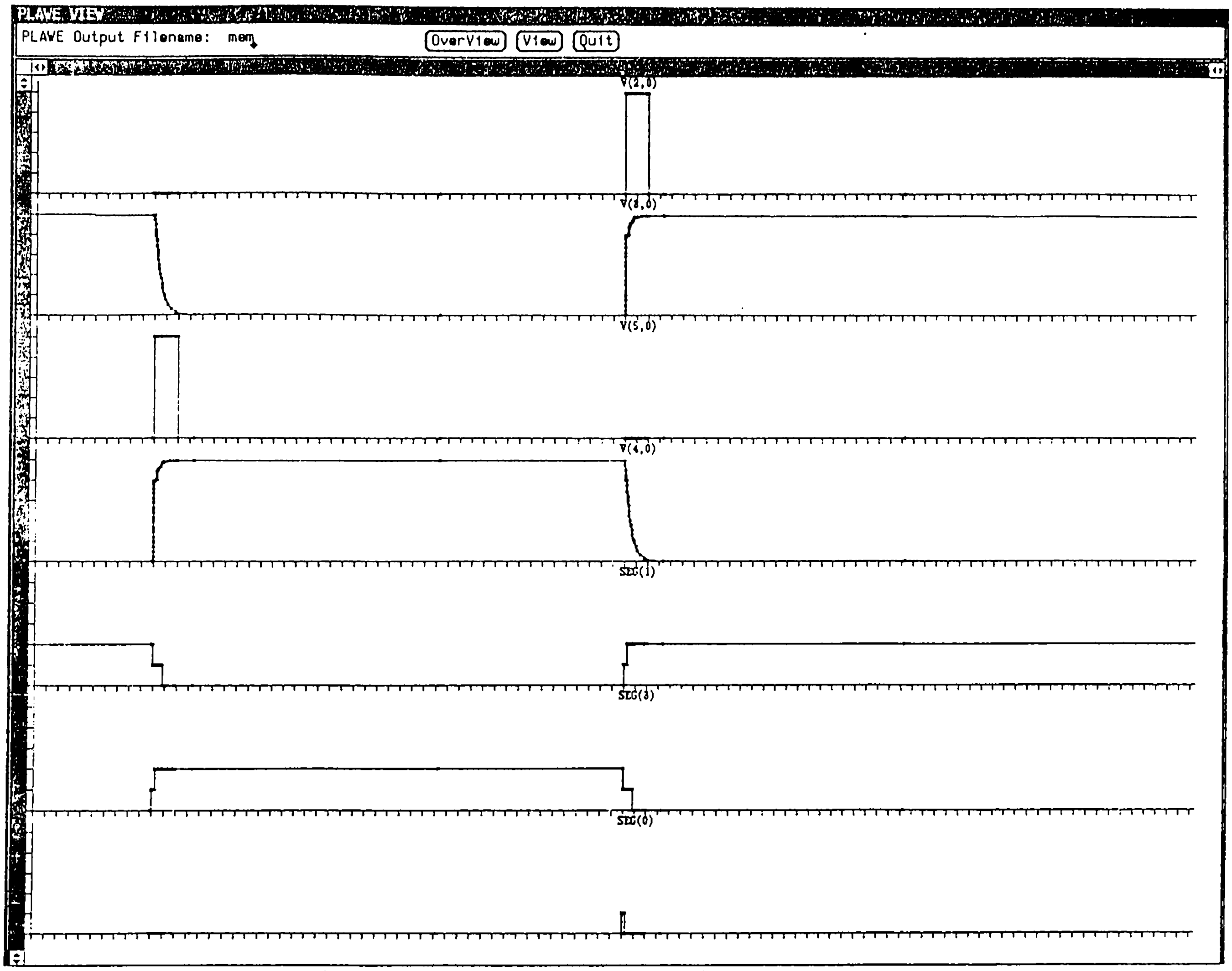


Figure 5.11: A Simple Flip-Flop. The input file for this circuit is given at the end of the 6th chapter.

are the inputs, whereas the second and fourth waveforms are the voltages of the capacitors. The operating regions of the NMOS transistors are shown in the following two plots. Finally the state of D1 can be seen in the last plot. The dots on the waveforms are the values calculated by BUSTLE, and the remaining waveform is a linear interpolation. Note that BUSTLE has worked on the rapidly changing parts and skipped computation along the time axis where the waveform does not change at all. The total execution time was 11.2 CPU seconds.

