



This is to certify that the  
thesis entitled

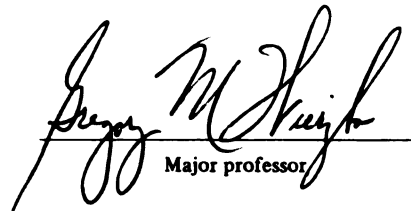
SSPICE: A SYMBOLIC ANALYZER OF  
LINEAR ACTIVE CIRCUITS

presented by

Anupam Srivastava

has been accepted towards fulfillment  
of the requirements for

Master's degree in Electrical  
Engineering

  
Major professor

Date Aug. 3, 1990

**LIBRARY**  
**Michigan State**  
**University**

PLACE IN RETURN BOX to remove this checkout from your record.  
 TO AVOID FINES return on or before date due.

DATE DUE	DATE DUE	DATE DUE
SEP 28 1991 1989	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____
_____	_____	_____

MSU Is An Affirmative Action/Equal Opportunity Institution

c:\circ\datedue.pm3-p.

**SSPICE: A SYMBOLIC ANALYZER OF LINEAR ACTIVE CIRCUITS**

**By**

**Anupam Srivastava**

**A THESIS**

**Submitted to**

**Michigan State University**

**in partial fulfillment of the requirements**

**for the degree of**

**MASTER OF SCIENCE**

**Department of Electrical Engineering**

**1990**

# ABSTRACT

## SSPICE: A SYMBOLIC ANALYZER OF LINEAR ACTIVE CIRCUITS

By

Anupam Srivastava

Understanding the small signal behaviour of a circuit is a key element of circuit design. A variety of circuit simulators are available in the market that can simulate the frequency response of a circuit, given the value of the circuit parameters. However, deeper insight into circuit behaviour can be gained if circuit parameters are treated as symbols instead of as numbers. The final result is then symbolic instead of numeric and explicitly embodies the role of each circuit parameter in the behaviour of the circuit. Symbolic computation involves manipulating strings of symbols and often generates large, unwieldy results. Thus, some measure of numerical analysis must accompany symbolic analysis so that only the dominant symbolic terms are generated. This process, termed symbolic approximation, helps keep symbolic results tractable and preserves most of their semantics. Sspice is a software tool, developed at MSU, that automates the symbolic analysis of linear active circuits.

**Dedicated To Poochie, The Triple A Fraternity, and my main men - Ponniah, VJ, and Inder**

## **ACKNOWLEDGEMENTS**

**I am deeply grateful to Shoba Krishnan and Sriman Ramabhadran for the selfless manner in which they helped me complete this thesis on time.**

**I am personally indebted to Dr. Wierzba for his constant support and guidance.**

## TABLE OF CONTENTS

LIST OF TABLES.....	vii
LIST OF FIGURES.....	viii
I. The symbolic analysis paradigm and the Sspice solution.....	1
1.1 Introduction to symbolic analysis.....	1
1.2 The Sspice approach.....	2
II. Theoretical Basis of Sspice.....	5
2.1 Introduction to nodal analysis.....	5
2.2 Passive network analysis.....	6
2.3 Introduction to nullators and norators.....	7
2.4 Active network analysis.....	8
2.5 Active filter design using symbolic transfer functions.....	11
2.5.1 Low pass filter function.....	11
2.5.2 High pass filter function.....	12
2.5.3 Band pass filter function.....	12
2.5.4 Notch filter function.....	13
2.5.5 All pass filter function.....	13
2.6 Estimating deviance in filter parameters.....	14



III. Numerical Evaluation and Symbolic Approximation.....	16
3.1 Drawbacks of symbolic computation.....	16
3.2 Mitigating problems through symbolic approximation.....	16
IV. Operational Specifics and Salient Features.....	19
4.1 Behavioral description of Sspice modules.....	19
4.2 Overall operation.....	19
4.3 Error detection and recovery.....	20
4.4 User friendly features.....	21
4.5 Operational description of Sspice modules.....	21
4.5.1 Module I: Forming nodal equations.....	21
4.5.2 Module II: Solving nodal equations.....	22
4.5.2.1 Example.....	25
4.5.3 Module III: Filter function identification.....	26
4.5.4 Module IV: Estimating deviance due to non ideal effects.....	26
4.6 Conclusion.....	27
V. Future Directions.....	28
APPENDIX A. Input Format And Modelling of Sspice Elements.....	30
APPENDIX B. Illustrative Example: State Variable Active Filter.....	44
APPENDIX C. Illustrative Example: CMOS Op-Amp.....	49
APPENDIX D. Interactive structure of Sspice.....	58
D1.1 Key to interactive prompts.....	60
BIBLIOGRAPHY.....	61

## LIST OF TABLES

4.1. Execution trace of determinant algorithm.....	25
B1. Sspice Input file SVAF.CIR.....	45
B2. Sspice Output file SVAF.FUN.....	46
B3. Sspice Output file SVAF.ERR .....	48
C1. PSpice/Sspice Input File.....	50
C2. PSpice Output File Bias Point Values.....	50
C3. Sspice Output file CMOS.DET.....	52
C4. Sspice Approximated Output file CMOS.DET.....	54

## LIST OF FIGURES

1. Resistor model and input specification.....	30
2. Capacitor model and input specification.....	30
3. Independent current source model (normalized to one).....	30
4. Nullator Norator pair.....	31
5. DC current source model (open circuit).....	31
6. AC voltage source model (normalized to one).....	31
7. DC Voltage source model (short circuit).....	32
8. Source transformed AC current source model.....	32
9. Source transformed AC current source model.....	32
10. Inductor model.....	33
11. Voltage Controlled Current Source model (Gain = $G_M$ ).....	33
12. Coupling Capacitor Model (short circuit).....	33
13. Voltage controlled voltage source model (Gain = $A$ ).....	34
14. Current controlled current source (Gain = $B$ ).....	34
15. Current controlled voltage source model (Gain = $R_M$ ).....	35
16. Voltage controlled voltage source (Gain = $1/K$ ).....	35
17. Lowest level diode model (short circuit).....	36
18. Low frequency diode model .....	36
19. High frequency diode model.....	36
20. Simplest hybrid pi model of a BJT (Gain = $G_M$ ).....	37
21. Simplest hybrid pi model of a BJT (Gain = $B$ ).....	37
22. Low frequency hybrid pi model of a BJT .....	38
23. High frequency hybrid pi model of a BJT.....	38
24. Lowest level JFET model.....	39
25. Low frequency JFET model.....	39
26. High frequency JFET model.....	40
27. Lowest level MOSFET model.....	40
28. Low frequency MOSFET model.....	41
29. High frequency MOSFET model.....	41
30. Ideal operational amplifier model.....	42
31. Non-ideal op amp model.....	42

32. Operational transconductance amplifier model .....	43
33. Transformer model with N:1 turns ratio.....	43
B1. State Variable Active Filter.....	44
C1. CMOS op-amp.....	49
C2. Open loop gain circuit.....	49
C3. Open loop magnitude comparison of PSpice and Sspice.....	55
C4. Open loop phase response comparison of PSpice and Sspice.....	56
D1. Interactive structure of Module I.....	58
D2. Interactive structure of Module II.....	59

## CHAPTER I

### **The symbolic analysis paradigm and the Sspice solution**

#### 1.1 INTRODUCTION TO SYMBOLIC ANALYSIS

Sspice is an acronym for SymbolicSPICE. It is a software tool for small signal, symbolic analysis of linear active circuits. Symbolic analysis is superior to behavioral simulation of circuits from a circuit designer's perspective since it records the identity of each circuit parameter in the final result. A symbolic result is dependent only on the circuit structure and is independent of any numerical instance of circuit parameters. All instances of a circuit that can be derived by varying its parameter values are encapsulated in one symbolic result. Behavioral simulation, on the other hand, analyzes one instance of a circuit. Moreover, it combines the contributions of various circuit parameters into a numeric result. The number itself is anonymous, and offers no insight into its pedigree. Behavioral simulation is simply a numerical evaluation of the symbolic result for a given set of parameter values. Its use is more appropriate during the verification and testing stage of circuit design. The design stage requires a deeper understanding of circuit semantics and symbolic analysis is means to this end.

