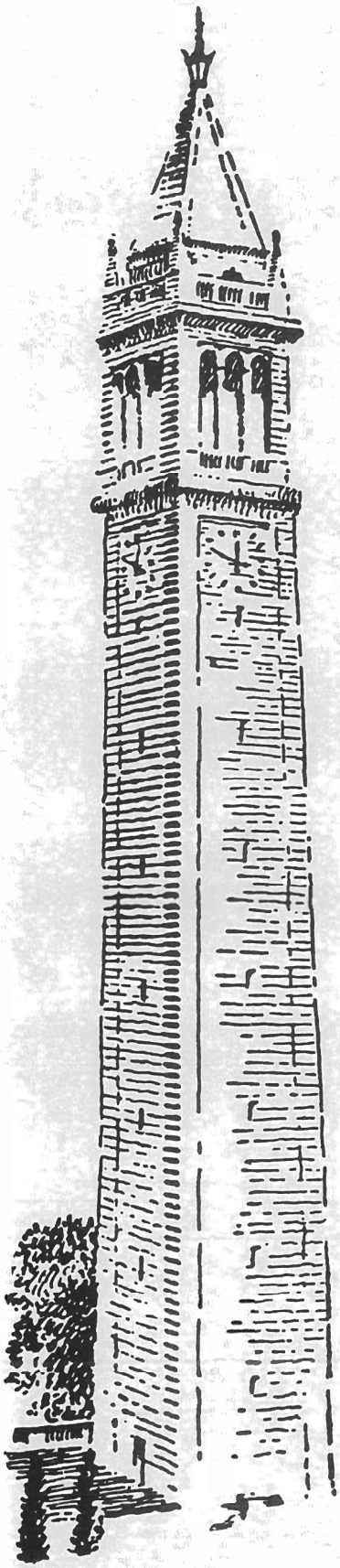


SPICE2: A COMPUTER PROGRAM TO SIMULATE SEMICODUCTOR CIRCUITS

by

Laurence W. Nagel

Memorandum No. UCB/ERL M520

9 May 1975

ELECTRONICS RESEARCH LABORATORY
College of Engineering
University of California, Berkeley, CA 94720

SPICE2: A COMPUTER PROGRAM TO SIMULATE SEMICONDUCTOR CIRCUITS

by

Laurence W. Nagel

Memorandum No. ERL-M520

9 May 1975

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

ABSTRACT

The size and complexity of present integrated circuits has increased to the point where computer aids are virtually indispensable. A circuit simulation program that characterizes the electrical performance of a circuit is one important computer aid in the circuit design process. The need for accurate and efficient circuit simulation has prompted the development of many circuit simulation programs as well as the advancement of the assoicated numerical methods.

Two purposes are served by this thesis. First, the numerical methods that are necessary ingredients of a circuit simulation program are detailed, and the implementations of these methods are described and compared. Second, the theoretical and practical aspects of the SPICE1 and SPICE2 programs are documented. SPICE is one of several circuit simulation programs that presently are used by a substantial portion of the electronics industry. The development of SPICE is due mainly to the author. The numerical methods that are employed in SPICE, of course, are the result of many researchers.

This thesis begins with a functional description of the SPICE program. The presentation of the overall task of circuit simulation is subdivided into the topics of equation formulation, linear equation solution, nonlinear equation solution, and numerical integration. The bulk of this thesis is devoted to a comparison of the available methods within each of these general topics. These comparisons are based upon an evaluation of the performance of the methods for typical electronic circuit simulation problems.

The numerical methods that are employed in both versions of SPICE are

presented in detail. These methods are chosen according to guidelines that are presented in the introductory sections of this thesis. Different guidelines probably would result in the implementation of different methods. The widespread use of SPICE, however, indicates that the algorithms that are employed in SPICE are applicable to a wide range of practical circuit simulation problems.

ACKNOWLEDGEMENTS

The development of SPICE entailed the effort of many people. SPICE evolved from the CANCER program which, in turn, emerged from a series of courses that were instructed by Professor R. A. Rohrer. My classmates in these courses, R. Berry, S. P. Fan, F. Jenkins, J. Pipkin, S, Ratner, and L. Weber, devoted a noteworthy amount of time and energy to the original development of CANCER. My research advisor, Professor D. O. Pederson, provided the ongoing guidance and support that was essential to the success of both CANCER and SPICE.

CANCER evolved into SPICE with the additional guidance of Professors D. A. Hodges, R. G. Meyer, and P. A. Gray. The notable assistance of R. I. Dowell and the helpful discussions with I. E. Getreau and W. G. McCalla furthered the development of SPICE. The evolution of SPICE2 was helped significantly by the tireless efforts of E. Cohen and valuable discussions with S. P. Fan and S. Vidanage. Many of the system-dependent features of SPICE2 could not have been implemented without the help of T. Sumner at the Computer Center. I owe a special note of appreciation to my wife, Harriet, whose love, patience, and understanding encouraged the development of SPICE as well as the writing of this thesis.

The research that is detailed in this thesis would not be possible without significant financial aid. The SPRAGUE Electric Company grant for the academic year 1970-1971 aided the preliminary development of CANCER. Continued advancement of CANCER and SPICE was supported by the NSF Grant GK-17931 and the ARO Grant DA-ARO-D-31-124-72-G52. The large amounts of computer resources that are vital to the development of any computer program were generously supplied by the Computer Center of the University of California, Berkeley.

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.) PAGE NOS.

TABLE OF CONTENTS (cont.)

I. INTRODUCTION

Electronic circuit design requires an accurate method of assessing circuit performance. For the design of discrete circuits, the traditional "breadboard" is a convenient method of measuring the electrical characteristics of a circuit. The circuit can be modified and probed at will to isolate the causes of a marginal or unacceptable design, and design improvements can be made immediately. Since a breadboard closely resembles the circuit that finally will be built, the laboratory measurements yield an accurate characterization of the final circuit performance.

The design of an integrated circuit (IC) poses an entirely different problem. A breadboard obviously bears no resemblance to the IC that will be produced. The parasitic components that are present in the breadboard are entirely different from the parasitic components that are present in an integrated circuit. For this reason, breadboard measurements often yield an inaccurate characterization of integrated circuit performance. Fabrication and testing of the IC certainly will verify an acceptable design; however, the very small size of the IC precludes extensive probing and modification to determine how a marginal or unacceptable design can be improved.

A computer program that simulates the electrical performance of an electronic circuit circumvents many of the practical problems that are encountered in circuit characterization. The circuit is represented in mathematical terms, and numerical analysis procedures that correspond to typical laboratory measurements are performed. The circuit designer chooses the analyses that are performed and, by

analogy, the measurements that are made upon the circuit. The output of the simulation program therefore simulates the results of laboratory measurements. Moreover, circuit simulation can provide information about circuit performance that virtually is impossible to obtain with laboratory measurements.

SPICE [1] is one of several successful circuit simulation programs that currently are available. The SPICE program was developed primarily by the author and is one of many simulation programs that have been developed by the Integrated Circuits group of the Electronics Research Laboratory at the University of California, Berkeley. SPICE evolved from the CANCER program [2,3]. The first version of SPICE essentially was finished in 1972. Since that time, more than one hundred copies of the program have been given to universities and companies in the electronics industry.

The widespread usage of SPICE attests to the applicability of the program to a large variety of circuit simulation problems. This usage also provided a valuable base of experience on the advantages and disadvantages of the computational methods that are employed in the SPICE program. This experience, in turn, prompted the development of SPICE2, the second version of SPICE.

This thesis describes the development and the design of the SPICE programs. The computational methods that are necessary for the task of circuit simulation are presented, and the specific algorithms that are employed in the SPICE program are detailed. The performance of these algorithms for typical electronic circuits is presented, and the techniques that are used in SPICE are compared with other available methods.

1.1. Description of SPICE

SPICE is a digital computer program that simulates the electrical performance of electronic circuits. This program will determine the quiescient operating point of the circuit, the time-domain response of the circuit, or the small-signal frequency-domain response of the circuit. SPICE contains models for the common circuit components and is capable of simulating most electronic circuits. The SPICE program consists of approximately 10000 FORTRAN IV and COMPASS statements and was written for the CDC 6400 computer that is available the Computer Center of the University of California, Berkeley.[1] However, the program also has been adapted for use on IBM, Honeywell, UNIVAC, RCA, and PDP computer systems.

The SPICE input syntax is a free-format style that does not require that data be entered in fixed column locations. The program supplies reasonable default values for circuit parameters that are not specified and performs a considerable amount of error-checking to insure that the circuit has been entered correctly. A beginning user need specify a minimal number of circuit parameters and simulation controls to obtain reasonable simulation results.

1.2. Circuit Definition in SPICE

The program input defines the circuit to be simulated on an element by element basis. The types of circuit elements that presently are included in the SPICE program are listed in Table 1.1. The models

[1] The Computer Center CDC 6400 operates under the CALIDOSCOPE operating system that was developed by the Computer Center staff. The FORTRAN portions of SPICE are compiled on the RUNW.2 compiler, and the assembly language portions are assembled with the COMPASS 1.1 assembler.

for these elements are detailed in Appendix 2 of this thesis. The SPICE elements include resistors, capacitors, inductors, coupled inductors, independent voltage and current sources, and the four most common semiconductor devices: the diode, the bipolar-junction transistor (BJT), the junction field-effect transistor (JFET), and the insulated-gate field-effect transistor (IGFET or MOSFET).

An example best illustrates the SPICE program input. Figure 1.1 shows the schematic diagram for a differential-pair amplifier stage. Each node in the circuit is assigned a unique integer number, with the number zero reserved for the ground node. Each element circuit is assigned a unique name. The first letter of the element name is a prefix that identifies the element type. For example, resistor names begin with the letter R, and BJT names begin with the letter Q. The prefix letter for each SPICE element is included in Table 1.1.

A SPICE input deck that defines the differential-pair circuit is given in Fig. 1.2. The first card in the deck is the title card; the contents of this card are printed as the heading for the various sections of the SPICE output. The last card in the deck is the .END card which serves only to signify the end of the input deck. Except for the title card and the .END card, the order of the cards is of no importance to the program.

Each element in the circuit is defined by an element card that contains the element name, the nodes that the element is connected to, and the values of the parameters that define the electrical characteristics of the element. For example, the voltage source VCC has its positive node connected to node 8 of the circuit and its negative node connected to ground, and the source value is 12 volts.

Linear Elements

- Resistor (R)
- Capacitor (C)
- Inductor (L)
- Mutual Inducotr (K)
- Independent Voltage Source (V)
- Independent Current Source (I)
- Linear Voltage-Controlled Current Source (G)

Nonlinear Elements

- Nonlinear Voltage-Controlled Current Source (N)
- Diode (D)
- Bipolar Junction Transistor (Q)
- Junction Field-Effect Transistor (J)
- Insulated-Gate Field-Effect Transistor (M)

Table 1.1. SPICE Circuit Elements

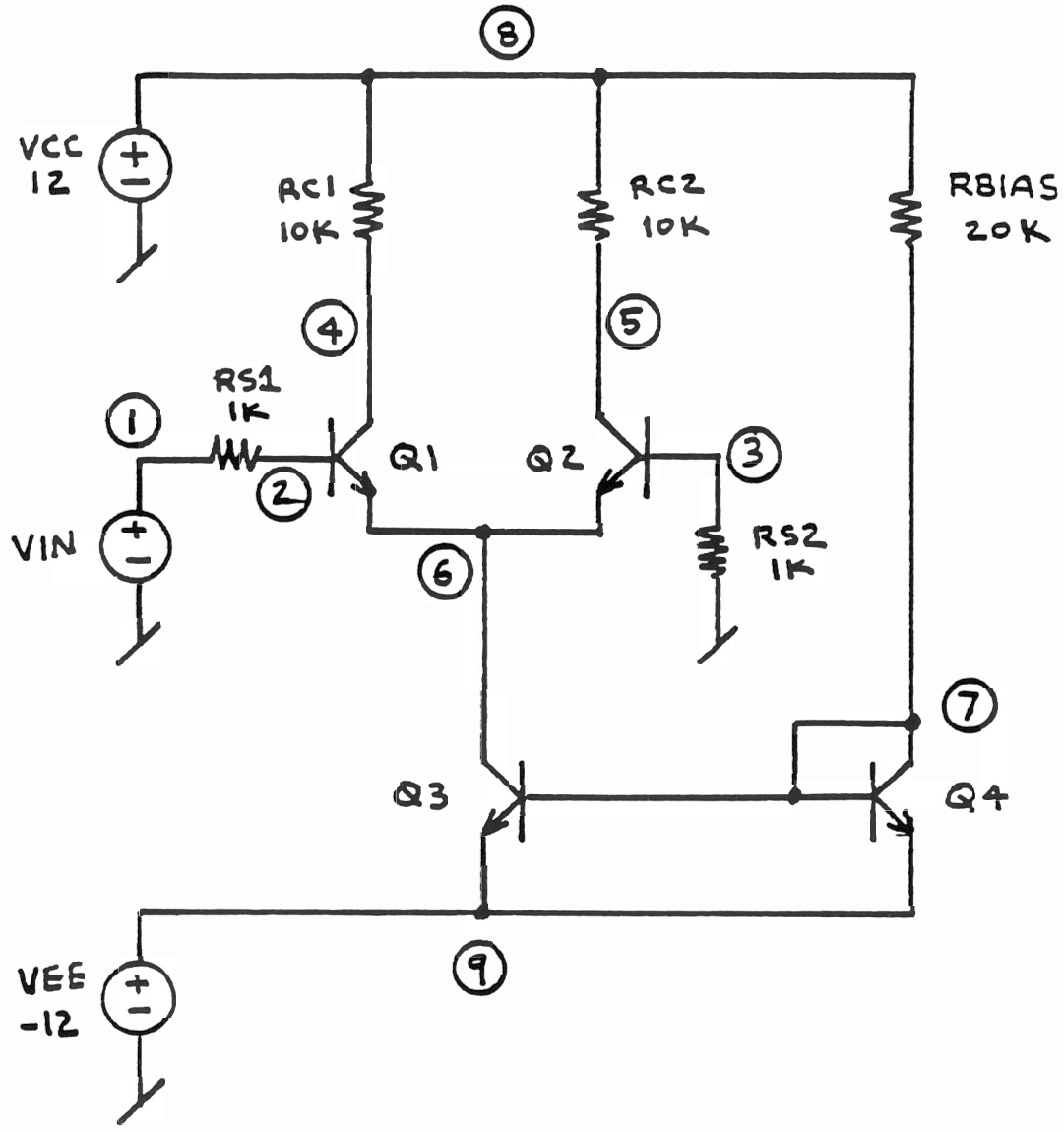


Fig. 1.1. Schematic Diagram for the Differential-Pair Circuit.

```
DIFFPAIR CKT - SIMPLE DIFFERENTIAL PAIR
VIN 1 0 SIN(0 0.1 5MEG 5NS) AC 1
VCC 8 0 12
VEE 9 0 -12
Q1 4 2 6 QNL
Q2 5 3 6 QNL
RS1 1 2 1K
RS2 3 0 1K
RC1 4 8 10K
RC2 5 8 10K
Q3 6 7 9 QNL
Q4 7 7 9 QNL
RBIAS 7 8 20K
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
+  CJC=2PF VA=50)
.END
```

Fig. 1.2. SPICE Input Deck for the Differential-Pair Circuit.

The value for the voltage source VIN is specified by a predefined SIN function; VIN is a sine wave with a zero-volt offset, a 0.1 volt amplitude (0.2 volts peak-to-peak), and a frequency of 5 MHz.

The linear elements that are given in Table 1.1 require only one or two parameter values to specify completely the electrical characteristics of the element. However, the models for the four semiconductor devices, which include the BJT, are more complicated and contain many parameters. In addition, several devices in the circuit often can be described by a common set of model parameters. Therefore, it is convenient to specify device model parameters on a separate .MODEL card; this set of parameters is assigned a unique model name that then is referenced by the device element cards. In this example, all four BJT's share the same set of model parameters that is defined on the .MODEL card and assigned the model name QNL.

1.3. SPICE Analysis Types

In addition to defining the circuit, the program input specifies the analyses to be performed and the output to be generated. This information is entered on control cards. The simulation of an electronic circuit usually requires a combination of three basic analyses: dc analysis, time-domain transient analysis, and small-signal ac analysis. SPICE contains all three of these analysis capabilities. In addition, several subanalysis capabilities also are incorporated into the SPICE program. The ten analysis types that are available in SPICE are given in Table 1.2.[2]

[2]Small-signal sensitivity analysis, noise analysis, and distortion analysis are implemented in the first version of SPICE but have not yet been incorporated into SPICE2.

DC Analysis

- DC Operating Point
- Linearized Device Model Parameterization
- Small-Signal Transfer Function
- Small-Signal Sensitivities
- DC Transfer Curves

Transient Analysis

- Time-Domain Response
- Fourier Analysis

AC Analysis

- Small-Signal Frequency-Domain Response
- Noise Analysis
- Distortion Analysis

Table 1.2. SPICE Analysis Capabilities

1.4. DC Analysis

A dc analysis determines the quiescient operating point of the circuit. All energy-storage elements in the circuit are ignored in a dc analysis by treating capacitors as open circuits and inductors as short circuits. At the conclusion of the operating point analysis, the program prints the circuit node voltages, independent voltage source currents, and the total quiescient power dissipation of the circuit.

For small-signal operation, the perturbational response of the circuit can be determined by modeling the nonlinear elements in the circuit by equivalent linearized models. The parameters of the linearized models depend, of course, upon the quiescient operating point. These linearized models are given in Appendix 2 of this thesis. For the case of the BJT, the familiar hybrid-pi model [4] is used. All nonlinear element operating points, as well as the linearized model parameters, are printed at the conclusion of the dc operating point analysis.

If an output variable and an input source is specified, SPICE also will determine the dc, small-signal transfer function value. This transfer function is the small-signal ratio of the output variable to the input source. In addition, the program will determine the small-signal input resistance and the small-signal output resistance of the circuit.

SPICE also contains a subprogram that computes and prints the dc sensitivities of specified output variables with respect to every circuit parameter.[3] The efficient adjoint network concept [5,6] is

[3]The sensitivities with respect to JFET and MOSFET model parameters are not presently implemented in SPICE.

employed to determine these sensitivities.

The SPICE input that is shown in Fig. 1.2 is sufficient to determine the dc operating point of the circuit. The output of the SPICE program, for the input shown in Fig. 1.2, is presented in Figs. 1.3 through 1.5. First, the program reads the input and lists the contents of the input as shown in Fig. 1.3. If errors are encountered, the program terminates after reading the input. Otherwise, the circuit summary, model parameter summary, and node table are printed, as shown in Fig. 1.4, and the circuit data structure is constructed and checked for errors. Finally, the program determines the dc operating point and prints the quiescient node voltages, voltage source currents, nonlinear element operating points, and the linearized device model parameters, as shown in Fig. 1.5.

1.5. DC Transfer Characteristics

The dc operating point can be determined repetitively for successive values of a designated input source to obtain a set of dc transfer characteristics of the circuit. The values of specified output variables (either voltages or currents) are stored for each source point. At the conclusion of the analysis, these output variables can be listed in tabular form or presented as line-printer plots. DC transfer curves are used to determine the noise margins of digital circuits as well as the large-signal characteristics of analog circuits. As an example, the SPICE line-printer plot of the voltage at node 5 for the differential-pair circuit is shown in Fig. 1.6. To generate this

```
**********
**********
**********
**********
**********

DIFFPAIR CKT - SIMPLE DIFFERENTIAL PAIR                    ----- SPICE -----

                    DATE  14 NOV 74        CLOCK  22:32:18        VERSION  2A

**********

*  AC DEC 10 1 10GHZ
*  DC VIN -0.25 0.25 0.005
*  TRAN 5NS 500NS
.OPTIONS ACCT LIST NODE
.DC VIN -0.25 0.25 0.005
.TRAN 5NS 500NS
VIN 1 0 SIN(0 0.1 5MEG) AC 1
VCC 8 0 12
VEE 9 0 -12
Q1 4 2 6 QNL
Q2 5 3 6 QNL
RS1 1 2 1K
RS2 3 0 1K
RC1 4 8 10K
RC2 5 8 10K
Q3 6 7 9 QNL
Q4 7 7 9 QNL
RBIAS 7 8 20K
.OUTPUT V4 4 0 PRINT DC TRAN
.OUTPUT V5 5 0 PRINT MAG PHASE DC TRAN PLOT MAG PHASE DC TRAN
.MODEL QNL NPN(BF=80 BR=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF CJC=2PF VA=50)
.END
```

Fig. 1.3. SPICE Listing of Input.

**** RESISTORS

NAME	NODES		VALUE
RS1	1	2	1.00E+03
RS2	3	0	1.00E+03
RC1	4	8	1.00E+04
RC2	5	8	1.00E+04
RBIAS	7	8	2.00E+04

**** INDEPENDENT SOURCES

NAME	+	-	DC VALUE	AC VALUE	AC PHASE	TRANSIENT	
VIN	1	0	0.	1.00E+00	0.	SIN	
						OFFSET	0.
						AMPLITUDE	1.00E-01
						FREQUENCY	5.00E+06
						DELAY	5.00E-09
						THETA	0.
VCC	8	0	1.20E+01	0.	0.		
VEE	9	0	-1.20E+01	0.	0.		

**** BIPOLAR JUNCTION TRANSISTORS

NAME	C	B	E	MODEL	AREA
Q1	4	2	6	QNL	1.000
Q2	5	3	6	QNL	1.000
Q3	6	7	9	QNL	1.000
Q4	7	7	9	QNL	1.000

**** BJT MODEL PARAMETERS

	QNL
TYPE	NPN
BF	80.000
BR	1.000
IS	1.00E-14
RB	100.000
VA	50.000
TF	3.00E-10
TR	6.00E-09
CCS	2.00E-12
CJE	3.00E-12
CJC	2.00E-12

**** NODE TABLE

NODE	ELEMENTS CONNECTED			
0	RS2	VIN	VCC	VEE
1	RS1	VIN		
2	RS1	Q1		
3	RS2	Q2		
4	RC1	Q1		
5	RC2	Q2		
6	Q1	Q2	Q3	
7	RBIAS	Q3	Q4	Q4
8	RC1	RC2	RBIAS	VCC
9	VEE	Q3	Q4	

Fig. 1.4. SPICE Circuit Summary

DIFFPAIR CKT - A SIMPLE DIFFERENTIAL PAIR ----- SPICE -----

SMALL SIGNAL BIAS SOLUTION TEMPERATURE 27.000 DEG C

NODE	VOLTAGE	NODE	VOLTAGE	NODE	VOLTAGE	NODE	VOLTAGE	NODE	VOLTAGE
(1)	0.	(2)	-.0077	(3)	-.0077	(4)	5.1676	(5)	5.1676
(6)	-.6509	(7)	-11.3405	(8)	12.0000	(9)	-12.0000		

VOLTAGE SOURCE CURRENTS

NAME	CURRENT
VIN	-7.739E-06 AMPS
VCC	-2.534E-03 AMPS
VEE	2.549E-03 AMPS

TOTAL POWER DISSIPATION 6.10E-02 WATTS

TRANSISTOR OPERATING POINTS

NAME	MODEL	IB	IC	VBE	VBC	VCE	GM	RPI	RO	CPI	CMU	BETADC	BETAAC	FT
Q1	QNL	7.74E-06	6.83E-04	.643	-5.175	5.818	2.64E-02	3.34E+03	8.08E+04	1.11E-11	8.05E-13	88.3	88.2	3.52E+08
Q2	QNL	7.74E-06	6.83E-04	.643	-5.175	5.818	2.64E-02	3.34E+03	8.08E+04	1.11E-11	8.05E-13	88.3	88.2	3.52E+08
Q3	QNL	1.42E-05	1.38E-03	.660	-10.680	11.349	5.34E-02	1.82E+03	4.39E+04	1.72E-11	5.85E-13	97.1	97.1	4.78E+08
Q4	QNL	1.42E-05	1.14E-03	.660	0.	.660	4.40E-02	1.82E+03	4.39E+04	1.72E-11	2.00E-12	80.0	80.0	3.65E+08

Fig. 1.5. SPICE DC Operating Point Solution

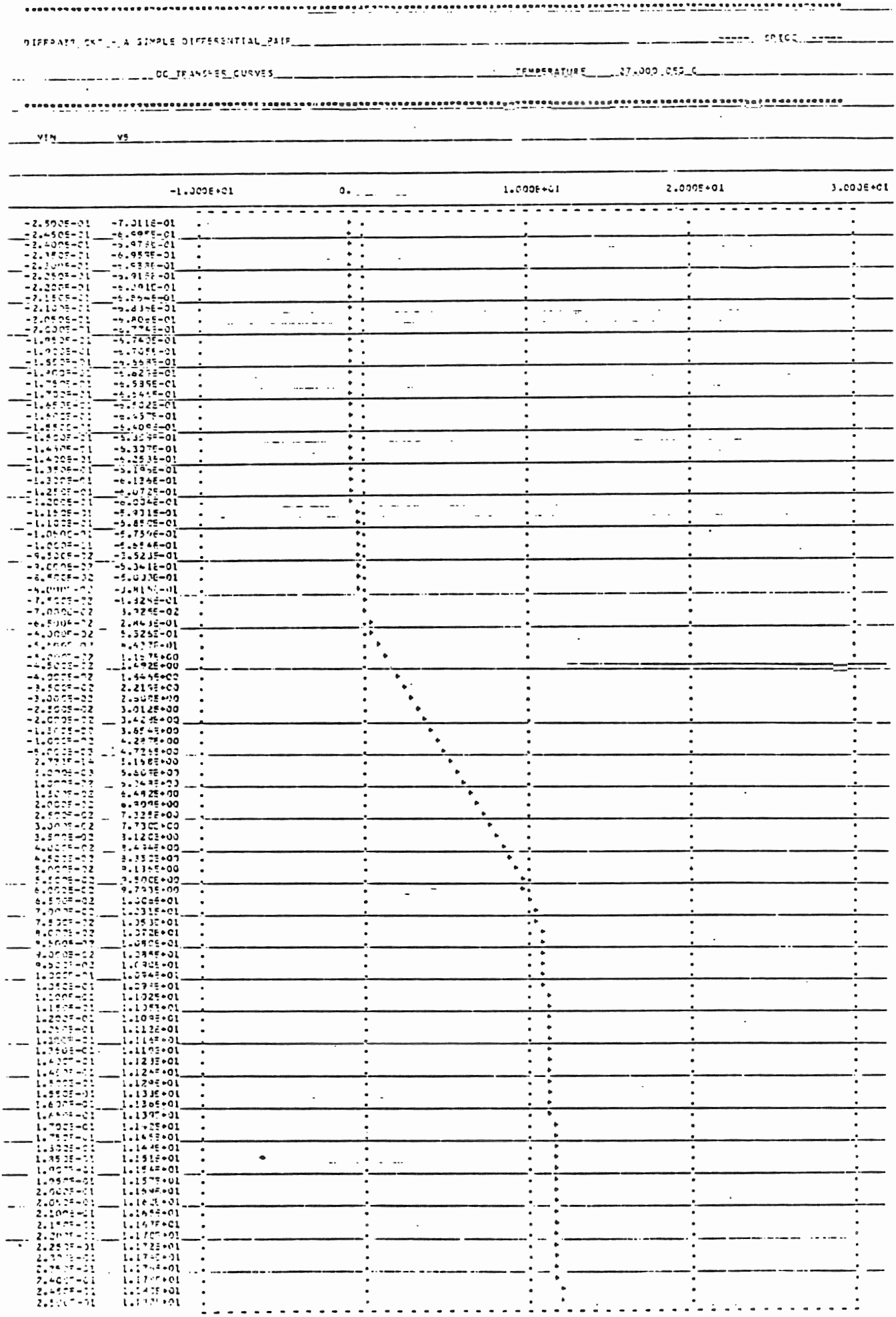


Fig. 1.6. SPICE DC Transfer Curve Solution

plot, the input source, VIN, was varied from -0.25 v to 0.25 v in 5 mv increments.

1.6. Nonlinear Time-Domain Transient Analysis

Transient analysis determines the time-domain response of the circuit to specified time-domain inputs. The initial timepoint, arbitrarily defined as time zero, is determined by a previous dc operating point solution. The independent sources in the circuit may have constant values or time-dependent values. The time interval (0,T) that is specified by the user is divided into discrete timepoints and the program determines the circuit solution at each successive timepoint starting from time zero. Voltage or current output variables are stored at each timepoint, and can be listed in tabular form or plotted at the conclusion of the analysis. The spacing between successive timepoints is controlled by the program to insure an accurate solution.[4] Output data is interpolated so that output occurs at the printing interval that is specified by the user.

The SPICE line-printer plot for the transient analysis of the differential-pair circuit is shown in Fig. 1.7. The waveform that is given in this figure is for the voltage at node 5 of the circuit; the time interval is 500 ns, and the print increment is 5 ns. The input source, VIN, is a sine wave with an offset of zero volts, an amplitude of 0.1 v (0.2 v peak-to-peak) and a frequency of 5 MHz.

The output routines contain a Fourier analysis subprogram that determines the first nine Fourier coefficients of a specified output.

[4] CANCER and the first version of SPICE do not have a timestep control. However, SPICE2 does have a timestep control.

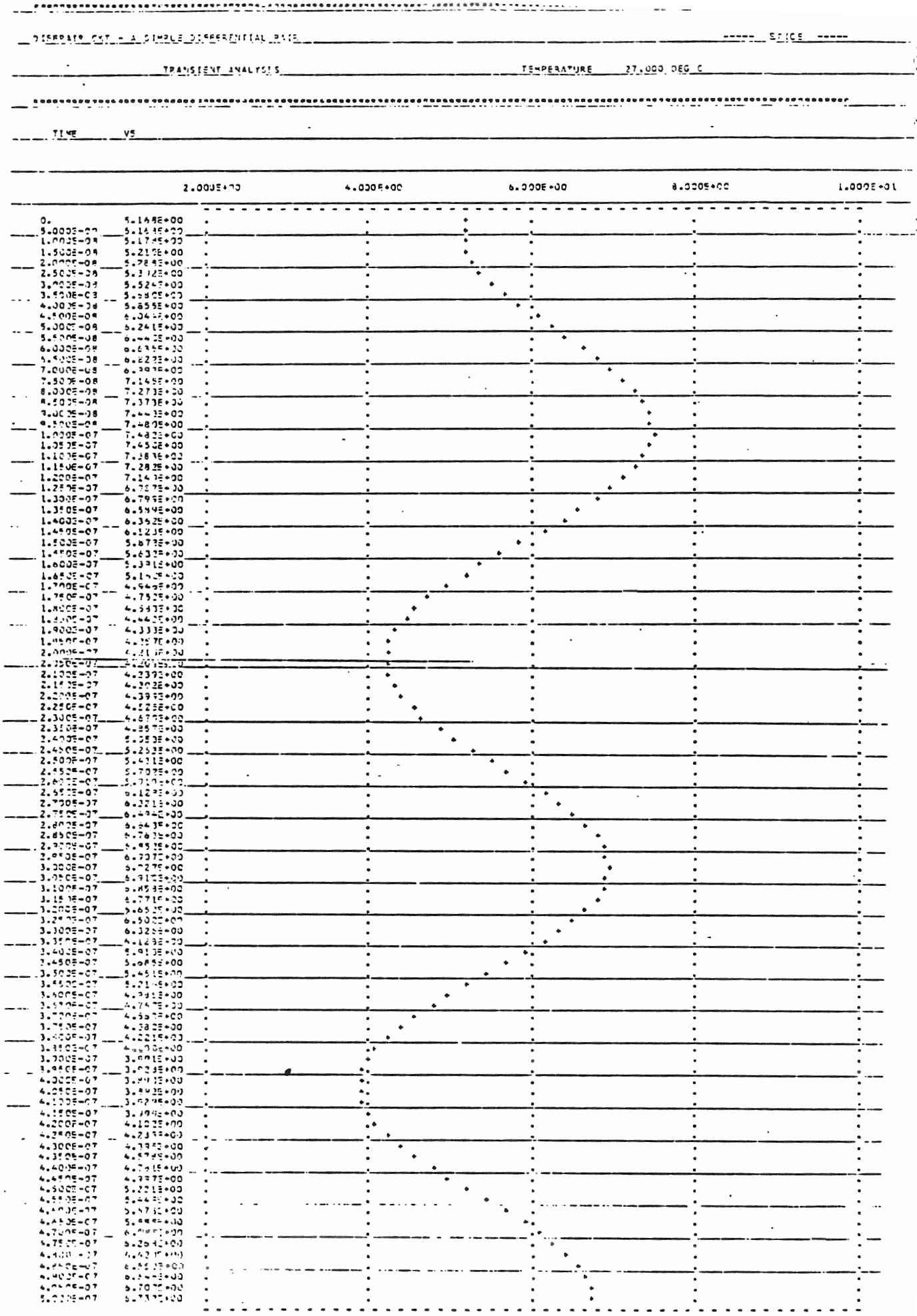


Fig. 1.7. SPICE Transient Analysis Solution.

This capability is useful for estimating the Fourier harmonic distortion components of a near-sinusoidal waveform. However, the inherent inaccuracies of fitting a Fourier series to a time-domain waveform may limit the usefulness of this subanalysis capability to relatively large values of harmonic distortion.

1.7. AC Analysis

The small-signal, frequency-domain analysis portion of SPICE is useful for the design of analog circuits that operate in small-signal modes. The perturbational response of the circuit is obtained by using linearized models for the nonlinear elements. The parameters for these linearized models are determined by the dc operating point analysis. The small-signal linear equivalent circuit is analyzed in the frequency domain with the phasor method [7]. All circuit voltages and currents are complex variables that usually are expressed in terms of magnitude and phase. Independent sources also have complex values which are expressed in terms of magnitude and phase.

An ac analysis usually is performed at successive values of frequency to determine the frequency response of circuit transfer functions. The specified voltage and current outputs are stored at each frequency point. At the conclusion of the analysis, these outputs can be listed or plotted as magnitude, magnitude in dB, phase, real part, or imaginary part. The plots of the magnitude and phase of the voltage at node 5 of the differential-pair circuit are given in Fig. 1.8 and Fig. 1.9, respectively. The frequency ranges from 1 Hz to 1 GHz, and the plot resolution is ten points per decade. The input source, VIN, is assigned a value of unity magnitude and zero phase.

DIFFAMP CKT - A SIMPLE DIFFERENTIAL PAIR ----- SPICE -----

AC ANALYSIS TEMPERATURE 27.000 DEG C

FREQUENCY MAGNITUDE OF V5

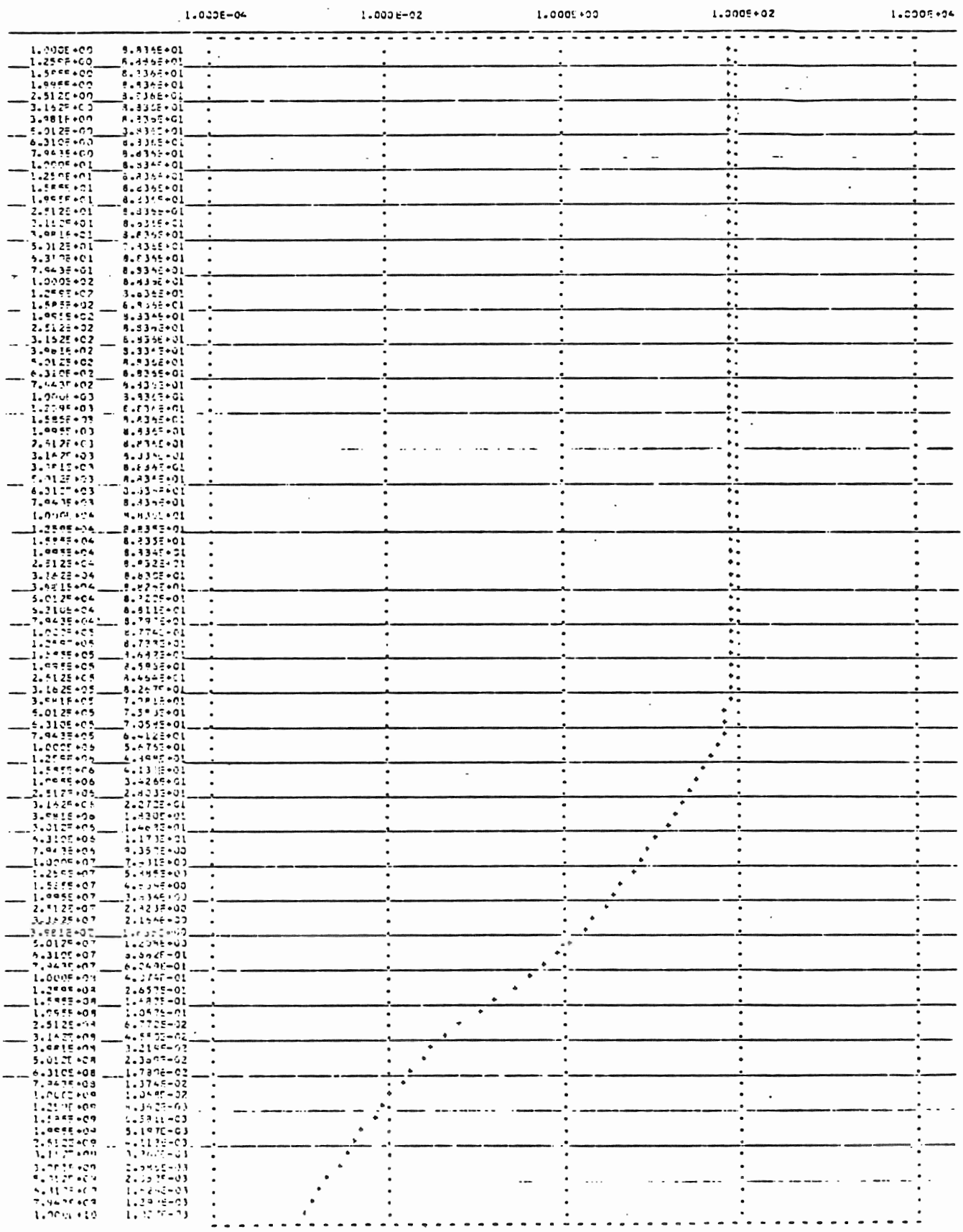


Fig. 1.8. SPICE Magnitude Response in AC Analysis.

DIFFPAIR CKT - A SIMPLE DIFFERENTIAL PAIR ----- SPICE -----

AC ANALYSIS TEMPERATURE 27.000 DEG C

FREQUENCY PHASE OF VR (DEGREES)

-1.800E+02 -9.000E+01 0. 9.000E+01 1.800E+02

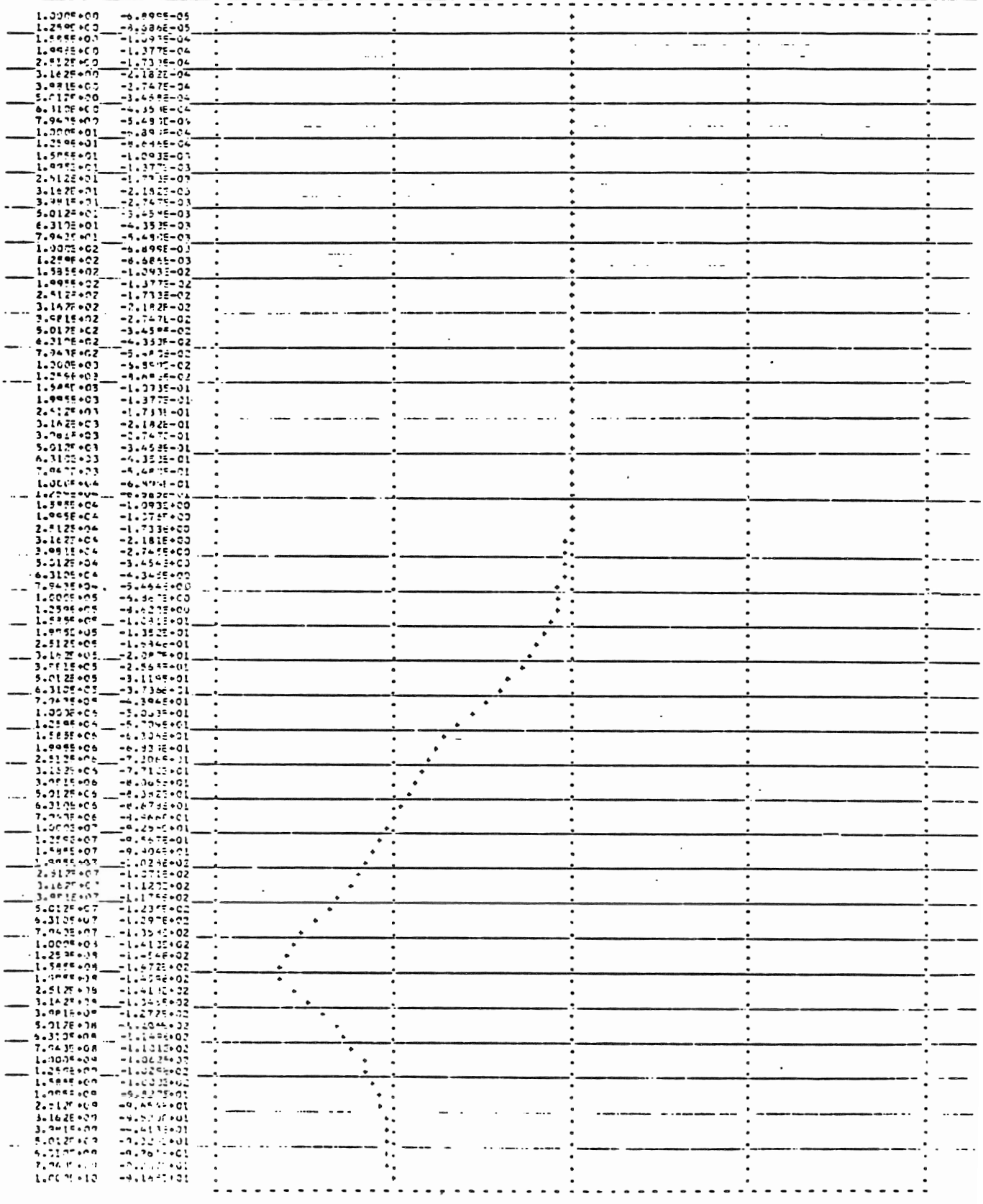


Fig. 1.9. SPICE Phase Response in AC Analysis.

1.8. Noise Analysis

The ac analysis portion of SPICE contains the added capability of evaluating the noise characteristics of an electronic circuit [3,8,9]. Each resistor in a circuit generates thermal noise, and each semiconductor device in the circuit generates both shot noise and flicker noise in addition to the thermal noise that is generated by the device ohmic resistances. This noise generation is modeled by equivalent independent sources that are associated with each noise-generating element. The noise models that are used in SPICE are given in Appendix 2 of this thesis.

Because the equivalent noise sources in the circuit are statistically uncorrelated, the contribution of each noise source to the total noise at a specified output must be computed separately. The total output noise is the RMS sum of the individual noise contributions. The noise analysis is performed in an efficient manner by employing the adjoint network concept [8,10] to evaluate this RMS sum with only one additional ac solution at each frequency point.

After the noise analysis has been completed, both the total output noise and the equivalent input noise can be presented either as tabular listings or as line-printer plots. In addition the contribution of each noise source in the circuit can be printed at selected frequency points.

1.9. Distortion Analysis

For relatively large levels of distortion, the method of fitting a Fourier series of a periodic waveform ot the time-domain

response yields a good approximation of the distortion components of the waveform. Small levels of distortion are determined more accurately by frequency-domain distortion analysis [11,12]. The distortion analysis that is implemented in SPICE presently is capable of evaluating second-order and third-order harmonic distortion as well as second-order and third-order intermodulation distortion.

The distortion analysis is performed by approximating the model equations for each nonlinear element by a Volterra series [13]. Each nonlinear element then can be modeled by the linearized equivalent element and an independent source that represents the distortion that is generated by that particular element. Distortion analysis therefore is analogous to noise analysis. For a particular distortion component, the small-signal circuit contains several "distortion generators," and the total distortion is summed at a particular output. However, the total distortion component is the vector sum, instead of the RMS sum, of the individual distortion contributions.

At the conclusion of the distortion analysis, any of the distortion components can be listed or plotted as a function of frequency. The contributions of every distortion source also can be printed at selected frequency points.

1.10. The Design of SPICE

The circuit elements and analysis capabilities that are included in SPICE reflect the needs of one particular integrated-circuit design group over a period of several years. Obviously, there are many simulation capabilities that are not included in

SPICE. For example, the capability of evaluating the poles and zeros of a small-signal ac transfer function is not included in SPICE because our SLIC program [14,15] already contains this feature. Since time-domain steady-state analysis [16-18] is implemented in our SINC program,[5] this capability also was not included in SPICE.

Many simulation programs include a statistical analysis [20-22] that evaluates the performance of a circuit for a random sampling of circuit parameters. In addition, many programs have been developed that use a simulation program for automated design optimization [23-27]. No new circuit analysis capabilities are required for statistical analysis or automated design. However, since both of these capabilities require a large number of repetitive circuit analyses, the efficiency of the analysis program is of utmost importance in the implementation of either a statistical analysis program or an automated design program.

The design of a simulation program is influenced by guidelines that depend upon the pattern of usage of the program. The guidelines that influenced the design of the SPICE program are adopted as a method of comparison in this thesis. These guidelines are, more or less in their order of priority: ease of use, efficiency, simplicity, and generality.

Clearly, a simulation program must be easy to use. Circuit designers by and large are not computer programmers, so the use of a simulation program should require a minimal knowledge of programming

[5] SINC is a nonlinear transient analysis program that was developed by S. P. Fan at the Electronics Research Laboratory, University of California, Berkeley. The SINC program evolved from the TIME program [19].

techniques. After all, a simulation program ideally parallels the laboratory workbench.

The efficiency of a simulation program determines the dollar cost of computer simulation. The computer resources, such as execution time, central memory, and disc storage, should be minimal if the simulation program is to be inexpensive to use. Moreover, any circuit design project requires the repetitive analysis of the circuit for different parameter values, different temperatures, and so on. This need for repetitive analysis also is present for the case of statistical analysis or automated design. Therefore, the simulation program must, as a very high priority, require a minimal amount of computer resources to perform a circuit simulation.

The structure of the simulation program, and the computational methods that are used in the simulation program, should be as simple as possible to insure ease of modification, enhancement, and support. Sophisticated and complicated programming methods therefore should be used only if they result in a commensurate increase in program efficiency or ease of use. The need for generality in a program usually conflicts with the need for simplicity and the need for efficiency. For the most part, the SPICE program is designed with generality as a much lower priority than simplicity or efficiency.

The remaining chapters of this thesis concentrate on the topics of circuit theory and numerical analysis that are central to the analysis portion of a circuit simulation program. The algorithms that are employed in SPICE are presented in detail, and the performance of these methods is compared with the performance of other available techniques. Since there is no "right" way to develop a simulation

program, the choice and implementation of the analysis methods in the SPICE program is justified on the basis of the guidelines set forth in this chapter. Clearly, different simulation needs and priorities often will result in a different choice or implementation of the various computational algorithms.

Chapter 2 of this thesis presents an overview of a circuit simulation program and presents the computational methods that are employed in the analysis subprogram. Chapter 3 presents the methods of equation formulation that are used to represent the circuit by a system of circuit equations. In Chapter 4, the methods of solving a system of linear equations are described. Chapter 5 presents and compares the methods of iteratively solving a system of nonlinear equations. Finally, Chapter 6 presents the methods of numerical integration that are necessary to perform a transient analysis. In Chapter 7, an overall summary of the algorithms that are implemented in SPICE is presented, together with some suggestions for further work.

Appendix 1 details the circuit examples that are used as a method of comparison of the analysis algorithms. Appendix 2 describes the element models that have been adopted in the SPICE program. The input syntax for the SPICE program is detailed in Appendix 3. Appendix 4 describes the implementation of the SPICE2 program. Finally, the entire listing of the SPICE2 program is given in Appendix 5.

II. OVERVIEW OF CIRCUIT SIMULATION

Although circuit simulation programs differ considerably in size and capability, the structure of most simulation programs is similar. This structure is illustrated schematically in Fig. 2.1. As this figure shows, a simulation program consists of five major subprograms: input, setup, analysis, output, and utility. The subprograms interact with each other through a data structure that is stored in a common-block area of the program.

The input subprogram reads the input file, constructs a data structure from this input, and checks the data structure for obvious user errors. The data structure, after the input phase, contains a complete, self-consistent description of the circuit.

After the input subprogram has executed successfully, the setup subprogram constructs additional data structures that are required by the analysis subprogram. Some simulation programs do not contain a setup subprogram. The setup subprogram in SPICE constructs the pointer system that is used by the sparse matrix subroutines of the analysis subprogram. Other programs, such as ASTAP [28,29], generate and compile a FORTRAN program that performs the analysis as a part of the setup procedure.

The analysis subprogram is the major part of the simulation program. This subprogram performs the circuit analyses that are specified in the input file. The output from these analyses are stored in central memory or on disc for later processing by the output subprogram.

Finally, the output subprogram generates the output that is specified by the user. Most simulation programs contain the

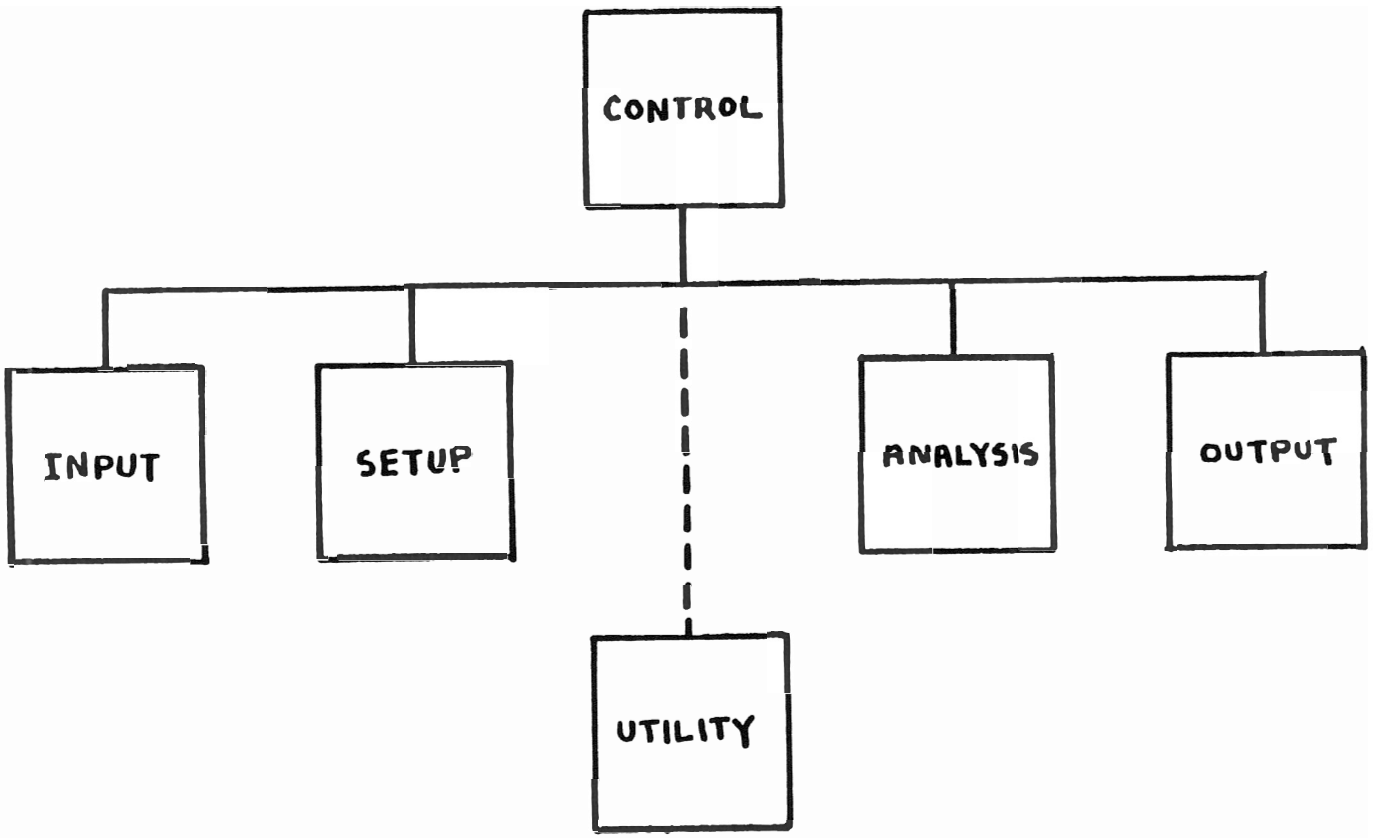


Fig. 2.1. Block Diagram of a Circuit Simulation Program.

capability of producing line-printer plots of the output data as well as tabular listings. Some programs, such as ISPICE [30], contain a graphics-terminal plot capability and the ability to perform extensive post-processing on the simulation output.

Many simulation programs, including ASTAP, ISPICE, ECAP2 [31,32], NET2 [33], and CIRCUS2 [34], contain a utility subprogram which creates and updates a disc library of circuits and element models. This capability allows users to store and retrieve circuits, and allows for several design engineers to share device model information. SPICE does not contain a utility subprogram because a disc library is not necessary for the usage pattern of SPICE at the University of California, Berkeley.

All of the subprograms merit careful attention in the development of a circuit simulation program. Since the program user interacts with the input and output subprograms, the flexibility and ease of use of these two portions of the program determines, to a large extents, the flexibility and ease of use of the entire simulation program.

Although the input and output subprograms determine how easy the simulation program is to use, the analysis subprogram plays the dominant role in determining the efficiency of the program. Moreover, the algorithms that are employed in the analysis subprogram are more complex than the algorithms in the other subprograms of the simulation program. For these reasons, the largest portion of the development effort that is expended on a simulation program is devoted to the implementation of reliable and efficient circuit analysis techniques.

2.1. Circuit Analysis

The analysis portion of a circuit simulation program determines the numerical solution of a mathematical representation of the circuit. To accomplish the transition from the physical circuit to a mathematical system of equations, each element in the circuit is represented by a mathematical model. The element models that are implemented in SPICE are described in Appendix 2 of this thesis. Several years of practical experience has shown that these models are adequate for most simulation problems. However, other element models also could be used.

After each element in the circuit is modeled, the system of equations that describe the complete circuit is determined by the model equations for each element and topological constraints that are determined by the interconnection of the elements. The topological constraints reflect Kirchoff's Current Law (KCL) and Kirchoff's Voltage Law (KVL). The circuit equations are, in general, a system of algebraic-differential equations of the form

$$F(x,\dot{x},t) = 0 \qquad (2.1)$$

where x is the vector of unknown circuit variables, $\dot{x}$ is the time-derivative of x, and F is, in general, a nonlinear operator.

Circuit analysis determines the solution of (2.1) for different special cases that are of interest in the design of an electronic circuit. A dc analysis, for example, usually is invoked to determine the equilibrium solution of (2.1). In this case, the vector $\dot{x}$ is zero, and (2.1) is simplified to a system of nonlinear equations.

The remainder of this chapter introduces the numerical analysis methods that are necessary to perform dc analysis, ac analysis, and transient analysis. These numerical methods are: equation formulation, linear equation solution, nonlinear equation solution, and numerical integration. Implementation of the analysis portion of a circuit simulation program requires the development of efficient and reliable algorithms to perform these four basic numerical procedures.

2.2. Linear DC Analysis

The most rudimentary analysis in a circuit simulation program is the dc solution of a linear circuit. Usually, the dc solution is determined for the equilibrium case, that is, for the case when the vector $\dot{x}$ is zero. Energy storage in the circuit then is ignored by treating capacitors as open circuits and inductors as short circuits.[1] Since all time-derivatives are zero, and the circuit is linear, the circuit equations for the dc analysis are a linear system of algebraic equations. The dc analysis of a linear circuit therefore requires only a method of formulating the circuit equations and a linear solution algorithm to solve the equations.

There are many different methods of formulating the circuit equations. Nodal Analysis [7] probably is the simplest computational

---

[1] If initial conditions that do not correspond to the equilibrium solution are specified, then energy-storage elements are treated either as voltage sources or current sources. For example, if the initial capacitor voltages and inductor currents are specified, then capacitors are treated as voltage sources and inductors are treated as current sources. In SPICE, the capacitor currents and inductor voltages are the initial conditions. Hence, in dc analysis capacitors are treated as current sources and inductors are treated as voltage sources.

method. With Nodal Analysis, the nondatum (ungrounded) node voltages are the unknown circuit variables. This choice of variables automatically insures that the circuit solution satisfies KVL. The circuit equations then are determined by writing an equation for KCL at each nondatum node.

If the capacitors in the circuit in Fig. 2.2 are ignored, Nodal Analysis produces the following system of equations:

$$\begin{bmatrix} \frac{1}{R_1} + \frac{1}{R_2} & -\frac{1}{R_2} \\ -\frac{1}{R_2} & \frac{1}{R_2} + \frac{1}{R_3} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_0 \\ 0 \end{bmatrix} \tag{2.2}$$

In general, Nodal Analysis produces the system of Nodal equations

$$Yv = j \tag{2.3}$$

where Y is the Nodal Admittance matrix, v is the vector of node voltages, and j is the current excitation vector.

The principal virtue of Nodal Analysis is its computational simplicity. For the dc analysis of a circuit that contians only resistors and independent current sources, Y and j are constructed by the following rules: y_{ii} is the sum of all conductances connected to node i, y_{ij} is the negative sum of all conductances connected between node i and node j, and j_i is the sum of all independent current sources that flow into node i. Clearly, the formulation of (2.3) for this special case of linear dc analysis can be implemented in ten or twenty lines of FORTRAN code.

The remaining task for linear dc analysis is the solution of (2.3). This is accomplished most efficiently by the equivalent

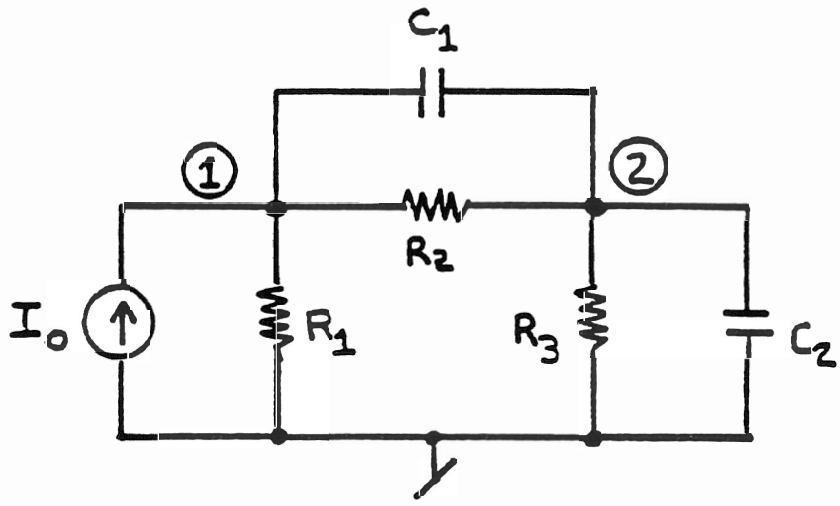


Fig. 2.2. An Example Circuit.

methods of Gaussian elimination or LU factorization [35]. For most circuit applications, the matrix Y in (2.3) contains few nonzero terms; that is, Y is very sparse. The computational effort that is required to solve (2.3), is diminished considerably if this inherent sparsity is recognized [36]. For this reason, sparse-matrix methods were implemented in the earliest version of CANCER and were retained in all versions of CANCER and SPICE. In fact, sparse-matrix methods are used in most simulation programs that have been developed recently.

2.3. Linear AC Analysis

Small-signal ac analysis is slightly more complicated than linear dc analysis. Since all elements in the circuit are, by definition, linear, nonlinear circuit elements are modeled by equivalent linearized models. The parameter values for the linearized models, in turn, are determined by a dc operating point analysis. The linearized models for the four nonlinear elements in SPICE are described in Appendix 2 of this thesis.

AC analysis determines the small-signal solution of the circuit in sinusoidal steady-state. All input sources are sinusoidal with the same input frequency, although sources may be assigned different values of relative phase. Since the circuit is in sinusoidal steady-state, the phasor method is invoked to transform the differential circuit equations into the frequency domain [7]. The admittance of a resistor, capacitor, and inductor are given by

$$Y_R = 1/R \tag{2.4}$$

$$Y_R = j\omega C \tag{2.5}$$

$$Y_I = 1/j\omega L \tag{2.6}$$

where ω is the value of the input frequency in radians.

The equations for a linear ac analysis are assembled by the same method that is used for a linear dc analysis, except, of course, that the circuit equations are complex for ac analysis. For example, the Nodal equations for the circuit in Fig. 2.2 are

$$\begin{bmatrix} \frac{1}{R_2} + \frac{1}{R_3} + j\omega C_1 & -\frac{1}{R_3} - j\omega C_1 \\ -\frac{1}{R_3} - j\omega C_1 & \frac{1}{R_3} + j\omega(C_1+C_2) \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_0 e^{-j\phi} \\ 0 \end{bmatrix} \tag{2.7}$$

where ϕ is the relative phase of the input source.

Usually, the value that is assigned to a particular source is different for dc analysis than for ac analysis. Power supplies, for example, usually have zero ac values whereas input sources often have zero dc values. If the circuit has only one ac input, then it is convenient to set that input to unity amplitude and zero phase. The value of each circuit variable in ac analysis then is equal to the value of the transfer function of that variable with respect to the input.

The implementation of an ac analysis capability in a simulation program is similar to the implementation of linear dc analysis. The circuit equations in ac analysis are, of course, complex. However, the same formulation and linear solution algorithms that are used for dc analysis may be used for ac analysis. Moreover, no new methods are required to implement ac small-signal analysis.

2.4. Nonlinear DC Analysis

If the circuit contains elements that are modeled by nonlinear equations, then the dc solution is obtained by an iterative sequence of linearized solutions. The Newton-Raphson algorithm [37,38] is the most common method of linearization. This method approximates each nonlinearity in the circuit by a Taylor series that is truncated after the first-order term.

For example, consider the diode characteristic that is given in Fig. 2.3. The diode current is modeled by the familiar diode equation[2]

$$I_D = I_S[\exp(\frac{V_D}{V_t}) - 1] \tag{2.8}$$

where the polarity of I_D and V_D are shown in Fig. 2.4a. At any operating point V_o, the diode characteristic is approximated by a tangent at V_o, as shown in Fig. 2.3. This approximation is equivalent to modeling the diode by a parallel connection of an equivalent linear conductance, g_{eq}, and an equivalent independent current source, I_{eq}. The value of g_{eq} is the derivative of (2.8) with respect to V_D:

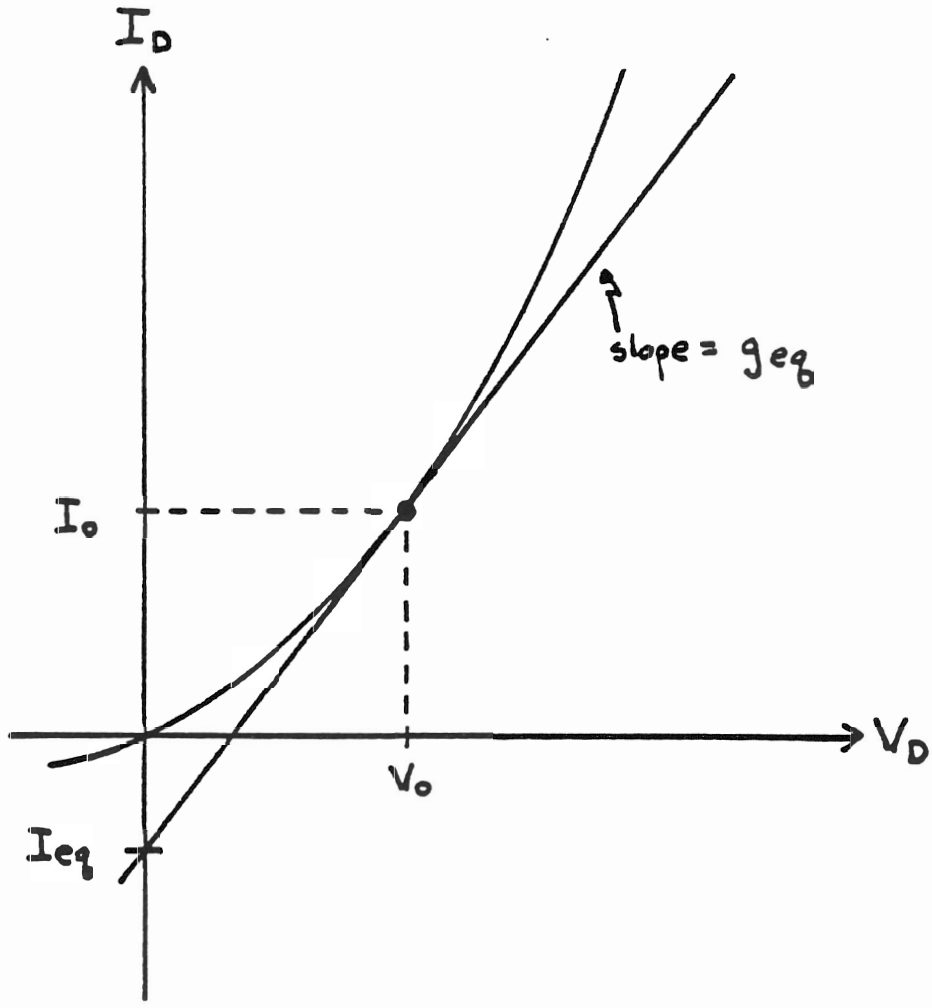
$$g_{eq} = \frac{I_S}{V_t} \exp(\frac{V_D}{V_t}) \tag{2.9}$$

The value of I_{eq} is given by

$$I_{eq} = I_D - g_{eq}V_D \tag{2.10}$$

---

[2] The parameter V_t is the thermal voltage and is equal to kT/q, where k is Boltzmann's constant, q is the electronic charge, and T is the absolute temperature in degrees Kelvin. At room temperature, V_t is approximately 26 mV.



Fog. 2.3. The Ideal Diode Characteristic.

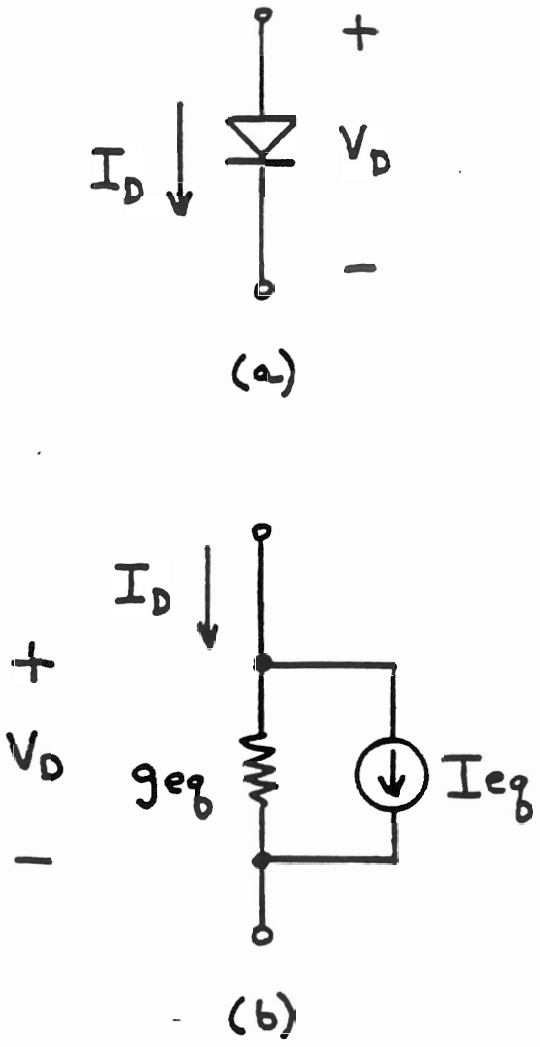


Fig. 2.4. (a) Diode Polarity Definition

(b) Linearized Equivalent Circuit

The circuit in Fig. 2.5a illustrates the linearization procedure. For the dc solution, the two capacitors are ignored. The diode is modeled by its linearized equivalent to produce the circuit that is shown in Fig. 2.5b. The linearized Nodal equations for this circuit are

$$\begin{bmatrix} \frac{1}{R_1} + \frac{1}{R_2} & -\frac{1}{R_2} \\ -\frac{1}{R_2} & \frac{1}{R_2} + g_{eq} \end{bmatrix} \begin{bmatrix} V_1 \\ V_2 \end{bmatrix} = \begin{bmatrix} I_o \\ -I_{eq} \end{bmatrix} \qquad (2.11)$$

where g_{eq} and I_{eq} are determined by (2.9) and (2.10).

The iterative solution sequence begins with an initial "guess" for the solution. All nonlinear model equations then are linearized about this presumed operating point. The circuit equations for this linear equivalent circuit are formulated and solved by the same procedure that is used for linear dc analysis.[3] The circuit solution then is used as the next "guess," and the process is repeated. The iteration stops when successive iterate solutions agree to within some tolerance.

Nonlinear dc analysis is a more complicated task than is linear dc analysis. First, the provision for evaluating the nonlinear model equations and their derivatives must be added to the program. Second, the linearized circuit equations must be assembled and solved many times to obtain the solution. In practice, 5 - 30 Newton

---

[3] It is assumed that the process of linearizing the model equations and then formulating the system of circuit equations yields the same results as formulating the system of nonlinear circuit equations and then linearizing the entire set of equations. It can be shown [37] that this assumption is valid if roundoff errors are negligible.

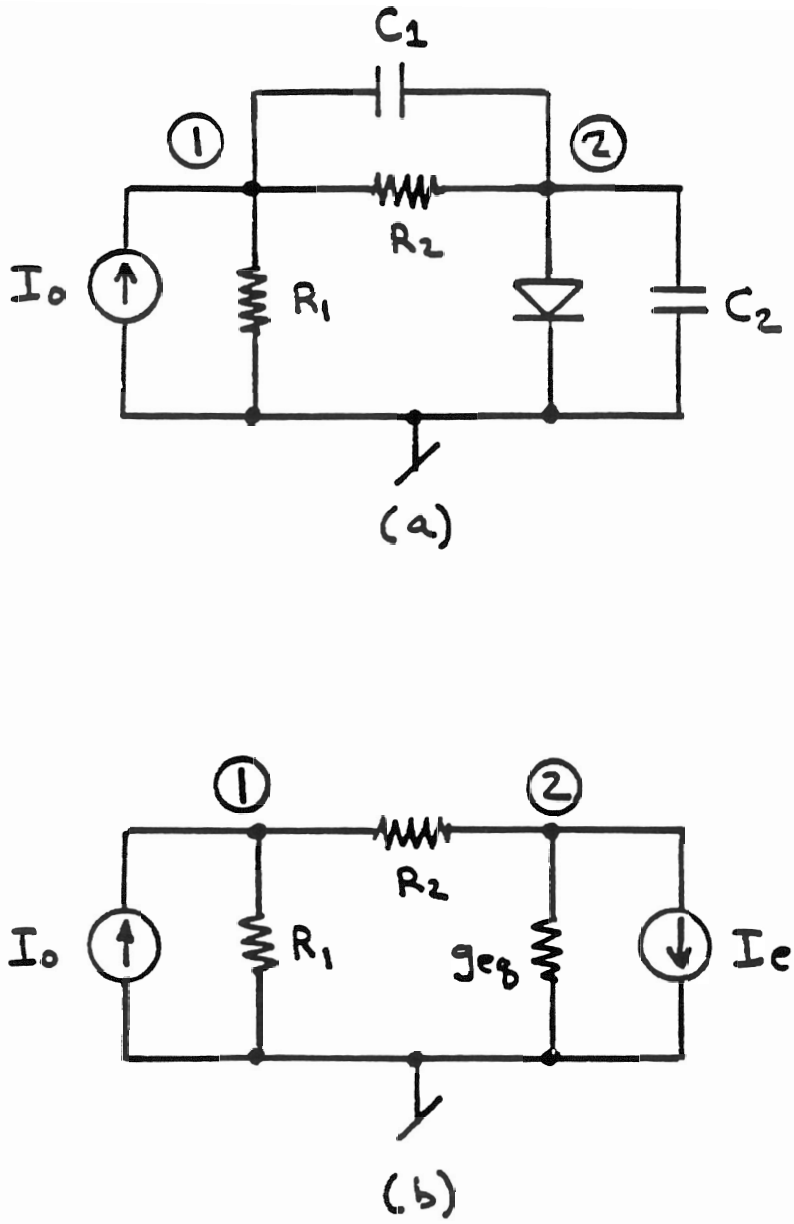


Fig. 2.5. (a) An Example Circuit

(b) Linearized Equivalent Circuit

iterations are required to determine the dc operating point of most circuits.

The iterative nature of the solution process also admits the possibility of nonconvergent solutions. If the algorithm fails to converge to a solution for a particular circuit, the entire simulation program is of no help in the design of that circuit. Therefore, a reliable and efficient nonlinear solution method is of crucial importance in a circuit simulation program.

2.5. Transient Analysis

Transient analysis determines the time-domain response of the circuit over a specified time interval (0,T). The initial timepoint arbitrarily is defined as time zero. This initial solution either is specified by the user, or, more conveniently, is determined by a dc operating point analysis.

The transient solution is determined computationally by dividing the time interval (0,T) into discrete timepoints $(0,t_1,t_2,\ldots T)$. At each timepoint, a numerical integration algorithm is employed to transform the differential model equations of each energy-storage element into equivalent algebraic equations. After this transformation, the timepoint solution is determined iteratively in the same fashion that the nonlinear dc operating point is determined.

Numerical integration algorithms can be either explicit or implicit; however, implicit methods are far superior for circuit simulation. The Backward Euler algorithm is the simplest implicit method. This method is defined by the relation

$$x_{n+1} = x_n + h_n \dot{x}_{n+1} \tag{2.12}$$

where $x_{n+1} = x(t_{n+1})$, $x_n = x(t_n)$, and the timestep, h_n, is defined by the equation

$$h_n = t_{n+1} - t_n \tag{2.13}$$

Consider, as an example, a capacitor which is modeled by the equations

$$Q_c = f(V_c) \tag{2.14}$$

$$I_c = \dot{Q}_c \tag{2.15}$$

where the polarity of I_c and V_c are shown in Fig. 2.6. If (2.12) is applied to (2.15), the following algebraic equation is obtained:

$$I_{n+1} = \frac{f(V_{n+1})}{h_n} - \frac{Q_n}{h_n} \tag{2.16}$$

If (2.14) is a nonlinear equation, then (2.16) clearly will be a nonlinear equation.

The circuit shown in Fig. 2.7a provides an example of the integration process. Capacitor C1 is a linear capacitor, and C2 is a nonlinear diffusion capacitance that is defined by the charge equation

$$Q_2 = \tau I_S[\exp(\frac{V_{C2}}{V_t}) - 1] \tag{2.17}$$

The diode in the circuit is defined by (2.8) as before.

When (2.12) is applied to the model equations for C1 and C2, the "quasi-dc" circuit that is shown in Fig. 2.7b results. The model equations for the elements Q1 and Q2 are given by the

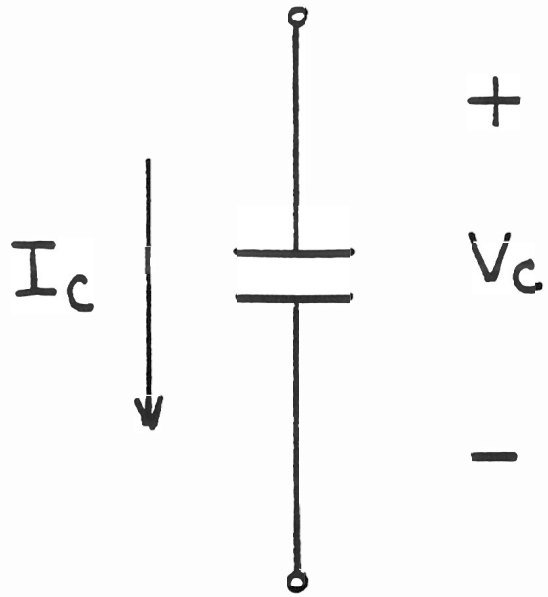


Fig. 2.6. Polarity Definition for a Capacitive Branch.

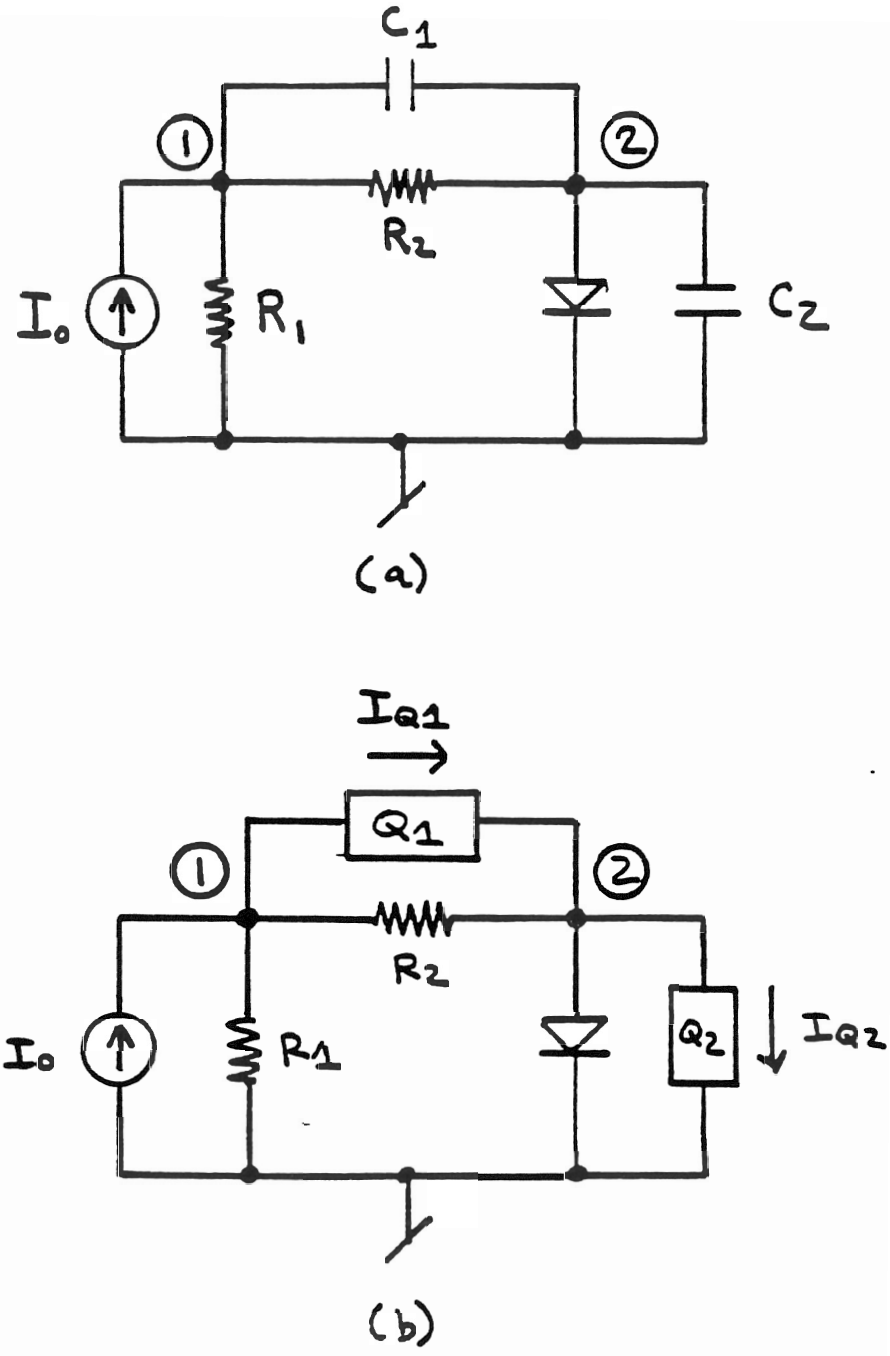


Fig. 2.7. (a) Example Circuit for Transient Analysis

(b) Equivalent "Quasi-DC" Circuit

$$I_{Q1}(t_{n+1}) = \frac{C_1}{h_n} [V_{Q1}(t_{n+1}) - V_{Q1}(t_n)] \qquad (2.18)$$

$$I_{Q2}(t_{n+1}) = \frac{\tau I_S}{h_n} \left[\exp\left(\frac{V_{Q2}(t_{n+1})}{V_t}\right) - 1\right] - \frac{Q_2(t_n)}{h_n} \qquad (2.19)$$

Determining the solution of the circuit in Fig. 2.7a at each timepoint therefore is equivalent to determining the dc solution of the circuit in Fig. 2.7b.

The error of the approximation of (2.12) is the local truncation error (LTE) of the integration method. This error, in turn is, proportional to the timestep h_n. If (2.12), and therefore (2.16), are to be accurate approximations, then the timestep must be sufficiently small that the local truncation error is negligible.

Addition of a transient analysis capability to a simulation program therefore requires two program additions. First, the transformation of energy-storage differential model equations to equivalent "quasi-dc" algebraic equations must be implemented. Second, provision must be made for estimating the local truncation error of the integration algorithm and adjusting the timestep to maintain reasonable levels of solution error.

2.6. Summary

A circuit simulation program contains several important subprograms, and each of these subprograms require careful attention in the overall design of the simulation program. The analysis portion is the central part of the simulation program. This subprogram consumes the largest fraction of total computational effort for a simulation. Moreover, the analysis subprogram requires more

sophisticated numerical techniques than do the other portions of the program. The development of a simulation program, therefore, depends in a vital manner on the development of an efficient analysis subprogram.

The analysis subprogram, in turn, requires four basic computational algorithms: equation formulation, linear equation solution, nonlinear solution, and numerical integration. The role of these four methods is illustrated in the flowchart that is shown in Fig. 2.8. This figure depicts a transient analysis with a fixed timestep.

At each timepoint, time is incremented and the time-derivative components in the model equations are approximated with numerical integration methods. A series of Newton iterations then is required to determine the solution at the timepoint. Each Newton iteration, in turn, requires the linearization of the model equations, followed by the formulation and solution of the linearized circuit equations. After convergence is obtained, the timepoint solution is stored and the next timepoint is attempted. After all the timepoints have been solved, the simulation results are printed and the program stops.

For a dc analysis, the outer time loop in Fig. 2.8 is not used; once the solution is obtained, the program prints the dc operating point and stops. Moreover, if the circuit contains only linear elements, the inner Newton iteration loop is eliminated since the solution always is obtained in one iteration.

The relative computational effort that is required for the various types of circuit analysis for SPICE is illustrated in

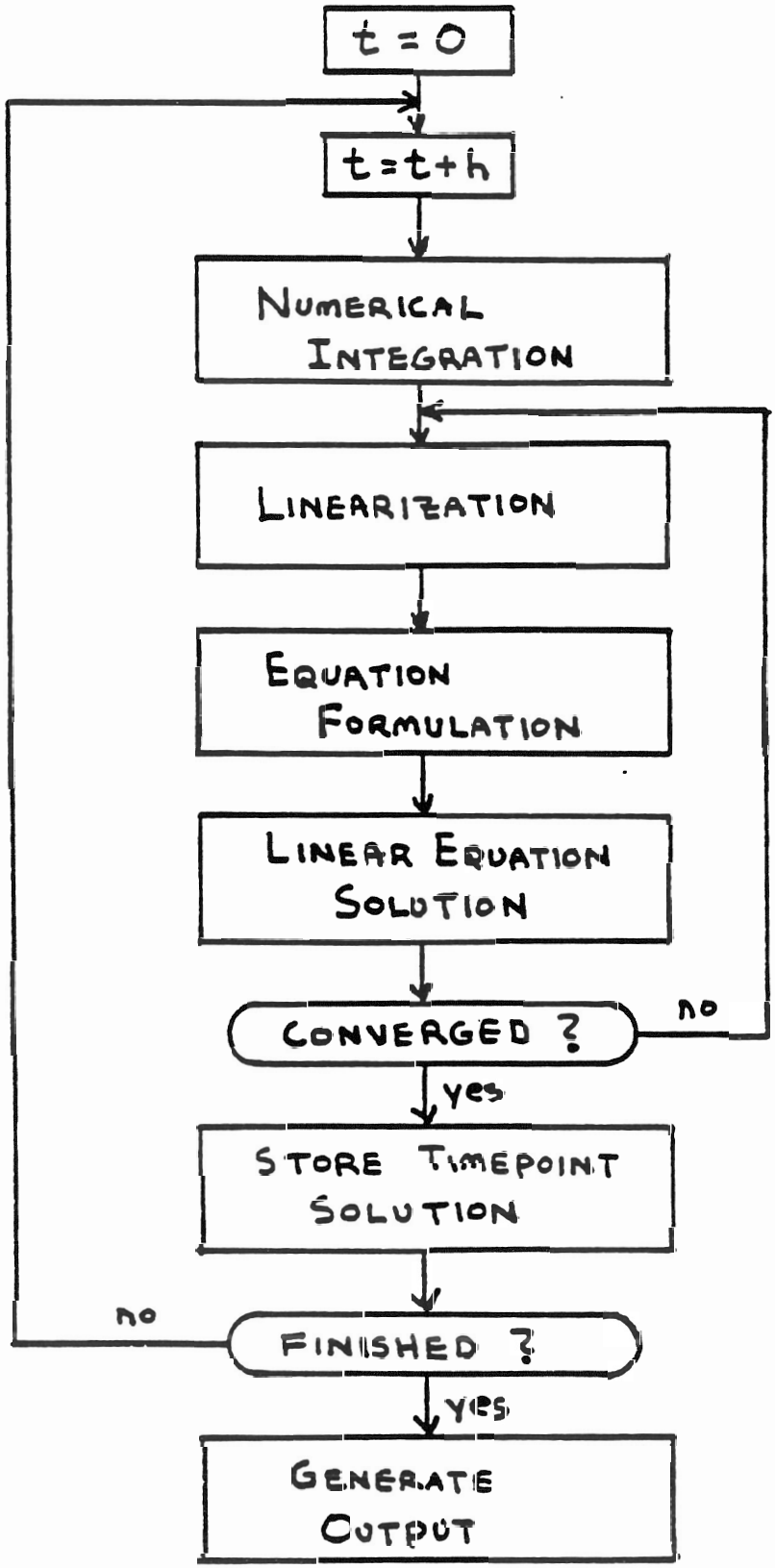


Fig. 2.8. Flowchart of a Transient Analysis Program.

Table 2.1. This table gives the points, iterations, and cpu execution time to determine the dc operating point, the dc transfer characteristics, the small-signal frequency response, and the transient response of a differential pair circuit. The simulation output for these analyses is presented in Chapter 1 of this thesis. This circuit contains 14 nodes, 4 transistors, 5 resistors, and 3 voltage sources.

The dc operating point solution requires 8 Newton iterations and 0.181 cpu seconds of execution time. A substantial portion of this cpu time is consumed in the printing of the operating point. The dc transfer curve solution for 101 points requires 233 Newton iterations and 2.20 cpu seconds of execution time. No output time is included in the dc transfer curve solution time. Hence, the computational effort in dc analysis is about 9.4 ms per Newton iteration. The ac analysis for 101 frequency points required 1.495 cpu seconds, or approximately 14.8 ms per iteration. The transient analysis, for 49 timepoints, requires 141 iterations and 1.990 cpu seconds, or approximately 14.1 ms per Newton iteration.

	Points	Iterations	CPU Time (Sec)
DC Operating Point	1	8	0.181
DC Transfer Characteristics	101	233	2.200
AC Analysis	101	101	1.495
Transient Analysis	49	144	1.990

Table 2.1. Comparison of Iterations and Cpu Time for Various Analysis Methods

III. EQUATION FORMULATION

Computer simulation of an electronic circuit involves the numerical analysis of mathematical model of the circuit. The mathematical model, therefore, is the first order of business in any circuit analysis. The physical circuit consists of an interconnection of circuit elements. The mathematical model of the circuit, on the other hand, is a system of equations. The solution of these equations, for specified special cases, simulates specified electrical characteristics of the circuit.

To determine the system of equations that describe the circuit, each circuit element first is modeled in mathematical terms. This modeling step requires that each element be represented by an interconnection of ideal network branches. Every branch, in turn, is specified by a mathematical relation that defines the branch voltage or current as a function of the unknown circuit variables. A resistor, for example, is defined by a linear equation that relates the branch current and the branch voltage via a single resistance parameter; namely, the branch current, I_R, is related to the branch voltage, V_R, by the equation $I_R R = V_R$, where R is the value of resistance.

The system of equations that model the electronic circuit must satisfy two constraints. First, each branch relation must be satisfied at each instant in time. Second, Kirchoff's topological laws, KCL and KVL, must be satisfied at each instant in time.

The constraint that each branch relation be satisfied at each instant of time implies a system of algebraic-differential equations

of the form

$$B(x,\dot{x},t) = 0 \tag{3.1}$$

where the vector x contains the branch voltages and currents, $\dot{x}$ is the time derivative of x, and B is, in general, a nonlinear operator.

Kirchoff's topological laws, KCL and KVL, imply a linear system of equations

$$T\,x = 0 \tag{3.2}$$

where the matrix T contains coefficients that are +1, -1, or zero.

If a particular circuit contains b branches, then (3.1) is a system of b equations and (3.2) also is a system of b equations [7]. A straightforward combination of (3.1) and (3.2) will produce a system of 2b algebraic-differential equations of the form

$$F(x,\dot{x},t) = 0 \tag{3.3}$$

where F is, in general, a nonlinear operator.

Fortunately, it is not necessary to include all of the branch voltages and currents into the vector x. In fact, x may contain any combination of circuit variables that result in an independent system of equations. The wide choice of unknown vectors leads to the wide variety of equation formulation methods that are available.

As an example, a 100 node, 100 transistor circuit might be modeled by a 700 branch, 100 node network model. A straightforward combination of (3.1) and (3.2) would lead to a system of 1400 equations. However, if the Nodal Analysis method of equation formulation were

used, the resultant mathematical model would be comprised of only 100 equations. The complexity of the mathematical model, therefore, is governed by the method that is used to formulate the model.

This chapter presents the different methods of equation formulation. First, ideal branch relations are presented. Next, the constraints that are implied by Kirchoff's Laws are formalized in the form of (3.2). Once (3.1) and (3.2) are established, the primary methods of Cutset Analysis and Loop Analysis are introduced. Both of these methods impose limitations on the types of branch relations that can be included in the circuit. Practical formulation methods, therefore, are combinations of these two methods.

Currently available simulation programs employ one of three distinct formulation methods. A modification of Nodal Analysis is used in SPICE, CANCER [2,3], SLIC [14,15], TIME [19], TRAC [39], and ECAP [40]. The second popular formulation technique, Hybrid Analysis, is used in ASTAP [28,29], ECAP2 [31,32], NET2 [33], CIRCUS2 [34], CIRPAC [41,42], SCEPTRE [43], NET [44], and CIRCUS [45]. The third formulation algorithm, which has not yet seen widespread use in simulation programs, is the Sparse Tableau formulation of Hachtel, et al. [46].