6) The sixth example, Fig. 5.13, is a ring oscillator. A transient analysis is performed again with SPICE and BUSTLE. Output waveforms of BUSTLE and SPICE can be seen in Fig. 5.14. The results are not very different. The normalized rms difference between the results of BUSTLE and SPICE is 8.4%. The frequency of oscillations are almost equal in both simulation results. However in terms of execution times, SPICE is faster than BUSTLE (29 versus 99sec). This may be due to the nature of the problem, because in an oscillator, the transistors change their operating regions frequently, which means a new DC Analysis and a new AWE. Except this single example, BUSTLE always finished the job in a shorter time than SPICE.

7) The last example is a CMOS Full Adder taken from [23]. The adder has 28 MOSFETS. The outputs CARRY and SUM are loaded with 1-pf capacitors. The adder is simulated using SPICE and BUSTLE with the inputs shown in Fig. 5.15. The "carry-in" input is grounded. The simulation is carried out by BUSTLE and results are shown in Fig. 5.15. The propagation delay can be observed in outputs SUM and CARRY. Simulation results of BUSTLE are acceptable. However we couldn't compare it with SPICE, because it gives an error message "internal time step too small in transient analysis" and


PLAWE VIEW

PLAWE Output Filename: mem

Figure 5.12: The plotting routine of BUSTLE showing the Flip-Flop outputs.

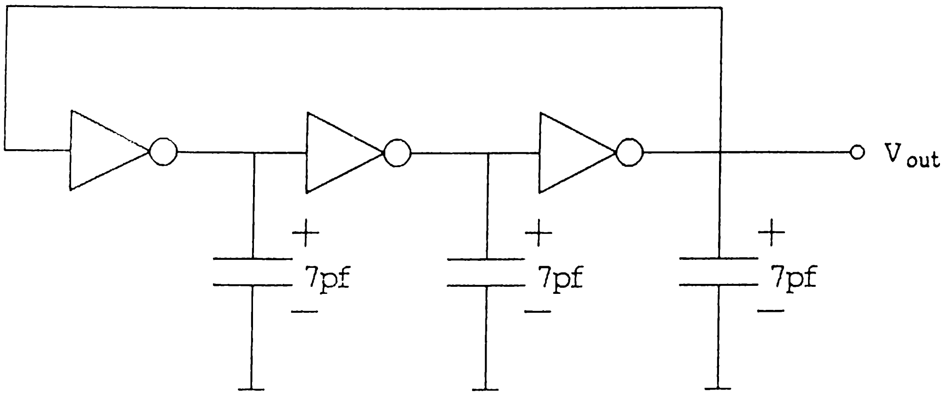


Figure 5.13: The Ring Oscillator Circuit

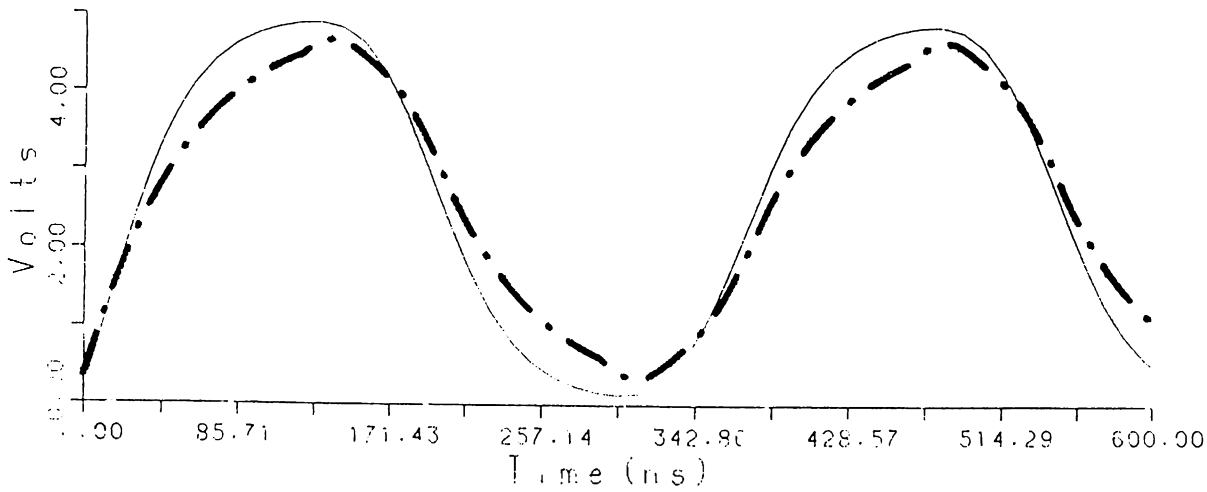


Figure 5.14: Transient analysis of the ring oscillator circuit

aborts from the program. SPICE refused to complete the analysis although we changed the parameters in the option card many times. The execution time was 95 CPU seconds for BUSTLE.