Circuit analysis involves building a set of equations from a circuit's description and then solving for the unknown currents and voltages in the circuit. It is possible to build circuit equations by mere inspection and without any prior knowledge of circuit theory. Sspice models each circuit element by a well defined set of primitive elements - resistors, capacitors, independent current sources, nullators, and norators. It produces a matrix formulation of circuit equations and solves for the unknowns by computing the determinant of the corresponding matrices. In fact, Sspice can analyze any process that reduces to a set of linear, independent, simultaneous equations. Sspice was developed primarily for active filter design. It detects any second order filter functions in the circuit, and solves for the ideal values of filter

parameters. The non-ideal effects of operational amplifiers in a circuit can shift the filter parameters from their ideal values. Provided that these shifts are small in magnitude, it is possible to compute a symbolic estimate of errors in terms of circuit parameters. The circuit designer can then minimize these errors by assigning appropriate values to circuit parameters. Some successful results have already been found and published using this strategy[1].

A frequent criticism of symbolic analysis is that it generates large amounts of data even for moderately sized problems. One approach is to choose circuits that have simple design equations and analyze them further. However, most of the time the designer does not know the final answer but just needs a handle on the circuit parameters that dominate the behavior of the circuit. Sspice can generate an approximated symbolic result given the values of the circuit parameters. This feature is invaluable in transistor design where the circuits are complicated and the exact symbolic result is intractable. It is clear that the approximated result is dependent on parameter values but this dependence is minor and is outweighed by the gain in insight.

## 1.2 THE SSPICE APPROACH

Symbolic computation involves manipulations on strings and unfortunately, most of them involve concatenation. The symbolic result of a computation can quickly increase in size and thus, efficiency is a major issue in automatic symbolic analysis. Sspice is implemented in 'C' for this reason as well as for its rich variety of string handling functions. Sspice is available on SUN, VAX and HP platforms. A micro-computer version, featuring stricter error checking, increased robustness, and user friendliness is scheduled to be commercialized shortly. The input file format is SPICE compatible. Both the program name and the input file formats were chosen to appeal to the large community of SPICE users.

Symbolic circuit analysis can be divided into two phases: building nodal equations, and solving for unknown voltages. The resulting node voltages can be further analyzed for filter applications. A logical path for filter design would be to find if any filter functions are present in the circuit and if so, to find an estimate for shifts in ideal filter parameters in terms of circuit parameters. Sspice is divided into four

modules, each of which fulfills one of the four functions outlined above. This approach is consistent with the behavior of its predecessor SLAP[10].

An interactive approach was chosen since it offers more flexibility in element modelling, and avoids computation of unnecessary node voltages. Besides, each module can be run separately and the user has the freedom to modify results generated from previous modules. The input file format was chosen to be SPICE compatible since it is envisioned that the design process would alternate between symbolic analysis and circuit simulation. Internal files are in ASCII so that the user can access them, unhampered.

Sspice views the entire memory as a linear sequence of bytes. The chief data structure is a pointer to an array of characters that stores the symbolic determinant. Due to the nature of symbolic computation, there is no simple way to predict the size of the final answer. Space is dynamically allocated from the heap as needed and extreme care is taken to release previously held storage.

The first module generates a matrix formulation of nodal equations from the description of the circuit in the input file. It also scans the input file for circuit parameter values in case the user wants numerical evaluation and/or symbolic approximation. The second module computes the characteristic equation and any node voltages specified by the user. The characteristic equation, as well as other node voltages are obtained by computing the determinant of the corresponding matrices. Various algorithms exist for computing the determinant of a symbolic matrix. Sspice implements one that is attributed to Sannuti and Puri [9]. The basic idea is to generate all permutations of the matrix and append them with the proper sign to the answer. A matrix of dimension  $n \times n$  has  $n!$  permutations. The time taken to compute its determinant grows exponentially with the size of the matrix. Formally, the worst case time complexity of the algorithm[9] is  $O(n^3)$ . The algorithm takes advantage of the sparse nature of active network matrices and generates only the non-zero permutations. However, it fails for a non-sparse matrix. A modified version of [9] that works for both kinds of matrices is implemented in Sspice.

Symbolic computation is essentially a manipulation of strings of symbols. Each term in the symbolic answer is in "sign magnitude" form. It is implemented as a '+' or '-' character prefixed to a string of symbols, such that each pair of symbols is

separated by a '\*' character. Addition, in the symbolic context, is defined as the concatenation of two strings. Similarly, subtraction is defined as the concatenation of the second string to the first, after all the signs have been reversed in it. Multiplication of two strings is defined as an iterative concatenation of each element of one string to all the elements of the other string. A multiplication symbol ('\*') is inserted between the two strings, before the concatenation takes place. Symbolic division is hard to program since it involves keeping track of a common denominator. A limited form of division is implemented that factors the numerator and denominator and cancels any common factors between them. This operation is analogous to reducing a fraction to its simplest form.

It should be mentioned that Sspice is not limited to circuit theory applications. The determinant algorithm is generalized to solve any system of simultaneous, linearly independent equations. Control theory applications for finding the effect of noise on the system transfer function can be solved using matrix theory and finding the transfer function between each noise input and the output. Sspice has been used on an experimental basis in graduate courses on circuit theory and as a research tool for improving existing op amp based active filter circuits. New topologies are generated from a seed circuit using a technique called Op Amp Relocation[2],[6],[4]. Sspice promises to be a successful research and instructional tool due to its versatility and the nature of its problem domain.



## CHAPTER II

### Theoretical Basis of Sspice

#### 2.1 INTRODUCTION TO NODAL ANALYSIS

An  $(n+1)$  node linear circuit can be completely described by a set of 'n' simultaneous, linearly independent nodal equations. Solving these equations for the 'n' unknown voltages ((n+1)th node is chosen as a reference and assigned zero voltage) yields all the behavioral information about the circuit. An exact relationship between the input excitation and the resulting value of the voltage at a particular node is obtained if the nodal equations have purely symbolic coefficients. One strategy for solving for unknown voltages transforms the set of 'n' nodal equations into a matrix format and then uses matrix theory to find each unknown voltage. Specifically, the nodal equations result in a matrix formulation of the form:  $YV = I$ , where  $Y$  is a square matrix of dimension 'n' and contains the (symbolic) coefficients of unknown voltages,  $V$  is an 'n' length column vector containing the set of unknown voltages, and  $I$  is an 'n' length column vector consisting of (symbolic) constants on the right side of the nodal equations. The determinant of  $Y$  is the characteristic equation of the circuit. The  $i$ th unknown voltage is found by substituting the  $I$  vector for the  $i$ th row of the  $Y$  matrix, computing the (symbolic) determinant of  $Y$ , dividing it by the characteristic equation, and multiplying the result with the value of the excitation signal. It should be noted here that if the excitation signal is normalized to 1, then the  $i$ th unknown voltage is just the transfer function between the respective nodes. Linear circuit analysis can thus, be broadly divided into two phases: building nodal equations from the circuit topology, and solving for unknown voltages by computing determinants from the corresponding matrix formulation.

We can, in principle, apply Kirchoff's Current Law at every node in a circuit and obtain nodal equations from the result. However, at some nodes the current leaving or entering the node may not be constrained. The nodal equation that results from

summing the currents at that node introduces an extra unknown and is therefore useless. An instance where this situation occurs is when we try to sum the currents at the output of an operational amplifier or a voltage source. The *sum* of the currents entering the node is zero but the magnitude of the current contributed by the operational amplifier or voltage source is unknown. Therefore, we have to *selectively* apply Kirchhoff's Current Law to the nodes in the circuit in order to build a useful set of nodal equations. There is no easy way to implement such a selection procedure on a computer.

## 2.2 PASSIVE NETWORK ANALYSIS

An algorithm is desired that forms nodal equations by mere inspection of the circuit topology. Consider a circuit that consists of only independent current sources and passive elements (resistors, capacitors, and inductors). Whenever we apply KCL at a node, we sum the currents leaving the node and equate it to zero. The currents in the passive elements are determined by taking the voltage at a node and subtracting it from other node voltages in the circuit times the admittance connected between those nodes. Based on this observation, we can predict some general properties of the matrix formulation that would result from applying KCL at every node in the circuit (except ground). Foremost, the  $I$  vector entries are the sum of independent current entries leaving the node. The diagonal entries of  $Y$ , hence termed the admittance matrix, will always be positive and each off-diagonal entry negative. Moreover, zeros in the admittance matrix will occur whenever there is no connection between the corresponding pairs of nodes (resistance is infinity). The  $Y$  matrix is also symmetric since the connection between a pair of nodes is not directed. A formal algorithm for this restricted set of elements is now presented [5].

**Assumption:** The  $(n + 1)$  circuit consists of only independent current sources and resistors, capacitors, and inductors.

Step I. Select a reference node and label it 0 (ground).

Step II. Label all other nodes consecutively from 1 to  $n$ .