In practice, the circuit equations are never formulated in the form of (3.3). Instead, the branch relations (3.1) are replaced by an equivalent system of linear equations at each iteration in a dc analysis, at each frequency point in an ac analysis, and at each iteration of each timepoint in a transient analysis. When the equation formulation algorithm is applied to this equivalent system of linear equations, the resultant system of circuit equations is

a linear system of equations of the form

$$A\,x = b \tag{3.4}$$

where A is the coefficient matrix and b is the excitation vector.

Since the coefficient matrix tends to contain few nonzero coefficients for most formulation techniques, the method of solving (3.4) will exploit the sparseness of the coefficient matrix. The formulation algorithm therefore should produce a coefficient matrix that contains a minimal number of nonzero terms. In theory, at least, the dimension of A is not as important as the total number of nonzero terms that are contained in A.

3.1. Branch Relations

To establish the mathematical model of the circuit, each circuit element is defined as an interconnection of ideal branches. The modeling of a given circuit element is, of course, determined by the physical properties of the element. The models for the SPICE elements are presented in Appendix 2 of this thesis.

The polarity of the branch voltage and the branch current arbitrarily is assigned according to the "associated reference direction" convention [7]. This convention is illustrated in Fig. 3.1. In words, the branch voltage is positive if the positive node of the branch is at a higher potential than the negative node; the branch current is positive if it flows from the positive node, through the branch, to the negative node.

The four general types of branch relations are given in Table 3.1. The vector x in this table contains all of the branch

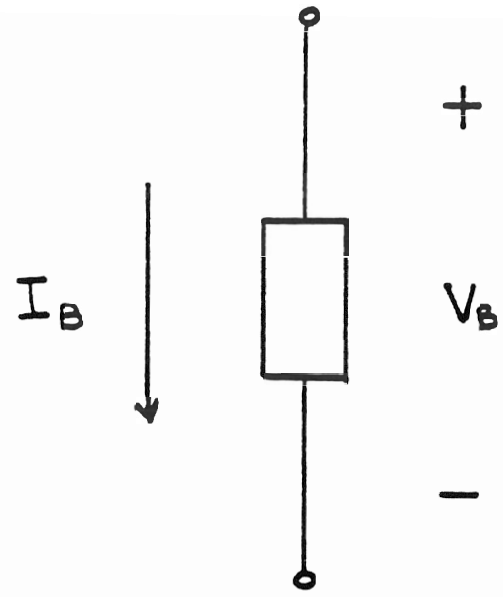


Fig. 3.1. Branch Polarity Definition.

I. Current-Defined Branch With No Energy Storage

$I = f(x)$	$I = I_0$	Ideal Current Source
	$I = V/R$	Ideal Resistor

II. Capacitive Branch

$Q = f(x)$	$Q = CV$	Ideal Capacitor
$I = \frac{d}{dt} Q$	$I = \frac{d}{dt} Q$	

III. Voltage-Defined Branch With No Energy Storage

$V = f(x)$	$V = V_0$	Ideal Voltage Source
	$V = IR$	Ideal Resistor

IV. Inductive Branch

$\phi = f(x)$	$\phi = LI$	Ideal Inductor
$V = \frac{d}{dt} \phi$	$V = \frac{d}{dt} \phi$	

Table 3.1. The Four General Classes of Branch Relations.

voltages and branch currents. A branch is current-defined if the branch current is defined in terms of circuit parameters and circuit variables. If the branch voltage is defined in terms of circuit parameters and circuit variables, then the branch is voltage-defined. For example, voltage sources and inductors are voltage-defined branches, whereas current sources and capacitors are current-defined branches. An ideal resistor with a finite, nonzero value of resistance, can be modeled as either a voltage-defined branch or a current-defined branch.

Usually, a branch relation defines the branch current in terms of the branch voltage or vice-versa. Controlled branches, on the other hand, are defined by branch relations that are dependent upon other branch variables. If a controlled branch depends on other branch voltages, then the branch is voltage-controlled. The branch is current-controlled if the branch related depends upon other branch currents. It is possible, of course, that a branch can be both voltage-controlled and current-controlled.

3.2. Kirchoff's Topological Laws

In addition to the constraints imposed by the branch relations, the circuit equations must satisfy Kirchoff's topological laws [7]:

a) (KCL) - The algebraic sum of currents traversing each cutset of the network must equal zero at every instant of time.

b) (KVL) - The algebraic sum of voltages around each loop of the network must equal zero at every instant of time.

To cast KCL and KVL into the form of (3.2), a tree of the circuit is selected. A tree is a subset of branches that is connected to every node and does not contain any loops. Network

branches that are included in the tree are termed tree branches. Branches that are excluded from the tree are referred to as tree links, tree chords, or cotree branches.

Each tree branch uniquely defines a fundamental cutset, that is, a cutset which contains the given tree branch and one or more tree links. Any tree of a network contains n-1 branches [7]; hence, there are n-1 fundamental cutsets. The n-1 fundamental cutset equations are obtained by applying KCL to each fundamental cutset:

$$[I_{n-1} \mid F] \begin{bmatrix} I_T \\ \hline I_L \end{bmatrix} = 0 \tag{3.5}$$

The matrix I_{n-1} is an $(n-1)\times(n-1)$ identity matrix, F is the fundamental cutset matrix of dimension $(n-1)\times(b-n+1)$, I_T is the vector of tree-branch currents, and I_L is the vector of tree-link currents.

Each tree-link uniquely defines a fundamental loop, that is, a loop which contains only the given tree-link and one or more tree branches. Any tree of the circuit necessarily contains b-n+1 links since it contains n-1 tree branches; hence, there are b-n+1 fundamental loops. The b-n+1 fundamental loop equations are obtained by applying KVL to each fundamental loop to obtain

$$[I_{b-n+1} \mid -F^T] \begin{bmatrix} V_L \\ \hline V_T \end{bmatrix} = 0 \tag{3.6}$$

where I_{b-n+1} is an identity matrix of dimension $(b-n+1)\times(b-n+1)$, $-F^T$ is the fundamental loop matrix of dimension $(b-n+1)\times(n-1)$, V_L is the vector of tree-link voltages, and V_T is the vector of tree-branch voltages.

Equations (3.5) and (3.6) together are equivalent to (3.2). Hence, the matrix T is defined by the equation

$$T = \left[\begin{array}{c|c} I_{n-1} & F \\ \hline -F^T & I_{b-n+1} \end{array}\right] \tag{3.7}$$

The determination of T is not attractive, practically, since a tree must be selected and the matrix F must be derived. Fortunately, there are equation formulation methods that do not require the matrix T.

3.3. Cutset Analysis

The formulation method of Cutset Analysis uses the n-1 tree-branch voltages as the unknown circuit variables. The fundamental cutset matrix, Q is defined by the equation

$$Q = [I_{n-1} \mid F] \tag{3.8}$$

Equations (3.5) and (3.6) are simplifed to yield

$$Qi_b = 0 \tag{3.9}$$

$$v_b = Q^T v_t \tag{3.10}$$

Cutset analysis is derived with the assumption that all branch-relations are current-defined and voltage-controlled. Hence, a voltage source or an inductor are not allowed in cutset analysis, and a current-controlled source also is not allowed. To simplify the arithmetic, consider only the case of linear branch relations. With these assumptions, the general system of branch relations in (3.1) can be simplified to

$$Bv_b + j_b = i_b \tag{3.11}$$

The matrix B is the branch relation coefficient matrix, and the vector j_b is the vector of independent current source values. Now, (3.9), (3.10) and (3.11) are combined to yield the network equations

$$QBQ^T v_t = -Qj_b \tag{3.12}$$

3.4. Loop Analysis

Loop Analysis is the dual of Cutset Analysis. The b-n+1 tree-link currents are used as the unknown vector, and the branch relations are constrained to be voltage-defined and current-controlled. The fundamental loop matrix, L, is defined as[1]

$$L = [F_t \mid I_{b+n-1}] \tag{3.13}$$

Equations (3.5) and (3.6) then are restated as

$$i_b = L^T i_L \tag{3.14}$$

$$Lv_b = 0 \tag{3.15}$$

The branch relations are constrained to be the dual of (3.11) namely

$$Bi_b + e_b = v_b \tag{3.16}$$

where e_b is the vector of independent voltage source values. Hence, combining (3.14), (3.15) and (3.16) yields the system of loop equations

[1] In many texts [7], the fundamental loop matrix is denoted as B. However, the B matrix in this chapter denotes the branch relation matrix, so L is used for the fundamental loop matrix.

$$LBL^T i_b = -Le_b \tag{3.17}$$

3.5. Nodal Analysis

The Nodal method of equation formulation is a special case of Cutset Analysis that does not require the selection of a network tree. Nodal Analysis is derived by using the n-1 nondatum node voltages as the unknowns. The b branch voltages, v_b, and the n-1 nondatum node voltages, v_n, are related by the Nodal incidence matrix as follows:

$$v_b = A^T v_n \tag{3.18}$$

The Nodal incidence matrix is determined by the coefficients

$$a_{ij} = \begin{cases} 0 \text{ if node } i \text{ is not connected to branch } j \\ +1 \text{ if node } i \text{ is the positive node of branch } j \\ -1 \text{ if node } i \text{ is the negative node of branch } j \end{cases}$$

The Nodal incidence matrix is a compact method of describing the network topology that does not depend on the selection of a network tree. In fact, A can be determined directly from the input data. Most methods of determining F begin with the Nodal incidence matrix and obtain F from A by a method such as Gauss-Jordan elimination [29].

It can be shown [7] that the Nodal incidence matrix also directly expresses KCL as follows:

$$Ai_b = 0 \tag{3.19}$$

Clearly, (3.17) is similar to (3.9) and (3.18) is similar to (3.10).

Combining (3.11), (3.18), and (3.19) yields the system of Nodal equations:

$$ABA^{T}v_{n} = -Aj_{b} \tag{3.20}$$

Equation (3.20) is then expressed in simpler notation to yield

$$Yv_{n} = j \tag{3.21}$$

where Y is the Nodal admittance matrix, v_{n} is the node-voltage vector, and j is the node-current excitation vector.

As with Cutset Analysis, the simplest form of Nodal Analysis is applicable only to branch relations that are current-defined and voltage-controlled. Other types of branch relations must be transformed to current-defined voltage-controlled branch relations by inserting series resistances. Voltage sources, for example, are replaced by the Norton equivalent branch relation that is shown in Fig. 3.2.

The Nodal admittance matrix can be constructed in a very efficient manner; namely, every branch relation is added to the row of Y corresponding to the branch positive node and subtracted from the row of Y corresponding to the branch negative node. For example, consider a resistor connected between node i and node j which is defined by the branch relation

$$I_{b} = \frac{1}{R}(v_{i}-v_{j}) \tag{3.22}$$

This branch alters the Nodal admittance matrix by adding 1/R to the y_{ii} and y_{jj} coefficients and subtracting 1/R from the y_{ij} and y_{ji} coefficients.

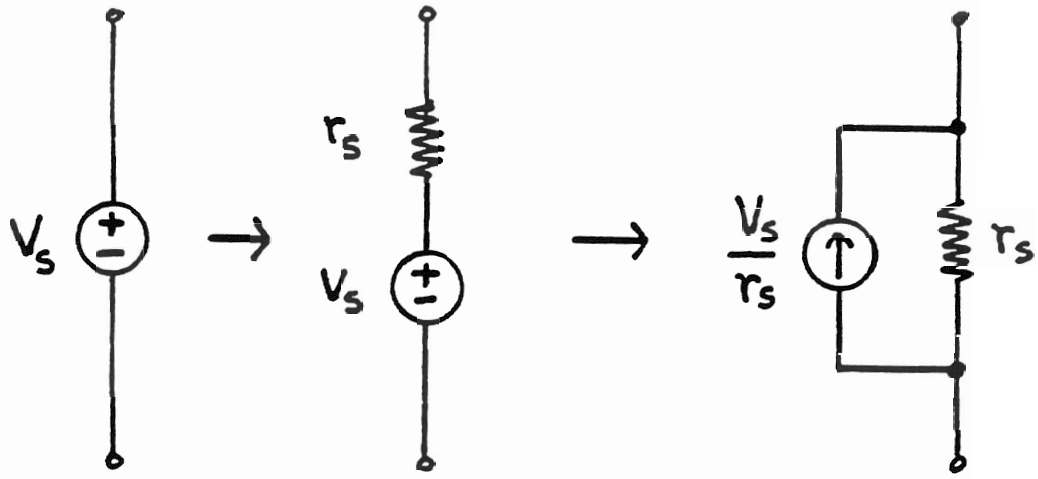


Fig. 3.2. Independent Voltage Source Transformation.

To simplify the implementation of Nodal Analysis, the datum node is included in (3.21) as, say, the first node. The Nodal equations then are assembled without the need to distinguish between grounded branches and non-grounded branches. In the solution of (3.21), the row and column of Y that corresponds to the datum node is ignored. The matrix Y, with the addition of a row and column for the datum node, is referred to as the indefinite admittance matrix [7].

3.6. Nodal Analysis With Voltage Sources

The most serious disadvantage of Nodal Analysis is the inability to handle voltage-defined branches. However, a very large class of integrated circuits can be accommodated with Nodal techniques by adding a provision for grounded voltage sources. The method that is presented here is implemented in SLIC [14,15] and TIME [19]. The addition of grounded voltage sources to the algorithm eliminates the ill-conditioning problems that may occur with series resistances and, as well, results in less computational effort.

The presence of a grounded voltage source in the network model reduces the number of unknown node voltages by one. Grounded voltage sources are included by partitioning the Nodal equations into source equations and reduced equations. If there are n nodes and n_v grounded voltage sources, the nodes are renumbered such that node 1 is the datum node, nodes 2 through n_v+1 are the nodes that are connected to grounded voltage sources, and nodes n_v+2 through n are the remaining circuit nodes. The partitioned set of Nodal equations is assembled in the manner described previously to yield

$$\left[\begin{array}{c} I_v \\ \hline 0 \end{array}\right] + \left[\begin{array}{c|c} y_{ss} & y_{sr} \\ \hline y_{rs} & y_{rr} \end{array}\right] \left[\begin{array}{c} v_s \\ \hline v_r \end{array}\right] = \left[\begin{array}{c} J_s \\ \hline J_r \end{array}\right] \tag{3.23}$$

The vector I_v contains the voltage-source branch currents; this vector must be included if (3.23) is to be a correct statement of KCL. Equation (3.23) is partitioned such that V_s is known, that is,

$$V_s = E \tag{3.24}$$

where $E = (0, E_1, E_2 \ \ldots\ )^T$, E_1 is the first grounded voltage source value, E_2 is the second grounded voltage source value, and so on. The vector V_r is the unknown set of reduced node voltages that is determined by the equation

$$y_{rr} V_r = J_r - y_{rs} E \tag{3.25}$$

The grounded voltage-source currents are then determined by the equation

$$I_V = J_S - y_{ss} V_s - y_{sr} V_r \tag{3.26}$$

Floating voltage sources, and inductors in dc analysis, can be included into Nodal Analysis at the expense of additional matrix operations. The Nodal equations are partitioned in the same manner as before. However, V_s no longer is known, but instead depends upon V_r. This implies that, for each source, the column in y_{ss} and y_{rs} corresponding to the positive node must be added to the column y_{sr} and y_{rr} corresponding to the negative branch node; in addition, the row in y_{ss} and y_{sr} that corresponds to the branch positive node must be added to the row in y_{rs} and y_{rr} that corresponds to the branch

negative node. The vector V_r then is obtained as before by solving the equation

$$\hat{y}_{rr}V_r = J_r - \hat{y}_{rs}E \tag{3.27}$$

where $\hat{y}_{rr}$ and $\hat{y}_{rs}$ denote the submatrices that are obtained by the row and column operations that have just been described. The vector V_s is determined from V_r and E. Finally, if desired, the voltage source currents can be obtained as before from (3.26)

CANCER [2] and the first version of SPICE use Nodal Analysis formulation with the addition of floating voltage sources. In dc analysis, inductors are treated as zero-valued voltage sources. This implementation has the drawback that the matrix structure is different for dc analysis than for ac or transient analysis if the circuit contains inductors. Moreover, the matrix operations that are required are not desirable since they introduce additional bookkeeping into the algorithm. However, if the circuits that are simulated seldom have inductors or floating voltage sources, as is often the case in IC design, then the CANCER equation formulation is simple and efficient.

3.7. Modified Nodal Analysis

The inclusion of voltage-defined branches and current-controlled branches is accomodated in a much easier fashion by the Modified Nodal Analysis (MNA) method that was developed by C. Ho, et al. [47]. If the circuit contains n nodes and n_v voltage-defined branches, then the n-1 nondatum node voltages and the n_v voltage-defined branch currents become the unknown circuit variables. The set of

modified nodal equations are obtained by applying KCL to each nondatum node and including the n_v voltage-defined branch relations. The set of equations is then

$$\left[\begin{array}{c|c} Y & B \\ \hline C & D \end{array}\right]\left[\begin{array}{c} V_n \\ \hline I_b \end{array}\right] = \left[\begin{array}{c} J \\ \hline E \end{array}\right] \tag{3.28}$$

where V_n is the vector of node voltages, I_v is the vector of voltage-defined branch currents, J is the node current excitation vector, and E is the voltage-defined branch voltage vector. Hence, Y and J are constructed in the same manner as in (3.21). If there are no current-controlled, current defined branch relations in the network, then the B matrix has integer coefficients b_{ij} that are determined by

$$b_{ij} = \begin{cases} 0 & \text{if node i is not connected to branch j} \\ 1 & \text{if node i is the positive node of branch j} \\ -1 & \text{if node i is the negative node of branch j} \end{cases}$$

If the network contains current-controlled, current-defined branch relations, then the B matrix also will contain the coupling coefficients that determine these relations.

The C and D matrices and the vector E together comprise the branch relations for the n_v voltage-defined branches. In the case where there are no branches that are voltage-defined or current-controlled, (3.28), reduces to the simple Nodal Eqs. (3.21). For the circuit that is shown in Fig. 3.3 the MNA circuit equations are

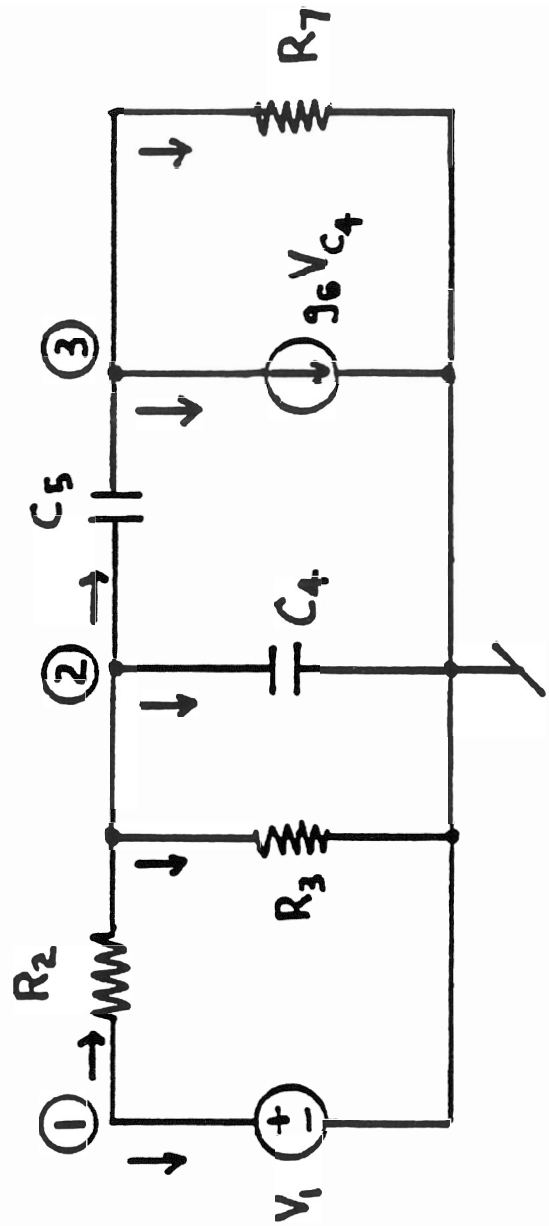


Fig. 3.3. An Example Circuit.

$$\left[\begin{array}{c|c|c|c} 1/R_2 & -1/R_2 & 0 & 1 \\ \hline -1/R_2 & 1/R_2+1/R_3 + C_4\frac{d}{dt} + C_5\frac{d}{dt} & -C_5\frac{d}{dt} & 0 \\ \hline 0 & g_6-C_5\frac{d}{dt} & C_5\frac{d}{dt}+1/R_7 & 0 \\ \hline 1 & 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} V_1 \\ \hline V_2 \\ \hline V_3 \\ \hline I_1 \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline 0 \\ \hline 0 \\ \hline E \end{array}\right] \qquad (3.29)$$

The presence of a zero coefficient on the diagonal of (3.29) is not a computational problem because the matrix structure is such that the zero diagonal element always will become nonzero in the course of the solution. However, the possibility of zero diagonal terms does imply that the order of the decomposition be such that rows containing zero diagonal coefficients are not processed until the fill-in has occurred.

There are computational problems with the MNA formulation, however. The ill-conditioning that can be encountered with the MNA formulation is illustrated by the circuit that is shown in Fig. 3.4. For a dc analysis, the MNA equations for this circuit are

$$\left[\begin{array}{c|c|c|c} 1/R_2 & -1/R_2 & 0 & 1 \\ \hline -1/R_2 & 1/R_2+1/R_4 & -1/R_4 & 0 \\ \hline 0 & -1/R_4 & 1/R_4 & 0 \\ \hline 1 & 0 & 0 & 0 \end{array}\right]\left[\begin{array}{c} V_1 \\ \hline V_2 \\ \hline V_3 \\ \hline I_1 \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline 0 \\ \hline 0 \\ \hline E \end{array}\right] \qquad (3.30)$$

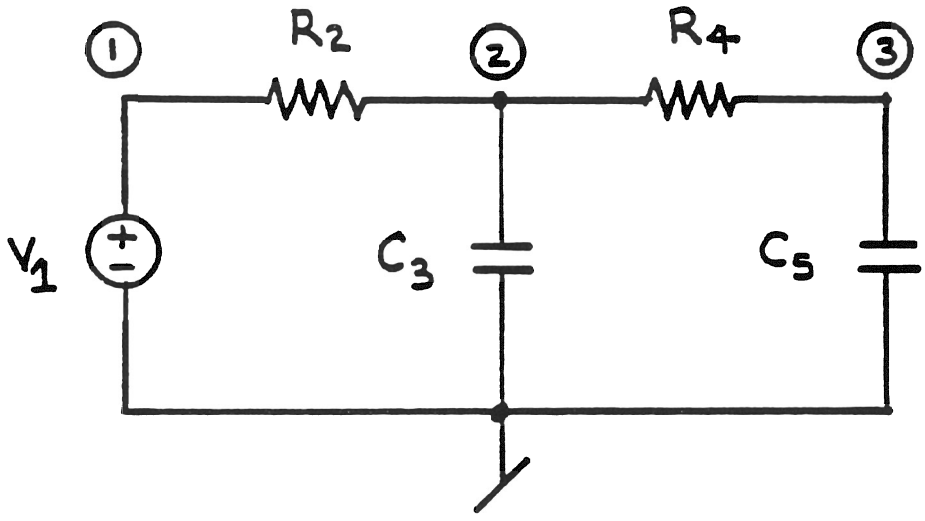


Fig. 3.4. A Circuit to Illustrate Ill-Conditioning With MNA.

Regardless of the order in which the equations are eliminated, one of the node voltage diagonal terms will become zero and the solution will fail.

This ill-conditioning problem, once recognized, can be solved with a straightforward modification. In the example of (3.30), if the first and last rows are interchanged, then the following system of equations is obtained.

$$\left[\begin{array}{c|c|c|c} 1 & 0 & 0 & 0 \\ \hline -1/R_2 & 1/R_2+1/R_4 & -1/R_4 & 0 \\ \hline 0 & -1/R_4 & 1/R_4 & 0 \\ \hline 1/R_2 & -1/R_2 & 0 & 1 \end{array}\right] \left[\begin{array}{c} V_1 \\ \hline V_2 \\ \hline V_3 \\ \hline I_1 \end{array}\right] = \left[\begin{array}{c} E \\ \hline 0 \\ \hline 0 \\ \hline 0 \end{array}\right] \qquad (3.31)$$

This interchange of rows can be generalized as follows. For every voltage-defined branch in the network, there is a branch current variable in the unknown vector. If no voltage-defined branches have coincident nodes, then the interchange is accomplished simply by swapping every voltage-defined current row with the row corresponding to the positive node of the branch. For the case when voltage-defined branches have coincident nodes, it is necessary to establish a mapping between voltage-defined branch currents and network nodes that may contain some voltage-defined branch negative nodes. This mapping can always be constructed unless the network contains a loop of voltage-defined branches; in this case, a possible violation of KVL is implied by the network topology.

The swapping of rows is not performed computationally. Since each matrix coefficient is accessed through an integer pointer system, the row interchange is accomplished by altering the pointer system.

Therefore, after a small amount of additional setup effort is expended, there is no difference in the amount of computational effort due to the row interchange.

Unfortunately, the necessary interchanging of rows destroys the near-symmetric structure of the Nodal equations. In the previous Nodal implementations, the Y matrix is near-symmetric in structure, and few additional zero-valued terms are introduced by assuming symmetry. The assumption of symmetry exactly halves the size of sparse-matrix pointer structure, and hence is highly desirable. It is questionable whether the added generality of the MNA2 algorithm is worth the increased size of the pointer structure.

3.8. Comparison of Nodal Analysis Methods

The three Nodal methods that have been presented were implemented in SPICE and tested on the ten standard benchmark circuits that are described in Appendix 1 of this thesis. The number of equations, the number of nonzero matrix coefficients, and the number of arithmetic operations for these three methods are presented in Table 3.2. In this table, NODAL refers to the Nodal method that is used in CANCER, MNA1 refers to the MNA method as originally proposed by C. Ho, et al. [47], and MNA2 refers to the MNA method with the row interchange. The number of operations that are cited in Table 3.2 are operations of the form A = A-B*C or A = A/B. The only voltage-defined branches that are present in these ten circuits are the grounded voltage sources that model the power supplies and the circuit inputs.

It is clear from this data that the MNA1 method of formulation requires more computational effort, when compared to either the NODAL

	NODAL			MNA1			MNA2		
	EQNS	SIZE	OPS	EQNS	SIZE	OPS	EQNS	SIZE	OPS
DIFFPAIR	13	51	74	16	73	158	16	51	77
RCA3040	29	131	218	32	175	432	32	131	221
UA709	40	258	615	43	302	883	43	258	618
UA741	48	290	601	51	340	945	51	290	604
UA727	57	337	701	61	424	1209	60	337	704
RTLINV	10	30	42	12	45	86	12	30	44
TTLINV	26	106	198	28	135	304	26	106	200
ECLGATE	35	153	292	37	185	412	37	153	294
MECLIII	48	202	374	50	255	580	50	202	376
SBDGATE	53	237	482	55	287	664	53	237	484

Table 3.2. Comparison of the Nodal Analysis Methods

method or the MNA2 method. For the UA741 circuit, as an example, the MNA1 method constructs a coefficient matrix that has 17% more nonzero terms, and 57% more arithmetic operations are required to solve the system of circuit equations.

To further illustrate the differences in these three formulation methods, these ten circuits were retested with the grounded voltage sources replaced by a series combination of a ungrounded voltage source and a small resistor. The results for this test are given in Table 3.3. The matrix size, and the operation count, have increased substantially for the NODAL method and the MNA2 method, yet these figures are approximately the same for the MNA1 method. For the same example of the UA741 circuit, the MNA1 method now produces a coefficient matrix that is 11% smaller than the coefficient matrix that results for the NODAL formulation, and the computational effort for the MNA1 equations is 5% smaller than the effort for the NODAL equations. However, the matrix for the MNA2 method is 4% smaller than for the MNA1 method, and the computational effort for the MNA2 equations is 18% smaller than the effort for the MNA1 method.

3.9. Hybrid Analysis Methods

Hybrid Analysis methods formulate the circuit equations in terms of branch voltages and branch currents. All of these methods require the selection of a network tree; the circuit equations are then derived from the fundamental cutset equations and the fundamental loop equations. Hybrid Analysis methods include the State Variable Analysis formulation that is used in CIRPAC [41,42], SCEPTRE [43], NET [44], and CIRCUS [45]. Tree-link formulation [48] and the formulation

	NODAL			MNA1			MNA2		
	EQNS	SIZE	OPS	EQNS	SIZE	OPS	EQNS	SIZE	OPS
DIFFPAIR	16	74	133	19	74	147	19	62	93
RCA3040	32	194	431	35	178	429	35	152	242
UA709	43	323	888	96	307	886	46	283	691
UA741	51	387	994	54	345	948	54	331	781
UA727	61	457	1185	65	420	1167	65	374	861
RTLINV	12	44	76	14	49	90	14	36	56
TTLINV	28	138	298	30	139	308	30	121	242
ECLGATE	38	186	389	41	186	403	41	142	312
MELLIII	50	278	618	52	261	592	52	240	523
SBDGATE	56	274	595	59	292	667	58	266	685

Table 3.3. Comparison of the Nodal Analysis Methods with Floating Sources.

method that is used in ECAP2 [31,32] also are Hybrid methods, as is the Modified Tableau method that is used in ASTAP [28,29].

The choice of a network tree is a critical part of the formulation algorithm. Kirchoff's laws impose certain constraints upon the tree selection process. Specifically, independent voltage sources cannot be tree links, and independent current sources cannot be tree branches. How well-conditioned the resultant set of circuit equations is depends upon the tree that is selected. Most Hybrid methods choose the tree according to the "ECRLJ" precedence rule. In words, independent voltage sources (E branches) always become tree branches, whereas independent current sources (J branches) always become tree links. Capacitors (C branches) become tree branches whenever possible, and inductors (L branches) become tree links whenever possible. If the circuit contains no capacitive loops or inductive cutsets, then all capacitors will be included in the tree and all inductors will be excluded from the tree. Resistors (R branches) become either tree branches or tree links.

Tree-link formulation [47] is a notable exception to the "ECRLJ" precedence rule. With this proposed formulation, the tree branches are chosen according to their value of equivalent conductances, with large conductances given the highest priority for selection as tree branches. Hence, the tree definition may change from iteration to iteration. This novel approach guarantees that the Hybrid equations will be well-conditioned; however, selecting a tree at each iteration constitutes a considerable amount of additional overhead, especially if sparese matrix methods are employed.

Once the network tree has been selected, all Hybrid Analysis

methods derive the system of fundamental cutset and loop equations in approximately the same fashion.

3.10. Modified Tableau Analysis

Modified Tableau Analysis [28,29] formulates the circuit equations in terms of the branch voltages and the branch currents. The tree is selected by forcing all independent voltage sources to be tree branches and all independent current sources to be tree links; the remaining tree branches are chosen in a manner that is designed to minimize the number of nonzero coefficients in the F matrix [29]. The circuit equations then are obtained from the fundamental cutset equations, the fundamental loop equations, and the branch relations to yield the following system of equations

$$\left[\begin{array}{c|c|c|c} I & 0 & 0 & F \\ \hline 0 & I & -F^T & 0 \\ \hline C_{TIT} & C_{TOL} & C_{TUT} & C_{TIL} \\ \hline C_{LIT} & C_{LVL} & C_{LVT} & C_{LIL} \end{array}\right] \left[\begin{array}{c} I_T \\ \hline V_L \\ \hline V_T \\ \hline I_L \end{array}\right] = \left[\begin{array}{c} 0 \\ \hline 0 \\ \hline E \\ \hline J \end{array}\right] \tag{3.32}$$

The eight C matrices are necessary to express the branch relations. Due to the identity matrices in (3.32), the reduced set of Hybrid equations can be obtained from (3.32) as follows

$$\left[\begin{array}{c|c} C_{TUT}+C_{TUL}F^T & C_{TIL}-C_{TIT}F \\ \hline C_{LUT}+C_{LUL}F^T & C_{LIL}-C_{LIT}F \end{array}\right] \left[\begin{array}{c} V_T \\ \hline I_L \end{array}\right] = \left[\begin{array}{c} E \\ \hline J \end{array}\right] \tag{3.33}$$

The matrix operations that are implied in (3.33) are trivial since F and $-F^t$ are integer matrices with coefficients that are either zero or ± 1.

As an example of the MTA formulation, the complete Modified Tableau for the circuit shown in Fig. 3.3 is given in Fig. 3.5; the tree that was selected includes V_1, C_4, and C_5. The reduced set of circuit equations for this example is

$$\left[\begin{array}{c|c|c|c|c|c|c} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ \hline 0 & C_4 \frac{d}{dt} & 0 & -1 & 1 & 1 & 1 \\ \hline 0 & 0 & C_5 \frac{d}{dt} & 0 & 0 & -1 & -1 \\ \hline -1 & 1 & 0 & R_2 & 0 & 0 & 0 \\ \hline 0 & -1 & 0 & 0 & R_3 & 0 & 0 \\ \hline 0 & -g_6 & 0 & 0 & 0 & 1 & 0 \\ \hline 0 & 1 & -1 & 0 & 0 & 0 & R_7 \end{array}\right] \left[\begin{array}{c} V_1 \\ \hline V_4 \\ \hline V_5 \\ \hline I_2 \\ \hline I_3 \\ \hline I_6 \\ \hline I_7 \end{array}\right] = \left[\begin{array}{c} E_1 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \\ \hline 0 \end{array}\right] \qquad (3.34)$$

Equation (3.34) points out one serious drawback of MTA formulation. In a dc analysis, the time-derivative terms in (3.34) are zero, and therefore the second and third diagonal terms are zero. To solve (3.34) for the dc solution, pivoting must be used to obtain a nonzero diagonal term.

The MTA formulation was applied to the DIFFPAIR, RTLINV, and TTLINV circuits that are part of the ten benchmark circuits that were tested with the Nodal Analysis methods. The results for these three circuits are compared with the MNA results in Table 3.4. For these three simple circuits, the MTA formulation produces a system of equations that contains twice the number of nonzero matrix terms; furthermore, the computational effort required to solve the MTA

Fig. 3.5. The Modified Tableau for an Example Circuit.

			MTA			MNA		
	NODES	BRANCHES	EQNS	SIZE	OPS	EQNS	SIZE	OPS
DIFFPAIR	13	32	32	168	320	16	67	74
RTLINV	10	20	20	77	130	16	36	42
TTLINV	26	52	52	253	491	28	116	198

Table 3.4. Comparison of MTA Formulation and MNA Formulation.

equations is three times the work required to solve the MNA equations. Furthermore, the computational overhead required to reduce the original 2b × 2b Modified Tableau is not included in the comparison shown in Table 3.4. In the paper by Ho, et al. [47], some comparisons of the MNA formulation and the MTA formulation indicated that the number of nonzero coefficients in the MTA equations was seven times the number of nonzero coefficients that were present in the MHA equations. However, it is unfair to count the number of coefficients in the original Modified Tableau since the integer coefficients do not have to be stored [29]; hence, a more meaningful comparison is the comparison of the reduced equations as given in Table 3.4.

The data that is presented here is by no means comprehensive; however, this data does indicate that the MTA formulation in particular, and Hybrid Analysis methods in general, produce a system of equations that are comparable sparsity, in terms of nonzero coefficients per row, to the sparsity of the Nodal Analysis equations. However, the Hybrid Analysis methods produce a much larger set of equations, and therefore the total number of nonzero coefficients for the Hybrid equations will be larger than for the Nodal equations. The additional complexity of tree selection does not result in a significant savings in computational effort; quite the opposite, the Hybrid equations actually require more computational effort to solve.

3.11. Sparse Tableau Formulation

The third formulation technique to be considered in this chapter is the general technique of Sparse Tableau Formulation [48]. With this method, all of the circuit variables are included in the

unknown vector, and a very large, very sparse system of circuit equations is assembled. No prior reduction or simplification of this set of equations is performed; instead, the entire system of Tableau equations is solved.

The Sparse Tableau unknown vector contains the node voltages, the branch voltages, the branch currents, the capacitor charges and the inductor fluxes. The equations are determined by KCL and KVL

$$\text{(KCL)} \quad AI_b = 0 \tag{3.35}$$

$$\text{(KVL)} \quad A^T V_n = V_b \tag{3.36}$$

and by the branch relations. As before, Kirchoff's laws constitute b constraints; the remaining equations are supplied by the branch relations. The resultant set of equations is shown in Fig. 3.6. The C matrices are again the coefficient matrices that are determined by the branch relations.

Either the Nodal Analysis formulation or the Hybrid Analysis can be derived from the Tableau by appropriate simplification [46]. However, the fundamental concept of the Sparse Tableau formulation is that there is no reduction or elimination to obtain a more compact set of equations. Instead, the complete system of equations is solved. Obviously, obtaining the solution of the Tableau equations by convential means will require an enormous amount of computational effort as compared to determining the solution of either the Nodal equations or the Hybrid equations. Therefore, additional sophistication in the equation solution algorithms is necessary.

All of the coefficients of the Tableau matrix belong to one of

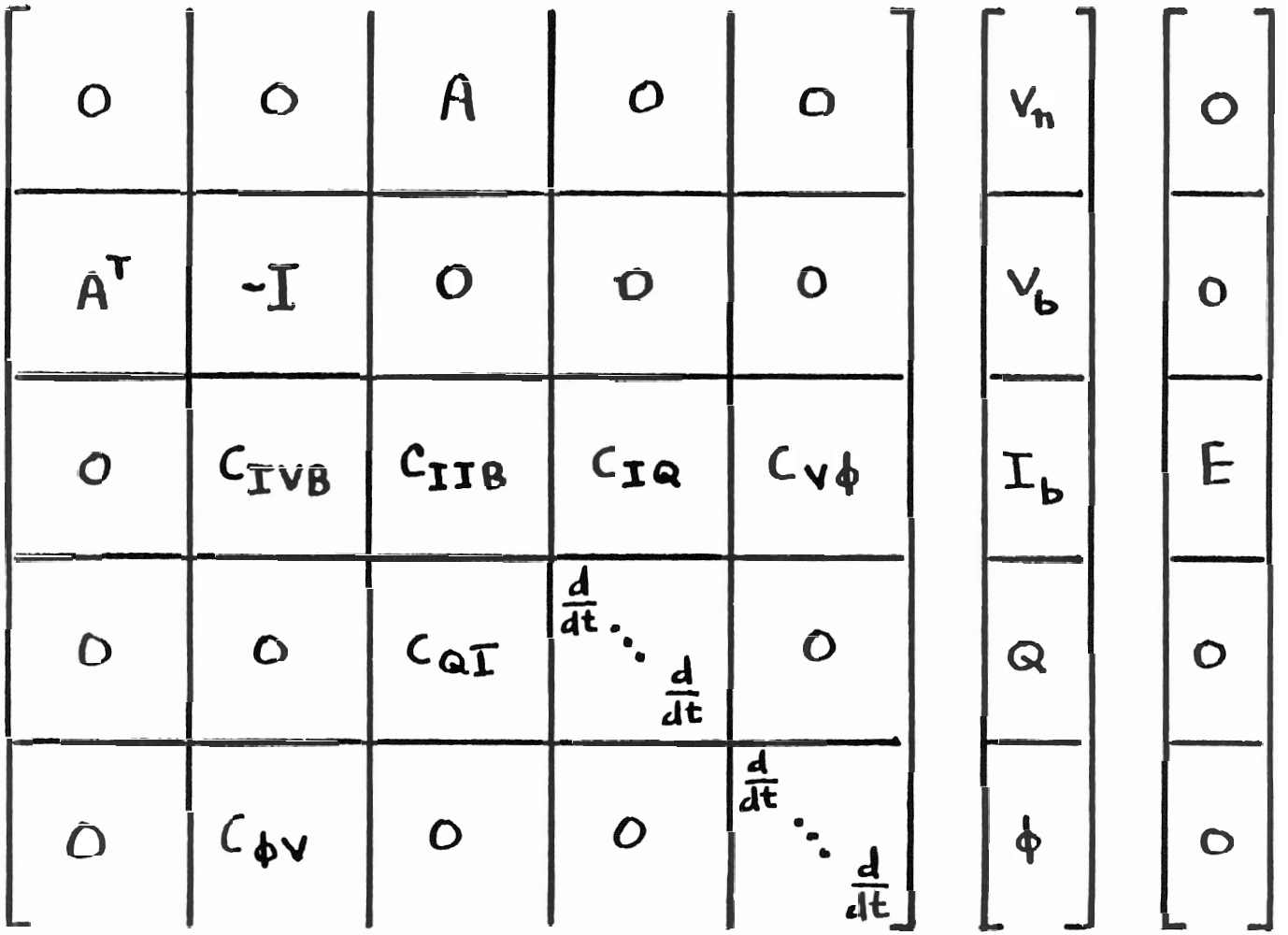


Fig. 3.6. The Sparse Tableau.

four variability types:

a) Topological — either +1 or -1

b) Constant — independent of time or the unknown vector

c) Time-dependent — dependent upon time

d) Nonlinear — dependent upon the unknown vector

For example, a linear resistor value is constant; the equivalent conductance of a linear capacitor is time-dependent, and the equivalent conductance of either a diode or a nonlinear capacitor is nonlinear. To determine the solution at each Newton iteration, only computations that involve nonlinear coefficients need be performed, since only nonlinear coefficients change from on Newton iteration to another. Operations that involve time-dependent coefficients need to be performed once at each timepoint. Finally, operations involving topological coefficients and constant coefficients need only be performed once.

The order in which the rows and columns of the Tableau are eliminated is determined in a manner that is designed to minimize the total computational effort. Operations involving nonlinear coefficients are given a higher weight than time-dependent operations, which, in turn, are given a highr weight than constant or topological operations. Such a reordering algorithm clearly is more complex than the methods that are required either for Nodal Analysis or for Hybrid Analysis.

The fundamental disadvantage of the Sparse Tableau formulation is the inherent problem of ill-conditioning. During the reordering step, there is no clue as to which coefficients are numerically suitable as pivot elements. Even if the reordering strategy is

modified to attempt to condition the Tableau equations by a parital pivoting technique, this pivoting strategy must rely on "average" values. Since many nonlinearities in a circuit have equivalent conductances that range from 10^{3} to 10^{-12} mhos, the concept of an average conductance clearly will lead to problems. In the event that ill-conditioning problems are encountered, the entire reordering algorithm must be invoked again to establish a better conditioned elimination sequence.

The Sparse Tableau technique cannot be expected to always yield a set of numerically well conditioned equations. To offset the size of the Tableau, complicated reordering and solution algorithms are necessary. Finally, the number of operations required by the Sparse Tableau solution was reported to be only 10% - 20% fewer than for Nodal Analysis formulation [46]. This minor improvement in computational effort hardly seems worth the enormous complexity of the technique and the prospects of ill-conditioning that may be encountered.

3.12. Conclusions

Circuit simulation requires, as a first step, the representation of the physical circuit by a system of circuit equations. These equations must satisfy the constraints of the branch relations and Kirchoff's topological laws. An equation formulation algorithm determines an independent system of circuit equations that satisfies these two constraints.

The equation formulation algorithm is by no means unique. The difference in formulation algorithms lies basically in the vector of

unknowns that is used to formulate the system of equations. Since any unknown vector that results in an independent system of equations can be used, there are a myriad of different equation formulation techniques. The two fundamental methods are Cutset Analysis, which formulates the equations in terms of an unknown vector of branch voltages, and Loop Analysis, which formulates the equations in terms of an unknown vector of branch currents. Since both of these fundamental methods of equation formulation impose restrictions upon the types of branch relations that can be present in the network, neither method is applicable to practical simulation problems. Instead, practical formulation methods are combinations of Cutset Analysis and Loop Analysis.

The three currently popular formulation methods are Nodal Analysis, Hybrid Analysis, and Sparse Tableau Analysis. All three algorithms possess sufficient generality to accomodate virtually all circuit configurations. The choice of a specific algorithm is determined by the computational effort that is required to solved the resultant system of circuit equations as well as the numerical conditioning of the system of circuit equations. The Nodal Analysis methods that were presented produce a well-conditioned system of equations that is solved with a minimal amount of computational effort. The Hybrid Analysis methods, by contrast, suffer significant numerical conditioning problems for the case of dc analysis. Moreover, the method of Hybrid Analysis that was presented produced a system of equations that required a larger amount of computational effort to solve as compared to the MNA method. Finally, the Sparse Tableau method that was presented, although totally general in scope, requires

much more complicated programming and, moreover, can produce a system of ill-conditioned equations. Yet, the Sparse Tableau method results in only a 10% to 20% decrease in computational effort.

For these reasons, Nodal Analysis is the formulation algorithm that is used in both versions of SPICE. The first version of SPICE contains the CANCER modification of Nodal Analysis which works well when the circuit contains a minimal number of floating voltage sources and/or inductors. The MNA1 formulation method presently is implemented in SPICE2. Although the MNA2 method is more efficient, the additional complication of row interchange and an unsymmetric coefficient matrix render the MNA2 method less attractive. Moreover, the pointer system for the MNA1 algorithm is half the size of the pointer system that is required by the MNA2 method.

IV. THE SOLUTION OF A SYSTEM OF LINEAR EQUATIONS

All circuit analysis procedures require the solution of a system of linear equations of the form

$$Ax = b \tag{4.1}$$

where A is the coefficient matrix, x is the unknown vector of circuit variables, and b is the excitation vector. The coefficient matrix that is generated in circuit simulation problems typically is of the order of 50 or 100 equations; however, it is not difficult to foresee simulation problems that require the solution of a system of several thousand equations.

The large size of the coefficient matrix is offset by the fact that, for circuit simulation applications, most of the coefficients in A are zero, that is, the system of circuit equations is very sparse. The inherent sparseness of the circuit equations must be recognized to achieve an efficient solution for even a moderate-sized circuit.

This chapter begins with the introduction of direct elimination methods that obtain the solution of (4.1). The implementation of LU factorization for a sparse-matrix solution is described next. The reordering algorithms that are necessary to preserve the sparseness of the coefficient matrix are introduced and compared. The methods of implementing these computational methods then are presented. Finally, the methods of determining the solution of the system of linear equations by iterative means are introduced.

4.1. Direct Elimination

Equation (4.1) is solved readily by multiplying both sides of (4.1) by A^{-1} to yield

$$x = A^{-1}b \tag{4.2}$$

If A^{-1} does not exist, there is no solution and any method of solving (4.1) will fail. Since determining the inverse requires three times the computational effort of other direct elimination methods [49], the method of direct inversion seldom is used in simulation programs.

There are two equivalent methods of direct elimination that do not require the inverse of A: Gaussian elimination and LU factorization. Both require an equal amount of computational effort for the solution with one excitation vector; however, if the solutions for several excitation vectors are required, LU factorization requires less computational effort [35].

With LU factorization, the original coefficient matrix is factored, or decomposed, into the product of a lower-triangular matrix L and an upper triangular matrix U. The original equations then can be restated as

$$Ax = LUx = b \tag{4.3}$$

To attain a unique decomposition, the diagonal terms of L or U are set to unity. After the decomposition is performed, the solution is determined by a forward substitution step and a backward substitution. The forward substitution step requires the L matrix:

$$y = Ux = L^{-1}b \tag{4.4}$$

The backward substitution step requires the U matrix:

$$x = U^{-1}y = U^{-1}L^{-1}b \tag{4.5}$$

Due to the triangular nature of L and U, neither L^{-1} or U^{-1} need be determined to perform (4.4) or (4.5).

4.2. LU Factorization

There are several equivalent methods of LU factorization. The method presented here is due to Doolittle [49]. The diagonal elements of L are set to unity. If there are N equations, the method requires N-1 steps. The process for the first step is

$$a_{1j} = a_{1j} \qquad j = 1,2,\ldots,N \tag{4.6}$$

$$\ell_{j1} = a_{j1}/a_{11} \qquad j = 2,3,\ldots,N \tag{4.7}$$

For the ith step (i=2,3,...,N-1), the calculation is more complicated:

$$u_{ij} = a_{ij} - \sum_{k=1}^{i-1} \ell_{ik}u_{ki} \qquad j = i,i+1,\ldots,N \tag{4.8}$$

$$\ell_{ji} = (a_{ji} - \sum_{k=1}^{i-1} \ell_{jk}u_{ki})/u_{ii} \qquad j = i+1,i+1,\ldots,N \tag{4.9}$$

The structure of L and U is such that both arrays can reside in the same storage array that is used for A. The LU factorization algorithm is modified so that, at the ith step, the ith upper-triangular row of A is replaced by the ith upper triangular row

of U and the ith lower-triangular column of A is replaced by the ith lower-triangular column of L. This is accomplished by replacing Eqs. (4.6-4.9) with the following algorithm for the ith step (i=1,2,...,N-1):

$$a_{ji} = a_{ji}/a_{ii} \qquad j = i+1,i+2,\ldots,N \tag{4.10}$$

$$a_{jk} = a_{jk} - a_{ji}a_{ik} \qquad \begin{matrix} j = i+1,i+2,\ldots,N \\ k = i+1,i+2,\ldots,N \end{matrix} \tag{4.11}$$

4.3. Forward and Backward Substitution

Once the matrix has been factored into L and U, the solution is obtained by a forward and backward substitution. The algorithm is arranged such that all intermediate calculations are stored in the excitation vector. Forward substitution requires N-1 steps; at the ith step (i=1,2,...,N-1), the process is

$$b_j = b_j - \ell_{ji}b_i \qquad j = i+1,i+2,\ldots,N \tag{4.12}$$

The Backward substitution also requires N-1 steps; at the ith step (i=N,N-1,...,2), the process is

$$b_i = b_i/a_{ii} \tag{4.13}$$

$$b_j = b_j - a_{ji}b_i \qquad j = 1,2,\ldots,i-1 \tag{4.14}$$

Finally, the Backward Substitution step is concluded by

$$b_1 = b_1/u_{11} \tag{4.15}$$

Sometimes, it is more efficient to process the upper triangular matrix U by rows instead of by columns. The modified algorithm,

for the first step, is

$$b_N = b_N/u_{NN} \tag{4.16}$$

At the ith step (i=N-1,N-2,...,1)

$$b_i = (b_i - \sum_{j=i+1}^{N} u_{ij})/u_{ii} \tag{4.17}$$

4.4. Operation Counts

Given the algorithm for the direct solution of the system of equations, an estimate of the computational effort required to obtain the solution can be made. All of the matrix operations are of two types:

$$A = A - B*C \tag{4.18}$$

$$A = A/B \tag{4.19}$$

To simplify the accounting, both of these operations are assumed to require about the same amount of central processor execution time. Although (4.19) requires considerably more cpu time than (4.13) on some computers, (4.18) and (4.19) require about the same amount of cpu execution time on the CDC 6400.

If r_i is the number of upper-triangular row coefficients in the ith row and c_i is the number of lower-triangular column coefficients in the ith column, then the number of operations for the ith row and column is during the decomposition is $c_i + c_i r_i$; the operations for the Forward Substitution at the ith row is c_i, and for the Backward Substitution, $r_i + 1$ operations are required

for the ith row. Hence, the total number of operations to obtain the solution is:

$$O = \sum_{i=1}^{N} [(c_i+c_i r_i) + c_i + (r_i+1)] \tag{4.20}$$

For nonsparse techniques,

$$c_i = r_i = i-1 \tag{4.21}$$

so (4.20) reduces to

$$O = [\frac{n^3-n}{3}] + [\frac{n^2-n}{2}] + [\frac{n^2+n}{2}] \tag{4.22}$$

Hence, the decomposition requires on the order of n^3 operations, and both the Forward and Backward Substitution steps require on the order of n^2 operations.

4.5. Roundoff Error

Most simulation programs are implemented on computers that either have a relatively large word size ($\geq$48 bits) or have extended precision. For this reason, the roundoff error that is incurred in the linear equation solution usually is a small fraction of the total solution error. However, for instances where notably ill-conditioned equations are encountered, special algorithms that minimize roundoff error may be necessary [49,50].

Minimal roundoff error is incurred in the LU factorization if, at each step, the diagonal element is larger in absolute value than any of the off-diagonal row or column elements. This condition can be partially satisfied by a partial pivoting which interchanges

either rows or columns at each step of the decomposition, or can be totally satisfied by interchanging both rows and columns at each step of the decomposition.

The search for an optimal pivot and the additional bookkeeping to record which rows and columns have been interchanged complicates a basically simple procedure. Furthermore, the concept of pivoting is not compatible with a sparse-matrix solution, since whenever the pivoting order is altered the entire set of sparse-matrix pointers must be reconstructed. Instead, a sparse-matrix implementation of the direct elimination method relies upon the fact that the equations are well-conditioned.

Even with a well-conditioned set of equations, roundoff error pessimistically can be expected to reduce the number of significant digits by a factor of $1+2\log_{10}N$, where N is the number of equations [49]. Hence, the solution of 100 equations will be accurate to at least the machine word size accuracy less five digits, whereas the solution of 1000 equations will be accurate to at least the machine word size accuracy less seven digits. Since an accuracy of three to six significant digits normally is required for circuit simulation, the machine word size must be compatible with 10 to 13 significant digits. For most computers, double-precision arithmetic will be necessary to preserve accuracy. Since the CDC 6400 has a 60 bit word length that is equivalent to 14 significant digits, single precision arithmetic is sufficient.

4.6. Sparse-Matrix Techniques

To effect a sparse-matrix solution, a set of integer pointers

is necessary to assign a unique location for each nonzero element in the coefficient matrix. If the diagonal terms are nonzero, they need no pointer reference. Moreover, if the excitation vector elements also are nonzero, the excitation vector also can be stored in a nonsparse fashion. The remaining off-diagonal coefficients are stored in a single-subscript array. In the SPICE program, the order of storage is upper triangle row terms followed by lower triangle column terms. The matrix pointer scheme that is used in SPICE is illustrated in Fig. 4.1. If there are N equations and N_o total off-diagonal terms, then this scheme requires two integer arrays of length N+1 and one integer array of length N_o. The meaning of the arrays is as follows. The IUR array is the pointer used to access rows. The $\underline{i}$th row elements are stored in addresses AO(IUR(I)) through AO(IUR(I+1)-1). If IUR(I) equals IUR(I+1) then there are no upper triangle row terms in the $\underline{i}$th row. The IO array is used to establish the row or column positions. For example, in the third row there are two upper triangle row terms (IUR(4)-IUR(3)=2). They are the a_{34} term (IUR(2)=2, IO(2)=4) which has the value -1.0 (AO(2)=-1.0) and the a_{35} term (IO(3)=5) which has the value -3.0 (AO(3)=-3.0). The ILC array is used to access columns in an identical fashion. This pointer scheme is not unique; the choice of an appropriate pointer scheme is arbitrary as long as every matrix coefficient is referenced in an efficient manner.

In some cases, the coefficient matrix has a symmetric or near-symmetric structure. Nodal Analysis, for example, produces a system of circuit equations that are very near-symmetric in structure.

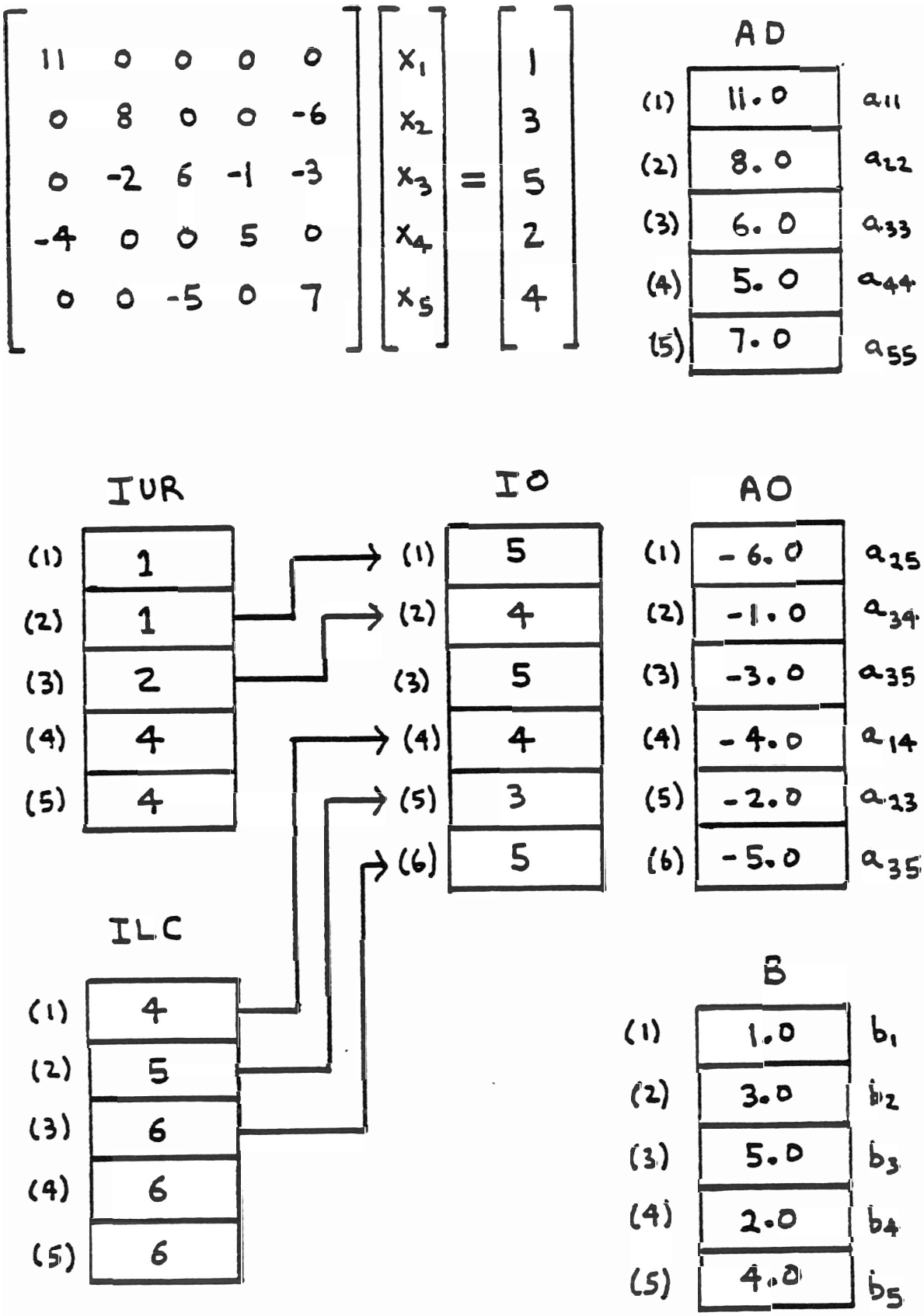


Fig. 4.1. The Sparse-Matrix Pointer System.

By including a few zero coefficients, the structure of the matrix can be made symmetric. This symmetry results in a pointer structure that is half as large as for the nonsymmetric case [36]. Specifically, the ILC array in Fig. 4.1 is not necessary, and the IO array need be only half as large. The reduced size of the pointers more than compensates for the additional zero coefficients that are retained in the sparse matrix array.

Directly after the description of the circuit has been read and checked for errors, all circuit elements are searched to establish which matrix terms are nonzero. Next, the rows and columns of the matrix are reordered in a manner that minimizes the number of additional fill-in terms that are created by the decomposition step, and the fill-in terms are added to the pointers.

4.7. Control of Fill-In

During the decomposition, matrix terms may become nonzero even though they were originally zero-valued. This is evident from the inspection of the decomposition step (4.11). Even if a_{jk} in (4.11) is zero, if both a_{ji} and a_{ik} are nonzero, then a_{jk} will become nonzero.

The occurance of this fill-in decreases the sparseness of the original coefficient matrix and decreases the advantages gained by using sparse matrix technqiues. Moreover, one fill-in term may generate additional fill-in terms; that is, fill in may propagate.

The single degree of freedom that is available in the decomposition is the order in which the rows and columns of the coefficient matrix are factored. If roundoff errors are neglected then different

orders of decomposition produce the same results but may require drastically different operation counts and fill-in. A reordering algorithm attempts to select a decomposition order that minimizes operation count.

The simplest reordering algorithm is due to Markowitz [51]. The maximum number of fill-in terms that can be generated is the product of the number of upper triangle row terms and the number of lower triangle column terms. Row terms that are in columns that already have been factored, and column terms that are in rows that already have been factored, are excluded from the product since they cannot generate fill-in. The Markowitz algorithm proceeds by choosing as the next pivot the row-column pair that has the minimum row-column product, and therefore, the row-column pair that can produce the minimum number of fill-in terms in the worst case. In case of a tie, the row-column pair with the fewest number of column elements is chosen, and in case of further tie the first encountered row-column pair is chosen. Before selecting the next pivot, all fill-in terms that are generated must be added to the pointers.

A more comprehensive reordering algorithm was proposed by Berry [36] and implemented in the CANCER program [2]. This algorithm computes the amount of fill-in that actually would be generated for each possible row-column pair, instead of assuming that the fill-in will be maximal. The row-column pair that generates the minimal amount of fill-in is then chosen as the next pivot. In case of ties, Berry proposed that the row-column pair with the maximum number of row and column elements be chosen.

In case of further ties, the first encountered row-column pair is then chosen. A later implementation of the Berry algorithm by Nahkla, et al. [52] proposed that the tie-breaking algorithm should be the row-column pair with the minimum number of row and column elements.

4.8. A Comparison of the Reordering Algorithms

The reordering algorithms of Markowitz, Berry, and Nahkla were implemented in SPICE and tested on the ten benchmark circuits that are described in Appendix 1 of this thesis. The results of this test are given in Table 4.1. The Berry algorithm and the Nahkla algorithm yield almost identical results for these ten circuits. The largest difference between either of the Berry algorithms and the Markowitz algorithm was a 5% difference in operations count for the MECLIII circuit. On the average, the Berry algorithms require twice the central processor time to determine the reordering as did the Markowitz algorithm.

Because the circuits that were tested yielded inconclusive results, a statistical test of the various reordering algorithms was conducted. In the first test, the matrix was assumed to be symmetric and the number of nonzero off-diagonal row and column terms was assigned the distribution shown in Fig. 4.2. Twenty matrices of dimension 50 and 100 were randomly generated and reordered with all three of the algorithms. The results for this test are shown in Table 4.2. These results indicate that the tie-breaking procedure of Nahkla is marginally better; however, there still is virtually no difference between any of the algorithms.

	MARKOWITZ			BERRY			NAHKLA		
	TERMS	OPS	CPU	TERMS	OPS	CPU	TERMS	OPS	CPU
DIFFPAIR	51	58	0.016	51	58	0.016	51	58	0.017
RCA3040	131	186	0.049	131	186	0.057	131	186	0.057
UA709	258	583	0.120	253	583	0.333	258	583	0.344
UA741	290	547	0.134	290	597	0.305	290	547	0.314
UA727	337	643	0.156	335	635	0.423	335	635	0.423
RTLINV	30	36	0.009	30	36	0.010	30	36	0.010
TTLINC	106	188	0.040	104	182	0.052	104	182	0.051
ECLGATE	153	276	0.059	151	270	0.092	151	270	0.092
MELLIII	202	352	0.080	196	334	0.105	196	334	0.103
SBDGATE	237	470	0.103	239	478	0.249	235	464	0.242

Table 4.1. Comparison of the Reordering Algorithms for the Ten Benchmark Circuits.

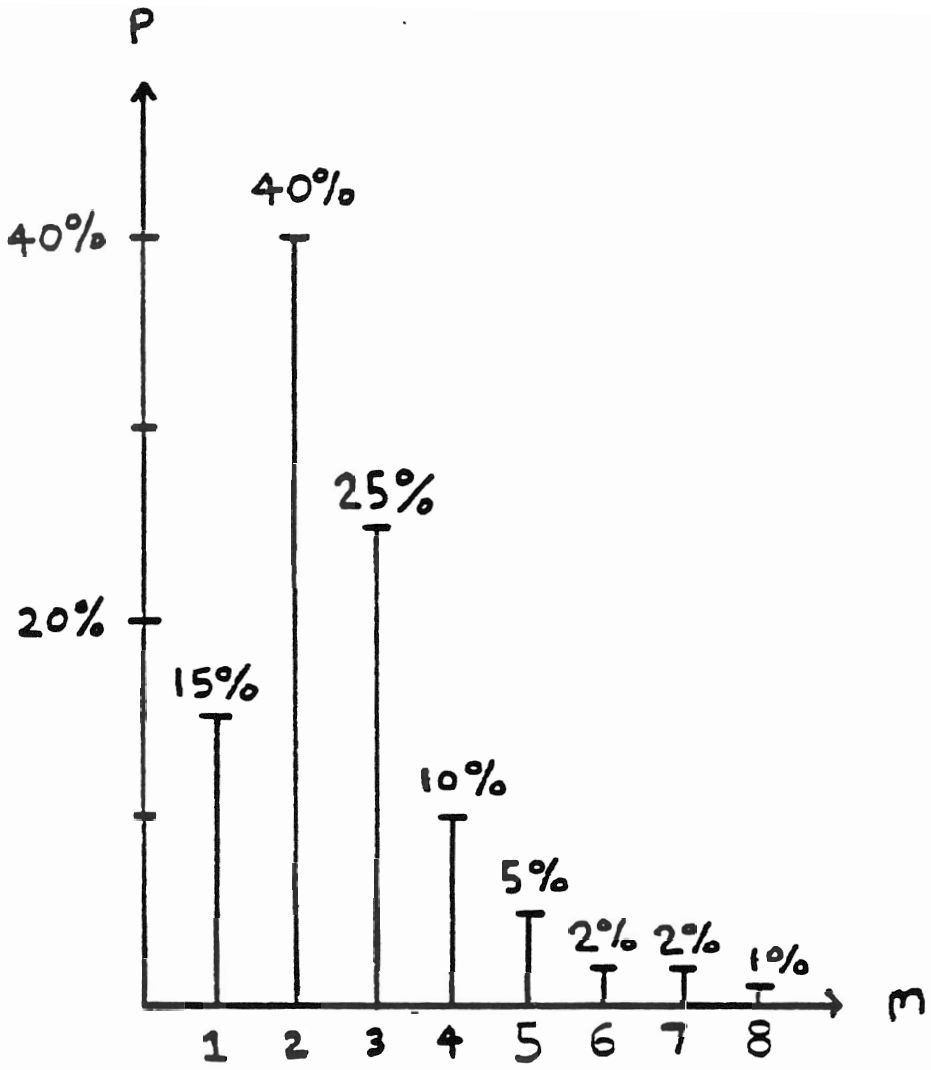


Fig. 4.2. Distribution of Row and Column Off-Diagonal Elements for the First Statistical Test.

	MARKOWITZ			BERRY			NAHKLA		
	TERMS	OPS	CPU	TERMS	OPS	CPU	TERMS	OPS	CPU
50 EQNS	318.3 ±35.2	943.6 ±192.5	0.109 ±0.023	317.1 ±34.1	933.9 ±187.0	0.528 ±0.139	315.8 ±34.6	925.7 ±189.2	0.548 ±0.150
100 EQNS	783.5 ±72.6	3015.9 ±582.7	0.446 ±0.095	779.1 ±70.4	2965.0 ±598.9	3.189 ±0.867	778.5 ±70.9	2962.9 ±556.1	3.235 ±0.877

± are standard deviation

20 samples for all cases

Table 4.2. Comparison of the Reordering Algorithms for the First Statistical Test.

In the second statistical test, the matrix again was assumed to be symmetric, and the number of nonzero off-diagonal row and column terms was chosen from the distribution shown in Fig. 4.3. This distribution yielded matrices that were more representative of actual network matrices. Only the Markowitz algorithm and the Nahkla algorithm were included in this sequence of test. Twenty matrices of dimension 50, 100, 200, 400, 600, 800, and 1000 were generated and reordered with the two algorithms. The results for this set of tests are given in Table 4.3. Again, for all of the cases that were tested, the major difference between the algorithms is the computational effort that is required to determine the reordering and not the reordering itself. Since both methods yield practically the same reordering, the Markowitz method is preferable because it requires considerably less computational effort to determine the reordering.

Another reordering algorithm that was proposed by Hsieh and Ghausi [53] proceeds in a similar fashion to the Berry algorithm, except that the row-column pair which is least likely to produce fill-in, in a probabilistic sense, is chosen as the next pivot. This algorithm produces a reordering that is less efficient than the Berry or Nahkla algorithms but more efficient than the Markowitz algorithm. The computational effort is greater than the Markowitz algorithm, but less than the Berry or Nahkla algorithms. Since the difference between the Berry algorithms and the Markowitz algorithms virtually is indistinguishable, the difference between the Berry algorithms and the Hsieh-Ghausi algorithm also is negligible, and it can be concluded that the

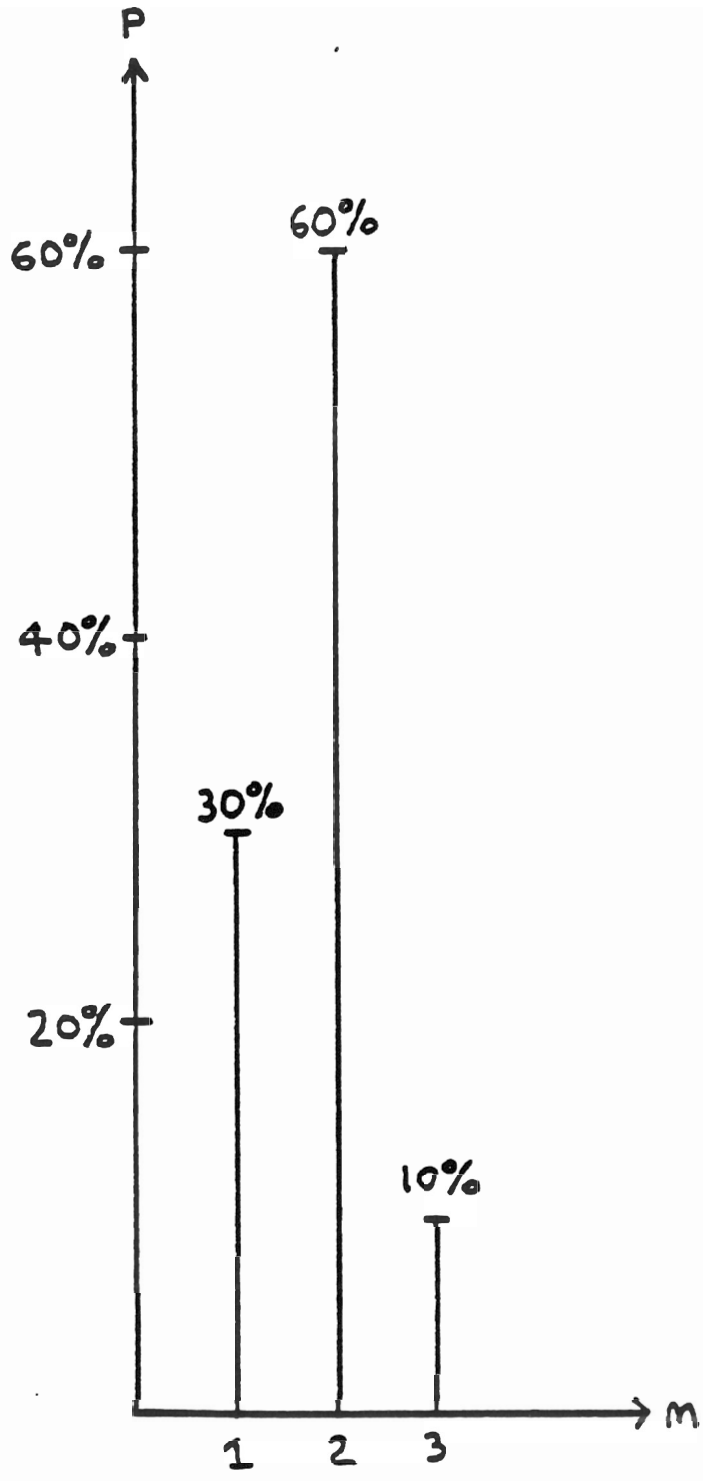


Fig. 4.3. Distribution of Row and Column Off-Diagonal Elements for the Second Statistical Test.

	MARKOWITZ			NAHKLA			
	TERMS	OPS	CPU SEC	TERMS	OPS	CPU	
50 EQNS	188.1 ±20.4	374.4 ±64.1	0.043 ±0.009	188.0 ± 20.4	379.1 ± 60.1	0.114 ±0.056	20 samples
100 EQNS	383.0 ±32.5	766.8 ±100.0	0.115 ±0.019	382.1 ± 32.6	763.6 ± 99.9	0.386 ±0.148	20 samples
200 EQNS	788.0 ±61.1	1619.9 ±212.5	0.346 ±0.057	786.3 ± 61.1	1611.6 ±209.9	1.423 ±0.556	20 samples
400 EQNS	1726.0 ±142.0	4014.3 ±730.8	1.281 ±0.194	1718.8 ±136.6	3954.6 ±679.9	6.947 ±2.158	20 samples
600 EQNS	2627.5 ±165.7	6377.1 ±879.0	2.665 ±0.314	2618.5 ±164.9	6304.0 ±865.1	15.216 ±3.934	16 samples
800 EQNS	3552.2 ±291.4	9370.9 ±1896.2	4.565 0.717	3535.6 ±285.1	9176.4 ±1813.9	26.089 ±8.498	9 samples
1000 EQNS	4732.8 ±222.5	14412.4 ±1980.5	7.666 ±0.573	4698.8 ±201.9	13831.6 ±1608.2	49.933 ±6.940	5 samples

± denotes standard derivation

Table 4.3. Comparison of the Reordering Algorithms for the Second Statistical Test

additional computational effort that is expended by the Hsieh-Ghausi algorithm is not worth the gain in reordering.

4.9. Program Implementation

The simplest implementation of the linear equation solution algorithm is a FORTRAN subroutine. During the decomposition, the coefficient a_{jk} in (4.11) cannot be accessed directly. Instead, the jth row must be searched for the pointer to the a_{jk} term. This search can be eliminated, at the expense of additional memory, by storing the locations of the a_{jk} coefficients in an auxiliary MEMO array.

Since the solution routines are small and straightforward, they can be coded easily in assembly language instead of FORTRAN. Using assembly language affords a better utilization of registors and eliminates many redundant load/stores. Again, the routines either can search for the location of the a_{jk} terms or can store their locations in a MEMO array.

Finally, at some expense in memory, a set of machine code instructions can be generated and stored in memory. Instead of executing a general solution routine, the set of machine code instructions is executed. Since all of the mathematical operations required to obtain the solution are of the two types (4.18) and (4.19), the generation of machine code is not a monumental task. The net result is a custom machine program that contains no loops or indexing and hence is as efficient as possible.

The five possible methods were implemented in SPICE[1] and tested

[1]The assembly language programs were coded by E. Cohen at the Electronics Research Laboratory, University of California, Berkeley.

on the ten benchmark circuits that are described in Appendix 1 of this thesis. The test results for dc and transient analysis are given in Table 4.4 and the results for ac analysis are cited in Table 4.5. The total memory requirements, excluding program length, are given in Table 4.6. The columns entitled "W/O MEMO" denote no auxiliary storage array, while the columns entitled "W/MEMO" denote the usage of an auxiliary storage array. The final column in Tables 4.4. - 4.5, entitled LOAD TIME, lists the average time that is required to construct the circuit equations. All times in Tables 4.4. - 4.5 are in ms.

The significant conclusion from this data is that coding the matrix routines in assembly code is well worth the effort. The assembly routines are 56% faster in dc and transient analysis, and 39% faster in ac analysis. Moreover, the assembly routines require no additional memory.

The use of the MEMO array yields little improvement. This array results in a 11% - 12% average improvement in dc and transient solution at a cost of a 9% increase in central memory. For ac analysis, the MEMO array yields a 6% - 9% average improvement at a cost of 8% additional memory.

Machine code generation can provide a substantial improvement in execution time. These tests show an improvement of 72% over FORTRAN coding and 34% over assembly coding for dc and transient analysis. However, the memory required to store the machine code increased the total memory requirement by 60%. Whether this increase in memory is acceptable depends, of course, on the computer system and billing algorithms that are used. In ac analysis, code

	FORTRAN w/o MEMO	FORTRAN w/MEMO	ASSY w/o MEMO	ASSY w/MEMO	CODE GENERATION		LOAD TIME
DIFFPAIR	4.07	3.88	1.88	1.84	1.37		9.13
RCA3040	12.08	11.21	5.36	5.07	3.70		23.71
UA709	35.97	28.66	15.18	12.20	9.35		33.19
UA741	33.48	28.47	14.24	12.34	9.34		46.09
UA727	38.37	33.11	16.61	14.49	10.94		48.04
RTLINV	2.65	2.61	1.23	1.24	0.88		4.68
TTLINV	11.40	10.59	5.08	4.75	3.43		13.15
ECLGATE	16.70	15.25	7.26	6.74	4.94		19.32
MECLIII	21.00	19.59	9.42	8.88	6.42		26.97
SBDGATE	28.80	25.85	12.01	11.00	8.02		25.80

ALL CPU TIMES IN MSEC

Table 4.4. Execution Times for the Solution Methods in DC or Transient Analysis.

	FORTRAN w/o MEMO	FORTRAN w/MEMO	ASSY w/o MEMO	ASSY w/MEMO	CODE GENERATION		LOAD TIME
DIFFPAIR	6.46	6.31	4.02	3.98	3.46		4.22
RCA3040	19.43	18.77	11.85	11.57	10.20		9.60
UA709	57.19	50.13	33.52	30.74	28.07		19.38
UA741	53.77	49.08	31.96	30.19	27.23		17.88
UA727	62.08	57.11	37.42	35.37	31.98		19.81

Table 4.5. Execution Times for the Solution Methods in AC Analysis.

	NO MEMO	MEMO	CODE	NO MEMO	MEMO	CODE
DIFFPAIR	746	761	910	996	1011	1393
RCA3040	1257	1312	1757	1588	1643	2821
UA709	1765	2032	3227	2222	2489	5863
UA741	2045	2265	3436	2534	2754	5999
UA727	2331	2588	3965	2868	3125	6940
RTLINV	589	596	697			
TTLINV	981	1040	1431			
ECLGATE	1268	1359	1992			
MELLIII	1572	1680	2506			
SBDGATE	1663	1825	2882			

Table 4.6. Memory Requirements for the Solution Methods for Both Transient Analysis and AC Analysis.

generation decreased the execution time by 50% over FORTRAN code and only 15% over assembly code. The increase in memory is 125% for ac analysis. Clearly, the modest decrease in execution time, when compared to assembly code, is not worth the increase in memory.[2]

The data in Tables 4.4. - 4.5 also establishes the relative importance of the equation solution routines. Each Newton iteration in a dc or transient analysis is comprised of a matrix load followed by an equation solution. For dc or transient analysis, then, the equation solution routines consume 45% of the execution time of a Newton iteration if FORTRAN routines are used. If assembly routines are used, this fraction drops to 26%. Finally, if machine code generation is adopted, the solution routines account for only 19% of a Newton iteration time.

For ac analysis, the importance of the load and solve portions of the program is appreciably different. Each frequency point again requires a load and an equation solution. If FORTRAN solution routines are used, the equation solution accounts for 74% of each frequency point solution. If assembly routines are implemented, the solution routines consume 63% of each frequency point solution. If machine code generation is used, this fraction can be reduced to 59%.

---

[2]After these tests were conducted a method of reducing the memory requirement with complex function calls was proposed by E. Cohen. However, the 15% decrease in execution time still does not justify the proposed 40% increase in memory.

4.10. Iterative Techniques

The system of linear equations can be solved iteratively, much the same way that nonlinear equations are solved. The Gauss-Seidel method [49] is the most common iterative method. Each complete iteration of the Gauss-Seidel method requires N^2 operations. If the number of iterations is less than N/3, the Gauss-Seidel method requires less computational effort than direct elimination. Furthermore, at the end of an iteration, the Gauss-Seidel method essentially "starts over" with a new initial guess; hence, any roundoff error that is incurred in the solution will come only from the last iteration.

The Gauss-Seidel method will converge if the set of equations is diagonally dominant, although the rate of convergence depends upon the particular coefficient matrix. Even if Nodal formulation is used, the coefficient matrix is not always diagonally dominant; hence, convergence of the Gauss-Seidel algorithms cannot be assured. Furthermore, it has been shown that although roundoff errors tend to be less severe for iterative methods than for direct methods, there are cases when both methods incur the same roundoff error [49]. However, for implementation on computers with relatively small word sizes, some modification of the Gauss-Seidel method may be worthwhile.

Iterative linear solution methods can be more generally stated as a minimization problem. The scalar performance index is defined as

$$r = (Ax-b)^T(Ax-b) \tag{4.23}$$

A minimization algorithm is employed to determine a vector x such that r is minimal. The use of the Fletcher-Powell algorithm [54] to determine x for linear circuits was reported to yield satisfactory results [48]; but in subsequent application to nonlinear circuits, the minimization method was discarded because it was less efficient than direct elimination. Another minimization algorithm was reported by Agnew [55], but subsequent application to linear network simulation [56] demonstrated that this method requires twice the computational effort that direct elimination does; furthermore, in some cases, the method did not converge to a correct solution.

4.11. Conclusions

The solution of a linear system of equations is an important prerequisite to the total task of circuit simulation. This solution is obtained easily by the direct elimination method of LU factorization. Because the system of circuit equations inherently are sparse, sparse-matrix methods are used to obtain reasonable central memory requirements and execution times.

The implementation of sparse matrix methods is a relatively simple task. An integer pointer system to access the sparse matrix is determined during the setup phase of the program execution. Once this pointer system is established, the construction and solution of the system of circuit equations requires little additional effort as compared to non-sparse methods.

The occurance of fill-in during the decomposition tends to reduce the sparsity of the original coefficient matrix and reduce

the advantage of sparse-matrix techniques. Tests on representative circuits and on randomly generated matrices that ranged from 50 equations to 1000 equations showed that there is virtually no difference in the amount of fill-in that is generated by any of the available reordering methods. The Markowitz algorithm for reordering a system of equations is the most attractive choice since this algorithm requires substantially less computational effort to determine the reordering sequence.

Several different implementations of the equation solution algorithms were compared. As compared to FORTRAN routines, assembly language routines require half the central processor execution time to solve the system of equations. Coding these routines in assembly language therefore is well worth the effort. A further reduction in the solution time is obtained by generating a non-looping set of machine code instructions specific to each circuit; however, code generation requires additional memory to store the code. For dc and transient analysis, where the system of equations is real, the increase in memory may be worth the decrease in execution time. However, for ac analysis, the set of equations is complex, and the generation of machine code almost doubles the memory requirements but yields only a 15% decrease in solution time. In SPICE, machine code is generated only for dc and transient analysis; for ac analysis, an efficient assembly language routine is used to solve the system of complex equations.

The tests that were performed on randomly generated matrices provides a good indication of how the linear solution routines will perform for larger circuit sizes. As the size of the system of

equations was increased, the proportional amount of fill-in also increased because fill-in terms can generate additional fill-in terms. Hence, the central memory requirements and the solution execution time are not simple linear functions of the size of the system of equations. Empirically, the number of operations is proportional to $n^{1.24}$ and the number of nonzero coefficients, including fill-in terms, is proportional to $n^{1.11}$. This data indicates that the solution of a system of 1000 equations with the sparse-matrix methods that are described is feasible. Finally, for typical circuits, the linear equation solution consumes only 10% - 20% of the computational effort that is required for a Newton iteration; the remainder of the computational work is consumed in the construction of the equations.

V. NONLINEAR ANALYSIS METHODS

If a circuit contains nonlinear elements, then both a dc analysis and a transient analysis of the circuit require a method of solving of nonlinear algebraic equations of the form

$$G(x) = 0 \tag{5.1}$$

where x is the vector of unknown circuit variables. For a dc analysis, the system of circuit equations is simplified to the form of (5.1) by the assumption that all time-derivatives of X are constant. In the case of transient analysis, a numerical integration algorithm reduces the system of differential circuit equations, at each timepoint, to a system of algebraic equations of the form of (5.1).

Nonlinear solution algorithms determine the solution of the system of nonlinear equations by an iterative sequence of linearized equations solutions. The method by which the equations are linearized at each iteration is determined by the specific nonlinear solution method that is used. Once the system of linearized equations is assembled, each iterate solution is obtained by a linear solution method such as Gaussian elimination or LU factorization.

In a circuit simulation program, the nonlinear solution method is applied to two markedly different problems. First, this method is used to determine the dc operating point of the circuit. For this case, no a priori knowledge of the solution is assumed, and the iterative sequence must begin with an initial "guess" that

sometimes may be a very poor approximation to the correct solution. Moreover, the failure of the algorithm to converge to a solution is catastrophic. Therefore, the algorithm for the dc operating point must, as first priority, have reliable convergence characteristics. The number of iterations that are required to converge is of secondary importance.

The second application of the nonlinear solution method is multiple-point transient and dc analysis. In a multiple-point analysis, the solution at the previous point usually is a good estimate of the solution value. Moreover, failure to converge in a multiple-point analysis usually is not as catastrophic. In a transient analysis, for example, if a timepoint solution does not converge, the timestep usually can be reduced until the solutions converge. This timestep reduction usually is necessary to maintain a reasonable level of truncation error in the analysis.

Each solution in a multiple-point analysis requires few iterations (typically 2 or 3), but oftentimes many points are computed (several hundred, for example). Therefore, the number of iterations that are required to converge at each point in the analysis is more important than for the dc operating point solution. Hence, the rate of convergence is more important than the reliability of convergence for a multiple-point analysis.

This chapter begins with the introduction of two familiar nonlinear solution methods: the Newton-Raphson algorithm and the Secant method. Next, the implementation of the Newton-Raphson algorithm in a circuit simulation program is presented. The criterion for determining when the sequence of Netwon iterations

has converged is considered. The problems of numerical overflow and nonconvergence are detailed, and practical methods of circumventing these problems are compared.

Finally, methods of improving the efficiency of the Newton-Raphson algorithm are introduced. Prediction methods for improving the initial "guess" at each timepoint in a transient analysis, or a multiple-point dc analysis, are presented. A bypass algorithm that avoids recomputing nonlinear functions that have not changed appreciably from the previous iteration solution also is introduced.

5.1. The Newton-Raphson Algorithm

Most circuit simulation programs use some modification of the Newton-Raphson algorithm to determine the solution of the nonlinear system of algebraic equations. If x_k is the solution at the kth iterate, and δx_k is the error between x_k and the exact solution, then the k+1 iterate solution, x_{k+1}, is chosen such that

$$x_{k+1} = x_k + \delta x_k \tag{5.2}$$

If the error δx_k is small, then the solution of (5.1) at the iterate value x_{k+1} can be expanded in a Taylor series to yield

$$G(x_{k+1}) = G(x_k+\delta x_k) \cong G(x_k) + J(x_k)\delta x_k = 0 \tag{5.3}$$

If (5.1) contains n equations, that is,

$$G(x) = [g_1(x), g_2(x), \ldots, g_n(x)]^T \tag{5.4}$$

then the Jacobian matrix, $J(x_k)$, is defined by

$$J(x_k) = \begin{pmatrix} \left.\frac{\partial g_1}{\partial x_1}\right|_{x_k} & \left.\frac{\partial g_1}{\partial x_2}\right|_{x_k} & \cdots & \left.\frac{\partial g_1}{\partial x_n}\right|_{x_k} \\ \vdots & & & \vdots \\ \left.\frac{\partial g_n}{\partial x_1}\right|_{x_k} & \left.\frac{\partial g_n}{\partial x_2}\right|_{x_k} & \cdots & \left.\frac{\partial g_n}{\partial x_n}\right|_{x_k} \end{pmatrix} \tag{5.5}$$

Equations (5.2) and (5.3) are combined to yield

$$J(x_k)\, x_{k+1} = - G(x_k) - J(x_k)\, x_k \tag{5.6}$$

Equation (5.6) is a linearized system of equations that is solved by Gaussian elimination or LU factorization techniques.

The Newton sequence begins with a starting point x_o. The function $G(x_o)$ and the Jacobian $J(x_o)$ are then evaluated, and (5.6) is assembled and solved to obtain x_1. This process continues until the values of x_k and x_{k+1} agree within some error tolerance.

The rate of convergence of the nonlinear solution algorithm determines how many iterations are required to converge to a specific tolerance. Unfortunately, the rate of convergence can be determined only in the vicinity of the correct solution. It can be shown that, if x_k is sufficiently close to the exact solution, then, for the Newton-Raphson method [49]

$$\delta x_{k+1} = C(\delta x_k)^2 \tag{5.7}$$

where C depends upon higher derivatives of the equations. In words, the Newton-Raphson method has a quadratic rate of convergence; that is, the error at each iteration is proportional to the square of the error at the previous iteration.

5.2. Branch Linearization with the Newton-Raphson Algorithm

The linearized system of Eq. (5.6) can be assembled without formulating the nonlinear equations in the form of (5.1). It is much more efficient, computationally, to linearize each branch relation individually [37,38]. The composite system of linearized equations are then formulated in the same manner that the equations for a linear circuit are formulated.

Each branch relation in the circuit is linearized by the same Taylor-series method. For a current-defined branch, the branch relation is of the form

$$I = f(x) \tag{5.8}$$

Expanding (5.8) in a Taylor series yields the branch relation

$$I(x_{k+1}) = \left.\frac{\partial f}{\partial x}\right|_{x_k} x_{k+1} + \left[f(x_k) - \left.\frac{\partial f}{\partial x}\right|_{x_k} x_k\right] \tag{5.9}$$

where

$$\left.\frac{\partial f}{\partial x}\right|_{x_k} = \left[\left.\frac{\partial f}{\partial x_1}\right|_{x_k}, \left.\frac{\partial f}{\partial x_2}\right|_{x_k}, \ \ldots \ \left.\frac{\partial f}{\partial x_n}\right|_{x_k}\right] \tag{5.10}$$

If the branch relation is current-defined and depends only upon its own branch voltage, then the linearization procedure given in (5.9) and (5.10) is equivalent to replacing the nonlinear branch, at each iteration, by a linear conductance in parallel with an

independent current source. This equivalence is illustrated in Fig. 5.1. The nonlinear element is replaced, conceptually, by the conductance g_{eq} and the current source I_{eq}

For the dual case of a voltage-defined branch, the branch relation is of the form

$$V = f(x) \tag{5.11}$$

The linearized branch relation for the voltage-defined branch is

$$V(x_{k+1}) = \left.\frac{\partial f}{\partial x}\right|_{x_k} x_{k+1} + \left[f(x_k) - \left.\frac{\partial f}{\partial x}\right|_{x_k} x_k\right] \tag{5.12}$$

If the voltage-defined branch relation depends only upon its own branch current, then the linearization procedure is equivalent to replacing the nonlinear branch, at each iteration, with a linear resistor in series with an independent voltage source. This dual replacement is illustrated in Fig. 5.2.