An analysis on the timings of the results show that the LU decomposition gains an important role as the size of the circuit increases. This brings the necessity of good LU decomposition algorithms and maybe the partitioning of the sparse matrix, in order to diminish the work done in the decomposition after a small change in the matrix.

As seen from the examples, the results of BUSTLE is accurate, and it is faster than SPICE, even in small circuits. Although the program is not optimized for speed yet, it is very fast. It is observed that the performance of the program increases as the size of the circuit grows.

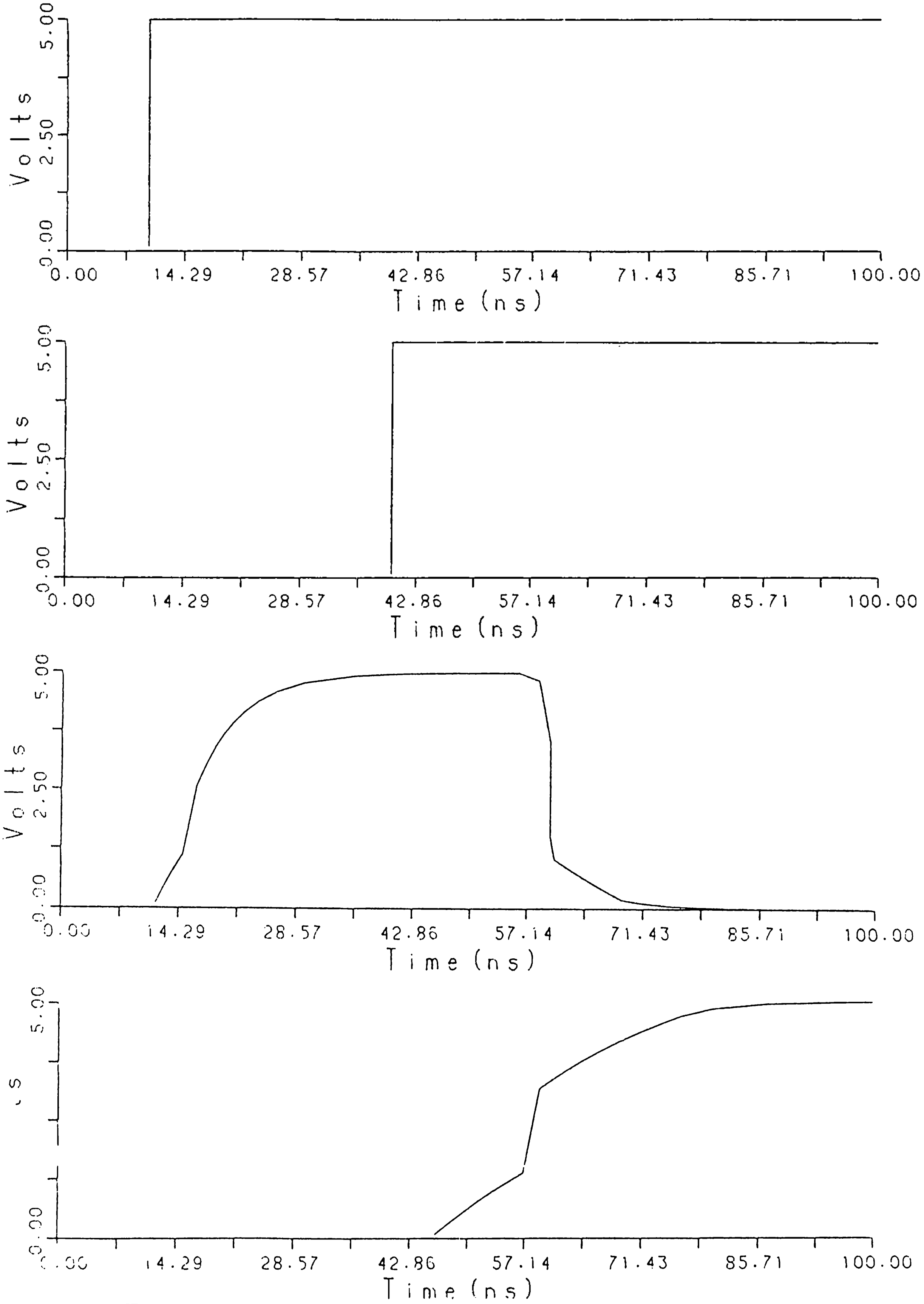


Figure 5.15: Simulation results of BUSTLE, for CMOS Full-Adder circuit. First two waveforms are the inputs, the other two are SUM and CARRY outputs resetively. (carry-in is grounded)

Chapter 6

HOW TO USE BUSTLE

BUSTLE is a general-purpose circuit simulation program. In order to use it efficiently its input format is made similar to that of existing simulators, i.e. SPICE.