Step III. The matrix formulation of the nodal equations contains a column vector

$$[I] = [I_1 \ I_2 \ I_3 \dots I_n]^T$$

where the  $i$ th component,  $I_i$ , is defined as the sum of the currents flowing into the  $i$ th node from independent current sources.

Step IV. The nodal admittance matrix  $Y$  has dimensions  $n \times n$  and is written by inspection using the following rules.

$Y_{ii}$ : sum of the admittances connected to node  $i$ .

$-Y_{ij} = -Y_{ji}$ : sum of admittances connected between nodes  $i$  and  $j$ .

Step V. The nodal equations are written in matrix form

$$[I] = [Y] [V]$$

where  $[V] = [V_1 \ V_2 \ V_3 \dots V_n]^T$

is the unknown voltages vector.

## 2.3 INTRODUCTION TO NULLATORS AND NORATORS

The inclusion of voltage sources, op amps, and other active elements disallows the blind application of KCL as mentioned earlier. We can generalize our passive network algorithm to include all linear active circuits by extending our set of modeling primitives. An independent voltage source has a fixed voltage across its terminals and an arbitrary current passing through it. The current is arbitrary in the sense that it can take any value, the exact value being determined by the circuit configuration. Similarly, an independent current source supplies a fixed current but has an arbitrary voltage across its terminals. We now introduce an artificial circuit element, called a norator, that supplies an arbitrary current and has an arbitrary voltage across its two terminals. Reasoning along the same lines, we hypothesize the existence of a circuit element that combines the terminal behavior of a short circuit and an open circuit in an analogous manner to the norator. This element is called a nullator, and it is characterized by zero voltage across its terminals and zero current flowing through it. It must be understood that nullators and norators are artifacts. They are not physically realizable circuit elements. They are included

in our set of modelling primitives since their unique terminal characteristics, in conjunction with that of passive elements and independent current sources, allows us to model the behavior of any linear active circuit element. For example, an independent voltage source can be modelled by an independent current source of the same magnitude in parallel with a one ohm resistor and a nullator norator pair. The voltage across the resistor must remain fixed. This implies that the current flowing through the resistor must be solely derived from the current source. We isolate the resistor from the rest of the circuit by connecting a nullator between the positive terminal and the rest of the circuit. The nullator forces the voltage on the external circuit node to equal that across the resistor while preventing any external circuit current from entering the resistor. The voltage source must also allow arbitrary current to flow through itself. This behavior is modelled by connecting a norator between the external node and the positive terminal of the current source. Likewise, we can model the behavior of an ideal operational amplifier by a nullator across its inputs and a grounded norator connected to its output. The ease with which we can model the behavior of an operational amplifier - and therefore the analysis of active filter circuits - is the major reason for choosing the nullator norator theory over others, for building nodal equations.

## 2.4 ACTIVE NETWORK ANALYSIS

The presence of nullators and norators in a circuit effects the matrix formulation of its nodal equations. The previous algorithm can be used to construct the nodal equations for a circuit without any nullators or norators. Addition of nullators and norators forces more constraints on the currents and voltages in the circuit. A nullator does not contribute any current into either one of its nodes and thus, the corresponding entries in the current vector  $I$ , are unaffected. However, it forces the voltage across its terminals to zero. The corresponding entries in the unknown voltages vector  $V$  have to be the same. The corresponding columns in the admittance matrix  $Y$  now multiply the same entry in  $V$ . We can preserve the semantics of the matrices by deleting one of the two identical entries in  $V$  and adding its counterpart row in  $Y$  to the surviving column. Adding a norator effects the current vector since it allows an arbitrary amount of current through itself. Thus, if there is a nullator between nodes  $i$  and  $j$  then an arbitrary amount of current will be added to the  $i$ th entry and the same amount will be subtracted from the  $j$ th entry in  $I$ . Adding the

two rows together in  $I$  and  $Y$  matrices restores one of the entries in the current vector. The other entry has one more unknown variable in it and thus, the associated nodal equation can be neglected. Hence, adding a nullator between nodes  $i$  and  $j$  results in adding columns  $i$  and  $j$  in the passive circuit  $Y$  matrix while adding a norator results in addition of rows  $i$  and  $j$  of the  $I$  and  $Y$  matrices.

The special case of grounded nullators and norators is considered next. A grounded nullator forces the node voltage on its other terminal to zero. The corresponding column in the  $Y$  matrix now multiplies a zero entry. This is the same as deleting that column. A grounded norator draws arbitrary current from its ungrounded terminal. Thus, the associated entry in  $I$  is diminished by an arbitrary amount. In other words, the nodal equation at the ungrounded node has an extra unknown and can therefore be discarded from the analysis. Hence, a grounded nullator (norator) results in deleting a column (row) in  $Y$  and an entry from the  $V$  ( $I$ ) matrix.

A generalized algorithm for forming nodal equations of linear active circuits by inspection is outlined below.

*Assumption:* The  $(n+1)$  node circuit is composed entirely of passive elements, independent current sources, and  $M$  nullator norator pairs.

Step I. Form the nodal equations using the previous algorithm, ignoring any nullators or norators in the circuit. We obtain

$$I = Y_n \times_n V$$

Step II. For a nullator between nodes  $i$  and  $j$ , add column  $j$  to  $i$  of the admittance matrix  $Y$ . Delete column  $j$  from  $Y$  and delete the  $j$ th entry in  $V$ . Re-label  $V_i$  as  $V_{i,j}$  and re-label column  $i$  as  $i,j$ .

Step III. For a nullator between nodes  $i$  and ground (0), delete column  $i$  from  $Y$  and delete  $V_i$ .

Step IV. For a norator between nodes  $i$  and  $j$ , add row  $j$  to  $i$  of the admittance matrix  $Y$ . Delete row  $j$  from  $Y$  and delete the  $j$ th entry in  $I$ . Re-label row  $i$  as  $i,j$ .

Step V. For a norator between nodes  $i$  and ground (0), delete row  $i$  from  $Y$  and  $I_i$ .

Step VI. Repeat steps II-V  $M$  times. The size of the  $Y$  matrix will be reduced by  $M$  rows and  $M$  columns (i.e.  $Y_{(n-M) \times (n-M)}$ ).

We now have

$$I = Y_{(n-M) \times (n-M)} V$$

$$\text{where } V = [V_1 \ V_2 \ V_3 \dots V_{n-M}]^T$$

is the unknown column vector of node voltages.

The preceding algorithm is valid for both symbolic and numerical circuit analysis. We need to determine the all unknown voltages in the circuit in order to get a description of its response to external stimuli. The current through each element is then computed through a straightforward application of Ohm's Law. The matrix formulation of nodal equations allows us to use matrix theory to solve for unknown quantities. The characteristic equation of the circuit is found by computing the determinant of the admittance matrix,  $Y$ . Knowledge of the characteristic equation is critical in determining the stability and natural response of the circuit. The transfer function between a node voltage  $V_i$  and the input voltage is found by substituting column  $i$  in the admittance matrix with the current vector, computing the determinant of the updated  $Y$  matrix, and dividing by the characteristic equation. This process is the Achilles heel of symbolic circuit analysis both in terms of time complexity ( $O(n^3)$ ) and space complexity ( $O(n^3)$ ). An efficient determinant algorithm coupled with the sparse nature of  $Y$  somewhat mitigate the "explosion" of symbols but symbolic analysis is still restricted to relatively modest problems. A discussion of some algorithms and associated data structures, for computing a symbolic determinant is presented in chapter IV.

## 2.5 ACTIVE FILTER DESIGN USING SYMBOLIC TRANSFER FUNCTIONS

Spice is oriented towards active filter design. It is desirable to automate the process of detecting filter functions in a circuit and solving for the relevant filter parameters in terms of circuit parameters. Furthermore, a symbolic estimate of errors due to non-ideal behavior of circuit parameters is crucial for a designer so that values of parameters may be chosen to minimize them. An overview of the theory behind second order filters is presented followed by a treatise on the effect of finite Gain-Bandwidth-Product on ideal filter parameters.

We can assume without loss of generality that the input is normalized to one. The voltage at any node is then just the transfer function between the node voltage and the input signal. The denominator of the transfer function is the characteristic equation of the circuit. A necessary condition for the existence of second order filter functions in the circuit is that the frequency domain characteristic equation be of the form

$$s^2 + (w_0/Q_0)s + w_0^2$$

where  $w_0$  is the frequency for which the filter is designed, and the magnitude of  $Q_0$  governs the amplification/attenuation at the frequency  $w_0$ . The form of the transfer functions's numerator classifies the filter function at the node as either low-pass, high-pass, band-pass, notch, or all-pass.