As an example, consider the diode circuit that is shown in Fig. 5.3a. The diode is modeled by the ideal idode branch relation[1]

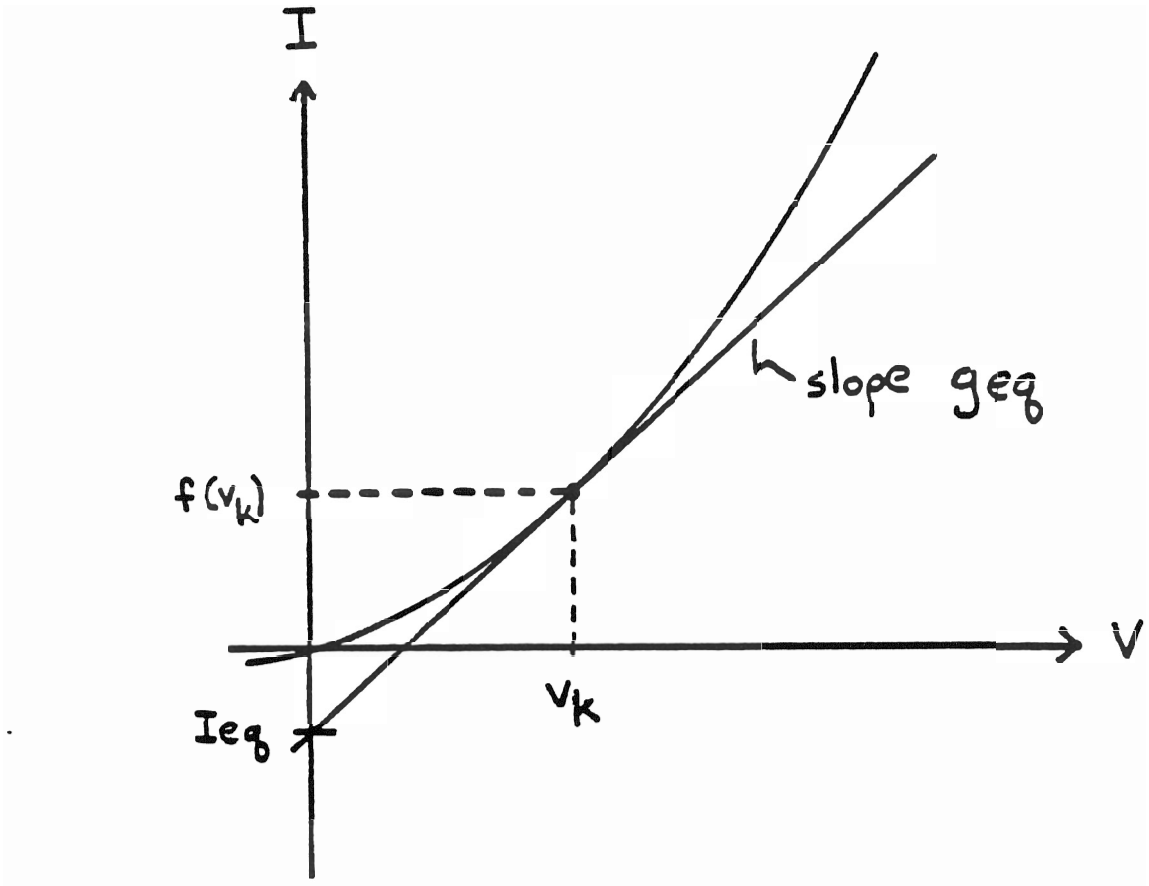
$$I_D = I_S\left[\exp\left(\frac{V}{Vt}\right) - 1\right] \tag{5.13}$$

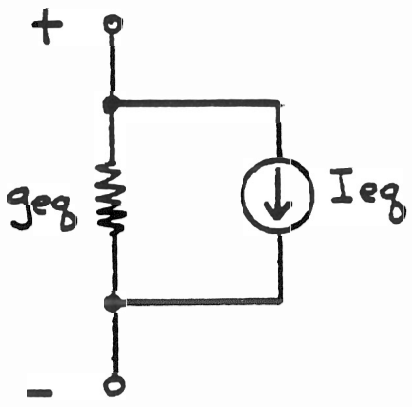
For this simple circuit, it is clear that

$$I_D = \frac{v}{R} - I_o \tag{5.14}$$

Equations (5.13) and (5.14) are presented graphically in Fig. 5.4. The exact solution is, of course the intersection of the curves A, corresponding to (5.13), and B, corresponding to (5.14). At

[1]The parameter V_t is the thermal voltage and is equal to kT/q, where k is Boltzmann's constant, T is the absolute temperature, in degrees Kelvin, and q is the electronic charge. V_t is about 25.85 mv at room temperature.

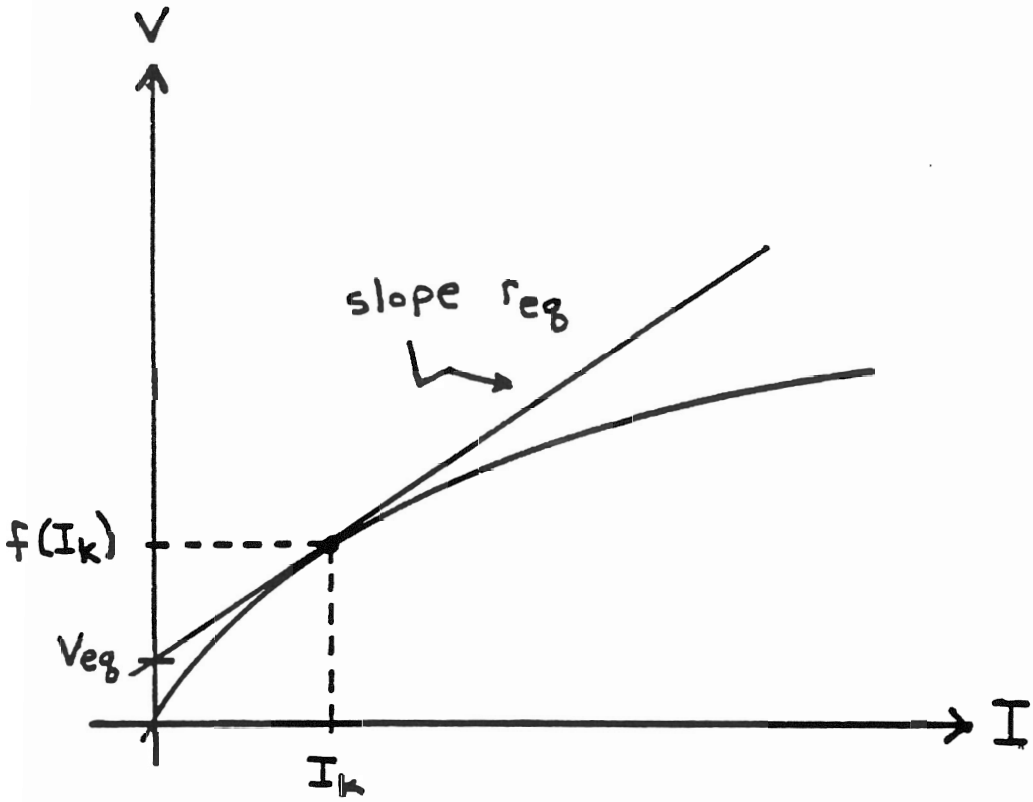


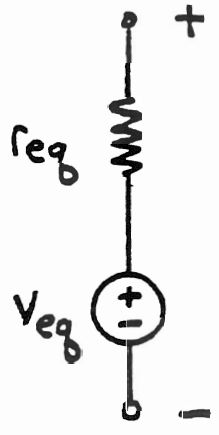


$$g_{eq} = \left.\frac{\partial f}{\partial v}\right|_{v_k}$$

$$I_{eq} = f(v_k) - g_{eq} v_k$$

Fig. 5.1. Branch Linearization for a Current-Defined Branch.





$$r_{eq} = \left.\frac{\partial f}{\partial I}\right|_{I_k}$$

$$V_{eq} = f(I_k) - r_{eq} I_k$$

Fig. 5.2. Branch Linearization for a Voltage-Defined Branch.

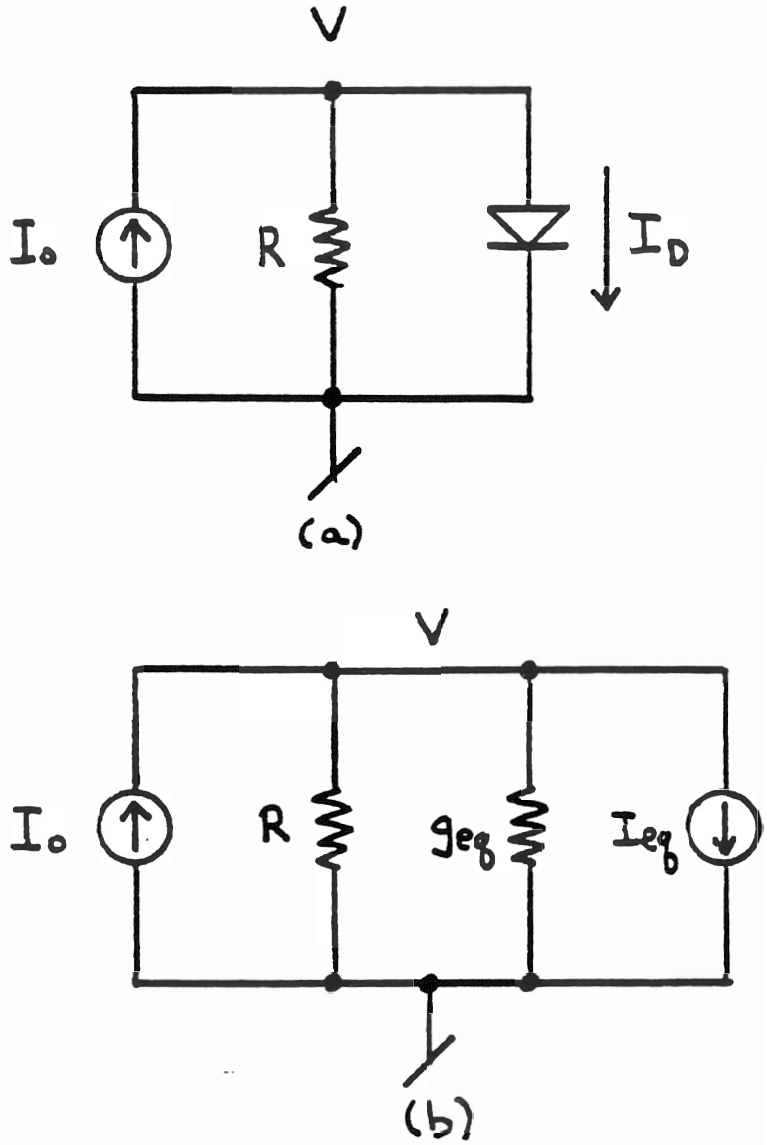


Fig. 5.3. (a) Example Diode Circuit

(b) Linearized Circuit

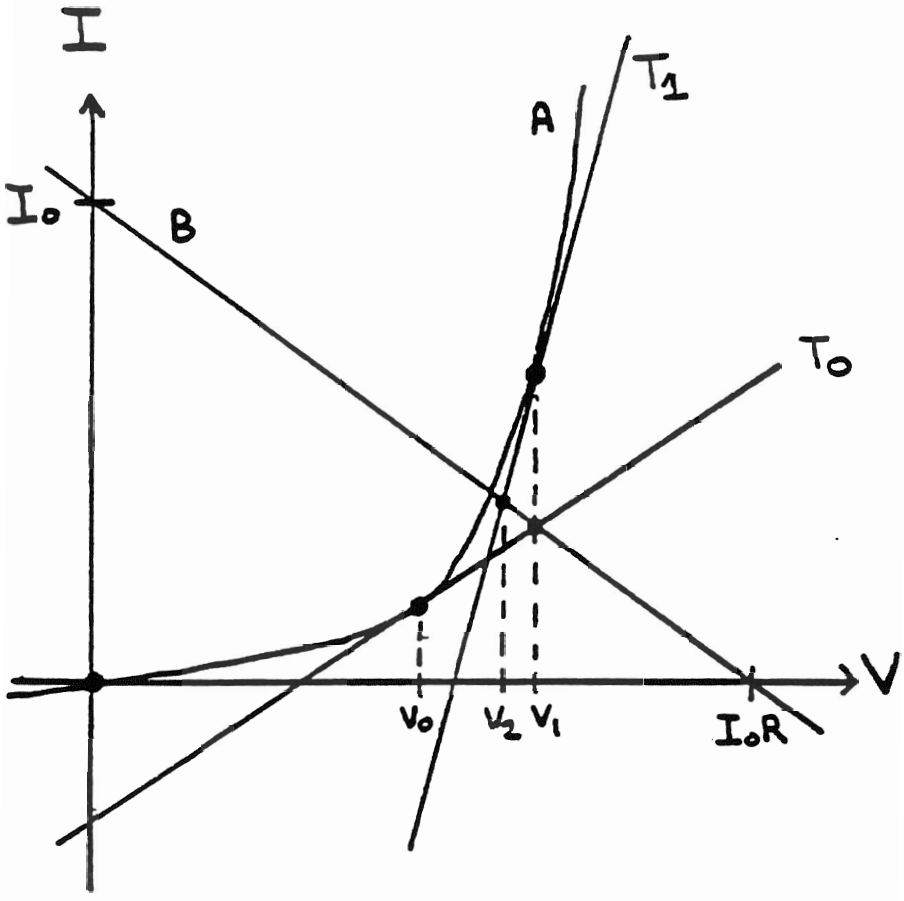


Fig. 5.4. Graphical Representation of the Diode Circuit.

each iteration, the diode element is replaced by the parallel combination of a conductance g_{eq} and an independent current source I_{eq} where

$$g_{eq} = \frac{I_S}{Vt} \exp(\frac{V}{Vt}) \tag{5.15}$$

$$I_{eq} = I_S[\exp(\frac{V}{Vt}) - 1] - g_{eq}V \tag{5.16}$$

The composite I-V equation for this parallel connection of g_{eq} and I_{eq} is the linearized branch relation that is given in (5.9). The equivalent linearized circuit is illustrated in Fig. 5.3b.

At each iteration, the Newton-Raphson algorithm approximates the diode nonlinearity by a linear equivalent relation that is tangent to the nonlinear curve. For example, at the point v_o in Fig. 5.4, the diode is approximated by the tangent T_o. The intersection of T_o and the load-line is the next iterate solution v_1. The diode nonlinearity is then approximated by the tangent T_1, and the intersection of T_1 and the load-line provides the second iterate solution v_2. Each intersection of the tangent and the load-line represents the solution of the linearized system of circuit equations.

5.3. The Secant Method

The Secant Method is similar to the Newton-Raphson method. The difference between these methods is that the Secant Method approximates the derivatives in the Jacobian matrix by divided differences, whereas the Newton-Raphson method evaluates the derivatives directly.

To illustrate the Secant Method, consider again the simple

diode circuit. The graphical representation of the Secant iterations is shown in Fig. 5.5. The Secant Method requires two past iterate solutions. Starting with the two initial guesses v_o and v_1, the diode nonlinearity is approximated by the secant S_1. This secant is equivalent to the parallel connection of a linear conductance, g_{eq}, and an independent current source, I_{eq}, where the values of g_{eq} and I_{eq} are given by

$$g_{eq} = \frac{I_D(v_1)-I_D(v_o)}{v_1-v_o} \tag{5.17}$$

$$I_{eq} = I_D(v_1) - g_{eq}\, v_1 \tag{5.18}$$

The intersection of the secant S_1 and the load-line provides the next iterate solution v_2. The secant S_2 then is constructed using the values of diode current at v_1 and v_2.

For a system of equations, the Secant Method is stated by the equation

$$\hat{J}(x_k)\, x_{k+1} = - G(x_k) - \hat{J}(x_k) x_k \tag{5.19}$$

where $\hat{J}$, the Jacobian estimate, is determined by

$$\hat{J}(x_k) = \begin{pmatrix} \frac{g_1(x_k)-g_1(x_{k-1})}{x_1(k)-x_1(k-1)} & \cdots & \frac{g_1(x_k)-g_1(x_{k-1})}{x_n(k)-x_n(k-1)} \\ \vdots & & \\ \frac{g_n(x_k)-g_n(x_{k-1})}{x_1(k)-x_1(k-1)} & \cdots & \frac{g_n(x_k)-g_n(x_{k-1})}{x_n(k)-x_n(k-1)} \end{pmatrix} \tag{5.20}$$

Equation (5.19) is identical, in form, to (5.6). Furthermore, the approximate Jacobian (5.20) approaches the Jacobian (5.5) in the vicinity of the exact solution. The Secant Method, therefore,

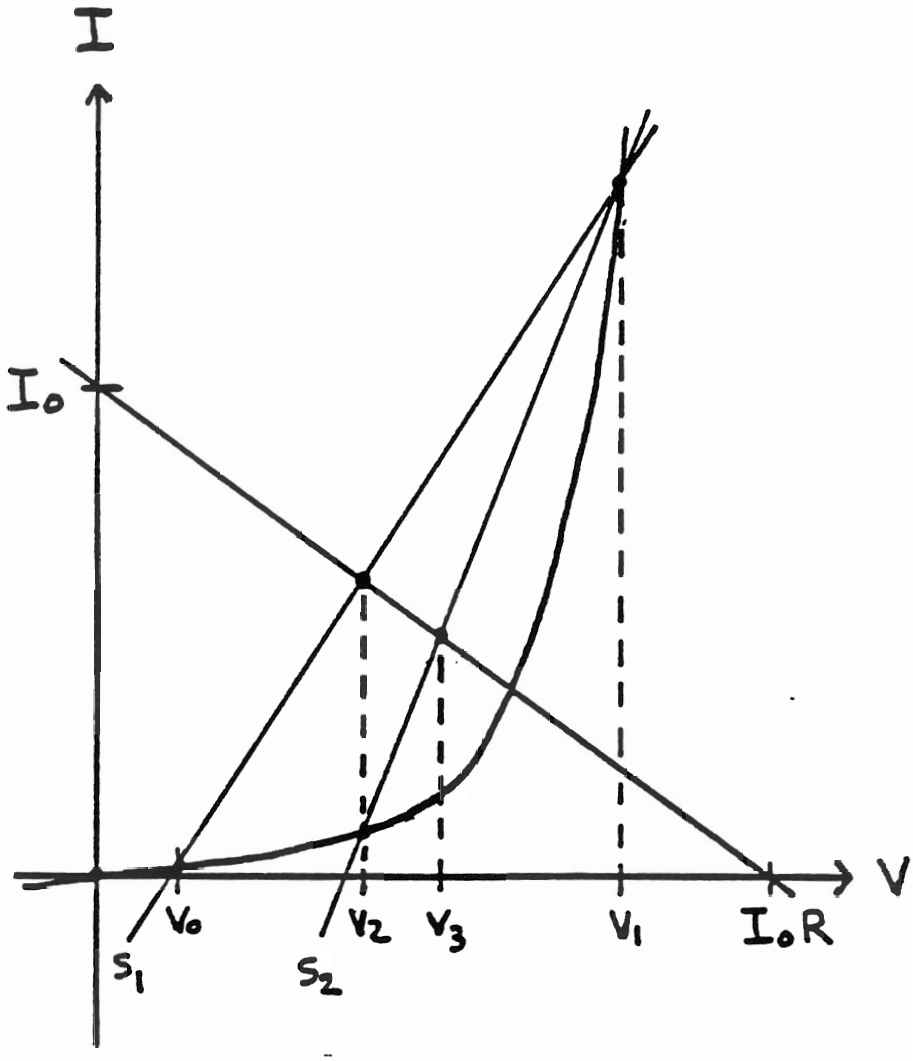


Fig. 5.5. The Secant Solution Method for the Diode Circuit.

approximates the Newton-Raphson method. The rate of convergence of the Secant Method $\frac{1 + \sqrt{5}}{2} = 1.62$ [49], as compared to 2 for the Newton-Raphson algorithm.

The advantage of the Secant Method, as compared to the Newton-Raphson method, is that the derivatives of the nonlinear functions are not evaluated. However, a comparison of (5.17-5.18) with (5.15-5.16) indicates that the computational effort that is involved in evaluating the derivatives is negligible for the case of the diode equation. In fact, the Jacobian for most circuit equations can be formed with little additional effort once the function G(x) has been evaluated. Therefore, the computational advantage of the Secant Method does not outweigh slower rate of convergence of the Secant Method.

Several modifications of the Secant Method have been proposed [57] with the aim of achieving more reliable convergence. However, these methods suffer the same convergence problems that the Newton-Raphson method suffers; hence, little benefit is gained by using the Secant Method or any of its modifications for the analysis of electronic circuits. This explains why, without exception, the general-purpose simulation programs that presently are available use the Newton-Raphson algorithm.

5.4. Convergence

Any nonlinear solution method requires a criterion to determine when the iterative sequence of solutions has converged. A logical choice for this criterion is the requirement that the vector of circuit variables x_{k+1} agree with the prior solution x_k within a

specified tolerance. This criterion requires that each component of the unknown vector, $x_k(n)$, satisfy the equation

$$|x_{k+1}(n)-x_k(n)| < \varepsilon_a + \varepsilon_r \text{ Min}\{|x_k(n)|,|x_{k+1}(n)|\} \qquad (5.21)$$

where ε_a is an absolute tolerance and ε_r is a relative tolerance. If Nodal Analysis formulation is used, this criterion requires that each node voltage must agree in value with the solution at the previous iteration by the specified tolerance. This convergence criterion was used in CANCER and the first version of SPICE. A value for ε_a of 0.001 and a value for ε_r of 50 μv produces acceptable results in most cases.

The convergence criterion of (5.21) sometimes yields false convergence. Consider, for example, the diode circuit that is shown in Fig. 5.6. The diode current is defined by (5.13) with $I_S = 1 \times 10^{-14}$. The results for the first eleven iterations of this problem are listed in Table 5.1. The first column, V2, is the iterate solution for the voltage at node 2. The second column, ID, is the diode current for the given value of V2. Finally, the column IR is the current flowing in the resistor for the given value of V2. Clearly, KCL requires that ID equal IR.

The data in Table 5.1 shows that the node voltage V2 converges to a relative tolerance of 0.1% in one iteration. However, the solution that is obtained after one iteration results in a violation of KCL that is much larger than the 0.1% error tolerance. In fact, eleven iterations are required to obtain a solution that satisfies KCL to a relative tolerance of 0.1%.

This false convergence results from the improper choice of

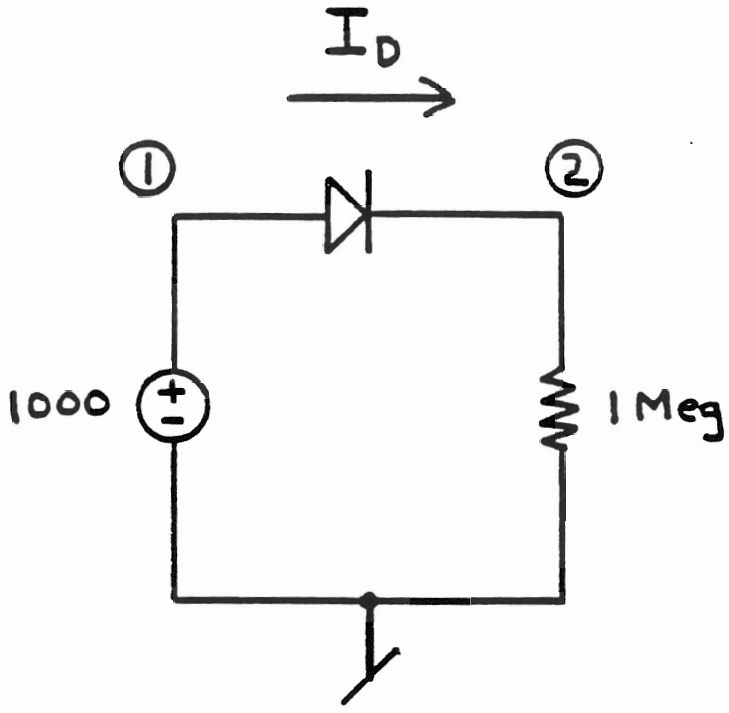


Fig. 5.6. A Numerical Example of False Convergence.

ITER	V_2	ID	IR
1	999.40	1.052×10^{-4}	9.994×10^{-4}
2	999.18	5.143×10^{-1}	9.992×10^{-4}
3	999.21	1.896×10^{-1}	9.992×10^{-4}
4	999.23	7.010×10^{-2}	9.992×10^{-4}
5	999.26	2.616×10^{-2}	9.993×10^{-4}
6	999.28	9.999×10^{-3}	9.993×10^{-4}
7	999.30	4.065×10^{-3}	9.993×10^{-4}
8	999.32	1.912×10^{-3}	9.993×10^{-4}
9	999.34	1.186×10^{-3}	9.993×10^{-4}
10	999.34	1.013×10^{-3}	9.993×10^{-4}
11	999.34	9.994×10^{-4}	9.993×10^{-4}

Table 5.1. The Solution of the Simple Diode Example for Eleven Iterations.

circuit variables that are used in the convergence criterion. The convergence test should not depend upon which circuit variables are chosen as unknowns in the equation formulation algorithm. Instead, the criterion should be based upon how accurately the linearized branch relations approximate the original nonlinear branch relations.

This observation prompts the following convergence criterion. Consider the graphical representation of the simple diode circuit that is given in Fig. 5.7. The starting value for the branch voltage is chosen arbitrarily. The branch relation is then linearized about the point (I_o,V_o) and the set of circuit equations is assembled and solved to determine the next iterate branch voltage V_1. The branch current $\hat{I}_1$ is the linearized branch current that corresponds to V_1, while the branch current I_1 is the actual value of current that corresponds to the branch current V_2.

If I_1 and $\hat{I}_1$ are not equal, then KCL is not satisfied for the branch voltage v_1. A logical convergence criterion then is

$$|I-\hat{I}| < \varepsilon_a + \varepsilon_r \text{ Min}\{|I|,|\hat{I}|\} \qquad (5.22)$$

Equation (5.22) is applied to every nonlinear branch relation. Clearly, if the circuit contains both current-defined branches and voltage-defined branches, then (5.22) is applied to every current-defined branch and the dual of (5.22) is applied to every voltage-defined branch.

The convergence criterion of (5.22) is implemented in the SPICE2 program. To provide a comparison of these two criterion, the transient analysis of the ten standard benchmarks was run both on the SPICE1 program and on SPICE2. The ten circuits are detailed in

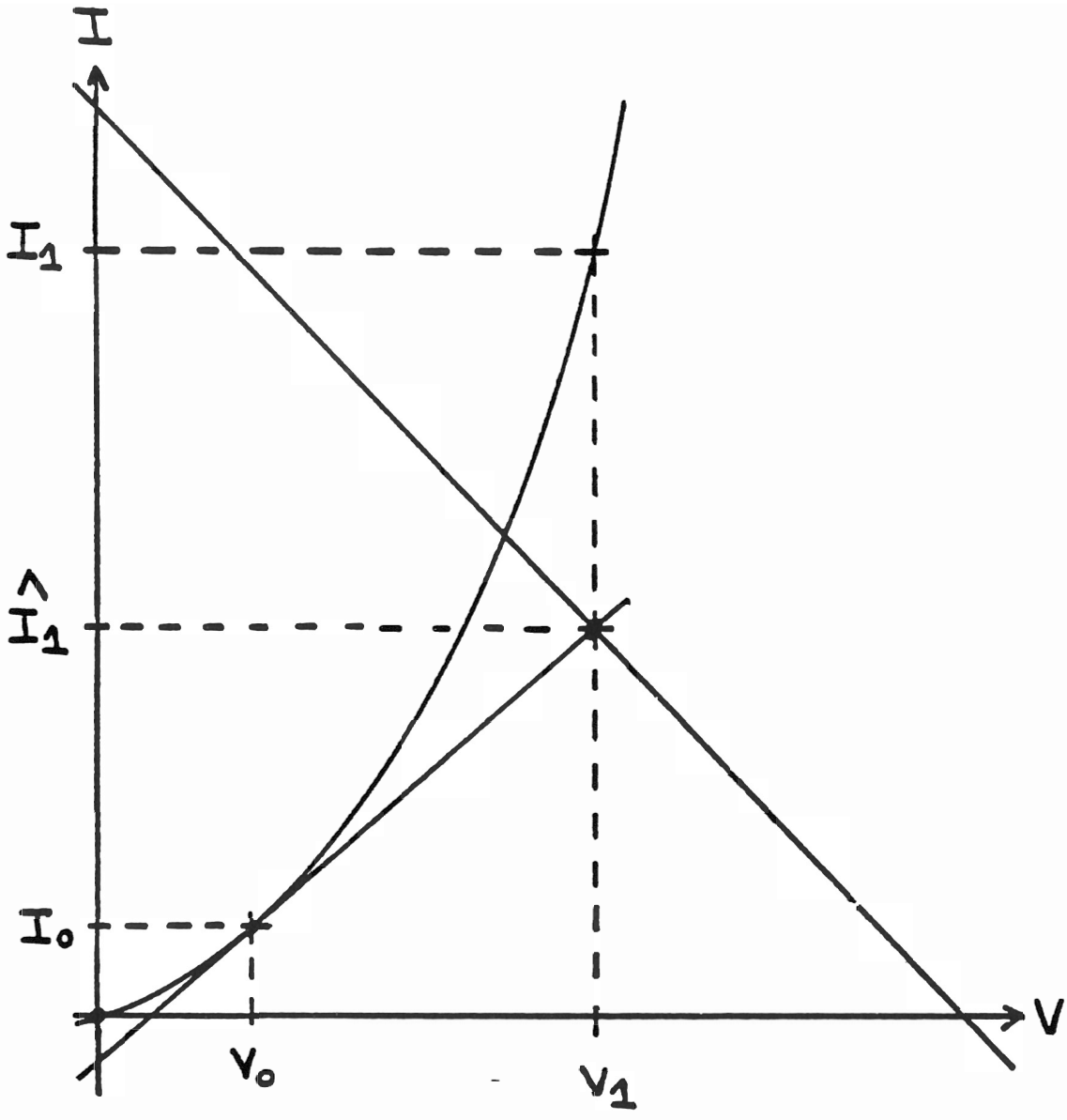


Fig. 5.7. An Example Sequence of Newton Iterations.

Appendix 1 of this thesis. SPICE 1 employs (5.21) with $\varepsilon_r = 0.001$ and $\varepsilon_a = 50$ μv. SPICE 2 employs (5.22) with $\varepsilon_r = 0.001$ and $\varepsilon_a = 10^{-12}$.

The results of these tests are listed in Table 5.2. For each circuit, the number of timepoints, the number of Newton iterations, and the cpu execution time are listed. Neither test used a timestep control; instead, both tests were run with a fixed timestep for 100 timepoints.

The data that is given in Table 5.2 indicates that the more conservative convergence criteria of (5.22) requires 10% to 50% more Newton iterations as compared to the node voltage criterion of (5.21). This increase is the direct result of requiring a more accurate solution. Since (5.22) guarantees a solution accuracy that is at least equal to the specified tolerance, where (5.21) guarantees nothing, the criterion of (5.22) is preferable.

A more subtle observation from the data in Table 5.2 is the fact that the cpu execution time, per Newton iteration, is 10% to 30% larger for SPICE2 than for SPICE1. This slight increase is due to the use of the MNA equation formulation method in SPICE2, as compared to the simpler CANCER modification of Nodal Analysis that is employed in SPICE1.

5.5. Modified Newton-Raphson Methods

The Newton-Raphson algorithm is ill-suited for the systems of equations that typically are generated in the analysis of typical electronic circuits. The problems that are encountered in the use of the Newton-Raphson algorithm occur when the iterate solution is not close to a correct solution.

	SPICE 1Q	SPICE 2
DIFFPAIR	100/272/2.803	100/299/3.867
RCA3040	100/264/6.896	100/302/10.132
UA709	100/252/11.012	100/303/15.777
UA741	100/203/10.897	100/299/20.022
UA727	100/220/12.677	100/323/23.885
RTLINV	100/289/1.722	100/353/2.498
TTLINV	100/340/5.888	100/388/7.923
ECLGATE	100/274/6.808	100/363/10.465
MECLIII	100/289/9.773	100/345/13.860
SBDGATE	100/305/11.142	100/393/16.183

Table 5.2. Comparison of the Node Voltage Convergence Criterion and the Branch Convergence Criterion.

The first problem that is encountered with the use of the Newton-Raphson algorithm is numerical overflow. This problem is illustrated by the graphical solution that is illustrated in Fig. 5.8. For the initial guess v_o, the next iterate solution v_1 produces a value of diode current that cannot be evaluated. It is not uncommon for the branch voltage of an ideal diode, at a given iteration, to be 10 volts or larger. The diode current that corresponds to a diode voltage of 10 volts requires the evaluation of the exponential of 400. Clearly, such a large number cannot be represented on most digital computers.

The second problem in the use of the Newton-Raphson algorithm is more subtle. This problem is illustrated in the graphical solution that is given in Fig. 5.9. This graph corresponds to the circuit that is given in Fig. 5.3 when the junction diode is replaced by a tunnel-diode. The tangent T_1 and the load-line intersect at the solution v_2. However, at v_2, the slope of the tunnel-diode characteristic has reversed, and the intersection of the tangent T_2 and the load-line result in a solution that is close to v_1. The Newton sequence of iterations for this example will oscillate between the solution of v_1 and v_2 and never approach the correct solution v_3. If the circuit contains only one nonlinearity, the oscillatory solutions are a direct result of a nonmonotonic nonlinearity. However, when more than one nonlinearity is present in the circuit, oscillatory solutions can be encountered even if all of the nonlinearities in the circuit are monotonic.

Several modifications of the Newton-Raphson method have been proposed that avoid the problems of numerical overflow and, to some

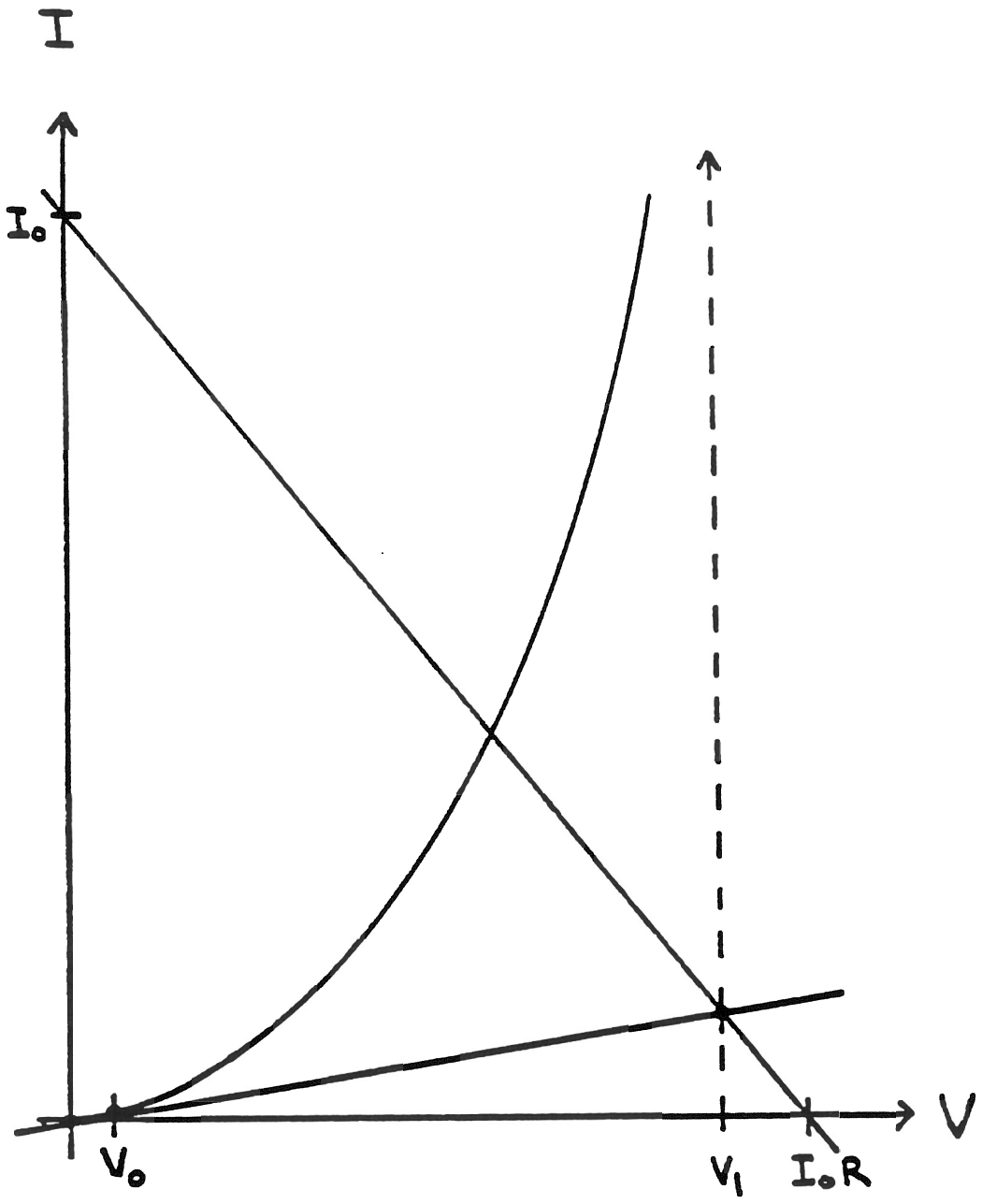


Fig. 5.8. Overflow Problem With the Diode Circuit.

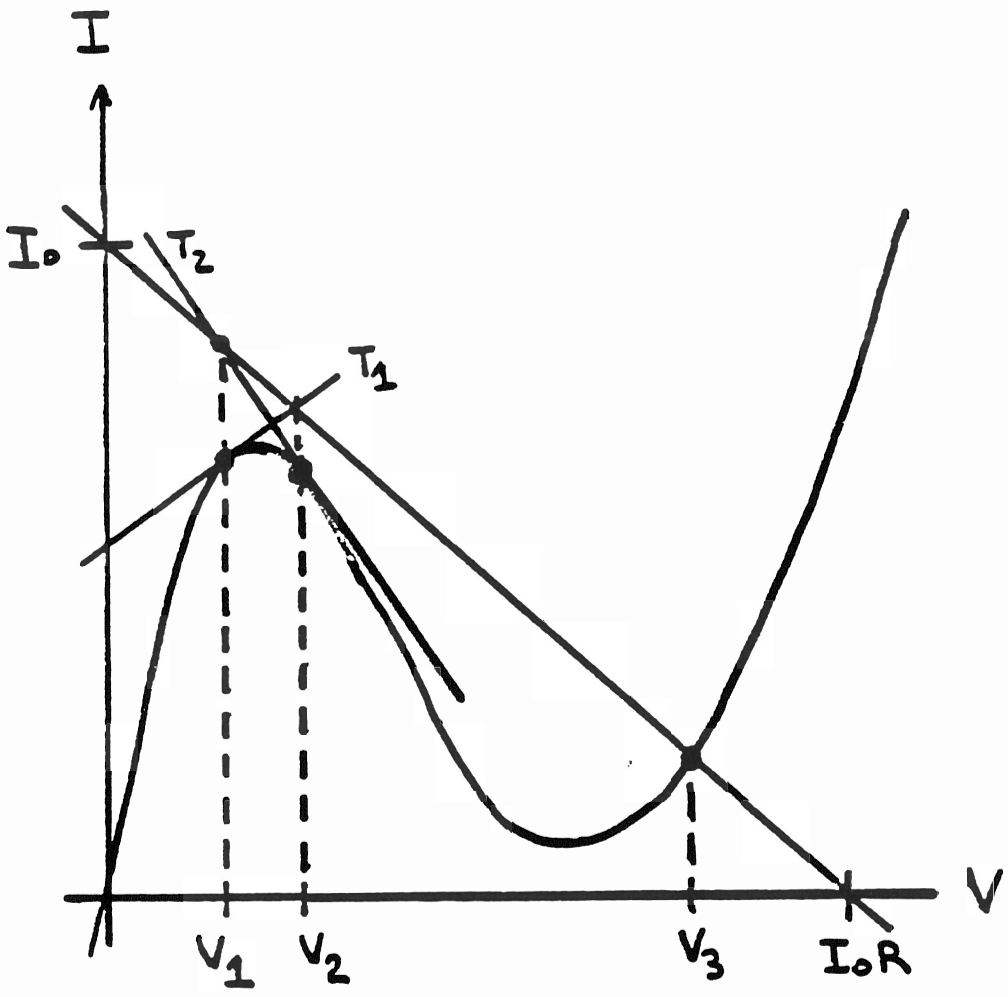


Fig. 5.9. Graphical Representation of a Tunnel-Diode Circuit.

extent, improve the convergence properties of the basic Newton-Raphson algorithm. The MS report of Kao [58] is devoted to the comparison of the various modifications of the Newton-Raphson algorithm that have been proposed.

The modifications of the Newton-Raphson method can be divided into three categories: simple-limiting algorithms, error-reduction algorithms, and source-stepping algorithms. Of the three modifications, the simple limiting methods are the easiest to implement, require the fewest number of iterations when they converge, and are the most prone to nonconvergence. Error-reduction algorithms and source-stepping techniques are more complicated and require more iterations to converge, however, these algorithms determine the solution in a more reliable fashion.

5.6. Simple-Limiting Algorithms

All of the algorithms of this class limit the nonlinear branch voltages in some fashion to prevent overflow and, hopefully, to aid convergence. The simplest method of this class is the method employed in CANCER [2]. For a diode nonlinearity, the branch voltage excursion from one iteration to the next is limited to an empirical factor of 2 V_t. The flowchart for this method is shown in Fig. 5.10. The choice of 2 V_t and 10 V_t was made from empirical observation.

The method of alternating bases was implemented first in the CIRCUS program [45] and later was implemented in the BIAS3 program [59]. The iterative solution method logically evaluates the next iterate solution in terms of the branch voltage, for current-defined branches, and in terms of current, for voltage-defined branches.

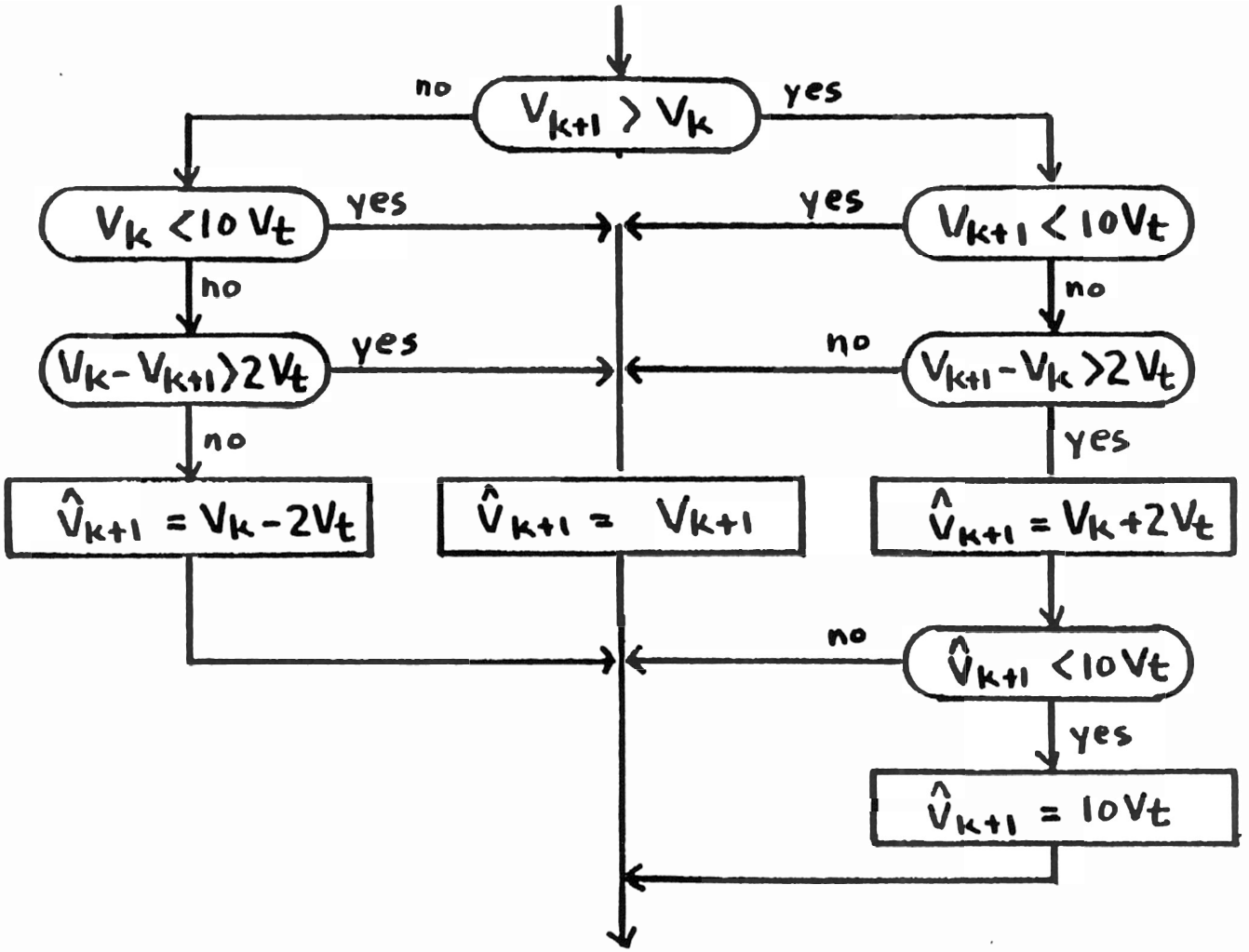


Fig. 5.10. Flowchart for the CANCER Simple-Limiting Method.

This corresponds to returning to the function value with the vertical line A in Fig. 5.11. However, if the function has an inverse, one could also return to the nonlinear function along the horizontal line B in Fig. 5.11. For a current-defined branch, line A in Fig. 5.11 is termed iteration on voltage whereas line B is termed iteration on current.

The method of alternating bases, as applied to a diode nonlinearity, uses iteration on voltage if the iterate branch voltage is negative or is less than the last iterate voltage, and uses iteration on current if the branch voltage is positive and is larger than the last iterate voltage.

Another modification of the Newton-Raphson algorithm that is based upon alternating bases was proposed by Colon and implemented by Kao [58]. For this algorithm, iteration on current is employed if V_{k+1} exceeds a critical junction voltage that corresponds to an empirically determined diode current. Hence, if $V_{k+1} > V_{crit}$ and $I_{k+1} > 0$, iteration on current is used to establish $\hat{V}_{k+1}$. If $V_{k+1} > V_{crit}$ and $I_{k+1} < 0$, the $\hat{V}_{k+1}$ is set the V_{crit}. Finally, if $V_{k+1} < V_{crit}$, the $\hat{V}_{k+1}$ is set to V_{k+1}, that is, iteration on voltage is used. Best empirical results were obtained when V_{crit} was set to the point of minimum radius of curvature:

$$V_{crit} = V_t \log\left(\frac{V_t}{I_S\sqrt{2}}\right) \tag{5.23}$$

These methods of simple limiting by no means exhaust the list of simple limiting methods that have been proposed. However, the remaining methods that have been proposed are either simple variations of these three methods, or were found by Kao [58] to be considerably

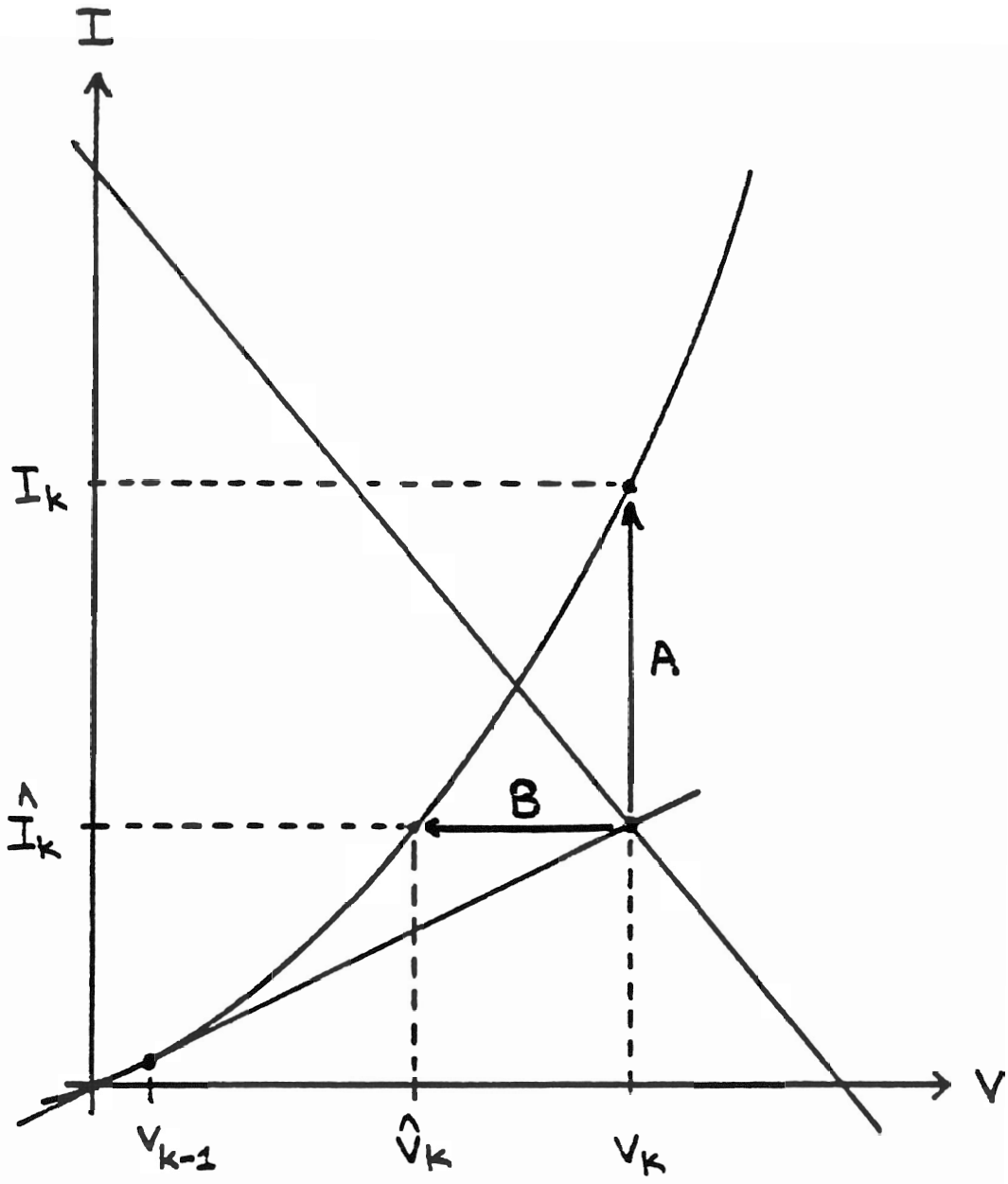


Fig. 5.11. Comparison of Iteration on Voltage and Iteration on Current.

less reliable or to require considerably more iterations.

5.7. Comparison of the Simple-Limiting Methods

All three of the simple-limiting methods were implemented in SPICE. The branch current convergence criterion that is given in (5.22) was used with $\varepsilon_r = 0.001$ and $\varepsilon_a = 10^{-12}$. The starting point was chosen as base-collector junctions off (0 volts) and diode and base-emitter junctions were started at their point of minimum radius of curvature [2] as given by (5.23).

These simple-limiting methods were tested with a total of eighteen test circuits. These circuits are detailed in Appendix 1 of this thesis. The results for these tests are given in Table 5.3. The number of Newton iterations, followed by the cpu execution time, is given for each circuit. The inevitable differences between the results that are given in Table 5.3 and the results that are cited by Kao are due to the different convergence criterion, the difference starting conditions, and a slightly different transistor model. The poor performance of the CANCER method that was cited by Kao was traced to a difference in coding of the CANCER method by Kao.

As the data in Table 5.3 indicates, there is little difference between the three methods for most cases. The five TTL circuits (TTLINV, 74TTL, 74STTL, 74LTTL, and 9200TTL) proved to be most troublesome convergence problems. The CANCER method fails on the 74STTL, 74LTTL, and the 9200TTL circuits. The BIAS 3 method, on the other hand, fails only on the 9200TTL. Finally, the COLON method solved all five TTL circuits successfully.

These tests indicate that the COLON method is the most reliable for the circuits that were tested. In addition the COLON method

	CANCER	BIAS3	COLON
DIFFPAIR	8/0.223	8/0.224	8/0.210
RCA3040	8/0.463	8/0.467	8/0.433
UA709	14/0.971	12/0.873	12/0.822
UA741	14/1.236	12/1.117	12/1.020
UA727	13/1.304	13/1.290	13/1.204
RTLINV	14/0.194	10/0.169	10/0.163
TTLINV	24/0.642	15/0.427	15/0.395
ECLGATE	14/0.577	11/0.507	9/0.447
MELLIII	14/0.782	12/0.715	13/0.770
SGOGATE	23/1.203	22/1.172	23/1.236
CCSOR	13/0.344	13/0.334	13/0.331
OSC	15/0.425	13/0.396	13/0.390
UA733	8/0.465	8/0.467	8/0.464
CFFLOP	7/0.215	7/0.217	7/0.225
74TTL	38/0.921	47/1.142	37/0.945
74STTL	*	72/2.043	27/0.888
74LTTL	*	92/2.102	54/1.301
9200TTL	*	*	27/0.792

*No convergence after 100 iterations

Table 5.3. Comparison of the Results for the Simple Limiting Methods.

requires the minimal number of iterations for every circuit tested except the MECLIII circuit, where the BIAS3 method required one less iteration.

These methods were tested only with bipolar transistor circuits. Both the CANCER method and the COLON method can be extended to limit FET nonlinearities as well. By nature, FET nonlinearities are much less severe than bipolar nonlinearities, and are not prone to the overflow problems that are encountered in the simulation of bipolar circuits. The very limited experience in the simulation FET circuits with SPICE has indicated that limiting algorithms are not necessary. However, in anticipation of FET convergence problems, a simpler CANCER-type limit is implemented in both SPICE programs.

5.8. Error-Reduction Algorithms

Error-reduction algorithms attempt to overcome the convergence difficulties that are encountered with the Newton-Raphson algorithm by limiting the nonlinear arguments at each iteration such that a measure of error is consistently reduced. This is accomplished by choosing the k+1 iterate solution, x_{k+1}, according to the equation

$$\hat{x}_{k+1} = x_k + \gamma_k \delta x_k \tag{5.24}$$

where γ_k is the stepsize. If γ_k is unity, then (5.24) is equivalent to (5.2). The error at the kth iteration is given by

$$\varepsilon_k = G(x_k)^T G(x_k) \tag{5.25}$$

The objective of the algorithm, then, is to choose a stepsize γ_k such that ε_k is minimal [32].

If the algorithm is successful, then each iterate solution will be closer to the exact solution. If the method does not converge, the last iterate solution is at least as accurate, as any of the proceeding iterations. There is, of course, no such guarantee for the simple-limiting algorithms.

The same stepsize, γ_k, is applied to every nonlinear argument in the circuit. This method differs markedly from the CANCER method or the COLON method which limit each branch voltage or current with a different equivalent stepsize.

The choice of the stepsize is determined in the following manner. Equation (5.1) is expanded in a Taylor series about $\hat{x}_{k+1}$ to yield

$$G(\hat{x}_{k+1}) = G(x_k+\gamma_k\delta x_k) \cong G(x_k) + \gamma_k J(x_k)\delta x_k \tag{5.26}$$

If γ_k is unity, then (5.26) reduces to (5.3). Combining (5.2) and (5.26) yields the equation.

$$G(\hat{x}_{k+1}) = (1-\gamma_k)\ G(x_k) \tag{5.27}$$

Finally, (5.25) and (5.27) are combined to yield

$$\frac{\epsilon_k(\gamma_k)}{\epsilon_k(0)} = 1 - 2\gamma_k + \gamma_k^2 = (1-\gamma_k)^2 \tag{5.28}$$

The ideal parabolic nature of the error that is given in (5.28) is distorted in proportion to the severity of the nonlinearities in the circuit. Four graphs of the relative error, as a function of stepsize, are given in Fig. 5.12. The curve D is the ideal parabolic error characteristic. The curves A and B correspond to error characteristics for which the minimum error occurs at a stepsize

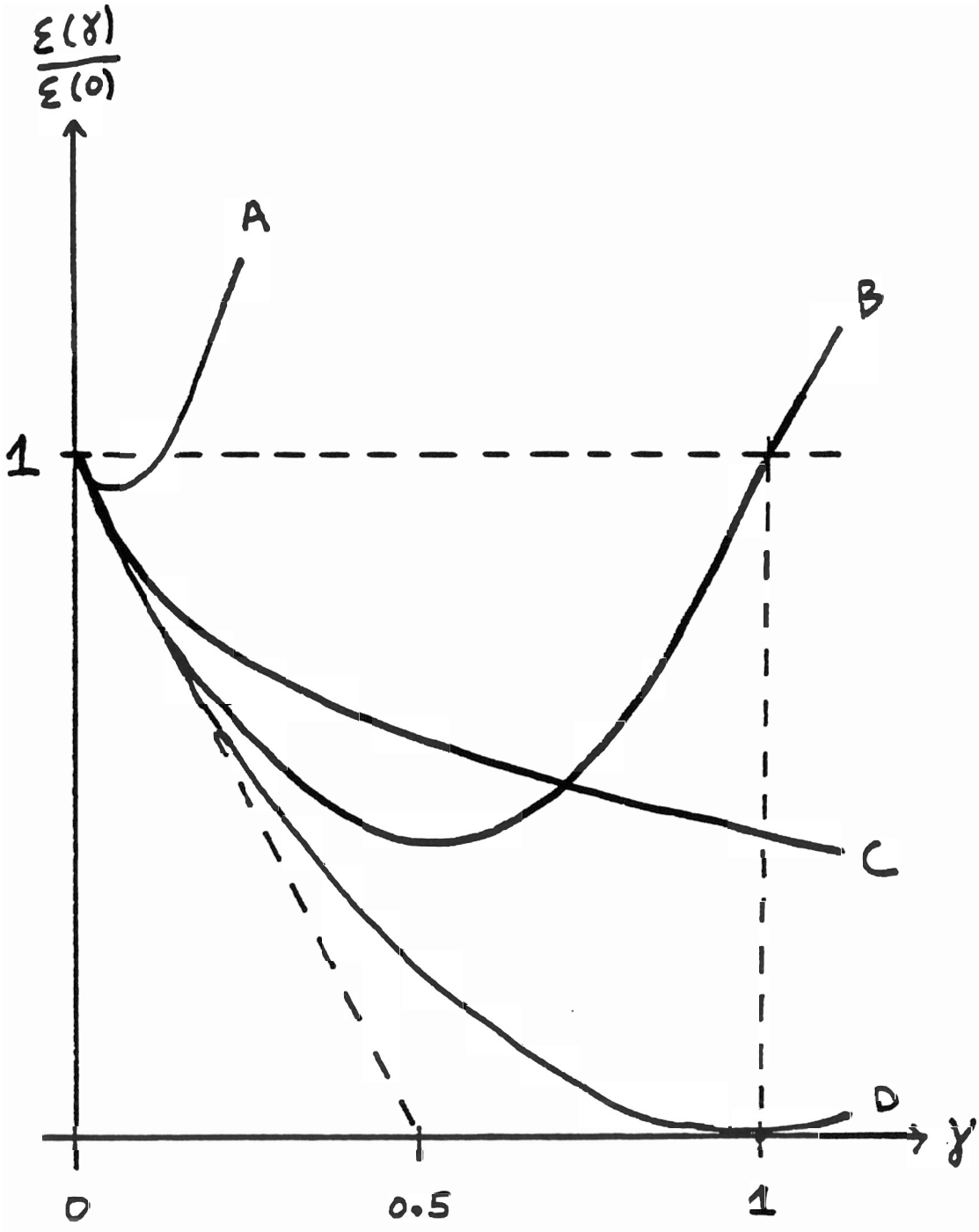


Fig. 5.12. Graph of Error Versus Stepsize for the Error Reduction Method.

value that is less than unity. Curve C, on the other hand, is an error characteristic for which the minimum error occurs at a stepsize value that is larger than unity. In all cases, the initial (γ_k=0) value of slope of the curves is -2 [32].

The strategy for determining γ_k is based on the parabolic nature of the error ε_k. The method that is given here closely follows the algorithm that is used in ECAP 2 [32]. At the kth iteration, the error for zero stepsize, $\varepsilon_k(0)$, and the error for unity stepsize, $\varepsilon_k(1)$, are evaluated. A trial stepsize, $\hat{\gamma}_k$, is ten determined by fitting parabola to these two error values and the initial slope of -2. The error for the trial stepsize then is evaluated.

If $\varepsilon(\hat{\gamma}_k)$ is less than both $\varepsilon(1)$ and $\varepsilon(0)$, the $\hat{\gamma}_k$ is used as the stepsize. This case corresponds to curve B in Fig. 5.12. If $\varepsilon(1)$ us less than both $\varepsilon(\hat{\gamma}_k)$ and $\varepsilon(0)$, a unity stepsize is used. This case corresponds to curve C in Fig. 5.12. Finally, if $\varepsilon(0)$ is smaller than both $\varepsilon(\hat{\gamma}_k)$ and $\varepsilon(1)$, as is the case for curve A in Fig. 5.12, $\hat{\gamma}_k$ is halved until $\varepsilon(\hat{\gamma}_k)$ is less than $\varepsilon(0)$. To prevent an infinite loop, the analysis is method if $\hat{\gamma}_k$ is less than 10^{-6}.

Every evaluation of the error ε_k requires an assembly of the circuit equations. As long as $\varepsilon(0)$ is greater than either $\varepsilon(\hat{\gamma}_k)$ or $\varepsilon(1)$, each error reduction step requires two equation formulation steps and one or two equation solutions, depending upon whether $\varepsilon(\hat{\gamma}_k)$ is greater than or less than $\varepsilon(1)$. Each construction of the circuit equations is counted as one Newton iteration.

This error-reduction algorithm was applied to the same circuits that were used to compare the simple-limiting metals. The same error tolerances and initial conditions were used. The final error, the

number of Newton iterations, and the cpu execution time for these eighteen circuits are given in Table 5.4.

The error-reduction algorithm did not find an acceptable solution for the UA 733 circuit, the 74 STTL circuit, the 74 LTTL circuit, and the 9200 TTL circuit. Subsequent adjustments to the algorithm failed to produce acceptable error levels for these four circuits. Furthermore, the number of Newton iterations that are required to solve the fourteen remaining circuits are between 40% and 1230% larger than the simple-limiting COLON method. The error-reduction method, therefore, does not yield "failsafe" convergence properties and requires a large increase in the number of Newton iterations that are required to determine the dc operating point.

5.9. Source Stepping Algorithms

This class of nonlinear solution methods originally was introduced by Davidenko [60]. In essence, the method of source-stepping is equivalent to determining the dc operating point with a dc transfer curve. The vector of independent source values, E, is determined by

$$E = \alpha S \tag{5.29}$$

where S is the actual value of source values and α is a scalar with a value between zero and unity. Scalar α then is gradually increased from zero, where the solution obviously is zero, to unity. At each step, α is increased by an amount for which convergence is obtained with the unmodified Newton-Raphson algorithm.

Although the method appears to be failsafe, it will not always converge to a solution. Consider again the simple tunnel-diode circuit

	ERROR	ITERATION	CPU
DIFFPAIR	1.5E-9	41	0.823
RCA3040	2.3E-7	28	1.479
UA709	7.5E-11	52	3.806
UA741	4.6E-10	46	4.375
UA727	4.8E-10	160	20.374
RTLINV	3.9E-10	14	0.215
TTLINV	1.7E-7	34	1.077
SCLGATE	6.0E-14	41	1.414
MELLIII	3.4E-9	25	1.234
SBDGATE	2.6E-3	90	2.171
CCSOR	8.3E-10	25	0.529
OSC	2.2E-8	42	0.963
UA733	10.75*	5*	0.811*
CFFLOP	5.6E-8	14	0.312
74TTL	5.3E-9	77	1.850
74STTL	9.2E-3*	112*	7.688*
74LTTL	1.2E-3*	68*	4.052*
9200TTL	7.3E-2*	87*	3.994*

* Algorithm failed to reduce error to a suitable level

Table 5.4 The Results for the Error-Reduction Algorithm

with the "load-line" solution that is given in Fig. 5.13. The load-line A in this figure corresponds to the solution for α equal to zero, the load-line B corresponds to α equal to 0.9, and the loud-line C corresponds to α equal to unity. Since only independent source values are affected by α, the slopes of A, B, and C are equal.

The source-stepping algorithm proceeds smoothly from $\alpha = 0$ to $\alpha = 0.9$. However, in this example, if α is greater than 0.9 by the least amount, the solution jumps abruptly from point a to point b. Now matter how small the variation in α is, the same oscillatory behavior that is encountered with simple-limiting methods also will occur with a source-stepping algorithm.

Kao has shown [58] that the particular source-stepping method of Broyden [61] requires twice the number of iterations that the ECAP 2 method [32] requires and four times the number iterations that the simple-limiting method of COLON requires.

The solution method that is used in ASTAP [29] is an interesting modification of the source-stepping algorithm. It is shown in Chapter 3 of this thesis that the Modified Tableau formulation method that is employed in ASTAP produces numerically ill-conditioned equations for dc analysis. For this reason, the methods that are presented in this chapter could not be used directly in ASTAP.

Instead, the dc solution actually is obtained by a transient analysis. This is accomplished by introducing "pseudo-reactances" [29] into the circuit for the dc analysis. A pseudo-inductor is inserted in series with each independent voltage source and each nonlinear voltage-defined branch, and a pseudo-capacitor is inserted

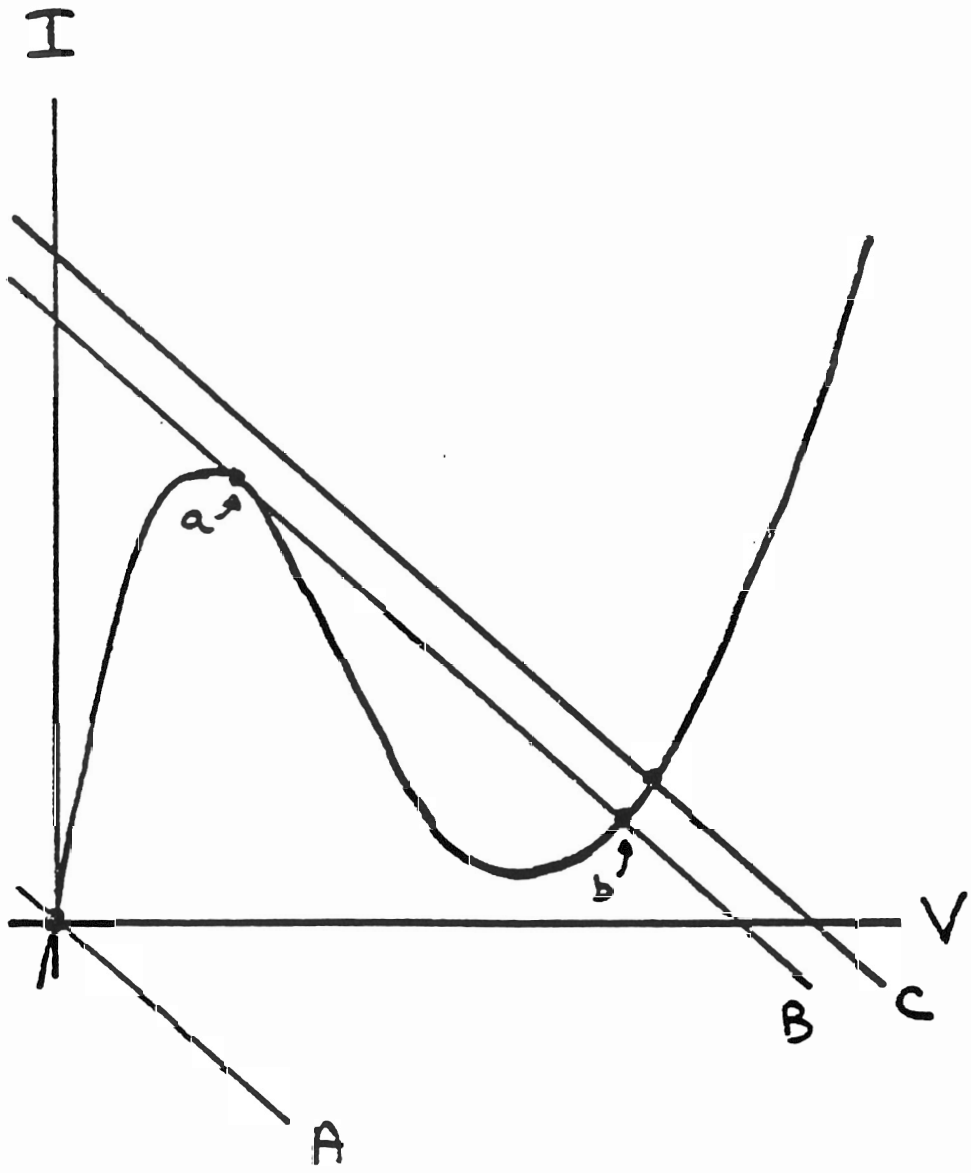


Fig. 5.13. A Nonconvergence Example Using the Source-Stepping Method.

in parallel with each independent current source and each nonlinear current-defined branch. The initial conditions for these pseudo-reactances are chosen such that the initial transient solution is zero.

A transient analysis then is performed upon this pseudo-circuit. The transition from the zero solution to the final solution is of no interest in this analysis. Hence, the truncation error can be ignored as long as the integration algorithm is stable, that is, as long as the solutions converge to the correct equilibrium solution. For this reason, the timestep that is chosen is not determined by accuracy considerations; instead, the timestep is taken as large as possible, consistant only with the convergence of the unmodified Newton-Raphson algorithm.

This method was implemented in SPICE in the following manner. A 1 henry inductor was inserted in series with each independent voltage source, and a 1 farad capacitor was inserted in parallel with each independent current source. The built-in capacitances of the BJT model were used as the pseudo-capacitors for the nonlinear branches. The initial conditions for the source reactances were chosen to produce a zero initial solution.

The ten standard benchwork circuits that are described in Appendix 1 of this thesis then were used to test the method. The transient analysis employed the Backward Euler integration method and the following timestep algorithm. If the number of Newton iterations at a timepoint exceeded 10, the timestep was reduced by a factor of 8. If the number of iterations was less than 5, the timestep was double. The analysis was halted when the independent

voltage source inductor voltages were less than 10^{-9} and the independent current source capacitor current were less than 10^{-12}.

The results of these tests are given in Table 5.4. The tests were conducted both with the unmodified Newton-Raphson algorithm and with the simple-limiting method of COLON. The first column for both cases lists the number of timepoints required to obtain convergence; the second column presents the number of Newton iterations that were required; the third column is the cpu execution time for the analysis.

The first observation to be implied from this data is that the simple-limit method results in appreciably fewer iterations in many cases even when this quasi-dc method is used. The UA 709 circuit, for example, requires in excess of 1000 iterations if no limiting is employed, yet with the COLON limit only 612 iterations are required. The remaining circuits require 17% fewer iterations when the COLON limit is used. Only in the case of the MELL111 and the SBOGATE circuits does the simple-limit fail to produce a significant reduction in iteration count.

The second, and more important, conclusion is that this quasi-dc analysis requires substantially more iterations than the simple-limit methods. The difference ranges from a factor of 9.6, for the TTL1NV circuit, to a factor of 51 for the UA 709 circuit. The average number of Newton iterations for the two methods differs by a factor of 19. This marked increase in computational effort clearly dues not justify any advantages that the method may contain.

5.10. Multiple-Point Analysis

The nonlinear solution algorithm is employed both in the dc

	ASTAP METHOD, NO LIMITING			ASTAP METHOD, COLON LIMIT		
	TIMEPOINTS	ITER	CPU	TIMEPOINTS	ITER	CPU
DIFFPAIR	36	148	2.339	30	109	1.746
RCA3040	47	224	8.307	41	183	6.356
UA709	>149	>1000	>56.964	103	612	34.906
UA741	42	184	12.789	40	169	11.819
UA727	76	414	33.247	60	319	26.044
RTLINV	53	265	2.530	38	160	1.567
TTLINV	40	166	3.632	37	144	3.174
ECLGATE	47	202	6.353	43	177	5.602
MELLIII	41	161	6.658	41	158	6.611
SBDGATE	51	246	10.643	52	248	10.797

Table 5.4. Results for the ASTAP Method of DC Analysis

operating point analysis and in the solution for the dc transfer characteristics of the circuit or the transient response of the circuit. In the latter two analyses, many sweep points are computed although the number of iterations per point is relatively small. The prediction and bypass methods that are presented in this section reduce the computational effort at each sweep point.

In a multiple-point analysis, the solution at the last point usually is a good estimate for the solution. This estimate can be improved, in most cases, by utilizing previous points to predict the solution. In a transient analysis, for example, the simplest predictor is the first-order equation

$$\hat{x}_{n+1} = x_n + \frac{h_{n+1}}{h_n} (x_n - x_{n-1}) \qquad (5.30)$$

Without prediction, the vector x_n is used to start the solution for x_{n+1}. The various predictior methods, on the other hand, use the vectors x_{n+1} as the starting guess. Equation (5.30) is a first-order predictor. The timepoints x_n, x_{n-1}, and x_{n-2} can be used to provide a second-order predictor. Similiarly, k-1 back timepoints produce a kth order predictor.

The transient analysis of the five transient test circuits and of the ten standard benchmark circuits was used to test the prediction algorithm. In this series of tests, the iteration timestep control was used. Specifically, the timestep is reduced by a factor of 8 if the iteration count at a given timepoint exceeds 10 and the timestep is doubled if the iteration count is less than 5. The timestep never is allowed to exceed the user-defined print increment. The implementation of a prediction algorithm, of course should have a

negligible effect on the number of timepoints that are computed. Moreover, the prediction algorithm is independent of the type of timestep control that is employed.

The results of the tests of the prediction algorithm are given in Table 5.5. The first column in this table lists the number of timepoints, number of Newton iterations, and the cpu execution time for the transient analyses without prediction. The second column presents the same three figures of merit for the analyses when prediction is employed.

It is evident from this data that the prediction algorithm given in (5.30) yields a decrease in Newton iterations for all of the analyses except the RC and SCHMITT circuits. The decrease in iterations ranges from 1% for the ASTABLE circuit to 20% for the DIFFPAIR circuit. Predictors of higher order than (5.30) resulted in little improvement over the results that are cited in Table 5.5.

The second computational improvement for multiple-point analyses is a bypass algorithm. With this algorithm, a nonlinear branch relation and its derivative is evaluated only if the difference in the arguments for that branch relation change appreciably. Otherwise, the values at the last iteration are used [62].

The diode characteristic that is shown in Fig. 5.7 illustrates this algorithm. After the first iteration, V_1 and $\hat{I}_1$ are computed without evaluating the diode Eq. (5.13). To determine the second iterate solution, the diode equation and its derivative must be evaluated either at point a (iteration on voltage) or at point b (iteration on current). However, if $|\hat{I}_1-I_o|$ and $|V_1-V_o|$ are small, then the equivalent conductance g_{eq} and independent current source

value I_{eq} will not differ appreciably between the first and second iteration. In this case, the values of g_{eq} and I_{eq} that were used for the first (or k) iteration. also can be used for the second (or k+1) iteration.

The bypass algorithm proceeds as follows. At the k iteration, $\hat{I}_k$ and V_k are evaluated. If $g_{eq} > 1$ and

$$|\hat{I}_k - I_{k-1}| < \varepsilon_a \text{ Max}\{|\hat{I}_k|, |I_{k-1}|\} \tag{5.31}$$

or if $g_{eq} < 1$ and

$$|V_k - V_{k-1}| < \varepsilon_a \text{ Max}\{|V_k|, |V_{k-1}|\} + \varepsilon_r \tag{5.32}$$

then the values of g_{eq} and I_{eq} for the k iteration also are used for the k+1 iteration.

The results of the test of the bypass algorithm also are given in Table 5.5. The Third column contains the number of timepoints, number of Newton iterations, and cpu execution time for the analyses of the fifteen test circuits. For these tests, the bypass algorithm yields a very minor improvement in cpu execution time. The best improvement is a 8.9% decrease in cpu time for the ASTABLE analysis; however, the cpu time for the RC, SCHMITT, and RTLINV analyses was not decreased at all. On the average, the cpu execution time was decreased only 4%.

5.11. Conclusions

The Newton-Raphson algorithm is the nonlinear solution method that is used in most circuit simulation programs. The Secant Method also has been proposed for use in circuit simulation, chiefly because the nonlinear function derivatives are not evaluated in the Secant

	No Prediction	Prediction	Prediction and Bypass
RC	109/218/0.418	109/218/0.422	109/218/0.435
CHOKE	110/385/1.599	110/397/1.689	110/395/1.650
ECL	110/264/3.448	109/247/3.349	109/247/3.168
SCHMITT	120/348/4.735	131/360/5.024	131/360/4.974
ASTABLE	217/957/7.896	218/946/7.913	198/874/7.235
DIFFPAIR	106/312/4.033	106/248/3.264	106/298/3.120
RCA3040	106/310/10.396	106/293/10.059	106/293/9.521
UA709	106/315/16.383	106/299/15.833	106/300/14.470
UA741	106/312/20.882	106/253/17.039	106/252/15.750
UA727	106/336/24.825	106/307/22.950	106/308/21.887
RTLINV	116/392/2.788	116/314/2.277	116/316/2.263
TTLINV	130/500/10.224	132/453/9.407	132/454/9.205
ECLGATE	121/422/12.132	121/361/10.489	121/361/10.191
MECLIII	116/396/15.911	116/335/13.646	116/338/12.957
SBDGATE	118/431/17.611	120/421/17.653	120/423/17.140

Table 5.5. The Results for the Prediction and Bypass Algorithms.

Method. However, the slower convergence rate of the Secant Method is not compensated by this minor computational advantage.

Neither the Newton-Raphson method nor the Secant Method can be applied directly to the solution of most nonlinear circuit equations. Difficulties arise due to the exponential nature of the physical equations that model junction diodes and bipolar junction transistors. These exponential nonlinearities often result in numerical overflow if the solution algorithm is not tailored to suit the needs of circuit simulation.

The first contribution of this chapter is the development of a meaningful convergence criterion. Most Nodal Analysis programs, SPICE1 included, use the criterion that each node voltage must agree within some tolerance before the iterative process is halted. The occurance of false convergence for this method was demonstrated, and a method of circumventing this false convergence was proposed. Basically, the new convergence criteria requires that, for each nonlinear branch, the linearized branch relation must agree with the nonlinear branch relation within a given tolerance. The convergence criterion is, therefore, a part of the nonlinear solution method and does not depend upon which circuit variables are used to formulate the circuit equations. This new convergence criterion is implemented in the SPICE2 program.

Next, simple-limiting methods were presented. The three algorithms that were compared are the CANCER method, the BIAS3 method, and the COLON method. Tests on several circuits indicated that there is little difference between these methods for most circuits. However, the COLON method proved to be the most reliable of the three;

moreover, this method required a minimal number of iterations to converge. For this reason, the COLON simple-limit method is implemented in the SPICE2 program. However, substantial experience with SPICE1, which utilizes the CANCER method, has shown that the CANCER method also is a very effective limiting technique.

The more elaborate methods of error-reduction and source-stepping then were detailed. Neither method was found to be as reliable as the COLON method for the circuits that were tested. Moreover, both methods require substantially more Newton iterations to attain convergence. Therefore, neither method is attractive for the main dc operating point algorithm. However, the dc transfer-curve capability of SPICE often is used as a "backup" source-stepping method when the simple-limiting method of COLON fails.

Finally, additional refinements to the solution method were presented which reduce the computational effort for a multiple-point analysis. The use of first-order prediction algorithm in transient analysis was found to result in a 5% to 10% decrease in iteration count. A bypass algorithm that avoids recomputing nonlinear functions that are not changing from iteration to iteration also was proposed. This bypass scheme resulted in an additional 4% improvement in cpu execution time. Both the first-order predictor and the bypass algorithm are retained in the SPICE2 program. However, the improvements afforded by these two algorithms is minimal and these methods could as well be omitted.

VI. NUMERICAL INTEGRATION

A numerical integration method is necessary to determine the transient response of a circuit over a specified time interval (0,T). The circuit is represented by a system of algebraic-differential equations of the general form:

$$F(x,\dot{x},t) = 0 \tag{6.1}$$

where x is the vector of unknown circuit variables, $\dot{x}$ is the time-derivative of x, and F is, in general, a nonlinear operator. A transient analysis of the circuit divides the interval (0,T) into the discrete timepoints $(0,t_1,t_2,\ldots,T)$ and determines the solution of (6.1) at each timepoint. The initial time-zero timepoint is specified by the user or, more conveniently, is determined by a separate dc analysis.

A numerical integration algorithm, then, determines the solution x_{n+1} at timepoint t_{n+1} given the previously computed solutions $(x_n,x_{n-1},\ldots)$. The local truncation error (LTE) of of the method is the error in the solution x_{n+1} assuming that the previous solutions $(x_n,x_{n-1},\ldots)$ are exact. LTE, in turn, is proportional to the timestep h_n which is defined by

$$h_n = t_{n+1} - t_n \tag{6.2}$$

The total error in x_{n+1} is determined both by the LTE that is incurred at the timepoint t_{n+1} and the LTE of the previous solutions. The stability characteristics of the numerical integration algorithm determine the significance of the LTE at

previous timepoints to the total error of the solution x_{n+1} [63,64,65].

Numerical integration methods, therefore, are compared on the basis of LTE and stability. This chapter begins with an introduction of explicit and implicit integration methods. The properties of LTE and stability are developed using the explicit Forward Euler algorithm and the implicit Backward Euler method.

Next, the integration algorithms that have been employed in circuit simulation programs are presented. These methods include the Trapezoidal Rule algorithm that is used in CANCER [2] and SPICE1 [1], the fixed timestep Gear methods [66,67], the Gear-2 method that is used in CIRPAC [41] and SINC,[1] and the Runge-Kutta method that is employed in the original version of SCEPTRE [43]. A variable-order, variable-timestep modification of Gear's methods is used in ECAP2 [31,32] and ASTAP [28,29].

Results for the fixed-timestep version of Trapezoidal integration that is employed in CANCER and SPICE1 then are presented. The addition of an iteration-count timestep is detailed. This algorithm was used first in the TRAC program [39]. Further use of this timestep control in TIME [19] and SINC has verified the usefulness of this algorithm for many circuit simulation problems.

A comparison of the results of the iteration timestep control with Trapezoidal integration and with Gear-2 integration

---

[1] SINC is a nonlinear transient analysis program that was developed by S.P. Fan at the Electronics Research Laboratory, University of California, Berkeley. The SINC program evolved from the TIME program [19].

is presented next. The results indicate that neither algorithm, with an iteration timestep control, always yields accurate results. This comparison, therefore, establishes the need for a timestep control that is based on the LTE at each timepoint.

The implementation of a LTE timestep control with Trapezoidal integration is detailed next. The pitfalls of inaccurate LTE estimation and iterative solution error are documented, and practical methods for circumventing these problems are presented. The results for the LTE timestep control then are compared with the results for the iteration-count timestep control.

Finally, the implementation of Gear's methods [66,67] is presented. These methods first are tested individually to establish the desirability of higher-order integration methods. Next, the implementation of a variable-order integration algorithm that employs Gear's methods is presented. The results of this variable order, variable-timestep modification are compared with the results for the LTE implementation of the Trapezoidal method.

6.1. Explicit Integration Methods

The explicit Forward Euler algorithm is obtained directly from a Taylor series expansion of the solution x_{n+1} at the timepoint h_n as follows:

$$x_{n+1} = x_n + h_n\dot{x}_n + \frac{h_n^2}{2}\frac{d^2x}{dt^2}(\xi) \qquad (6.3)$$

where $x_{n+1} = x(t_{n+1})$, $x_n = x(t_n)$, and $t_n \leq \xi \leq t_{n+1}$. The second-derivative form of the remainder in (6.3) is the local truncation error of the Forward Euler method.

If an explicit integration method is used, the circuit equations must be formulated in normal form as follows:

$$\dot{x} = f(x,t) \tag{6.4}$$

The circuit equations are cast in this form with the State-Variable formulation method [7].

The solution of (6.4) at the timepoint t_{n+1} is obtained by combining (6.3), without the LTE term, and (6.4) to yield

$$x_{n+1} = x_n + h_n \dot{x}_n \tag{6.5}$$

$$\dot{x}_{n+1} = f(x_{n+1}, t_{n+1}) \tag{6.6}$$

Given the initial solution x_0, (6.6) is used to determine the derivative $\dot{x}_0$. Equations (6.5-6.6) then are solved at each timepoint to determine the transient response.

The salient feature of any explicit method is its simplicity. Each timepoint solution requires one evaluation of (6.4) plus a minor amount of bookkeeping. The simplicity of an explicit method is offset by two drawbacks. First, State-Variable formulation must be used in conjunction with an explicit method. As shown in Chapter 3 of this thesis, State-Variable formulation is more complicated and less efficient than the Nodal Analysis formulation that is employed in SPICE. Second, the stability properties of most explicit methods are notably poor. The instability of explicit methods, in turn, often results in an unnecessarily small value of timestep and a commensurate increase in the number of timepoints that are computed in the transient analysis.

6.2. Implicit Integration Methods

The implicit Backward Euler algorithm is derived by expanding both x_{n+1} and $\dot{x}_{n+1}$ about the timepoint t_n as follows:

$$x_{n+1} = x_n + h_n\dot{x}_n + \frac{h_n^2}{2}\frac{d^2x}{dt^2}(\xi) \tag{6.7}$$

$$\dot{x}_{n+1} = \dot{x}_n + h_n\frac{d^2x}{dt^2}(\xi) \tag{6.8}$$

By substituting (6.8) into (6.7), the Backward Euler formula is obtained:

$$x_{n+1} = x_n + h_n\dot{x}_{n+1} - \frac{h_n^2}{2}\frac{d^2x}{dt^2}(\xi) \tag{6.9}$$

The second-derivative term in (6.9) is the local truncation error of the Backward Euler algorithm.

An implicit integration algorithm yields a relation between x_{n+1} and $\dot{x}_{n+1}$ at each timepoint. This relation is combined with the circuit equations to produce a system of algebraic equations for each timepoint. For example, if (6.9), without the LTE term, is combined with (6.4), the result is

$$x_{n+1} = x_n + h_n f(x_{n+1}, t_{n+1}) \tag{6.10}$$

The simplified system of algebraic equations for the timepoint t_{n+1} is solved by an iterative method, just as the dc operating point of the circuit is determined. The integration algorithm, then, reduces the task of a transient analysis of N timepoints to the simpler problem of N repetitive "quasi-dc" analyses.

The classical method of solving (6.10) employs functional iteration [37]. The solution x_{n+1} is predicted by an explicit predictor, such as the Forward Euler method, to yield

$$x_{n+1}^{(0)} = x_n + h_n \dot{x}_n \tag{6.11}$$

This prediction then is corrected iteratively with an implicit corrector, such as the Backward Euler algorithm, in the following manner:

$$x_{n+1}^{(k)} = x_n + h_n f(x_{n+1}^{(k-1)}, t_{n+1}) \tag{6.12}$$

The superscripts in (6.11) and (6.12) denote iteration number.

Predictor-corrector methods such as (6.11-6.12) imply a severe limitation on timestep. For the Backward Euler corrector, (6.12) will converge [37] only if

$$h_n < \left| \frac{\partial f}{\partial x} \right|^{-1} \tag{6.13}$$

The limitation imposed by (6.13) is the result of using functional iteration and not the result of using an implicit integration method.

Fortunately, there are more efficient iterative methods. The Newton-Raphson algorithm is the most commonly used method in circuit simulation programs. The timestep limitation that is imposed by Newton-Raphson iteration is much less severe than (6.13). Shichman [41] demonstrated that CIRPAC, with a predictor-corrector method, required 13 times as much execution time to perform a transient analysis compared to CIRPAC with the implicit Gear-2 method and Newton-Raphson iteration.

6.3. Polynomial Integration Methods

Most integration algorithms approximate the solution x_{n+1} in terms of the p backward solutions $(x_n,\ldots,x_{n-p+1})$. This general integration algorithm is stated by the equation

$$x_{n+1} = \sum_{i=1}^{p} \alpha_i x_{n-i+1} + \sum_{i=0}^{p} \beta_i \dot{x}_{n-i+1} \qquad (6.14)$$

If β_o in (6.14) is zero, then the method is explicit since x_{n+1} is approximated solely in terms of past timepoint solutions. Implicit methods, on the other hand, have nonzero values of β_o.

If p is unity, then (6.14) is a one-step method since x_{n+1} is determined only in terms of the solution x_n. Multi-step methods are characterized by values of p that are greater than unity.

A kth order polynomial method determines the coefficients α_i and β_i such that, if x is a polynomial of degree k or less, then (6.14) is exact. Since k+1 values of x or $\dot{x}$ are necessary to fit a polynomial of degree k, the maximum order of (6.14) is 2p. If $2p > k$, then the 2p-k coefficients are chosen to improve the accuracy or stability of the method, or to include some other useful feature.

The integration method need not be chosen such that (6.14) is exact for a polynomial. Liniger and Willoughby [68] derived several methods that are based on fitting an exponential rather than a polynomial. However, these methods require an estimate of the eigenvalues of the circuit equations.

6.4. Local Truncation Error (LTE) and Stability

The local truncation error (LTE) of an integration method is

the difference between x_{n+1} and the exact solution at t_{n+1}, given that the past solutions $(x_n, x_{n-1}, \ldots)$ are exact. If x_o is exact, then the error of x_1 is due only to the LTE in computing the solution at t_1. However, if x_1 is used to determine x_2, the error of x_2 clearly will depend on the LTE that is incurred both at t_2 and at t_1. The error at x_n, therefore, depends upon the LTE at all of the timepoints $(t_1, t_2, \ldots, t_n)$.

If an integration method is stable, then the contribution of the LTE at a particular timepoint t_k to the total error at $t_n (n>k)$ diminishes as n increases. Conversely, the contribution of the LTE at the timepoint t_k will increase without limit if the algorithm is unstable. Some integration methods are stable regardless of the timestep that is used, whereas other methods are stable only for a certain range of timestep values.

The Forward Euler and Backward Euler methods contain LTE terms of equal magnitude. However, these two algorithms have markedly different stability properties. To illustrate this point, consider the simple test equation

$$\dot{x} = \lambda x \tag{6.15}$$

The exact solution of (6.15) is, of course,

$$x = x_o \exp(\lambda t) \tag{6.16}$$

where x_o is the solution at time-zero.