6.1 INPUT FORMAT

The input format for BUSTLE is of free format type. Fields on a card are separated by one or more blanks. In order to pass from one card to another, < *RETURN* > must be entered. A card may be continued by entering a + sign in the beginning of the following card; BUSTLE continues reading after the + sign.

A name field must begin with a letter (A through Z), and cannot contain any delimiters.

A number field may be an integer field, a floating point field, either an integer or floating point number followed by an integer exponent, or either an integer or a floating point number followed by one of the following scale factors.

G=1E9 MEG=1E6 K=1E3 M=1E-3 U=1E-6

N=1E-9 P=1E-12 F=1E-15

6.2 CIRCUIT DESCRIPTION

The circuit to be analyzed is described to BUSTLE by a set of element cards, which define the circuit topology and element values, and a set of control cards, which define the required type of circuit analysis and a set of model cards which define the model parameters.

The first card must be the BEGIN card which initiates the reading of the input, and the last card must be the END card. The other cards must be in the following order:

- BEGIN card
- MODEL cards
- ELEMENT cards
- CONTROL cards
- END card

Each element in the circuit is specified by an element card that contains the element name, the nodes to which the element is connected and the values of the parameters which determine the electrical characteristics of the element. The first letter of the element name specifies the element type.

Nodes must be nonnegative integers and should be numbered sequentially. The ground node must be numbered zero. The branch numbers are given internally. The circuit can not contain a loop of inductors and a cutset of capacitors. Each node in the circuit must have a dc path to ground.

6.3 BEGIN CARD, COMMENT CARDS, END CARD

6.3.1 Begin Card

Examples:

```
.BEGIN
```

The input deck must always begin with the begin card.

6.3.2 Comment Card

General Form

$* <$ any comment $>$

or

$\# <$ any comment $>$

"*" and "#" in the beginning of a line indicates that this card is a comment card. Comment cards may be placed anywhere in the circuit description.

6.3.3 End Card

Examples:

```
.END
```

The input deck must always end with the end card.

6.4 ELEMENT CARDS

6.4.1 Resistors

General form:

RXXXXXXX N1 N2 VALUE

Examples:

```
RC 1 2 100
r1 12 0 1K
```

N1 and N2 are the two element nodes. VALUE is the resistance (in ohms), and may be positive , negative or zero.

6.4.2 Capacitors and Inductors

General form:

CXXXXXXX N+ N− VALUE $< INCOND >$

LXXXXXXX N+ N− VALUE $< INCOND >$

Examples:

```
C1 1 2 3uf 2v
lshunt 71 20 1mh
```

N+ and N− are the positive and negative nodes of the element, respectively. VALUE is the capacitance in Farads or the inductance in Henries. For the capacitor, the (optional) initial condition is the initial value of capacitor voltage in volts. For the inductor, the (optional) initial condition is the initial value of inductor current in amperes that flows from N+ to N−.

6.4.3 Linear Dependent Sources

BUSTLE allows circuits to contain linear dependent sources characterized by any of the four equations

$$i = gv_c \qquad v = ev_c \qquad i = fi_c \qquad v = hi_c$$

where g, e, f, and h are constants representing transconductance, voltage gain, current gain, and transresistance, respectively.

Linear Voltage-Controlled Current Sources

General form:

GXXXXXXX N+ N− NC+ NC− VALUE

Examples:

```
G5 3 4 7 1 1mmho
```

N+ and N− are the positive and negative nodes respectively. NC+ and NC− are the positive and negative controlling nodes respectively. VALUE is the transconductance (in mhos).

Linear Voltage-Controlled Voltage Sources

General form:

EXXXXXXX N+ N− NC+ NC− VALUE

Examples:

```
eampl 13 5 3 0 4.2
```

N+ and N− are the positive and negative nodes respectively. NC+ and NC− are the positive and negative controlling nodes respectively. VALUE is the voltage gain.

Linear Current-Controlled Current Sources

General form:

FXXXXXXX N+ N− NC+ NC− VALUE

Examples:

```
F1 15 5 2 7 5
```

N+ and N− are the positive and negative nodes respectively. NC+ and NC− are the node numbers of the controlling branch. VALUE is the current gain.