### 2.5.1 LOW PASS FILTER FUNCTION

A transfer function that attenuates frequencies beyond a certain cutoff frequency  $w_0$  but is transparent to frequencies below the cutoff frequency performs low-pass filtering. For instance, the voltage across the capacitor in a series RC circuit is attenuated at high frequencies but is unaffected at low frequencies. A low-pass filter function is said to exist at the node joining the resistor and the capacitor. A second order low-pass filter exists at a node if its transfer function is of the form

$$H(s) = (H_0 w_0^2) / (s^2 + (w_0/Q_0)s + w_0^2)$$

It is apparent that at high frequencies the transfer function behaves like  $1/s^2$  and at

low frequencies it is approximately independent of frequency (i.e  $H(s) \sim H_0$ ). Thus, the low frequencies are “passed” at the node while high frequencies are “blocked”.

### 2.5.2 HIGH PASS FILTER FUNCTION

High-pass filtering is the complement of low-pass filtering. Signals whose frequencies are above a critical frequency are passed unhampered while those whose frequencies are below the cutoff are attenuated. Drawing on the previous example, the voltage across the resistor in a series CR circuit is attenuated at low frequencies but approaches the value of the input signal at high frequencies. The  $1/s^2$  behavior of the low-pass filter at high frequencies can be counteracted if an  $s^2$  is present in the numerator. This leads to the form of a high-pass, second order filter function

$$H(s) = (H_0 s^2) / (s^2 + (w_0/Q_0)s + w_0^2)$$

The magnitude of the transfer function is approximated by  $H_0$  at high frequencies and approaches zero at low frequencies. A high-pass filter exists at a node if its transfer function has the preceding structure.

### 2.5.3 BAND PASS FILTER FUNCTION

Some applications require that only a band of frequencies be transmitted through a circuit. For example, a tuner has to recognize a very narrow band of frequencies in order to “tune” into a particular station. One way of accomplishing this function is by routing the incoming signal through a filter that allows only a band of frequencies to pass through, and discards the rest of the spectrum. The voltage across the resistor in a series RLC circuit exhibits band-pass behavior. At low frequencies most of the voltage is across the capacitor while at high frequencies the inductor behaves like an open circuit and thus has most of the voltage across itself. At intermediate frequencies the voltage across the resistor increase due to phasor cancellation between the capacitor and inductor voltages. A band-pass filter function therefore, exists across the resistor. The general form for a second order band-pass filter is given by



$$H(s) = (H_0 (w_0/Q_0)s) / (s^2 + (w_0/Q_0)s + w_0^2)$$

The transfer function is proportional to  $1/s$  at high frequencies and approaches zero at low frequencies. Thus, it behaves like a high-pass filter at low frequencies and as a low-pass filter at high frequencies. Signals whose frequencies fall around  $w_0$  are transmitted with a gain of approximately  $H_0$ .  $Q_0$  in this context can be interpreted as the selectivity factor (i.e the narrower the band of allowed frequencies, the higher the  $Q_0$ ).

#### 2.5.4 NOTCH FILTER FUNCTION

A filter that performs the complementary function of a band-pass filter is called a notch filter. The notch filter can be used to reject unwanted frequency components in a signal. Applications include removing tape hiss, low frequency rumble from a record player, et cetera. The second order filter function of a band-pass filter is

$$H(s) = H_0 (s^2 + w_z^2) / (s^2 + (w_0/Q_0)s + w_0^2)$$

where  $w_z$  is called the notch frequency. The notch filter function may be combined with high-pass and low-pass filter functions by adjusting the relative magnitudes of  $w_z$  and  $w_0$ . A true notch filter function results if  $w_z$  is equal to  $w_0$  (i.e. the gain at frequencies below and above  $w_0$  are transmitted without attenuation but a band of frequencies centered at  $w_0$  is "blocked"). Low-pass filtering can be combined with notch filtering if  $w_0$  is less than the notch frequency. The gain at frequencies below  $w_0$  is much higher than the high frequency gain in this scenario. Similarly, notch filtering can be supplemented by high-pass filtering if  $w_0$  is greater than  $w_z$ .

#### 2.5.5 ALL PASS FILTER FUNCTION

It is sometimes desirable to alter the phase of a signal without changing its magnitude. For example, phase shifts cannot be tolerated when transmitting data using Pulse Coded Modulation or a similar coding scheme. A filter that corrects for phase but leaves the magnitude of the incoming signal unchanged over all frequencies is

called an all-pass filter. The only way that the magnitude response can be the same is if the ratio of the numerator and the denominator of the transfer function is constant and frequency independent. However, such a transfer function does not alter the phase of the input signal. If we consider a second order filter, we note that the magnitude of the characteristic equation is unchanged if we flip the sign of the coefficient of  $s^1$ . The phase however changes by  $180^\circ$ . This observation leads to the general form of a second order all-pass filter

$$H(s) = H_0(s^2 - (w_0/Q_0)s + w_0^2) / (s^2 + (w_0/Q_0)s + w_0^2)$$

The phase shift is given by

$$\arctan(H(s)) = -2 \tan^{-1}\{(ww_0/Q_0) / (w_0^2 - w^2)\}$$

The filter functions discussed above are based on ideal behavior of circuit components. The ideal filter parameters  $w_0$  and  $Q_0$  represent the design poles of the circuit's transfer functions. The behavior of a filter is particularly sensitive to shifts in the value of the filter parameters. This problem becomes acute in the case of band-pass and notch filters since a small change in the value of the center frequency can cause the filter to miss the intended signal altogether. A shift in  $Q$  effects the selectivity of the filter, usually for the worse. These errors can be minimized if a closed form estimate in terms of circuit parameters can be found.

## 2.6 ESTIMATING DEVIANCE IN FILTER PARAMETERS

Sspice analyzes the effect of non-ideal operational amplifier behavior on the design poles of second order filters. The motivation for focusing on this particular brand of non-ideality is due to the widespread use of operational amplifiers in active filter design[2],[3],[6]. An operational amplifier should, theoretically, deliver infinite gain in a frequency independent manner. In practice, performance degradation starts as low as 22Hz. An empirical macromodel of an op amp can be obtained through its frequency response. More poles are introduced into the macromodel as we increase the frequency of excitation. Adding more poles to the macromodel increases its accuracy but if used in a circuit, causes the number of terms in the symbolic result to "explode". It is observed that the contribution of these poles becomes significant

only at very high frequencies. Thus, we can assume a one pole model of an op amp and successively add more poles into the model as we go higher in frequency.

The open loop gain times the frequency at which the first pole becomes significant is termed as the Gain Bandwidth Product (GBP) of the amplifier. Ideally, the GBP is infinite. The one pole model of the op amp yields a finite GBP. Introducing the non-ideal model of an op amp causes a shift in the ideal filter parameters  $\omega_0$  and  $Q_0$ . Each op amp increases the order of the characteristic equation by one. However, the extra poles introduced into the transfer function are far from the  $j\omega$ -axis and the behavior of the circuit is dominated by the design poles. The characteristic equation is therefore, still approximately second order. The effect of the new poles is to shift the design poles to new values. An estimate of the departure from ideal  $\omega_0$  and  $Q_0$  has been derived by Wilson, Bedri and Bowron[1]. The error estimate is in terms of circuit parameters. Thus, errors due to finite GBP can be mitigated by choosing appropriate values of circuit elements. However, it must be remembered that the estimate is valid only as long as the one pole model holds true and second order and higher error terms can be neglected. As we increase the operating frequency, additional modelling must be introduced and a new estimate generated to minimize the errors in filter parameter values.

## **CHAPTER III**

### **Numerical Evaluation and Symbolic Approximation**

#### **3.1 DRAWBACKS OF SYMBOLIC COMPUTATION**

Symbolic computation is often criticized for generating huge strings of symbols that are hard to analyze and whose physical meaning is obscure. This situation occurs because symbolic analysis is exact and the contribution of every parameter, however insignificant it might be, has to be embodied in the final answer. Moreover, an operation on symbolic operands usually results in an answer whose size is larger than either one of the operands. Contrastively, an operation on numeric operands results in an answer whose representational requirements are comparable to those of the operands. The numerical value of the result may be larger than either of its operands but there is no explosion of the representational size (not the value) of the final answer. Another galling aspect of symbolic computation is that the time taken to produce the result depends on the size of input operands. The speed of numeric computation, on the other hand, is largely independent of the value of input operands. Symbolic computation is, as a result, slow and prone to generating intractable results. A strategy is desired that combines the speed and low overhead of numeric computation with the semantic advantages offered by symbolic computation to generate a result that is at once concise and meaningful.