To solve (6.15) with the Forward Euler method, (6.3), without the LTE term, is combined with (6.15) to yield

$$x_{n+1} = x_n[1 + h_n\lambda] \tag{6.17}$$

If the timestep is constant, (6.17) reduces to the equation

$$x_n = x_o[1 + h\lambda]^n \tag{6.18}$$

Similarly, the Backward Euler solution is obtained by combining (6.9), without the LTE term, and (6.15) to produce the equation

$$x_{n+1} = x_n \left[\frac{1}{1-h_n\lambda}\right] \tag{6.19}$$

Equation (6.19), for a fixed stepsize, reduces to

$$x_n = x_o \left[\frac{1}{1-h\lambda}\right]^n \tag{6.20}$$

If the eigenvalue, λ, is real and positive, (6.16) is unstable in the sense that x increases without limit as time approaches infinity. However, if λ is real and negative, then (6.16) is stable in the sense that x approaches zero as time approaches infinity. If the exact solution is stable, then the numerical solutions (6.18) and (6.20) also should be stable in the sense that x_n should approach zero as n approaches infinity.

For the case of a negative real eigenvalue, the Forward Euler solution, (6.18), is stable only if the timestep is in the range $0 \leq h \leq -\frac{2}{\lambda}$. The Backward Euler solution, (6.20), has markedly better stability properties since (6.20) is stable for any value of timestep.

The choice of an integration method usually involves a tradeoff between LTE and stability. Algorithms with relatively small LTE

terms tend to have poor stability characteristics, whereas methods that have good stability properties tend to be less accurate. For polynomial methods, the order of the method reflects this tradeoff. The LTE of a polynomial method [37,64] is given by

$$\varepsilon_{n+1} = C\, h_n^{k+1} \frac{d^{k+1}x}{dt^{k+1}}(\xi) \tag{6.21}$$

where C is a constant that depends upon the specific integration method. As the order is increased, the error ε_{n+1} clearly diminishes. The stability of polynomial methods, on the other hand, deteriorates as the order is increased. Dahlquist demonstrated that the order of a polynomial method cannot exceed 2 if the method is to be stable for any value of stepsize [63].

6.5. Regions of Stability

A system of different equations is characterized by a set of eigenvalues that may be complex. The stability of an integration method therefore is characterized by a region of stability in the complex $h\lambda$ plane. For test equation (6.15), the right-hand side of the $h\lambda$ plane denotes eigenvalues that imply an unstable solution, whereas the left-hand side of the $h\lambda$ plane denotes eigenvalues that imply a stable solution.[2]

For the Forward Euler method, (6.18) implies the stability constraint

---

[2] Many authors consider stability in the context of the test equation $\dot{x}=-\lambda x$. In this case, the right-hand side of the $h\lambda$ plane denotes a stable solution, whereas the left-hand side of the $h\lambda$ plane denotes an unstable solution. The regions of stability for these two cases are, of course, mirror images.

$$|1 + h\lambda| \leq 1 \tag{6.22}$$

Equation (6.22) describes the stability region that is illustrated in Fig. 6.1. The shaded region in this figure depicts the stable region of the Forward Euler algorithm.

The stable region for the Backward Euler algorithm is obtained from (6.20). This region is defined by

$$|1 - h\lambda| \geq 1 \tag{6.23}$$

The stability region for the Backward Euler method is given in Fig. 6.2. Again, the shaded region in this figure depicts the stable region of the algorithm.

Dahliquist proposed the definition of A-Stability [63]. If an integration algorithm is A-Stable, then the region of stability of the algorithm includes the entire left-hand side of the $h\lambda$ plane. In other words, an A-Stable method produces a stable solution whenever the exact solution is stable, regardless of the value of stepsize.

The requirement of A-stability excludes many integration methods from consideration. Gear [66,67] recognized that practical systems of differential equations can be solved with methods that are stiffy-stable even if the method is not A-stable. A numerical integration algorithm is stiffly-stable if the algorithm is stable as the timestep approaches infinity. The region of stability of a hypothetical stiffly-stable algorithm is illustrated in Fig. 6.3. This algorithm is not stable for all eigenvalues that have a negative real part; however, as h approaches infinity, the algorithm is stable. Clearly, any A-stable method also is stiffly-stable.

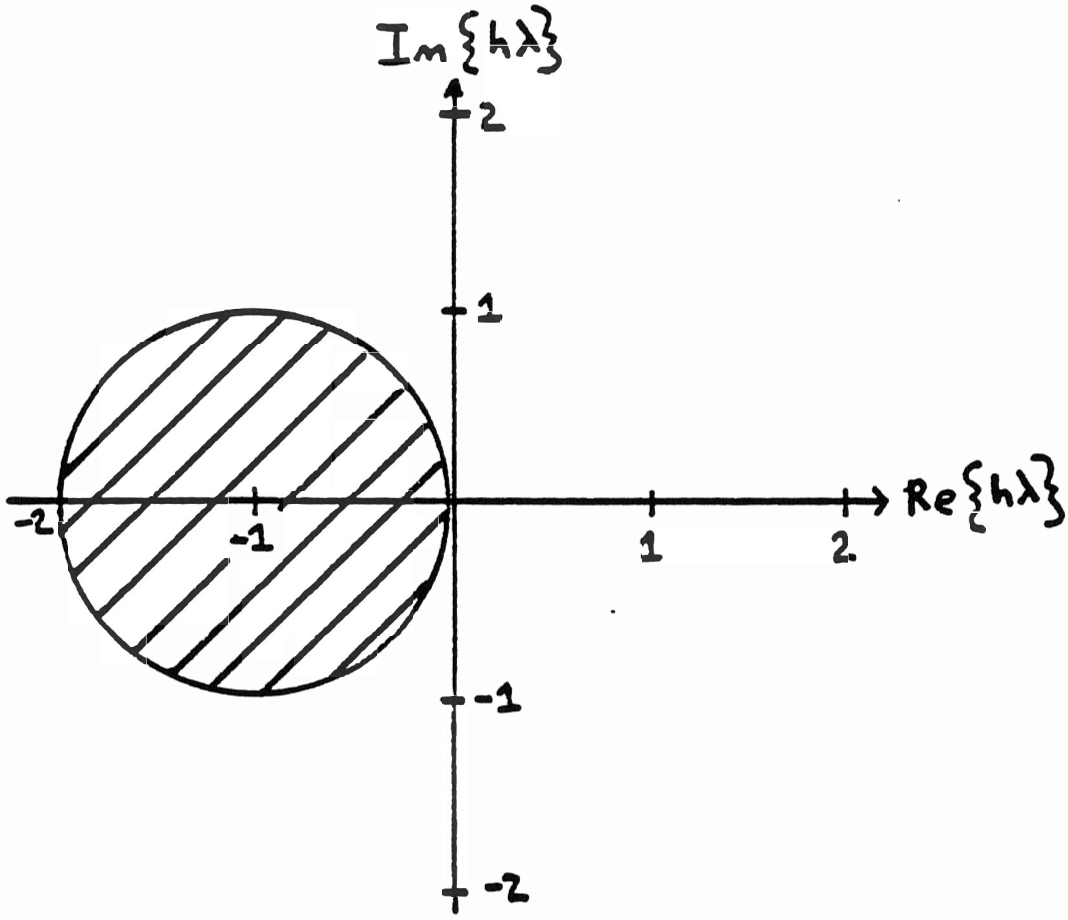


Fig. 6.1. The Region of Stability for the Forward Euler Algorithm.

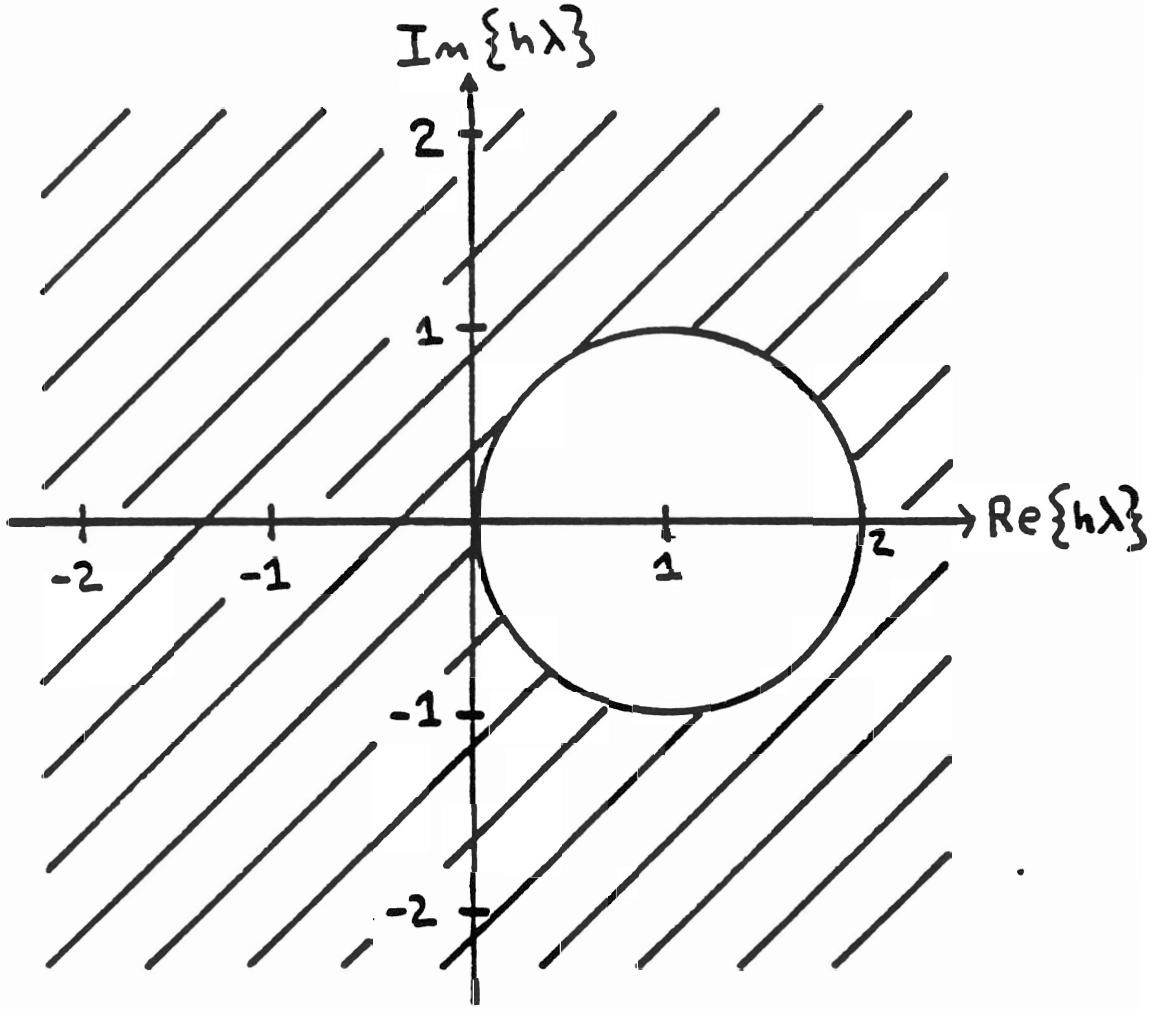


Fig. 6.2. The Region of Stability for the Backward Euler Algorithm.

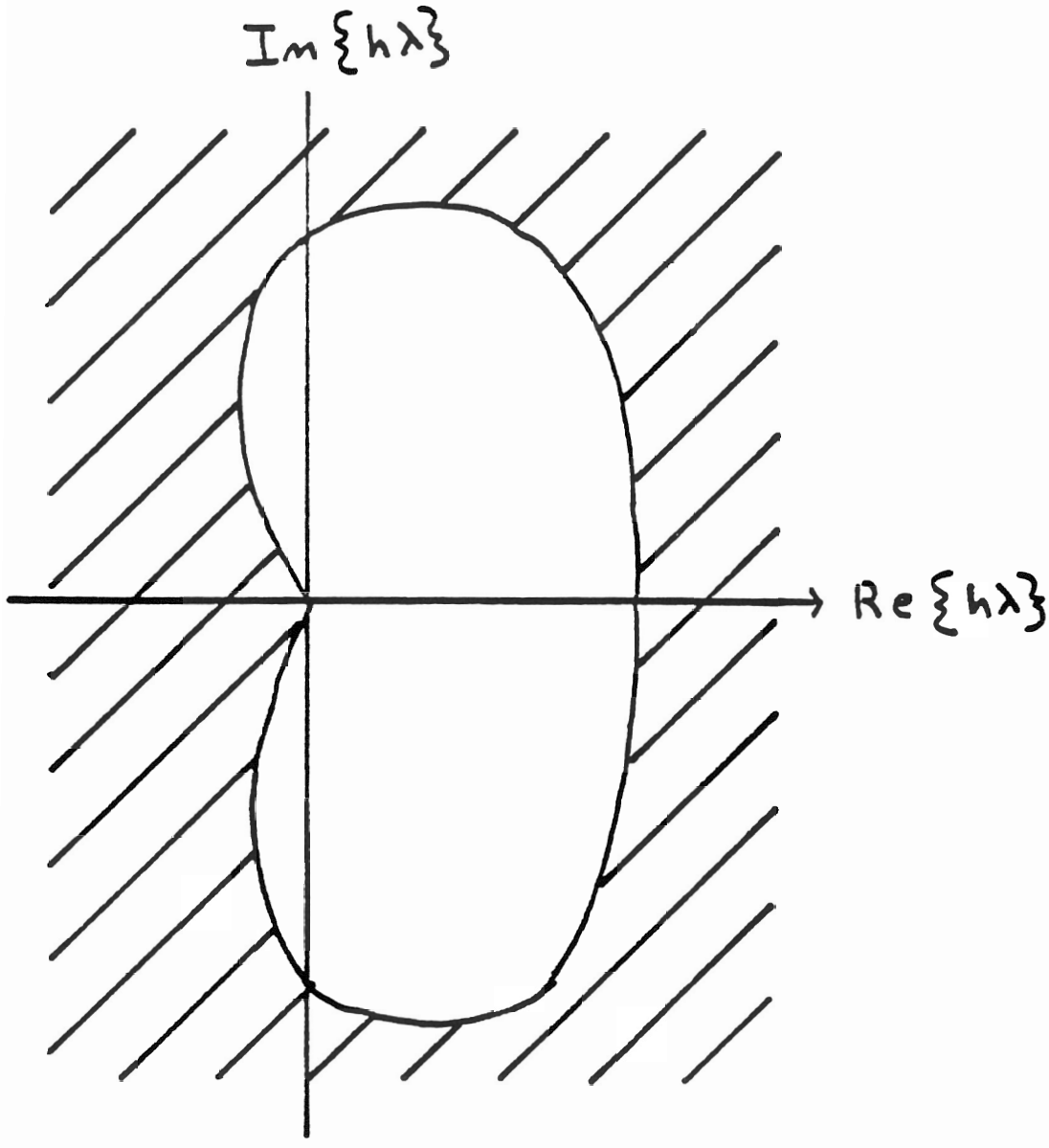


Fig. 6.3. The Region of Stability for a Hypothetical Stiffly-Stable Algorithm.

6.6. Stability and Stiff Systems of Equations

If the integration method is not stiffly-stable, then stability considerations impose an upper limit on timestep that is inversely proportional to the smallest eigenvalue of the circuit equations. This timestep limitation is a computational problem when the circuit equations are stiff, that is, when the ratio $|\lambda_{max}|/|\lambda_{min}|$ is several orders of magnitude, where λ_{max} is the eigenvalue of largest magnitude, and λ_{min} is the eigenvalue of smallest magnitude. As an example, a stiff system of circuit equations results from a circuit that contains small time-constants that are due to parasitic components and large time-constants that are due to coupling or bypass elements.

Stability considerations require that the timestep h be chosen such that $h\lambda_i$ for each eigenvalue is contained in the stable region of the integration algorithm. This constraint, in turn, usually requires that h be on the order of $1/|\lambda_{max}|$ if the integration method is not stiffly-stable. On the other hand, the time interval (0,T) over which the circuit equations are integrated usually is determined by the smallest eigenvalue. Specifically, T usually is on the order of $1/|\lambda_{min}|$. If the integration method is not stiffly-stable, then the number of timepoints that are computed in the transient analyses can be no less than $|\lambda_{max}|/|\lambda_{mun}|$ regardless of the error tolerance that is required.

If the circuit equations are not stiff, the ratio $|\lambda_{max}|/|\lambda_{min}|$ is 10 or 100, and the timestep limitation imposed by stability requirements is less severe than the timestep limitation that is required to obtain accurate answers. However, many electronic

circuits result in stiff circuit equations where the ratio $|\lambda_{max}|/|\lambda_{min}|$ may be 1000 or larger. If stiff equations are encountered, then stiffly-stable integration methods must be employed to insure that the timestep is limited by LTE considerations and not by stability requirements.

6.7. Local Truncation Error Criteria

Local truncation error can be measured either by the error in x or by the error in $\dot{x}$. If an implicit method is employed, both LTE measures are obtained easily. For example, the Backward Euler algorithm is stated either by (6.9) or, equivalently, by

$$\dot{x}_{n+1} \frac{x_{n+1}-x_n}{h_n} - \frac{h_n}{2} \frac{d^2x}{dt^2}(\xi) \tag{6.24}$$

Equation (6.9) provides a LTE estimate in terms of x_{n+1}. This error, ε_x, has the units of charge for a capacitor of flux for an inductor. If $E_x(t_{n+1})$ is the total error at timepoint t_{n+1}, then a stable integration algorithm obeys the inequality

$$|E_x(t_{n+1})| \leq \sum_{i=1}^{n+1} |\varepsilon_x(t_i)| \tag{6.25}$$

If E_T is the total absolute error, (6.25) will be satisfied if, at each timepoint,

$$|\varepsilon_x(t_{n+1})| \leq \frac{h_n}{T} E_T \tag{6.26}$$

If Backward Euler integration is employed, then (6.26) yields the timestep constraint

$$h_n \leq \frac{2E_T}{T\left|\frac{d^2x}{dt^2}(\xi)\right|} \tag{6.27}$$

The LTE estimate in (6.24) is expressed in terms of $\dot{x}_{n+1}$ and has the units of current for capacitors or voltage for inductors. If $\varepsilon_{\dot{x}}$ is the LTE estimate in terms of $\dot{x}_{n+1}$, and E_D is the absolute value of error that is allowed per timepoint, then the LTE constraint is stated by the equation

$$|\varepsilon_{\dot{x}}| \leq E_D \tag{6.28}$$

If Backward Euler integration is used, (6.28) yields the timestep constraint

$$h_n \leq \frac{2E_D}{\left|\frac{d^2x}{dt^2}(\xi)\right|} \tag{6.29}$$

A comparison of (6.27) and (6.29) leads to the conclusion that these timestep constraints are equivalent if $E_D = E_T/T$. For implicit polynomial methods, ε_x and $\varepsilon_{\dot{x}}$ are related by the equation

$$\varepsilon_{\dot{x}} = \beta_o \varepsilon_x \tag{6.30}$$

where β_o is defined in (6.14). For the Backward Euler method, $\beta_o = h_n$ so (6.27) and (6.29) are equivalent. For most polynomial methods, $\beta_o < h_n$, and (6.29) therefore requires a smaller timestep than (6.27) does if $E_D = E_T/T$.

Both the error tolerances E_D and E_T are absolute tolerances. However, relative tolerances are more meaningful to a program

user. Although a relative tolerance can be included in either (6.26) or (6.28), a relative tolerance is more relevant in terms of $\varepsilon_{\dot{x}}$. Adding a relative tolerance to (6.27) results in the timestep criteria

$$h_n \leq \frac{2[\varepsilon_r|\dot{x}_{n+1}|+\varepsilon_a]}{\left|\frac{d^2x}{dt^2}(\xi)\right|} \tag{6.31}$$

In (6.31), ε_r is the relative tolerance, and ε_a is the absolute tolerance.

6.8. Estimating Local Truncation Error

Any LTE timestep control requires an estimate of the LTE at each timepoint. If a kth order polynomial method is employed, this LTE estimate amounts to estimating the k+1 time-derivative of the solution vector x. The derivative estimate is obtained by fitting a k+1 order polynomial to the solutions $(x_{n+1},x_n,\ldots,x_{n-k})$.

The Backward Euler method uses a second-order polynomial of the form

$$y(t) = a_o + a_1(t_{n+1}-t_n) + a_2(t_{n+1}-t)^2 \tag{6.32}$$

The second-derivative of (6.32) clearly is equal to $2a_2$. Equation (6.32) is fitted to x_{n+1}, x_n, and x_{n-1} to yield

$$\begin{pmatrix} x_{n+1} \\ x_n \\ n_{n-1} \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 1 & h_n & h_n^2 \\ 1 & h_n+h_{n-1} & (h_n+h_{n-1})^2 \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \\ a_2 \end{pmatrix} \tag{6.33}$$

The solution of (6.33) for the derivative yields

$$\frac{d^2x}{at^2}(\varepsilon) \cong 2a_2 = 2 \quad \frac{\frac{x_{n+1}-x_n}{h_n} - \frac{x_n-x_{n-1}}{h_{n-1}}}{(h_n+h_{n-1})} \tag{6.34}$$

Equation (6.34) can be generalized to yield [49]

$$\frac{d^kx}{dt^k}(\varepsilon) \cong k!\ DD_k \tag{6.35}$$

where DD_k is the kth divided difference. The kth divided difference is defined by the recursive relation

$$DD_k = \frac{DD_{k-1}(t_{n+1})-DD_{k-1}(t_n)}{\sum_{i=1}^{k} h_{n+1-i}} \tag{6.36}$$

The first divided difference is defined by the equation

$$DD_1 = \frac{x_{n+1}-x_n}{h_n} \tag{6.37}$$

The LTE, and therefore the divided difference, is determined for each energy storage branch relation at each timepoint. For a kth order integration formula, k+1 values must be stored to estimate the LTE. A straightforward application of (6.36) requires the solutions $(x_{n+1},x_n,\ldots,x_{n-k})$ as well as the k timesteps $(h_n,h_{n-1},\ldots,h_{n+1-k})$.

If the backward timepoint solutions are stored in terms of the divided difference vector $(DD_1(t_n), DD_2(t_{n-1}),\ldots,DD_k(t_{n+1-k}))$, then (6.36) can be evaluated more efficiently. Alternatively, the backward information can be stored in terms of the Nordsieck vector [69]

or the Backward difference vector [70]. All of these methods require identical amounts of central memory storage (k words per energy-storage element). The computational effort that is required to estimate the LTE at each timepoint depends, of course, on the method that is used to store the back timepoint information. However, the computational work that is consumed in the LTE estimate usually is a small fraction of the effort that is expended in each timepoint solution.

6.10. The Trapezoidal Rule Algorithm

The implicit second-order Trapezoidal rule can be derived directly from Taylor series expansions in the same manner that was used for the derivation of the Backward Euler algorithm. Specifically x_{n+1} and $\dot{x}_{n+1}$ are expanded to yield

$$x_{n+1} = x_n + h_n\dot{x}_n + \frac{h_n^2}{2}\ddot{x}_n + \frac{h_n^3}{6}\frac{d^3x}{dt^3}(\xi) \tag{6.38}$$

$$\dot{x}_{n+1} = \dot{x}_n + h_n\ddot{x}_n + \frac{h_n^2}{2}\frac{d^3x}{dt^3}(\xi) \tag{6.39}$$

By solving (6.39) for $\ddot{x}_n$ and substituting the result into (6.38), the equation for the Trapezoidal rule is obtained:

$$x_{n+1} = x_n + \frac{h_n}{2}[\dot{x}_{n+1}+\dot{x}_n] - \frac{h_n^3}{12}\frac{d^3x}{dt^3}(\xi) \tag{6.40}$$

The derivative term in (6.40) is the LTE of the Trapezoidal method.

If (6.40), without the LTE estimate, is combined with (6.15), the following recursive relation is obtained:

$$x_{n+1} = x_n \left[\frac{1 + \frac{h_n \lambda}{2}}{1 - \frac{h_n \lambda}{2}} \right] \tag{6.41}$$

For the case of fixed stepsize, (6.41) is simplified to obtain

$$x_n = x_o \left[\frac{1 + \frac{h\lambda}{2}}{1 - \frac{h\lambda}{2}} \right]^n \tag{6.42}$$

Equation (6.42) approaches zero as n approaches infinity if the real part of λ is negative; hence, the Trapezoidal method is A-stable. The region of stability for the Trapezoidal method is shown in Fig. 6.4. In this figure, the shaded portion denotes the stable region of the algorithm. As shown in this figure, the Trapezoidal algorithm is stable if the exact solution is stable, and is unstable if the exact solution is unstable.

The manner in which (6.42) approaches zero as n approaches infinity is a peculiar feature of the Trapezoidal algorithm. If $|h\lambda| > 2$, then the numerator of (6.42) is negative. In this case, x_n will be positive for even values of n and will be negative for odd values of n. In other words, the solution will oscillate about the correct solution. This oscillation is harmless if the LTE is maintained at a satisfactory level.

6.10. The Gear Algorithms

The methods that were proposed by Gear [66,67] are defined by the equation

$$\dot{x}_{n+1} = \sum_{i=0}^{k} \alpha_i x_{n+1-i} \tag{6.43}$$

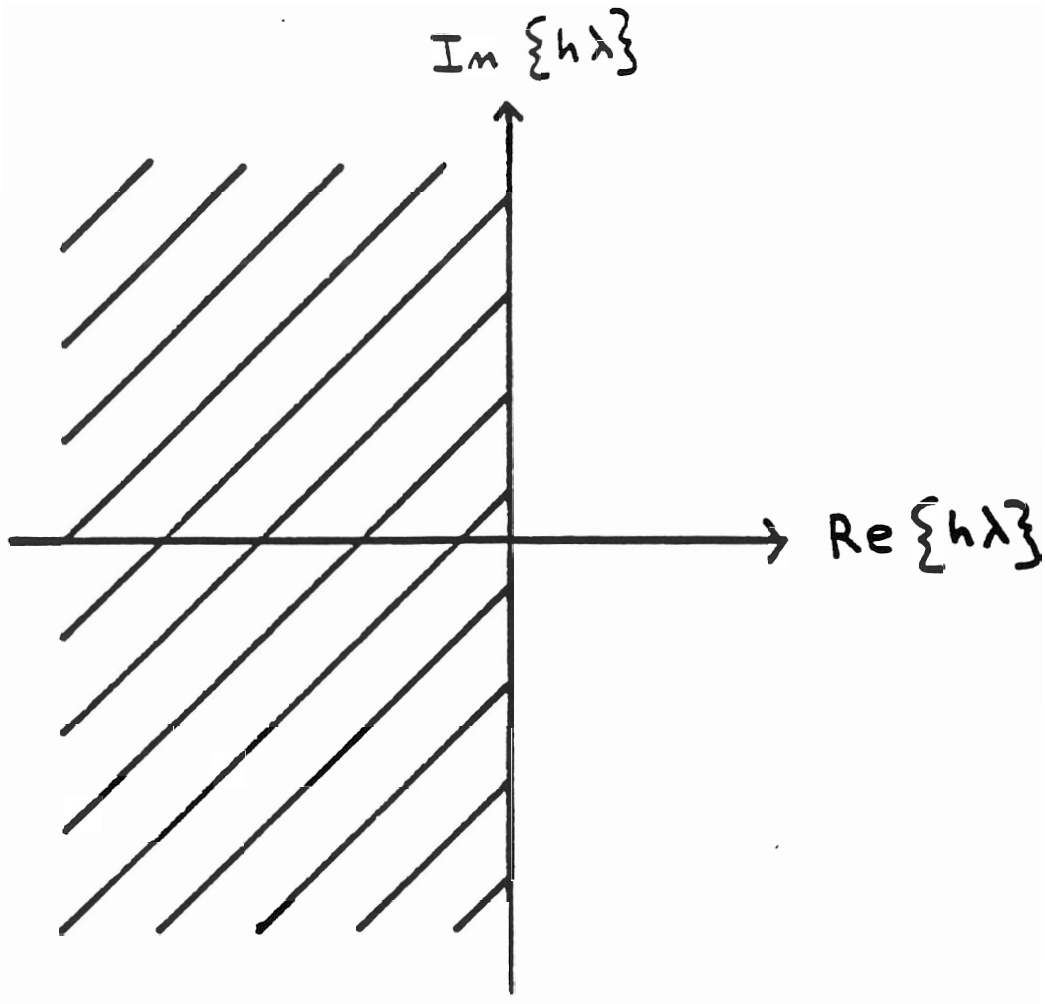


Fig. 6.4. The Region of Stability for the Trapezoidal Method.

Equation (6.43) contains k+1 coefficients that must determined interpolation coefficients. The coefficients are obtained using the method of undetermined coefficients. Let y be a kth order backward interpolation polynomial of the form

$$y(t) = a_0 + a_1(t_{n+1}-t) + \ldots + a_k(t_{n+1}-t)^k \tag{6.44}$$

The coefficients a_i are obtained by fitting (6.44) to the k+1 values $(x_{n+1},x_n,\ldots,x_{n+1-k})$ to produce the equation

$$\begin{bmatrix} x_{n+1} \\ x_n \\ \vdots \\ x_{n+1-k} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & h & h^2 & \ldots & h^k \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & kh & (kh)^2 & \ldots & (kh)^k \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_k \end{bmatrix} \tag{6.45}$$

Substituting (6.45) into (6.43) yields the equation

$$\begin{aligned} -a_1 = {} & \alpha_0[a_0+a_1h + \ldots + a_kh^k] \\ & + \alpha_1[a_0+a_1(2h) + \ldots + a_k(2h)^k] \\ & \vdots \\ & + \alpha_k[a_0+a_1(kh) + \ldots + a_k(kh)^k] \end{aligned} \tag{6.46}$$

The coefficients on the right-hand side of (6.46) independently must equal the coefficients of the left-hand side of (6.46). This constraint yields the system of equations

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ h & 2h & \cdots & kh \\ & & & \\ & & & \\ h^k & (2h)^k & \cdots & (kh)^k \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \cdot \\ \cdot \\ \cdot \\ \alpha_k \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ \cdot \\ \cdot \\ \cdot \\ 0 \end{bmatrix} \qquad (6.47)$$

The solution of (6.47) for values of k from 1 to 6 yields the six Gear methods that are given in Table 6.1. Gear's first-order method is the Backward Euler method.

The regions of stability for these six methods were derived by Gear [66] and are shown in Fig. 6.5. Only the first and second-order Gear methods are A-stable; however, all of Gear's methods are stiffly stable.

Gear's algorithms, for order greater than one, are multi-step methods. In contrast to the Backward Euler and Trapezoidal methods, Gear's methods yield the solution for x_{n+1} in terms of $\dot{x}_{n+1}$, x_n, and k-1 backward solutions $(x_{n-1},\ldots,x_{n-k+1})$. Any multi-step algorithm implies additional bookkeeping and storage since additional past timepoints must be processed and stored.

Another complication of multi-step methods is that they must be "started." Specifically, the second-order method cannot be used for the first timepoint since x_{-1} is not known. In general, a kth-order multi-step method cannot be used until k-1 timepoints have been computed. Gear proposed using the first-order method for the first timepoint, the second-order method for the second timepoint, and so on. After six timepoints have been computed, any of the Gear methods may be used.

Gear 1: $$x_{n+1} = x_n + h\,\dot{x}_{n+1} - \frac{h^2}{2}\frac{d^2x}{dt^2}\,(\xi)$$

Gear 2: $$x_{n+1} = \frac{4}{3}x_n - \frac{1}{3}x_{n-1} + \frac{2h}{3}\dot{x}_{n+1} - \frac{2h^3}{9}\frac{d^3x}{dt^3}\,(\xi)$$

Gear 3: $$x_{n+1} = \frac{18}{11}x_n - \frac{9}{11}x_{n-1} + \frac{2}{11}x_{n-2} + \frac{6h}{11}x_{n-3} - \frac{3h^4}{22}\frac{d^4x}{dt^4}\,(\xi)$$

Gear 4: $$x_{n+1} = \frac{48}{25}x_n - \frac{36}{25}x_{n-1} + \frac{16}{25}x_{n-2} - \frac{3}{25}x_{n-3} + \frac{12h}{25}\dot{x}_{n+1} - \frac{12h^5}{125}\frac{d^5x}{dt^5}\,(\xi)$$

Gear 5: $$x_{n+1} = \frac{300}{137}x_n - \frac{300}{137}x_{n-1} + \frac{200}{137}x_{n-2} - \frac{75}{137}x_{n-3} + \frac{12}{137}x_{n-4} + \frac{60h}{137}\dot{x}_{n+1} - \frac{10h^6}{137}\frac{d^6x}{dt^6}\,(\xi)$$

Gear 6: $$x_{n+1} = \frac{360}{147}x_n - \frac{450}{147}x_{n-1} + \frac{400}{147}x_{n-2} - \frac{225}{147}x_{n-3} + \frac{72}{147}x_{n-4} - \frac{10}{147}x_{n-5} + \frac{60h}{197}\dot{x}_{n+1} - \frac{180h^7}{3087}\frac{d^7x}{dt^7}\,(\xi)$$

Table 6.1. Gear's Methods for Fixed Stepsize.

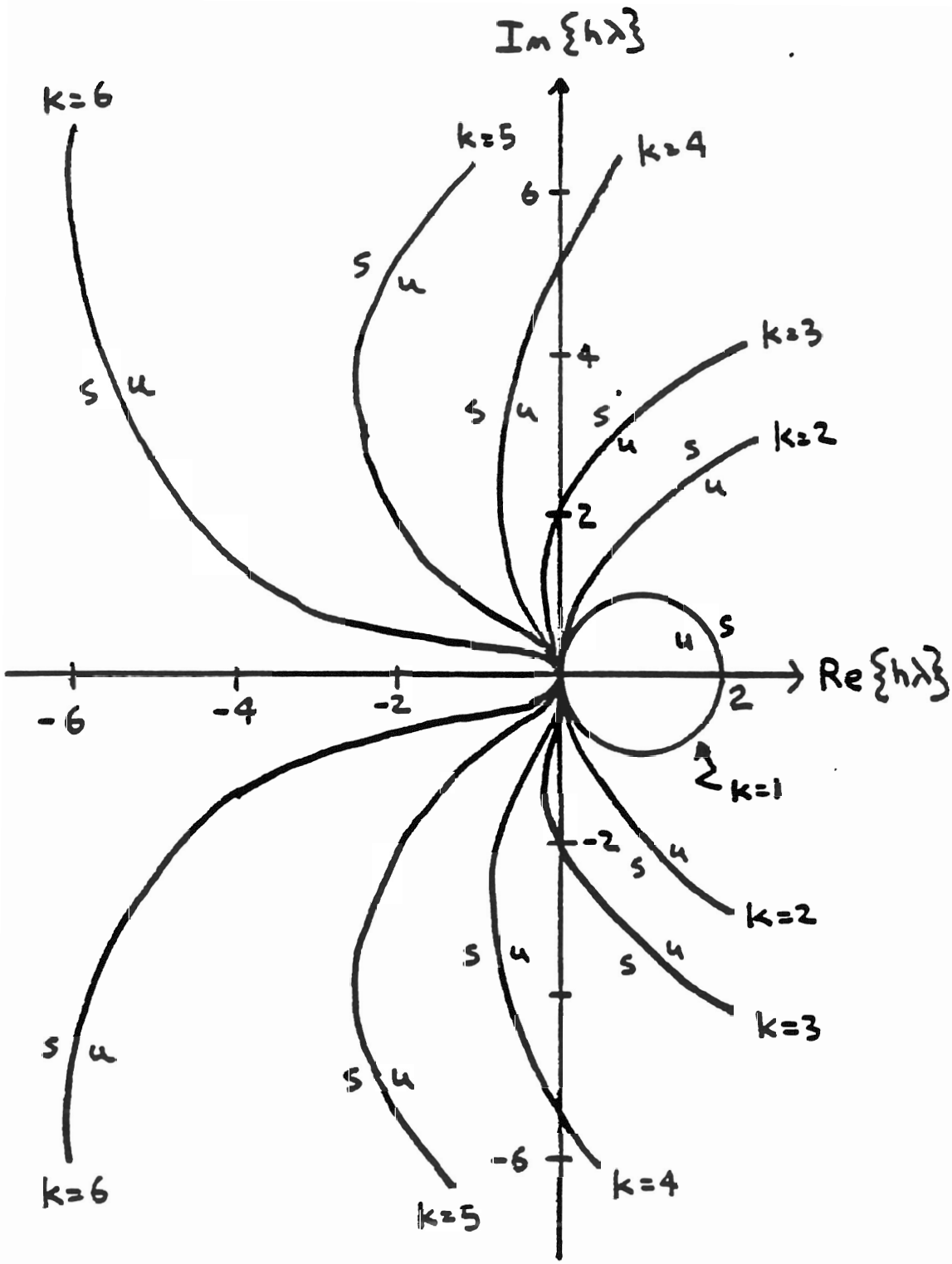


Fig. 6.5. The Regions of Stability for the Gear Methods.

The unstable regions of the third through sixth-order Gear methods may result in unstable solutions in some problems. A general-purpose integration algorithm that employs Gear's methods must include, therefore, a method of order control which insures that the higher-order methods are used only if they produce stable solutions.

6.11. The Gear-2 Method

Shichman [41] first reported the use of Gear's second-order algorithm in a circuit simulation program. The second-order Gear algorithm, for a fixed step size, is given in Table 6.1. Because this algorithm is a multi-step method, the results for fixed timestep cannot be used if the timestep is varied during the integration. Instead, the method must be modified for variable timestep [41]:

$$\dot{x}_{n+1} = \frac{2h_n+h_{n-1}}{h_n(h_n+h_{n-1})} x_{n+1} + \frac{h_n+h_{n-1}}{h_n h_{n-1}} x_n + \frac{h_n}{h_{n-1}(h_n+h_{n-1})} x_{n-1} \tag{6.48}$$

The regions of stability that are shown in Fig. 6.5 are for the fixed-timestep version of Gear's methods. The regions of stability for (6.48) have not been derived. It is possible, therefore, that (6.48) may not be stable for some combination of h_n and h_{n-1}. However, practical experience with CIRPAC [41,42] and SINC indicates that the Gear-2 method is stable for a large class of circuit simulation problems.

6.12. Runge-Kutta Methods

Classical Runge-Kutta methods [37,49] yield an explicit solution for x_{n+1} in terms of intermediate solutions. For example, the fourth-order Runge-Kutta method proceeds as follows

$$y_1 = f(x_n, t_n) \tag{6.49}$$

$$y_2 = f(x_n + \frac{h_n}{2} y_1,\ t_n + \frac{h_n}{2}) \tag{6.50}$$

$$y_3 = f(x_n + \frac{h_n}{2} y_2,\ t_n + \frac{h_n}{2}) \tag{6.51}$$

$$y_4 = f(x_n + h_n y_3,\ t_n + h_n) \tag{6.52}$$

$$x_{n+1} = x_n + \frac{1}{6} y_1 + \frac{1}{3} y_2 + \frac{1}{3} y_3 + \frac{1}{6} y_4 \tag{6.53}$$

The intermediate solutions (y_1, y_2, y_3, y_4) are used only for the final solution (6.53) and are not required in subsequent calculations.

Classical Runge-Kutta methods are not stiffly-stable, and these methods are very inefficient if stiff systems of equations are encountered. However, A-stable Runge-Kutta methods can be developed with an extension of the Runge-Kutta methods that was proposed by Rosenbrock [71]. Allen [72] employed Rosenbrock's extension to develop third-order and fourth-order A-stable Runge-Kutta methods. Allen's fourth-order extended Runge-Kutta method is employed in the CIRCUS2 program [34].

In common with all explicit algorithms, Runge-Kutta methods must be used in conjunction with State-Variable formulation, since the circuit equations must be cast in the form of (6.4). Moreover, explicit Runge-Kutta methods contain an additional drawback that is not encountered with other explicit methods. Specifically, the

LTE of Runge-Kutta methods is proportional to the Jacobian $\frac{\partial f}{\partial x}$. Extimating the LTE of Runge-Kutta methods is more difficult than with polynomial methods, and algorithms that dynamically vary the timestep with a Runge-Kutta method tend to use a smaller stepsize than is necessary for the required accuracy.

6.13. Implicit A-Stable Runge-Kutta Methods

An extension of classical Runge-Kutta methods by Miller [73,74] yields implicit A-stable methods of both second-order and third-order. Both of Miller's methods have considerably better error coefficients than are available with implicit polynomial methods. The second-order method requires one intermediate point as follows:

$$x^*_{n+\frac{1}{3}} = x_n + \frac{h}{3}\,\dot{x}^*_{n+\frac{1}{3}} \tag{6.54}$$

$$x_{n+1} = -\frac{5}{4}\,x_n + \frac{9}{4}\,x^*_{n+\frac{1}{3}} + \frac{h}{4}\,\dot{x}_{n+1} - \frac{h^3}{24}\,\frac{d^3x}{dt^3}\,(\xi) \tag{6.55}$$

The derivative term in (6.55) is the LTE of the second-order Miller algorithm. This truncation error is five times less than Gear's second-order method and twice as small as the Trapezoidal method. However, the need for the intermediate point implies that the computational effort per timepoint is twice as large for Miller's second-order method as it is for a polynomial method.

Miller's third-order implicit Runge-Kutta algorithm requires two intermediate points as follows:

$$x^*_{n+1} = x_n + h\,\dot{x}^*_{n+1} \tag{6.56}$$

$$x^*_{n+\frac{1}{3}} = \frac{13}{12}\,x_n - \frac{1}{12}\,x^*_{n+1} + \frac{5}{12}\,h\,\dot{x}^*_{n+\frac{1}{3}} \tag{6.57}$$

$$x_{n+1} = -\frac{19}{20} x_n + \frac{36}{20} x^*_{n+\frac{1}{3}} + \frac{3}{20} x^*_{n+1} + \frac{h}{4} \dot{x}_{n+1} - \frac{h^4}{22} \frac{d^4x}{dt^4}(\xi) \quad (6.58)$$

The derivative term in (6.58) is the LTE of the third-order Miller method. This LTE is three times less than for Gear's third-order method. However, the need for two intermediate points implies that the computational effort of Miller's third-order method is three times larger than the effort for a polynomial method.

6.14. A Comparison of the LTE of Implicit Methods

A brief survey of integration methods indicates that implicit algorithms are superior for the purpose of transient analysis. The need for a stiffly-stable integration method excludes most explicit algorithms with the notable exception of Allen's extended Runge-Kutta methods. The added difficulty in estimating the LTE of Allen's methods, as well as the necessity of employing State-Variable formulation, render Allen's explicit methods less attractive than the available implicit algorithms.

The choice of a particular integration method is determined by the LTE and the stability of the algorithm. The LTE terms for the Backward Euler method, the Trapezoidal Rule algorithm, the second through sixth-order Gear methods, and the second and third-order Miller methods are summarized in Table 6.2. In this table, the error ε_x is the LTE in terms of x, and $\varepsilon_{\dot{x}}$ is the LTE in terms of $\dot{x}$. All of the methods listed in Table 6.2 are stiffly-stable, and only the third through sixth-order Gear methods are not A-stable.

The most relevant figure of merit for different integration methods is the number of timepoints that is required to determine

	ε_x	$\varepsilon_{\dot{x}}$
BACKWARD EULER	$-\frac{h^2}{2}\frac{d^2x}{dt^2}(\xi)$	$\frac{h}{2}\frac{d^2x}{dt^2}(\xi)$
TRAPEZOIDAL	$-\frac{h^3}{12}\frac{d^3x}{dt^3}(\xi)$	$\frac{h^2}{6}\frac{d^3x}{dt^3}(\xi)$
GEAR-2	$-\frac{2h^3}{9}\frac{d^3x}{dt^3}(\xi)$	$\frac{h^2}{3}\frac{d^3x}{dt^3}(\xi)$
GEAR-3	$-\frac{3h^4}{22}\frac{d^4x}{dt^4}(\xi)$	$\frac{h^3}{4}\frac{d^4x}{dt^4}(\xi)$
GEAR-4	$-\frac{12h^5}{125}\frac{d^5x}{dt^5}(\xi)$	$\frac{h^4}{5}\frac{d^5x}{dt^5}(\xi)$
GEAR-5	$-\frac{10h^6}{137}\frac{d^6x}{dt^6}(\xi)$	$\frac{h^5}{6}\frac{d^6x}{dt^6}(\xi)$
GEAR-6	$-\frac{180h^7}{3087}\frac{d^7x}{dt^7}(\xi)$	$\frac{h^6}{7}\frac{d^7x}{dt^7}(\xi)$
MILLER-2	$-\frac{h^3}{24}\frac{d^3x}{dt^3}(\xi)$	$\frac{h^2}{6}\frac{d^3x}{dt^3}(\xi)$
MILLER-3	$-\frac{h^4}{22}\frac{d^4x}{dt^4}(\xi)$	$\frac{4h^3}{22}\frac{d^4x}{dt^4}(\xi)$

Table 6.2. Summary of the LTE Terms for Selected Implicit Integration Algorithms

the transient response with a given LTE tolerance. Unfortunately, this figure of merit must be determined empirically and, therefore, will depend upon the specific implementation of the integration method and the particular test problems that we used. However, some insight into the relative merits of the different integration methods can be gained from the integration of the simple test equation (6.15).

Since the exact solution of (6.15) is known, all of the derivatives in Table 6.2 can be evaluated exactly. Moreover, since the exact solution (6.16) and all derivatives of (6.16) are monotonically decreasing, evaluating these derivatives at timepoint t_n yields a consistant worst-case value for the LTE at timepoint t_{n+1}. Combining (6.16) and the two error terms for the Backward Euler estimate, for example, yields the two equations.

$$\frac{\varepsilon_x(t_{n+1})}{x_n} = \frac{(h_n\lambda)^2}{2} \tag{6.59}$$

$$\frac{\varepsilon_{\dot{x}}(t_{n+1})}{\dot{x}_n} = \frac{h_n\lambda}{2} \tag{6.60}$$

Equation (6.59) yields the worst-case relative error in x at any timepoint t_{n+1}, whereas the worst-case relative error in $\dot{x}$ at any timepoint t_{n+1} is determined by (6.60).

Integration algorithms can be compared on the basis of the value of $h_n\lambda$ that is required to obtain a specified relative accuracy at the timepoint t_{n+1}. The method that yields the largest value of $h_n\lambda$, for a given accuracy, should require the minimum number of timepoints in the transient analysis.

The values of $h_n\lambda$ for the nine implicit methods listed in

Table 6.2 are tabulated in Table 6.3 for relative accuracies of 0.1, 0.01, 0.001, 0.0001, and 0.00001. In this table, the LTE in $\dot{x}$ is used for comparison. The values of $h_n\lambda$ for the second and third-order Miller methods are divided by two and three, respectively, since the second-order Miller method requires twice the computational effort of the polynomial methods and Miller's third-order method requires three times the computational effort of the polynomial methods.

It is clear that the use of the higher-order Gear algorithms imply theoretical values of timestep that are much larger than the Trapezoidal method. The difference in timestep for the higher-order methods increases markedly as the error tolerance decreases. For example, the Gear-6 algorithm yields a timestep that is 22% larger than the timestep that is allowed by the Trapezoidal method for a relative error tolerance of 0.1. If the error tolerance is reduced to 0.001, the advantage of the Gear-6 method increases to a factor of 5.7, and if the error tolerance is reduced further to 0.00001, the advantage of the Gear-6 method increases to a factor of 26.2.

The two Runge-Kutta methods of Miller yield equivalent values of $h_n\lambda$ that are smaller than the Trapezoidal method in most cases. Only when the error tolerance is reduced to 0.00001 does Miller's third-order algorithm allow a value of timestep that is larger than the Trapezoidal method. Hence, the extra computational effort that is required to solve the intermediate points for these Runge-Kutta methods does not produce a commensurate increase in allowable timestep.

	Relative LTE in $\dot{x}$				
	0.1	0.01	0.001	0.0001	0.00001
BACKWARD EULER	0.20	0.02	0.002	0.0002	0.00002
TRAPEZOIDAL	0.77	0.24	0.077	0.0245	0.00775
GEAR-2	0.55	0.17	0.055	0.0173	0.00548
GEAR-3	0.74	0.34	0.159	0.0737	0.03420
GEAR-4	0.84	0.47	0.266	0.1495	0.08409
GEAR-5	0.90	0.57	0.359	0.2268	0.14310
GEAR-6	0.94	0.64	0.437	0.2980	0.20301
MILLER-2	0.39	0.12	0.039	0.0122	0.00387
MILLER-3	0.27	0.13	0.059	0.0273	0.01268

Table 6.3. Theoretical Values of $h_n \lambda$ for Selected Implicit Integration Methods.

6.15. Trapezoidal Integration Without Timestep Control

The Trapezoidal method is the most accurate A-stable polynomial method [63]. Moreover, this algorithm is simple and requires a minimal amount of past timepoint information. For these reasons, the Trapezoidal method is employed in CANCER [2] and SPICE1.

Neither CANCER nor SPICE1 contain a method of dynamically varying the integration timestep. Instead, the total time interval (0,T) can be divided into subintervals and different timesteps can be specified for each subinterval. This arrangement clearly leaves all matters of truncation-error control to the user, and more expertise on the part of the user is required to obtain accurate results. The widespread use and acceptance of the SPICE1 program indicates that the problem of selecting a proper timestep is not overwhelming. The circuit designer usually is familiar with the circuit and can estimate the dominant time-constants. Moreover, after a few preliminary analyses, the designer can adjust the timestep to yield satisfactory results.

As a starting point, the five transient test circuits and the ten standard benchmark circuits were simulated with a version of SPICE2 that did not contain a timestep control. The test circuits are detailed in Appendix 1 of this thesis. The number of timepoints, number of Newton iterations, and cpu execution time for these transient analyses are presented in the first column of Table 6.4. All of the analyses were run for 100 timepoints.

The important observation from the results without a timestep control is the nonconvergence of the SCHMITT and ASTABLE examples.

	SPICE2 No Timestep	SPICE2 Iteration Timestep	SPICE2 Iteration Timestep Prediction/Bypass
RC	100/200/0.372	109/218/0.418	109/218/0.435
CHOKE	-	110/385/1.599	110/395/1.650
ECL	100/237/3.091	110/264/3.448	109/247/3.168
SCHMITT	*	120/348/4.735	131/360/4.974
ASTABLE	*	217/957/7.896	198/874/7.235
DIFFPAIR	100/299/3.867	106/312/4.033	106/248/3.120
RCA3090	100/302/10.132	106/310/10.396	106/293/9.521
UA709	100/303/15.777	106/315/16.383	106/300/14.470
UA741	100/299/20.022	106/312/20.882	106/252/15.750
UA727	100/323/23.885	106/336/24.825	106/308/21.887
RTLINV	100/353/2.498	116/392/2.788	116/316/2.263
TTLINV	100/388/7.923	130/500/10.224	132/454/9.205
ECLGATE	100/363/10.465	121/422/12.132	121/361/10.191
MECLIII	100/345/13.860	116/396/15.911	116/338/12.957
SBDGATE	100/393/16.183	118/431/17.611	120/423/17.140

* No convergence at a timepoint after 100 iterations

Table 6.4. A Comparison of No Timestep Control, Iteration Timestep Control, and Iteration Timestep Control with Prediction and Bypass.

Both of these circuits have regenerative switching regions that result in extremely rapid transitions in the circuit waveforms. Unless the timestep is reduced drastically in these regenerative regions, the Newton sequence of iterations will not converge.

6.16. Iteration-Count Timestep Control With Trapezoidal Integration

The problems of nonconvergence during the transient analysis are avoided with the following improvement. If nonconvergence occurs, the timestep is reduced until the timepoint solution converged. If the number of iterations at a timepoint is less than a preset value, then the timestep is increased. However, the timestep never is allowed to exceed the user-supplied timestep. The user, therefore, still must select an appropriate maximum timestep to obtain accurate results.

The iteration timestep control produced best results when the timestep was reduced by a factor of 8 if the number of iterations exceeded 10 and the timestep was increased by a factor of 2 if the iteration count was less than five. Different criteria produce better results for some circuits, but the above implementation produces results that, on the average, require the fewest number of iterations.

The solution error in the transient response is larger in regions where there are rapid transitions in the solution. These transitions often are the result of input source transitions which can be predicted from the input data. To reduce the solution error in the region of source transitions, the timestep is reduced. The following method yields an appreciable improvement in solution accuracy. If the timestep for a particular timepoint implies a value

of t_{n+1} that exceeds a source breakpoint, then the timestep is reduced such that the value t_{n+1} coincides with the breakpoint. After the solution for the breakpoint has been determined, the timestep is reduced to 1/10 of the user-supplied timestep unless the timestep already is smaller than 1/10 of the user-supplied timestep by the iteration timestep algorithm.

The fifteen transient analyses were repeated with the SPICE2 program after the iteration timestep control and the source breakpoint recognition was implemented. The results for this series of tests are presented in the second column in Table 6.4. The first number is the number of timepoints, followed by the number of Newton iterations, followed by the execution time.

As the data shows, the problems of nonconvergence that are encountered in the analysis of the SCHMITT and ASTABLE circuits are eliminated when iteration timestep control is added. However, the number of timepoints and the number of iterations that is required by the iteration timestep control is larger than for the case of no timestep control. Specifically, the number of timepoints is 5% to 30% larger, and the number of Newton iterations is 3% to 30% larger. For most of these analyses, the increase in computational effort is due only to source breakpoint recognition. The timestep was reduced due to iteration count only in the analyses of the SCHMITT, ASTABLE, and TTLINV circuits.

To improve the computational efficiency of the transient analysis, two additional enhancements, prediction and bypass, were added to the SPICE2 program. Both of these additions are detailed in Chapter 5 of this thesis, and neither enhancement affects the

timestep control or the LTE. The prediction algorithm yields a better initial estimate for the Newton iteration sequence, and the bypass algorithm avoids recomputing the nonlinear functions that have not changed appreciably from the previous iteration.

The fifteen example circuits were tested again with the addition of these two improvements. The results for this final test are given in the third column in Table 6.4. With the exception of the RC and SCHMITT circuits, the addition of these two enhancements yields a 2% to 20% decrease in the number of Newton iterations and a 3% to 25% decrease in execution time.

6.17. Iteration-Count Timestep-Control with Gear-2 Integration

Since the Gear-2 algorithm has been used with considerable success, and the tests were repeated with the Gear-2 algorithm. The results for these two algorithms is given in Table 6.5. The results that are given in Table 6.5 include source breakpoint recognition, prediction, and bypass. The Gear-2 algorithm, as implemented, uses the Backward Euler method for startup. The startup procedure is repeated after a source breakpoint is reached or when nonconvergence is encountered.

A comparison of the data in Table 6.5 shows that there is little difference between the Trapezoidal method and the Gear-2 method if these algorithms are used with an iteration timestep control. For most of the circuits, the Gear-2 algorithm results in fewer iterations. At best, the Gear-2 algorithm requires 11% fewer iterations for the TTLINV circuit and the SBDGATE circuit. However, the Trapezoidal method requires less execution time for every circuit except the

	Trapezoidal	Gear–2
RC	109/218/0.435	109/218/0.469
CHOKE	110/395/1.650	110/300/1.397
ECL	109/247/3.168	109/247/3.532
SCHMITT	131/360/4.974	140/376/5.446
ASTABLE	198/874/7.235	236/867/7.732
DIFFPAIR	106/248/3.120	106/242/3.178
RCA3040	106/293/9.521	106/284/9.822
UA709	106/300/14.470	106/301/15.286
UA741	106/252/15.750	106/248/16.423
UA727	106/308/21.887	106/305/23.041
RTLINV	116/316/2.263	116/312/2.387
TTLINV	132/454/9.205	121/904/3.566
ECLGATE	121/361/10.191	121/360/10.788
MECLIII	116/338/12.957	116/336/13.603
SBDGATE	120/423/17.140	120/377/16.423

Table 6.5. A Comparison of the Trapezoidal and Gear-2 Methods with an Iteration Timestep Control.

TTLINV and the SBDGATE circuits. The disparity between the iteration count different and the execution time difference was traced to the inefficient method in which the Gear-2 algorithm was implemented. Specifically, the three coefficients in (6.43) were evaluated for each energy-storage branch at each iteration, instead of evaluating these coefficients once per timepoint. This additional overhead could be reduced, but not eliminated, with better programming.

6.18. A Comparison of Iteration-Count Timestep Control Methods

To this point, the iteration-count timestep-control implementations of Trapezoidal integration and Gear 2 integration have been compared solely on the basis of iteration count and cpu execution time. The numerical results for these two methods is, of course, as important to a circuit designer. This section will show that neither of these two methods yields results that are within reasonable error bounds for all of the circuits that were tested. To illustrate this accuracy problem, the numerical results for the CHOKE, ECL, and SCHMITT examples are presented.

The transient waveform of the voltage at node 3 of the CHOKE circuit is given in Fig. 6.6. The response in Fig. 6.6a was obtained with the LTE timestep control and Trapezoidal integration. The response in Fig. 6.6b was obtained with an iteration-count timestep control and Trapezoidal integration, and the response in Fig. 6.6c is the result with Gear-2 integration and an iteration-count timestep control. The time interval between 10 ms and 13 ms is of particular interest in this simulation. The response in Fig. 6.6b has a "ringing" in this time interval, whereas the response in Fig. 6.6c has a marked overshoot. For this example, neither iteration-count

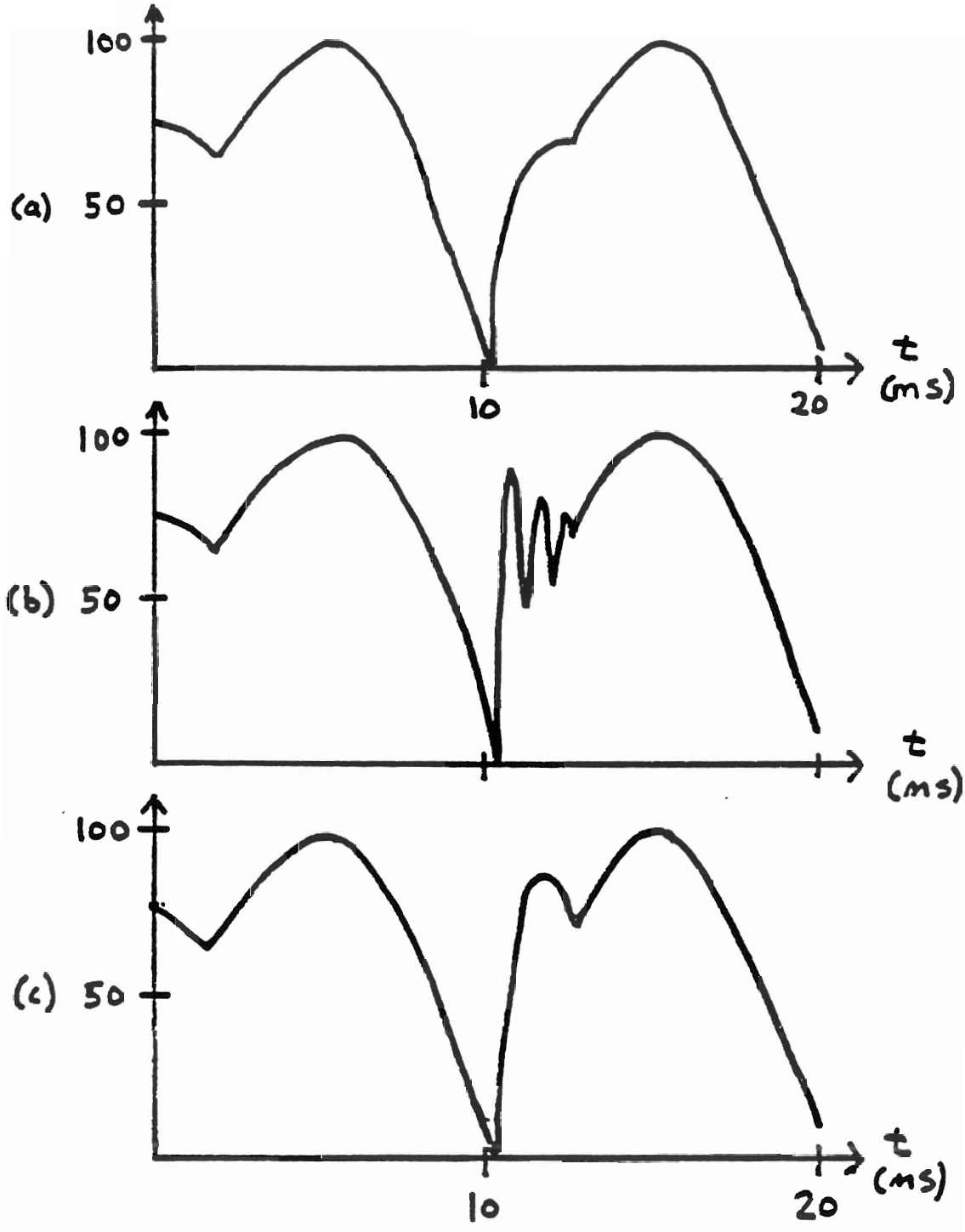


Fig. 6.6. Transient Response of Node Voltage 3 in the CHOKE Circuit.

algorithm yields results that are accurate to within 5%.

The next example is the response of the base current of Q1 of the ECL circuit. This response is illustrated in Fig. 6.7. Figure 6.7a shows the "correct" response that was computed with Trapezoidal integration and an LTE timestep control. Figures 6.7b and 6.7c show the responses that resulted from Trapezoidal integration and Gear-2 integration, respectively, and the iteration-count timestep control. For this example, Figs. 6.7a and 6.7b agree within 0.5%. However, Fig. 6.7c shows an overshoot in the vicinity of 2.5 ns that is not present in Fig. 6.7a. In other words, the iteration-count timestep control yields accurate answers for this example if Trapezoidal integration is employed, but the results with Gear-2 integration are not accurate.

Finally, the response of the voltage at node 5 in the SCHMITT circuit is considered. This response is illustrated in Fig. 6.8. Again, Fig. 6.8a is the response that results from the LTE timestep control, and Figs. 6.8b and 6.8c are the responses that are obtained with the iteration-count timestep control and Trapezoidal and Gear-2 integration, respectively. In this example, Fig. 6.8b contains a "ringing" in the response that is absent in the actual response. Figure 6.8c, on the other hand, does not contain the "ringing." For this example, the iteration-count timestep control produces satisfactory results if Gear-2 integration is employed but the results with Trapezoidal integration are inaccurate.

These three examples demonstrate the fact that neither the Gear-2 method nor the Trapezoidal method always yields acceptable results if the timestep is controlled on the basis of iteration count. Since

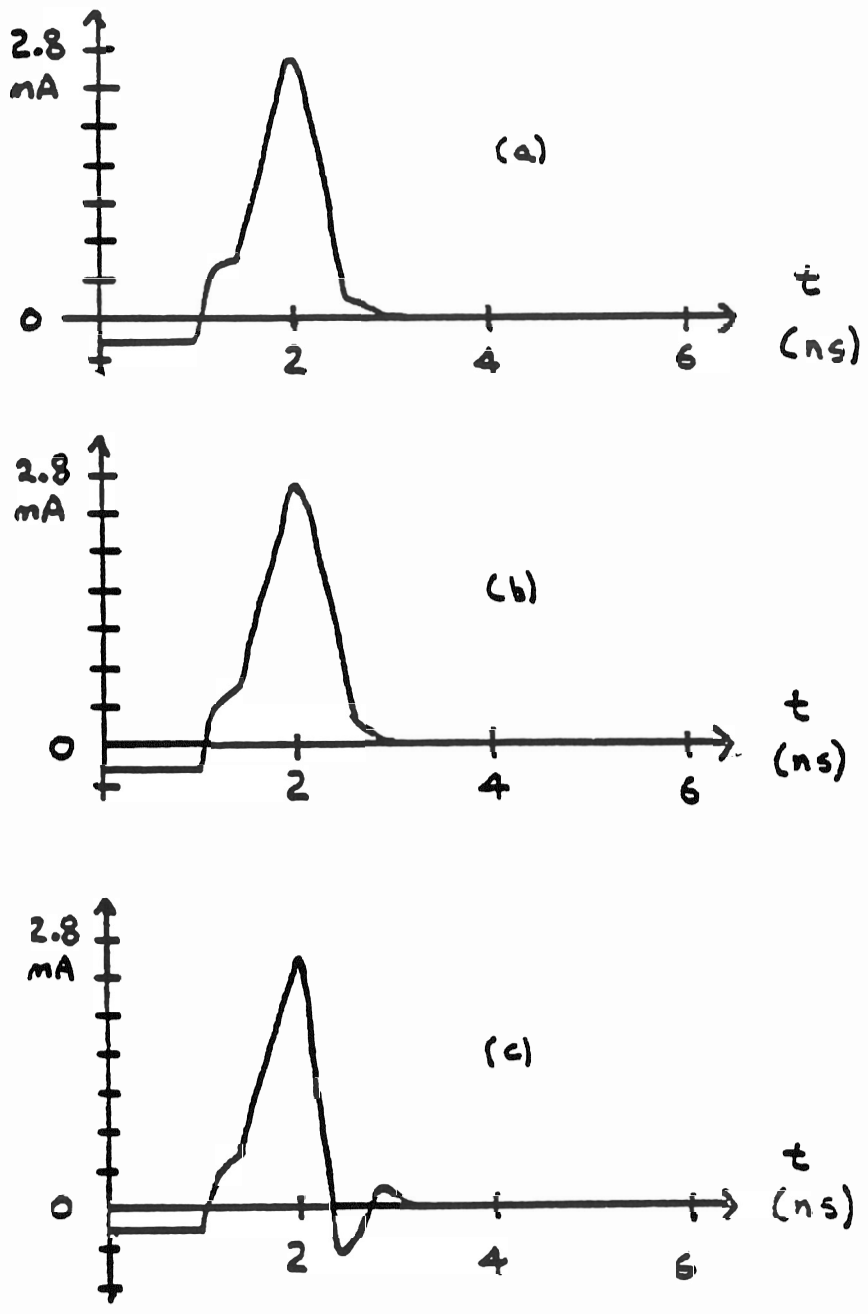


Fig. 6.7. Transient Response of Base Current in Q1 in the ECL Circuit.

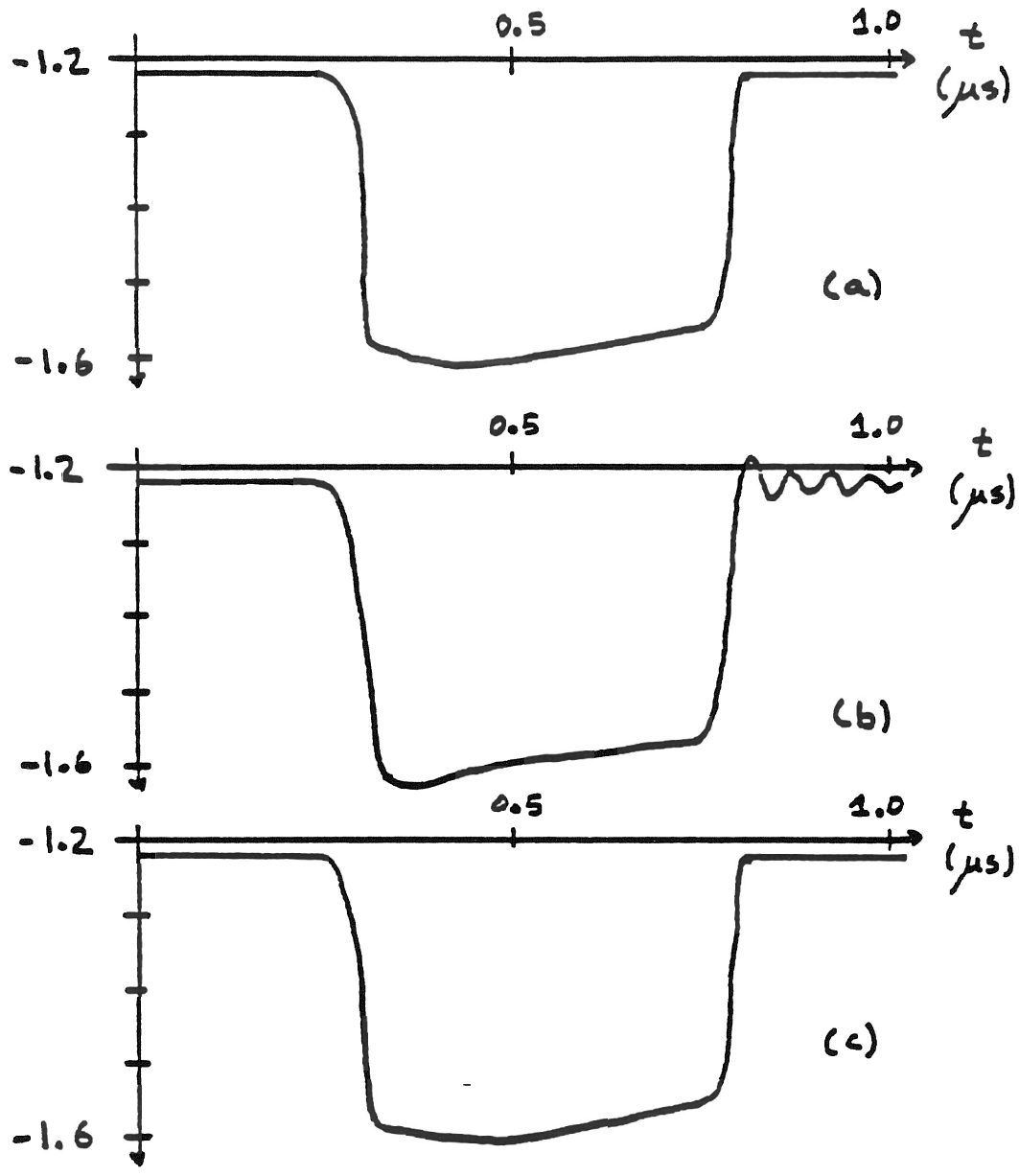


Fig. 6.8. Transient Response of Node Voltage 5 in the SCHMITT Circuit.

the iteration timestep control does not control truncation error, the error of a particular analysis depends upon the itegration method that is used as well as the timestep that is specified by the user.

6.19. Trapezoidal Integration With a LTE Timestep Control

The implementation of a truncation error timestep for the Trapezoidal methid is accomplished by adding a method of estimating the LTE at each timespoint and adjusting the timestep such that this LTE is maintained within reasonable bounds. This was accomplished in SPICE2 by using the relative derivative error estimate (6.31) together with the DD_3 method of estimating the third derivative to form the equation

$$h_{n+1} = \sqrt{\frac{\varepsilon_r |\dot{x}_{n+1}| + \varepsilon_a}{DD_3(t_{n+1})}} \qquad (6.61)$$

The timestep algorithm with Trapezoidal integration originally was implemented in SPICE2 as follows. After the timepoint t_{n+1} is determined, h_{n+1} is computed with (6.61). If $h_{n+1} \leq 0.9\ h_n$, then the truncation error for timepoint t_{n+1} is considered satisfactory, and the timestep h_{n+1} is used to compute the timepoint t_{n+2}. If, on the other hand, h_{n+1} 0.9 h_n, then the truncation error for this timepoint is too large, and the timepoint t_{n+1} is recomputed using $h_n = h_{n+1}$.

When this timestep strategy was applied to a calibration RC circuit, the error that was produced by the algorithm was much smaller than the error that was required. In other words, the algorithm was requiring too many timepoints and iterations to

determine a solution to the required amount of accuracy. This problem was traced to the inaccuracy of the polynomial error estimate, DD_3, when it is applied to an exponential solution.

To illustrate this inaccuracy, the test equation (6.15) is integrated using Trapezoidal integration with $\lambda = -3$ and $x_o = 1$. The results of this integration, with a timestep of 0.1, are given in Table 6.6. The exact value of LTE for this integration is given together with the DD_3 estimate. From this results, it is clear that the DD_3 estimate is about seven times larger than the exact value of LTE. To compensate for this discrepancy, an additional error control, ε_T, was introduced. The derivative estimate that is obtained by the divided differences is divided by ε_T before (6.61) is evaluated. Since a value of ten for ε_T yielded satisfactory results, it can be concluded that the divided difference formulae are about an order of magnitude smaller than the actual LTE for a simple test equation.

Although this timestep strategy worked well for linear circuits, the algorithm did not solve the nonlinear test circuits. In the nonlinear circuits, a timepoint was encountered where the timestep invariably was reduced until a preset minimum value was reached. This problem of timepoint "lockup" was traced to an iterative error in the timepoint solution.

The accuracy of each capacitor charge or inductor flux is limited by the convergence criteria ε_r and ε_a. For the case of a capacitor, Q_{n+1} and Q_n may differ by as much as the convergence criteria regardless of the value of h_n. Therefore, the accuracy of the capacitor current is limited by the iterate error

TIME	EXACT SOLUTION	TRAPEZOIDAL SOLUTION	LTE	$\frac{h^3 DD_3}{2}$
0.0	1.0000	1.0000		
0.1	0.9048	0.9048	0.76×10^{-4}	
0.2	0.8187	0.8186	0.68×10^{-4}	
0.3	0.7408	0.7406	0.62×10^{-4}	4.30×10^{-4}
0.4	0.6703	0.6701	0.56×10^{-4}	3.90×10^{-4}
0.5	0.6065	0.6063	0.51×10^{-4}	3.60×10^{-4}
0.6	0.5488	0.5486	0.46×10^{-4}	3.15×10^{-4}
0.7	0.4966	0.4463	0.41×10^{-4}	2.90×10^{-4}
0.8	0.4493	0.4490	0.37×10^{-4}	2.65×10^{-4}
0.9	0.4066	0.4063	0.34×10^{-4}	2.35×10^{-4}
1.0	0.3679	0.3676	0.31×10^{-4}	2.10×10^{-4}
1.1	0.3329	0.3326	0.28×10^{-4}	1.95×10^{-4}
1.2	0.3012	0.3009	0.25×10^{-4}	1.85×10^{-4}
1.3	0.2725	0.2722	0.23×10^{-4}	1.45×10^{-4}
1.4	0.2466	0.2643	0.21×10^{-4}	1.55×10^{-4}
1.5	0.2231	0.2229	0.19×10^{-4}	1.25×10^{-4}

Table 6.6. A Comparison of the Actual LTE and the DD_3 Estimate for Trapezoidal Integration.

$$\delta i = \frac{\varepsilon_r Q_{n+1}}{h_n} \tag{6.62}$$

Theoretically, the value of Q_{n+1} will converge to Q_n as the timestep is made small enough, but this theoretical property does not account for the error of the iterative process that is used to solve the timepoint. Unless this iterative error is accounted for in the timestep algorithm, then at a particular timepoint, the timestep can be reduced indefinitely. This occurance was observed in the transient analysis of the CHOKE, ECL, SCHMITT, and ASTABLE circuits.

Once recognized, this error can be accounted for by a trivial change in the algorithm. The revised error criterion is

$$|\varepsilon_{\dot{x}}| \leq \max\{\varepsilon_r|\dot{x}_{n+1}| + \varepsilon_a, \frac{\varepsilon_r|x_{n+1}|}{h_n}\} \tag{6.63}$$

In words, (6.63) requires that the LTE at timepoint t_{n+1} be less than the relative tolerance times the absolute value of $\dot{x}_{n+1}$ the absolute value of iterate error. Therefore, the timepoint never is required to be more accurate, from the standpoint of LTE, than is possible given the convergence tolerance.

The adoption of (6.63) yields the following strategy for timestep control. First, the allowable timepoint error, E_D, is computed using the equation

$$E_D = \max\{\varepsilon_r|\dot{x}_{n+1}| + \varepsilon_a, \frac{\varepsilon_r|x_{n+1}|}{h_n}\} \tag{6.64}$$

The timestep h_{n+1} is then computed with the equation

$$h_{n+1} = \sqrt{\frac{\varepsilon_T E_D}{|DD_3(t_{n+1})|}} \tag{6.65}$$

The use of (6.65) instead of (6.61) eliminated the problem of timepoint "lockup." However, (6.65) often yields a value of h_{n+1} that is too large; that is, if h_{n+1} is used to compute timepoint t_{n+2}, there is a good possibility that timepoint t_{n+2} will be rejected. Substantially better results were obtained if h_{n+1} was constrained to be less than 2 h_n. In other words, the timestep is allowed only to double at each timepoint.

The performance of this improved timestep control is illustrated by the results that are given in Table 6.7. This table gives the number of timepoints, the number of iterations, and the execution time that is required for the analysis of the five transient test circuits with four different values of error tolerance. The transient tolerance, ε_T, is equal to 10 for all cases.

As expected, the computational effort that is expanded in a transient analysis depends markedly on the error tolerance that is required. As an example, the ECL circuit requires 87, 172, 292, and 538 iterations, respectively, for an error tolerance of 0.1%, 0.01%, 0.001%, and 0.0001%. If a truncation-error timestep is used, the efficiency of the transient analysis program requires a reasonable choice of error tolerance. Empirically, each order of magnitude decrease in error tolerance results in almost a factor of two increase in computational effort.

A comparison between the iteration timestep and the truncation timestep, for Trapezoidal integration, is given in Table 6.8. This table presents the results for the analyses of the five transient examples and the ten standard benchmark circuits. The results for the iteration timestep are taken from Table 6.4 the results for

	0.01 10^{-11}	0.001 10^{-12}	0.0001 10^{-13}	0.00001 10^{-14}
RC	23 46 0.115	30 60 0.149	45 90 0.218	106 212 0.493
CHOKE	29 126 0.554	172 619 2.826	376 1033 4.942	769 1977 9.603
ECL	34 87 1.312	62 179 2.642	104 292 4.362	194 538 8.064
SCHMITT	89 294 4.421	169 441 6.707	313 798 12.208	537 1309 20.283
ASTABLE	203 883 7.843	312 1172 10.446	576 1832 20.025	1191 3387 30.722

Table 6.7. Results for the Trapezoidal Integration Truncation Error Timestep Control.

	TRAPEZOIDAL ITERATION TIMESTEP	TRAPEZOIDAL TRUNCATION TIMESTEP
RC	109/218/0.435	30/60/0.149
CHOKE	110/395/1.650	172/619/2.826
ECL	109/247/3.168	62/179/2.642
SCHMITT	131/360/4.974	169/441/6.707
ASTABLE	198/874/7.235	312/1172/10.446
DIFFPAIR	106/248/3.120	49/141/1.990
RCA3040	106/293/9.521	95/280/10.424
UA709	106/300/14.470	78/235/12.949
UA741	106/252/15.750	87/234/16.581
UA727	106/308/21.887	59/220/17.491
RTLINV	116/316/2.263	112/380/2.942
TTLINV	132/454/9.205	194/700/15.192
ECLGATE	121/361/10.191	207/632/21.429
MECLIII	116/338/12.957	149/471/19.670
SBDGATE	120/423/17.140	211/698/30.204

Table 6.8. Comparison of Iteration Timestep and Truncation-Error Timestep with Trapezoidal Integration.

the truncation error timestep control were obtained with $\varepsilon_T = 10$, $\varepsilon_r = 0.001$, and $\varepsilon_a = 10^{-12}$.

A truncation-error timestep control yields a substantial savings in computational effort for some circuits (such as the RC, ECL, and DIFFPAIR circuits) yet requires a substantial increase in computational effort for other circuits (such as the CHOKE, SCHMITT, ASTABLE, TTLINV, ECLGATE, and SBDGATE circuits). The remaining circuits required comparable amounts of computational effort for either algorithm.

The comparison of an iteration timestep control and a truncation error timestep control is rather futile effort. In some cases, truncation error requirements demand a smaller timestep than would be used with an iteration timestep control. In other cases, just the opposite is true. Moreover, the LTE for a particular energy-storage element may have little importance in the accuracy of a specified output variable. In this case, the specified outputs that are obtained with either timestep control will agree, within reasonable limits, even though the computational requirements for the two timestep controls may be vastly different. To wit, the computational results for the circuits that are given in Table 6.8, are essentially the same, although the data in Table 6.8 shows substantially different amounts of computational effort for the two methods.

6.20. Gear's Methods With a LTE Timestep Control

The Gear algorithms appear to be a superior choice when they are compared to the Trapezoidal method. Although only the first and second-order Gear methods are A-stable, all six methods

are stiffly-stable. It would be expected, then, that an integration method of order higher than 2 can be used for most transient analyses. The use of a higher-order method results in smaller values of local truncation error for a given timepoint; this smaller value of LTE, in turn, implies that a transient analysis can be performed with fewer timepoints.

Because the Gear algorithms, except the first-order method, are multi-step algorithms, the coefficients in (6.43) depend both on the current timestep, h_n, and the previous timesteps $h_{n-1},\ldots,h_{n+1-k}$. If the timestep varies from timepoint to timepoint, then (6.47) is restated for the case of variable timestep to yield

$$\begin{bmatrix} 1 & \cdots & 1 \\ h_n & h_n+h_{n-1} & \cdots & \sum_{i=1}^{k} h_{n+1-i} \\ \vdots & \vdots & & \vdots \\ h_n^k & (h_n+h_{n-1})^k & \cdots & \left(\sum_{i=1}^{k} h_{n+1-i}\right)^k \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \vdots \\ \alpha_k \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ \vdots \\ 0 \end{bmatrix} \qquad (6.66)$$

Equation (6.66) is solved to determine the coefficients α_i. This solution is repeated at each timepoint; however, the computational effort that is required to construct and solve (6.66) usually is a small fraction of the total computational effort that is required to solve each timepoint.

The timestep strategy for the Gear algorithms is basically the same strategy that is used for the Trapezoidal method.

Equation (6.63) is unchanged, and (6.65) is replaced by the equation in Table 6.9 that corresponds to the order of integration. The implementation of Gear's algorithms therefore requires three changes. First, (6.66) must be constructed and solved at the start of each timepoint. Second, the code that computes the equivalent algebraic brnach relations must be modified to use (6.43). Finally, the code that determines the timestep h_{n+1} is modified to use the equations in Table 6.9.

To verify the improved performance of the higher-order Gear methods, all six methods were tested on the five transient examples. The results for $\varepsilon_T = 10$, $\varepsilon_r = 0.001$, and $\varepsilon_a = 10^{-12}$ are given in Table 6.10, together with the results for the Trapezoidal method with the same error tolerances. For startup, the first timepoint is computed with the second-order method, and so on until the desired integration order is attained. The startup procedure is reinitiated whenever a source breakpoint is encountered or nonconvergence occurs.

For the relatively large error tolerances that were used to obtain the results in Table 6.10, none of the Gear methods produce computational results that are substantially better than the results for the Trapezoidal method. For the RC circuit, both the second and third-order methods yield an identical number of iterations that is 27% larger than the Trapezoidal results. The fourth-order method yields best results for the CHOKE circuit, and the iteration count for the Gear-4 method is 30% less than for the Trapezoidal method. The Gear-3 method produces the least number of iterations for the three remaining circuits, and the difference between the

(1) $$h_{n+1} = \frac{\varepsilon_T E_D}{|DD_2|}$$

(2) $$h_{n+1} = \left[\frac{\varepsilon_T E_D}{2|DD_3|}\right]^{1/2}$$

(3) $$h_{n+1} = \left[\frac{\varepsilon_T E_D}{6|DD_4|}\right]^{1/3}$$

(4) $$h_{n+1} = \left[\frac{\varepsilon_T E_D}{24|DD_5|}\right]^{1/4}$$

(5) $$h_{n+1} = \left[\frac{\varepsilon_T E_D}{120|DD_6|}\right]^{1/5}$$

(6) $$h_{n+1} = \left[\frac{\varepsilon_T E_D}{720|DD_7|}\right]^{1/6}$$

Table 6.9. The Predicted Timestep Equations for Gear's Six Methods.

	TRAP	GEAR 1	GEAR 2	GEAR 3	GEAR 4	GEAR 5	GEAR 6
RC	30 60 0.149	177 354 0.816	38 76 0.206	38 76 0.238	45 90 0.322	57 114 0.473	89 178 0.904
CHOKE	172 169 2.826	557 1294 6.453	218 642 3.288	168 590 3.212	128 432 2.556	137 460 2.913	212 626 4.564
ECL	62 179 2.642	221 488 7.407	69 189 3.001	59 165 2.864	67 186 3.435	109 265 5.548	220 484 11.594
SCHMITT	169 441 6.707	520 1149 17.750	204 503 8.233	178 453 8.217	198 495 9.551	273 642 13.566	> 1873 > 5000 >110.627
ASTABLE	312 1172 10.446	814 2142 19.538	395 1325 12.572	332 1174 12.098	353 1190 12.930	417 1292 18.730	813 2061 29.226

Table 6.10. Results for the Gear Methods without Variable Order with $\varepsilon_r = 0.001$ and $\varepsilon_a = 10^{-12}$

iteration count for the Gear-3 method and the Trapezoidal method is less than 10% for these three cases.

These tests were repeated with the error tolerances reduced by two orders of magnitude; specifically $\varepsilon_T = 10$, $\varepsilon_r = 0.00001$, and $\varepsilon_a = 10^{-14}$. The results for these tests are given in Table 6.11. With the reduced error tolerances, the results of the higher-order Gear methods are markedly better than the results for the Trapezoidal method. The third-order Gear method produced the minimal iteration count for the RC circuit and this iteration count is 34% smaller than the iteration count for the Trapezoidal method. The Gear-5 method produced the minimal iteration count for the CHOKE circuit and resulted in a 40% decrease in iterations when compared to the Trapezoidal method. The Gear-4 method yielded the minimal iteration count forthe remaining circuits and the iteration count was 27%, 18%, and 33% smaller, respectively, than the Trapezoidal method iteration count for the ECL, SCHMITT, and ASTABLE circuits.

The 18% to 40% decrease in iteration count for these examples is accompanied by at best a 13% decrease in execution time. For the SCHMITT circuit, the execution time for the Gear-4 method actually is 3% larger than the execution time for the Trapezoidal method, despite an 18% decrease in iteration count. The reason for the disparity between the execution time and the iteration count is due to the additional overhead of the Gear methods. For example, the ECL circuit requires 15.0 ms of execution time per iteration with the Trapezoidal method. By comparison, the execution time, per iteration, for the Gear methods is 16.0 ms, 17.6 ms, 18.6 ms, 20.0 ms, and 22.5 ms, respectively, for the

	TRAP	GEAR 1	GEAR 2	GEAR 3	GEAR 4	GEAR 5	GEAR 6
RC	106 212 0.493	—	91 188 0.508	70 140 0.457	75 150 0.566	110 220 0.961	237 474 2.534
CHOKE	769 1977 9.603	—	952 2383 12.796	584 1559 9.380	459 1270 8.230	409 1189 8.333	600 1588 12.834
ECL	194 538 8.064	—	238 632 10.106	159 460 8.082	136 393 7.305	161 463 9.246	318 811 18.248
SCHMITT	537 1309 20.283	—	635 1500 25.128	434 1119 20.488	412 1068 20.910	540 1324 28.392	2282 5000 121.421
ASTABLE	1191 3387 30.722	—	1522 4121 40.582	436 2705 29.687	752 2260 26.421	759 2286 28.652	1391 3670 52.208

Table 6.11. Results for the Gear Methods Without Variable Order With $\epsilon_r = 0.00001$ and $\epsilon_a = 10^{-14}$

Gear-2, Gear-3, Gear-4, Gear-5, and Gear-6 methods. The 6% difference between the Gear-2 method and the Trapezoidal method is due to the additional overhead of constructing the coefficients α_i. The 10% to 40% difference between the Gear algorithms is due principally to the inefficient method in which the divided differences were formed in these tests. More efficient programming of this subroutine would decrease, but not eliminate, the added overhead for the higher-order Gear methods.

Neither the fifth-order nor the sixth-order Gear algorithms yield results that are consistently better, in terms of iteration count, than the third-order and fourth-order methods. The Gear-6 method does not produce optimal results for any of the cases that were tested, and the Gear-5 method produced minimal iteration count only for the case of the CHOKE circuit at the smaller error tolerance.

This increase in iteration count for the fifth and sixth-order methods is not due solely to the lack of A-stability of the methods. The RC circuit is characterized by two negative real eigenvalues, and all six methods should be stable for this test circuit. However, the Gear-4, Gear-5, and Gear-6 methods require 18%, 50% and 134% more iterations, respectively, than the Gear-3 method for the analysis of the RC circuit with the larger error tolerance; for the smaller error tolerance, these methods required 7%, 57%, and 238% more iterations than did the Gear-3 method.

It is conjectured that the failure of the higher-order algorithms to produce substantially smaller iteration counts is due to the inaccuracies that are inherent in the truncation

error estimation. The estimation of derivatives is a notably inaccurate process [44]. As the order increases, higher-order derivatives must be estimated, and the error in the estimate of the LTE can be expected to increase. Apparently, this increase in the error of the LTE estimate overshadows the theoretical decrease in the actual LTE. Consequently, the best order involves a tradeoff between a theoretically lower value of LTE and an inherent increase in the inaccuracy of the LTE estimate. The results given here indicate that the Gear methods of order higher than four do not produce commensurate decreases in iteration count and execution time.

6.21. Gear's Methods With Variable Order

The results that are given in Tables 6.10-6.11 indicate that the optimal order of integration, as measured by iteration count, will be different for different simulations. Hence, the implementation of Gear's algorithms requires a method of varying the order of integration, as well as the timestep, during the integration.

The order strategy that is developed here was proposed by Gear [66,67] and modified by Brayton, et al. [70]. If the present order is k, then after k timepoints have been computed, h_{n+1} is computed using the equation for k-1 order, k order, and k+1 order. If k is one, the k-1 order computation is skipped, and if k is at the maximum order, the k+1 computation is skipped. The order that yields the largest value of h_{n+1} is then selected. However, the order is changed only if the optimal h_{n+1} exceeds the present

h_{n+1} by a factor of R_o [75]; a typical value of R_o is 1.05 to 1.5. As an example, if the integration is proceeding at third order, then after three timepoints, h_{n+1} is computed for second order, third order and fourth order. If the timestep that is determined with the second-order formula is at least R_o times larger than the timestep that is computed with the third-order formula, then the order is decreased to two. On the other hand, if the timestep that is computed with the fourth-order formula is at least R_o times larger than the third-order value, then the order is increased to four. Otherwise, the order is unchanged.

Experimentation showed that fewer iterations were required if the order test was performed every timepoint, instead of every kth timepoint. The results for this variable-order algorithm for the five transient examples is given in Table 6.12; MAXORD is the maximum order that the algorithm is allowed to use. The data in Table 6.12 corresponds to the error tolerance $\varepsilon_T = 10$, $\varepsilon_r = 0.001$, and $\varepsilon_a = 10^{-12}$, and $R_o = 1.3$.

The most interesting observation from Table 6.12 is that, no matter what the maximum integration order is, the algorithm usually stays at second order. Only in the case of the ASTABLE circuit was the third-order method used, and when the third-order method was used, it resulted in 12% more iterations.

In an attempt to isolate why the higher-order methods were not being used, the variable-order test was repeated for the same error tolerance but with $R_o = 1.05$. The results for this test are given in Table 6.13. For this test, the order stays at two for the RC and CHOKE circuits and stays at three for the

	TRAP	MAXORD 2	MAXORD 3	MAXORD 4	MAXORD 5	MAXORD 6
RC	30 60 0.149	47 94 0.280	47 94 0.319	47 94 0.319	47 94 0.319	47 94 0.317
CHOKE	172 619 2.826	394 966 5.372	394 966 5.573	394 966 5.582	394 966 5.563	394 966 5.544
ECL	62 179 2.692	73 201 3.419	73 202 3.918	73 202 4.019	73 202 4.007	73 202 4.000
SCHMITT	169 441 6.707	342 900 15.384	392 900 15.873	392 900 15.804	392 900 15.794	392 900 15.795
ASTABLE	312 1172 10.446	584 1690 17.052	674 1907 19.897	674 1907 19.902	674 1907 19.885	674 1907 19.917

Table 6.12. Results for Variable-Order Method With $R_o = 1.3$, $\epsilon_r = 0.001$. and $\epsilon_a = 10^{-12}$.