Linear Current-Controlled Voltage Sources

General form:

HXXXXXXX N+ N− NC+ NC− VALUE

Examples:

```
hv1 33 0 7 0 1.2M
```

N+ and N− are the positive and negative nodes respectively. NC+ and NC− are the node numbers of the controlling branch. VALUE is the transresistance (in ohms).

6.4.4 Independent Sources (Time Invariant)

General form:

VXXXXXXX N+ N– VALUE

IXXXXXXX N+ N– VALUE

Examples:

```
vin 1 2 2v
Is 5 0 4.5mA
```

N+ and N– are the positive and negative nodes respectively. VALUE is the value of the source (in volts or amperes).

6.4.5 Time Varying Independent Sources

Any independent source can be assigned a time-dependent value for transient analysis. There are two independent source functions : pulse, piece-wise linear.

1. **PULSE:** PULSE V1 V2 TD TR TF PW PER

   Examples:

   ```
   VIN 1 0 pulse -1v 1v 2ms 2ms 2ms 40ms 90ms
   Vs 3 0 pulse 0v 5v 5ns 0ns 0ns 20ns 50ns
   ```

Parameter	Description	Units
V1	initial value	Volts or Amps
V2	pulsed value	Volts or Amps
TD	delay time	seconds
TR	rise time	seconds
TF	fall time	seconds
PW	pulse width	seconds
PER	period	seconds

2. **PIECE-WISE LINEAR:** PWL T1 V1 <T2 V2 ..... >

   Examples:

   ```
   VIN 3 0 pwl 0ms 0v 1ms 1v 2ms 1v 2ms 0v
   ```

   Each pair of values (T_i,V_i) specifies that the value of the source is V_i (in volts or amperes) at time = T_i. The value of the source at intermediate values of time is determined by using linear interpolation on the input values.

Note that it is possible to define an ideal step using time varying independent sources.

6.5 PWL DEVICES

The nonlinear elements of a circuit must be modeled as Piece-wise Linear in order to be used in BUSTLE. Each PWL element card contains the device name, the nodes to which the device is connected and the device model name. The characteristics of the PWL device is described in a separate model card. More than one element having the same characteristics may use the same model.

6.5.1 Two Terminal PWL Devices

General form:

DXXXXX N+ N– "MNAME"

Examples:

```
DCLMP 3 4 "diode"
```

N+ and N– are the positive and negative nodes respectively. MNAME is the model name.

6.5.2 Three Terminal PWL Devices

General form:

TXXXXX N1 NC N2 "MNAME"

Examples:

```
T1 3 5 7 "npn"
```

N1, NC, N2 are the nodes to which nonlinear device is connected. NC is the common node. MNAME is the model name.

6.6 MODEL CARDS

6.6.1 Two Terminals

General form:

.MODEL2 "MNAME" NOP pt V1 I1 pt V2 I2 < pt V3 I3 ... >

Examples:

```
.MODEL2 "d1" 3 pt -50v -50uA pt 0.6v 0.6uA pt 10v 10A
```

MNAME is the model name. NOP is the number of points used in the description. V_iT_i gives the corner of each linear segment describing the PWL characteristics.

6.6.2 Three Terminals

General form:

.MODEL3 "MNAME" < C_1 C_2 C_3 > NOR pl e_{1,v_1} e_{1,i_1} e_{1,v_2} e_{1,i_2} $e_{1,c}$ e_{2,v_1} e_{2,i_1} e_{2,v_2} e_{2,i_2} $e_{2,c}$ NOB bd NBN nb b_{v_1} b_{i_1} b_{v_2} b_{i_2} b_c <bd NBN nb b_{v_1} ... > <pl e_{1,v_1} ... >

Examples:

```
.MODEL3 "nmos" 1pf 0pf 0pf 4
+ pl 0 1 0 0 0 0 0 1 -1e7 0 2
  + bd 1 nb -1 0 0 0 1 bd 3 nb -1 0 1 0 1
+ pl 0 1 0 0 0 -400 0 -1 1e7 400 2
```

```
    + bd 0 nb 1 0 0 0 -1 bd 2 nb -1 0 1 0 1
  + pl 0 1 0 0 0 0 0 401 -1e7 0 2
    + bd 3 nb 1 0 0 0 -1 bd 1 nb 1 0 -1 0 -1
  + pl 0 1 0 0 0 400 0 -401 1e7 -400 2
    + bd 2 nb -1 0 0 0 1 bd 0 nb 1 0 -1 0 -1

  .MODEL3 "pmos" 1pf 0pf 0pf 4
  + pl 0 1 0 0 0 0 0 1 -1e7 0 2
    + bd 1 nb 1 0 0 0 1 bd 3 nb 1 0 -1 0 1
  + pl 0 1 0 0 0 400 0 1 -1e7 400 2
    + bd 0 nb -1 0 0 0 -1 bd 2 nb 1 0 -1 0 1
  + pl 0 1 0 0 0 0 0 401 -1e7 0 2
    + bd 3 nb -1 0 0 0 -1 bd 1 nb -1 0 1 0 -1
  + pl 0 1 0 0 0 -400 0 401 -1e7 -400 2
    + bd 2 nb 1 0 0 0 1 bd 0 nb -1 0 1 0 -1
```