#### **3.2 MITIGATING PROBLEMS THROUGH SYMBOLIC APPROXIMATION**

Symbolic approximation based on numerical evaluation of circuit parameters is a methodology to generate symbolic results that are shorter than the exact result but not as accurate. Frequently, a circuit designer is working on a circuit whose behavior is not apparent through numerical simulation. Moreover, the circuit is such that its symbolic analysis yields a ponderous result. This situation frequently occurs in

macromodelling, where the designer is trying to generate a model that duplicates the behavior of the circuit. It should be noted that we start with a working circuit whose circuit parameters have already been defined. We are seeking an understanding of the factors determining circuit behavior. A simpler symbolic result can be generated if we can ascertain the terms that dominate the final answer. Symbolic terms whose contribution to the final result is negligible can be omitted. The symbolic result is not independent of circuit parameter values. That is, there is no guarantee that the same set of symbolic terms will be generated if we analyze the problem with a different set of parameter values. However, the insight provided to the designer about the circuit's behavior may make such a trade off worthwhile. Keeping these limitations in mind, a brief outline of a systematic procedure for approximating the exact symbolic result is presented.

Step I. Read the parameter values from the input file. Parameters that are found from the DC operating point are obtained from the user.

Step II. Carry out an exact symbolic analysis of the circuit.

Step III. Evaluate each term in the exact answer using the parameter values provided.

Step IV. The exact symbolic result is a polynomial in the Laplace symbol 's'. Each symbolic coefficient is approximated separately so that the original structure of the polynomial is preserved. A symbolic coefficient is approximated by comparing the magnitude of each term in the coefficient with the largest magnitude found in the group, and neglecting those whose value falls below a certain threshold of the maximum value.

Step V. Repeat step IV for each coefficient of  $s^i$  in the polynomial.

The definition of neglect has yet to be elucidated. As mentioned earlier, we neglect terms whose contribution to the final result is negligible. A term is deemed negligible if it falls below a certain percentage of the largest term in its group (Step IV). This threshold is controlled by the user. A low threshold will result in an approximated result that is closer to the exact answer than if a high threshold is chosen. The choice of the threshold is thus dependent on the amount of accuracy that

is desired. Presently, Sspice uses a step function to implement symbolic approximation. (i.e terms whose values are strictly less than the threshold are dropped). A more natural approach would be to use some kind of decay function, the decay constant being left to the user.

## CHAPTER IV

### Operational Specifics and Salient Features

#### 4.1 BEHAVIORAL DESCRIPTION OF SSPICE MODULES

Sspice is implemented as a set of four programs that can be run separately or under the control of a driver. Each program executes one phase of circuit analysis. Thus, the first module generates a matrix formulation of nodal equations from an input file. The second module takes this output and computes the characteristic equation and various node voltages. The second module, optionally, generates the input file for the third or fourth module depending on whether ideal filter analysis, or error estimation is desired by the user. The third module performs ideal filter analysis on node voltages that are supplied to it as input. The fourth module requires an input file containing the ideal, and first order error terms in the characteristic equation. It estimates the shifts in ideal filter parameters based on this data.

#### 4.2 OVERALL OPERATION

The command Sspice invokes a driver that sequentially invokes processes to execute the first and second modules, and (optionally) the third or fourth module. In this scenario, the user can only control the execution of Sspice but is prohibited from accessing any intermediate data generated by the modules. Intermediate data, such as the matrix formulation of nodal equations, is stored in temporary files between successive module invocations. The names of these files, plus some book-keeping information, is written to another file, called a messenger file. Information pertaining to the whereabouts of this messenger file is supplied when invoking the relevant module. Safe names must be generated for these temporary files so that existing files that are named the same, are not overwritten. This is accomplished through a set of system calls to the operating system. Both UNIX and DOS support

this facility. Furthermore, generating safe names for intermediate files allows multiple invocations of Sspice. Multiple invocations are only possible in a windowing environment due to the interactive character of Sspice. One cannot easily circumvent this feature by writing a batch file, since the number of interactive queries depends on the nature of the response to those queries. The driver checks the exit status returned by each module for satisfactory completion before continuing execution. The user is thus, buffered from run time errors arising from an erroneous circuit description or bugs in the program.

### 4.3 ERROR DETECTION AND RECOVERY

Each module has one entry and one exit point. The reason for termination of the module is returned in the exit status of the process executing it. A non-zero value signals an abnormal termination of the module. There are two main sources that cause a module to exit abnormally. Erroneous input data, whether supplied by the user or by the preceding module, is the foremost reason causing the program to crash. A less likely, but equally debilitating circumstance arises when some system imposed limit is exceeded. Other reasons behind abnormal termination include, user generated software interrupts and logical errors in the code. The resulting signal generated in each case, is trapped by an error handling routine that removes any temporary files or data generated by the erring module, and returns the exit status to the driver. A message explaining the reason for the abort is written to the terminal. Error messages caused by bugs in the program are further accompanied by the line number and function name in which the error occurred. The omission of floating point exceptions from the discussion might seem as an oversight given the numerical evaluation feature of Sspice. Special provisions - software checks for divide-by-zero, argument to square root function, e.t.c - are built into the code so that such errors never occur. The rationale is that a floating point exception during numerical analysis should not impede an exact symbolic analysis since Sspice is predominantly a symbolic analysis package. No such error recovery is possible when an erroneous description of the circuit topology is detected. For instance, if one of the resistor terminals has not been specified in the input file then the circuit is not completely defined and therefore, there is not much point in continuing execution. By the same token, execution is aborted if a system call returns an error status. This could occur, most likely, during a dynamic memory allocation call, and



perhaps less likely, if an error in file creation is caused due to some pre imposed limit on the number of simultaneously open files. Thus, considerable effort is directed towards enforcing a graceful degradation in performance on encountering pathological situations.

#### 4.4 USER FRIENDLY FEATURES

The user can gain finer control over the execution of Sspice if each module is executed separately. This feature allows the user to access and modify temporary data generated by the modules. Resistors can be removed from the circuit, for example, by deleting the respective matrix entries generated by the first module, before running the second module. The user can also shrink the size of the matrices by using some heuristics from matrix theory in order to speed up determinant computation. Determinants containing symbolic entries can be computed independently of the first module, a feature that is particularly attractive to control theory and mathematics applications. As a more sophisticated example, consider approximating the exact symbolic analysis of a circuit topology containing a JFET, resistors and capacitors. The values of the transconductance, and the drain to source resistance and capacitance are obtained by calculating the DC operating point at the beginning. However, these values have to be prompted for during each invocation of Sspice since they are not specified in the original SPICE input file. Sspice creates a data file containing the names of parameters and their corresponding numerical values during the first analysis of the circuit so that successive invocations can read from it without prompting the user. A different symbolic answer is generated by simply changing the parameter values in this file before the next run. It should be realized that the flexibility obtained by this feature is prone to malicious use and is thus, inappropriate for the naive user.

#### 4.5 OPERATIONAL DESCRIPTION OF SSPICE MODULES

##### 4.5.1 MODULE I: FORMING NODAL EQUATIONS

The first module generates a matrix formulation of a circuit's nodal equations given its topological description. Sspice accepts input in a format popularized by SPICE

so that symbolic analysis and numerical simulation may alternate without intervening editing of the input file. Format compatibility is assured from SPICE to Sspice but compatibility does not always hold in the reverse direction. The Sspice input file format allows much more latitude in element definitions than does SPICE. Sspice elements of the same type may have identical names, and need not be followed by their numerical value. In fact, two symbols are considered equal in a symbolic sense if their names are the same. A rich element library is supported although the majority of the models can be interpreted as predefined macros that consist of just resistors, capacitors, nullators, norators, and independent current sources. Multiple models for the same circuit element - distinguished by their degree of complexity - are available so that the desired level of modelling may be incorporated in the result. Appendix A lists the complete Sspice element library together with the format in which each element should appear in the input file. The first module makes two passes over the input file. The first pass checks to see if the circuit is fully specified and also establishes an upper bound to the size of the matrices. The space for the I, Y, and V matrices is then dynamically allocated. The second pass over the input file recognizes, and inserts the appropriate model for, circuit elements into the matrices. The user is asked to decide between the ideal and the one pole model if any op amps are encountered in the circuit description. A flow chart depicting the interactive structure of the first module is given in Appendix D. The modelling phase is followed by a nullator norator reduction of the matrices as explained in the first chapter. The resulting matrix formulation is written to a temporary file and the exit status returned to the driver.