ECL, SCHMITT, and ASTABLE circuits. For this change in R_o, the RC circuit requires 11% more iterations, the CHOKE circuit requires 9% fewer iterations, the ECL circuit requires the same number of iterations, the SCHMITT circuit requires 31% fewer iterations, and the ASTABLE circuit requires 15% fewer iterations. For the most part, the smaller value of R_o results in a more efficient simulation.

These tests were then repeated with a smaller value of error tolerance; specifically, $\varepsilon_T = 10$, $\varepsilon_r = 0.00001$, $\varepsilon_a = 10^{-14}$, and $R_o = 1.05$. The results of this test are shown in Table 6.14. With this error tolerance, the results for the variable-order Gear algorithm are better than the results that were obtained with the Trapezoidal method in some cases. For the RC circuit, identical results were obtained for maximum order of three or higher, the iteration count with MAXORD $\geq$ 3 is 13% less than the iteration count that was obtained with the Trapezoidal method. A maximum order of five produced best results for the CHOKE circuit; the iteration count for MAXORD = 5 is 21% less than the iteration count for the Trapezoidal method. For the ECL circuit, a maximum order of 5 or 6 produced an iteration count that is 11% less than the Trapezoidal method. However, a maximum order of two is best for the SCHMITT circuit, and the iteration count for MAXORD = 2 is 15% larger than the iteration count for the Trapezoidal method. If MAXORD is greater than two, the analysis of the SCHMITT circuit requires twice the iteration count as for the case when MAXORD is 2. Finally, the analysis of the ASTABLE circuit required more than 5000 iterations no matter what value of MAXORD was used. The

	TRAP	MAXORD 2	MAXORD 3	MAXORD 4	MAXORD 5	MAXORD 6
RC	30 60 0.149	52 104 0.306	52 104 0.337	52 104 0.342	52 104 0.336	52 104 0.337
CHOKE	172 619 2.826	359 880 4.869	359 880 5.094	359 880 5.079	359 880 5.077	359 880 5.092
ECL	62 179 2.692	74 203 3.454	74 201 3.807	74 201 3.930	74 201 3.945	74 201 3.931
SCHMITT	169 441 6.707	301 678 11.837	258 623 11.514	258 623 11.513	258 623 11.519	258 623 11.525
ASTABLE	312 1172 10.946	577 1725 17.307	692 1931 20.072	692 1931 20.083	692 1931 20.122	692 1931 20.124

Table 6.13. Results for Variable-Order Method with $R_o = 1.05$, $\varepsilon_r = 0.001$, and $\varepsilon_a = 10^{-12}$.

	TRAP	MAXORD 2	MAXORD 3	MAXORD 4	MAXORD 5	MAXORD 6
RC	106 212 0.493	94 188 0.584	92 184 0.702	92 184 0.748	92 184 0.745	92 184 0.742
CHOKE	769 1977 9.603	970 2421 13.908	660 1701 11.218	615 1605 11.624	591 1565 11.842	603 1610 12.703
ECL	194 538 8.064	1599 3399 57.758	181 517 10.172	169 484 10.397	167 480 10.746	167 480 10.779
SCHMITT	537 1309 20.283	640 1509 27.373	1539 3401 62.306	1539 3401 63.240	1539 3401 63.167	1539 3401 63.221
ASTABLE	1191 3387 30.722	> 2121 > 5000 >51.871	> 2136 > 5000 >54.163	> 2167 > 5000 >55.069	> 2167 > 5000 >55.540	> 2167 > 5000 >55.557

Table 6.14. Results for the Gear Variable-Order Method with $R_o = 1.05$, $\epsilon_r = 0.00001$, and $\epsilon_r = 10^{-14}$

Trapezoidal method, on the other hand, required 3387 iterations to complete the transient analysis of the ASTABLE circuit.

The mechanics of order variation considerably diminish the advantage of using the higher-order algorithms. For nominal error tolerances, the fixed-order Gear-2 method and the Trapezoidal method both require less computational effort than the variable-order Gear method that is presented in this chapter. When more stringent accuracy requirements are imposed, the variable-order Gear algorithm yields a 10% to 20% decrease in iteration count for some circuits when compared to the Trapezoidal method.

6.22. Conclusions

Numerical integration methods consist of two types: explicit algorithms and implicit algorithms. Since many electronic circuits result in a stiff system of circuit equations, the numerical integration technique must be stiffly-stable. The need for a stiffly-stable method excludes most explicit methods. Moreover, the advantages of implicit methods, as compared to explicit methods, yield the conclusion that implicit integration methods are superior for the task of transient analysis.

Of the implicit methods that are available, the Trapezoidal method and Gear's methods are the most attractive. Both of these algorithms are polynomial methods; the competitive Runge-Kutta methods require more computational effort than polynomial methods.

An accurate and efficient transient analysis program must contain some method of dynamically varying the timestep. The

easiest method of timestep control is an iteration timestep control that determines the timestep solely upon the number of iterations that are required to solve each timepoint. However, several examples of nonlinear circuits were presented for which an iteration timestep failed to produce acceptable computational results.

To insure an accurate transient analysis, the timestep must be controlled to produce an acceptable amount of LTE at each timepoint. The implementation of an LTE timestep control with Trapezoidal integration was described in detail, and results for typical circuits were presented.

Both the Gear algorithms contain the theoretical advantage of less LTE for a given timestep, as compared to the Trapezoidal method, these algorithms also were implemented and tested. The theoretical advantage of a higher-order method is overshadowed by the twin problems of computational inaccuracy of the LTE estimate and the computational overhead involved in varying the integration order. Even for the smallest error tolerance that was considered, the variable-order Gear algorithm failed to produce substantial computational improvements when compared to the Trapezoidal method.

Trapezoidal integration, with a LTE timestep control, yielded computational results that were as accurage as any of the methods tested. Moreover, the use of Trapezoidal integration invariably resulted in less computational efforts for these reasons, the SPICE2 program employs Trapezoidal integrating with the LTE timestep control that is detailed in this chapter.

VII. CONCLUSIONS

Few general numerical methods can be enlisted for a particular application without some modification. The algorithms that are employed in the SPICE programs and are documented in this thesis are the result of careful implementation, modification, and comparison of existing numerical methods in the special context of circuit simulation. This thesis has attempted, as best as possible, to compare separately the major numerical techniques that are utilized in circuit simulation programs. The object of these comparisons is, of course, the combination of the "best" methods into a program that simulates electronic circuits accurately and efficiently with minimal interaction on the part of the program user.

In Chapter 2 of this thesis, the task of circuit simulation is separated into the algorithmic methods of equation formulation, linear equation solution, nonlinear equation solution, and numerical integration. Fortunately, these four areas are relatively independent of one another in the sense that the choice of a particular implementation in one area has little bearing on the choice of algorithms in the remaining areas. This independence enables the separate comparisons of methods within a given topic that are presented in Chapters 3, 4, 5, and 6. The important results from each of these chapters is summarized here.

The many equation formulation methods that are available can be grouped into the three classes of Nodal Analysis, Hybrid Analysis, and Sparse Tableau Analysis. Nodal Analysis methods were found to have virtually the same generality as the other methods, providing

modifications are incorporated to accommodate voltage-defined branches. For the sample circuits that were tested, the Modified Nodal Analysis method produced a system of circuit equations that requires substantially less computational effort to solve than does the Hybrid Analysis methods. Moreover, Nodal methods are considerably easier to implement than either Hybrid methods or Sparse Tableau Analysis.

Modified Nodal Analysis (MNA) is implemented in the SPICE2 program. This method provides the same generality that other formulation methods offer, yet it produces a near-symmetric system of equaitons that are solved with an amount of computational effort that is comparable to the simplest Nodal methods. However, two problems were encountered in the implementation of MNA in SPICE2. First, numerical ill-conditioning problems arose in certain cases. Second, the manner in which grounded voltage sources are handled with MNA is not optimal. A modification of MNA, namely MNA2, removed these problems but resulted in an unsymmetric coefficient matrix, and a commensurate increase in the overhead of the sparse-matrix techniques. Therefore, the original MNA method is retained in SPICE2. Further research should yield a modified Nodal Analysis method that eliminates the aforementioned problems without destroying the symmetry of the coefficient matrix.

Implementing the linear equation solution method proved to be reasonably straightforward. Since SPICE is implemented on the CDC 6400 which has a large word size, roundoff errors in the linear equation solution usually are negligible and the solution method can be optimized in terms of computational effort instead of accuracy.

The direct elimination method of LU factorization, therefore, is the logical choice of linear solution methods. The sparse-matrix implementation of LU factorization that is contained in SPICE is detailed in Chapter 4.

To minimize the computational effort in the equation solution, it is necessary to reorder the circuit equations to minimize the amount of fill-in and therefore preserve the sparsity of the original equations throughout the solution. The surprising result of Chapter 4 is that all of the proposed reordering algorithms yield substantially the same coefficient matrix size and operations count. The result held both for typical circuits and for a statistical sample of randomly-generated coefficient matrices. Because of the lack of improvement that is afforded by the more sophisticated reordering algorithms, the simple Markowitz method is employed in the SPICE2 program since it requires the least amount of cpu time to determine the reordering.

The particular method of coding the linear equation solution routines has a profound effect upon efficiency. It was found that assembly language solution routines require half the computational effort that is consumed by FORTRAN routines. Generating executable machine code produces another reduction in execution time at a cost of additional memory. The enormous increase in memory for ac analysis does not justify the moderate savings in execution time that is attained. In SPICE2, the dc and transient equations are solved with executable machine code, while the ac equations are solved with an assembly language subroutine.

The Newton-Raphson algorithm appears to be the unanimous choice for a nonlinear equation solution method. However, the Newton-Raphson algorithm cannot be applied to circuit simulation problems unless suitable modifications are made. As detailed in Chapter 5, a large variety of modified Newton-Raphson algorithms have been employed in circuit simulation programs. These algorithms fall naturally into the three groups of simple-limiting methods, error-reduction methods, and source-stepping methods. A comparison of the three methods showed that simple-limiting methods are the most efficient and possess satisfactory convergence characteristics. Of the simple-limiting methods, the method due to Colon proved to be the most reliable as well as the most efficient. For this reason, SPICE2 employs the simple-limiting method of Colon.

Transient analysis requires a numerical integration algorithm and a method of dynamically varying the integration timestep to maintain reasonable solution accuracy. There are many integration methods that are applicable to circuit simulation. The preliminary investigation that was made in Chapter 6 showed that a detailed comparison of integration techniques could be limited to three methods: the Trapezoidal Rule algorithm, the Gear-2 method, and Gear's variable-order algorithm. Of the three, Gear's variable-order method, which dynamically varies both the order of the integration and the timestep, seemed most promising since the use of higher-order methods theoretically should result in fewer number of timepoints in the integration to maintain a specified order. However, the mechanics of varying the order resulted in additional overhead that deteriorated the expected computational savings, and

the Trapezoidal implementation that is detailed in Chapter 6 actually resulted in less computational effort for all of the examples that were tested. Therefore, Trapezoidal integration, with a truncation-error timestep control, is employed in the SPICE2 program.

The analysis techniques that are utilized in a simulation program depend heavily upon the capabilities that are required of the overall program. The manner in which the circuit is described on input, for example, imposes notable constraints upon the choice of analysis methods. This point is illustrated by a comparison of the ASTAP program and the SPICE program. The branch relations in SPICE are predefined, so the derivatives of these branch relations, which are required in the Newton-Raphson linearization, are easy to evaluate. In the ASTAP program, the branch relations are supplied in the form of arithmetic expressions or tables, and the derivatives of these branch relations must be evaluated numerically. Clearly, an optimal linearization scheme developed for SPICE may be drastically different from an optimal linearization scheme developed for ASTAP. The conclusions that are summarized in this chapter, therefore, assume the type of program input and structure that is incorporated in SPICE.

The ultimate utility of any simulation program is measured by its usefulness to the circuit design process. In this respect, SPICE has performed admirably. The SPICE program presently is used by more than one hundred universities and semiconductor companies, and the consensus from this usage is that SPICE is a valuable design tool. There are, of course, many simulation problems for which

SPICE is not suited. Notable examples of these problems are near-sinusoidal nonlinear circuits such as oscillators and mixers. The inability to accept arbitrary branch relations as input also has hindered the application of SPICE to some special-purpose applications. Further research should provide new techniques for analyzing near-sinusoidal circuits. Additional development of SPICE also should provide for a more general input description without destroying the simplicity and efficiency of the present program.

APPENDIX 1

SPICE BENCHMARK CIRCUITS

The utility of a circuit simulation program ultimately is determined by applying the program to practical circuits. Different groups of engineers may be involved in the design of totally different types of circuits, and therefore may require totally different capabilities in a simulation program. The comparison of simulation programs, or of different algorithms within a simulation program, is of necessity partially subjective since the test circuits are chosen to reflect typical applications.

The circuits that are used throughout this thesis are detailed in this appendix. Most of these circuits are representative of typical analog and digital integrated circuits. Some of the circuits were chosen to emphasize particular simulation problems, such as convergence in dc analysis or truncation error in transient analysis.

The group of standard benchmark test circuits is listed in Table A.1. These ten circuits include five linear integrated circuits (LIC's) and five digital integrated circuits (DIC's). The LIC's include the differential pair that is shown in Fig. A1.1, the RCA 3040 wideband amplifier that is illustrated in Fig. A1.2, the μA 709 operational amplifier that is given in Fig. A1.3, the μA 741 operational amplifier that is illustrated in Fig. A1.4, and the μA 727 amplifier that is shown in Figs. A1.5a - A1.5b. The complexity of these five LIC's ranges from 14 nodes and 32 branches,[1]

[1]The BJT model that is implemented in SPICE2 contains 5 branches if no ohmic resistances are included. If r_b is included, the model contains 6 branches, and if r_c is included, the model contains 7 branches. The SPICE2 diode model contains 2 branches, if the diode ohmic resistance is omitted, and 3 branches if r_s is included.

	NODES	ELEMENTS	DIODES	BJT's	BRANCHES
DIFFPAIR	14	12	0	4	32
RCA3040	30	26	0	11	81
UA709	41	39	0	15	114
UA741	49	41	0	23	156
UA727	58	53	0	22	163
RTLINV	11	8	0	2	20
TTLINV	27	16	3	5	52
ECLGATE	36	25	2	8	77
MECLIII	49	35	4	11	109
SBDGATE	54	35	11	8	105

Table A1.1. The Ten Standard Benchmark Circuits

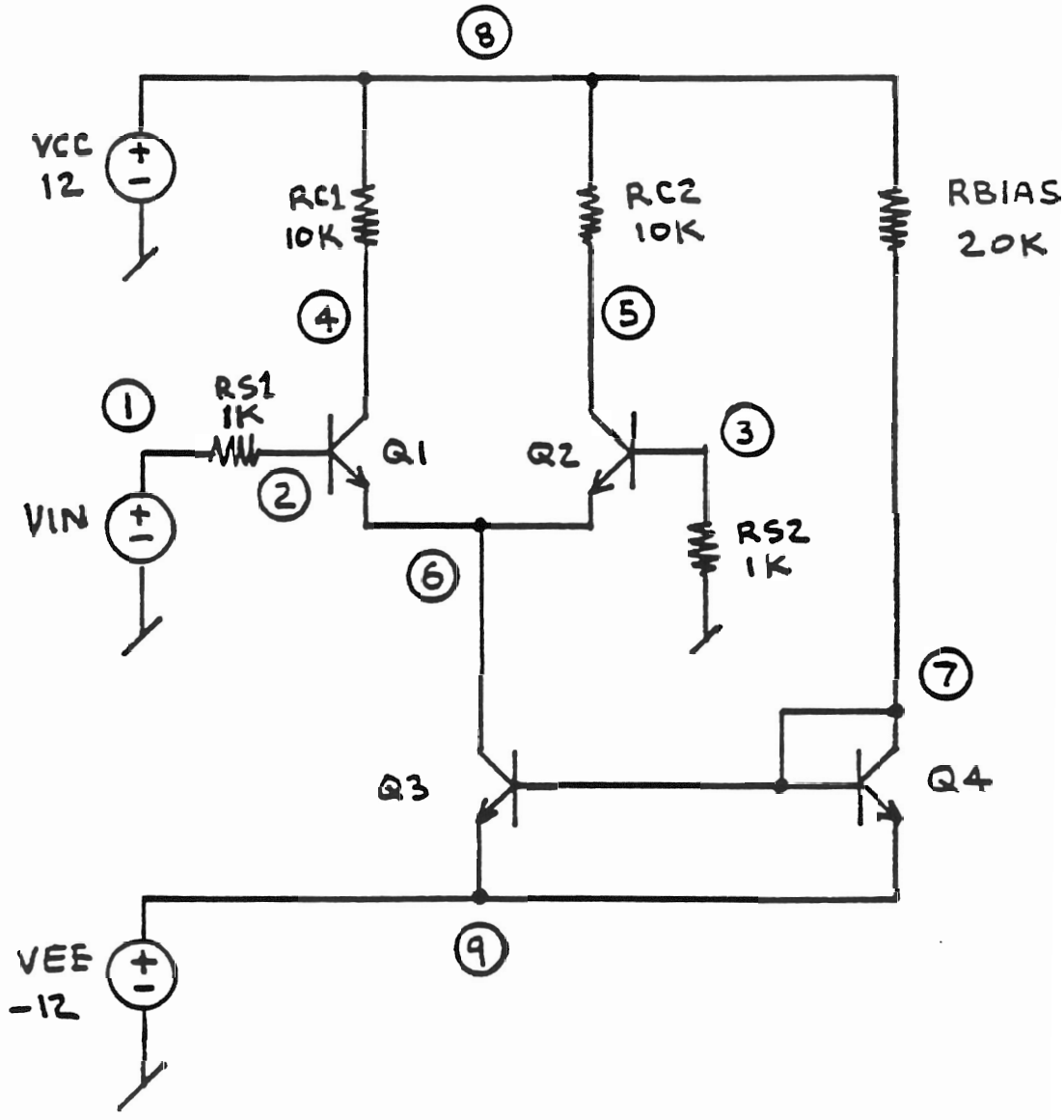


Fig. A1.1. Differential-Pair Circuit (DIFFPAIR).

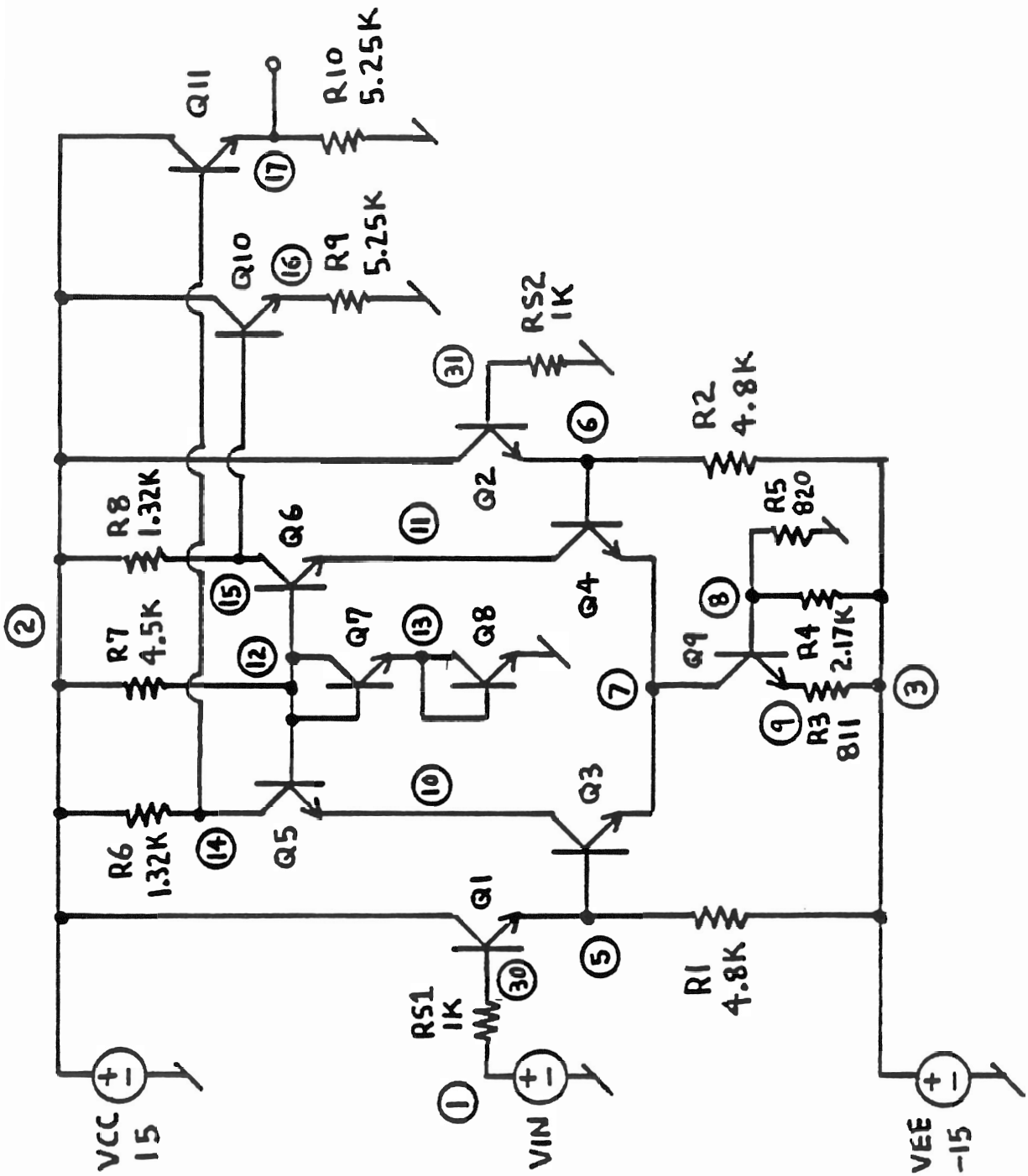


Fig. A1.2. RCA3040 Wideband Amplifier (RCA3040).

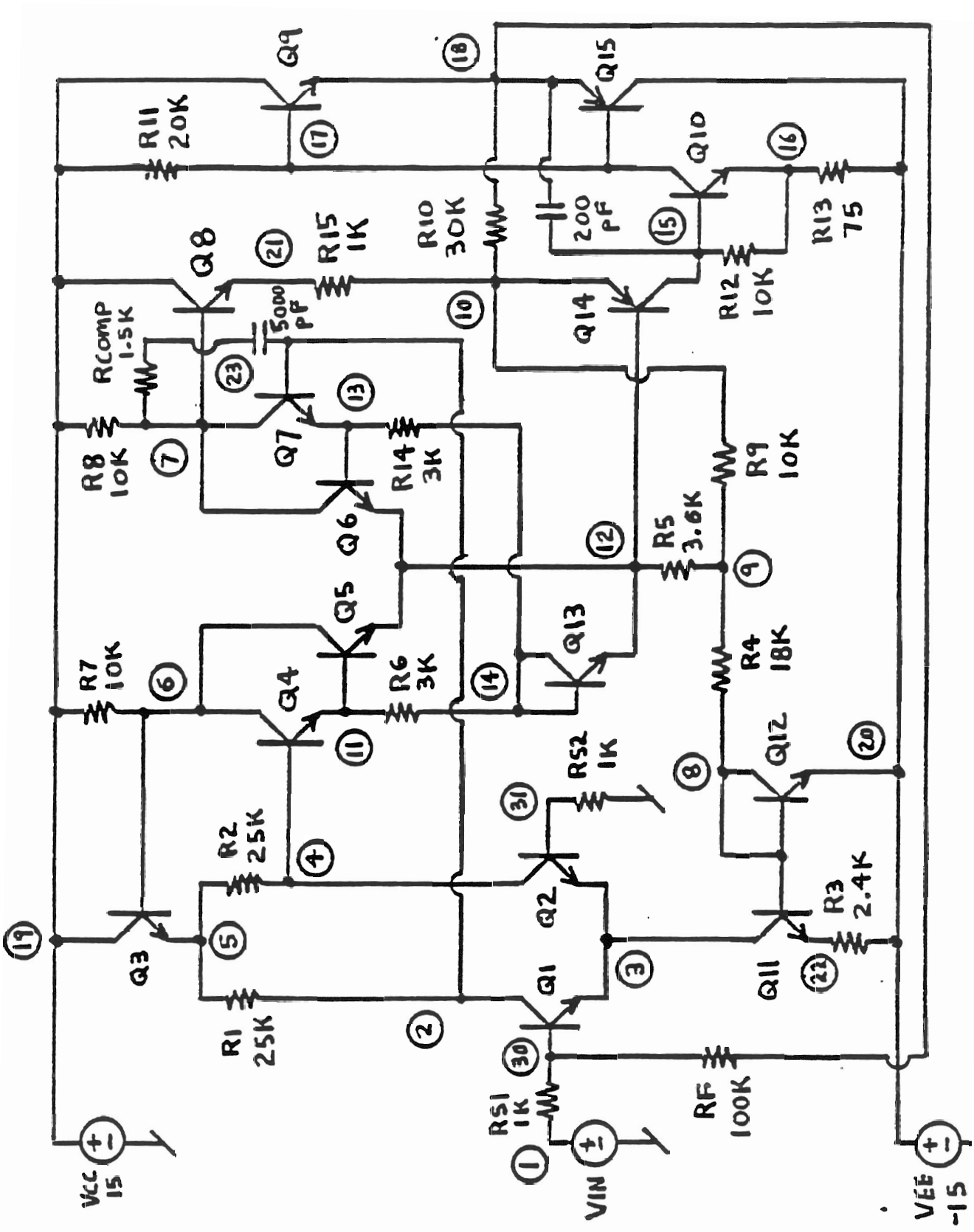


Fig. A1.3. μA 709 Operational Amplifier (UA709).

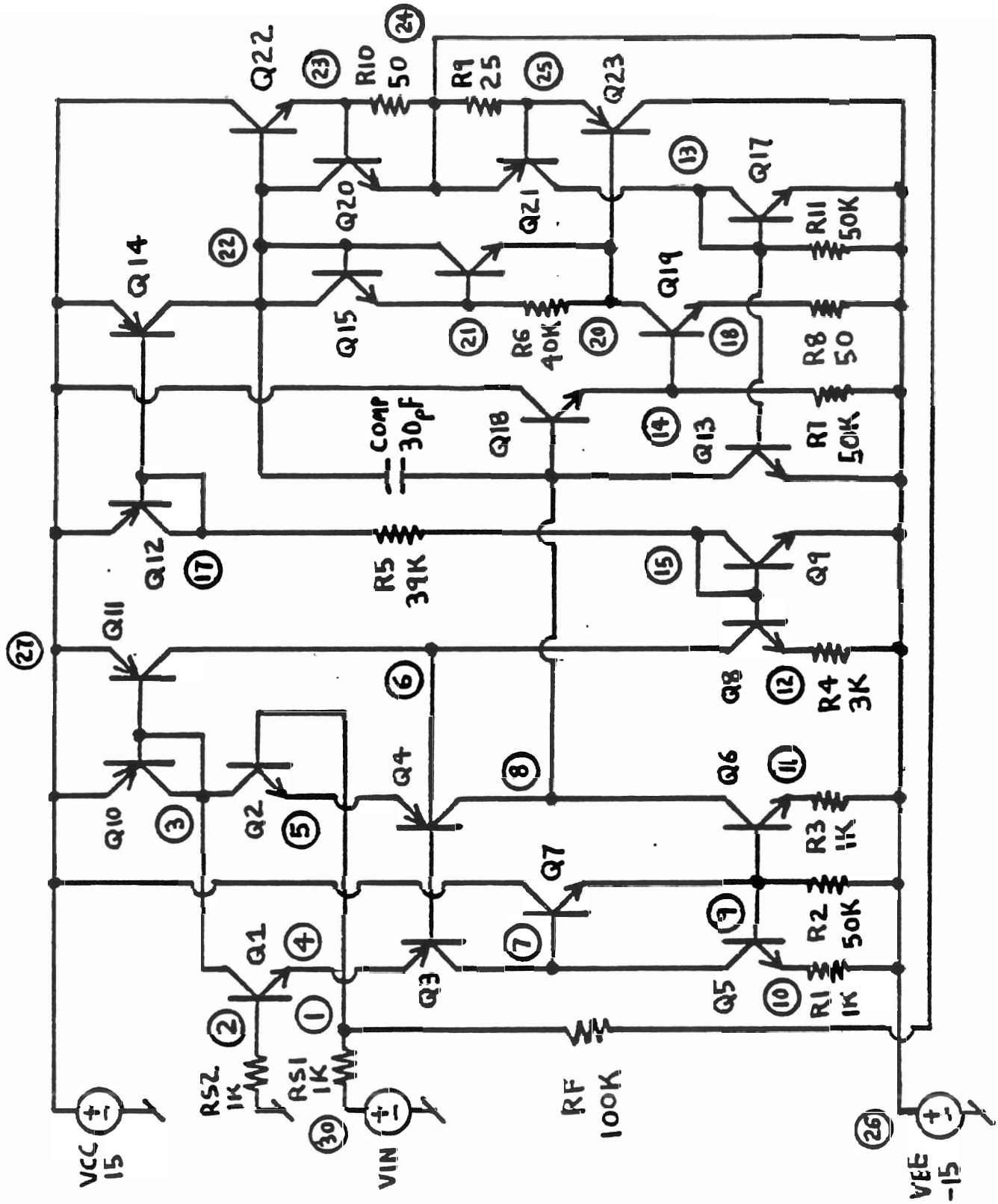


Fig. A1.4. μA 741 Operational Amplifier (UA741).

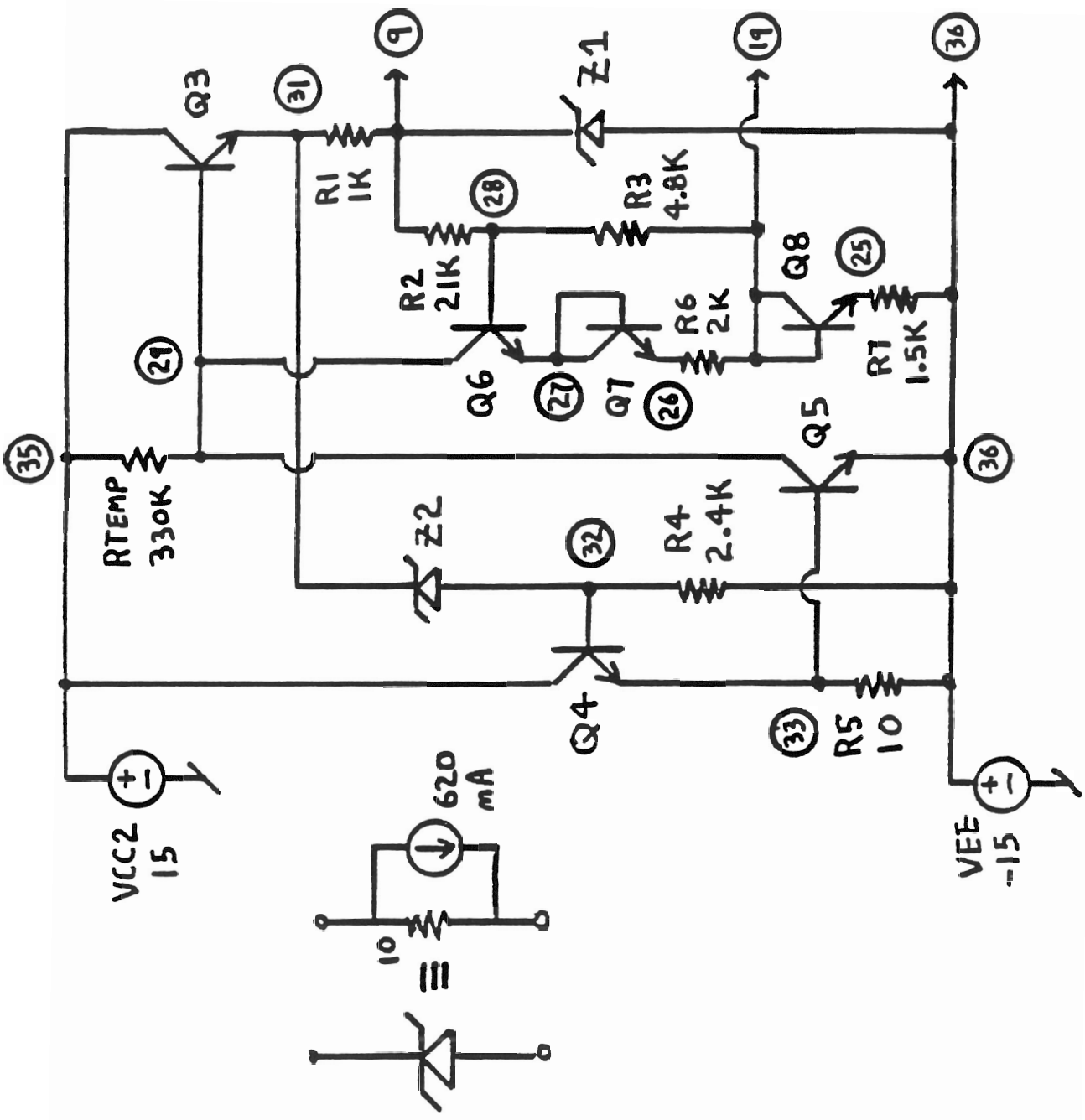


Fig. A1.5a. Voltage-Reference Portion of μA727 Amplifier (UA727).

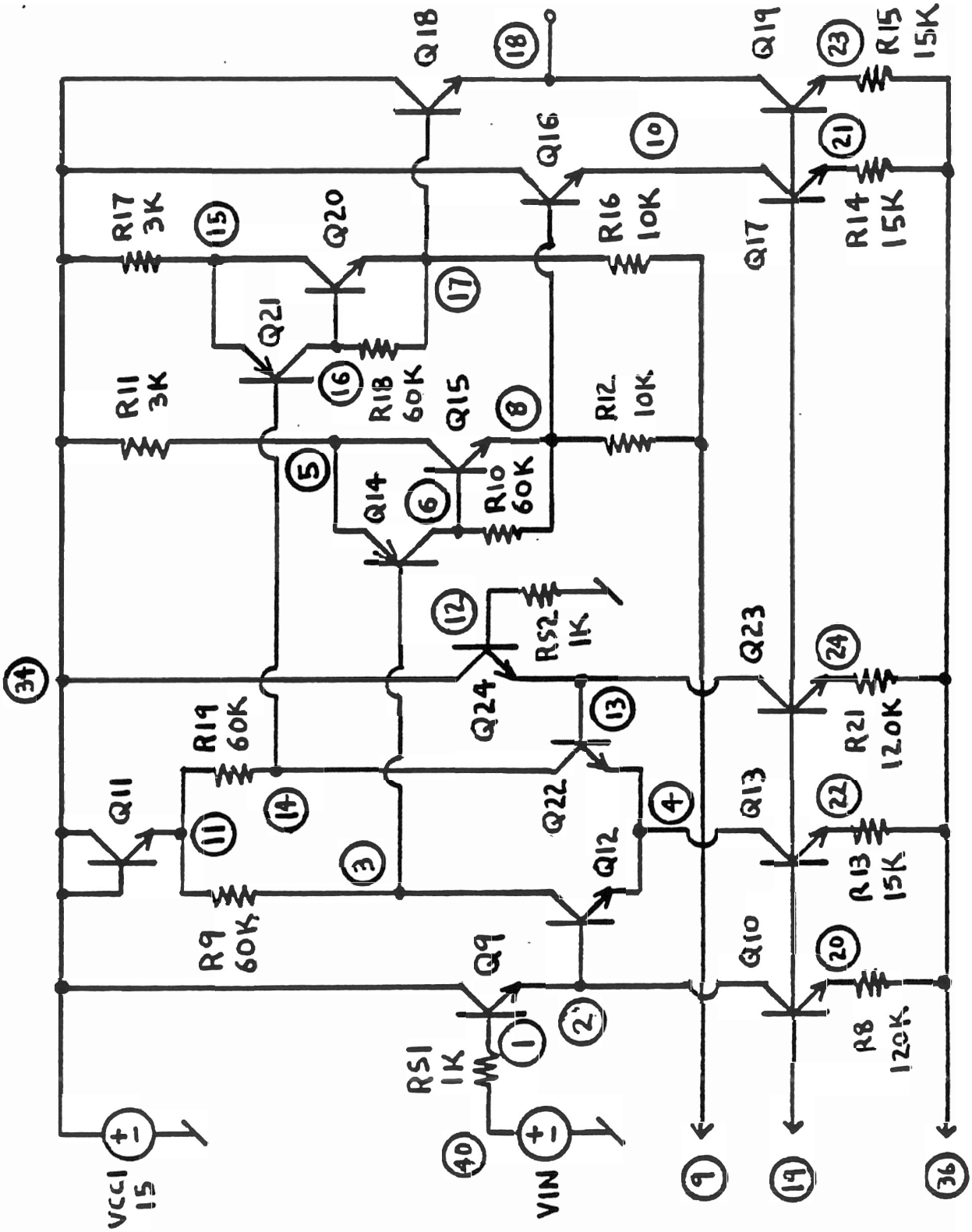


Fig. A1.5b. Amplifier Portion of μA727 Circuit (UA727).

for the differential pair, to 58 nodes and 163 branches for the μA 727 amplifier. The BJT model parameters that are used for these five circuits are given in Table A1.2. These parameter values are typical of LIC devices.

The five DIC's include the two cascaded RTL inverters that are given in Fig. A1.6, the TTL inverter that is given in Fig. A1.7, the ECL logic gate that is shown in Fig. A1.8, the MECL III ECL gate that is shown in Fig. A1.9, and the Schottky-TTL inverter that is shown in Fig. A1.10. The BJT and diode models that are used for these five circuits are given in Table A1.3. These parameter values are typical for standard DIC devices. Although the collector ohmic resistance, r_c, is omitted in the LIC's, it is included for the DIC's because the effect of r_c is more important for saturating digital circuits. The complexity of these five DIC's ranges from 11 nodes and 20 branches, for the RTL inverter, to 54 nodes and 105 branches for the Schottky-TTL inverter.

The reliability of convergence of the nonlinear solution algorithm is exceptionally important in a simulation program. However, the convergence properties of nonlinear methods are determined only by empirical tests. Eight additional circuits were included in the convergence tests that are described in Chapter 5 of this thesis to provide a larger class of test circuits. These circuits are listed in Table A1.4.

The first four circuits that are given in Table A1.4 are taken from the Master's report of Kao [58]. These circuits include the constant-current source that is shown in Fig. A1.11, the oscillator circuit that is shown in Fig. A1.12, the μA 733 video amplifier

	(NPN's) QNL	(PNP's) QPL	Units
BF	80	10	-
BR	1	1	-
IS	10^{-14}	10^{-14}	a
RB	100	20	ohms
VA	50	50	v
TF	0.3	1.0	ns
TR	6.0	20.0	ns
CCS	2.0	0	pf
CJE	3.0	6.0	pf
CJC	2.0	4.0	pf

Table A1.2. BJT Model Parameters for the Five LIC's.

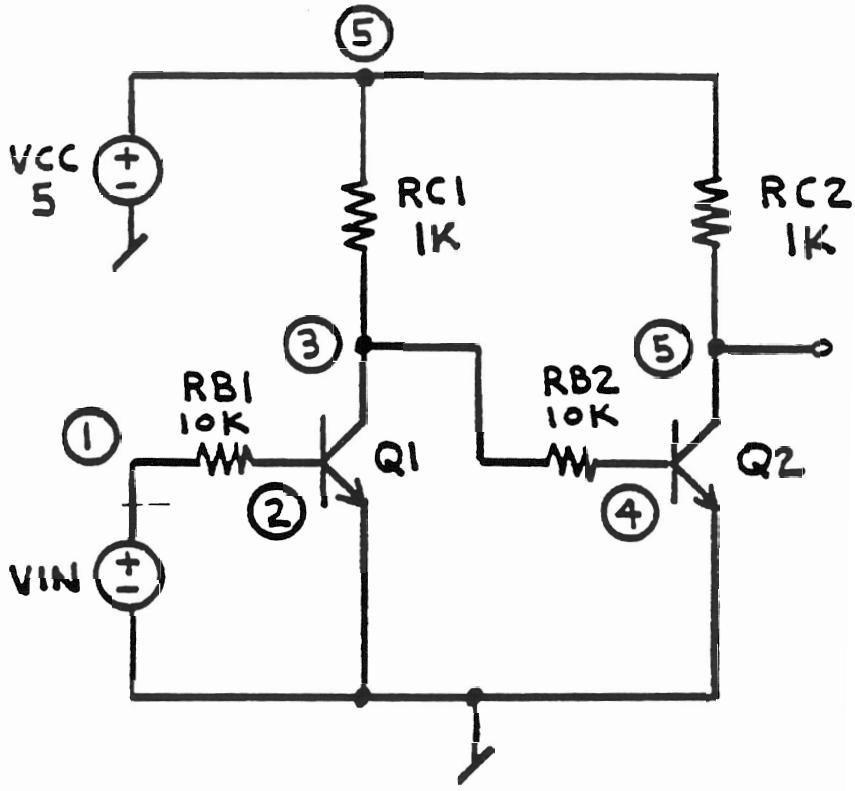


Fig. A1.6. Cascaded RTL Inverters (RTLINV).

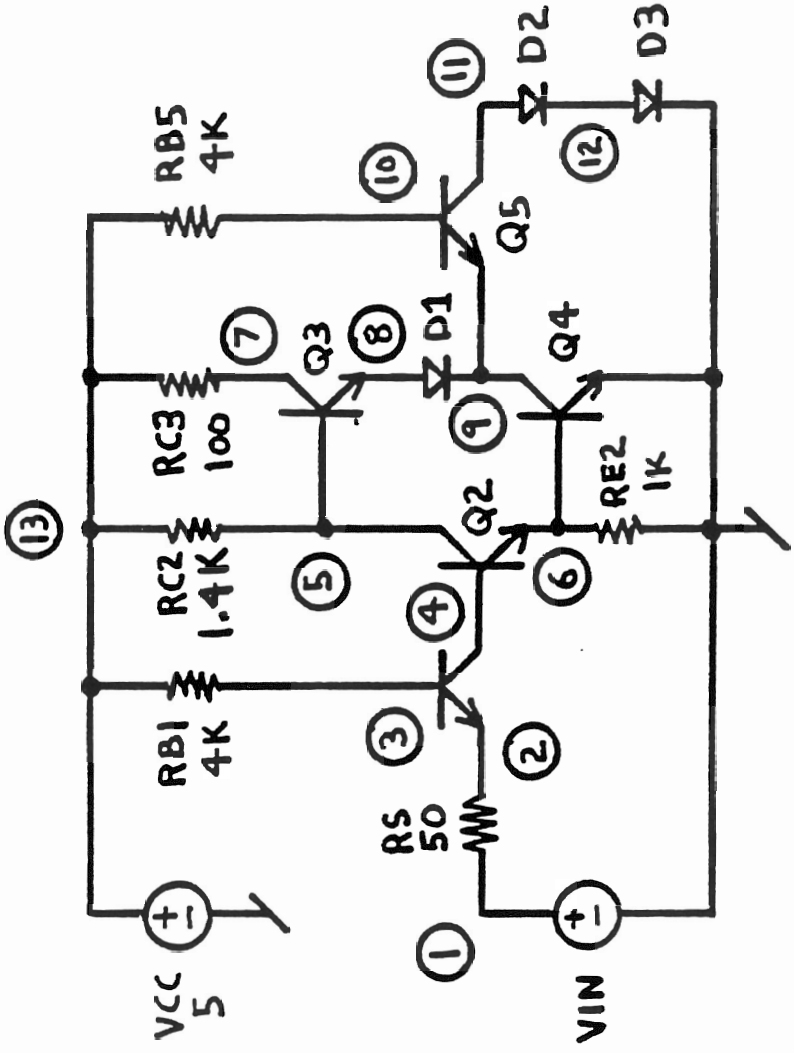


Fig. A1.7. TTL Inverter (TTLINV).

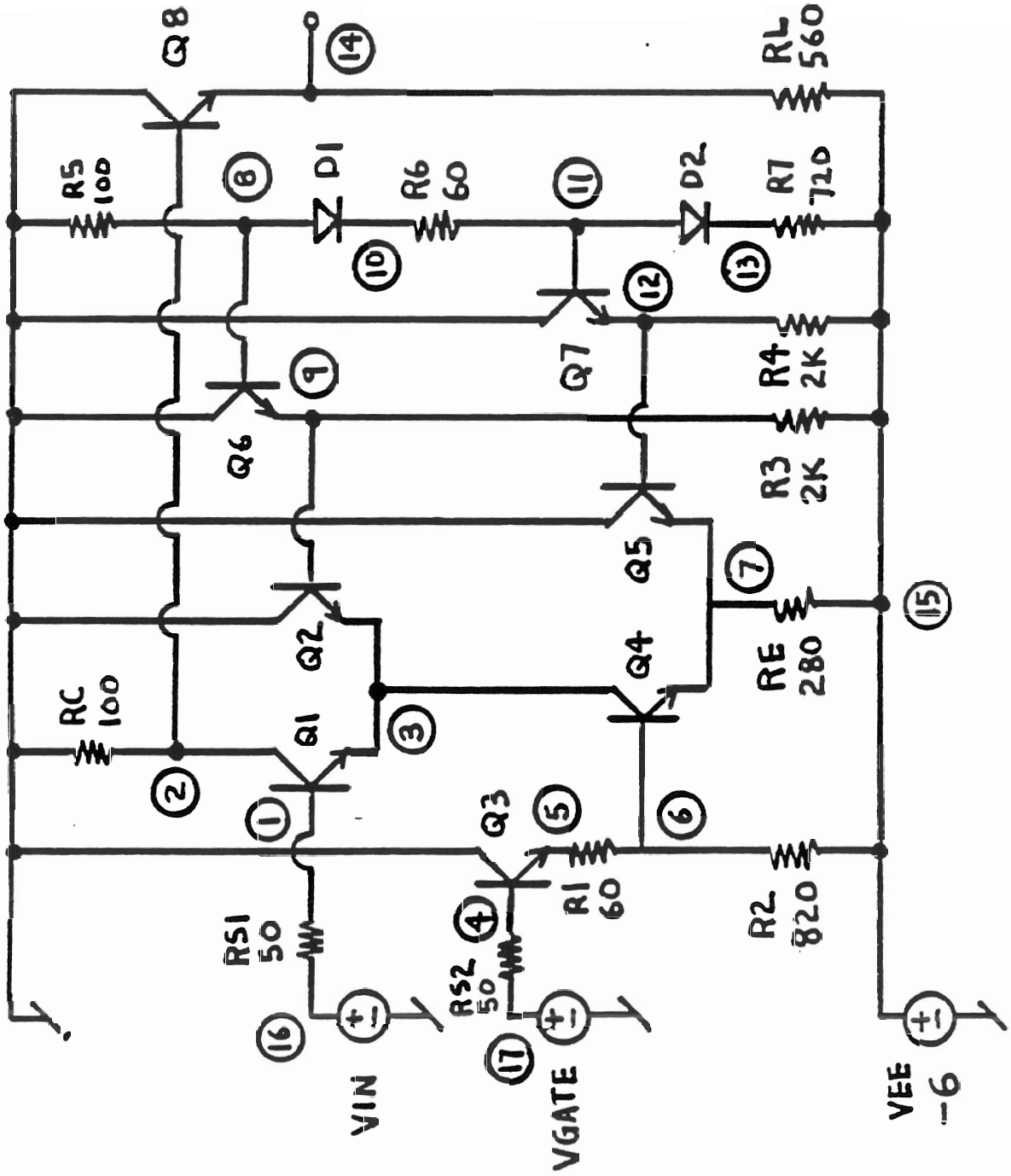


Fig. A1.8. ECL Logic Gate (ECLGATE).

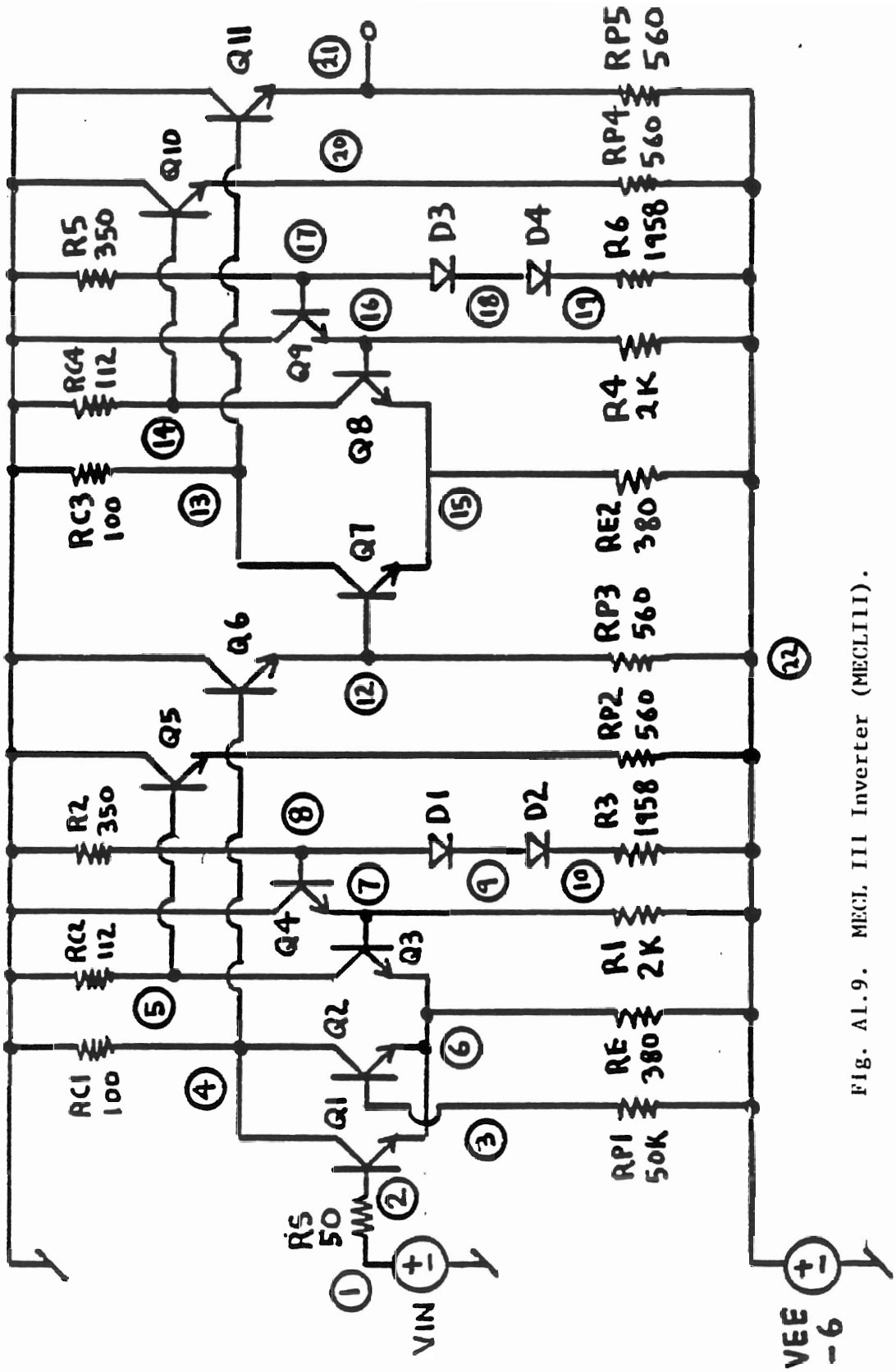


Fig. A1.9. MECL III Inverter (MECL111).

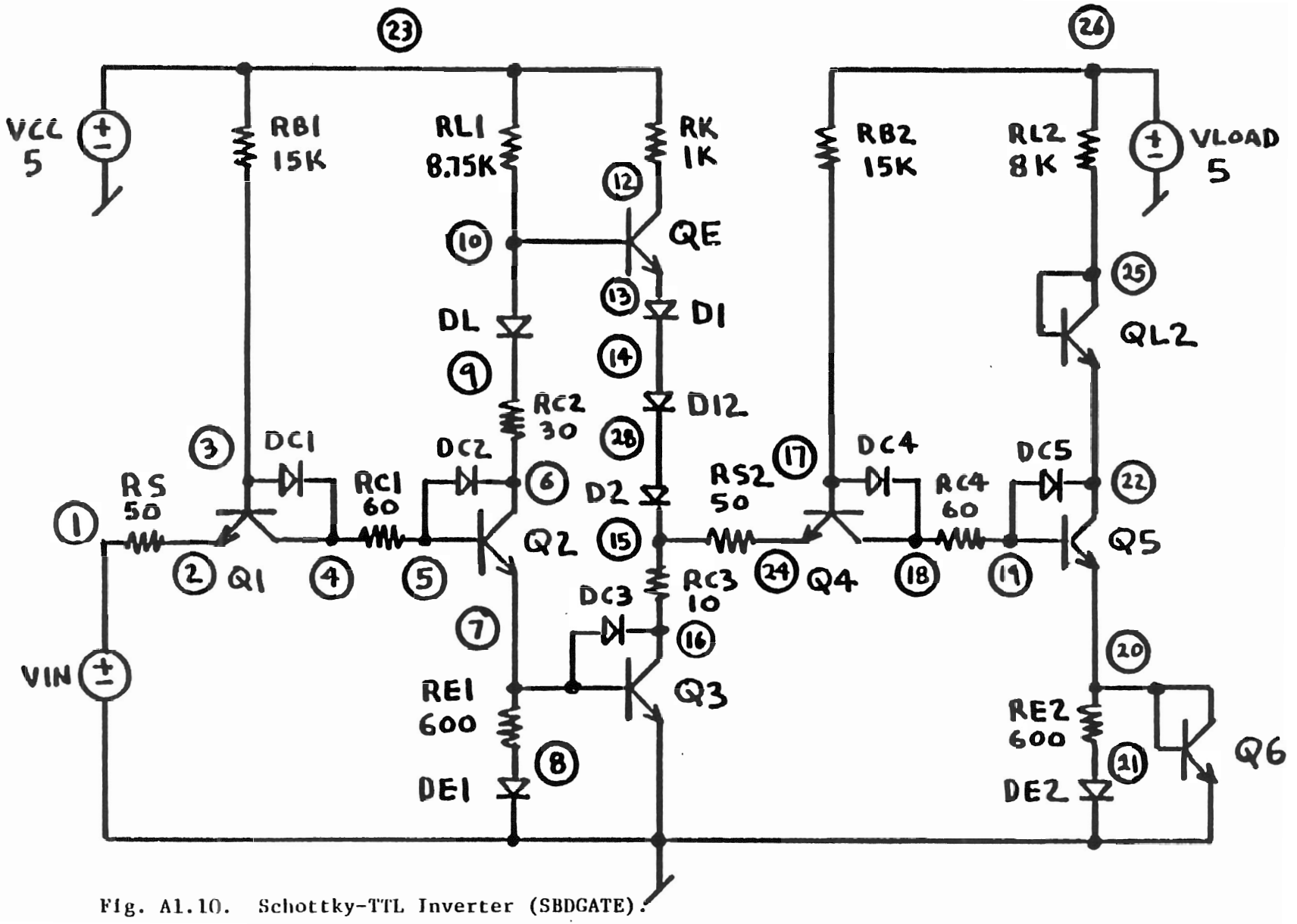


Fig. A1.10. Schottky-TTL Inverter (SBDGATE).

	QND	UNITS
BF	50	-
BR	1	-
IS	10^{-14}	amps
RB	70	ohms
RC	40	ohms
VA	50	volts
TF	0.1	ns
TR	10	ns
CCS	2	pf
CJE	0.9	pf
CJC	1.5	pf
PC	0.85	volts

	(DIODES)		
	D1	D2	UNITS
IS	10^{-14}	$5x10^{-10}$	amps
RS	40	15	ohms
TT	0.1	0	ns
CJD	0.9	0.2	pf
PB	1.0	0.6	volts

Table A1.3. The BJT and Diode Model Parameters for the Five DIC's.

	NODES	ELEMENTS	DIODES	BJT's	BRANCHES
CCSOR	11	13	0	6	37
OSC	14	19	0	7	47
UA733	23	30	0	11	74
CFFLOP	11	17	0	4	33
74TTL	27	16	3	5	52
74STTL	32	18	2	7	64
74LTTL	27	16	3	5	52
9200TTL	29	18	2	6	58

Table A1.4. The Eight DC Convergence Test Circuits.

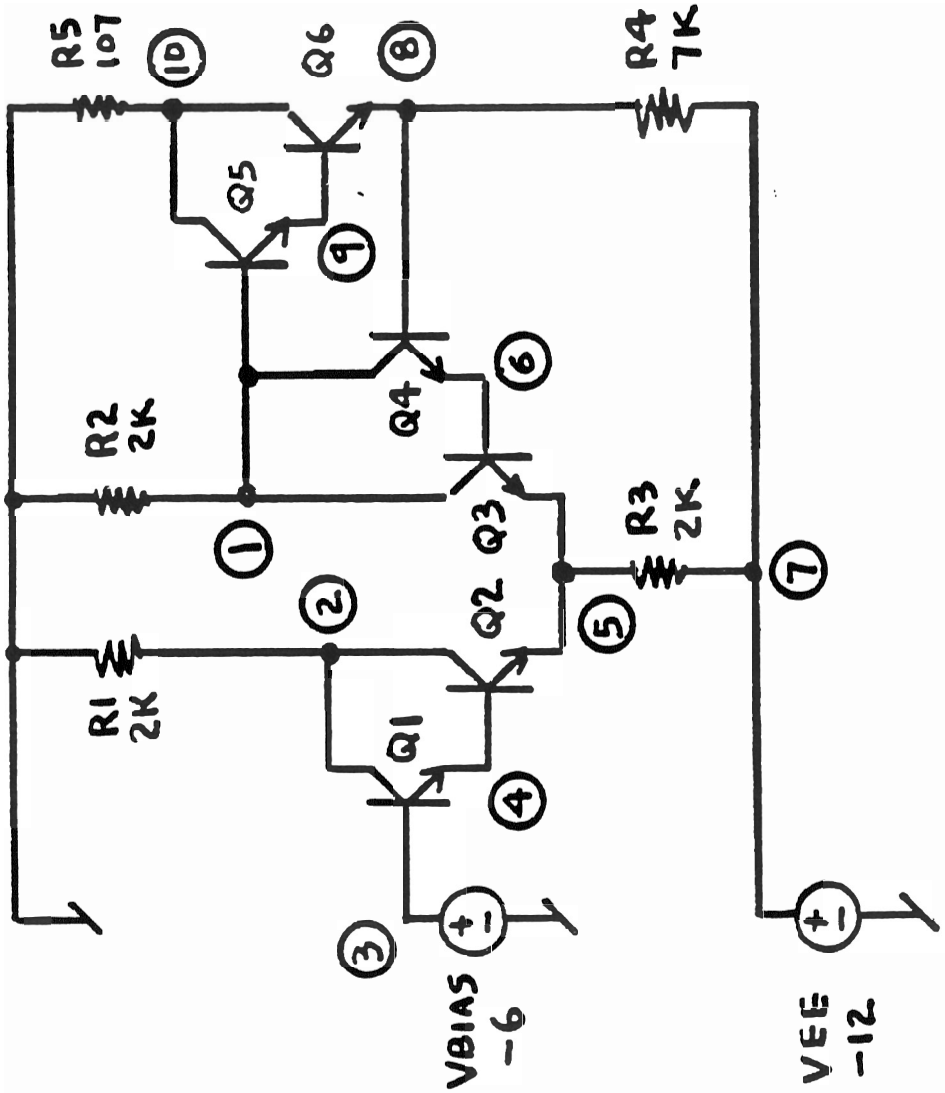


Fig. A1.11. Constant Current Source (CCSOR).

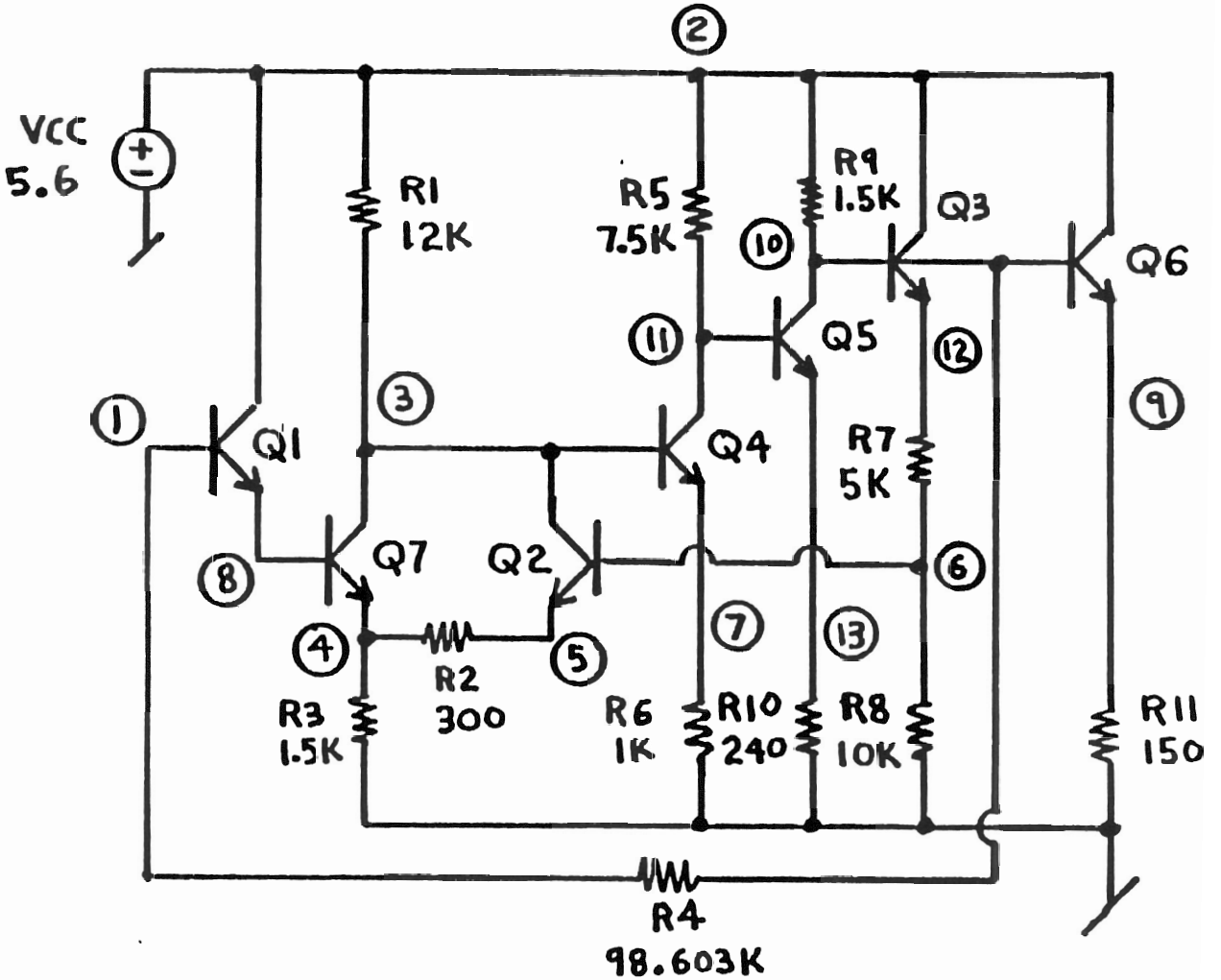


Fig. A1.12. Oscillator Circuit (OSC).

that is given in Fig. A1.13, and the complementary flip-flop that is illustrated in Fig. A1.14. The BJT model parameters for these four circuits are listed in Table A1.5.

The remaining four circuits that are listed in Table A1.4 are TTL circuits that were provided by D. A. Hodges [76]. Considerable convergence problems have been encountered in the simulation of these circuits with SPICE. These circuits include the 74TTL inverter that is given in Fig. A1.15, the 74STTL circuit that is illustrated in Fig. A1.16, the 74LTTL circuit that is given in Fig. A1.17, and the 9200TTL circuit that is shown in Fig. A1.18. Although the input voltage source (VIN) typically is either high (2.5-5.0 v) or low (0-0.5 v), the convergence problems are encountered when the source VIN is set to the values that are given in these figures. The BJT and diode model parameters for these four circuits are given in Table A1.6.

Five additional circuits are included for testing the numerical integration algorithms that are used in transient analysis. Each of these circuits contain time-constants that are widely separated, so the timestep that is used must vary over several orders of magnitude to determine the circuit response accurately. These circuits were made as small as possible to minimize the cost of repetitive transient analyses.

This group of circuits includes the RC circuit that is illustrated in Fig. A1.19, the choke circuit that is shown in Fig. A1.20, the ECL circuit that is given in Fig. A1.21, the Schmitt trigger circuit that is shown in Fig. A1.22, and the astable multivibrator that is shown in Fig. A1.23. The BJT and diode model

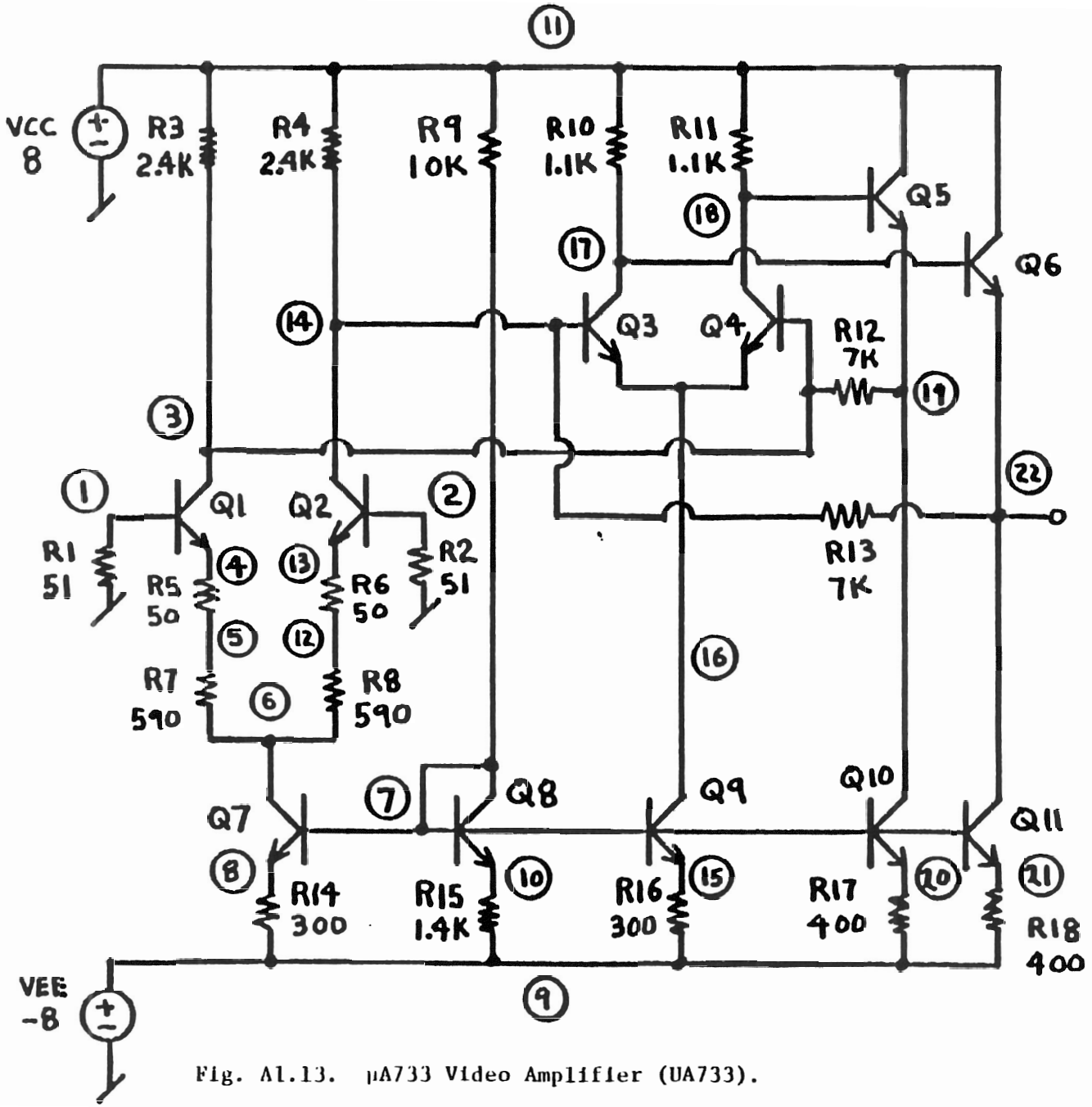


Fig. A1.13. μA733 Video Amplifier (UA733).

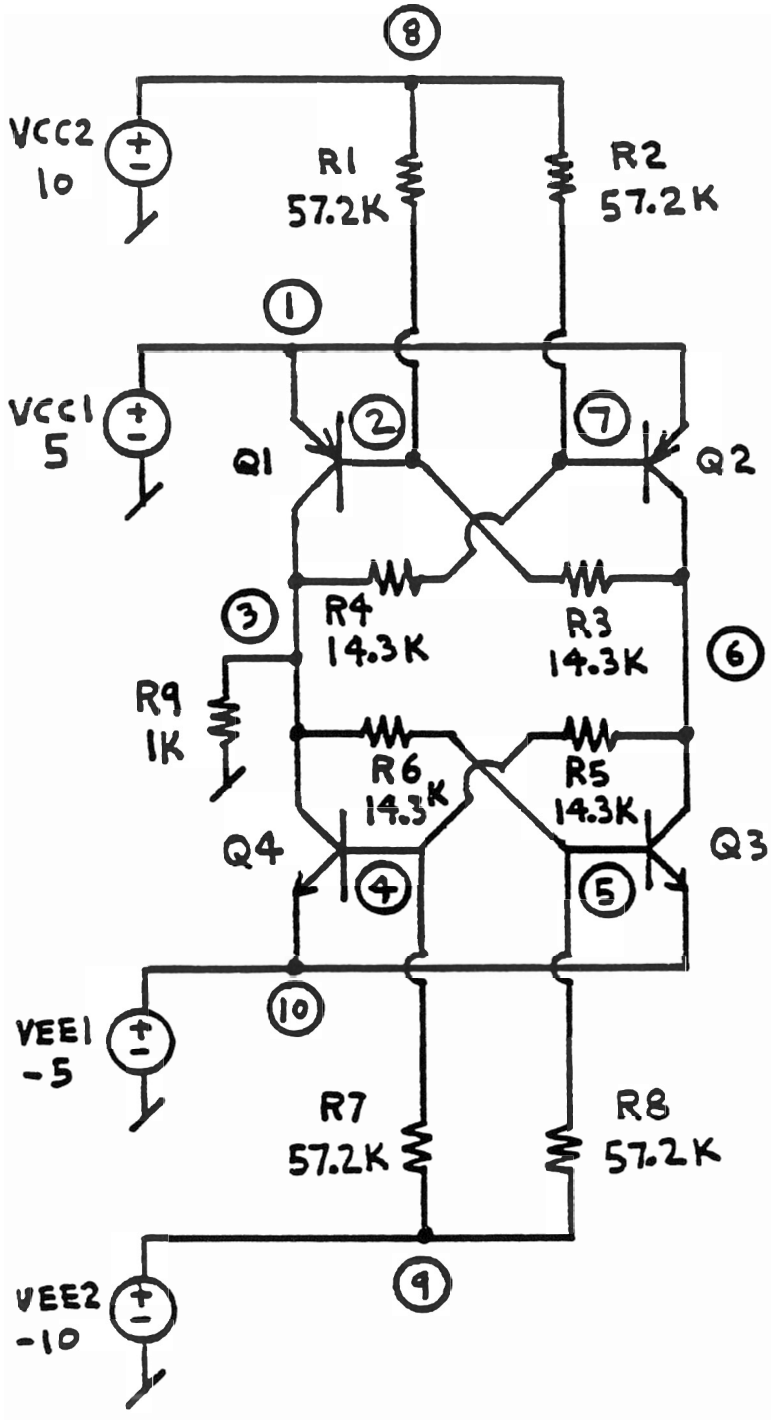


Fig. A1.14. Complementary Flip-Flop Circuit (CFFLOP).

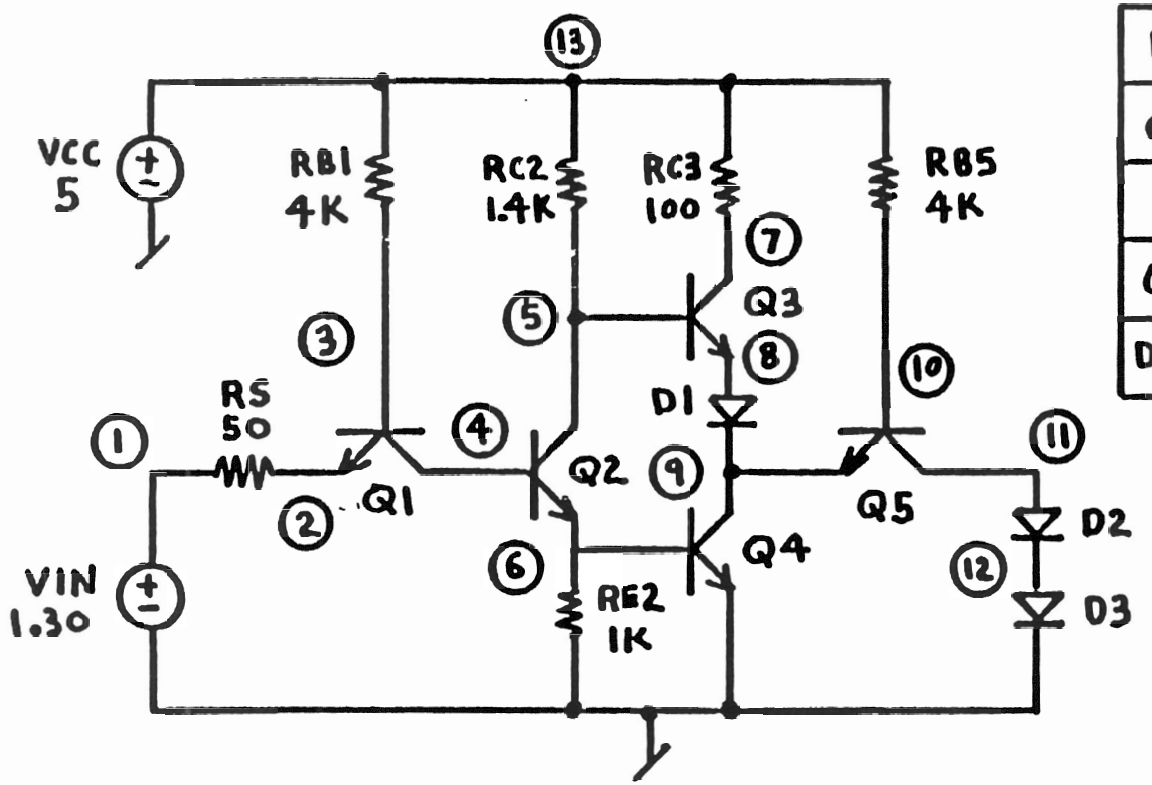
CCSOR	BF	49.5
	BR	0.5
	IS	9.802×10^{-16}

OSC	BF	60
	BR	0.205
	IS	1.21×10^{-15}

UA733	BF	100
	BR	2
	IS	9.901×10^{-16}

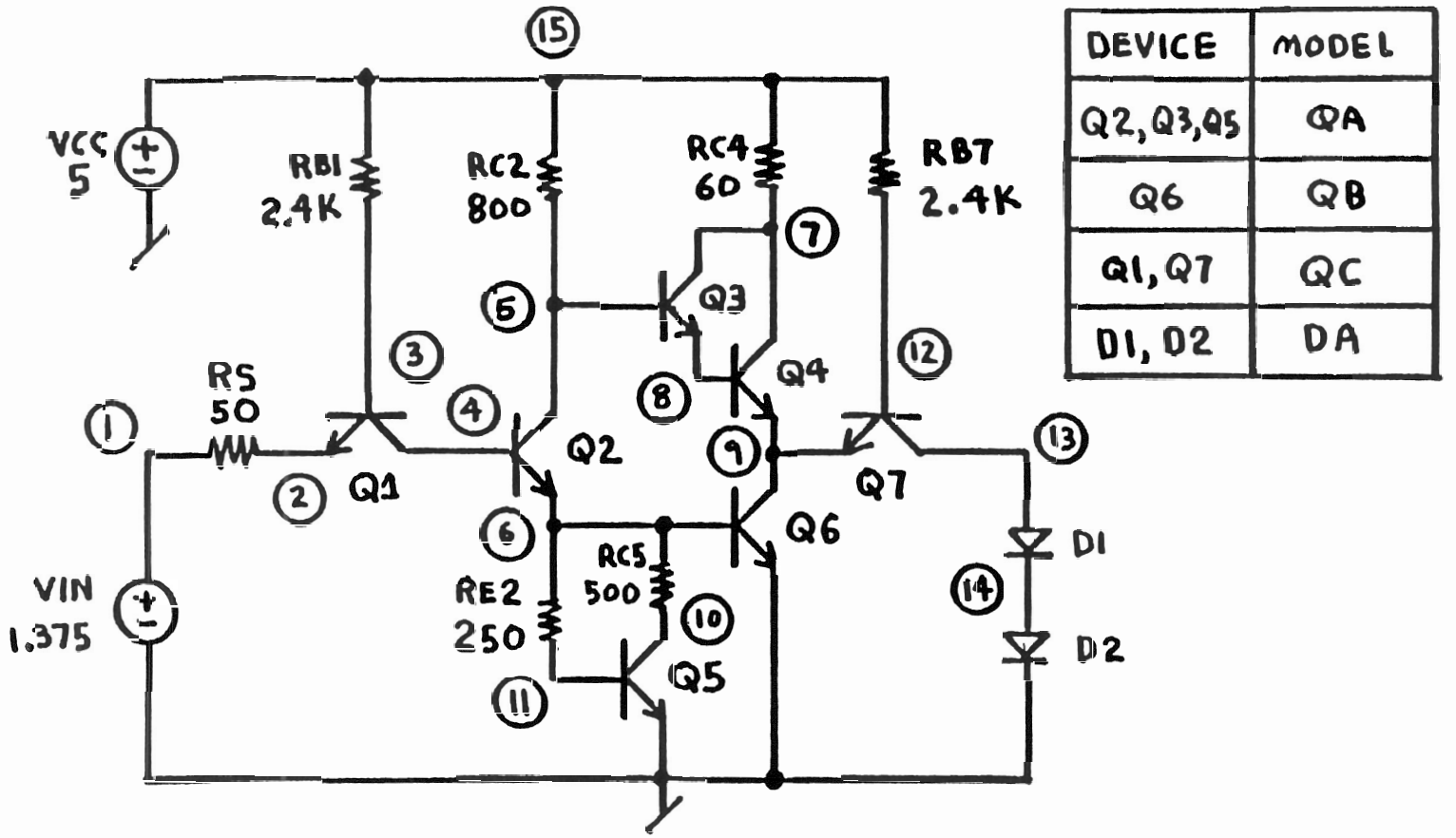
CFFLOP (npn's)	BF	10
	BR	1
	IS	9.1×10^{-15}
CFFLOP (pnp's)	BF	10
	BR	1
	IS	9.1×10^{-15}

Table A1.5. BJT Model Parameters for the Four Circuits from Kao.



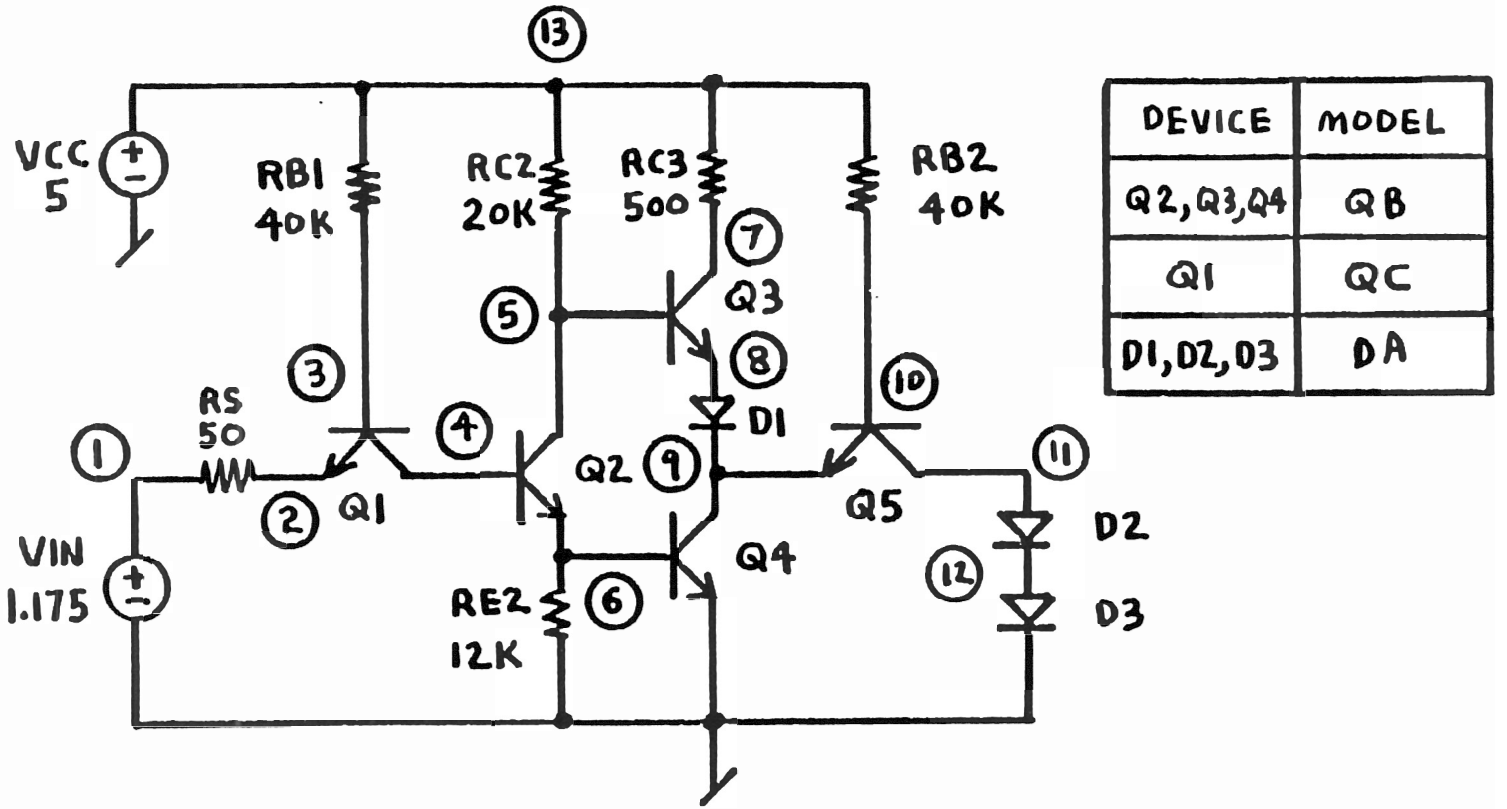
DEVICE	MODEL
Q2, Q3	QA
Q4	QB
Q1, Q5	QC
D1, D2, D3	DA

Fig. A1.15. The 74TTL Inverter (74TTL).



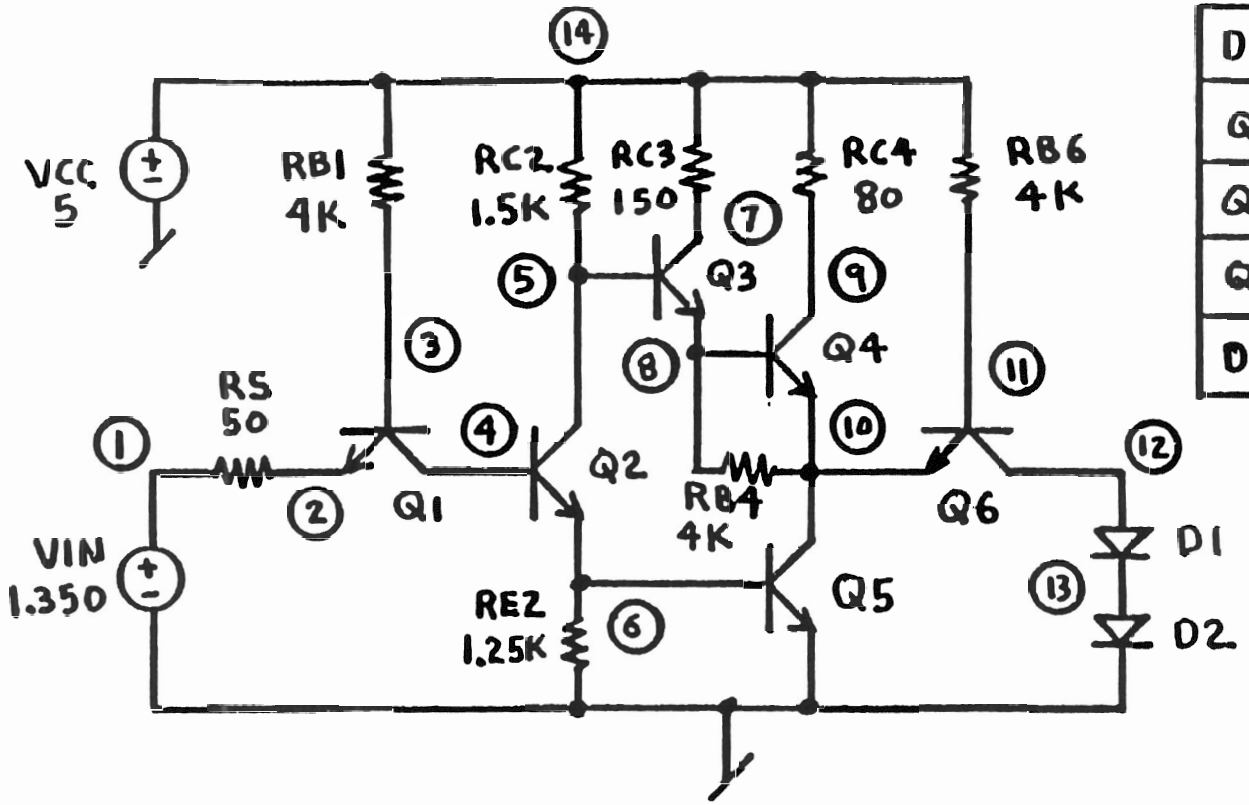
DEVICE	MODEL
Q2, Q3, Q5	QA
Q6	QB
Q1, Q7	QC
D1, D2	DA

Fig. A1.16. The 74STTL Inverter (74STTL).



DEVICE	MODEL
Q2,Q3,Q4	QB
Q1	QC
D1,D2,D3	DA

Fig. A1.17. The 74LTTL Inverter (74LTTL).



DEVICE	MODEL
Q2, Q3	QA
Q4, Q5	QB
Q1, Q6	QC
D1, D2	DA

Fig. A1.18. The 9200TTL Inverter (9200TTL).

	QA	QB	QC	Units
BF	20	20	20	-
BR	1	0.2	0.02	-
IS	$1x10^{-14}$	$1.6x10^{-14}$	$1x10^{-14}$	amps
RB	70	20	500	ohms
RC	40	12	40	ohms
VA	50	50	50	volts

	DA	Units
IS	$1x10^{-14}$	amps
RS	40	ohms

Table A1.6. The Model Parameters for the Four TTL Circuits.

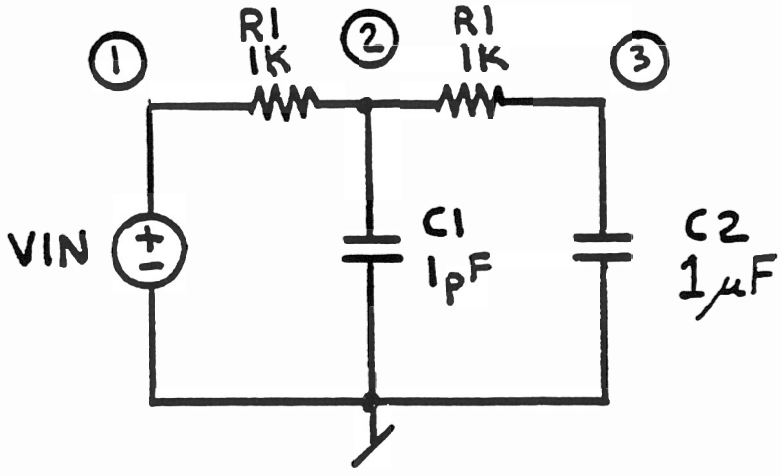


Fig. A1.19. Split Time-Constant RC Circuit (RC).

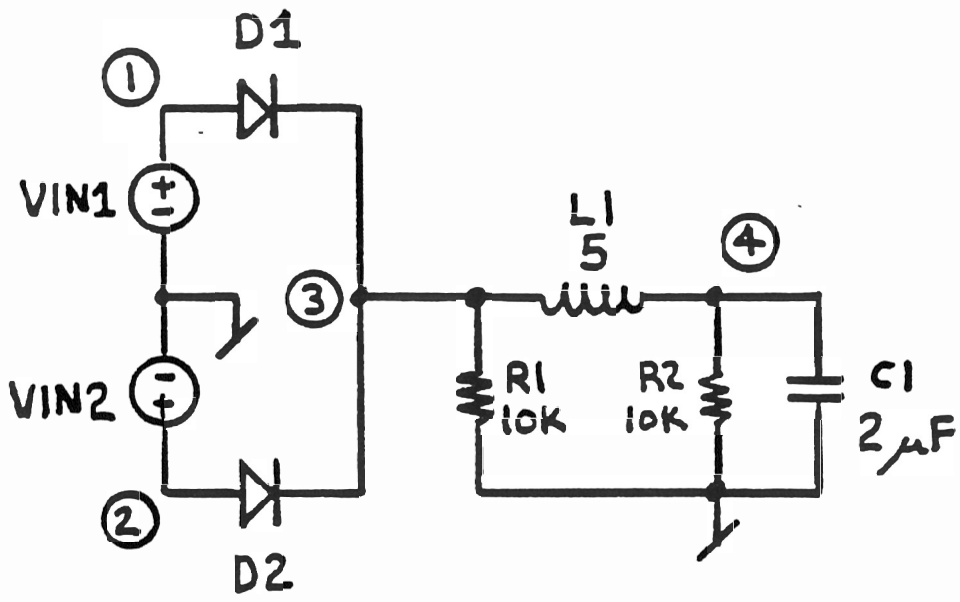


Fig. A1.20. Choke Circuit (CHOKE).