MNAME is the model name. C_1, C_2, and C_3 are the intrinsic capacitance values between the nodes 1 and common, 2 and common, and 1 and 2 of the PWL element, respectively. If the value of an intrinsic capacitor is given as zero then it is omitted. **NOR** is the number of regions used in the description. **pl** is written to indicate the beginning of a region. Each region is defined by two branch equations and a number of boundary equations. Branch equations are as follows:

$$e_{1,v_1}v_1 + e_{1,i_1}i_1 + e_{1,v_2}v_2 + e_{1,i_2}i_2 + e_{1,c} = 0$$
$$e_{2,v_1}v_1 + e_{2,i_1}i_1 + e_{2,v_2}v_2 + e_{2,i_2}i_2 + e_{2,c} = 0$$

where v_1, i_1, v_2, i_2 are defined in Fig. 2.2.

NOB is the number of the boundaries related to the given region. A boundary is defined by the following equation:

$$b_{v_1}v_1 + b_{i_1}i_1 + b_{v_2}v_2 + b_{i_2}i_2 + b_c \geq 0$$

bd indicates the beginning of a boundary. **NBN** is the id. number of the region which is the other neighbor of this boundary. Id. number of the region which is passing through the origin (satisfying 0) is 0, and this region must be given at the first place in the region list. Id. numbers of the other regions are numbered sequentially according to their order in the list.

6.7 CONTROL CARDS

6.7.1 TRAN Card

General form:

.TRAN <TSTEP> TSTOP <TSTART> <UTS>

Examples:

```
.TRAN 1ns 100ns 10ns

.TRAN 1ns 100ns uts

.TRAN 100ns
```

TSTEP is the maximum internal time step that is allowed. TSTART and TSTOP are the initial time and final time of the transient analysis respectively. If TSTART is omitted, it is assumed to be zero. One can omit TSTEP if TSTART is also omitted. The effect of TSTEP can be removed by giving it a large value. If UTS (Use Time Step) is used then the internal time step is directly chosen as TSTEP.

6.7.2 PRINT Card

General form:

.PRINT PRTYPE OV1 <OV2 ... OV8>

Examples:

```
.PRINT TRAN VOUT 8 0 IS 1 0

.PRINT TRAN VIN 1 0 IIN 1 0

.PRINT TRAN V5 5 0 SEG2 12

.PRINT TRAN VIN 2 0 VOUT 16 0 IDD 1 0 SEG3 8 SEG2 7
```

PRTYPE shows whether the output(s) is (are) for a transient or for a dc analysis. The form for voltage, current or segment output variables is as follows:

V(N1<,N2>) specifies the voltage difference between nodes N1 and N2. If N2 is omitted, ground (0) is assumed.

I(N1,N2) specifies the current flowing in the branch that is between the nodes N1 and N2. There should be an element between the nodes N1 and N2.

SEG2 IDN specifies the segment number of the two terminal PWL device with the device id. no IDN. The device id. no's are numbered sequentially according to their order in the input deck.

SEG3 IDN specifies the segment number of the three terminal PWL device with the device id. no IDN. The device id. no's are numbered for three terminal PWL devices same as the two terminal elements.

6.7.3 PLOT CARD

General form:

.PLOT PRTYPE OV1 <OV2 ... OV8>

Examples:

```
.PLOT TRAN VOUT 8 0 IS 1 0
```

This card very similar to PRINT card. It performs everything that PRINT card do and additionally it calls the BUSTLE-view routine if you are working at a SUN workstation.

6.7.4 OPTIONS Card

General form:

.OPTIONS OPT1 OPT2 ... (or OPT=OPTVAL)

Examples:

```
.OPTIONS ORDER=2 REFNUM=1
```

ORDER is the minimum order of approximation in AWE. Default is 1.