#### 4.5.2 MODULE II: SOLVING NODAL EQUATIONS

The second module computes the determinant of the matrices specified in the input file and performs a user controlled level of analysis on the result. The determinant of the admittance matrix, Y, supplies the characteristic equation. Cramer's rule can be used to evaluate other node voltages in the circuit. Each node voltage requires computing a determinant, and rather than computing all the node voltages, the user is asked to select those of interest to him/her. This feature avoids time consuming, and usually unnecessary computation of determinants. Sorting the determinants on basis of a user supplied key is also supported. The default sorting key is the Laplace symbol 's'. The user is allowed to selectively enable this option for each

determinant, since sorting is a time consuming activity. Numerical evaluation of the symbolic result is another option available to the user. The user is prompted for parameter values only if the parameter is part of the answer. These values along with the corresponding parameter names are saved in an external file at the request of the user. Subsequent invocations of Sspice that analyze the same circuit can obtain the values from this file instead of prompting the user. The user can, of course, direct the program to ignore the data file when searching for parameter values. Appendix D charts the sequence of queries output by the program.

A simpler symbolic result can be obtained through the symbolic approximation feature. Its implementation is relatively straightforward. The exact symbolic answer is broadly divided into ideal and non ideal terms. Each class of terms is further grouped by orders of the Laplace symbol 's' and each such subgroup is simplified. Each subgroup consists of a number of terms in sign magnitude form. Associated with each term is a numeric field that contains the numerical evaluation of the term. The terms are sorted in decreasing order based on their numerical magnitude and terms below a certain user controlled threshold are dropped from the answer. This procedure is repeated for each subgroup and the results are merged together to form a simplified answer. The CMOS opamp example in Appendix C spectacularly illustrates this powerful idea. Further analysis (ideal filter analysis or error estimation) of the simplified or exact result is controlled by the user in which case the second module writes the result to a temporary file. If no further analysis is desired the various node voltages and the characteristic equation are written to a file. The original I, Y, and V matrices are also printed in the output file in conventional matrix format. The matrix printing routine automatically adapts to accommodate different sized matrices. Implementing all these features efficiently made the design of the second module especially challenging. However, the time spent in computing determinants is the dominant performance statistic of the second module, and indeed, of Sspice and its implementation is discussed now.

A detailed description of the algorithm for determinant computation is relevant due to its decisive role in determining Sspice's performance. Gaussian elimination and similar numerical techniques are inappropriate for computing determinant of matrices with symbolic entries. These techniques require some measure of division and reduction of entries to zero. Reducing an entry to zero is an expensive operation symbolically for it entails sorting and searching of terms. Symbolic division is

limited to cancellation of common factors in the numerator and denominator and is therefore, ineffective compared to its numeric counterpart. Computing determinants based on the definition of the determinant is the key idea behind Sannuti and Puri's method [9], one that has been implemented in Sspice.

Broadly, determinant computation involves generating a set of permutations, multiplying out the corresponding entries in the matrix, and cancelling all terms that have the same magnitude but opposite sign. Unlike terms are cancelled by first sorting terms lexicographically irrespective of their sign, and then searching for a pair of terms that are lexicographically equal but differ in their first character (i.e. sign). Generating all the permutations of the matrix - called  $Y$  for the purposes of this discussion - is both inefficient and unnecessary. Only the non zero permutations are needed. This can be accomplished by creating another matrix of the same dimension that contains the locations of the zero entries in the original matrix. This matrix is termed the routing matrix,  $R$ , in [9]. Specifically, each column  $k$  in  $Y$  maps to column  $k$  in  $R$  in the following fashion: The row number of the first non zero entry in the  $Y$  column vector is the first entry in the  $R$  column vector. Scan the  $Y$  column vector, updating successive entries in the  $R$  column vector with the row numbers of non zero entries in  $Y$  until either a zero entry is encountered or the end of the column is reached. In either case, repeat the procedure for the  $k+1$  column. A sparse matrix would result in at least one row of zeros in  $R$ . Non zero permutations can be generated by scanning  $R$ . The basic idea is to fix a non zero entry in the first column in  $R$  and vary non zero entries in the other columns. This procedure is repeated for each non zero entry in the first column. The algorithm terminates when a zero is encountered. As each permutation is generated, a procedure is called to multiply out the corresponding entries in  $Y$  and append them to a string representing the determinant. The algorithm in this form fails for a matrix that has at least one column consisting entirely of non zero entries. The corresponding routing matrix has no row of zeros and therefore, the algorithm never terminates. This situation can be remedied by appending a row of zeros after the last row in  $R$ . This simple provision forces the algorithm to work for all matrices. It should be noted that the routing matrix is now of size  $(n+1)$  by  $(n)$  assuming that  $Y$  is a square matrix of dimension  $(n)$ .

## 4.5.2.1 EXAMPLE

The determinant finding algorithm[9] can best be illustrated by an example. Consider the following upper triangular matrix, whose determinant is, from matrix theory, the product of its diagonal elements. The corresponding routing matrix, R, is also shown.

$$\begin{matrix}
 \begin{bmatrix} -G & G-1 & -1 & 1 \\ 0 & -sL & 1+sC & sL-1 \\ 0 & 0 & sC & G \\ 0 & 0 & 0 & -sL \end{bmatrix} & \longrightarrow & \begin{bmatrix} 1 & 1 & 1 & 1 \\ 0 & 2 & 2 & 2 \\ 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 4 \end{bmatrix} \\
 Y_{4 \times 4} & & R_{4 \times 4}
 \end{matrix}$$

Vectors F, IS, and P, each of dimension 4, together with an integer variable, k, are also required by the algorithm. The following table represents an execution trace for generating the first permutation of the matrix R. The vector, IS, stores the permutation generated.

Table 4.1 Execution trace of determinant algorithm

Step executed	K	F <sub>1 x 4</sub>	P <sub>1 x 4</sub>	IS <sub>1 x 4</sub>
Initialization	1	[1, 0, 0, 0]	[1, 1, 1, 1]	[1, *, *, *]
Forward	2	[1, 0, 0, 0]	[1, 1, 1, 1]	[1, *, *, *]
Search (1), (2), (4)	2	[1, 0, 0, 0]	[1, 2, 1, 1]	[1, *, *, *]
Search (1), (2), (3)	2	[1, 1, 0, 0]	[1, 2, 1, 1]	[1, 2, *, *]
Forward	3	[1, 1, 0, 0]	[1, 2, 1, 1]	[1, 2, *, *]
Search (1), (2), (4)	3	[1, 1, 0, 0]	[1, 2, 2, 1]	[1, 2, *, *]
Search (1), (2), (4)	3	[1, 1, 0, 0]	[1, 2, 3, 1]	[1, 2, *, *]
Search (1), (2), (3)	3	[1, 1, 1, 0]	[1, 2, 3, 1]	[1, 2, 3, *]
Forward	4	[1, 1, 1, 0]	[1, 2, 3, 1]	[1, 2, 3, *]
Search (1), (2), (4)	4	[1, 1, 1, 0]	[1, 2, 3, 2]	[1, 2, 3, *]
Search (1), (2), (4)	4	[1, 1, 1, 0]	[1, 2, 3, 3]	[1, 2, 3, *]
Search (1), (2), (4)	4	[1, 1, 1, 0]	[1, 2, 3, 4]	[1, 2, 3, *]
Search (1), (2), (3)	4	[1, 1, 1, 1]	[1, 2, 3, 4]	[1, 2, 3, 4]
Flower (Decode permutation)	3	[1, 1, 0, 0]	[1, 2, 4, 1]	[1, 2, 3, 4]
Search (1), (5)	3	[1, 1, 0, 0]	[1, 2, 4, 1]	[1, 2, 3, 4]
Backward	2	[1, 0, 0, 0]	[1, 3, 1, 1]	[1, 2, 3, 4]
Search (1), (5)	2	[1, 0, 0, 0]	[1, 3, 1, 1]	[1, 2, 3, 4]
Backward	1	[0, 0, 0, 0]	[2, 1, 1, 1]	[1, 2, 3, 4]
Search (1), (5), TERMINATE	1	[0, 0, 0, 0]	[2, 1, 1, 1]	[1, 2, 3, 4]

### 4.5.3 MODULE III: FILTER FUNCTION IDENTIFICATION

The third module identifies ideal filter functions and solves for filter parameters given an input file containing the characteristic equation and node voltages. Filter function identification is implemented by associating the general form of second order filter functions with the transfer function at each node and extracting the relevant coefficients. Appendix B describes the design of a state variable active filter that aptly illustrates these principles. Solving for filter parameters usually involves the limited form of division described previously. Computing the ideal  $Q_0$  requires taking a square root of a symbolic operand. This is achieved by extracting simple factors from the operand and searching it for two occurrences of the same simple factor. Numerical evaluation of filter parameters is also supported. Circuit parameter values are obtained from the messenger file. It is possible that a divide-by-zero exception may occur while calculating the numerical value of filter parameters. It is desirable, in this situation, to abort numerical evaluation but to continue with the exact symbolic analysis. Therefore, a variety of software checks are implemented that detect and prevent such an exception from occurring.