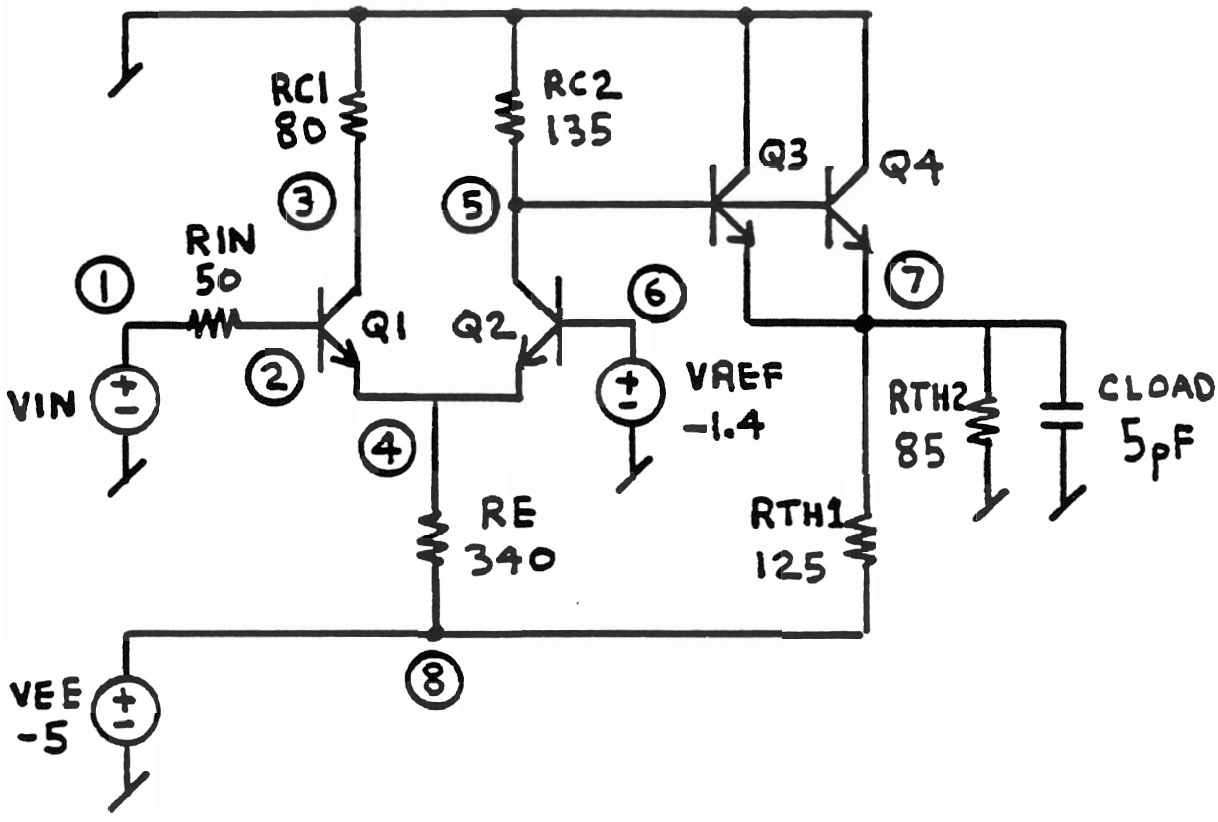


Fig. A1.21. ECL Inverter (ECL).

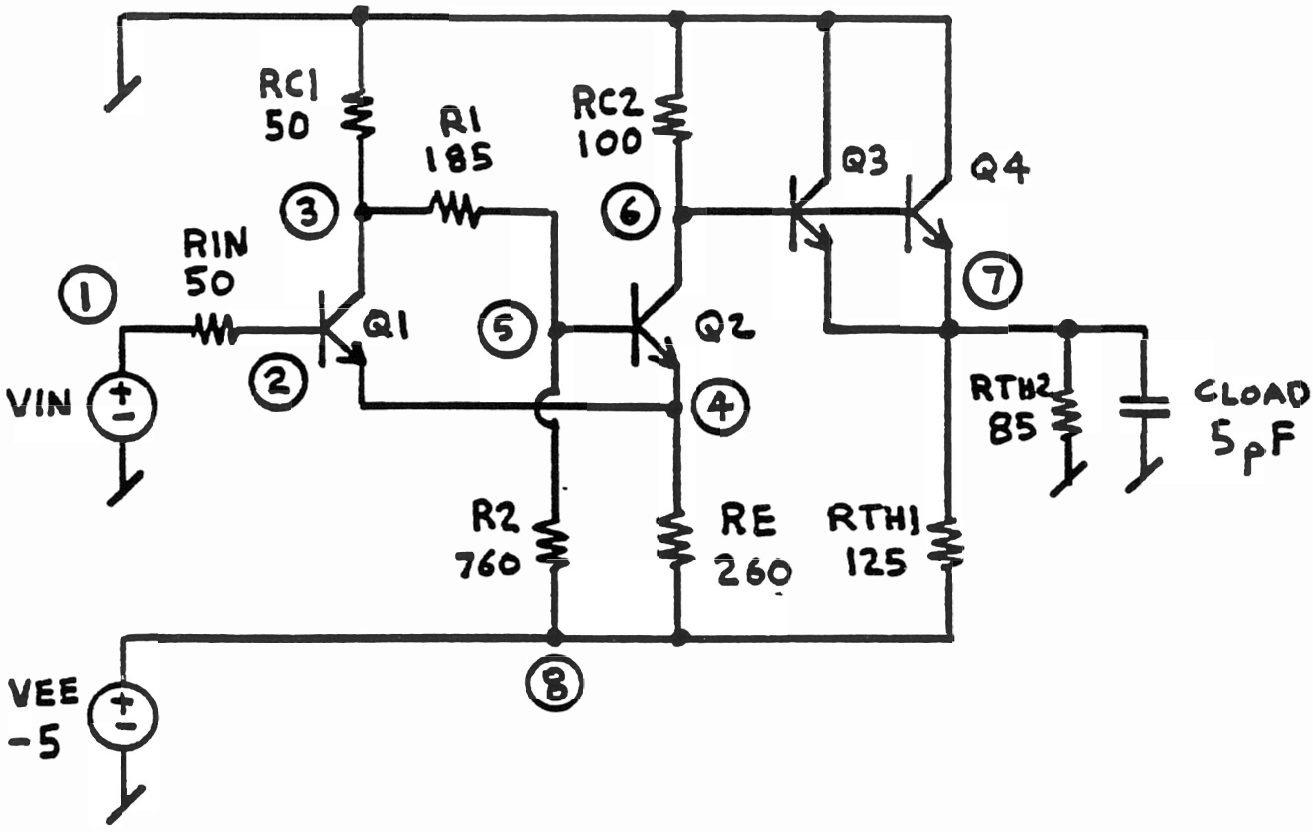


Fig. A1.22. Schmitt Trigger Circuit (SCHMITT).

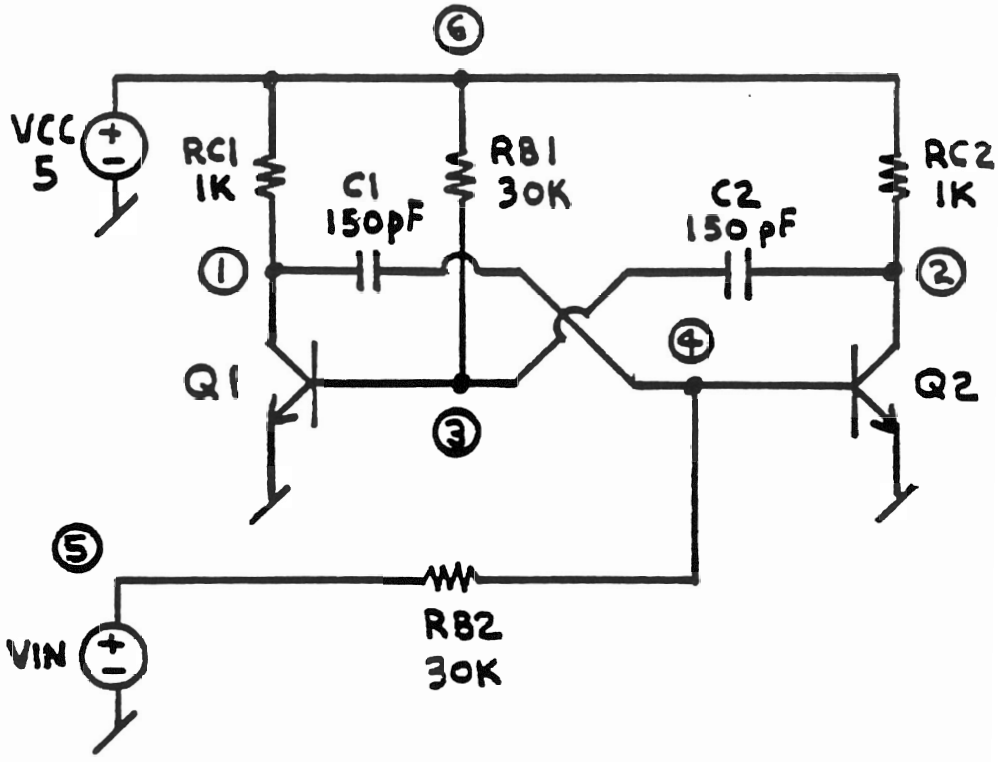


Fig. A1.23. Astable Multivibrator (ASTABLE).

parameters that are used in these five circuits are listed in Table A1.7.

The remainder of this Appendix contains the listings of the SPICE input decks for the circuits that have been described in this appendix.

CHOKE Diode Model Parameters

IS	$1x10^{-14}$	amps
CJO	10	pf

ECL, SCHMITT, and ASTABLE BJT Model Parameters

BF	50	-
BR	0.1	-
IS	$1x10^{-16}$	amps
RB	50	ohms
RC	10	ohms
TF	0.12	ns
TR	5	ns
CJE	0.4	pf
PE	0.8	v
ME	0.4	-
CJC	0.5	pf
PC	0.8	v
MC	0.333	-
CCS	1	pf
VA	50	v

Table A1.7. Device Model Parameters for the Transient Test Circuits

```
DIFFPAIR CKT - SIMPLE DIFFERENTIAL PAIR
*   AC DEC 10 1 10GHZ
*   DC VIN -0.25 0.25 0.005
*   TRAN 5NS 500NS
.TRAN 5NS 500NS
VIN 1 0 SIN(0 0.1 5MEG 5NS) AC 1
VCC 8 0 12
VEE 9 0 -12
Q1 4 2 6 QNL
Q2 5 3 6 QNL
RS1 1 2 1K
RS2 3 0 1K
RC1 4 8 10K
RC2 5 8 10K
Q3 6 7 9 QNL
Q4 7 7 9 QNL
RBIAS 7 8 20K
.OUTPUT V4 4 0 PRINT DC TRAN
.OUTPUT V5 5 0 PRINT MAG PHASE DC TRAN PLOT MAG PHASE DC TRAN
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
+   CJC=2PF VA=50)
.END
```

```
RCA3040 CKT - RCA 3040 WIDEBAND AMPLIFIER
*   AC DEC 10 1 10GHZ
*   DC VIN -0.25 0.25 0.005
*   TRAN 0.5NS 50NS
.TRAN 0.5NS 50NS
VIN 1 0 SIN(0 0.1 50MEG 0.5NS) AC 1
VCC 2 0 15.0
VEE 3 0 -15.0
RS1 30 1 1K
RS2 31 0 1K
R1 5 3 4.8K
R2 6 3 4.8K
R3 9 3 811
R4 8 3 2.17K
R5 8 0 820
R6 2 14 1.32K
R7 2 12 4.5K
R8 2 15 1.32K
R9 16 0 5.25K
R10 17 0 5.25K
Q1 2 30 5 QNL
Q2 2 31 6 QNL
Q3 10 5 7 QNL
Q4 11 6 7 QNL
Q5 14 12 10 QNL
Q6 15 12 11 QNL
Q7 12 12 13 QNL
Q8 13 13 0 QNL
Q9 7 8 9 QNL
Q10 2 15 16 QNL
Q11 2 14 17 QNL
.OUTPUT V16 16 0 PRINT DC TRAN
.OUTPUT V17 17 0 PRINT MAG PHASE DC TRAN PLOT MAG PHASE DC TRAN
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
+   CJC=2PF VA=50)
.END
```

```
UA709 CKT - UA 709 OPERATIONAL AMPLIFIER
*  AC DEC 10 1 10GHZ
*  DC VIN -0.25 0.25 0.005
*  TRAN 2.5US 250US
.TRAN 2.5US 250US
VIN 1 0 SIN(0 0.1 10KHZ 2.5US) AC 1
VCC 19 0 15.0
VEE 20 0 -15.0
RS1 30 1 1K
RS2 31 0 1K
RF 30 18 100K
RCOMP 7 23 1.5K
CICOMP 23 2 5000PF
COCOMP 18 15 200PF
Q1 2 30 3 QNL
Q2 4 31 3 QNL
Q3 19 6 5 QNL
Q4 6 4 11 QNL
Q5 6 11 12 QNL
Q6 7 13 12 QNL
Q7 7 2 13 QNL
Q8 19 7 21 QNL
Q9 19 17 18 QNL
Q10 17 15 16 QNL
Q11 3 8 22 QNL
Q12 8 8 20 QNL
Q13 14 14 12 QNL
Q14 15 12 10 QPL
Q15 20 17 18 QPL
R1 5 2 25K
R2 5 4 25K
R3 22 20 2.4K
R4 8 9 18K
R5 9 12 3.6K
R6 11 14 3K
R7 19 6 10K
R8 19 7 10K
R9 9 10 10K
R10 10 18 30K
R11 19 17 20K
R12 15 16 10K
R13 16 20 75.0
R14 13 14 3K
R15 21 10 1K
.OUTPUT V7 7 0 PRINT DC TRAN
.OUTPUT V18 18 0 PRINT MAG PHASE DC TRAN PLOT MAG PHASE DC TRAN
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
+  CJC=2PF VA=50)
.MODEL QPL PNP(BF=10 RB=20 TF=1NS TR=20NS CJE=6PF CJC=4PF VA=50)
.END
```

```
UA741 CKT - UA 741 OPERATIONAL AMPLIFIER
*  AC DEC 10 1 10GHZ
*  DC VIN -0.25 0.25 0.005
*  TRAN 2.5US 250US
.TRAN 2.5US 250US
VCC 27 0 15.0
VEE 26 0 -15.0
VIN 30 0 SIN(0 0.1 10KHZ 2.5US) AC 1
RS1 1 30 1K
RS2 2 0 1K
RF 24 1 100K
R1 10 26 1K
R2 9 26 50K
R3 11 26 1K
R4 12 26 3K
R5 15 17 39K
R6 21 20 40K
R7 14 26 50K
R8 18 26 50.0
R9 24 25 25.0
R10 23 24 50.0
R11 13 26 50K
COMP 22 8 30PF
Q1 3 2 4 QNL
Q2 3 1 5 QNL
Q3 7 6 4 QPL
Q4 8 6 5 QPL
Q5 7 9 10 QNL
Q6 8 9 11 QNL
Q7 27 7 9 QNL
Q8 6 15 12 QNL
Q9 15 15 26 QNL
Q10 3 3 27 QPL
Q11 6 3 27 QPL
Q12 17 17 27 QPL
Q13 8 13 26 QNL
Q14 22 17 27 QPL
Q15 22 22 21 QNL
Q16 22 21 20 QNL
Q17 13 13 26 QNL
Q18 27 8 14 QNL
Q19 20 14 18 QNL
Q20 22 23 24 QNL
Q21 13 25 24 QPL
Q22 27 22 23 QNL
Q23 26 20 25 QPL
.OUTPUT V8 8 0 PRINT DC TRAN
.OUTPUT V24 24 0 PRINT MAG PHASE DC TRAN PLOT MAG PHASE DC TRAN
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
+  CJC=2PF VA=50)
.MODEL QPL PNP(BF=10 RB=20 TF=1NS TR=20NS CJE=6PF CJC=4PF VA=50)
.END
```

```
UA727 CKT - UA 727 AMPLIFIER
*  AC DEC 10 1 10GHZ
*  DC VIN -0.5 0.5 0.01
*  TRAN 0.5US 5US
.TRAN 0.05US 5US
VCC1 34 0 15.0
VCC2 35 0 15.0
VEE 36 0 -15.0
VIN 40 0 SIN(0 0.2 500KHZ 0.05US) AC 1
RS1 40 1 1K
RS2 12 0 1K
IZ1 36 9 620MA
RZ1 36 9 10
IZ2 32 31 620MA
RZ2 32 31 10
R1 9 31 1K
R2 28 9 21K
R3 28 19 4.8K
R4 32 36 2.4K
R5 33 36 10
R6 26 19 2K
R7 25 36 1.5K
R8 20 36 120K
R9 11 3 60K
R10 6 8 60K
R11 34 5 3K
R12 8 9 10K
R13 22 36 15K
R14 21 36 15K
R15 23 36 15K
R16 17 9 10K
R17 34 15 3K
R18 16 17 60K
R19 11 14 60K
R21 24 36 120K
RTEMP 35 29 330K
Q3 35 29 31 QNL
Q4 35 32 33 QNL
Q5 29 33 36 QNL
Q6 29 28 27 QNL
Q7 27 27 26 QNL
Q8 19 19 25 QNL
Q9 34 1 2 QNL
Q10 2 19 20 QNL
Q11 34 34 11 QNL
Q12 3 2 4 QNL
Q13 4 19 22 QNL
Q14 6 3 5 QPL
Q15 5 6 8 QNL
Q16 34 8 10 QNL
Q17 10 19 21 QNL
Q18 34 17 18 QNL
Q19 18 19 23 QNL
Q20 15 16 17 QNL
```

```
Q21 16 14 15 QPL
Q22 14 13 4 QNL
Q23 13 19 24 QNL
Q24 34 12 13 QNL
.OUTPUT V10 10 0 PRINT DC TRAN
.OUTPUT V18 18 0 PRINT MAG PHASE DC TRAN PLOT MAG PHASE DC TRAN
.MODEL QNL NPN(BF=80 RB=100 CCS=2PF TF=0.3NS TR=6NS CJE=3PF
+  CJC=2PF VA=50)
.MODEL QPL PNP(BF=10 RB=20 TF=1NS TR=20NS CJE=6PF CJC=4PF VA=50)
.END
```

```
RTLINV CKT - CASCADED RTL INVERTERS
*  DC VIN 0 5 0.05
*  TRAN 2NS 200NS
.TRAN 2NS 200NS
VCC 6 0 5.0
VIN 1 0 PULSE(0 5 2NS 2NS 2NS 80NS)
RB1 1 2 10K
RC1 6 3 1K
Q1 3 2 0 QND
RB2 3 4 10K
Q2 5 4 0 QND
RC2 6 5 1K
.OUTPUT V3 3 0 PRINT DC TRAN PLOT DC TRAN
.OUTPUT V5 5 0 PRINT DC TRAN
.MODEL QND NPN(BF=50 RB=70 RC=40 CCS=2PF TF=0.1NS TR=10NS
+  CJE=0.9PF CJC=1.5PF PC=0.85 VA=50)
.END
```

```
TTLINV CKT - 74 SERIES TTL INVERTER
*  DC VIN 0 5 0.05
*  TRAN 1NS 100NS
.TRAN 1NS 100NS
VCC 13 0 5.0
VIN 1 0 PULSE(0 3.5 1NS 1NS 1NS 40NS)
RS 1 2 50
Q1 4 3 2 QND
RB1 13 3 4K
Q2 5 4 6 QND
RC2 13 5 1.4K
RE2 6 0 1K
Q3 7 5 8 QND
RC3 13 7 100
D1 8 9 D1
Q4 9 6 0 QND
Q5 11 10 9 QND
RB5 13 10 4K
D2 11 12 D1
D3 12 0 D1
.OUTPUT V5 5 0 PRINT DC TRAN
.OUTPUT V9 9 0 PRINT DC TRAN PLOT DC TRAN
.MODEL D1 D(RS=40 TT=0.1NS CJO=0.9PF)
.MODEL QND NPN(BF=50 RB=70 RC=40 CCS=2PF TF=0.1NS TR=10NS
+  CJE=0.9PF CJC=1.5PF PC=0.85 VA=50)
.END
```

```
ECLGATE CKT - ECL STACKED LOGIC GATE
*   DC VIN -2 0 0.02
*   TRAN 0.2NS 20NS
.TRAN 0.2NS 20NS
VEE 15 0 -6.0
VIN 16 0 PULSE(-1.8 -0.8 1NS 1NS 1NS)
VGATE 17 0 PULSE(-0.8 -1.8 5NS 1NS 1NS 5NS)
RS1 16 1 50
Q1 2 1 3 QND
Q2 0 9 3 QND
RC 0 2 100
RS2 17 4 50
Q3 0 4 5 QND
R1 5 6 60
R2 6 15 820
Q4 3 6 7 QND
RE 7 15 280
Q5 0 12 7 QND
R5 0 8 100
Q6 0 8 9 QND
R3 9 15 2K
D1 8 10 D1
R6 10 11 60
Q7 0 11 12 QND
R4 12 15 2K
D2 11 13 D1
R7 13 15 720
Q8 0 2 14 QND
RL 14 15 560
.OUTPUT V6 6 0 PRINT DC TRAN
.OUTPUT V14 14 0 PRINT DC TRAN PLOT DC TRAN
.MODEL D1 D(RS=40 TT=0.1NS CJO=0.9PF)
.MODEL QND NPN(BF=50 RB=70 RC=40 CCS=2PF TF=0.1NS TR=10NS
+   CJE=0.9PF CJC=1.5PF PC=0.85 VA=50)
.END
```

```
MECLIII CKT - MOTOROLA MECL III ECL GATE
*  DC VIN -2.0 0 0.02
*  TRAN 0.2NS 20NS
.TRAN 0.2NS 20NS
VEE 22 0 -6.0
VIN 1 0 PULSE(-0.8 -1.8 0.2NS 0.2NS 0.2NS 10NS)
RS 1 2 50
Q1 4 2 6 QND
Q2 4 3 6 QND
Q3 5 7 6 QND
Q4 0 8 7 QND
D1 8 9 D1
D2 9 10 D1
RP1 3 22 50K
RC1 0 4 100
RC2 0 5 112
RE 6 22 380
R1 7 22 2K
R2 0 8 350
R3 10 22 1958
Q5 0 5 11 QND
Q6 0 4 12 QND
RP2 11 22 560
RP3 12 22 560
Q7 13 12 15 QND
Q8 14 16 15 QND
RE2 15 22 380
RC3 0 13 100
RC4 0 14 112
Q9 0 17 16 QND
R4 16 22 2K
R5 0 17 350
D3 17 18 D1
D4 18 19 D1
R6 19 22 1958
Q10 0 14 20 QND
Q11 0 13 21 QND
RP4 20 22 560
RP5 21 22 560
.OUTPUT V12 12 0 PRINT DC TRAN PLOT DC TRAN
.OUTPUT V21 21 0 PRINT DC TRAN
.MODEL D1 D(RS=40 TT=0.1NS CJO=0.9PF)
.MODEL QND NPN(BF=50 RB=70 RC=40 CCS=2PF TF=0.1NS TR=10NS
+  CJE=0.9PF CJC=1.5PF PC=0.85 VA=50)
.END
```

```
SBDGATE CKT - SCHOTTKY-BARRIER TTL INVERTER
*  DC VIN 0 5 0.05
*  TRAN 2NS 200NS
.TRAN 2NS 200NS
VCC 23 0 5.0
VLOAD 26 0 5.0
VIN 1 0 PULSE(0.2 3.6 2NS 2NS 2NS 50NS)
RS 1 2 50
RB1 23 3 15K
RB2 26 17 15K
RC1 4 5 60
RC2 6 9 30
RC3 16 15 10
RC4 18 19 60
RE1 7 8 600
RE2 20 21 600
RL1 23 10 8.75K
RL2 26 25 8K
RK 23 12 1K
RS2 24 15 50
Q1 4 3 2 QND
Q2 6 5 7 QND
Q3 16 7 0 QND
Q4 18 17 24 QND
Q5 22 19 20 QND
Q6 20 20 0 QND
QL2 25 25 22 QND
QE 12 10 13 QND
DC1 3 4 D2
DC2 5 6 D2
DC3 7 16 D2
DC4 17 18 D2
DC5 19 22 D2
DE1 8 0 D2
DE2 21 0 D2
D1 13 14 D2
D12 14 28 D2
D2 28 15 D2
DL 10 9 D2
.OUTPUT VOUT 15 0 PRINT DC TRAN PLOT DC TRAN
.OUTPUT IPWR VCC PRINT DC TRAN
.MODEL D2 D(RS=15 CJO=0.2PF IS=5E-10 PHI=0.6)
.MODEL QND NPN(BF=50 RB=70 RC=40 CCS=2PF TF=0.1NS TR=10NS
+  CJE=0.9PF CJC=1.5PF PC=0.85 VA=50)
.END
```

```
CCSOR CKT - CONSTANT CURRENT SOURCE
VEE 7 0 -12
VBIAS 3 0 -6.0
Q1 2 3 4 Q1
Q2 2 4 5 Q1
Q3 1 6 5 Q1
Q4 1 8 6 Q1
Q5 10 1 9 Q1
Q6 10 9 8 Q1
R1 2 0 2K
R2 1 0 2K
R3 5 7 2K
R4 8 7 2K
R5 10 0 107
.MODEL Q1 NPN(BF=49.5 BR=0.5 IS=9.802E-16)
.END
```

```
OSC CKT - 1KHZ OSCILLATOR
VCC 2 0 5.6
Q1 2 1 8 Q1
Q2 3 6 5 Q1
Q3 2 10 12 Q1
Q4 11 3 7 Q1
Q5 10 11 13 Q1
Q6 2 10 9 Q1
Q7 3 8 4 Q1
R1 2 3 12K
R2 4 5 300
R3 4 0 1.5K
R4 10 1 98.603K
R5 2 11 7.5K
R6 7 0 1K
R7 12 6 5K
R8 6 0 10K
R9 2 10 1.5K
R10 13 0 240
R11 9 0 150
.MODEL Q1 NPN(BF=60 BR=0.205 IS=1.21E-15)
.END
```

```
UA733 CKT - UA 733 VIDEO PREAMPLIFIER
VCC 11 0 8
VEE 9 0 -8
Q1 3 1 4 Q1
Q2 14 2 13 Q1
Q3 17 14 16 Q1
Q4 18 3 16 Q1
Q5 11 18 19 Q1
Q6 11 17 22 Q1
Q7 6 7 8 Q1
Q8 7 7 10 Q1
Q9 16 7 15 Q1
Q10 19 7 20 Q1
Q11 22 7 21 Q1
R1 1 0 51
R2 2 0 51
R3 11 3 2.4K
R4 11 14 2.4K
R5 4 5 50
R6 13 12 50
R7 5 6 590
R8 12 6 590
R9 11 7 10K
R10 11 17 1.1K
R11 11 18 1.1K
R12 3 19 7K
R13 14 22 7K
R14 8 9 300
R15 10 9 1.4K
R16 15 9 300
R17 20 9 400
R18 21 9 400
.MODEL Q1 NPN(BF=100 BR=2 IS=9.901E-16)
.END
```

```
CFFLOP CKT - SATURATING COMPLEMENTARY FLIP-FLOP
VCC1 1 0 5.0
VCC2 8 0 10.0
VEE1 10 0 -5.0
VEE2 9 0 -10.0
Q1 3 2 1 QP1
Q2 6 7 1 QP1
Q3 6 5 10 QN1
Q4 3 4 10 QN1
R1 8 2 57.2K
R2 8 7 57.2K
R3 2 6 14.3K
R4 7 3 14.3K
R5 6 4 14.3K
R6 3 5 14.3K
R7 4 9 57.2K
R8 5 9 57.2K
R9 3 0 1K
.MODEL QN1 NPN(BF=10 BR=1 IS=9.1E-15)
.MODEL QP1 PNP(BF=10 BR=1 IS=9.1E-15)
.END
```

```
74TTL CKT - SERIES 74 TTL INVERTER
VIN 1 0 1.30
VCC 13 0 5.0
RS 1 2 50
Q1 4 3 2 QC
Q2 5 4 6 QA
Q3 7 5 8 QA
Q4 9 6 0 QB
Q5 11 10 9 QC
D1 8 9 DA
D2 11 12 DA
D3 12 0 DA
RB1 13 3 4K
RC2 13 5 1.4K
RE2 6 0 1K
RC3 13 7 100
RB5 13 10 4K
.MODEL QA NPN(BF=20 BR=1 RB=70 RC=40 IS=1.0E-14 VA=50)
.MODEL QB NPN(BF=20 BR=0.2 RB=20 RC=12 IS=1.6E-14 VA=50)
.MODEL QC NPN(BF=20 BR=0.02 RB=500 RC=40 IS=1.0E-14 VA=50)
.MODEL DA D(RS=40 IS=1.0E-14)
.END
```

```
74STTL CKT - SERIES 74S TTL INVERTER
VIN 1 0 1.375
VCC 15 0 5.0
RS 1 2 50
Q1 4 3 2 QC
RB1 15 3 2.4K
RC2 15 5 800
Q2 5 4 6 QA
Q3 7 5 8 QA
Q4 7 8 9 QB
RC4 15 7 60
Q5 10 11 0 QA
RE2 6 11 250
RC5 6 10 500
Q6 9 6 0 QB
Q7 13 12 9 QC
RB7 15 12 2.4K
D1 13 14 DA
D2 14 0 DA
.MODEL QA NPN(BF=20 BR=1 RB=70 RC=40 IS=1.0E-14 VA=50)
.MODEL QB NPN(BF=20 BR=0.2 RB=20 RC=12 IS=1.6E-14 VA=50)
.MODEL QC NPN(BF=20 BR=0.02 RB=500 RC=40 IS=1.0E-14 VA=50)
.MODEL DA D(RS=40 IS=1.0E-14)
.END
```

```
74LTTL CKT - SERIES 74L TTL INVERTER
VIN 1 0 1.175
VCC 13 0 5.0
RS 1 2 50
Q1 4 3 2 QC
Q2 5 4 6 QA
Q3 7 5 8 QA
Q4 9 6 0 QA
Q5 11 10 9 QC
D1 8 9 DA
D2 11 12 DA
D3 12 0 DA
RB1 13 3 40K
RC2 13 5 20K
RE2 6 0 12K
RC3 13 7 500
RB2 13 10 40K
.MODEL QA NPN(BF=20 BR=1 RB=70 RC=40 IS=1.0E-14 VA=50)
.MODEL QC NPN(BF=20 BR=0.02 RB=500 RC=40 IS=1.0E-14 VA=50)
.MODEL DA D(RS=40 IS=1.0E-14)
.END
```

```
9200TTL CKT - SERIES 9200 TTL INVERTER
VIN 1 0 1.35
VCC 14 0 5.0
RS 1 2 50
Q1 4 3 2 QC
Q2 5 4 6 QA
Q3 7 5 8 QA
Q4 9 8 10 QB
Q5 10 6 0 QB
Q6 12 11 10 QC
D1 12 13 DA
D2 13 0 DA
RB1 14 3 4K
RC2 14 5 1.5K
RE2 6 0 1.25K
RC3 14 7 150
RB4 8 10 4K
RC4 14 9 80
RB6 14 11 4K
.MODEL QA NPN(BF=20 BR=1 RB=70 RC=40 IS=1.0E-14 VA=50)
.MODEL QB NPN(BF=20 BR=0.2 RB=20 RC=12 IS=1.6E-14 VA=50)
.MODEL QC NPN(BF=20 BR=0.02 RB=500 RC=40 IS=1.0E-14 VA=50)
.MODEL DA D(RS=40 IS=1.0E-14)
.END
```

```
RC CKT - SPLIT TIME-CONSTANT RC CIRCUIT
.TRAN 0.1MS 10MS
VIN 1 0 PULSE(0 1 0.1MS 0.1MS)
R1 1 2 1K
C1 2 0 1PF
R2 2 3 1K
C2 3 0 1UF
.OUTPUT V1 1 0 PRINT TRAN PLOT TRAN
.OUTPUT V2 2 0 PRINT TRAN PLOT TRAN
.OUTPUT V3 3 0 PRINT TRAN PLOT TRAN
.END
```

```
CHOKE CKT - FULL WAVE CHOKE INPUT
.TRAN 0.2MS 20MS
VIN1 1 0 SIN(0 100 50)
VIN2 2 0 SIN(0 -100 50)
D1 1 3 DIO
D2 2 3 DIO
R1 3 0 10K
L1 3 4 5.0
R2 4 0 10K
C1 4 0 2UF -15.2MA
.OUTPUT V1 1 0 PRINT TRAN
.OUTPUT V2 2 0 PRINT TRAN
.OUTPUT V3 3 0 PRINT TRAN PLOT TRAN
.OUTPUT V4 4 0 PRINT TRAN PLOT TRAN
.MODEL DIO D(IS=1E-14 CJC=10PF)
.END
```

```
ECL CKT - EMITTER COUPLED LOGIC INVERTER
.TRAN 0.2NS 20NS
VIN 1 0 PULSE(-1.0 -1.8 1NS 1NS 8NS 20NS)
VEE 8 0 -5.0
VREF 6 0 -1.4
Q1 3 2 4 QSTD
Q2 5 6 4 QSTD
Q3 0 5 7 QSTD
Q4 0 5 7 QSTD
RIN 1 2 50
RC1 0 3 120
RC2 0 5 135
RE 4 8 340
RTH1 7 8 125
RTH2 7 0 85
CLOAD 7 0 5PF
.OUTPUT V1 1 0 PRINT TRAN PLOT TRAN
.OUTPUT V3 3 0 PRINT TRAN PLOT TRAN
.OUTPUT V5 5 0 PRINT TRAN PLOT TRAN
.OUTPUT V7 7 0 PRINT TRAN PLOT TRAN
.OUTPUT IVIN VIN PRINT TRAN PLOT TRAN
.MODEL QSTD NPN(IS=1E-16 BF=50 BR=0.1 RB=50 RC=10 TF=0.12NS
+  TR=5NS CJE=0.4PF PE=0.8 ME=0.4 CJC=0.5PF PC=0.8 MC=0.333
+  CCS=1PF VA=50)
.END
```

```
SCHMITT CKT - ECL COMPATIBLE SCHMITT TRIGGER
.TRAN 10NS 1000NS
VIN 1 0 PULSE(-1.6 -1.2 10NS 400NS 400NS 100NS 1000NS)
VEE 8 0 -5.0
RIN 1 2 50
RC1 0 3 50
R1 3 5 185
R2 5 8 760
RC2 0 6 100
RE 4 8 260
RTH1 7 8 125
RTH2 7 0 85
CLOAD 7 0 5PF
Q1 3 2 4 QSTD OFF
Q2 6 5 4 QSTD
Q3 0 6 7 QSTD
Q4 0 6 7 QSTD
.OUTPUT V1 1 0 PRINT TRAN PLOT TRAN
.OUTPUT V3 3 0 PRINT TRAN PLOT TRAN
.OUTPUT V5 5 0 PRINT TRAN PLOT TRAN
.OUTPUT V6 6 0 PRINT TRAN PLOT TRAN
.MODEL QSTD NPN(IS=1E-16 BF=50 BR=0.1 RB=50 RC=10 TF=0.12NS
+  TR=5NS CJE=0.4PF PE=0.8 ME=0.4 CJC=0.5PF PC=0.8 MC=0.333
+  CCS=1PF VA=50)
.END
```

```
ASTABLE CKT - A SIMPLE ASTABLE MULTIVIBRATOR
.TRAN 0.1US 10US
VIN 5 0 PULSE(0 5 0 1US 1US 100US 100US)
VCC 6 0 5.0
RC1 6 1 1K
RC2 6 2 1K
RB1 6 3 30K
RB2 5 4 30K
C1 1 4 150PF
C2 2 3 150PF
Q1 1 3 0 QSTD
Q2 2 4 0 QSTD
.OUTPUT V1 1 0 PRINT TRAN PLOT TRAN
.OUTPUT V2 2 0 PRINT TRAN PLOT TRAN
.OUTPUT V3 3 0 PRINT TRAN PLOT TRAN
.OUTPUT V4 4 0 PRINT TRAN PLOT TRAN
.MODEL QSTD NPN(IS=1E-16 BF=50 BR=0.1 RB=50 RC=10 TF=0.12NS
+  TR=5NS CJE=0.4PF PE=0.8 ME=0.4 CJC=0.5PF PC=0.8 MC=0.333
+  CCS=1PF VA=50)
.END
```

APPENDIX 2

SPICE ELEMENT MODELS

SPICE2 contains built-in models for the twelve circuit elements that are listed in Table A2.1. The linear elements include ideal resistors, capacitors, inductors, coupled inductors, independent voltage and current sources, and the voltage-controlled current source. The nonlinear elements that have been implemented in SPICE include a nonlinear voltage-controlled current source as well as the four common semiconductor devices: diodes, bipolar junction transistors (BJT's), junction field-effect transistors (JFET's), and insulated-gate field effect transistors (IGFET's or MOSFET's).

A2.1. Linear Elements

The seven linear elements that are listed in Table A2.1 are modeled by ideal branch relations. In dc and transient analysis, the branch relations that are given in Fig. A2.1 specify the electrical characteristics of these elements. The phasor relations that are given in Fig. A2.2 are used in the small-signal ac analysis.

Capacitors and inductors may be assigned an initial condition to accomodate cases where the circuit is not in equilibrium at the beginning of the transient response. The initial condition for capacitors is the time-zero value of capacitor current. For inductors, the initial condition is the time-zero value of inductor voltage. If capacitor and inductor initial conditions are omitted, the equilibrium case of zero capacitor current and zero inductor voltage is assumed.

Coupling between inductors is included with a separate coupling element. The mutual inductance between two inductors is

Linear Elements

Resistor

Capacitor

Inductor

Coupled Inductors

Independent Voltage Source

Independent Current Source

Linear Voltage-Controlled Current Source

Nonlinear Elements

Nonlinear Voltage-Controlled Current Source

Diode

Bipolar Junction Transistor (BJT)

Junction Field-Effect Transistor (JFET)

Insulated-Gate Field-Effect Transistor (IGFET or MOSFET)

Table A2.1. SPICE Elements -

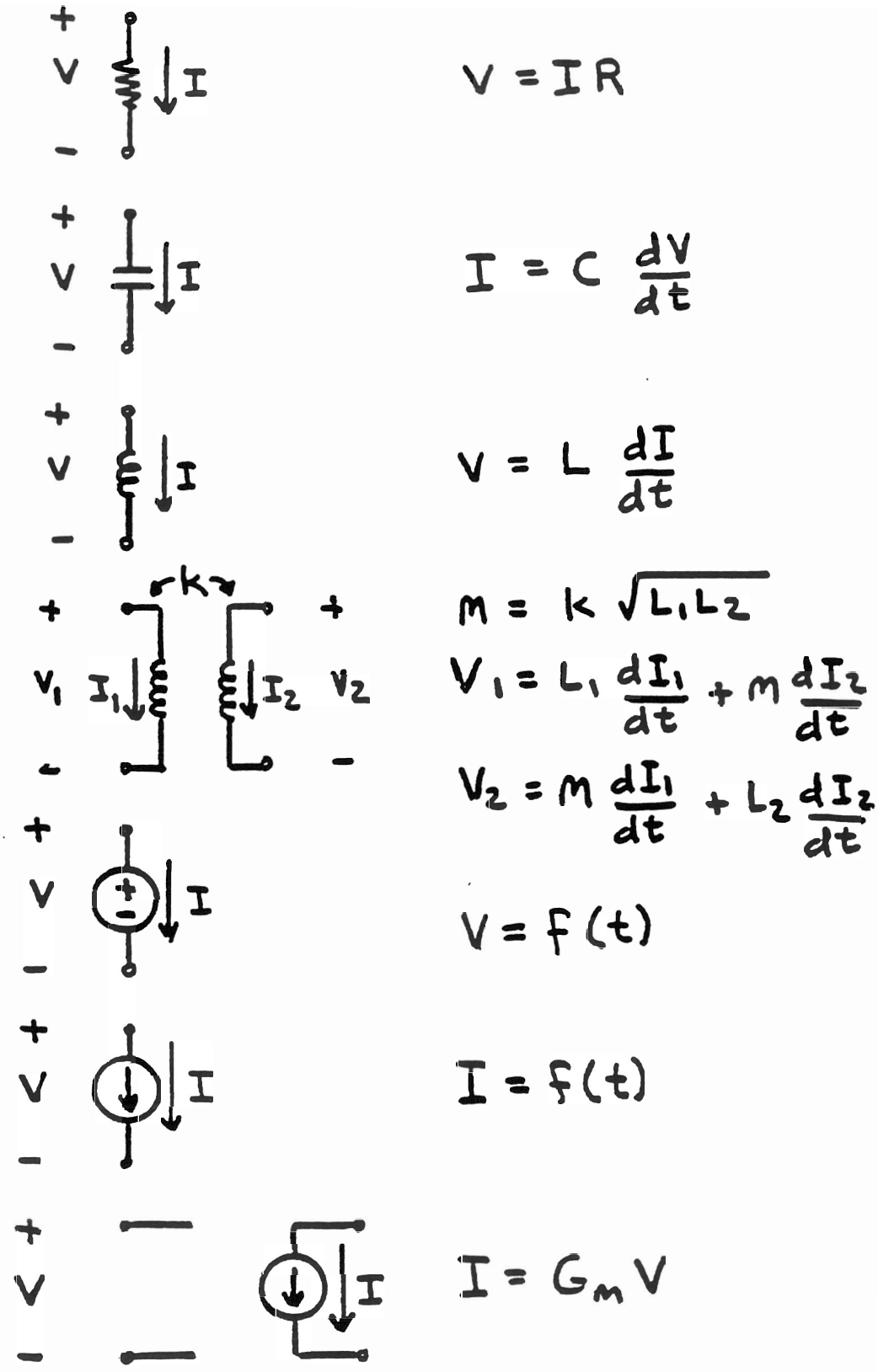


Fig. A2.1. SPICE Linear Branch Relations for DC and Transient Analysis.

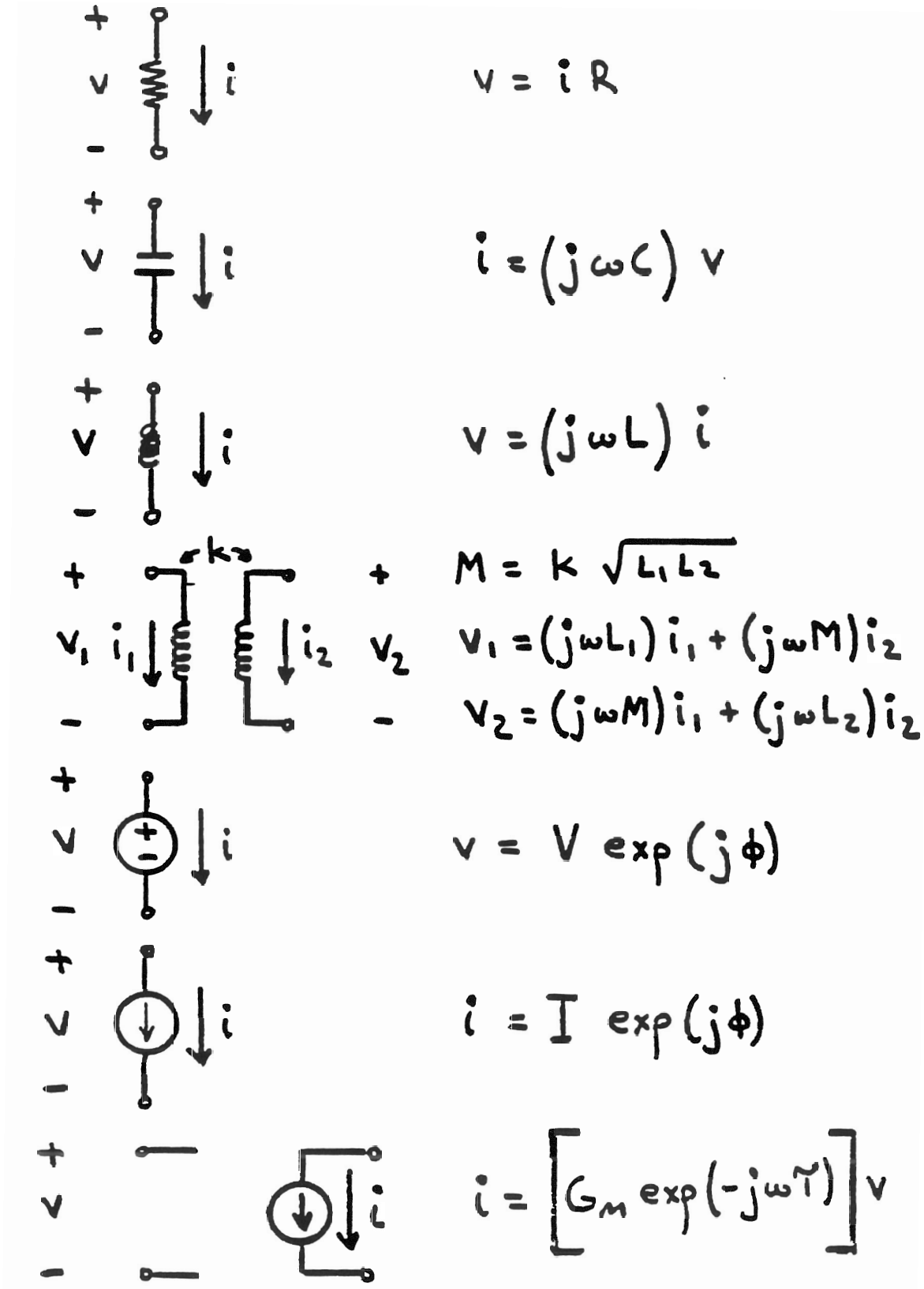


Fig. A2.2. SPICE Linear Branch Relations for AC Analysis.

specified by the coefficient of coupling, k which is defined by the equation

$$k = \frac{M}{\sqrt{L_1 L_2}} \qquad \text{(A2.1)}$$

where M is the mutual inductance between the inductances L_1 and L_2. The absolute value of k must be less than unity; that is, the case of ideal coupling is not allowed. If k is negative, the direction of coupling is reversed; this reversal is equivalent to reversing the polarity of either of the coupled inductors.

Independent voltage and current sources are input excitations to the circuit. The value of an independent source therefore depends upon the type of analysis. For a dc operating point analysis, all independent sources have a constant dc value. In a transient analysis, independent sources have either a constant value or a time-dependent value. For ac analysis, independent sources have a phasor value that is specified by a magnitude and relative phase.

The dc and transient values of an independent source are specified as a single value that may be time-dependent. If the source value is time-dependent, the time-zero value is used for dc analysis.

The ac value for the independent source is specified separately by values of magnitude and relative phase. If the circuit has only one ac input, as is the usual case, it is convenient to set the ac input source value to unity magnitude and zero phase. The value of every ac output variable then is equal to the transfer function of that output with respect to the input source.

The voltage-controlled current source is the only linear controlled source that is included in the SPICE program. A linear delay operator is included for this element to model "excess-phase" in ac analysis. The complex, frequency-dependent transconductance is determined by the equation

$$g_m(j\omega) = G_m \exp[-j\omega\tau] \qquad \text{(A2.2)}$$

where G_m is the zero-frequency transconductance, τ is the delay, and ω is the angular frequency ($\omega=2\pi f$). For dc and transient analysis, the delay is ignored.

A2.2 The Nonlinear Voltage-Controlled Current Source

The nonlinear voltage-controlled current source is included in SPICE to allow, to a limited extent, the analysis of circuits that contain arbitrary nonlinearities. The model for this element is identical to the linear voltage-controlled current source except that the transfer characteristic is defined by the polynomial

$$I = p_1 v + p_2 v^2 + p_3 v^3 + \ldots + p_{20} v^{20} \qquad \text{(A2.3)}$$

The order of the polynomial (A2.3) cannot exceed 20.

For ac analysis, each nonlinear voltage-controlled current source is modeled by a linear voltage-controlled current source with a value of transconductance that is determined by the equation

$$g_m = \left.\frac{\partial I}{\partial v}\right|_{op} = p_1 + 2p_2 v + 3p_3 v^2 + \ldots + 20p_{20} v^{19} \qquad \text{(A2.4)}$$

Nonlinear voltage-controlled current sources do not contain a delay operator.

A2.3. The Diode Model

The SPICE diode model is applicable to either junction diodes or Schottky-Barrier diodes (SBD's). This model is shown schematically in Fig. A2.3. The ohmic resistance of the diode is modeled by the linear resistor r_s.

The dc characteristics of the diode are modeled by the nonlinear current source I_D. The value of I_D is determined by the equation[1]

$$I_D = I_S[\exp(\frac{V_D}{nV_t}) - 1] \qquad \text{(A2.5)}$$

The parameters I_S, r_S, and n are estimated from dc measurements of the forward-biased diode characteristics. A graph of log (I_D) versus V_D is shown in Fig. A2.4. In the ideal region of operation, which for this graph corresponds to diode voltages of less than 600 mv, the diode characteristics is determined by the equation

$$\log_{10}(I_D) = \log_{10}(I_S) + \frac{2.3V_D}{nV_t} \qquad \text{(A2.6)}$$

The inverse slope of (A2.6) is approximately 60 mV per decade at room temperature if n is unity. The saturation current, I_S, is the extrapolated intercept current of (A2.6). Experimentally, I_S is estimated from several (V_D,I_D) points in the ideal region of operation.

---

[1] The parameter V_t is the thermal voltage and is equal to kT/q, where k is Boltzmann's constant, T is the absolute temperature, in degrees Kelvin, and q is the electronic charge. V_t is about 25.85 mv at room temperature.

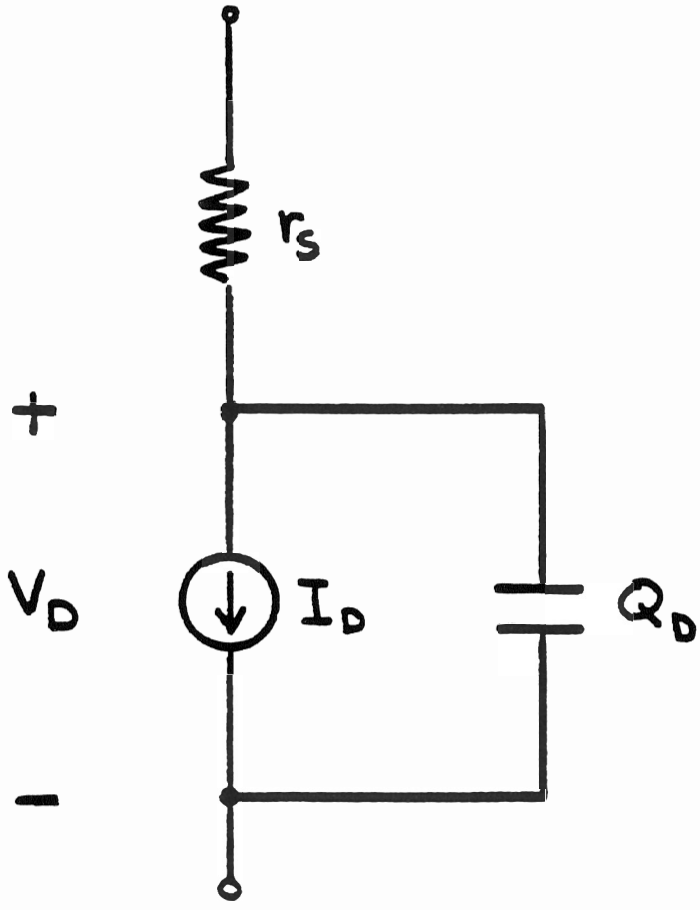


Fig. A2.3. The SPICE Diode Model.

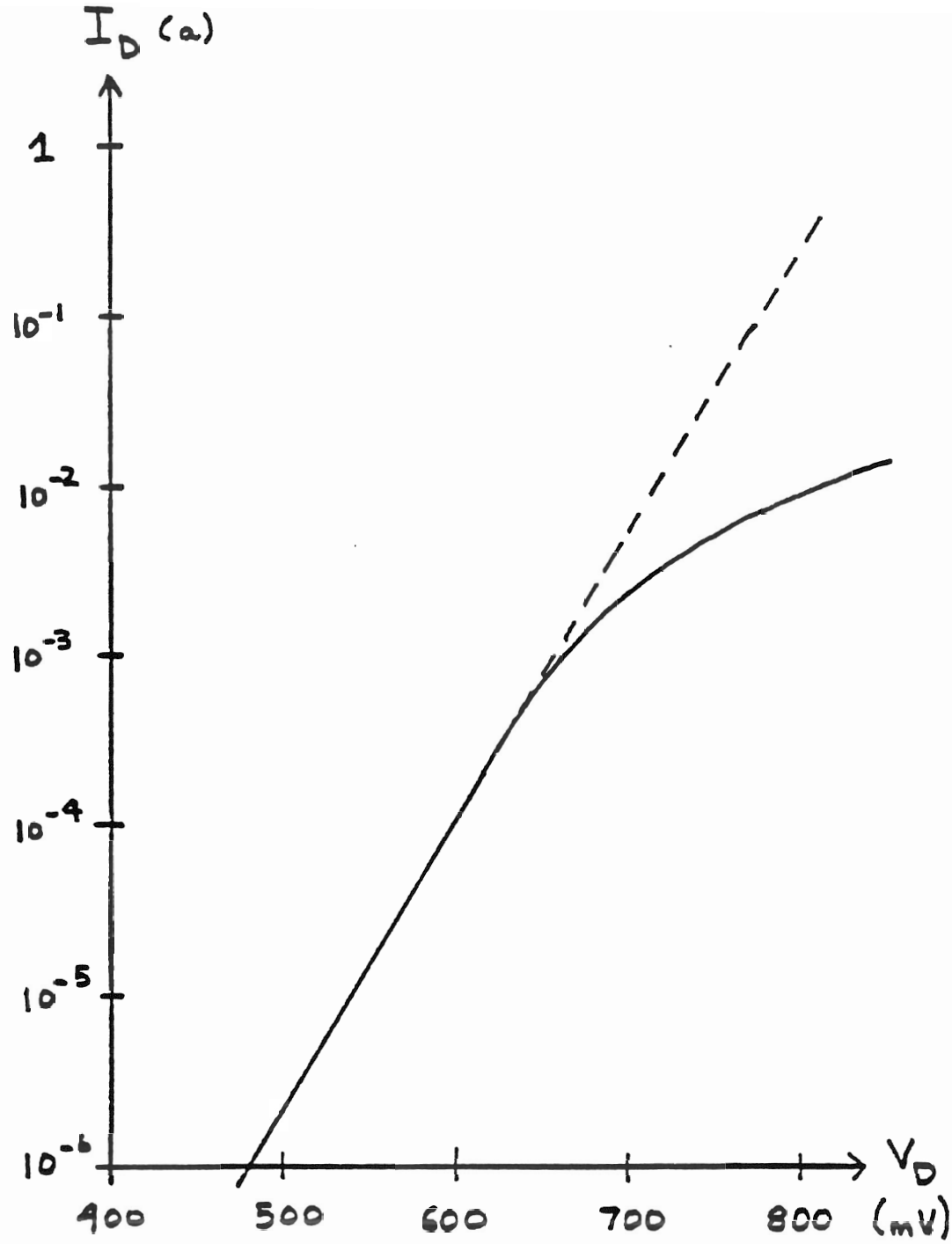


Fig. A2.4. Graph of $\log(I_D)$ <u>versus</u> V_D in the Forward-Bias Region.

The emission coefficient, n, is determined from the slope of the diode characteristic in the ideal region. In almost all cases, the emission coefficient is unity.

In practice, the diode current deviates from the ideal exponential characteristic given by (A2.5) at higher levels of bias. This deviation is due to the presence of ohmic resistance in the diode as well as high-level injection effects [77]. High-level injection is not included in the SPICE diode model. Therefore, the effects of both ohmic resistance and high-level injection are modeled by the ohmic resistance r_S. The value of r_S is determined from the deviation of the actual diode voltage from the ideal exponential characteristic at a specific current. In practice, r_S is estimated at several values of I_D and averaged since the value of r_S depends upon diode current.

A2.4. Diode Charge Storage

The charge-storage element Q_D that is shown in Fig. A2.3 models the charge storage of the diode. The charge Q_D is determined by the equation

$$Q_D = \tau_t I_S[\exp(\frac{V_D}{nV_t}) - 1] + C_{jo}\int_0^{V_D}[1 - \frac{V}{\phi_B}]^{-m} dv \qquad (A2.7)$$

This element equivalently can be defined by the capacitance relation

$$C_D = \frac{\partial Q_D}{\partial V_D} = \frac{\tau_t I_S}{nV_t}\exp(\frac{V_D}{nV_t}) + C_{jo}[1 - \frac{V_D}{\phi_B}]^{-m} \qquad (A2.8)$$

The charge-storage element Q_D models two distinct charge-storage mechanisms in a diode. Charge-storage in the junction depletion region

is modeled by the parameters C_{jo}, ϕ_B, and m. These parameters are obtained from capacitance bridge measurements at several values of reverse bias. Typically, ϕ_B is 0.7 - 0.8 volts and m is 0.3 - 0.5 for both junction diodes and SBD's.

Charge-storage due to injected minority carriers is modeled by the exponential term in (A2.7) and (A2.8). This mechanism of charge-storage is determined by the transit time τ_t. In practice, the parameter τ_t is estimated from pulsed delay-time measurements.

A2.5. The Small-Signal Diode Model

The small-signal linearized model for the diode is shown in Fig. A2.5. The conductance, g_D, is the derivative of the diode current, I_D, with respect to the diode voltage:

$$g_D = \left.\frac{\partial I_D}{\partial V_D}\right|_{op} = \left.\frac{I_S}{V_t}\exp\left(\frac{V_D}{nV_t}\right)\right|_{op} \qquad \text{(A2.9)}$$

The small-signal capacitance C_D is given in (A2.8). Naturally, g_D and C_D are evaluated at the dc operating point.

A2.6. The BJT Model

The first version of SPICE contains two separate models for the BJT [1]. The simpler model is based upon the familiar Ebers-Moll model [78] with the addition of basewidth modulation [79]. The more complex model for the BJT in SPICE1 is based upon the intergral charge model that was proposed by Gummel and Poon [80,81,82].

The two BJT models that are employed in SPICE1 are unified into a single BJT model in SPICE2. The SPICE2 BJT model is illustrated

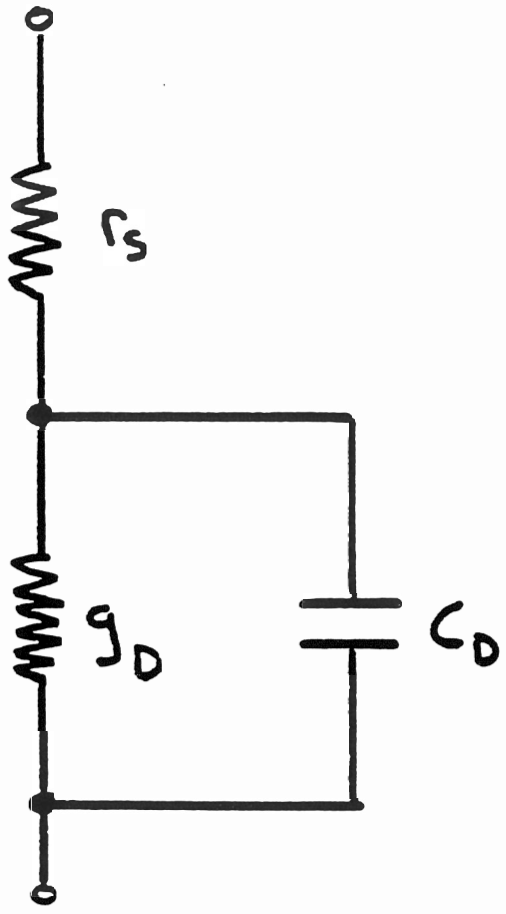


Fig. A2.5. The Linearized, Small-Signal Diode Model.

schematically in Fig. A2.6.[2] The ohmic resistances of the base, collector, and emitter regions of the BJT are modeled by the linear resistors r_b, r_c, and r_e.

The dc characteristics of the intrinsic BJT are determined by the nonlinear current sources I_C and I_B. The values of I_C and I_B are defined by the equations.

$$I_C = \frac{I_S}{Q_B}\left[\exp\left(\frac{V_{BE}}{V_t}\right) - \exp\left(\frac{V_{BC}}{V_t}\right)\right] - \frac{I_S}{B_R}\left[\exp\left(\frac{V_{BC}}{V_t}\right) - 1\right] - C_4 I_S\left[\exp\left(\frac{V_{BC}}{n_{CL}V_t}\right) - 1\right] \quad \text{(A2.10)}$$

$$I_B = \frac{I_S}{B_F}\left[\exp\left(\frac{V_{BE}}{V_t}\right) - 1\right] + C_2 I_S\left[\exp\left(\frac{V_{BE}}{n_{EL}V_t}\right) - 1\right] + \frac{I_S}{B_R}\left[\exp\left(\frac{V_{BC}}{V_t}\right) - 1\right] + C_4 I_S\left[\exp\left(\frac{V_{BC}}{n_{CL}V_t}\right) - 1\right] \quad \text{(A2.11)}$$

The base charge, Q_B, is the total majority-carrier charge in the base region divided by the zero-bias majority-carrier charge. A simple representation of Q_B [82,83] yields the following definition of Q_B:

$$Q_B = 1 + \frac{1}{V_A}\int_0^{V_{BE}} \frac{dV}{\left[1 - \frac{V}{\phi e}\right]^{m_e}} + \frac{1}{V_B}\int_0^{V_{BC}} \frac{dV}{\left[1 - \frac{V}{\phi c}\right]^{m_c}} - \frac{I_S}{Q_B I_K}\left[\exp\left(\frac{V_{BE}}{V_t}\right) - 1\right] + \frac{I_S}{Q_B I_{KR}}\left[\exp\left(\frac{V_{BC}}{V_t}\right) - 1\right] \quad \text{(A2.12)}$$

[2] The schematic in Fig. A2.6 is for an npn device. For a pnp transistor, the polarity of V_{BE}, V_{BC} and V_{CE} must be reversed, and the polarity of the current sources I_B and I_C must be reversed. All of the equations that follow are for an npn device.

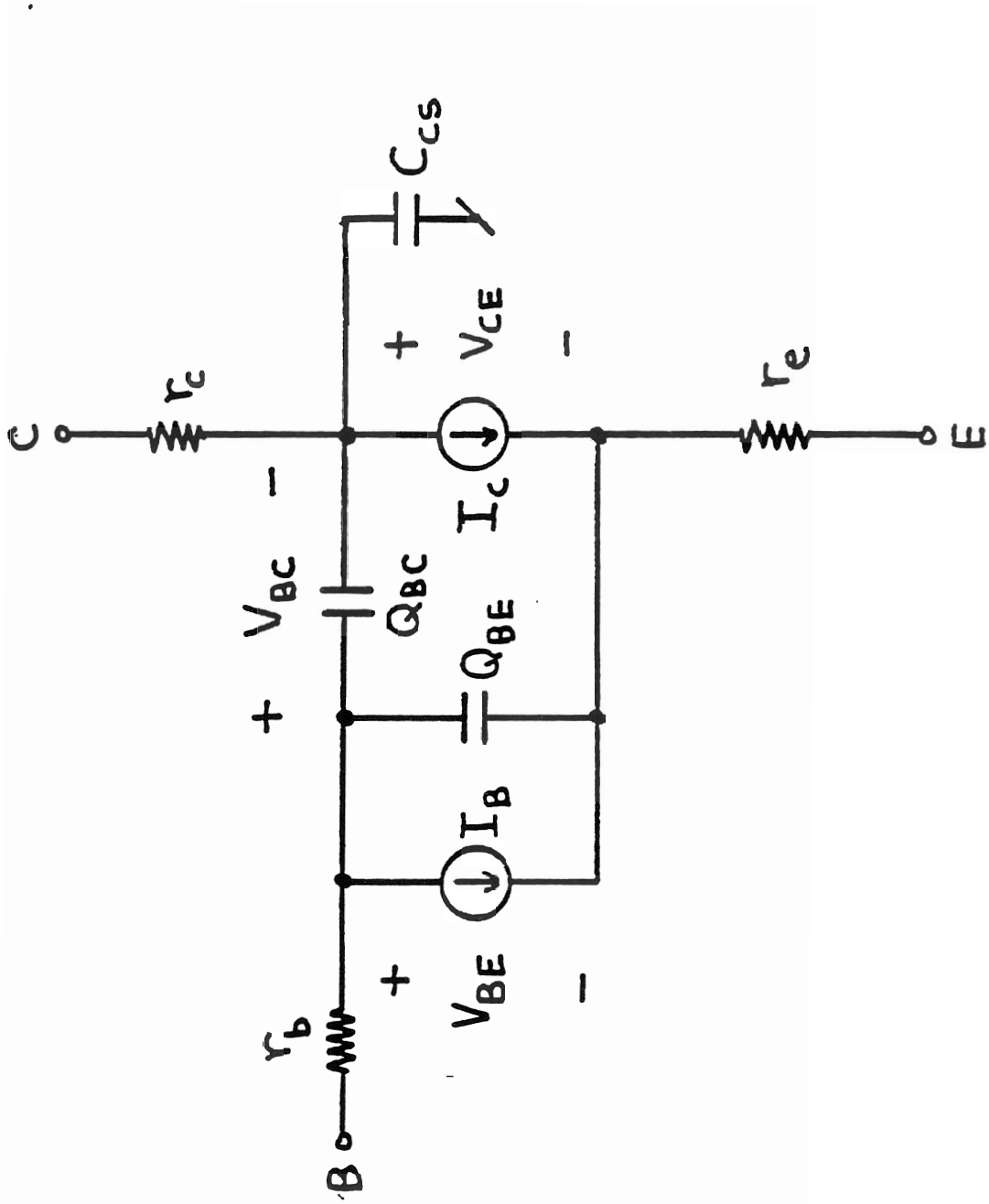


Fig. A2.6. The SPICE BJT Model.

At zero-bias, (A2.12) yields the result that Q_B is unity. The integral terms in (A2.12) model charge-storage in the emitter and collector depletion regions, whereas the exponential terms in (A2.12) model majority-carrier excess charge in the base region. This excess majority-carrier charge equals the injected minority-carrier charge because of charge neutrality.

Equation (A2.12) may be restated as the simpler equation

$$Q_B = Q_1 + \frac{Q_2}{Q_B} \qquad \text{(A2.13)}$$

A direct solution of (A2.13) for Q_B yields the result

$$Q_B = \frac{Q_1}{2} + \frac{1}{2}\sqrt{Q_1^2+4Q_2} \qquad \text{(A2.14)}$$

In SPICE2, (A2.14) is simplified further to yield the approximate expression

$$Q_B = \frac{Q_1}{2}\left[1 + \sqrt{1+4Q_2}\right] \qquad \text{(A2.15)}$$

Depletion-layer charge-storage in the base is accounted for by the Q_1 component of base charge. By definition, Q_1 is unity at zero-bias; this statement reflects the fact that the physical basewidth and the electrical basewidth are equal at zero bias. If both junctions are forward-biased, the electrical basewidth is larger than the physical basewidth, and Q_1 is larger than unity. Conversely, if both junctions are reverse-biased, then Q_1 is less than unity since the electrical basewidth is less than the physical basewidth.

In SPICE2, Q_1 is approximated by the equation

$$Q_1 = \left[1 - \frac{V_{BC}}{V_A} - \frac{V_{BE}}{V_B}\right]^{-1} \qquad \text{(A2.16)}$$

Hence, the relative significance of bias upon basewidth, and therefore base charge, is determined by the two parameters V_A and V_B. These parameters are referred to as the forward and reverse Early voltages, respectively.

The Q_2 component of base charge accounts for the excess majority carrier base charge that results from injected minority carriers. This charge, is defined by the equation

$$Q_2 = \frac{I_S}{I_K}\left[\exp\left(\frac{V_{BE}}{V_t}\right) - 1\right] + \frac{I_S}{I_{KR}}\left[\exp\left(\frac{V_{BC}}{V_t}\right) - 1\right] \qquad \text{(A2.17)}$$

Excess majority-carrier base charge clearly is insignificant until the injected minority-carrier charge is comparable to the zero-bias majority carrier charge; that is, until high-level injection is reached. The parameters I_K and I_{KR}, therefore, determine the bias conditions at which high-level injection is encountered. These two parameters are termed the forward and reverse knee currents, respectively.

In summary, the dc characteristics of the BJT are determined by the eleven model parameters: I_S, B_F, B_R, C_2, n_{EL}, C_4, n_{CL}, V_A, V_B, I_K, and I_{KR}. The simple Ebers-Moll model [78] is obtained from (A2.10-A2.11) if the nonideal gain coefficients, C_2 and C_4, are zero and V_A, V_B, I_K and I_{KR} are infinite.[3]

The saturation current, I_S, is a fundamental parameter that is related directly to the zero-bias majority-carrier profile in the base [84]. The parameter I_S is the extrapolated intercept current of

[3]Since V_A, V_B, I_K, and I_{KR} cannot be zero, a zero value for these parameters is interpreted as an infinite value in the SPICE progarm.

the graph of log (I_C) *versus* V_{BE} in the forward region, or of the graph of log (I_E) *versus* V_{BC} in the reverse region. Experimentally, I_S usually is estimated from several (I_C,V_{BE}) points in the region of operation where I_C nearly obeys the ideal exponential characteristic.

The forward current gain, β_F, and the reverse current gain, β_R, are defined by the equations

$$\beta_F = \frac{I_C}{I_B} \quad V_{BE} > 0, \; V_{BC} < 0 \tag{A2.18}$$

$$\beta_R = \frac{I_E}{I_B} \quad V_{BE} < 0, \; V_{BC} > 0 \tag{A2.19}$$

If C_2 and C_4 are zero, and V_A, V_B, I_K, and I_{KR} are infinite, then β_F equals B_F and β_R equals B_R. For this case, neither β_F nor β_R depend upon the operating point.

The second-order effect of basewidth modulation [79] is modeled by the Q_1 constituent of Q_B. If V_A is finite, but V_B, I_K, and I_{KR} are infinite, (A2.10) yields the equation

$$I_C = I_S \exp(\frac{V_{BE}}{vt}) \; [1 - \frac{V_{BC}}{V_A}] \tag{A2.20}$$

for forward-active region operation. The small-signal output conductance, g_0, is determined by the equation

$$g_0 = \left.\frac{\partial I_C}{\partial V_{CE}}\right|_{op} \cong \frac{I_C}{V_A} \tag{A2.21}$$

Equation (A2.21) yields the result that g_0 is proportional to I_C. Because of the simplified representation of Q_1 in (A2.16), actual devices exhibit an output conductance that deviates slightly from (A2.21). However, this simplified model of basewidth modulation

is adequate for most circuit simulation problems.

The output conductance is estimated at a particular operating point either from curve-tracer measurements or from small-signal measurements. The forward Early voltage, V_A, then is determined from (A2.21). The inverse Early voltage, V_B, has an analogous role for the inverse region of operation. However, the effects of basewidth modulation in the inverse region usually have little bearing on circuit performance and the parameters V_B usually can be igonored.

The base current in a BJT consists of an ideal component which has a unity emission coefficient and a nonideal component which has an emission coefficient that typically ranges from 1.5 to 2. Nonideal base current results from depletion-layer or surface recombination [85]. The parameters C_2 and n_{EL} in (A2.11) determine the relative magnitude of the nonideal base-current component in the forward region, and the parameters C_4 and n_{CL} perform the same function in the reverse region.

The effect of the nonideal base-current component is illustrated in Fig. A2.7. Here, $\log(I_C)$ and $\log(I_B)$ are plotted as a function of V_{BE} in the forward region. The effect of the series resistances r_b and r_e has been omitted for simplicity. The collector current therefore is ideal with an inverse slope of $2.3V_t$. The base current is the sum of an ideal component, with an inverse slope of $2.3V_t$, and a nonideal component, with an inverse slope of $2.3\, n_{EL}V_t$. For small values of V_{BE}, the nonideal component of base current dominates the total base current. For larger values of V_{BE}, the ideal component of base current dominates the total base current. The asymptotic

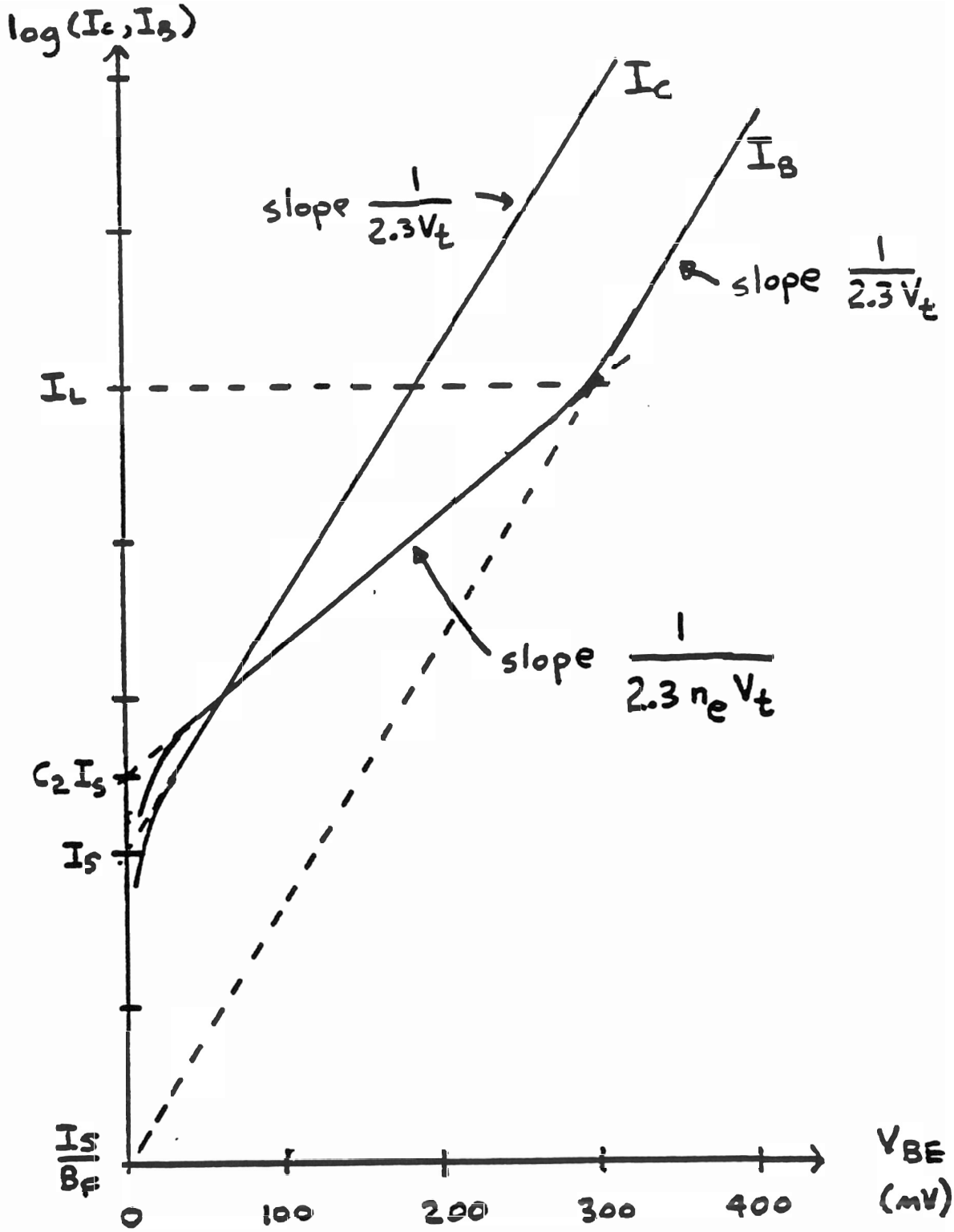


Fig. A2.7. Graph of $\log(I_C)$ and $\log(I_B)$ <u>versus</u> V_{BE} in the Forward Active Region.

breakpoint between nonideal and ideal base-current regions occurs when the collector current equals I_L, where I_L is given by the equation

$$I_L = I_S[C_2 B_F]^{\frac{n_{EL}}{n_{EL}-1}} \qquad \text{(A2.22)}$$

The salient effect of the nonideal component of base current is a drop in the common-emitter current-gain, β_F, at low levels of collector current. Asymptotically, β_F is constant at a value B_F if the collector current is larger than I_L. On the other hand, β_F decreases with a slope $1 - 1/n_{EL}$ as the collector current decreases below I_L.

The parameters C_2 and n_{EL} either are determined from the graph of $\log(I_B)$ versus V_{BE} in the forward region or are inferred from the graph of $\log(\beta_F)$ versus $\log(I_C)$ in the forward region. The parameters C_4 and n_{EL} have the identical roles of C_2 and n_{EL} in the reverse region of operation. Since the current dependence of the reverse region of operation. Since the current dependence of the reverse current gain, β_R, usually is of little importance, the parameters C_4 and n_{EL} usually can be ignored.

At larger values of collector current, high-level injection effects cause a marked deviation from the ideal Ebers-Moll characteristics [86]. The knee currents I_K and I_{KR} determine the levels of device bias where high-level injection becomes significant. If V_A and V_B are infinite, then the collector current is determined by the equation

$$I_C = \frac{I_S \exp(\frac{V_{BE}}{V_t})}{\frac{1}{2}\left[1 + \sqrt{1 + \frac{4I_S}{I_K}\exp(\frac{V_{BE}}{V_t})}\right]} \qquad \text{(A2.23)}$$

in the forward active region.

If I_C is considerably smaller than I_K, then (A2.23) simplifies to the ideal relation

$$I_C \cong I_S \exp(\frac{V_{BE}}{V_t}) \qquad \text{(A2.24)}$$

On the other hand, if I_C is considerably larger than I_K, then the collector current obeys the asymptotic relation

$$I_C \cong \sqrt{I_S I_K} \exp(\frac{V_{BE}}{2V_t}) \qquad \text{(A2.25)}$$

A comparison of (A2.24) and (A2.25) yields the result that the effective emission coefficient of the collector current characteristic changes from unity, in low-level injection, to 2, in high-level injection [86].

The inverse knee current, I_{KR}, has the analogous role of I_K in the inverse region of operation. Usually, high-level injection effects in inverse operation are not important, so the parameter I_{KR} usually can be ignored.

The dc characteristics of the BJT are illustrated in the graph of $\log(I_C)$ and $\log(I_B)$ versus V_{BE} for the forward region; these graphs are given in Fig. A2.8. The asymptotic behavior of the collector current, I_C, is ideal when the collector current is less than I_K. When I_C exceeds I_K, high-level injection causes an effective change in the collector current emission coefficient from unity to 2.

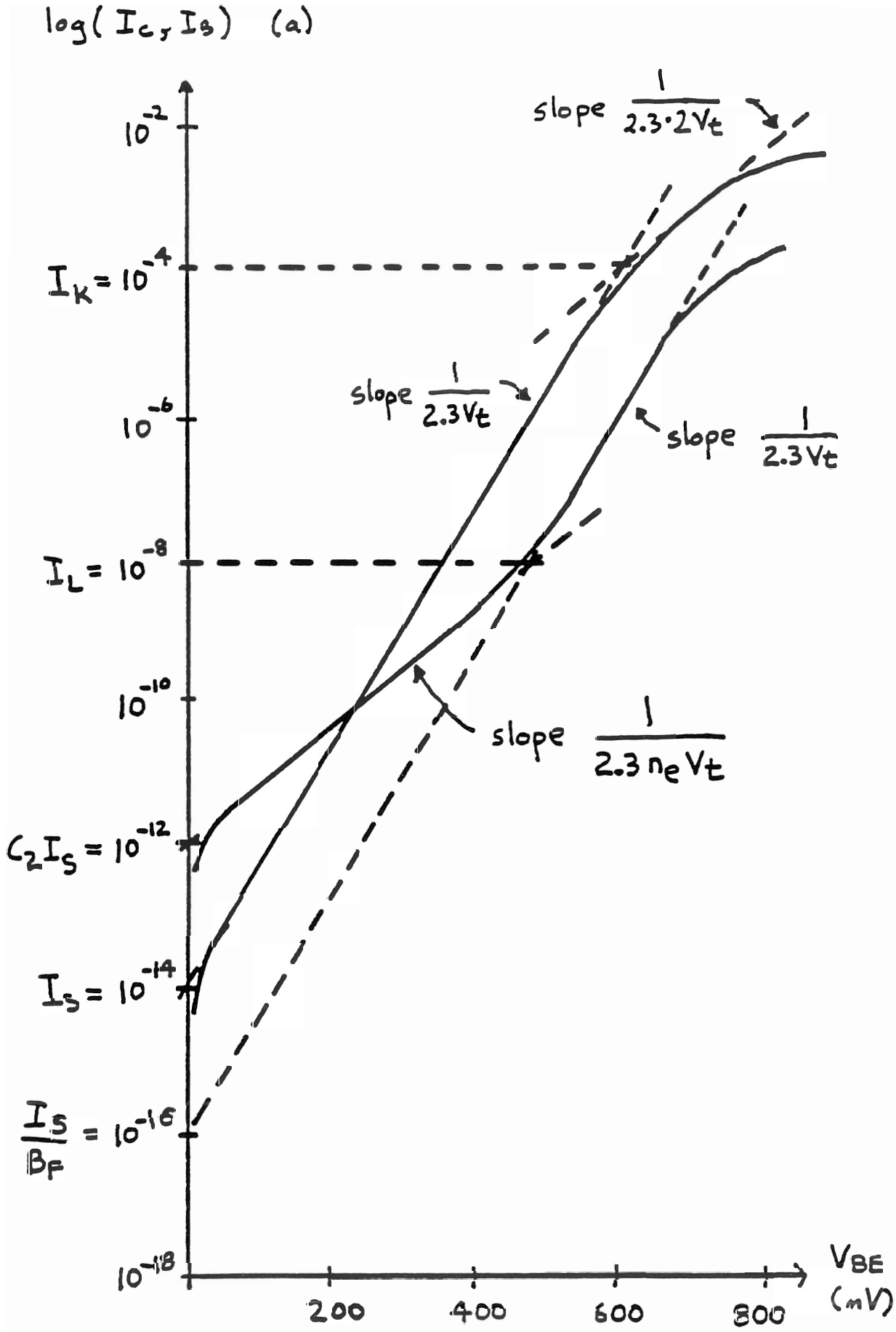


Fig. A2.8. Graph of $\log(I_C)$ and $\log(I_B)$ versus V_{BE} in the Forward Active Region.

The base current, I_B, is nonideal with an emission coefficient n_{EL} until the collector current exceeds the value of I_L. For values of I_C greater than I_L, the base current is ideal.

The "bending" of the I_B curve in Fig. A2.8 is due to the ohmic base resistance, r_b. The parameter r_b is estimated from the departure of the actual base current from the ideal exponential at a particular value of I_B. Usually, the emitter resistance, r_e, is ignored and the effects of r_e are lumped together with the effects of r_b. The collector resistance, r_c, is measured with the device in saturation. Since r_b, r_c, and r_e all depend upon bias currents, the values for these parameters should be averaged over a typical range of operating points. For applications where the value of r_b is critical to circuit performance, the value for r_b should be determined by more accurate methods [87].

A graph of $\log(\beta_F)$ versus $\log(I_C)$ is shown in Fig. A2.9. The current gain, β_F, is asymptotically constant for collector currents greater than I_L and less than I_K. For collector currents less than I_L, β_F decreases with a slope of $1 - 1/n_{EL}$. For collector greater than I_K, β_F decreases with a slope of -1. This dependence of β_F upon I_C is, of course, an asymptotic relation. For the cases where low-level effects and high-level effects overlap, as is true for most BJT's, some trial and error is required to obtain the correct parameters.

A2.7. BJT Charge Storage

Charge storage in the BJT is modeled by a linear collector-substrate capacitance, C_{CS}, and the two nonlinear charge elements

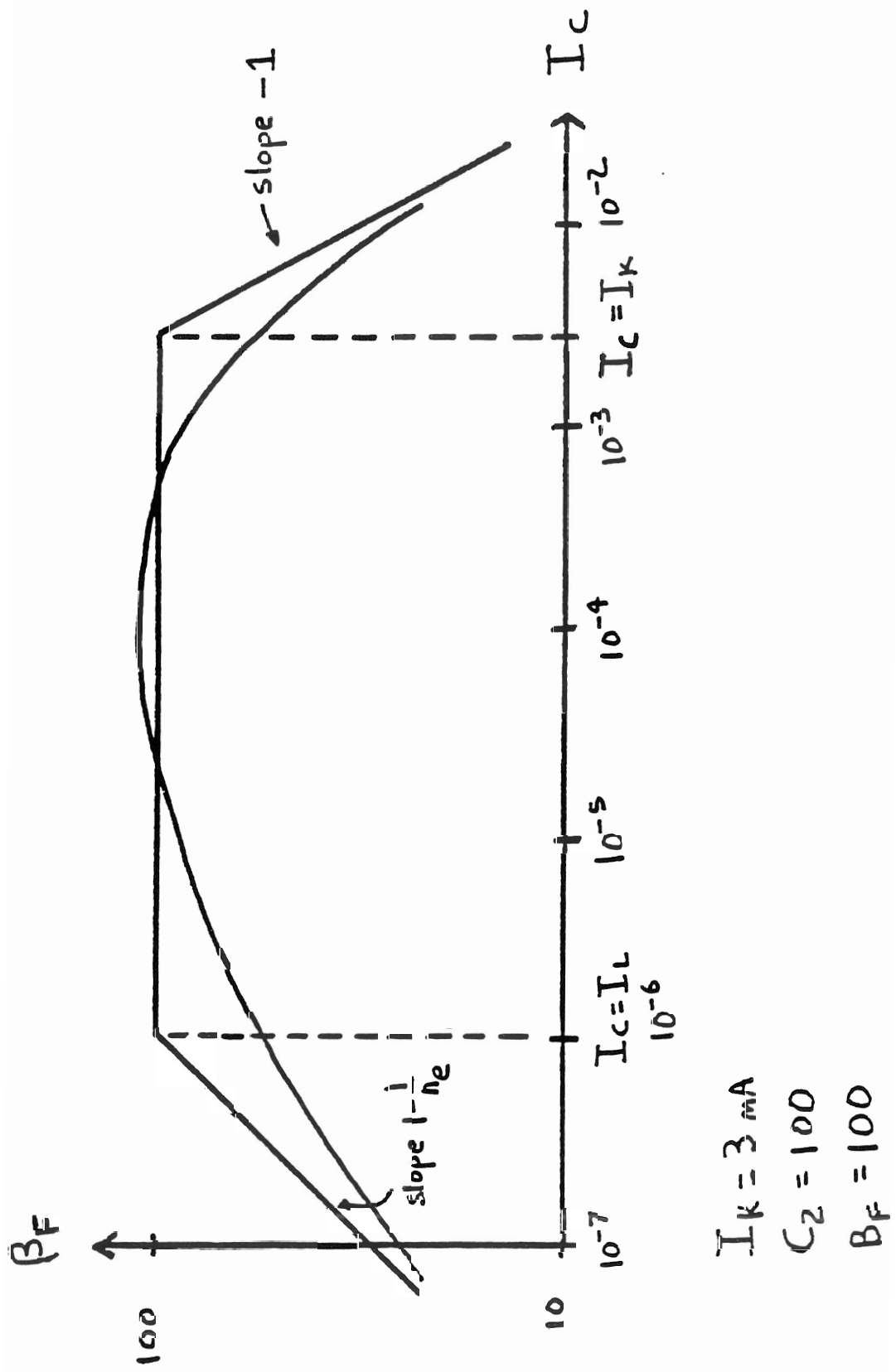


Fig. A2.9. Graph of $\log(\beta_F)$ versus $\log(I_C)$ in the Forward Active Region.

Q_{BE} and Q_{BC}. In reality, the collector-substrate capacitance also is voltage-dependent; however, a constant C_{CS} usually is a reasonable simplification. If C_{CS} is constant, then the BJT can be treated as a three-terminal device since C_{CS} can be returned directly to ground. The value of C_{CS} can be measured with a capacitance bridge or estimated from the substrate doping and mask dimensions.

The nonlinear charge elements Q_{BE} and Q_{BC} are determined by the equations

$$Q_{BE} = \tau_F I_S[\exp(\frac{V_{BE}}{V_t}) - 1] + C_{jeo}\int_0^{V_{BE}} [1 - \frac{V}{\phi_e}]^{-m_e} \qquad \text{(A2.26)}$$

$$Q_{BC} = \tau_R I_S[\exp(\frac{V_{BC}}{V_t}) - 1] + C_{jco}\int_0^{V_{BC}} [1 - \frac{V}{\phi_c}]^{-m_c} \qquad \text{(A2.27)}$$

These charge-storage elements equivalently can be represented by the voltage-dependent capacitance equations

$$C_{BE} = \frac{\tau_F I_S}{V_t} \exp(\frac{V_{BE}}{V_t}) + C_{jeo}[1 - \frac{V_{BE}}{\phi_e}]^{-m_e} \qquad \text{(A2.28)}$$

$$C_{BC} = \frac{\tau_R I_S}{V_t} \exp(\frac{V_{BC}}{V_t}) + C_{jco}[1 - \frac{V_{BC}}{\phi_c}]^{-m_c} \qquad \text{(A2.29)}$$

The charge elements Q_{BE} and Q_{BC} model the stored base charge in the depletion regions as well as injected minority carrier charge-storage.

Depletion-layer charge-storage is determined by six parameters: C_{jeo}, ϕ_e, m_e, C_{jco}, ϕ_c, and m_c. These parameters are determined from capacitance bridge measurements of both junctions at different values of reverse bias. Typically, the junction potentials, ϕ_e and

ϕ_c, are between 0.6 - 0.8 volts and the grading coefficients, m_e and m_c, are between 0.3 and 0.5 for silicon devices.

Injected minority-carrier base charge is modeled by the exponential terms in (A2.26-A2.29). This charge-storage mechanism is characterized by the model parameters τ_F and τ_R. The forward transit time, τ_F, usually is determined from measurements of the common-emitter cutoff frequency, f_T. Recovery-time measurements typically are used to estimate the reverse transit time, τ_R.

An interesting feature of the SPICE BJT model is that the injected base charge is modeled differently in (A2.26-A2.29) than it is in (A2.12). Consider, for example, the injected charge in the forward active region. The forward injected charge that is employed in (A2.26) is determined by the equation

$$Q_F = \tau_F I_S [\exp(\frac{V_{BE}}{V_t}) - 1] \qquad \text{(A2.30)}$$

However, the forward injected charge in (A2.12) is modeled by the equation

$$\hat{Q}_F = \frac{\hat{\tau}_F I_S}{Q_B} [\exp(\frac{V_{BE}}{vt}) - 1] \qquad \text{(A2.31)}$$

where $\hat{\tau}_F = Q_{BO}/I_K$ and Q_{BO} is the zero-bias base charge.