NOFDER is the initial number of the derivative moments used in AWE. Default is zero, which means no derivatives is used in the approximation.

REFNUM is the number of the refinements done while solving the circuit. In general there is no need for refinement. Default is zero, which means no refinement.

DEBUG is the level of printing the debugging material in the *infile*.info file. Default is zero, which does not create the *infile*.info file.

TSFP is the Time Step Finding Period. BUSTLE computes the time step in transient analysis dynamically after every TSFP time steps. Default is 1.

SAFETY is the safety factor used in the calculation of the internal time step. The larger the SAFETY, smaller the internal time step. Default is 4.

6.8 EXAMPLE INPUT FILE

```
************* MEMORY CELL ******************

begin
**********************************************
**********       MOSFET MODELS    ***********
**********************************************
MODEL3 "nmos" 4
+ pl 0 1 0 0 0 0 0 1 -1e7 0 2      #CUT-OFF Ig=0,
                                   # Vds - 10e7*Ids =0
  + bd 1 nb -1 0 0 0 1 bd 3 nb -1 0 1 0 1
+ pl 0 1 0 0 0 -400 0 -1 1e7 400 2 # SAT Ig=0,
                                   # 400*Vgs+Vds-1e7*Ids=400
  + bd 0 nb 1 0 0 0 -1 bd 2 nb -1 0 1 0 1
+ pl 0 1 0 0 0 0 0 401 -1e7 0 2    # LINEAR Ig=0,
                                   # 401*Vds-1e7*Ids =0
  + bd 3 nb 1 0 0 0 -1 bd 1 nb 1 0 -1 0 -1
+ pl 0 1 0 0 0 400 0 -401 1e7 -400 2 #REV-SAT
  + bd 2 nb -1 0 0 0 1 bd 0 nb 1 0 -1 0 -1

MODEL3 "pmos" 4
+ pl 0 1 0 0 0 0 0 1 -1e7 0 2         #CUT-OFF
  + bd 1 nb 1 0 0 0 1 bd 3 nb 1 0 -1 0 1
+ pl 0 1 0 0 0 400 0 1 -1e7 400 2     #SAT
  + bd 0 nb -1 0 0 0 -1 bd 2 nb 1 0 -1 0 1
+ pl 0 1 0 0 0 0 0 401 -1e7 0 2       #LINEAR
  + bd 3 nb -1 0 0 0 -1 bd 1 nb -1 0 1 0 -1
+ pl 0 1 0 0 0 -400 0 401 -1e7 -400 2 #REV-SAT
  + bd 2 nb 1 0 0 0 1 bd 0 nb -1 0 1 0 -1
*********************************************

*****************************************************
*************        DIODE MODELS    ****************
*****************************************************
.MODEL2 "d1" 3 pt -50v -50nA pt 0.6v 0.6nA pt 10v 10A
.MODEL2 "d3" 4 pt -50v -50nA pt 0.6v 0.6nA
             + pt 0.7v 10mA pt 10v 100A
*****************************************************



****************************************************
***********   THE CIRCUIT DEFINITION   *************
****************************************************
vdd 1 0 5v
vin1 2 0 pwl 0ns 0v 25ns 0v 25ns 5v 26ns 5v 26ns 0v
       ### do not forget to try for another vin1  ###
vin2 5 0 pwl 0ns 0v 5ns 0v 5ns 5v 6ns 5v 6ns 0v

c1 4 0 0.01pf 0v
c2 3 0 0.01pf 5v

d1 2 3 "d1"
d2 5 4 "d1"

tp1 3 1 4 "pmos"
tn1 3 0 4 "nmos"
tp2 4 1 3 "pmos"
tn2 4 0 3 "nmos"

.TRAN 50ns
.PRINT tran v1 2 0 v2 3 0 v3 5 0 v4 4 0
***+ seg3 0   *tp1
+ seg3 1   *tn1
***+ seg3 1   *tp2
```

```
+ seg3 3   *tn2
+ seg2 0
.END
```

Chapter 7

Conclusion

BUSTLE is a new circuit level simulator especially for large circuits. It uses piece-wise linear models and *asymptotic waveform evaluation* techniques. One of the important drawbacks of the AWE technique was the instability caused by the right-hand-plane poles. BUSTLE has handled this instability problem using combinations of derivatives and integral moments in the moment matching algorithm. The most important feature of PWL approach is 100% convergence with a high speed. Transient analysis results, using simple models with few segments are quite good. Since BUSTLE gives the user the capability of defining his own models in a simple manner, it has the power to keep pace with the new developments. We have obtained some advantages and many others seem to appear with a future work. Some of the points which needs future study are below.