### 4.5.4 MODULE IV: ESTIMATING DEVIANCE DUE TO NON IDEAL EFFECTS

The fourth module estimates the first order errors in ideal filter parameters due to finite gain bandwidth product of op amps. The error estimate is in terms of circuit parameters and is based on the Wilson Bedri Bowron approximation [1]. The symbolic value for ideal  $Q_0$  and  $F_0$  are also computed as a reference. The design of a state variable active filter in Appendix B analyzes non ideal effects on filter parameters using these ideas. Numerical evaluation of the errors and the ideal filter parameters is implemented in accordance with previous modules. Matching the GBPs of different op amps results in a simpler formula for the error estimate. The user is provided with an option to equate, symbolically, the various GBPs in the answer. The fourth module is almost entirely built on the code from the first three modules. It is expected that future code development will follow the same trend.

## 4.6 CONCLUSION

Considerable effort has been directed towards making Sspice reliable and easy to maintain. The code, though extensive, is simple and highly portable. There are no inherent limits to the size of internal data structures due to the nature of symbolic computation. The size of the problems it can analyze is limited only by the computational resources available to it. Its implementation in 'C' results in high performance coupled with small native code. The microcomputer version occupies less than 150K and analyzes the example in Appendix C in approximately fifteen seconds. An aggressive advertizing campaign targeted at both academe and industry may expose a wider audience to a systematic approach to analog circuit design.

## CHAPTER V

### Future Directions

A symbolic stability analysis of the circuit from its characteristic equation would be a useful addition to Sspice. An algorithm for implementing this feature is described in [11]. The approach is eminently feasible since it requires computing a number of determinants of sparse matrices, a feature that is already implemented in Sspice. The entries in the matrices are the symbolic coefficients of the characteristic equation that is computed in the second module. A stability analysis would result in another set of constraints on circuit parameter values and would offer valuable insight into circuit behavior.

Symbolic sensitivity analysis is yet another feature that can be implemented on a short term basis. Sensitivity analysis is based on an association of terms from the characteristic equation according to a set of well defined rules. A formula in terms of circuit parameters for estimating sensitivity to variations in a certain circuit parameter yields a set of constraints a circuit designer should be aware of.

The symbolic approximation feature of Sspice is still in a state of infancy. The current implementation discards symbolic terms from the exact result based on comparison of relative magnitudes, ignoring the sign of each term, altogether. This approach might fail for a combination of parameter values that should result in a cancellation of terms that have large, equal magnitudes but opposite signs. Terms with smaller magnitude should then, become significant. However, this intelligence is not incorporated in the current version with the result that an erroneous result might be generated. Such situations were not encountered during the extensive testing of the software. However, a further refinement is necessary if only for the sake of completeness.

ZNAP is a software tool that automatically performs Op Amp Relocation on a seed



circuit and retains only stable configurations of relocated circuits [12]. A major step towards computer aided active filter design would be to integrate ZNAP with Sspice in a master slave relationship. It is envisioned that ZNAP would invoke multiple instances of Sspice so that each stable configuration may be symbolically analyzed in parallel.

Major portions of Sspice are amenable to parallel computation. For instance, all the permutations of the admittance matrix can be generated in parallel. Similarly, numerical evaluation of terms during symbolic approximation can proceed independently of each other. A parallelizing compiler or a rewrite in a parallel language such as Occam would allow Sspice to analyze a new, larger class of problems. Symbolic analysis combined with symbolic approximation seems to be the correct approach, in principle, to analog circuit design. In practice, it requires extensive computer resources to produce a tractable result.

## **APPENDICES**

## APPENDIX A

### Input format and modelling of Sspice elements

#### PRIMITIVE ELEMENTS

R[name] <node> <node> [value of R] [optional text, ignored by Sspice]



Figure 1. Resistor model and input specification

C[name] < node > < node > [value of C] [optional text, ignored by Sspice]

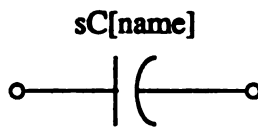


Figure 2. Capacitor model and input specification

I[name] < (+) node > < (-) node > AC [optional text, ignored by Sspice]

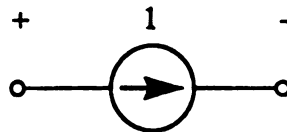


Figure 3. Independent current source model  
(normalized to one)

**XNN** < node (nullator) > < node (nullator) > < node (norator) > < node (norator) >  
 [optional text, ignored by Sspice]

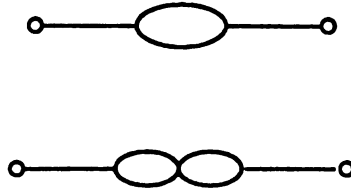


Figure 4. Nullator Norator pair

## MACROS

**I[name]** < (+) node > < (-) node > [optional text, ignored by Sspice]

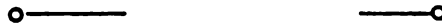


Figure 5. DC current source model (open circuit)

**V[name]** < (+) node > < (-) node > AC [optional text, ignored by Sspice]

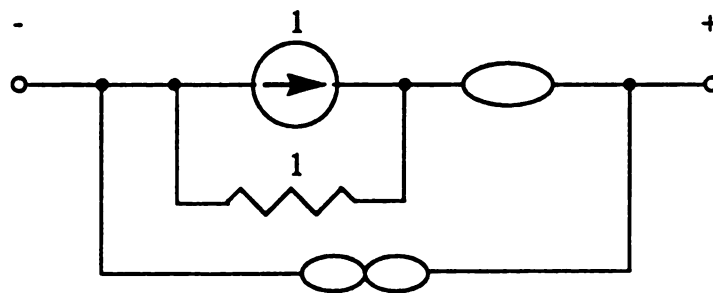
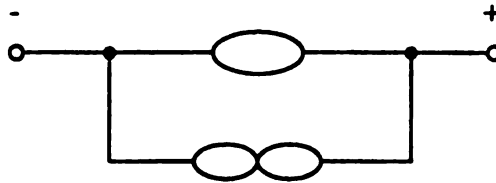


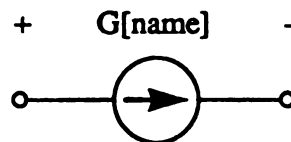
Figure 6. AC voltage source model (normalized to one)

**V[name] < (+) node > < (-) node > [optional text, ignored by Sspice]**



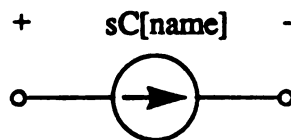
**Figure 7. DC Voltage source model (short circuit)**

**I {ignore} / R[name] < (+) node > < (-) node > [optional text, ignored by Sspice]**



**Figure 8. Source transformed AC current source model**

**I {ignore} / \*sC[name] < (+) node > < (-) node > [optional text, ignored by Sspice]**



**Figure 9. Source transformed AC current source model**

L[name] <node> <node> [value of L] [optional text, ignored by Sspice]

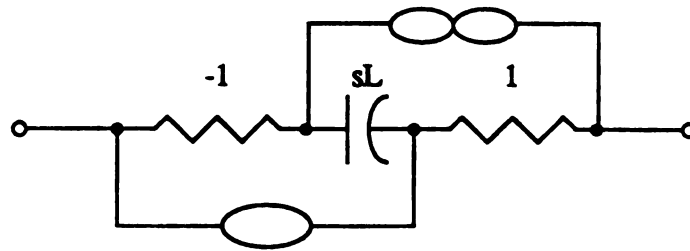


Figure 10. Inductor model

G[name < (+) node > < (-) node > < (+ controlling) node > < (- controlling) node >  
[value of GM] [optional text, ignored by Sspice]