The difference between (A2.30) and (A2.31) is fundamental to the SPICE BJT model. The reasoning that leads to choosing (A2.30) for adoption in (A2.26) is based upon the effective transit-time that is defined by the equation

$$\tau_F(\text{eff}) = \left.\frac{\partial Q_B}{\partial I_C}\right|_{op} \qquad \text{(A2.32)}$$

Equation (A2.32) is valid, of course, only in the forward active region. If the extrinsic ohmic resistances of the BJT are ignored, then f_T is related to the effective transit-time by the equation

$$f_T = \frac{1}{2\pi\tau_F(\text{eff})} \tag{A2.33}$$

Physically, Q_{BE} and Q_{BC} should be chosen such that

$$Q_B = 1 + \frac{Q_{BE}+Q_{BC}}{Q_{BO}} \tag{A2.34}$$

Equation (A2.34) is a simple statement of charge neutrality. If Q_{BE} and Q_{BC} are chosen such that (A2.34) is satisfied, and if depletion-layer charge storage is ignored, then it can be shown that $\tau_F(\text{eff})$ is a constant equal to the model parameter τ_F.

The effective transit-time is not, however, independent of bias. Specifically, high-level injection effects tend to increase the value of effective transit-time [88,89], whereas increased depletion-layer widths tend to decrease the value of effective transit time [79]. Both of these effects are accounted for, to first order, by the assumption

$$\tau_F(\text{eff}) = \tau_F Q_B \tag{A2.35}$$

Substituting (A2.35) into (A2.31) yields (A2.30); hence, (A2.26-A2.29) implicitly contain the assumption that is defined by (A2.35). The SPICE BJT model, therefore, contains a first-order representation of transit-time dependence upon bias.

To illustrate the dependence of effective transit-time upon bias, consider the case of forward-active operation. If I_C is substantially larger than I_K, then Q_B is proportional to I_C. Therefore,

τ_F(eff) is proportional to I_C, and the SPICE BJT model predicts that f_T will be inversely proportional to I_C, excluding depletion-layer charge storage. On the other hand, the effective transit-time is constant and equal to τ_F if I_C is substantially less than I_K if depletion-layer charge storage and basewidth modulation effects are ignored.

The effective transit-time also depends upon collector bias (in forward-active region operation) because of basewidth modulation [79]. The dependence of effective transit-time upon depletion-layer width is accounted for in the SPICE BJT model by the Q_1 component of Q_B. If the BJT is biased in the forward-active region, then larger collector voltages result is smaller values of Q_B and, hence, smaller values of effective transit time.

The dependence of f_T upon bias is illustrated in Fig. A2.10. This figure shows the graph of $\log(f_T)$ versus $\log(I_C)$ for two values of V_{CE}($V_{CE1} > V_{CE2}$). The value of f_T is given in (A2.33), where τ_F(eff) is given by the approximate equation

$$\tau_F(\text{eff}) = \tau_F Q_B + \frac{V_t}{I_C}\left\{\frac{C_{jeo}}{\left[1 - \frac{V_{BE}}{\phi_e}\right]^{m_e}} + \frac{C_{jco}}{\left[1 - \frac{V_{BC}}{\phi_c}\right]^{m_c}}\right\} \qquad \text{(A2.36)}$$

At bias levels where I_C is substantially larger than I_K, f_T is inversely proportional to I_C, as predicted by simple theory [88]. At low levels of bias, f_T is directly proportional to I_C. This dependence is a direct result of the depletion capacitance terms in (A2.36). For a given value of I_C, f_T increases as V_{CE} is increased. This increase in f_T is due to the decrease in effective transit-time and the decrease in collector depletion capacitance.

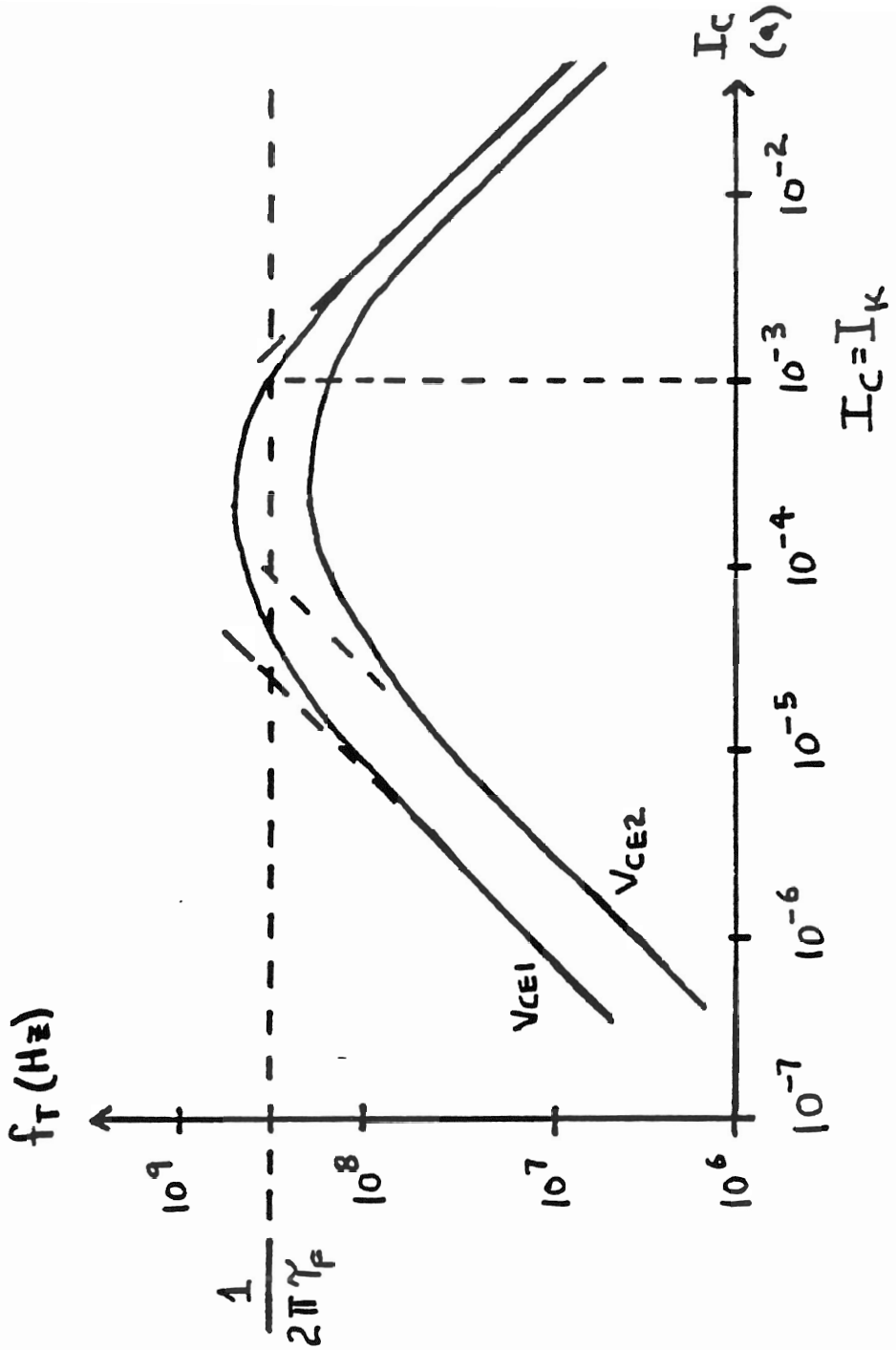


Fig. A2.10. Graph of $\log(f_T)$ versus $\log(I_C)$ in the Forward Active Region.

The parameter τ_F, therefore, is estimated from measurements of f_T at a bias level where $I_C << I_K$. Moreover, τ_F is determined from f_T measurements after depletion-layer capacitances and basewidth-modulation have been accounted for.

A2.8. The Small-Signal BJT Model

The linearized model for the BJT is the familiar hybrid-pi model [4] that is shown in Fig. A2.11. The four conductances g_π, g_μ, g_o, and g_m are related to the derivatives of (A2.10-A2.11) by the equations

$$g_\pi = \left.\frac{\partial I_B}{\partial V_{BE}}\right|_{op} \tag{A2.37}$$

$$g_\mu = \left.\frac{\partial I_B}{\partial V_{BC}}\right|_{op} \tag{A2.38}$$

$$g_o = -\left.\frac{\partial I_C}{\partial V_{BC}}\right|_{op} - \left.\frac{\partial I_B}{\partial V_{BC}}\right|_{op} \tag{A2.39}$$

$$g_m = \left.\frac{\partial I_C}{\partial V_{BE}}\right|_{op} + \left.\frac{\partial I_C}{\partial V_{BC}}\right|_{op} + \left.\frac{\partial I_B}{\partial V_{BC}}\right|_{op} \tag{A2.40}$$

The capacitance C_π is equal to the capacitance C_{BE} in (A2.28), and the capacitance C_μ is equal to the capacitance C_{BC} in (A2.29).

In addition to printing the hybrid-pi model parameters, SPICE also computes and prints the values of β_{DC}, β_{ac}, and f_T, which are determined by the equations

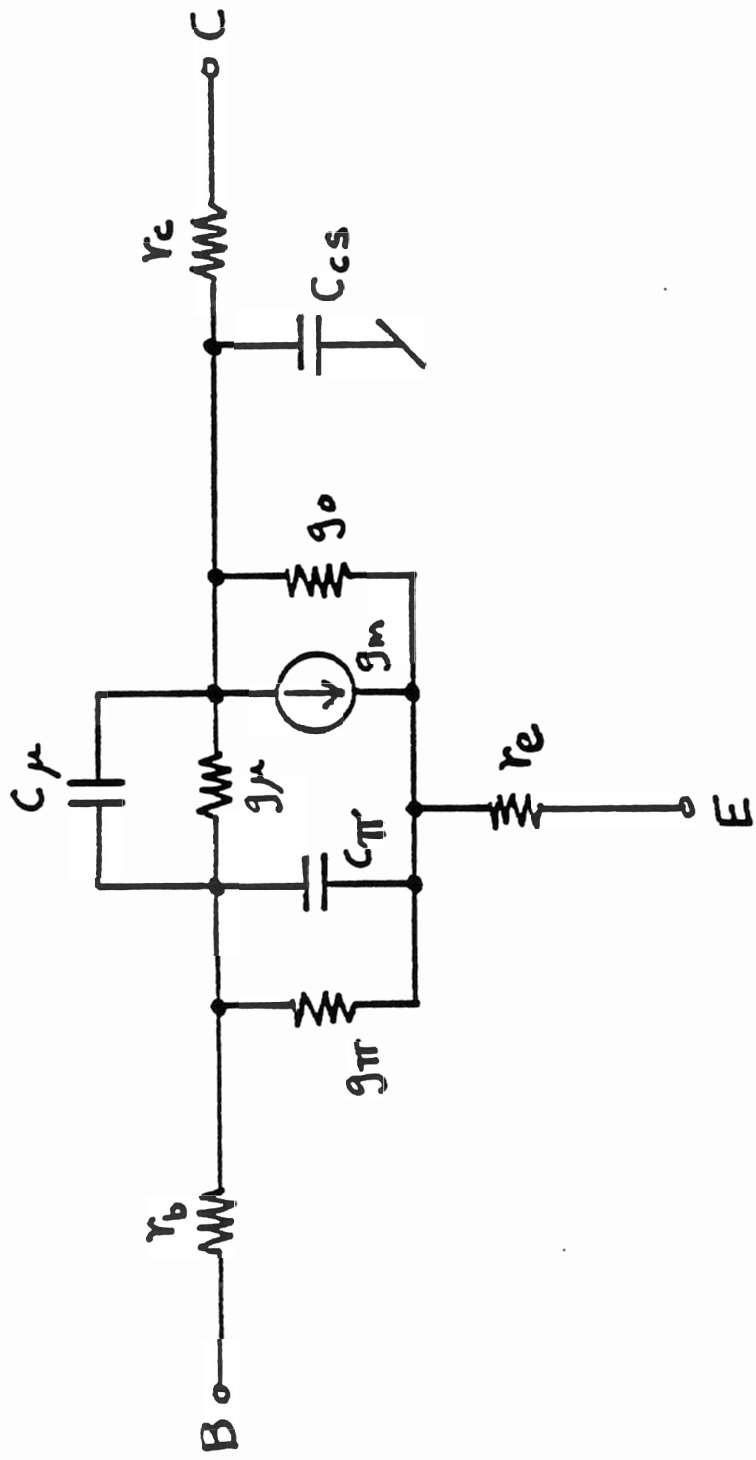


Fig. A2.11. The Linearized, Small-Signal BJT Model.

$$\beta_{DC} = \left.\frac{I_C}{I_B}\right|_{op} \tag{A2.41}$$

$$\beta_{ac} = \left.\frac{\partial I_C}{\partial I_B}\right|_{op} = g_m r_\pi \tag{A2.42}$$

$$f_T = \left.\frac{g_m}{2\pi(C_\pi+C_\mu)}\right|_{op} \tag{A2.43}$$

A2.9. The JFET Model

The SPICE model for an n-channel JFET is shown schematically in Fig. A2.12. For a p-channel device, the polarities of the terminal voltages V_{GD}, V_{GS}, and V_{DS} must be reversed, the direction of the two gate junctions must be reversed, and the direction of the nonlinear current source I_D must be reversed. The ohmic resistance of the drain and source regions of the JFET are modeled by the two linear resistors r_d and r_s.

The dc characteristics of the JFET are represented by the nonlinear current source I_D. The value of I_D is determined by the following equations:

$V_{DS} > 0$ (Forward region)

$$I_D = \begin{cases} 0 & V_{GS} - V_{TO} < 0 \\ \beta(V_{GS}-V_{TO})^2(1 + \lambda V_{DS}) & 0 < V_{GS} - V_{TO} < V_{DS} \\ \beta V_{DS}[2(V_{GS}-V_{TO}) - V_{DS}](1 + \lambda V_{DS}) & 0 < V_{DS} < V_{GS} - V_{TO} \end{cases} \tag{A2.44}$$

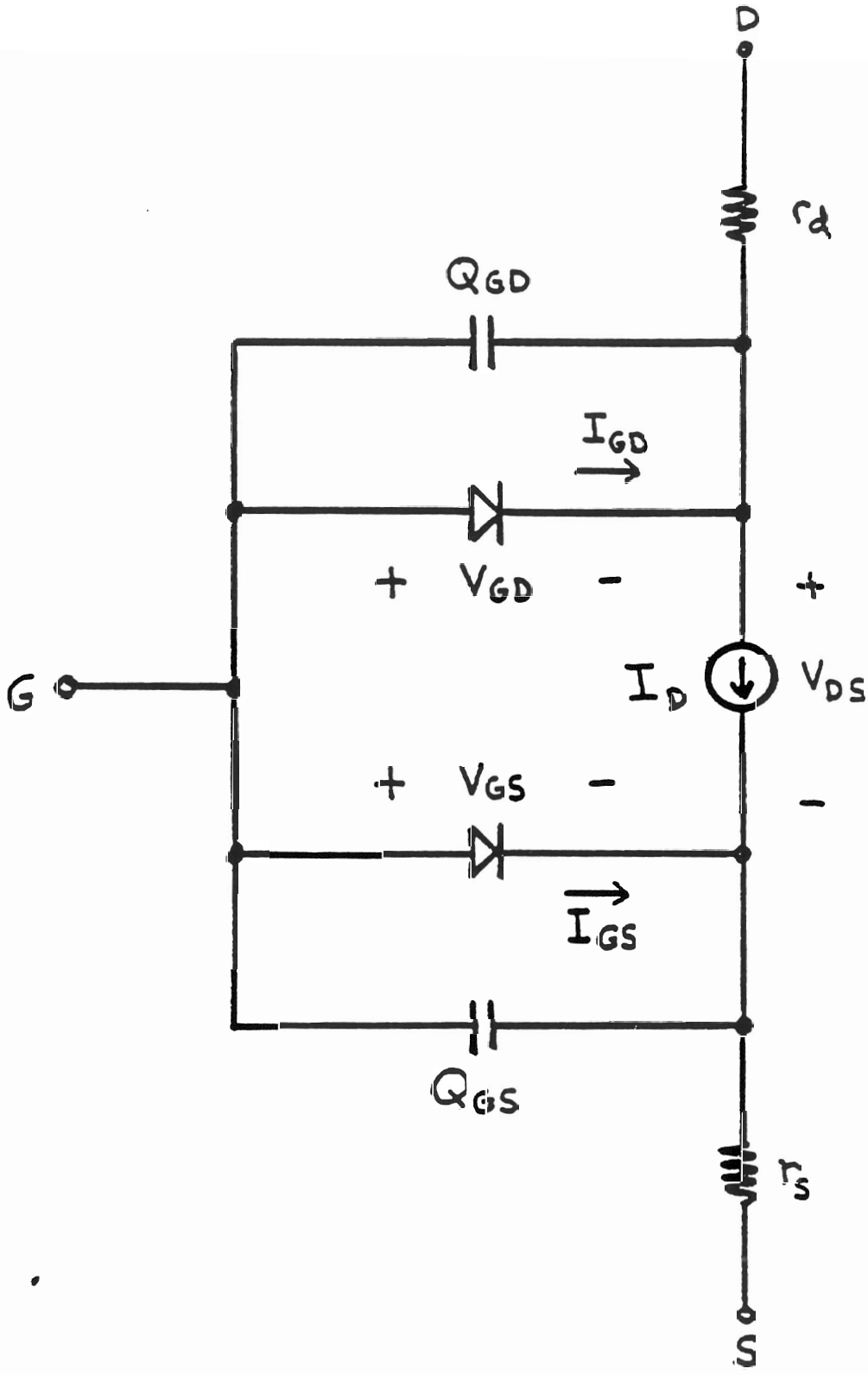


Fig. A2.12. The SPICE JFET Model.

$V_{DS} < 0$ (reverse region)

$$I_D = \begin{cases} 0 & V_{GD} - V_{TO} < 0 \\ -\beta(V_{GD}-V_{TO})^2 (1 - \lambda V_{DS}) & 0 < V_{GD} - V_{TO} < -V_{DS} \\ \beta V_{DS}[2(V_{GD}-V_{TO}) + V_{DS}] (1 - \lambda V_{DS}) & 0 < -V_{DS} < V_{GD} - V_{TO} \end{cases} \quad \text{(A2.45)}$$

In SPICE, the JFET drain current is modeled by a simple "square-law" characteristic that is determined by the parameters V_{TO} and β. The convention in SPICE is that V_{TO} is negative for all JFET's regardless of polarity. The parameters V_{TO} and β are usually determined from a graph of $\sqrt{I_D}$ versus V_{GS}. An example graph, for an n-channel device, is shown in Fig. A2.13. The parameter V_{TO} is the x-axis intercept of this graph, whereas the parameter β is the slope of the $\sqrt{I_D}$ versus V_{GS} characteristic. One standard figure of merit for a JFET is the drain current at zero V_{GS} bias. This parameter is related to V_{TO} and β by the equation

$$I_{DSS} = \beta V_{TO}^2 \quad \text{(A2.46)}$$

The parameter λ determines the effect of channel-length modulation on the JFET characteristics. The output conductance of the device, in the forward saturation region, is given by

$$g_{DS}(sat) = \beta\lambda(V_{GS}-V_{TO})^2 \cong \lambda I_D \quad \text{(A2.47)}$$

Hence, the saturation conductance of the JFET is directly proportional to drain current. The parameter λ therefore has the same role in the JFET characteristics that the parameter V_A has for the BJT.

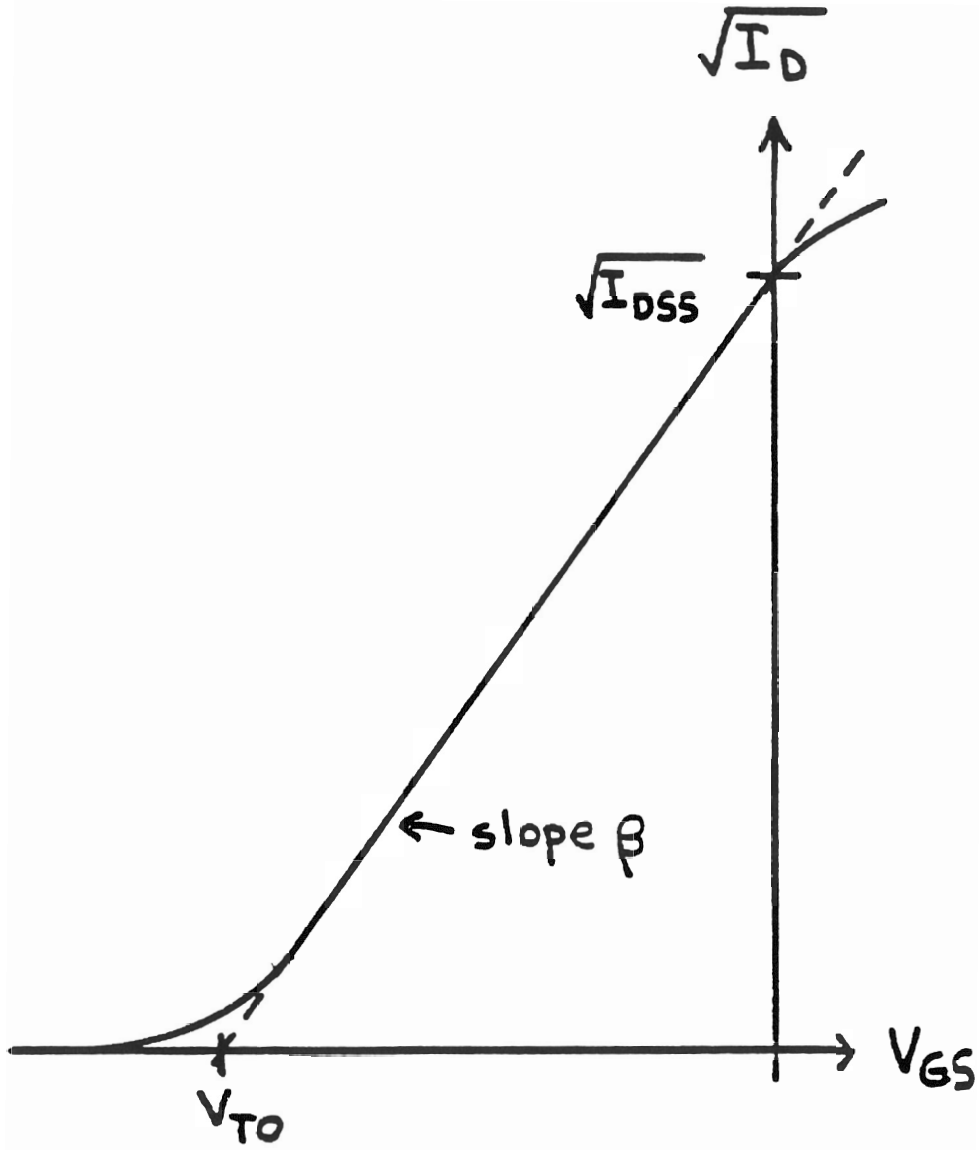


Fig. A2.13. Typical JFET Drain Current Characteristics.

The two diodes that are shown in Fig. A2.12 are modeled by the ideal diode equations

$$I_{GD} = I_S[\exp(\frac{V_{GD}}{V_t}) - 1] \tag{A2.48}$$

$$I_{GS} = I_S[\exp(\frac{V_{GS}}{V_t}) - 1] \tag{A2.49}$$

The saturation current, I_S, is a JFET model parameter. Since JFET's normally operated with both gate junction reverse-biased, I_S should be chosen to correspond to observed values of gate-junction leakage current.

A2.10 JFET Charge Storage

Charge storage in a JFET occurs in the two gate junctions. This charge storage is modeled by the nonlinear elements Q_{GS} and Q_{GD}. Since neither gate junction normally is forward-biased, the injected charge storage of these junctions is ignored. The elements Q_{GS} and Q_{GD} are defined by the equations

$$Q_{GS} = C_{GSO}\int_0^{V_{GS}} [1 - \frac{V}{\phi_B}]^{-1/2} dV \tag{A2.50}$$

$$Q_{GD} = C_{GDO}\int_0^{V_{GD}} [1 - \frac{V}{\phi_B}]^{-1/2} dV \tag{A2.51}$$

Alternatively, these two charges can be expressed as voltage-dependent capacitors with values determined by

$$C_{GS} = C_{GSO}[1 - \frac{V_{GS}}{\phi_B}]^{-1/2} \tag{A2.52}$$

$$C_{GD} = C_{GDO}\left[1 - \frac{V_{GD}}{\phi_B}\right]^{-1/2} \qquad \text{(A2.53)}$$

The values for C_{GSO}, C_{GDO}, and ϕ_B are determined from capacitance bridge measurements at different values of reverse bias.

A2.11. The Small-Signal JFET Model

The small-signal linearized model for the JFET is given in Fig. A2.14. The conductances g_{gs} and g_{gd} are the conductances of the two gate junctions. Since neither gate junction is forward-biased in normal operation, both g_{gs} and g_{gd} usually are very small. The transconductance, g_m, and the output conductance, g_{ds}, are defined by the equation

$$g_m = \left.\frac{\partial I_D}{\partial V_{GS}}\right|_{op} \qquad \text{(A2.54)}$$

$$g_{ds} = \left.\frac{\partial I_D}{\partial V_{DS}}\right|_{op} \qquad \text{(A2.55)}$$

The capacitances C_{GS} and C_{GD} are defined in (A2.52-A2.53).

A2.12. The MOSFET Model

The MOSFET model that is implemented in SPICE is illustrated in Fig. A2.15 for an n-channel MOSFET. This model is applicable for any insulated-gate FET (IGFET) including, of course, silicon-gate IGFET's. For a p-channel device, the polarities of the five terminal voltages (V_{GS}, V_{GD}, V_{DS}, V_{BS}, and V_{BD}), the two substrate junctions, and the nonlinear current source I_D are reversed. The substrate node is labeled B (for bulk or body) to avoid confusion

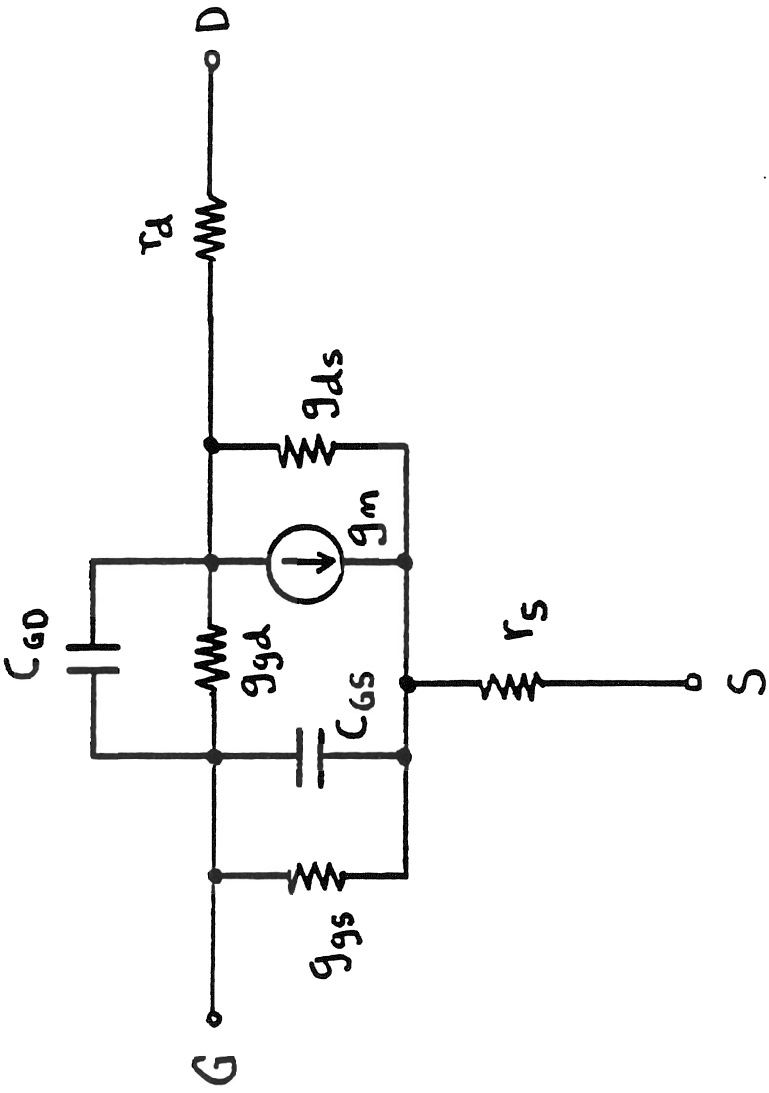


Fig. A2.14. The Linearized, Small-Signal JFET Model.

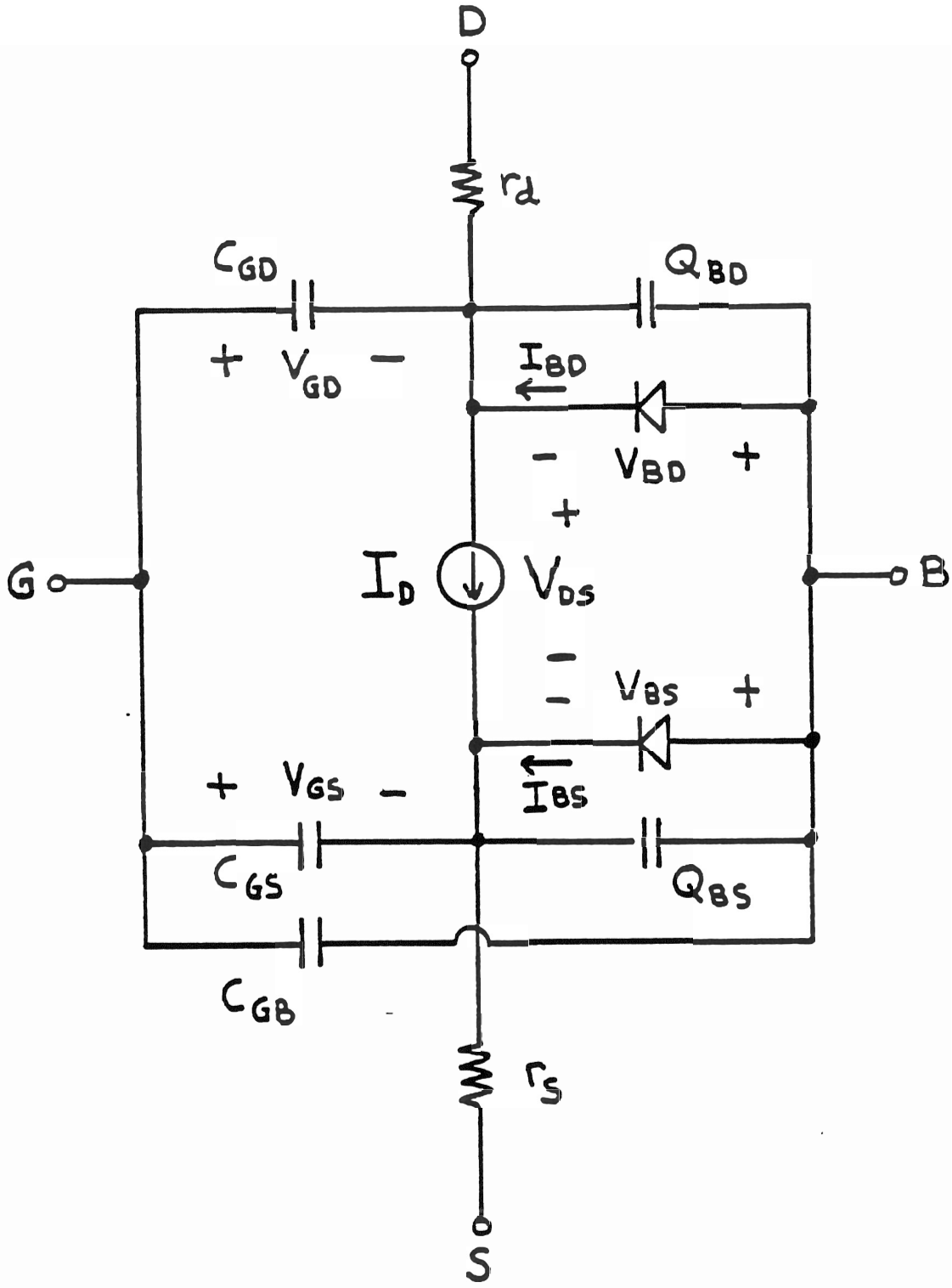


Fig. A2.15. The SPICE MOSFET model.

with the source node which is labeled S. The ohmic resistance of the drain and source regions of the device are modeled by the two linear resistances r_d and r_s.

The dc characteristics of the MOSFET are determined by the nonlinear current source I_D. In SPICE, the value of I_D is obtained from the equations proposed by Shichman and Hodges [90]. These equations are:

$V_{DS} > 0$ (forward region)

$$V_{TE} = V_{TO} + \gamma[\sqrt{\phi - V_{BS}} - \sqrt{\phi}] \tag{A2.56}$$

$$I_D = \begin{cases} 0 & V_{GS} - V_{TE} < 0 \\ \beta(V_{GS}-V_{TE})^2 (1+\lambda V_{DS}) & 0 < V_{GS} - V_{TE} < V_{DS} \\ \beta V_{DS}[2(V_{GS}-V_{TE}) - V_{DS}] (1+\lambda V_{DS}) & 0 < V_{DS} < V_{GS} - V_{TE} \end{cases} \tag{A2.57}$$

$V_{DS} < 0$ (reverse region)

$$V_{TE} = V_{TO} + \gamma[\sqrt{\phi - V_{BD}} - \sqrt{\phi}] \tag{A2.58}$$

$$I_D = \begin{cases} 0 & V_{GD} - V_{TE} < 0 \\ -\beta(V_{GD}-V_{TE})^2 (1-\lambda V_{DS}) & 0 < V_{GD} - V_{TE} < -V_{DS} \\ \beta V_{DS}[2(V_{GD}-V_{TE}) + V_{DS}] (1-\lambda V_{DS}) & 0 < -V_{DS} < V_{GD} - V_{TE} \end{cases} \tag{A2.59}$$

The Shichman-Hodges model provides a good first-order representation of the dc characteristics of an IGFET. Several more accurate and more complicated models have been proposed [91-94] since the paper by Shichman and Hodges.

The drain current equations are determined by the five model parameters V_{TO}, β, λ, γ, and ϕ. With the exception of the voltage dependence of the threshold voltage, V_{TE}, the MOSFET drain current is represented by the same "square-law" approximation that is used for the JFET. In SPICE, the threshold voltage, V_{TO}, is positive for enhancement-mode devices and negative for depletion-mode devices, regardless of the channel polarity. For the case of an n-channel device operating in the forward region with $V_{BS} = 0$, the parameter V_{TO} is the value of V_{GS} where the channel begins to conduct. For the complementary case of a p-channel device operating in the forward region with $V_{BS} = 0$, V_{TO} is the negative of the value of V_{GS} where the channel begins to conduct.

The parameter β is determined from a graph of $\sqrt{I_D}$ versus V_{GS} in the forward region for small values of V_{DS}. The parameter λ can be estimated from curve tracer measurements of I_D versus V_{DS} in the same manner as for BJT's and JFET's.

A graph of the effective threshold voltage of an n-channel device as a function of V_{BS} in the forward region is shown in Fig. A2.16. This graph shows that the effective threshold voltage increases as V_{BS} becomes more negative (more positive for p-channel devices). The dependence of V_{TE} upon substrate bias is determined by the parameter γ, which is estimated from a graph of $\sqrt{V_{TE}}$ versus V_{BS}. For typical IGFET's, γ is 0.5 - 1.5, and ϕ is 0.4 - 0.8 volts.

The two substrate junctions are modeled by the ideal diode equations

$$I_{BS} = I_S[\exp(\frac{V_{BS}}{V_t}) - 1] \qquad \text{(A2.60)}$$

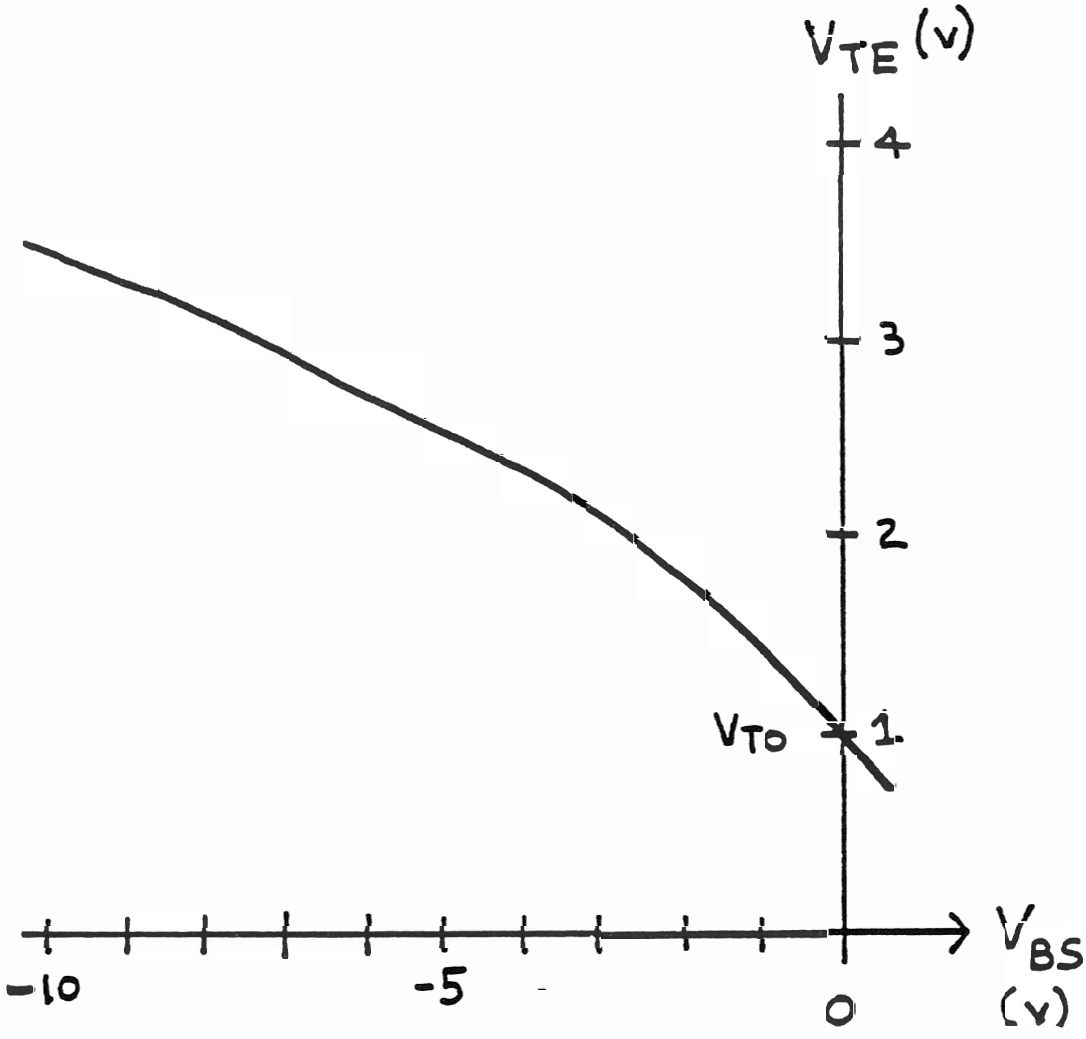


Fig. A2.16. Graph of Effective Threshold Voltage versus V_{BS} for a N-Channel Device.

$$I_{BD} = I_S[\exp(\frac{V_{BD}}{V_t}) - 1] \qquad \text{(A2.61)}$$

The saturation current, I_S, is a model parameter. Since neither substrate junction is normally forward-biased, the value of I_S should be chosen to model the leakage characteristics of the substrate junctions.

A2.13. MOSFET Charge Storage

Charge storage in the gate region of the MOSFET is modeled by the three constant capacitors C_{GS}, C_{GD}, and C_{GB}. In reality, these capacitors are voltage-dependent, so some care must be taken in assigning the values of C_{GS}, C_{GD}, and C_{GB}. One approximate method is to determine the total diode capacitance (from mask dimensions and oxide thickness) and to split this capacitance equally between C_{GS} and C_{GD}. This method is a good approximation for static MOS circuits [90]. In other instances, it may be more appropriate to divide the total oxide capacitance between all three capacitors.

The charge storage of the two substrate junctions is modeled by the nonlinear charges Q_{BD} and Q_{BS}. These charges are determined by the equations

$$Q_{BD} = C_{BDO}\int_0^{V_{BD}} [1 - \frac{V}{\phi_B}]^{-1/2}\, dV \qquad \text{(A2.62)}$$

$$Q_{BS} = C_{BSO}\int_0^{V_{BS}} [1 - \frac{V}{\phi_B}]^{-1/2}\, dV \qquad \text{(A2.63)}$$

Injected charge storage in the substrate junctions is omitted since the substrate junctions normally are reverse-biased. The two charge elements Q_{BD} and Q_{BS} can be expressed equivalently as the voltage dependent capacitances C_{BD} and C_{BS} whose values are determined by the equations

$$C_{BD} = C_{BDO}\left[1 - \frac{V_{BD}}{\phi_B}\right]^{-1/2} \tag{A2.64}$$

$$C_{BS} = C_{BSO}\left[1 - \frac{V_{BS}}{\phi_B}\right]^{-1/2} \tag{A2.65}$$

The values for C_{BDO}, C_{BSO}, and ϕ_B are determined either from capacitance bridge measurements or from mask dimensions and substrate doping information.

A2.14. The Small-Signal MOSFET Model

The linearized small-signal model for the MOSFET is shown in Fig. A2.17. The conductances g_{bd} and g_{bs} are the equivalent conductances of the two substrate junctions. Since these junctions are reverse-biased in normal operation, the conductances g_{bd} and g_{bs} typically are very small. The conductances g_m, g_{ds}, and g_{mbs} are given by the equations

$$g_m = \left.\frac{\partial I_D}{\partial V_{GS}}\right|_{op} \tag{2.66}$$

$$g_{ds} = \left.\frac{\partial I_D}{\partial V_{DS}}\right|_{op} \tag{A2.67}$$

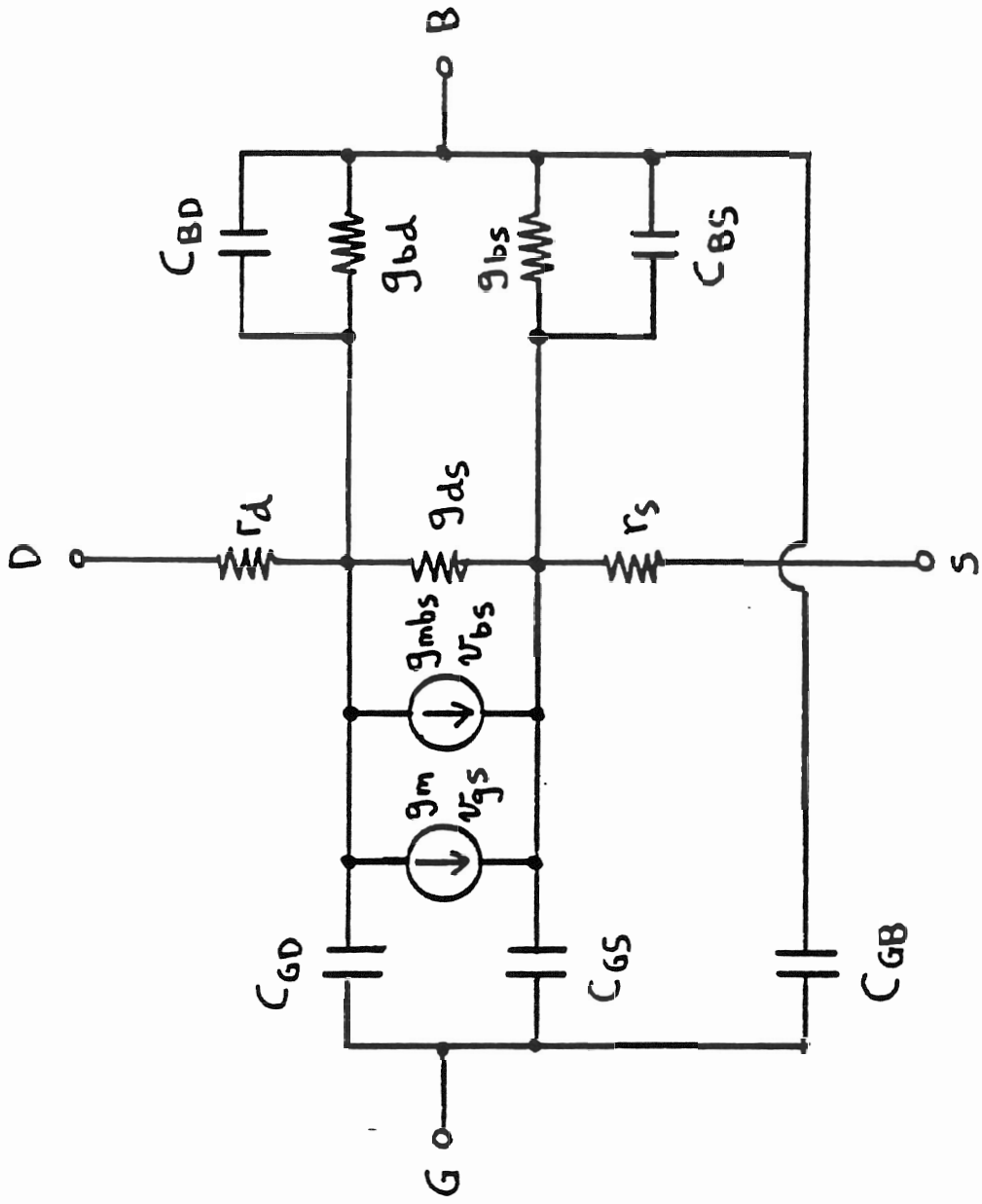


Fig. A2.17. The Linearized, Small-Signal MOSFET Model.

$$g_{mbs} = \left.\frac{\partial I_D}{\partial V_{BS}}\right|_{op} \qquad \text{(A2.68)}$$

The capacitances C_{BS} and C_{BD} are determined from (A2.64-A2.65)

A2.15. Noise Models

The ac analysis portion of SPICE contains the capability of evaluating the noise characteristics of an electronic circuit. Every resistor in the circuit generates thermal noise, and every semiconductor device in the circuit generates both shot and flicker noise in addition to the thermal noise that is generated by the ohmic resistances of the device.

The noise that is generated by a circuit element can be modeled as an electrical excitation to the small-signal circuit. Each noise source in the circuit is statistically uncorrelated to the other noise sources of the circuit. The contribution of each noise source in the circuit to the total noise at a specified output is determined separately. The total noise is then the RMS sum of the individual noise contributions.

Noise is specified with respect to a specific noise bandwidth and has the units volts/$\sqrt{\text{Hz}}$ or amps/$\sqrt{\text{Hz}}$. The output noise of a circuit is the RMS level of noise at the output divided by the square-root of the bandwidth. The equivalent input noise is the output noise divided by the transfer function of the output with respect to the specified input.

The thermal noise that is generated by a resistor is modeled by an independent current source i_{nR} in parallel with the resistor, as shown in Fig. A2.18. The value of i_{nR} is determined by the equation

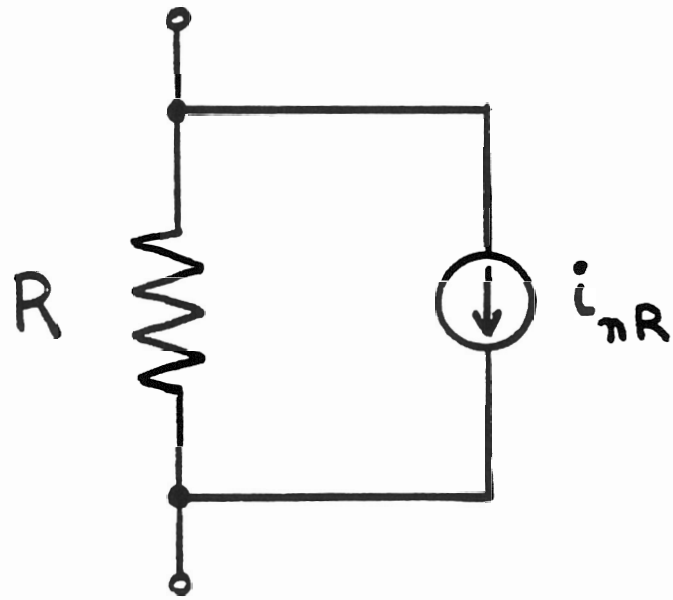


Fig. A2.18. The Noise Model for a Resistor.

$$i_{nR} = \sqrt{\frac{4kT}{R}} \qquad \text{(A2.69)}$$

where k is Boltzmann's constant, and T is the absolute temperature, in degrees Kelvin. The units of i_{nR} are a / $\sqrt{Hz}$.

The noise model for the diode is illustrated in Fig. A2.19. Thermal noise generation in the ohmic region of the diode is modeled by the current source i_{nr_s}, and the value of i_{nr_s} is determined by (A2.69). The shot and flicker noise of the diode are modeled by the current source i_{nD} which is defined by the equation

$$i_{nD} = \sqrt{2qI_D + \frac{K_f I_D^{a_f}}{f}} \qquad \text{(A2.70)}$$

The model parameters K_f and a_f are estimated from measurements of the diode noise at low frequency. For silicon diodes, typical values of for these parameters are $K_f = 10^{-16}$ and $a_f = 1$.

The noise model for the BJT is given in Fig. A2.20. Thermal noise generation in the three ohmic regions of the BJT is modeled by the current sources i_{nr_c}, i_{nr_b}, and i_{nr_e}; these three sources are determined by (A2.69). Shot noise and flicker noise are modeled by the two sources i_{nC} and i_{nB} which, in turn, are defined by

$$i_{nC} = \sqrt{2qI_c} \qquad \text{(A2.71)}$$

$$i_{nB} = \sqrt{2qI_B + \frac{K_f I_B^{a_f}}{f}} \qquad \text{(A2.72)}$$

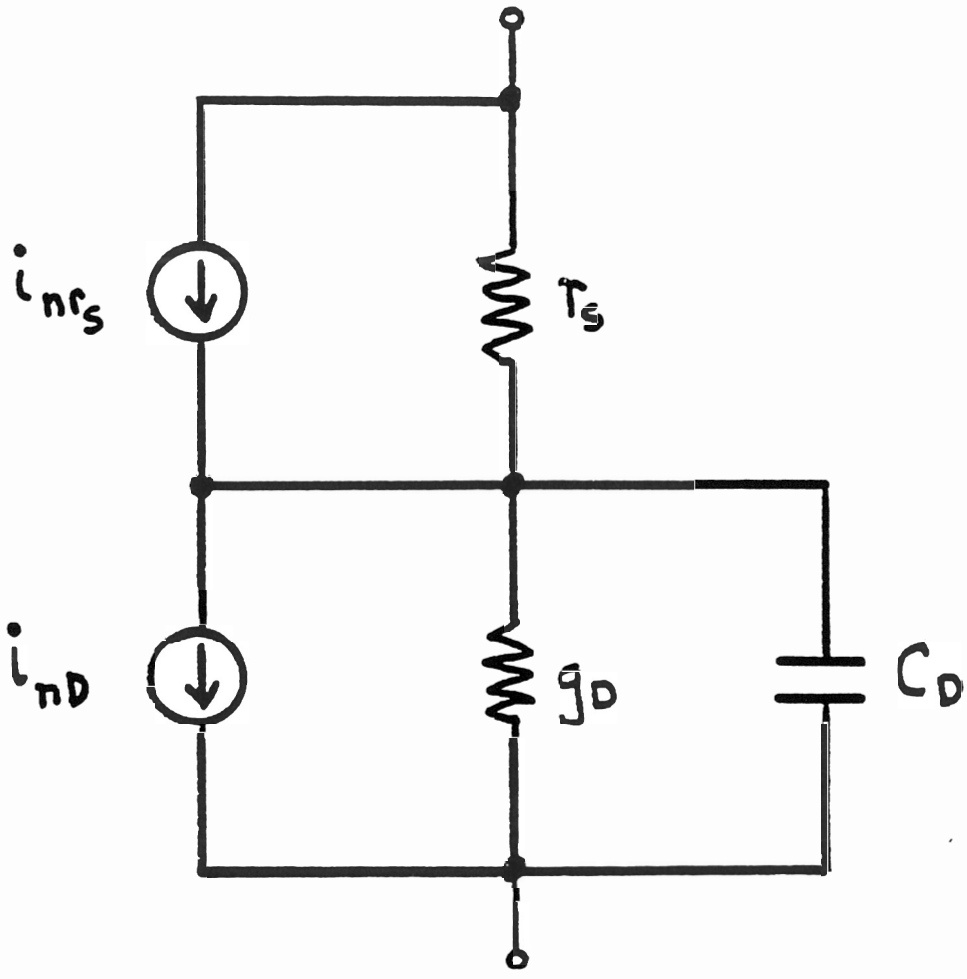


Fig. A2.19. The Noise Model for a Diode.

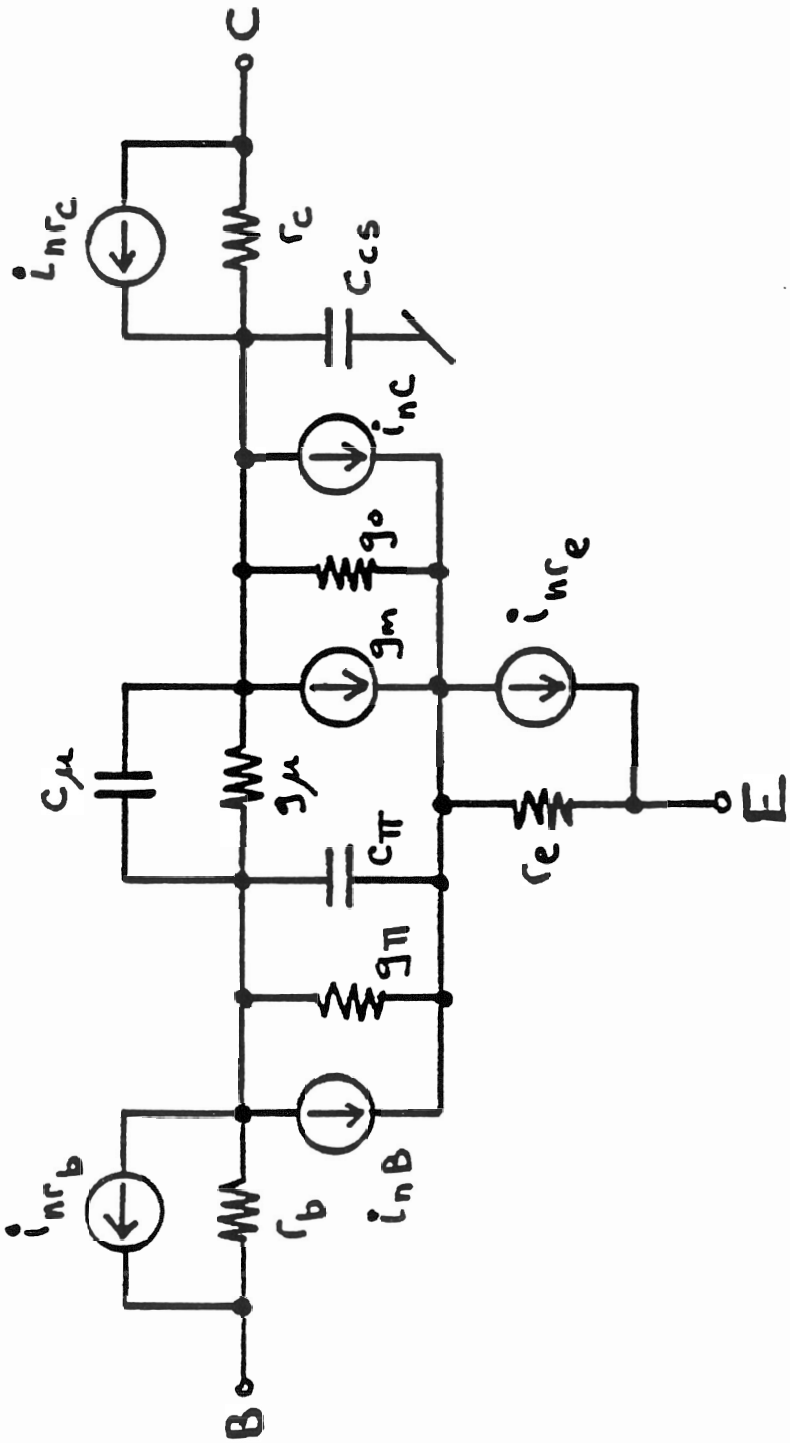


Fig. A2.20. The Noise Model for a BJT

Again, the model parameters K_f and a_f control the flicker noise characteristics of the device. A characterization of the devices in a 741 operational amplifier [9] produced values for the npn input devices of $K_f = 6.6 \times 10^{-16}$ and $a_f = 1$. The pnp input devices in this circuit were characterized by the values $K_f = 3.0 \times 10^{-12}$ and $a_f = 1.5$. The values of K_f are for I_B in amps and frequency in Hertz.

The JFET and MOSFET noise models are given in Figs. A2.21 and A2.22, respectively. The thermal noise generation in the drain and source regions of the FET's is modeled by the two sources i_{r_d} and i_{r_s}. The values of these sources are determined by (A2.69). Shot and flicker noise is modeled by the current source i_{nD} which is defined by the equation.

$$i_{nD} = \sqrt{\frac{8kTg_m}{3} + \frac{K_f I_D^{a_f}}{f}} \tag{A2.73}$$

where g_m is the small-signal transconductance of the FET. The parameters K_f and a_f determine the flicker noise characteristics of the FET's. Typical values for these parameters are $K_f = 10^{-14}$ and $a_f = 1$ [95].

A2.16. The Temperature Dependence of Junction Saturation Currents

The saturation current of the diode, the BJT, the gate junctions of the JFET, and the substrate junctions of the MOSFET vary with temperature according to the equation

$$\frac{I_S(T_1)}{I_S(T_2)} = \left(\frac{T_1}{T_2}\right)^{p_T} \exp\left\{\frac{q\varepsilon_g}{kT_1}\left[\frac{T_1}{T_2} - 1\right]\right\} \tag{A2.74}$$

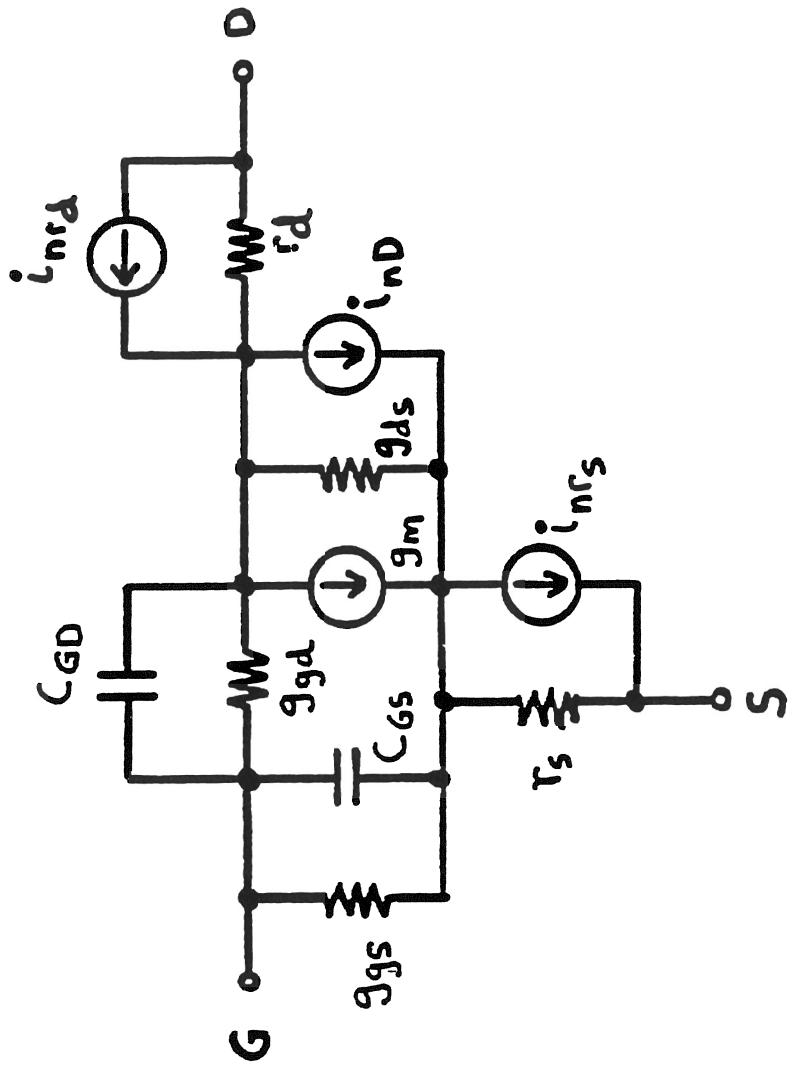


Fig. A2.21. The Noise Model for a JFET.

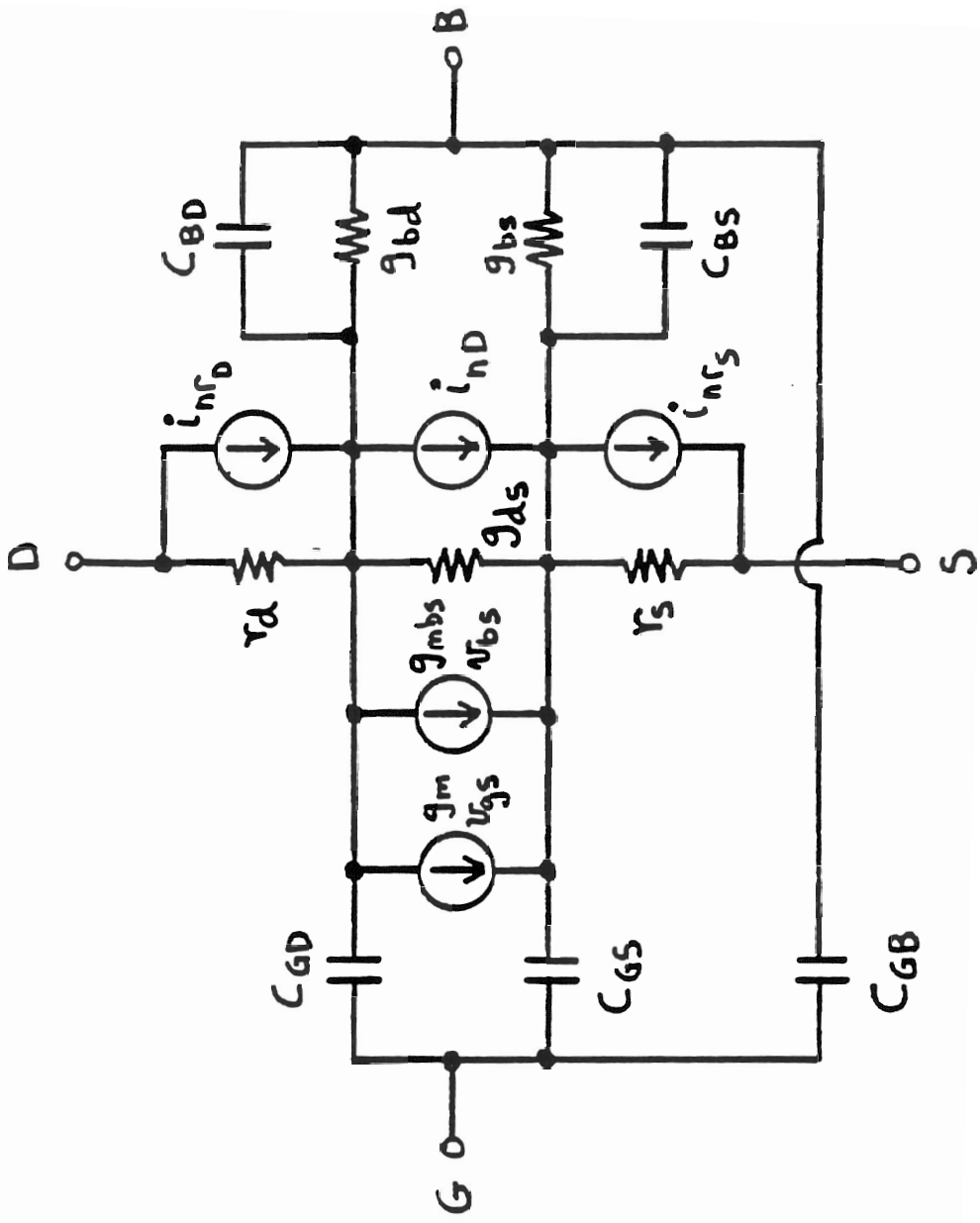


Fig. A2.22. The Noise Model for a MOSFET.

For the JFET and MOSFET models, the parameter ε_g is fixed at a value of 1.11 and the parameter p_T is fixed at a value of 3. For silicon diodes and silicon BJT's ε_g typically is 1.11 and p_T typically is 3. For Schottky-Barrier diodes, ε_g typically is 0.69 (for Al-Si barriers) an p_T typically is 2.

A2.17. Computational Considerations

To aid convergence, a small conductance is inserted in parallel with every junction in the circuit. The value of this conductance, GMIN, is a program parameter that can be set by the user. The default value for GMIN is 10^{-12} mhos.

The exponential terms that are associated with junctions in the four semiconductor models are approximated by a linear function for reverse bias. Specifically, the expansion

$$\exp(-a) \cong 1 - a \qquad \text{(A2.75)}$$

is used to evaluate all exponentials with negative arguments.

The depletion-layer capacitance equations in the four semiconductor models are of the form

$$C = C_o[1 - \frac{V}{\phi}]^{-n} \qquad \text{(A2.76)}$$

If V is positive, (A2.76) can become numerically unstable. Therefore, if V is positive, (A2.76) is approximated by the equation

$$C = C_o[1 + \frac{nV}{\phi}] \qquad \text{(A2.77)}$$

APPENDIX 3

SPICE INPUT SYNTAX

The input file[1] for the SPICE program defines the circuit and specifies what analyses are to be performed and what output is to be generated. The input format is a "free-format" style that does not require that data be entered in specific column locations on a input line. If possible, the program will supply reasonable "default" values for parameters that are not specified. Hence, the user need specify only a minimum amount of information in a simple format to simulate a circuit.

Data fields are delimited by one or more blanks, a comma, an equal sign, a right parenthesis, or a left parenthesis. Blanks that precede or follow a delimiter are ignored. An input line is continued onto the next line by including a plus sign (+) before the first field of the continuation line. Note, however, that a particular data field cannot be split between lines since end-of-line is a treated as a delimiter.

A name field begins with an alphabetic letter. Only the first seven alphanumeric characters of the name are retained by the program. Therefore, the name "VERYLONGNAME" is abbreviated to the name "VERYLON." Names identify circuit elements, model parameters, output variables, and program keywords.

A number field may be an integer, such as 12 or -44, a floating point number, such as 3.14159 or -6.32, or a number expressed in

---

[1] In this appendix the terms "file" and "input deck" are used interchangeably, as are the terms "card" and "line."

exponential notation, such as 1E-14 or -2.73E3. Any number can be followed immediately by one of the scale factors that are given in Table A3.1. Notice that there can be no blanks between the number and its scale factor; the field 2PF and the two fields 2 PF are not equivalent. Letters that immediately follow a number field that are not scale factors are ignored. Letters that immediately follow a scale factor also are ignored. Hence 10, 10V, 10VOLTS, and 10HZ all represent the number ten, and M, MA, MSEC, and MMHOS all represent the scale factor 10^{-3}. The number fields 1000, 1000.0, 1000HZ, 1E3, 1.0E3, 1KHZ, and 1K are equivalent representations of the number 10^{3}.

A void field is specified by two commas that optionally may be separated by blanks. Void fields are equivalent to zero numeric fields. Hence, the card

```
.MODEL QMOD NPN ,,  , ,4.7,,,  , 19.0
```

is equivalent to the card

```
.MODEL QMOD NPN 0 0 0 4.7 0 0 0 19.0
```

The first line of input is the title; the contents of the title are printed <u>verbatim</u> as the heading of each section of SPICE output. The last card in the deck is a .END card. The only purpose of the .END card is to signify the end of the SPICE input. Except for the title card and the .END card, the input cards can be in any order.

Several circuits can be simulated in one SPICE run by including the SPICE decks for each simulation in the input. The program reads the input until a .END card is encountered, performs the appropriate simulation, and then reads the input again until the next .END card

G	10^{9}
MEG	10^{6}
K	10^{3}
M	10^{-3}
U	10^{-6}
N	10^{-9}
P	10^{-12}

Table A3.1. The SPICE Scale Factors

is encountered. This process continues until all of the input cards have been read. Note, however, that if a .END card is omitted, two separate input decks will be read as a single input deck, and neither simulation will run successfully.

A3.1. SPICE Circuit Description

The circuit is defined by a set of element cards. Each element in the circuit is assigned a unique alphanumeric name. Each node in the circuit is assigned a unique integer number. The node number zero is reserved for the ground node.

To insure a unique mathematical description of the circuit, certain limitations are imposed upon the circuit interconnection. Specifically, every node in the circuit, including the ground node, must have at least two elements connected to it. Every node in the circuit must have a dc path to ground. The circuit cannot contain a loop of voltage sources and/or inductors.

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

The format for the twelve SPICE element types is given in Table A3.2. Data fields that are enclosed in brackets are optional. The names AC and OFF are program keywords that will be explained in later sections of this appendix. The strings "XXXXXX", "YYYYYY", and "ZZZZZZ" in Table A3.2 denote arbitrary alphanumeric strings. A resistor name, for example, must begin with the letter R and can

RESISTORS:	RXXXXXX N+ N- VALUE
CAPACITORS:	CXXXXXX N+ N- VALUE [INCOND]
INDUCTORS:	LXXXXXX N+ N- VALUE [INCOND]
COUPLED INDUCTORS:	KXXXXXX LYYYYYY LXXXXXX VALUE
INDEPENDENT VOLTAGE SOURCE:	VXXXXXX N+ N- [DC/TRVAL] ["AC" ACVAL [ACPHS]]
INDEPENDENT CURRENT SOURCE:	IXXXXXX N+ N- [DC/TRVAL] ["AC" ACVAL [ACPHS]]
LINEAR VOLTAGE-CONTROLLED CURRENT SOURCE	GXXXXXX N+ N- NC+ NC- VALUE [DELAY]
NONLINEAR VOLTAGE-CONTROLLED CURRENT SOURCE	NXXXXXX N+ N- NC+ NC- INCOND P1 [P2...P2ϕ]
DIODE	DXXXXXX NA NC MODEL [AREA] ["OFF"]
BJT	QXXXXXX NC NB NE MODEL [AREA] ["OFF"]
JFET	JXXXXXX ND NG NS MODEL [AREA] ["OFF"]
MOSFET	MXXXXXX ND NG NS NB MODEL [AREA] ["OFF"]

Table A3.2. SPICE Element Card Format.

contain from one to seven characters. Hence, R, R1, RSE, ROUT, and R3AC2ZY are valid resistor names.

A3.2. Resistors

A resistor name begins with the letter R. The resistor card contains the name, the positive node, the negative node, and the resistance (in ohms):

RXXXXXX N+ N- value

Resistor values cannot be zero or negative. The order of the two resistor nodes is not important.

EXAMPLES: R37 19 14 1K

RFEED 37 3 1MEG

A3.3. Capacitors

A capacitor name begins with the letter C. The capacitor card contains the name, the positive node, the negative node, the capacitance (in farads), and an optional initial condition:

CXXXXXX N+ N- value [incond]

Capacitor values cannot be zero or negative. The initial condition is the time-zero value of capacitor current (in amps) that flows from the positive node, through the capacitor, to the negative node. If the initial condition is not specified, a zero value is supplied.

EXAMPLES: CBYP 13 0 1UF

COSC 17 19 10UF 7.6MA

A3.4. Inductors

An inductor name begins with the letter L. The inductor card

contains the name, the positive node, the negative node, the inductance (in henries), and an optional initial condition:

LXXXXXX N+ N- value [incond]

The initial condition for an inductor is the time-zero value of inductor voltage (in volts). If the initial condition is not specified, a zero value is supplied.

EXAMPLES: LSHUNT 13 7 1MH

L31 2 3 10UH 4.7VOLTS

A3.5. Coupled Inductors

Coupling between inductors is specified by a coupled-inductor card. The coupling element name begins with the letter K. The card contains the name, the names of the two coupled inductors, and the value of the coefficient of coupling:

KXXXXXX LYYYYYY LZZZZZZ value

The absolute value of the coefficient of coupling must be less than unity. Specifying a negative value for the coefficient of coupling is equivalent to inverting the polarity of either of the coupled inductors. The two coupled inductors, LYYYYYY and LZZZZZZ, can be entered in either order.

EXAMPLES: K43 LAAA LBB 0.999

A3.6. Independent Voltage and Current Sources

An independent voltage source name begins with the letter V, and an independent current source name begins with the letter I.

These element cards contain the source name, the positive node, the negative node, the dc and transient value, and the letters AC followed by the magnitude and relative phase of the ac value:

VXXXXXX N+ N- [dctrval] [AC acmag [acphs]]

IXXXXXX N+ N- [dctrval] [AC acmag [acphs]]

Either the dc/transient value or the ac values or both may be omitted; if a value is not specified, a zero value is supplied.

```
EXAMPLES:  VCC    17    0    5.0
           IIN    1     3    AC 1
           IIN    1     7    16    0.168    AC    0.3    45.0
           VMEAS  16    17
```

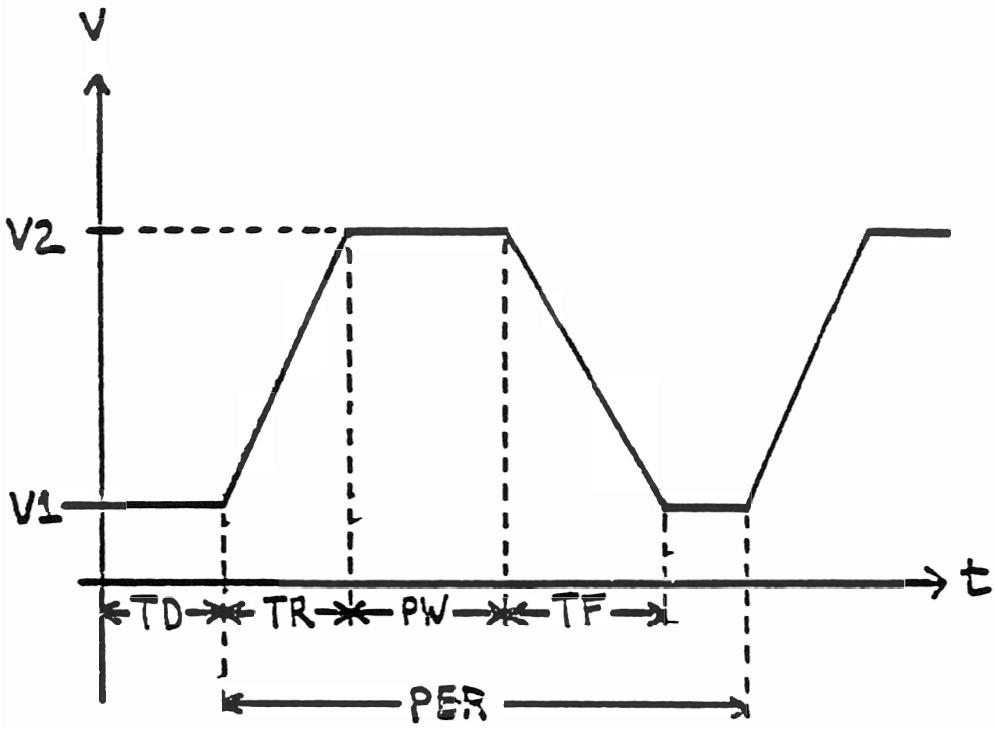
An independent source can be assigned a time-dependent value for transient analysis. If a source is assigned a time-dependent value, the time-zero value is used for dc analysis. Three time-dependent source functions are available: PULSE, SIN, and EXP.

The PULSE function is illustrated in Fig. A3.1. This function contains seven parameters: the initial value, the pulsed value, the dealy time, the risetime, the falltime, the pulse width, and the period:

PULSE (V1 V2 TD TR TF PW PER)

Only the initial and pulsed values must be specified; the default values that are given in Fig. A3.1 will be supplied for the remaining parameters if they are not specified. The variables

PULSE (V1,V2,TD,TR,TF,PW,PER)



V1	Initial Value	Must be Specified
V2	Pulsed Value	Must be Specified
TD	Delay Time	TSTEP
TR	Rise Time	TSTEP
TF	Fall Time	TSTEP
PW	Pulse Width	TSTEP
PER	Period	TSTOP

Fig. A3.1. The PULSE Function.

TSTEP and TSTOP are the print increment and the stop time that are specified on the .TRAN card, which is explained in a later section of this appendix.

The SIN function is illustrated in Fig. A3.2. This function contains five parameters: the offset value, the amplitude, the frequency (in Hz), the delay time, and a decay factor:

SIN (VO VA FREQ TD THETA)

The SIN function is defined by the equation

$$V = \begin{cases} V_o & t \le t_d \\ V_o + V_A\{\exp[-\theta(t-t_d)]\}\{\sin[2\pi f(t-t_d)]\} & t \ge t_d \end{cases} \quad \text{(A3.1)}$$

The parameters VO, VA, and FREQ must be specified. The default values for the remaining parameters, TD and THETA, are given in Fig. A3.2.

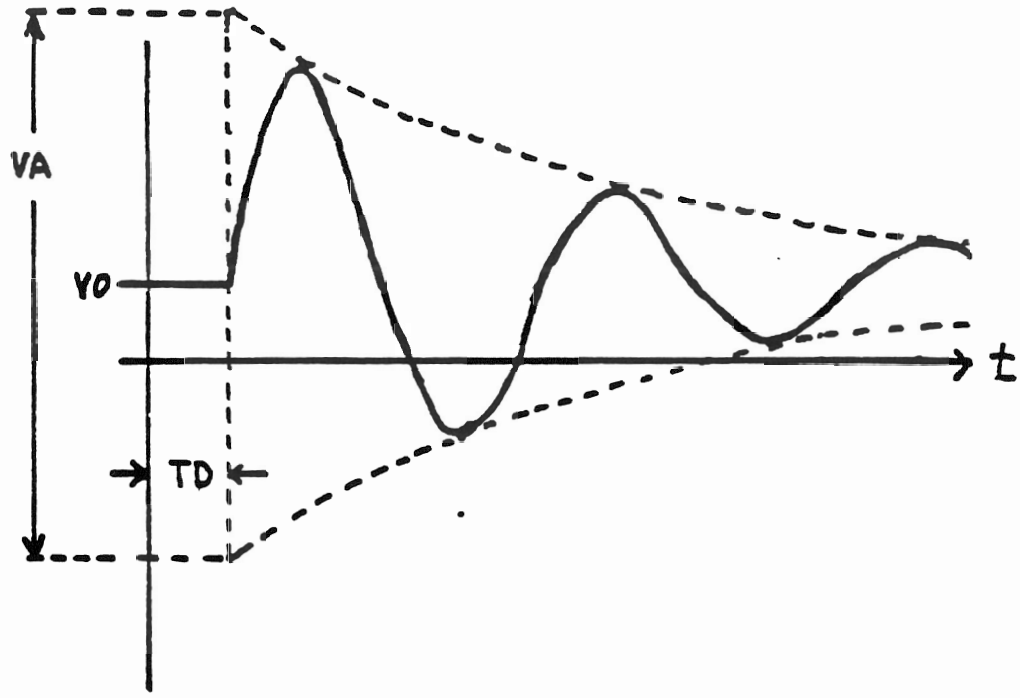
The EXP function is illustrated in Fig. A3.3. This function contains six parameters: the initial value, the final value, the leading-edge delay, the leading-edge time-constant, the falling-edge delay, and the falling-edge time-constant:

EXP (V1 V2 TD1 TAU1 TD2 TAU2)

The EXP function is defined by the equation

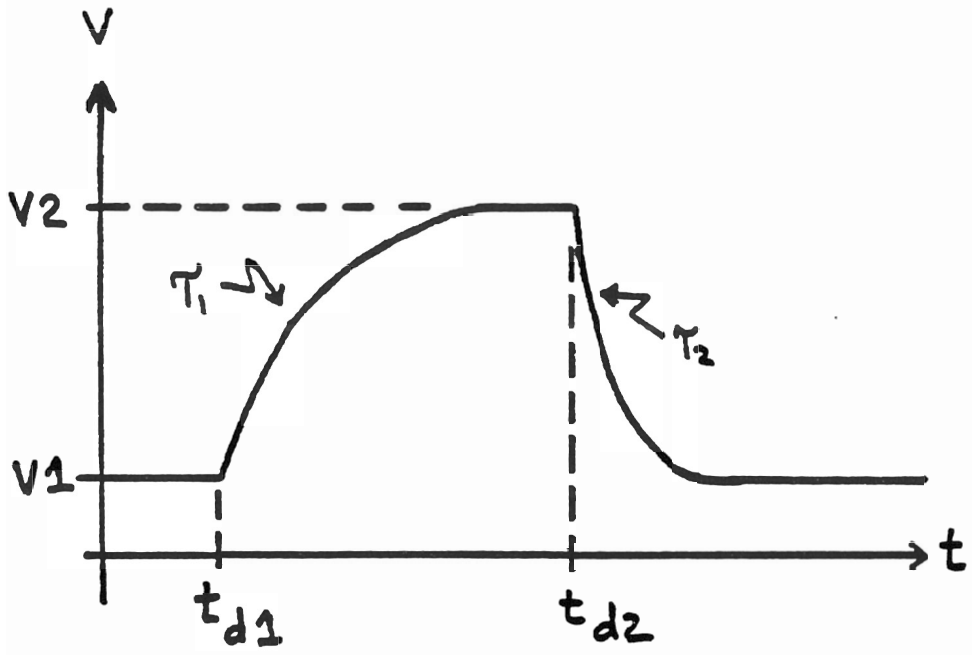
$$V = \begin{cases} V_1 & 0 \le t \le t_{d1} \\ V_1 + (V_2-V_1)[1 - \exp(-\frac{t-t_{d1}}{\tau_1})] & t_{d1} \le t \le t_{d2} \\ V_1 + (V_2-V_1)[1 - \exp(-\frac{t-t_{d1}}{\tau_1})] & \\ \quad + (V_1-V_2)[1 - \exp(-\frac{t-t_{d2}}{\tau_2})] & t \ge t_{d2} \end{cases} \quad \text{(A3.2)}$$

SIN (VO,VA,FREQ,TD,THETA)



VO	Offset	Must be Specified
VA	Amplitude	Must be Specified
FREQ	Frequency	Must be Specified
TD	Delay Time	TSTEP
θ	Damping Coef.	0

EXP (V1,V2,TD1,TAU1,TD2,TAU2)



V1	Initial Value	Must be Specified
V2	Pulsed Value	Must be Specified
TD1	Lead Delay	TSTEP
τ_1	Lead Time-Constant	TSTEP
TD2	Fall Delay	TD1 + TSTEP
τ_2	Fall Time-Constant	TSTEP

Fig. A3.3. The EXP Function.

The initial value and the pulsed value must be specified. The default values for the remaining values are given in Fig. A3.3.

A3.7. Linear Voltage-Controlled Current Sources

The name for a linear voltage-controlled current source begins with the letter G. This element card contains the name, the positive node, the negative node, the positive controlling node, the negative controlling node, the value of transconductance (in mhos), and an optional value for the delay (in secs):

GXXXXXX N+ N- NC+ NC- value [delay]

If the delay is not specified, a zero value is supplied.

EXAMPLES: G35 2 0 5 0 0.1MMHO

GXZ 17 3 19 7 20MMHO 5NS

A3.8. Nonlinear Voltage-Controlled Current Sources

The name for a nonlinear voltage-controlled current source begins with the letter N. The element card contains the name, the positive node, the negative node, the positive controlling node, the negative controlling node, the initial condition, and the polynomial coefficients:

NXXXXXX N+ N- NC+ NC- incond P1[P2 P3 ... P20]

The initial condition, which must be specified, is an initial "guess" for the voltage between the controlling nodes. The program uses this initial condition to obtain the dc operating point of the circuit. After convergence has been attained, the program continues iterating to obtain the exact value for the controlling

voltage. The initial condition need not be exact, but to reduce the computational effort for the dc operating point, the initial condition should be fairly close to the actual controlling voltage.

Only the first polynomial coefficient need be specified; the remaining polynomial coefficients will be set to zero if they are not specified. No more than twenty coefficients can be specified.

EXAMPLE: N31 21 0 21 0 0.0 0.2 0.05 0.007

A3.9. Semiconductor Devices

The models for the four semiconductor devices that are included in the SPICE program contain many parameters. Moreover, many devices in a circuit often are defined by the same set of device model parameters. For these reasons, a set of device model parameters are defined on a separate .MODEL card and assigned a unique model name. The device element cards in SPICE then reference the model name. This scheme alleviates the need to specify repetitively the model parameters on each device element card.

Each device element card contains the device name, the nodes that the device is connected to, and the device model name. In addition, two optional parameters may be specified for each device: an area factor and an initial condition.

The area factor determines the number of equivalent parallel devices of the specified model. For example, an area factor of 3.0 implies that the device is equivalent to three devices of the specified model name.

The initial condition is included to improve the dc convergence for circuits that contain more than one stable state. If a device

is specified OFF, the dc operating point is determined with the terminal voltages for that device set to zero. After convergence is obtained, the program continues to iterate to obtain the exact value for the terminal voltages. If a circuit has more than one dc stable state, the OFF option can be used to force the solution to correspond to a desired state. If a device is specified OFF when in reality the device is conducting, the program will obtain the correct solution (assuming the solutions converge) but more iterations will be required since the program must independently converge to two different solutions.

A3.10. Diodes

A diode name begins with the letter D. A diode card contains the name, the anode node, the cathode node, the model name, an optional value for the are factor, and an optional initial condition:

DXXXXXX NA NC MODEL [area] ["OFF"]

EXAMPLES: DBRI 23 25 DM1

DS1 24 27 SBD37X 2.5 OFF

A3.11. Bipolar Junction Transistors

A BJT name begins with the letter Q. A BJT card contains the name, the collector node, the base node, the emitter node, the model name, an optional value for the area factor, and an optional initial condition:

QXXXXXX NC NB NE MODEL [area] ["OFF"]

```
EXAMPLES:  Q23  4  6  9  QMODEL
           QLS  23  26  29  FAST  2.0  OFF
```

A3.12. Junction Field-Effect Transistors

A JFET name begins with the letter J. The JFET card contains the name, the drain node, the gate node, the source node, the model name, an optional value for the area factor, and an optional initial condition:

```
JXXXXXX  ND  NG  NS  MODEL  [area]  ["OFF"]

EXAMPLES:  J34  6  7  8  JMOD2
           JIN  1  17  20  JM0  1.5  OFF
```

A3.13. MOSFET's

A MOSFET name begins with the letter M. The MOSFET card contains the name, the drain node, the gate node, the source node, the substrate node, the model name, an optional value for the area factor, and an optional initial condition:

```
MXXXXXX  ND  NG  NS  NB  MODEL  [area]  ["OFF"]

EXAMPLES:  MLONG 23  25  27  28  MOS1  0.1
           MIN  12  14  16  1  MOS2
           MGATE  11  14  17  10  MOS2  2.5  OFF
```

A3.14. The .MODEL Card

The device model parameters for a given model name are specified on a .MODEL card. This card contains the word ".MODEL", the model name, the device model type, and the model parameter values:

```
.MODEL  name  type  [parameter values]
```

The model name is any arbitrary name. The model type is one of the seven keywords that are given in Table A3.3. Notice that device polarity is specified as a part of the model definition.

Every SPICE model parameter is referenced via a unique keyword. Model parameters that are not specified are assigned default values. The diode model parameters, keywords, and default values are listed in Table A3.4. Table A3.5 contains the parameter keywords and default values for the BJT model. The JFET parameter keywords and default values are given in Table A3.6, and Table A3.7 contains the keywords and default values for the MOSFET model parameters. These four models are presented in detail in Appendix 2 of this thesis.

Model parameters may be specified by two different methods. The first, and simplest, method is to specify the parameter keyword, followed by an equal sign (=), followed by the parameter value. For example, the following card defines the model name QX:

```
.MODEL  QN  NPN  (BF=30, RB=50, IS=1E-15, TF = INS)
```

Model QX is an npn BJT with B_F equal to 30, r_b equal to 50, I_S equal to 10^{-15}, and τ_F equal to 10^{-9}. The parameters that are not specified are assigned the default values that are given in Table A3.5.

The second method of defining model parameter values is by a string of numbers. The order of the values must be the same as the order of the keywords that is given in Tables A3.4 - A3.7. Parameter locations are skipped with successive commas. The model card

```
.MODEL  QN  NPN  30,, 1E-15,50,,,,,,,,,INS
```

Keyword	Model type
D	diode model
NPN	npn BJT model
PNP	pnp BJT model
NJFET	n-channel JFET model
PJFET	p-channel JEFT model
NMOS	n-channel MOSFET model
PMOS	p-channel MOSFET model

Table A3.3. Model Type Keywords

SYMBOL	KEYWORD	PARAMETER NAME	DEFAULT VALUE	UNITS
I_S	I_S	Saturation current	10^{-14}	amps
r_s	RS	Ohmic resistance	0	ohms
n	N	Emission coefficient	1	-
τ_t	TT	Transit time	0	sec
C_{jo}	CJO	Zero-bias junction capacitance	0	farad
ϕ_B	PB	Junction potential	1	volts
m	M	Grading coefficient	0.5	-
ε_g	EG	Energy gap	1.11	eV
p_T	PT	Saturation current temperature exponent	3	-
K_f	KF	Flicker-noise coefficient	0	-
a_f	AF	Flicker-noise exponent	1	-

Table A3.4. SPICE Diode Model Parameters.