- Efficient PWL modeling.
- Partitioning in LU decomposition.
- Direct AWE for the nonlinear devices and the outputs, and different time step for each.
- An algorithm to determine the order of approximation and the number of derivative moments in the approximation automatically that will check the accuracy, and guarantees the stability.
- A simple method to obtain the moments of the transmission lines directly from the geometry and use the effect of it in the calculations.

References

[1] Cemal T. Dikmen. "BUSTLE: A New Simulation Tool Using Asymptotic Waveform Evaluation and PWL Approach," M.S. Thesis, Bilkent University, Ankara, Turkey, 1990.

[2] Lawrence T. Pillage , Ronald A. Rohrer. "Asymptotic Waveform Estimation," *IEEE Trans. Computer-Aided Design*, pp. 352-366, April 1990.

[3] Lawrence T. Pillage. "Asymptotic Waveform Evaluation for Timing Analysis," Ph.D. dissertation, Carnegie Mellon University, Pennsylvania, USA, 1989.

[4] X. Huang, V. Raghavan, R. A. Rohrer. "AWEsim : A Program for the Efficient Analysis of Linear(ized) Circuits," *In ICCAD 90* , 1990.

[5] J. Y. Lee, X. Huang, R. A. Rohrer. "Efficient Pole Zero Sensitivity Calculation in AWE," *In ICCAD 90* , 1990.

[6] L. T. Pillage, X. Huang, R. A. Rohrer. "Asymptotic Waveform Evaluation for Circuits containing Floating Nodes," In Proceedings of the 1990 International Symposium on Circuits and Systems, 1990.

[7] J. Katzenelson. *Bell Syst. Tech. J.,* Vol. 44, pp. 1605-1620, October 1965.

[8] T. Fujisawa , E. S. Kuh. "Piecewise-Linear Theory of Nonlinear Networks," *J. Appl. Math.* , Vol. 22 , no. 2, pp. 307-328, March 1972.

[9] M. J. Chien, E. S. Kuh. "Solving nonlinear resistive networks using piecewise-linear analysis and simplical subdivision," *IEEE Trans. on Circuits and Systems,* Vol. 24, pp. 305-317, June 1977.

[10] H. J. Strayer, D. J. Roulston, P. R. Bryant. "DC Solution Speed in Piecewise Linear Network Analysis Programs," *Electronics Letters* , Vol. 22 , no. 3, pp. 165-166, January 1986.

[11] D. J. Evans. *Sparsity and Its Applications.* Cambridge Univesity Press, 1985.

[12] G. W. Stewart. *Computer Science and Applied Mathematics.* Academic Press, Inc., 1973.

[13] Z. Zlatev, J. Wasniewski, K. Schaumburg. *Y12M, Solution of Large Systems of Linear Algebraic Equations.* Springer-Verlag, 1981.

[14] A. F. Schwarz. *Computer Aided Design of Microelectronic Circuits and Systems*

[15] Hu Xiheng. "FF-Pade Method of Model Reduction in Frequency Domain," *IEEE Trans. Automatic Control*, Vol. AC-32, No. 3, pp. 243-246, March 1987.

[16] V. Zakian. "Simplification of Linear Time-Invariant Systems by Moment Approximants," Int. J. Control, Vol. 18, No. 3, pp. 455-460, 1973.

[17] R. A. Rohrer. *Circuit Theory : An Introduction to the State Variable Approach.* McGraw-Hill, Inc. , 1970.

[18] L. O. Chua, P. M. Lin. *Computer-Aided Analysis of Electronic Circuits.* Prentice-Hall, Englewood Cliffs, 1975.

[19] J. Gilbert, L. Gilbert. *Elements of Modern Algebra.* PWS-KENT Publishing Company, Boston, 1988.

[20] P. R. Adby. *Applied Circuit Theory: Matrix and Computer Methods.* John Wiley & Sons, NewYork, 1980.

[21] J. Vlach, K. Singhal. *Computer Methods for Circuit Analysis and Design.* Van Nostrand Reinhold Company, NewYork, 1983.

[22] W. H. Press, B. P. Plannery, S. A. Teukolsky, W. T. Vetterling. *Numerical Recipes in C.* Cambridge University Press, Cambridge, 1988.

[23] N. Weste, K. Eshraghian. *Principles of CMOS VLSI Design, A Systems Perspective.* Addison-Wesley Publishing Company, USA, 1985.

[24] C. A. Desoer, E. S. Kuh. *Basic Circuit Theory.* McGraw-Hill,