SYMBOL	KEYWORD	PARAMETER NAME	DEFAULT VALUE	UNITS
B_F	BF	Ideal forward current-gain coefficient	100	-
B_R	BR	Ideal reverse current-gain coefficient	1	-
I_S	IS	Saturation current	10^{-14}	amps
r_b	RB	Base ohmic resistance	0	ohms
r_c	RC	Collector ohmic resistance	0	ohms
r_e	RE	Emitter ohmic resistance	0	ohms
V_A	VA	Forward Early voltage	∞*	volts
V_B	VB	Reverse Early voltage	∞*	volts
I_K	IK	Forward Knee current	∞*	amps
C_2	C2	Forward nonideal base-current coefficient	0	-
n_{EL}	NEL	Nonideal b-e emission coefficient	2	-
I_{KR}	IKR	Reverse knee current	∞*	amps
C_4	C4	Reverse nonideal base-current coefficient	0	-
n_{CL}	NCL	Nonideal b-c emission coefficient	2	-
τ_F	TF	Forward transit time	0	sec
τ_R	TR	Reverse transit time	0	sec
C_{CS}	CCS	Collector-substrate capacitance	0	farads
C_{je}	CJE	Zero-bias b-e junction capacitance	0	farads
$_e$	PE	B-E junction potential	1	volts
m_e	ME	B-E junction grading coefficient	0.5	-
C_{jc}	CJC	Zero-bias b-c junction capacitance	0	farads

ϕ_C	PC	B-C junction potential	1	volts
m_c	MC	B-C junction grading coefficient	0.5	-
ϵ_g	EG	Energy gap	1.11	eV
p_T	PT	Saturation current temperature exponent	3	-
K_f	KF	Flicker-noise coefficient	0	-
a_f	AF	Flicker-noise exponent	1	-

*Since V_A, V_B, I_K, and I_{KR} cannot be zero-valued, a zero value for these four parameters is interpreted to be an infinite value

Table A3.5. SPICE BJT Model Parameters

SYMBOL	KEYWORD	PARAMETER NAME	DEFAULT VALUE	UNITS
V_{TO}	VTO	Pinchoff voltage	0	volts
β	BETA	Transconductance parameter	10^{-4}	amps/V^2
λ	LAMBDA	Channel-length modulation parameter	0	1/V
r_d	RD	Drain ohmic resistance	0	ohms
r_s	RS	Source ohmic resistance	0	ohms
C_{GS}	CGS	Zero-bias gate-source capacitance	0	farads
C_{GD}	CGD	Zero-bias gate-drain capacitance	0	farads
ϕ_B	PB	Gate-junction potential	1	volts
I_S	IS	Gate-junction saturation current	10^{-14}	amps
K_f	KF	Flicker-noise coefficient	0	-
a_f	AF	Flicker-noise exponent	1	-

Table A3.6. SPICE JFET Model Parameters

SYMBOL	KEYWORD	PARAMETER NAME	DEFAULT VALUE	UNITS
V_{TO}	VTO	Threshold voltage	0	volts
β	BETA	Transconductance parameter	10^{-4}	amps/V^2
γ	GAMMA	Bulk threshold parameter	0	1/ v
ϕ	PHI	Surface potential	0.5	volts
λ	LAMBDA	Channel-length modulation parameter	0	1/v
r_d	RD	Drain ohmic resistance	0	ohms
r_s	RS	Source ohmic resistance	0	ohms
C_{GS}	CGS	Gate-source capacitance	0	farads
C_{GD}	CGD	Gate-drain capacitance	0	farads
C_{GB}	CGB	Gate-substrate capacitance	0	farads
C_{BD}	CBD	Zero-bias substrate-drain capacitance	0	farads
C_{BS}	CBS	Zero-bias substrate source capacitance	0	farads
ϕ_B	PB	Substrate-junction potential	1	volts
I_S	IS	Substrate-junction saturation current	10^{-14}	amps
K_f	KF	Flicker-noise coefficient	0	-
a_f	AF	Flicker-noise exponent	1	-

Table A3.7. SPICE MOSFET Model Parameters

is equivalent to the previous definition of model QX.

A3.15. Subcircuits

A subcircuit that consists of SPICE elements can be defined and referenced in a similar fashion to device models. The subcircuit is defined in the input deck by a grouping of element cards; the program then automatically inserts the group of elements whenever the subcircuit is referenced. There is no limit on the size or complexity of subcircuits. The three circuits that are illustrated in Fig. A3.4 constitute possible subcircuits.

A subcircuit name begins with the letter X. The device card contains the name, the subcircuit nodes, and the subcircuit name. The mulit-emitter device that is shown in Fig. A3.4 could be specified in an input deck by the following card:

```
X13  2  4  17  3  1  MULTI
```

The nodes must be in the order that they are defined in on the subcricuit card.

A3.16. The .SUBCKT Card

A subcircuit is defined with a .SUBCKT card. This card contains the letters ".SUBCKT", the subcircuit name, and the external nodes. A subcircuit may have from 1 to 20 external nodes. The group of element cards that follow the .SUBCKT card define the subcircuit. The last card in a subcircuit definition in a .FINIS card. The multi-emitter device that is shown in Fig. A3.4 would be specified by the following group of cards:

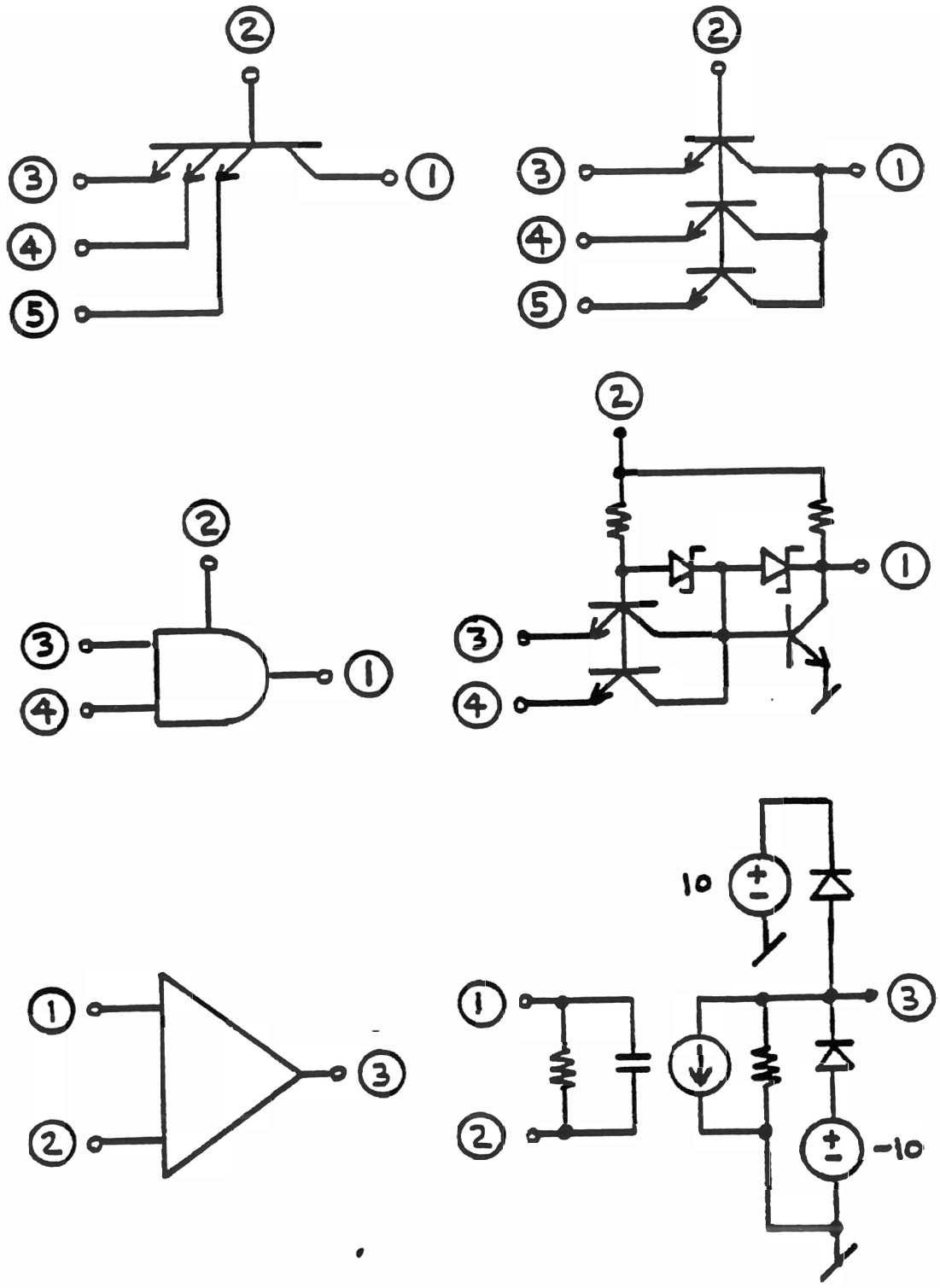


Fig. A3.4. Possible Subcircuits for SPICE.

```
.SUBCKT  MULTI  1  2  3  4  5
Q1  1  2  3  QLONG
Q2  1  2  4  QLONG
Q3  1  2  5  QLONG
.FINIS
```

Model or control cards which occur between the .SUBCKT card and the .FINIS card have the same effect as if they wer out of the nest. Subcircuits cannot be nested; that is, a subcircuit cannot contain another subcircuit.

A3.17. Control Cards

In addition to specifying the circuit, the SPICE input deck specifies the analyses to be performed and the output that is desired by the user. These additional specifications are entered by the SPICE control cards that are summarized in Table A3.8. The syntax for these control cards is described in the next sections of this appendix.

A3.18. The Title Card

This card must be the first card in the deck. It contains the alphanumeric title of the job which is printed as a heading for the various sections of program output.

A3.19. The .END Card

This card contains the letters ".END" and must be the last card in the deck. If only one simulation is run, the .END card may be omitted.

Title Card

.END Card

Comment Card

.OUTPUT Card

.DC Card

.TF Card

.SENS Card

.TRAN Card

.FOUR Card

.AC Card

.NOISE Card

.DISTO Card

.TEMP Card

.OPTIONS Card

Table A3.8. The SPICE Control Cards

A3.20. The Comment Card

This card allows documentation of input decks. The first nonblank character of the card is an asterisk (*); the card is printed with the listing but otherwise is ignored by the program.

A3.21. The .OUTPUT Card

This card defines an output variable and specifies the analyses in which the output is to be printed or plotted. Outputs may be node voltage differences, or voltage-source currents. Additionally, in ac analysis, the output noise, the input noise, and any of the distortion measures can be specified as outputs and printed or plotted.

The general syntax for the .OUTPUT card is as follows:

```
                ┌                    ┐
                │ VXXXXXX  N+  N-    │
                │ IXXXXXX  VYYYYYY   │
                │ ONOISE             │
                │ RINOISE            │
.OUT            │ HD2                │   "PRINT" [options] "PLOT" [options]
                │ HD3                │
                │ SIM2               │
                │ DIM2               │
                │ DIM3               │
                └                    ┘
```

For voltage outputs, the output name is followed by the positive node and the negative node. For current outputs, the output name is followed by the voltage source name. The current is positive if it flows from the source positive node, through the source, to

the source negative node.

The keywords ONOISE and RINOISE refer to the output noise refer to the output noise and the reflected input noise, respectively, in noise analysis. The keywords HD2, HD3, SIM2, DIM2, DIM3 refer to the distortion products in distortion analysis. These products are detailed in a later section of this appendix.

An output variable can be printed or plotted in any of the analyses that are performed with SPICE. The .OUTPUT card also controls when and how the output variable is presented as output. The various output options that are available in SPICE are listed in Table A3.9. An output is printed in tabular form by adding the desired output options after the keyword PRINT. For example, the .OUTPUT card.

```
.OUTPUT  V7  7  0  PRINT  DC  TR  MAG  PHS
```

causes node voltage 7 to be printed in the dc transfer curve analysis and the transient analysis. Additionally, both the magnitude and phase of V7 will be printed in the ac analysis.

A line-printer plot of an output is obtained by adding the desired output options after the keyword PLOT. Plot limits may be added after each output option. If plot limits are not included, the program automatically determines plot limits. The .OUTPUT card

```
.OUTPUT  V7  7  0  PLOT  DC  TR  0  5  MA
```

causes a line-printer plot of V7 to be generated in the dc transfer curve analysis, the transient analysis, and the ac analysis. In the ac analysis, the magnitude of V7 is plotted. The program will use the plot limits (0,5) in the transient plot and will determine suitable limits in the dc and ac analysis plots.

KEYWORD OPTION	MEANING
DC	DC transfer-curve analysis
TR	Transient analysis
MA	AC analysis, magnitude
PH	AC analysis, phase
DB	AC analysis, magnitude in DB
RE	AC analysis, real part
IM	AC analysis, imaginary

Table A3.9. SPICE Output Options

A3.22. The .DC Card

This card defines the dc transfer curve source and limits. It contains the letters ".DC" followed by a voltage or current source name followed by the start value, the stop value, and the additive increment. For example, the card

```
.DC  VIN  0.5  5.0  0.25
```

will result in the voltage source VIN being swept from 0.5 volts to 5 volts in increments of 0.25 volts.

A3.23. The .TF Card

This card defines the small-signal input and output for the dc small-signal analysis. It contains the letters ".TF" followed by the small-signal output variable and the small-signal input source. If the card

```
.TF  VOUT  VIN
```

is included in the input deck, then the program will compute, as a part of the dc operating-point analysis, the small-signal dc value of transfer function (the ratio of VOUT to VIN), the small-signal input impedence at VIN, and the small-signal output impedence at VOUT. VOUT must be an output variable that has been defined on an .OUTPUT card, and VIN must be source that has been defined in the circuit description.

A2.24. The .SENS Card

This card contains the letters ".SENS" followed by one to ten output variable names that have been defined on .OUTPUT cards. As an example, if the card

```
.SENS  VOUT  V12  VINT
```

is included in the input deck, the program will determine the dc small-signal sensitivities of VOUT, V12, and VINT with respect to every circuit parameter. The JFET and MOSFET model parameters are not included in the sensitivity computation.

A3.25. The .TRAN Card

This card contains the transient analysis simulation controls. The card contains, as a minimum, the letters ".TRAN" followed by the printing time increment and the stop time. The program will begin at time zero and compute each timepoint up to the stop time using the print increment given.

A start time can be affixed after the stop time. In this case, the program will start at time zero and compute each timepoint up to the stop time; however, only the timepoints between the start time and the stop time will be printed. This option is designed for simulations where the circuit is allowed to reach a steady-state solution.

A3.26. The .FOUR Card

The Fourier analysis can be specified on the .FOUR card. This card contains the letters ".FOUR" followed by the output variable name and the fundamental frequency. If specified, the program will compute the first nine Fourier harmonics of the specified output variable response.

A3.27. The .AC Card

This card contains the AC analysis controls. The .AC card

contains the letters ".AC" followed by one of the following three frequency variation types:

a) the letters "DEC" followed by the points per decade, the start frequency, and the stop frequency.

b) the letters "OCT" followed by the points per octave, the start frequency, and the stop frequency.

c) The letters "LIN" followed by the number of points, the start frequency, and the stop frequency.

If a .AC card is included in the deck, the program will perform an ac analysis of the circuit over the frequency range that is specified on the .AC card.

A3.28. The .NOISE Card

This card controls the noise analysis of the circuit. It contains the letters .NOISE followed by the noise output reference, the noise input reference, and the summary interval. The program will compute the equivalent output noise at the specified output as well as the equivalent input noise at the specified input. In addition, the contribution of every noise generator in the circuit will be printed at every nth point where n is the summary interval. If the summary interval is zero, no summary printout will occur.

A3.29. The .DISTO Card

This card contains the parameters that control the distortion analysis. Distortion analysis, in turn, is performed in conjunction with the ac small-signal analysis. This subanalysis will determine both harmonic distortion products and intermodulation distortion products. The user specifies an output load resistor, RLOAD. Each

distortion product is the power delivered to RLOAD at the particular distortion frequency relative to the power delivered to RLOAD at the fundamental frequency.

Harmonic distortion products are the relative magnitudes of the higher harmonics of the output. If the circuit contains one sinusoidal excitation at the frequency f_1, then HD2, the second-order harmonic-distortion product, is the relative power in RLOAD at the frequency $2f_1$. Similarly, HD3, the third-order harmonic-distortion product, is the relative power in RLOAD at the frequency $3f_1$.

Intermodulation distortion products are measured with two sinusoidal excitations at the frequencies f_1 and f_2. SIM2, the second-order sum intermodulation product, is the relative power that is delivered at the frequency $f_1 + f_2$. Similarly, DIM3, the second-order difference intermodulation product, is the relative power that is delivered at the frequency $f_1 - f_2$. Finally, DIM3, the third-order difference intermodulation product, is the relative power that is delivered at the frequency $2f_1 - f_2$.

Any of these distortion measures can be printed or plotted at each frequency point with the .OUTPUT card. In addition, the contribution of each nonlinearity of the circuit to the total distortion can be printed.

The .DISTO card contains the letters ".DISTO" followed by the load resistor name, the summary interval, the reference power level, the ratio of the two distortion frequencies, and the amplitude of the f_2 signal as follows:

```
.DISTO  RLOAD  INTER  SKW2  REFPWR  SPW2
```

RLOAD is the name of the output load resistor into which all distortion power products are computed. The program will print the contributions of each distortion source in the circuit at every nth point, where n is the summary interval. If the summary interval is zero, no summary printout occurs. SKW2 is the ratio f_2/f_1; f_1, of course, is the frequency of the fundamental that is controlled by the .AC card. If SKW2 is not specified, a value of 0.9 is supplied. REFPWR is the reference power level that is used in computing the distortion products. If not specified, a value of 1 mW is used; that is, all distortion products are expressed in dBm. SPW2 is the amplitude of the f_2 signal relative to the amplitude of the fundamental (f_1) signal. If SPW2 is not specified, a unity value is assumed.

A3.30. The .TEMP Card

This card contains the temperatures at which the simulation is performed. Model data always is specified at 27°C (300°K). If this card is deleted, the simulation also will be performed at 27°C. The temperature card contains the letters ".TEMP" followed by one to five temperatures. For example, the card

```
.TEMP  -25  0  25  50  100
```

will cause the SPICE simulation to be performed at -25°C, 0°C, 25°C, 50°C, and 100°C.

A3.31. The .OPTIONS Card

This card allows the user to reset program controls for specific simulation purposes. The card contains the letters ".OPTIONS"

followed by any of the following options in any order:

ACCT	Causes the execution time for the various sections of the program to be printed as well as other accounting information to be printed.
LIST	Causes the summary listing of the input data to be printed.
NOMOD	Suppresses the printout of the model parameters
NODE	Causes the node table to be printed
OPTIONS	Causes the present values of options to be printed
GMIN=x	Resets the value of GMIN, the minimum conductance allowed by the program. The default value for GMIN is 10^{-12}.
RELTOL=x	Resets the relative error tolerance of the program. The default value for RELTOL is 0.001 (0.1%).
ABSTOL=x	Resets the absolute error tolerance of the program. The default value for ABSTOL is 10^{-12}.
TRTOL=x	Resets the transient error tolerance. The default value for TRTOL is 10.0.
CHGTOL=x	Resets the charge tolerance of the program. The default value for CHGTOL is 10^{-14}.
TNOM=x	Resets the nominal temperature. The default nominal temperature is 27°C (300°K).
ITL1=x	Resets the dc iteration limit. The default is 100.
ITL2=2	Resets the dc transfer curve iteration limit. The default is 20.
ITL3=x	Resets the lower transient analysis iteration limit. The default is 4.

ITL4=x	Resets the transient analysis timepoint iteration limit. The default is 10.
ITL5=x	Resets the transient analysis total iteration limit. The default is 5000.
LIMTIM=x	Resets the time limit for the program. The default is 2000 ms.
LIMPTS=x	Resets the total number of points that can be computed in a dc, ac, or transient analysis. The default is 201.
LVLCOD=x	If x is two, then machine code for the matrix solution will be generated. Otherwise, no machine code is generated. The default for LVLCOD is one.
LVLTIM=x	If x is one, the iteration timestep control is used. If x is two, the truncation-error timestep is used. The default for LVLTIM is two.

APPENDIX 4

THE SPICE2 PROGRAM

The SPICE2 program contains approximately 7500 FORTRAN IV and COMPASS assembly language statements. SPICE2 was written for use on the CDC 6400 computer that is available through the Compter Center at the University of California, Berkeley. This computer operates under the CALIDOSCOPE operating system that was developed by the staff of the Computer Center. The FORTRAN portions of SPICE2 are complied with the RUNW.2 compiler, and the COMPASS portions of the program are assembled with the COMPASS 1.1 assembler.

The SPICE2 program consists of the seven major overlays that are shown in Fig. A4.1. The root segment, SPICE2, is the main control portion of the program. The root calls in the various overlays in the sequence that is required for the specific simulation. In addition, any subroutines that are referenced by more than one of the SPICE2 overlays are included in the root. Any system-supplied subroutines also are included in the root.

A4.1. The SPICE2 Root

The root segment of SPICE2 contains the main control loop of the program. This control loop is illustrated by the flowchart that is given in Fig. A4.2. The program begins by initializing some program constants and reading the job title card. If an end-of-file is encountered on the input file, the program terminates. Otherwise, the READIN overlay is called to read the remainder of the input file. The READIN program stops reading after a .END card or an input end-of-file is encountered.

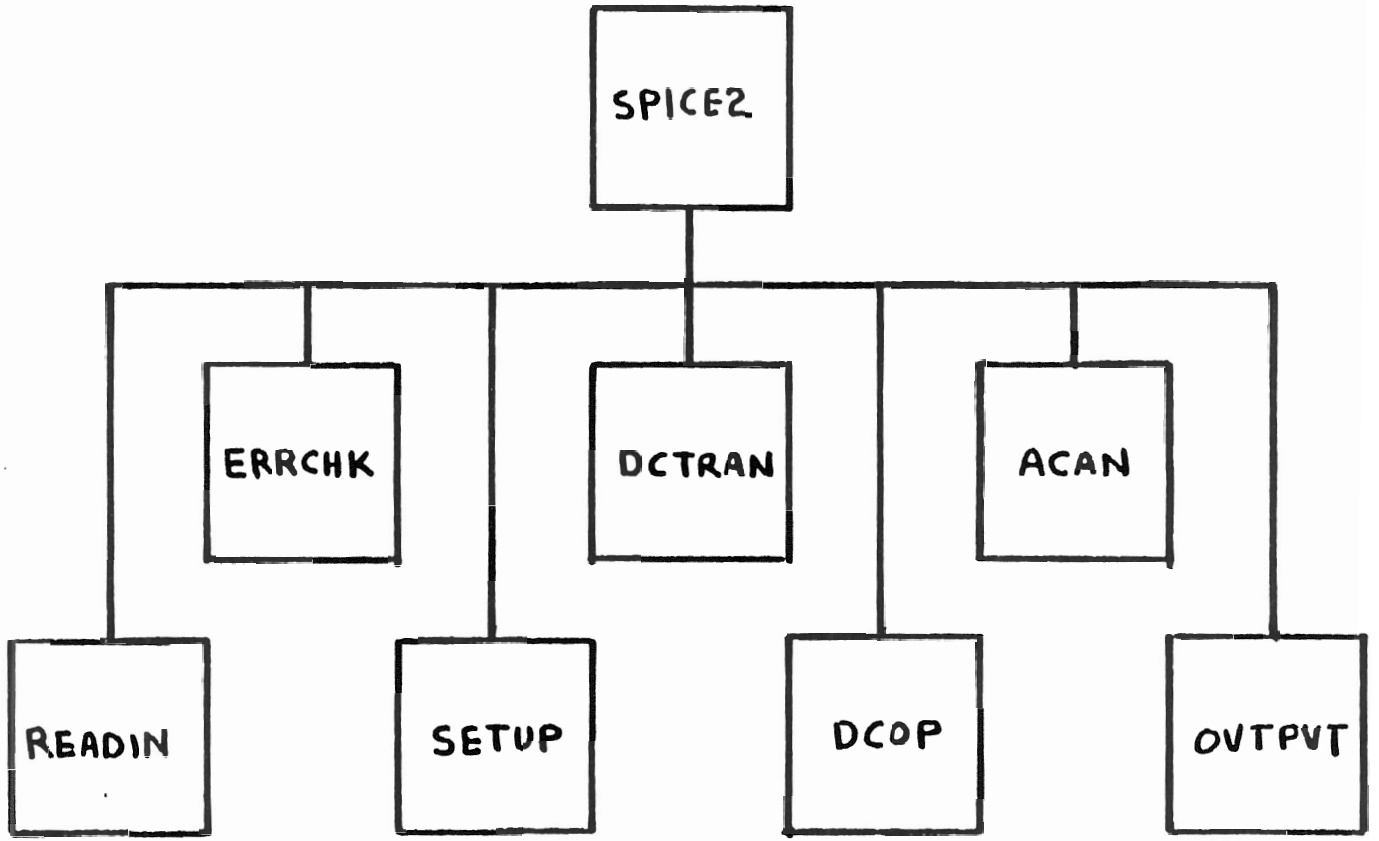


Fig. A4.1. Block Diagram of SPICE2.

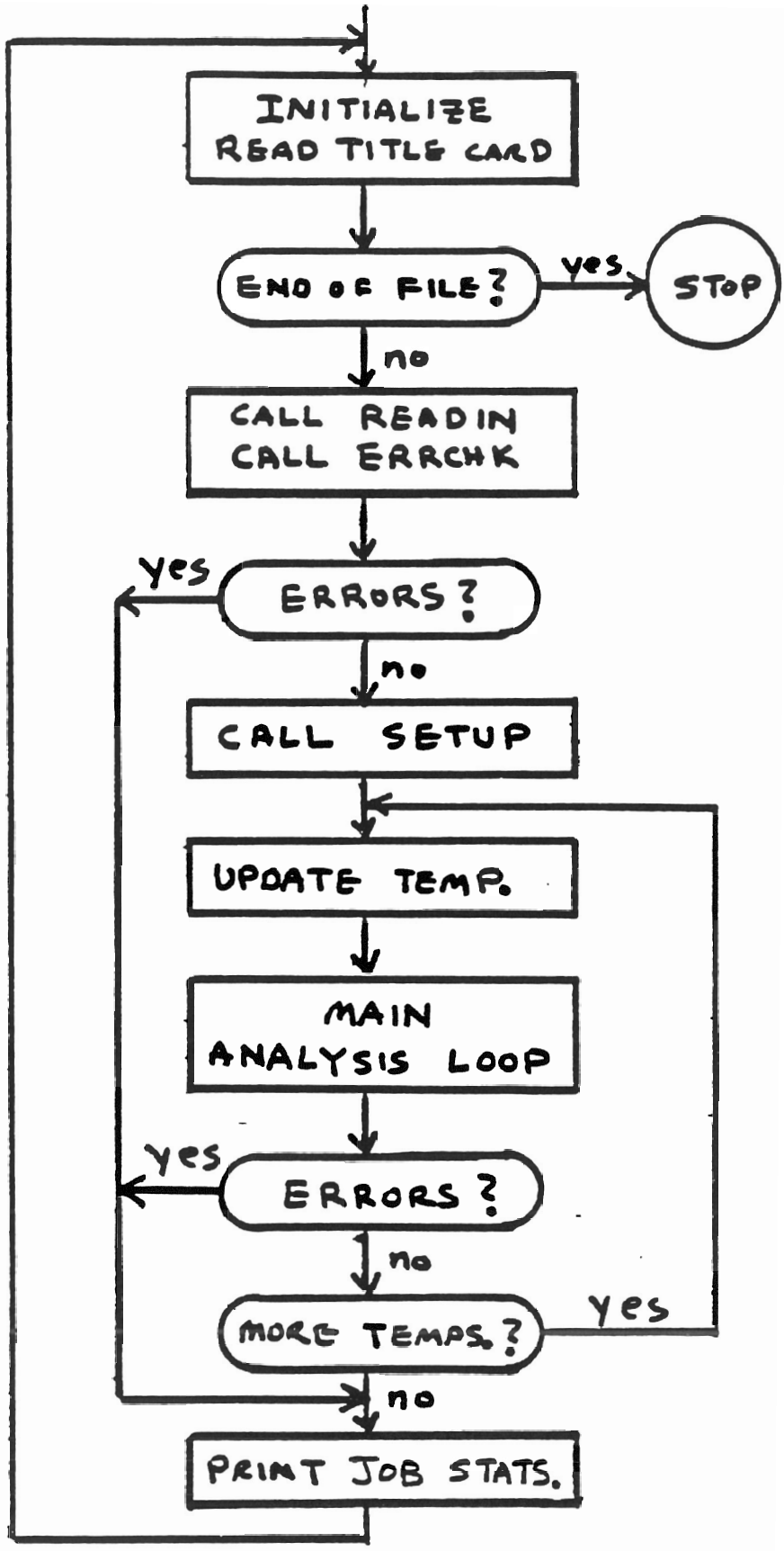


Fig. A4.2. Main Control Flowchart of SPICE2.

As the READIN overlay reads the input file, the circuit data structure is constructed. This data structure consists of a set of linked lists that define each circuit element, each device model, and each output variable. In addition, common block variables are set to indicate the types of analysis that have been requested as well as the simulation control parameters that have been specified.

After the READIN overlay has executed successfully, the ERRCHK overlay is called to check the circuit description for common user errors. In addition, the ERRCHK overlay also prints the circuit listing, the device model parameter listing, and the node table.

Once the READIN and the ERRCHK overlays have executed, the circuit data structure is self-consistent, and preparation for circuit analysis can begin. The SETUP overlay is called to construct the integer pointer structure that is used by both the DCTRAN overlay and the ACAN overlay. In addition, the SETUP overlay generates executable machine code to solve the linearized dc or transient circuit equations.

After the SETUP overlay has executed successfully, the circuit analysis can proceed. As shown in Fig. A4.2, the main analysis loop is repeated for each of the user-specified temperatures. After the main analysis loop is finished for each temperature, the program prints the job statistics for the job and reads the next input deck.

A4.2. The Main Analysis Loop

The main analysis loop consists of three parts: the dc transfer curve loop, which is shown in Fig. A4.3; the dc operating point

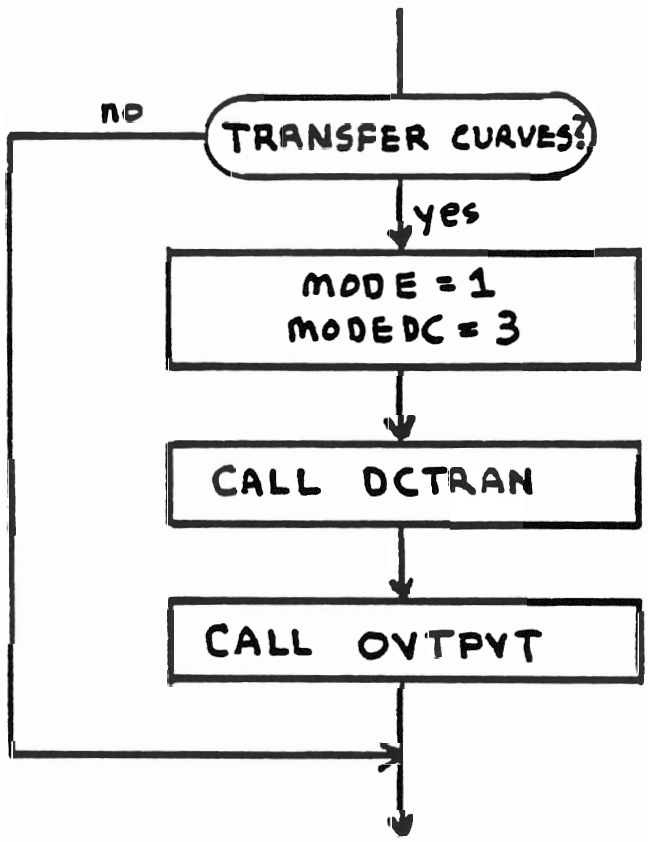


Fig. A4.3. DC Transfer-Curve Flowchart.

and ac analysis loop which is shown in Fig. A4.4; and the transient analysis loop which is shown in Fig. A4.5. These loops are executed in succession, one after the other.

The dc transfer curve loop, which is shown in Fig. A4.3, first checks to insure that a dc transfer curve is requested. The flags MODE and MODEDC are set to one and three, respectively, and the DCTRAN overlay is called to perform the dc transfer curve analysis. The OVTPVT overlay is then called to construct the tabular listings and line-printer plots that are requested for the dc transfer curve analysis.

The dc operating point and ac analysis is illustrated in Fig. A4.4. First, the loop checks if a dc operating point has been requested. If not, the dc operating point analysis will be performed only if an ac analysis of a nonlinear circuit is requested, since, for this case, the linearized device model parameters are a necessary prerequisite for the ac analysis. Once it is established that a dc operating point is necessary, the flags MODE and MODEDC are set to one, and the DCTRAN overlay is called to compute the dc operating point. The DCOP overlay is then called to print the linearized device model parameters.

If requested, the ac analsysis is performed next by setting the flag MODE to 3 and calling the ACAN overlay. After the ac analysis has been obtained, the OVTPVT overlay is called to generate the tabular listings and line-printer plots that are requested in conjunction with the ac analysis.

The final analysis loop, transient analysis, is shown in Fig. A4.5. First, the loop checks that a transient analysis is

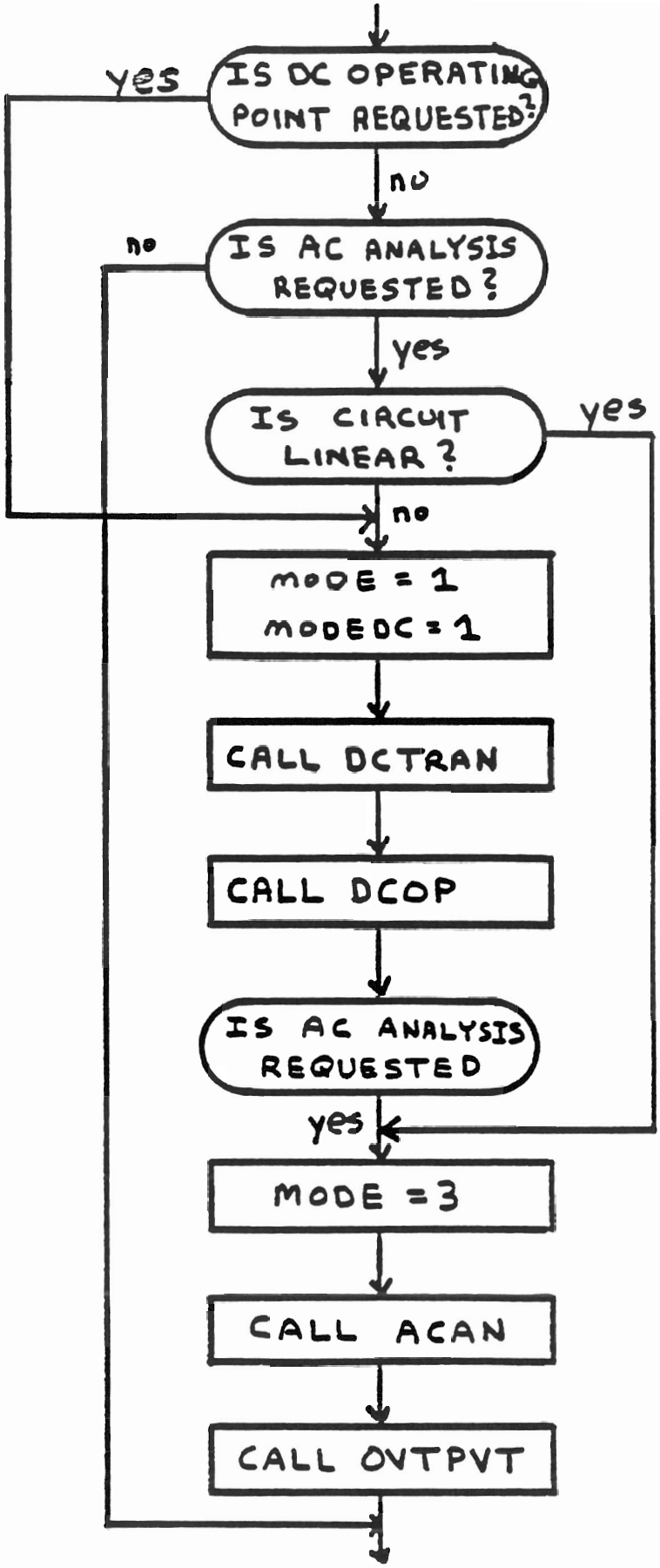


Fig. A4.4. DC Operating Point and AC Analysis Flowchart.

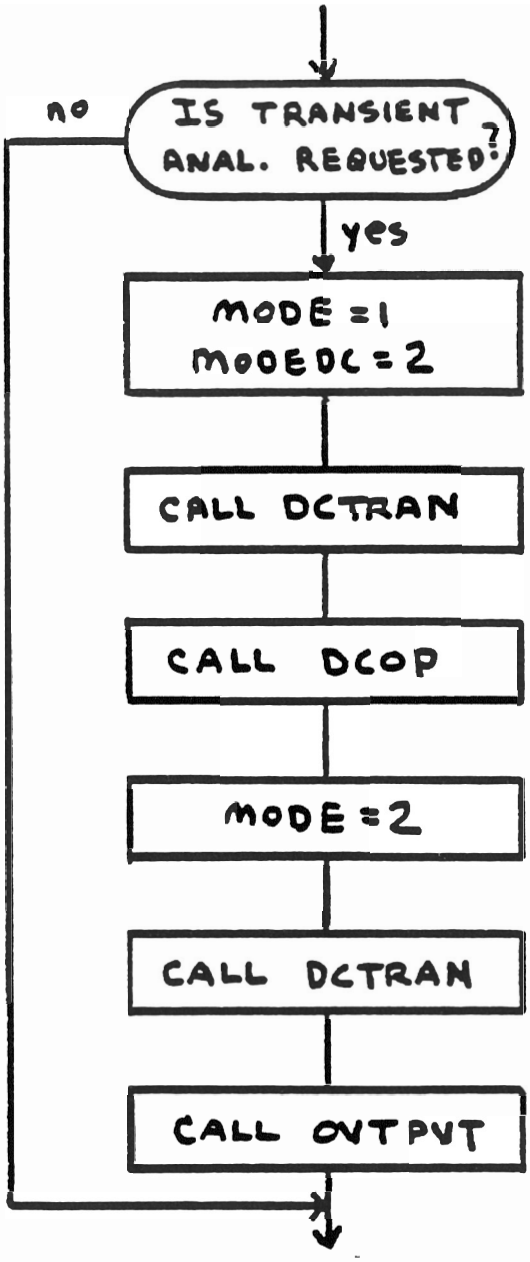


Fig. A4.5. Transient Analysis Flowchart.

requested. Then, the flags MODE and MODEDC are set to one and two, respectively, and the DCTRAN overlay is called to determine the transient initial conditions. Next, the DCOP overlay is invoked to print the nonlinear device operating points. Finally, the flag MODE is set to two and the DCTRAN overlay is called again to determine the transient analysis. The OVTPVT overlay then is called to generate the tabular listings and line-printer plots that are requested. In addition, the OVTPVT overlay will perform the Fourier analysis of time-domain waveforms after the transient analysis has been determined.

A4.3. Memory Management

Central memory is allocated dynamically in SPICE2 to correspond to the needs of the simulation. The memory map of the SPICE2 program is shown in Fig. A4.6. The SPICE root starts at the beginning of the user field length, that is, at low core. Each of the seven overlays, in turn, begin at the end of the SPICE root. Blank common begins at the end of the longest overlay and extends to the end of the user field length, that is, to high core. The total field length that is required by the program is therefore the sum of the lengths of the SPICE root, the longest overlay, and the blank common.

The CALIDOSCOPE operating system allows a program to vary the user field length at run time. If the field length is increased, the additional memory is appended to high core. Hence, increasing field length is equivalent to increasing the blank common area that is available to the program. When loaded with the CALIDOSCOPE

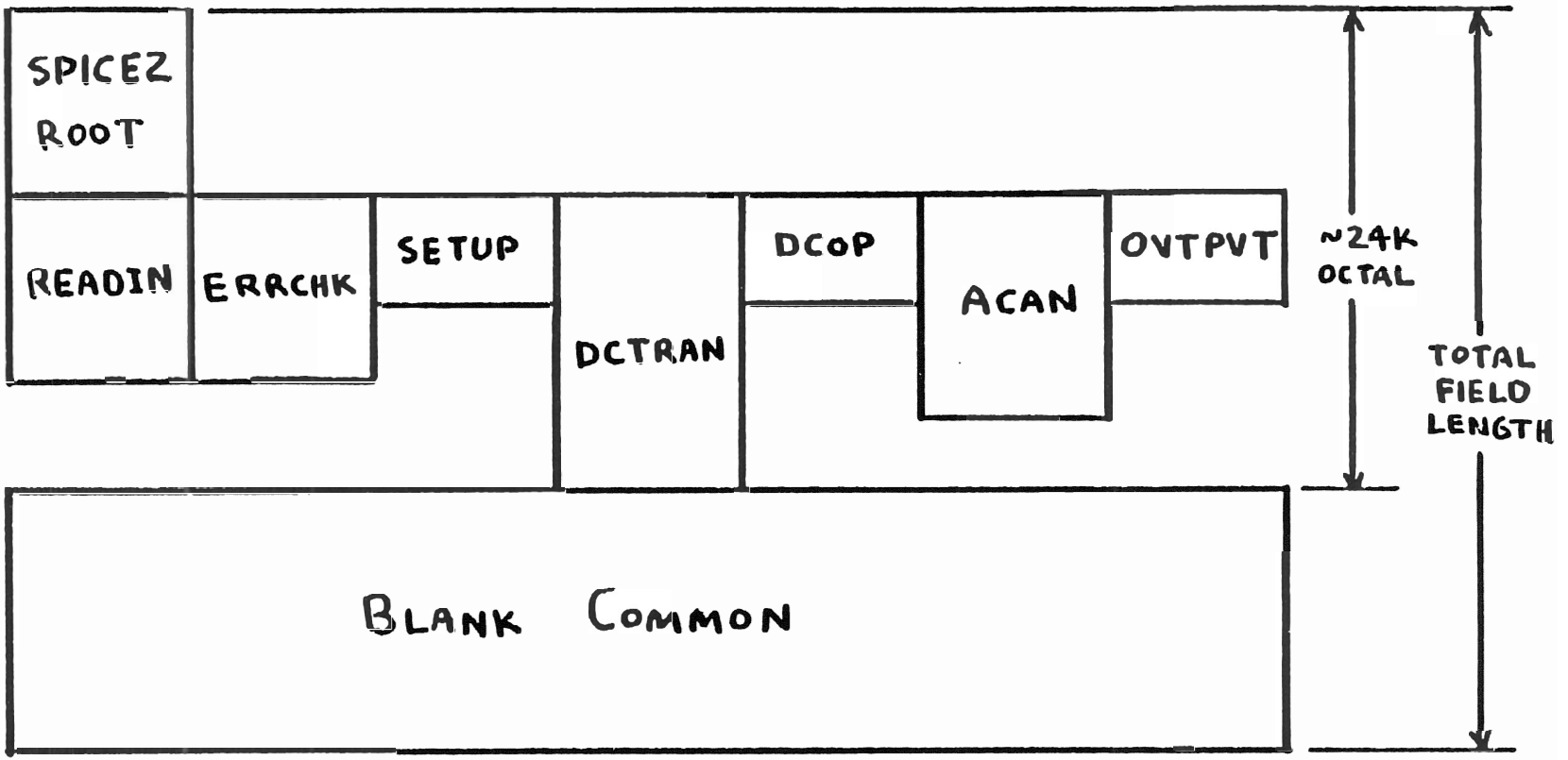


Fig. A4.6. Memory Map for SPICE2.

loader (CLDR2.3), the SPICE root requires approximately 16K octal. The longest overlay, DCTRAN, requires 6K octal, and the program begins with a blank common length of 4K octal. Therefore, the SPICE2 program requires a minimum of 30K octal to load on the CDC 6400 at the Computer Center. Since a total of 120K octal of central memory is available, the blank common length can vary between 4K octal and 74K octal. In other words, the blank common space ranges from 2048 decimal words to 30720 decimal words.

The SPICE2 data structure resides in two arrays, NODPLC and VALUE. These arrays are equivalenced such that NODPLC(1) shares the same address as VALUE(1). Both of these arrays reside in blank common. Therefore, the maximum dimension of these arrays is controlled by the amount of central memory that is assigned to the program.

All lists, tables, and arrays that are used by SPICE2 are stored in the NODPLC and VALUE arrays and are referenced by pointers that are stored in the labeled common blocks of the program. The remainder of the labeled common block variables are flags, constants, and simulation controls.

The pointer system for any array is simple and direct. For example, the JUNODE array contains the user node numbers and is of length NUNODS (the number of user nodes). If JUNODE were declared an integer array, then the members of JUNODE would be stored in JUNODE(1), JUNODE(2), ... JUNODE(NUNODS). In SPICE2, the variable JUNODE is the pointer to the NODPLC array, and the members of the JUNODE array are stored in NODPLC(JUNODE+1), NODPLC(JUNODE+2), ... , NODPLC(JUNODE+NUNODS). This method of array allocation is used consistently throughout the entire SPICE2 program.

A4.4. The READIN Overlay

The READIN overlay consists of the subroutines READIN, RUNCON, CARD, and FIND. This overlay reads the input file, checks each line of input for syntax, and constructs the data-structure for the circuit. The data-structure that defines the circuit is a set of linked lists.

Each call to subroutine CARD returns the contents of the next line of input, together with the contents of any continuation lines. The input line is broken into fields, with each field containing a decimal number (a number field) or an alphanumeric name (a name field). The first field of the card determines whether the card is an element card or a control card, since every control card must begin with a period (.). If the card is an element card or a .MODEL card, then the card is processed by the READIN subroutine. If the card is a control card, then the card is processed by the RUNCON subroutine.

The FIND subroutine handles all matters of the linked-list structure. Whenever a specific element of a linked-list (that is, a bead) is desired, the FIND subroutine is called to locate the beas. If a bead is to be added to the data-structure, then the FIND subroutine handles the linking and memory extension.

A4.5. The ERRCHK Overlay

The ERRCHK overlay consists of the subroutines ERRCHK, ELPRNT, MODCHK, and TOPCHK. These routines are executed sequentially. First, ERRCHK checks for undefined elements (as might happen, for example, if a coupled inductor element references an inductor that

is not defined). Then, ERRCHK constructs an ordered list of the user node numbers, processes the sources in the circuits, processes the coupled inductors in the circuit, and establishes a list of source breakpoints for later use in the transient analysis.

The ELPRNT subroutine prints the circuit summary. The MODCHK subroutine assigns default values for model parameters that have not been specified, prints the summary of device model parameters, performs some additional amount of model parameter processing (such as inverting series resistances), and , finally, reserves any internal device nodes that are generated by nonzero device ohmic resistances. The TOPCHK subroutine constructs the node table, prints the node table, and checks that every node in the circuit has at least two elements connected to it and that every node in the circuit has a dc path to ground.

A4.6. The SETUP Overlay

The SETUP overlay consists of the subroutines SETUP, MATPTR, RESERV, REORDR, MATLOC, INDEX, CODGEN, and MINS. The SETUP routine is the control routine for the overlay. First, the subroutine MATPTR is called to determine which the nodal admittance, matrix coefficients that are nonzero. MATPTR cylces through the circuit elements, and, with the aid of the RESERV subroutine, establishes the initial integer pointer structure of nonzero Y-matrix terms. Next, the pointer structure is reordered to minimize fill-in, and the fill-in that does occur is added to the pointers. The reordering is accomplished by the REORDR subroutine. Finally, the MATLOC subroutine computes and stores the pointer locations that are used

to construct the Y-matrix. The INDEX routine is used by MATLOC to determine the location of a specific Y-matrix coefficient. The CODGEN and MINS subroutines then generate the executable machine code that solves the circuit equations for dc analysis and transient analysis.

A4.7. The DCTRAN Overlay

The DCTRAN overlay is the largest and most complicated overlay in the SPICE2 program. This overlay performs the dc operating point analysis, the transient initial-condition analysis, the dc transfer-curve analysis, and the transient analysis. This overlay contains the following subroutines: DCTRAN, TRUNC, TERR, SORUPD, ITER8, DCDCMP, DCSOL, CODEXC, LOAD, INTGR8, DIODE, BJT, JUNCT, FETLIM, JFET, and MOSFET.

The DCTRAN subroutine is the control program for the DCTRAN overlay. Within this subprogram is the controlling logic for the four different types of analysis that are performed by the DCTRAN overlay. The flowgraph for the dc operating point analysis is given in Fig. A4.7. The flowgraph for the transient initial timepoint solution is given in Fig. A4.8. The flowgraph for the dc transfer curve analysis is given in Fig. A4.9. Finally, the flowgraph for the transient analysis is given in Fig. A4.10.

The logic for a dc operating point is uncomplicated. As shown in Fig. A4.7, the program first evaluates the source values at time zero. Next, the INITF flag is set to two, and the subroutine ITER8 is called to determine the dc solution iteratively. Providing the solution converges, the INITF flag is reset to four, and the device

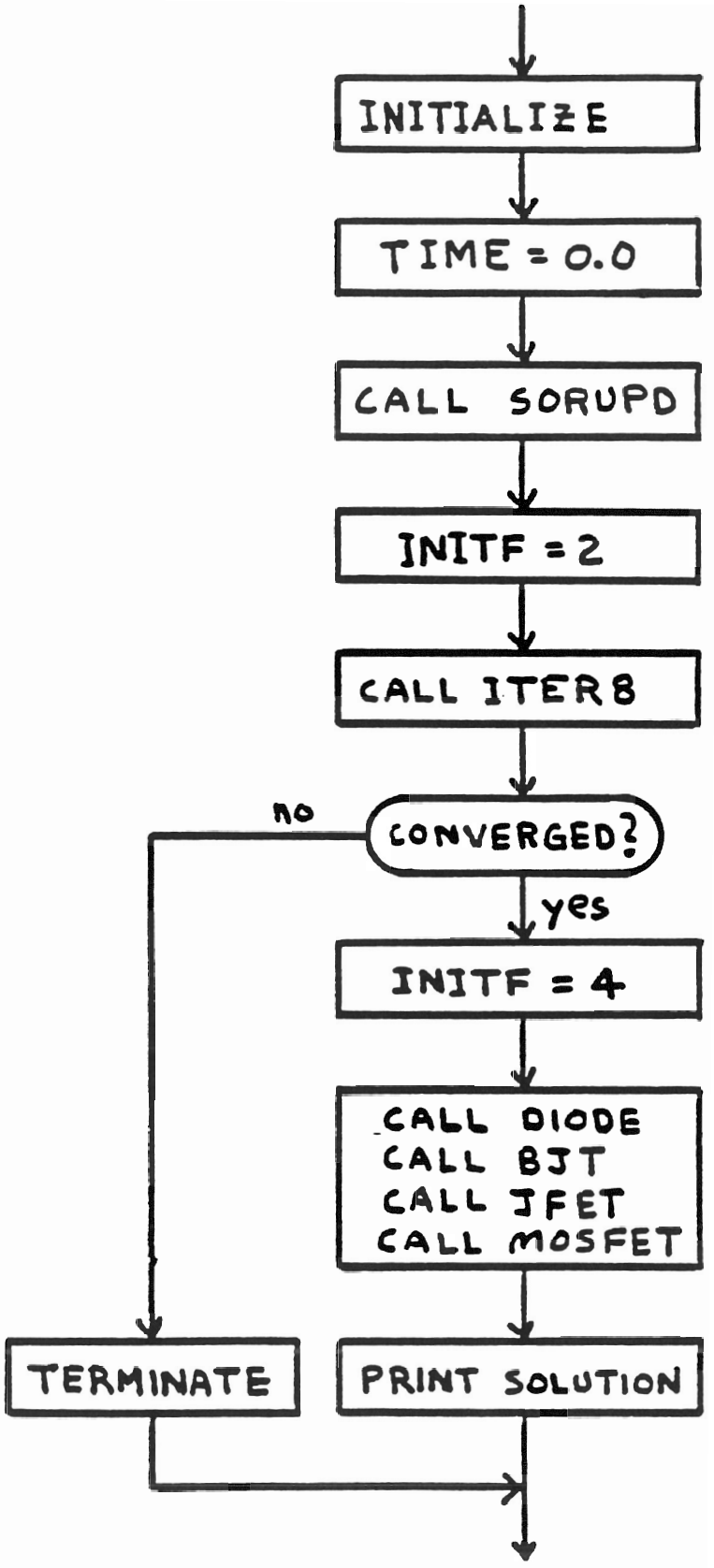


Fig. A4.7. DC Operating-Point Flowchart.

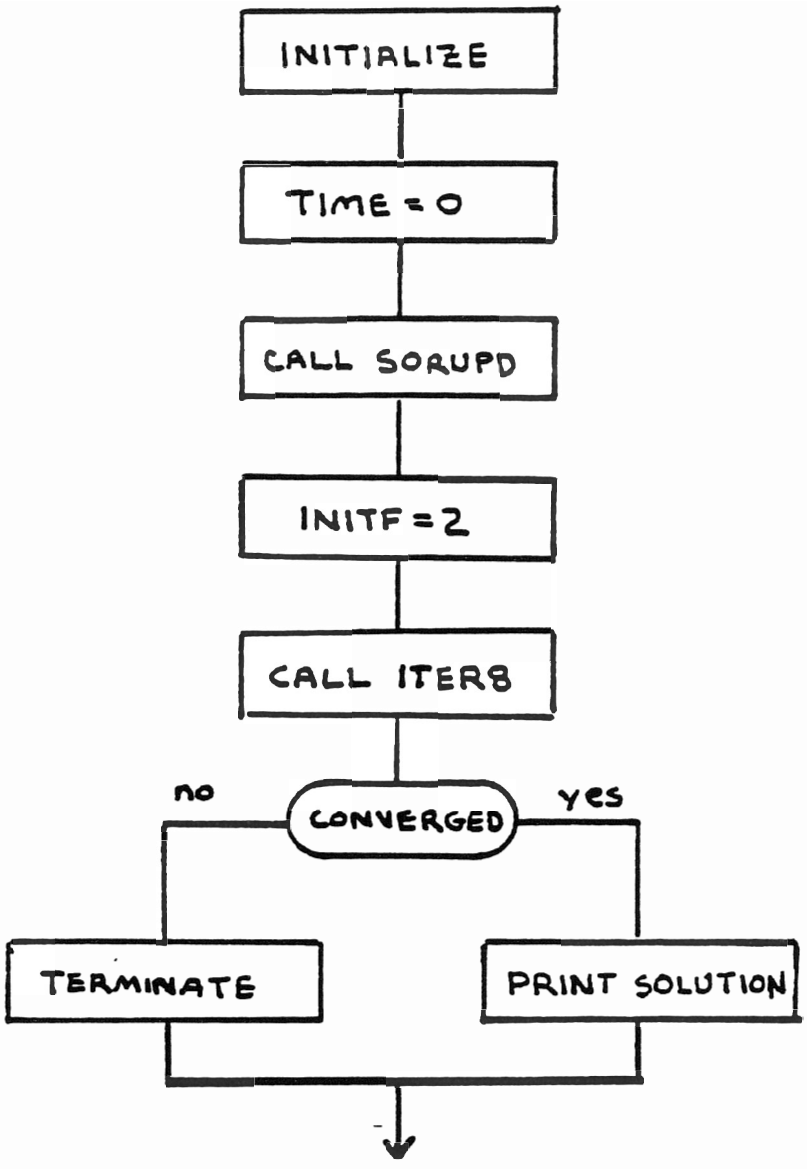


Fig. A4.8. Transient Initial Condition Flowchart.

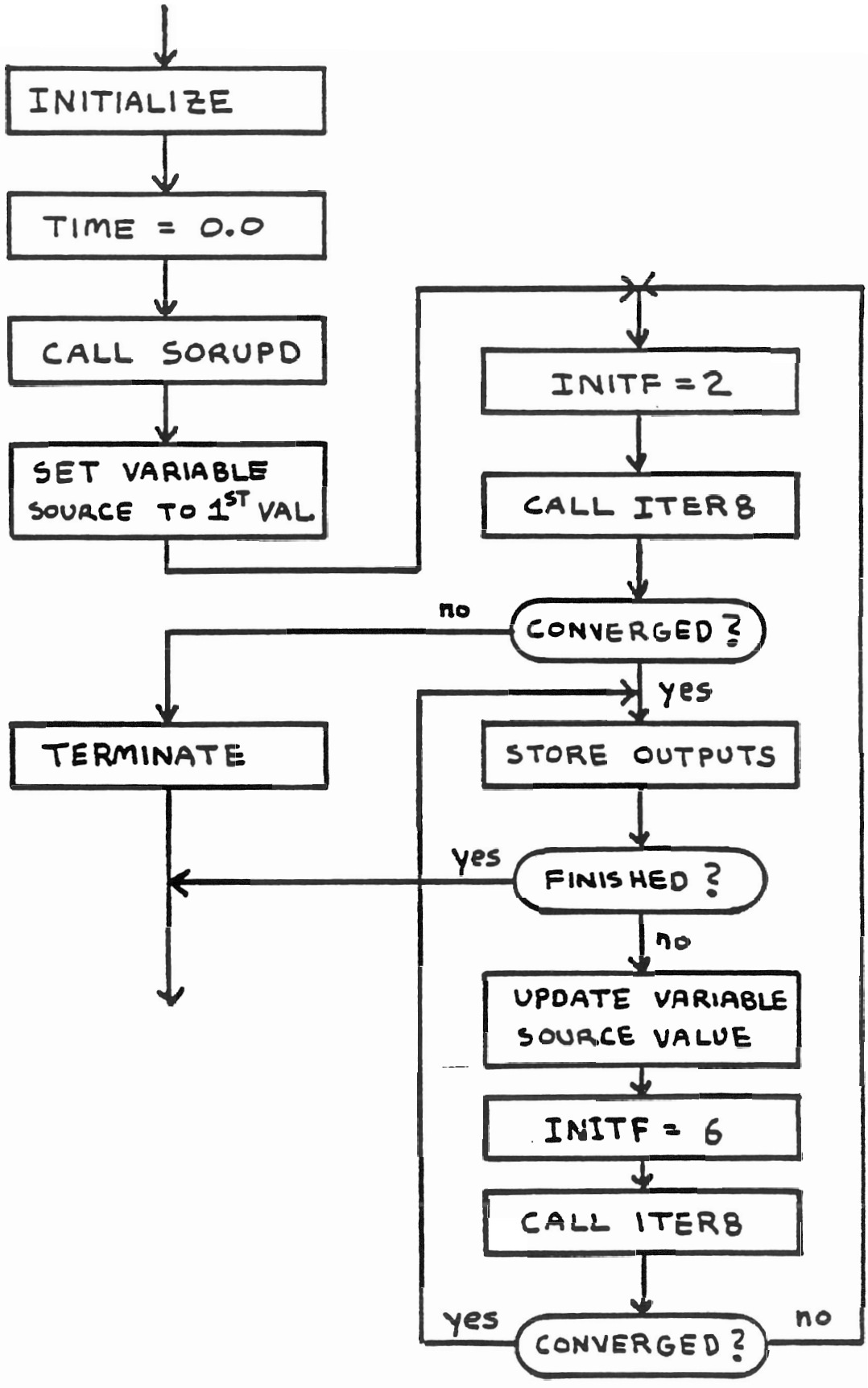


Fig. A4.9. DC Transfer-Curve Flowchart.

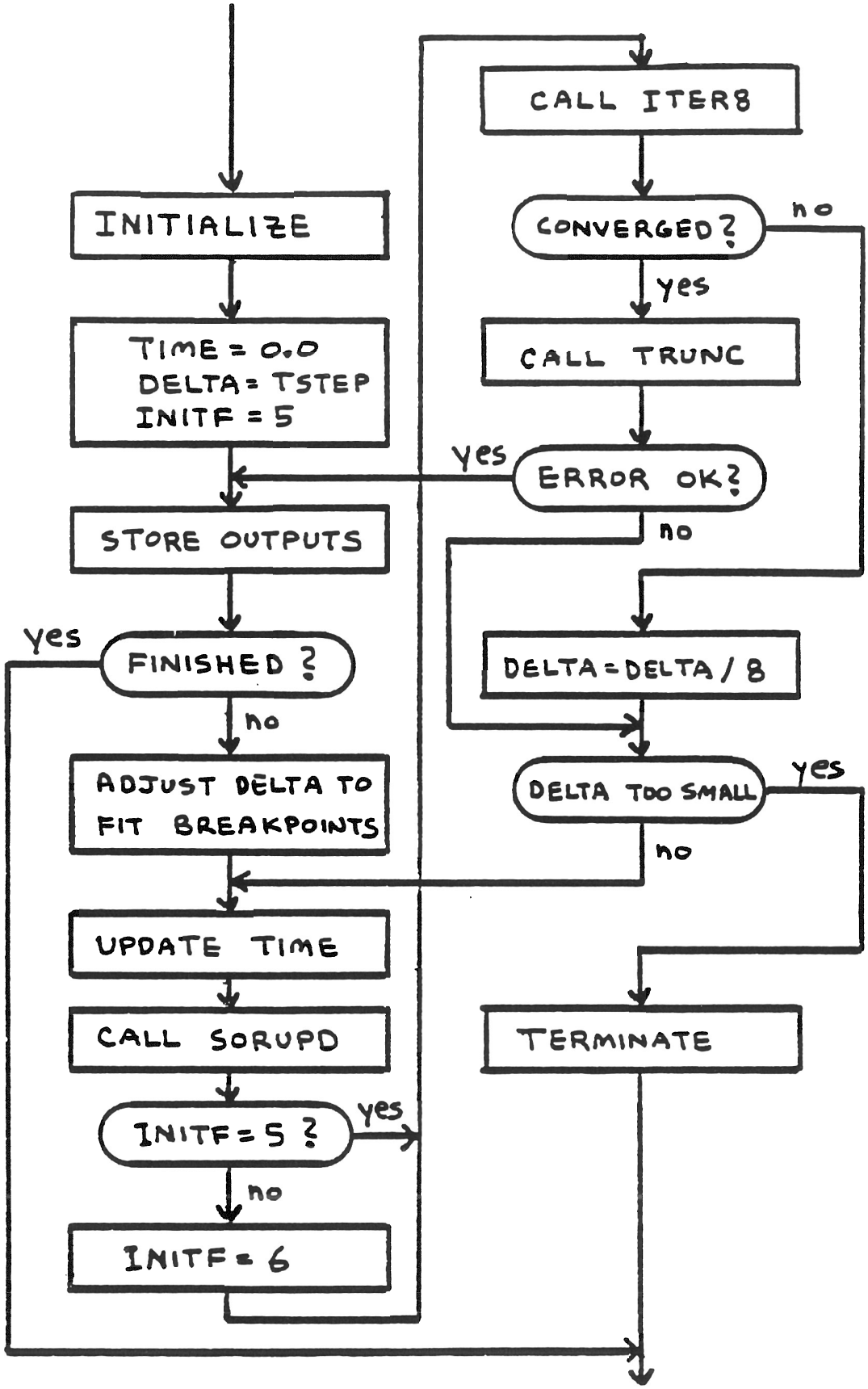


Fig. A4.10. Transient Analysis Flowchart.

model routines DIODE, BJT, JFET, and MOSFET, are called to compute the linearized, small-signal value of the nonlinear capacitors in the device models. The node voltages for the solution then are printed, and the DCTRAN overlay exits back to the root.

The case of the solution of the initial transient timepoint is simpler than the case of the dc operating point. As shown in Fig. A4.8, the difference between these two single point ac analyses is that, for the case of the transient initial timepoint, the linearized capacitance values are not computed since they are of no use in the transient analysis.

The dc transfer analysis requires a multiple-point analysis and the subsequent storage of output variables. As shown in Fig. A4.9, the dc transfer-curve analysis begins in an identical fashion as the dc operating-point analysis. Specifically, the variable source value is set to the first value, all other source values are set to their time-zero value, and ITER8 is called with INITF equal to two to obtain the first point solution. The values of the specified output variables are stored in central memory after this transfer point has been computed. Then, the variable source value is incremented, the INITF flag is set to six. This process then continues until the required number of dc transfer-curve points have been computed.

The transient analysis loop is shown in Fig. A4.10. As this figure shows, the transient analysis and the dc transfer-curve analysis loops are very similar. The time-zero solution has already been computed before this loop is entered. Therefore, the first step in the analysis is the storage of the time zero output variables.

The first timepoint is then computed by calling ITER8 with the flag INITF set to 5. Each subsequent timepoint is computed with the flag INITF set to 6. If the timepoint does not converge, the timestep DELTA is reduced by a factor of 8 and the timepoint is re-attempted. If the timepoint does converge, then the subroutine TRUNC is called to determine the new timestep that is consistent with the truncation error tolerance that has been specified. As long as the timestep is larger than the timestep that presently is in effect, then the timespoint is accepted and the output variables are stored. If the truncation error is too large, as indicated by a timestep that is smaller than the present timestep, then the timepoint is rejected and re-attempted with the smaller timestep. This process continues until the time interval that is specified by the user has been covered.

All four of the analysis procedures that are implemented in the DCTRAN overlay use the subroutine ITER8 to determine the solution of the circuit equations. The flowchart for the ITER8 subroutine is shown in Fig. A4.11. The solution commences as follows. First, the subroutine LOAD is called. This subroutine constructs the linearized system of Nodal equations for the specific iterate. The contributions to the Y-matrix from nonlinear devices are computed separately in the subroutines DIODE, BJT, JFET, and MOSFET. The NONCON flag is the number of nonconvergent nonlinearities in the circuit. This flag is incremented during the LOAD procedure, since the convergence is checked as an integral part of the linearization procedure. Immediately following the construction of the Y matrix and the forcing vector, a check on NONCON is performed

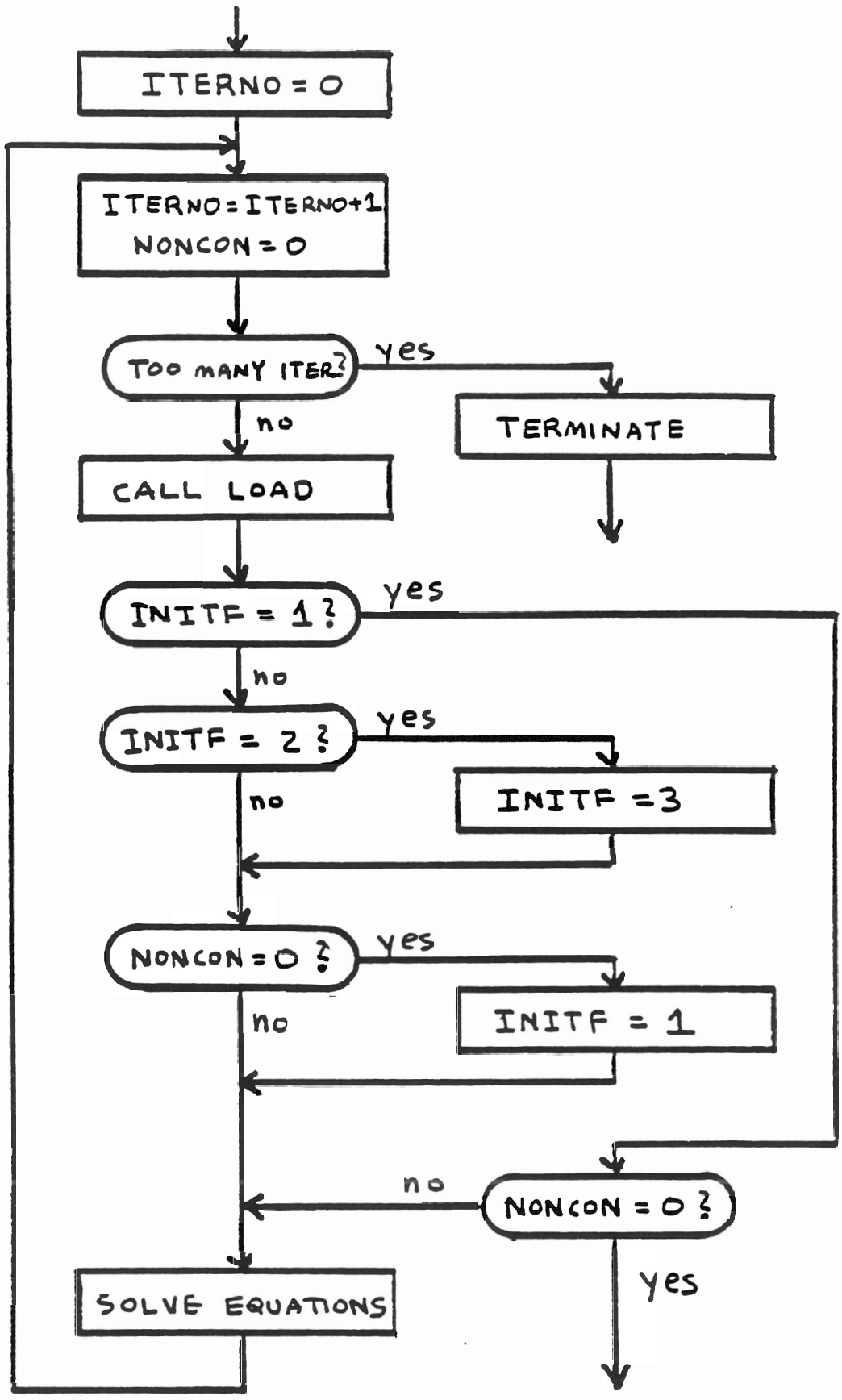


Fig. A4.11. ITER8 Flowchart.

to determine if the solution has converged. If convergence is obtained, then the ITER8 subroutine is exited. Otherwise, the system of linearized equations is solved for the next iterate value, and the system of circuit equations is linearized again. This process continues until the solutions converge or until the number of iterations exceeds a preset number.

The solution of the linearized equations is determined by the assembly language routines DCDCMP and DCSOL. Alternatively, if executable machine code has been generated, the solution of the equations is determined by executing the machine code; this, in turn, is performed by calling the assembly language routine CODEXC.

A4.8. The DCOP Overlay

At present, the DCOP overlay contains only one subroutine, DCOP, and has the single task of printing the nonlinear device operating points and the linearized device model parameters. Eventually, this overlay will be expanded to include the capability of dc transfer-function analysis and dc, small-signal sensitivity analysis as is included in the SPICE1 program.

A4.9. The ACAN Overlay

The ACAN overlay in SPICE2 determines the small-signal frequency-domain analysis. This overlay consists of four subroutines: ACAN, ACDCMP, ACSOL, and ACLOAD. The ACLOAD subprogram constructs the complex system of linear equations for each frequency point. This system of equations is then solved with the assembly language routines ACDCMP and ACSOL. Eventually, the ACAN overlay will be expanded to include both noise analysis and distortion analysis.

A4.10. The OVTPVT Overlay

The final overlay in the SPICE2 program is the OVTPVT overlay. This overlay consists of three subroutines: OVTPVT, PLOT, and FOURAN. The OVTPVT subroutine is the controlling program of this overlay. This subroutine interpolates the output variables to match the specified printing increment and generates the tabular listings that are specified by the used. The PLOT subroutine is called to generate the line-printer plots in the simulation. Finally, the FOURAN subroutine performs the Fourier analysis for transient analysis.

A4.11. The Linked List Structure in SPICE2

The circuit element data for SPICE2 is stored in a set of linked lists. This linked-list structure was chosen over a comparable table structure because the linked-list structure can be constructed with comparable efficiency yet no central memory is wasted by the "holes" that normally accompany a table structure.

The first entry in any linked-list bead is a pointer to the next bead in the list. If the bead is the last bead on the list, then this pointer is set to zero. In SPICE2, it was decided to maintain the NODPLC and VALUE arrays independently, so the second entry in the linked-list bead (in NODPLC) is a pointer, LOCV, to the beginning of the bead list in the VALUE.

The resistor bead structure that is shown in Table A4.1 is a example of the linked list structure. The first entry in the NODPLC array, NPTR, is an integer pointer to the next resistor bead and is zero if no more resistor beads are present. The next

entry, LOCV, is a pointer to the start of the VALUE entries for this bead. The third and fourth entries are the two resistor node numbers, and the fifth and sixth entries are the two matrix pointers that are referenced by the resistor. The first VALUE entry is the alphanumeric name of the resistor, and the second VALUE entry is the resistor value.

The structure of all of the SPICE2 beads is similar to the resistor bead that has just been described. The first NODPLC entry always is the address of the next bead, and the second NODPLC entry always is the address of the VALUE entries. The third through mth NODPLC entries contain the m-2 integer variables for the bead. The first value entry always is the alphanumeric name of the bead. The second through nth VALUE entries contain the n-1 real variables for the bead.

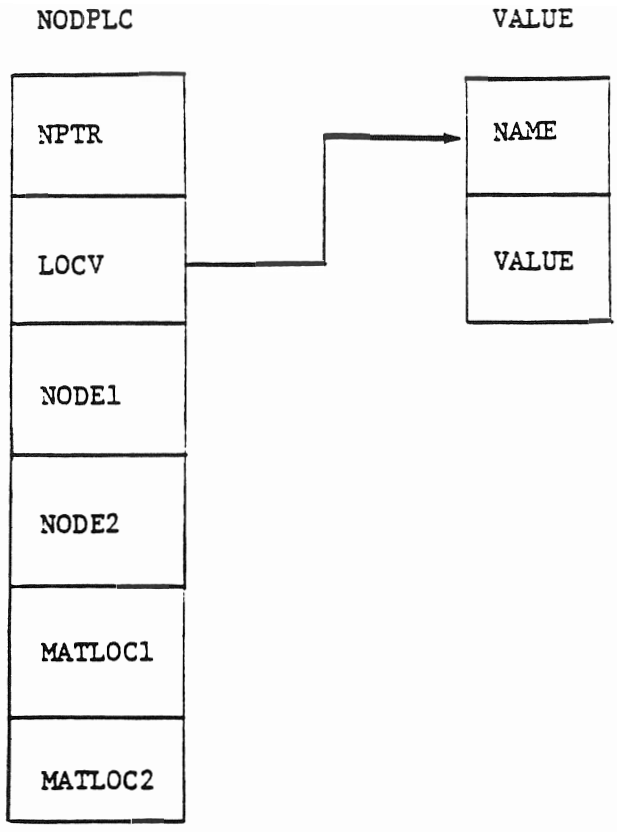


Table A4.1. The Linked-List Bead for a Resistor Element.

APPENDIX 5

LIST OF THE SPICE2 PROGRAM

Persons who wish to obtain either the SPICE1 program or the SPICE2 program should request updated versions of these programs from Doris R. Simpson, ERL Publications Office, 433 Cory Hall, University of California, Berkeley, California 94720. The Electronics Research Laboratory charges only a nominal handling charge for any of the programs that it has available.

REFERENCES

[1] L. W. Nagel and D. O. Pederson, "Simulation Program with Integrated Circuit Emphasis," Proc. Sixteenth Midwest Symposium on Circuit Theory, Waterloo, Canada, April 12, 1973; available as Memorandum No. ERL-M382, Electronics Research Laboratory, University of California, Berkeley.

[2] L. W. Nagel and R. A. Rohrer, "Computer Analysis of Nonlinear Circuits, Excluding Radiation (CANCER)," IEEE J. Solid-State Circuits, vol. SC-6, August 1971, pp. 166-182.

[3] R. A. Rohrer, L. W. Nagel, R. G. Meyer, and L. Weber, "CANCER: Computer Analysis of Nonlinear Circuits, Excluding Radiation," Proc. 1971 IEEE International Solid-State Circuits Conference, Philadelphia, February 18, 1971, pp. 124-125.

[4] C. L. Searle, A. R. Boothroyd, E. J. Angelo, Jr., P. E. Gray, and D. O. Pederson, Elementary Circuit Properties of Transistors, New York: J. Wiley and Sons, 1964.

[5] S. W. Director and R. A. Rohrer, "The Generalized Adjoint Network and Network Sensitivities," IEEE Trans. on Circuit Theory, vol. CT-16, August 1969, pp. 318-323.

[6] S. W. Director and R. A. Rohrer, "Automated Network Design -- The Frequency Domain Case," IEEE Trans. on Circuit Theory, vol. CT-16, August 1969, pp. 330-337.

[7] C. A. Desoer and E. S. Kuh, Basic Circuit Theory, New York: McGraw-Hill, 1969.

[8] R. A. Rohrer, L. W. Nagel, R. G. Meyer, and L. Weber, "Computationally Efficient Electronic Circuit Noise Calculations," IEEE J. Solid-State Circuits, vol. SC-6, August 1971, pp. 204-213.

[9] R. G. Meyer, L. W. Nagel, and S. K. Lui, "Computer Simulation of 1/f Noise Performance of Electronic Circuits," IEEE J. Solid-State Circuits, vol. SC-8, June 1973, pp. 237-240.

[10] S. W. Director and R. A. Rohrer, "Interreciprocity and its Implications," Proc. International Symposium Network Theory, Belgrad, September 1968, pp. 11-30.

[11] Y. L. Kuo, "Distortion Analysis of Bipolar Transistor Circuits," IEEE Trans. on Circuit Theory, vol. CT-20, November 1973, pp. 709-717.

[12] S. H. Chisholm and L. W. Nagel, "Efficient Computer Simulation of Distortion in Electronic Circuits," IEEE Trans. on Circuit Theory, vol. CT-20, November 1973, pp. 742-745.

[13] S. Narayanan, "Transistor Distortion Analysis Using Volterra Series Representation," Bell Syst Tech J, May-June 1967.

[14] T. E. Idleman, F. S. Jenkins, W. J. McCalla, and D. O. Pederson, "SLIC---A Simulator for Linear Integrated Circuits," IEEE J. Solid-State Circuits, vol. SC-6, August 1971, pp. 188-204.

[15] W. J. McCalla, "Computer-Aided Design of Integrated Bandpass Amplifiers," University of California, Berkeley Ph.D. dissertation, June 15, 1972.

[16] T. J. Aprille, Jr., and T. N. Trick, "Steady State Analysis of Nonlinear Circuits and Periodic Inputs," Proc. IEEE, vol. 60, January 1972, pp. 108-114.

[17] T. J. Aprille, Jr., and T. N. Trick, "A Computer Algorithm to Determine the Steady-State Response of Nonlinear Oscillators," IEEE Trans. Circuit Theory, vol. CT-19, July 1972, pp. 354-360.

[18] F. R. Colon and T. N. Trick, "Fast Periodic Steady-State Analysis for Large Signal Electronic Circuits," IEEE J. Solid-State Circuits, vol. SC-8, August 1973, pp. 260-269.

[19] F. S. Jenkins and S. P. Fan, "TIME--A Nonlinear DC and Time-Domain Circuit-Simulation Program," IEEE J. Solid-State Circuits, vol. SC-6, August 1971, pp. 192-188.

[20] I. A. Cermak and D. B. Kirby, "Nonlinear Circuits and Statistical Design," Bell Syst. Tech. J., vol. 50, April 1971, pp. 1173-1195.

[21] C. L. Semmelman, E. D. Walsh, and G. T. Daryanani, "Linear Circuits and Statistical Design," Bell Syst. Tech. J., vol 50, April 1971, pp. 1149-1171.

[22] P. Balaban, B. J. Karafin, and D. B. Snyder, "A Monte-Carlo Analysis of the Integrated, Single-Substrate, RC, Touch-Tone Oscillator," Bell Syst. Tech. J., vol. 50, April 1971, pp. 1263-1291.

[23] S. W. Director, "Survey of Circuit-Oriented Optimization Techniques," IEEE Trans. on Circuit Theory, vol. CT-18, January 1971, pp. 3-10.

[24] R. A. Rohrer, "Fully Automated Network Design by Digital Computer: Preliminary Considerations," Proc. IEEE, vol. 55, November 1967, pp. 1929-1939.

[25] R. I. Dowell and R. A. Rohrer, "Automated Design of Biasing Circuits," IEEE Trans. on Circuit Theory, vol. CT-18, January 1971, pp. 85-89.

[26] B. A. Wooley, "Automated Design of DC-Coupled Monolithic Broad-Band Amplifiers," IEEE J. Solid-State Circuits, vol. SC-6, February 1971, pp. 24-34.

[27] S. W. Director, W. A. Bristol, and A. J. Broderson, "Fabrication-Based Optimization of Linear Integrated Circuits," IEEE Trans. on Circuit Theory, vol. CT-20, November 1973, pp. 690-697.

[28] "ASTAP General Information Manual (GH20-1271-0)," IBM Corp., Mechanicsburg, Pa.

[29] H. Qassemzadeh, and T. R. Scott, "Algorithms for ASTAP---A Network Analysis Program," IEEE Trans. on Circuit Theory, vol. CT-20, November 1973, pp. 628-634.

[30] "ISPICE Reference Guide," Form 968-2, National CSS, Inc., Norwalk, Connecticut, 1974.

[31] "ECAP II Application Description Manual (GH20-0983)," IBM Corp., Mechanicsburg, Pa.

[32] F. H. Branin, G. R. Hogsett, R. L. Lunde, and L. E. Kugel, ECAP II---A New Electronic Circuit Analysis Program," IEEE J. Solid-State Circuits, vol. SC-6, August 1971, pp. 146-165.

[33] A. F. Malmberg, "NET-2 Network Analysis Program," Harry Diamond Laboratories, Washington, D.C., 1970.

[34] B. Dembart and L. Milliman, "CIRCUS-2, A Digital Computer Program for Transient Analysis of Electronic Circuits," Harry Diamond Laboratories, Washington, D.C., July 1971.

[35] W. J. McCalla and D. O. Pederson, "Elements of Computer-Aided Circuit Analysis," IEEE Trans. on Circuit Theory, vol. CT-18, January 1971, pp. 14-26.

[36] R. D. Berry, "An Optimal Ordering of Electronic Circuit Equations for a Sparse Matrix Solution," IEEE Trans. on Circuit Theory, vol. CT-18, January 1971, pp. 40-50.

[37] L. O. Chua and P. M. Lin, "Computer-Aided Analysis of Electronic Circuits: Algorithms and Computational Techniques. Englewood Cliff, New Jersey: Prentiss-Hall, 1975.

[38] D. A. Calahan, Computer-Aided Network Design. New York: McGraw-Hill, 1968.

[39] E. D. Johnson, C. T. Kleiner, L. R. McMurray, E. L. Steele, and F. A. Vassallo, "Transient Radiation Analysis by Computer Program (TRAC)," Autonetics Division, North American Rockwell Corp., Anaheim, California, Harry Diamond Lab, Tech. Rep., June 1968.

[40] "1620 Electronic Circuit Analysis Program (ECAP) 1620-EE-02x," IBM Application Program File H20-0170-1, 1965.

[41] H. Shichman, "Integration System of a Nonlinear Network Analysis Program," IEEE Trans. on Circuit Theory, vol. CT-17 August 1970, pp. 378-386.

[42] H. Shichman, "Computation of DC Solutions for Bipolar Transistor Networks," IEEE Trans. on Circuit Theory, vol. CT-16, November 1969, pp. 460-466.

[43] H. W. Mathers, S. R. Sedore, and J. R. Seuts, "Automated Digital Computer Program for Determining Responses of Electronic Circuits to Transient Nuclear Radiation (SCEPTRE)," IBM Space Guidance Center, Oswego, N. Y., IBM File 66-928-611, February 1967.

[44] A. F. Malmberg, F. L. Cornwell, and F. N. Hofer, "NET-1 Network Analysis Program," Los Alamos Scientific Lab., Los Alamos, New Mexico, Rep LA-3119, 7090/94 version, August 1964.

[45] L. D. Milliman, W. A. Massena, and R. H. Dickhaut, "CIRCUS--- A Digital Computer Program for Transient Analysis of Electronic

Circuits," Harry Diamond Lab., Washington, D.C., Rep. AD-346-1, January 1967.

[46] G. D. Hachtel, R. K. Brayton, and F. G. Gustavson, "The Sparse Tableau Approach to Network Analysis and Design," IEEE Trans. on Circuit Theory, vol. CT-18, January 1971, pp. 101-118.

[47] C. W. Ho, A. E. Ruehli, and P. A. Brennan, "The Modified Nodal Approach to Network Analysis," Proc. 1974 International Symposium on Circuits and Systems, San Francisco, April 1974, pp. 505-509.

[48] P. M. Russo and R. A. Rohrer, "The Tree-Link Approach to the Time Domain Analysis of a Class of Nonlinear Networks," IEEE Trans. on Circuit Theory, vol. CT-18, May 1971, pp. 400-403.

[49] A. Ralston, A First Course in Numerical Analysis. New York: McGraw-Hill, 1965.

[50] E. Isaacson and H. B. Keller, Analysis of Numerical Methods. New York: Wiley, 1966.

[51] H. M. Markowitz, "The Elimination Form of the Inverse and Its Application to Linear Programming," Management Sci., vol. 3, April 1957, pp. 255-269.

[52] M. Nakhla, K. Singhal, and J. Vlach, "An Optimal Pivoting Order for the Solution of Sparse Systems of Equations," IEEE Trans. on Circuits Systems, vol. CAS-21, March 1974, pp. 222-225.

[53] H. Y. Hsieh and M. S. Ghausi, "A Probabilistic Approach to Optimal Pivoting and Prediction of Fill-in for Random Sparse Matrices," IEEE Trans. on Circuit Theory, vol. CT-19, July 1972, pp. 329-336.

[54] R. Fletcher and M. J. D. Powell, "A Rapidly Convergent Descent Method for Minimization," Comput. J., vol. 6, June 1963, pp. 163-168.

[55] D. Agnew, "Iterative Improvement of Network Solutions," Proc. Sixteenth Midwest Symposium on Circuit Theory, Waterloo, Ontario, April 1973.

[56] W. A. Schuppan, "A Comparison of the Gaussian Elimination Method and an Optimization Technique Using the Adjoint Network for the Analysis of Linear Circuits," University of Illinois at Urbana-Champaign, MS Thesis, 1973.

[57] R. A. Rohrer, "Successive Secants in the Solution of Nonlinear Equations," AMS, 1970, pp. 103-112.

[58] W. H. Kao, "Comparison of Quasi-Newton Methods for the DC Analysis of Electronic Circuits," University of Illinois at Urbana-Champaign, MS Thesis, 1972.

[59] W. J. McCalla and W. G. Howard, Jr., "BIAS-3---A Program for the Nonlinear DC Analysis of Bipolar Transistor Circuits," IEEE J. Solid-State Circuits, vol. SC-6, February 1971, pp. 14-19

[60] D. F. Davidenko, "On a New Method of Numerical Solution of Systems of Nonlinear Equations," Dokl Akad Nauk SSSR, vol. 88, 1953, pp. 601-602.

[61] C. G. Broyden, "A New Method of Solving Nonlinear Simultaneous Equations," Comput J., vol. 12, February 1969, pp. 94-99.

[62] D. L. Scharfetter, private communication.

[63] C. G. Dahlquist, "A Special Stability Problem for Linear Multistep Methods," BIT, vol. 3, 1963, pp. 27-43.

[64] C. W. Gear, Numerical Initial Value Problems in Ordinary Differential Equations. Englewood Cliffs, New Jersey: Prentice-Hall, 1971.

[65] M. L. Blostein, "The Transient Analysis of Nonlinear Electronic Networks," IEEE Short Course Notes, 735C06, 1973.

[66] C. W. Gear, "Numerical Integration of Stiff Ordinary Equations," University of Illinois at Urbana-Champaign, Report 221, January 20, 1967.

[67] C. W. Gear, "The Control of Parameters in the Automatic Integration of Ordinary Differential Equations," University of Illinois at Urbana-Champaign, File No. 757, May 17, 1968.

[68] W. Liniger and R. A. Willoughby, "Efficient Integration Methods for Stiff Systems of Ordinary Differential Equations," SIAM J Numerical Anal., vol. 7, March 1970, pp. 47-66.

[69] A. Nordsieck, "On Numerical Integration of Ordinary Differential Equations," Math Comp., vol. 16, 1962, pp. 22-49.

[70] R. K. Brayton, F. G. Gustavson, and G. D. Hachtel, "A New Efficient Algorithm for Solving Differential-Algebraic Systems Using Implicit Backward Differentiation Formulas," Proc. IEEE, vol. 60, January 1972, pp. 98-108.

[71] H. H. Rosenbrock, "Some General Implicit Processes for the Numerical Solution of Differential Equations," Computer J., vol. 5, January 1963, pp. 329-330.

[72] R. H. Allen, "Numerically Stable Explicit Integration Techniques Using a Linearized Runge Kutta Extension," Report No. 39, Information Sciences Laboratory, Boeing Scientific Research Laboratories, October 1969.

[73] K. Miller, "Diagonally Implicit Runge-Kutta Methods for Partial Differential Equations and Stiff Ordinary Differential Equations," to be published.

[74] K. Miller, Math 228A Notes, Dept. of Mathematics, University of California, Berkeley, Fall 1973.

[75] G. D. Hachtel, private communication.

[76] D. A. Hodges, private communication.

[77] R. N. Hall, "Power Rectifiers and Transistors," Proc. IRE, vol. 40, November 1952, pp. 1512-1519.

[78] J. J. Ebers and J. L. Moll, "Large Signal Behavior of Junction Transistors," Proc. IRE, vol. 42, December 1954, pp. 1761-1772.

[79] J. M. Early, "Effects of Space-Charge Layer Widening in Junction Transistors," Proc. IRE, vol. 46, November 1952, pp. 1401-1406.

[80] H. K. Gummel, "A Charge Control Relation for Bipolar Transistors," Bell Syst. Tech. J., vol. 49, January 1970, pp. 115-120.

[81] H. K. Gummel and H. C. Poon, "An Integral Charge Control Model for Bipolar Transistors," Bell Syst. Tech. J., vol 49, May/June 1970, pp. 827-852.

[82] R. I. Dowell, "Automated Biasing of Integrated Circuits," University of California, Berkeley, Doctoral Dissertation, March 1972.

[83] R. Beaufory and J. J. Sparks, "The Junction Transistor as a Charge-Controlled Device," ATE J., vol. 13, Oct. 1957, pp. 310-324.

[84] H. K. Gummel, "Measurement of the Number of Impurities in the Base Layer of a Transistor," Proc. IRE, vol. 49, April 1961, pp. 384-388.

[85] C. T. Sah, R. N. Noyce, and W. Shockley, "Carrier Generation and Recombination in p-n Junctions and p-n Junction Characteristics," Proc. IRE, vol. 45, September 1957, pp. 1228-1243.

[86] W. M. Webster, "On the Variation of Junction Transistor Current Gain Amplification Factor with Emitter Current," Proc. IRE, vol. 42, June 1954, pp. 914-920.

[87] W. M. C. Sansen and R. G. Meyer, "Characterization and Measurement of the Base and Emitter Resistances of Bipolar Transistors," IEEE J. Solid-State Circuits, vol. SC-7, Dec. 1972, pp. 492-498.

[88] C. T. Kirk, Jr., "A Theory of Transistor Cutoff Frequency (f_T) at High Current Densities," IRE Trans. Elec. Dev., vol. ED-9, March 1962, pp. 164-174.

[89] R. J. Whittier and D. A. Tremere, "Current Gain and Cutoff Frequency Falloff at High Currents," IEEE Trans. Elec. Dev., vol. ED-16, January 1969, pp. 39-57.

[90] H. Schichman and D. A. Hodges, "Modeling and Simulation of Insulated-Gate Field-Effect Transistor Switching Circuits," IEEE J. Solid-State Circuits, vol. SC-3, September 1968, pp. 285-289.

[91] D. Frohman-Bentchskowsky and L. Vadasz, "Computer-Aided Design and Characterization of Digital MOS Integrated Circuits," IEEE J. Solid-State Circuits, vol. SC-4, April 1969, pp. 57-64.

[92] F. S. Jenkins, E. R. Lane, W. W. Lattin, and W. S. Richardson, "MOS-Device Modeling for Computer Implementation," IEEE Trans. on Circuit Theory, vol. CT-20, November 1973, pp. 649-658.

[93] H. C. Poon, L. D. Yau, R. L. Johnston, and D. Beecham, "DC Model for Short-Channel IGFET's," Tech. Dig. Int. Electron Device, Washington, December 3, 1973, pp. 156-159.

[94] T. K. Young and R. W. Dutton, "MSINC - A Modular Simulator for Integrated Nonlinear Circuits with MOS Model Example," Proc. 1974 International Symposium on Circuits and Systems, San Francisco, April 1974, pp. 720-724.

[95] R. G. Meyer, private communication.