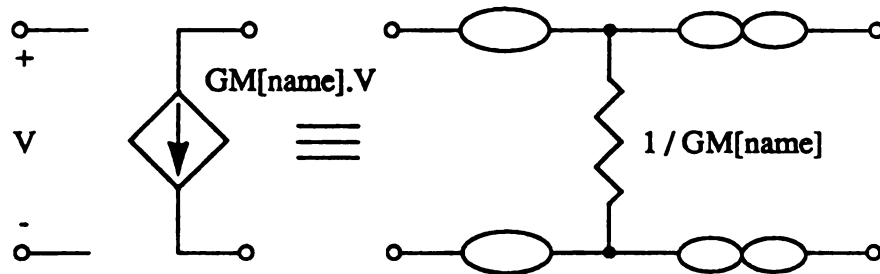


Figure 11. Voltage Controlled Current Source model (Gain = GM)

CC[name] < node > < node > [optional text, ignored by Sspice]

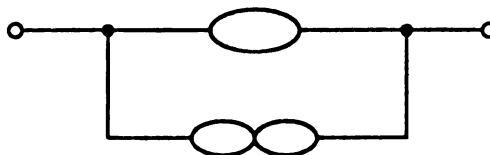
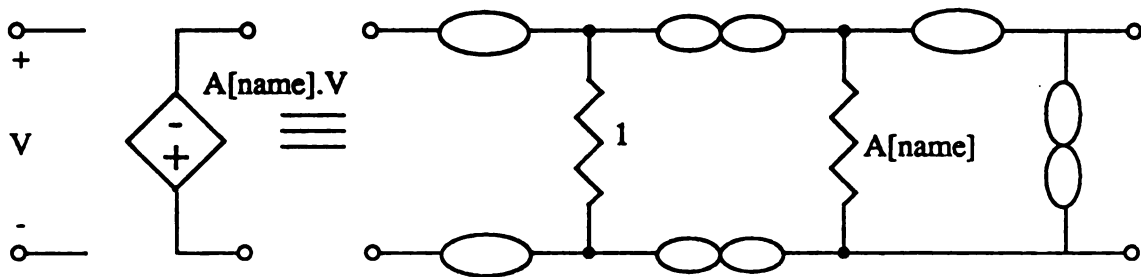


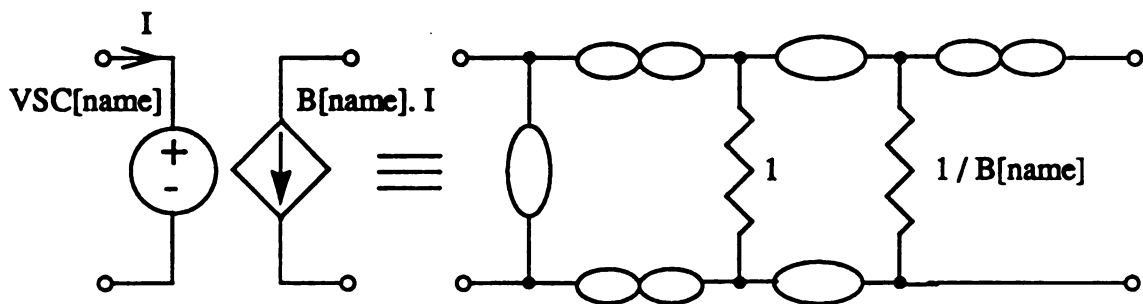
Figure 12. Coupling Capacitor Model (short circuit)

**E[name < (+) node > < (-) node > < (+ controlling) node > < (- controlling) node >  
[value of A] [optional text, ignored by Sspice]**



**Figure 13. Voltage controlled voltage source model (Gain = A)**

**F[name < (+) node > < (-) node > < Unique Controlling V device VSC[name] >  
[value of B] [optional text, ignored by Sspice]**



**Figure 14. Current controlled current source (Gain = B)**

H[name < (+) node > < (-) node > < Unique Controlling V device VSC[name] >  
[value of RM] [optional text, ignored by Sspice]

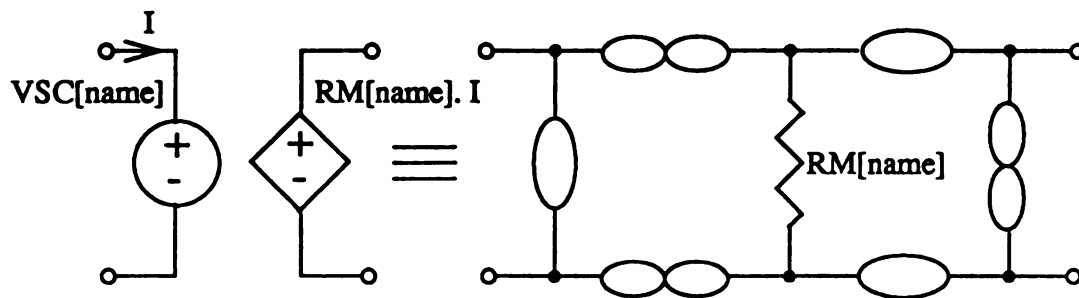


Figure 15. Current controlled voltage source model (Gain = RM)

EK[name < (+) node > < (-) node > < (+ controlling) node > < (- controlling) node >  
[value of K] [optional text, ignored by Sspice]

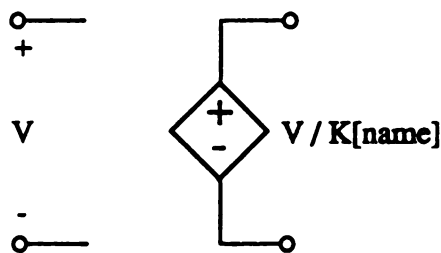


Figure 16 Voltage controlled voltage source (Gain = 1/K)



## HIGHER LEVEL MACROS

D[name] < node > < node > [optional text, ignored by Sspice]

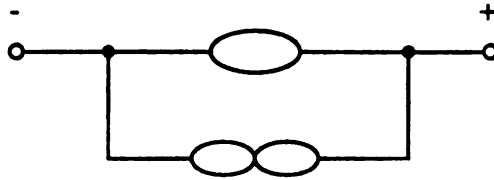


Figure 17. Lowest level diode model (short circuit)

DL[name] < node > < node > [optional text, ignored by Sspice]



Figure 18. Low frequency diode model

DH[name] < node > < node > [optional text, ignored by Sspice]

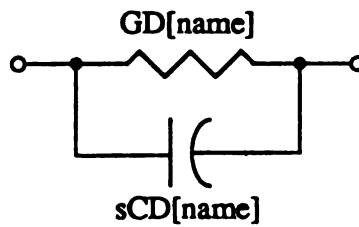


Figure 19. High frequency diode model.

Q[name] < (collector) node > < (base) node > < (emitter) node >  
 [optional text, ignored by Sspice]

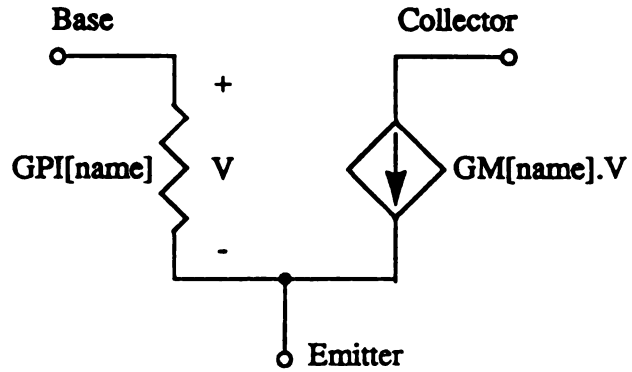


Figure 20. Simplest hybrid pi model of a BJT (Gain = GM)

QB[name] < (collector) node > < (base) node > < (emitter) node >  
 [optional text, ignored by Sspice]

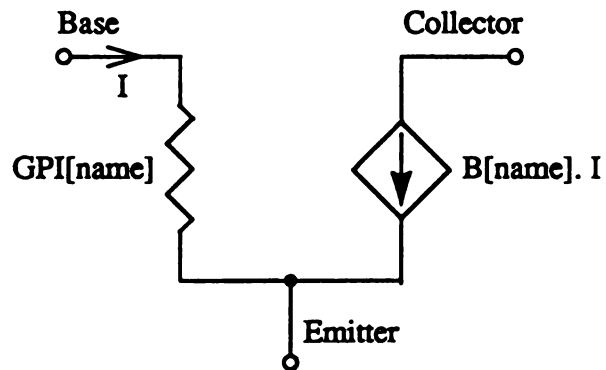


Figure 21. Simplest hybrid pi model of a BJT (Gain = B)

QL[name] < (collector) node > < (base) node > < (emitter) node >  
 [optional text, ignored by Sspice]

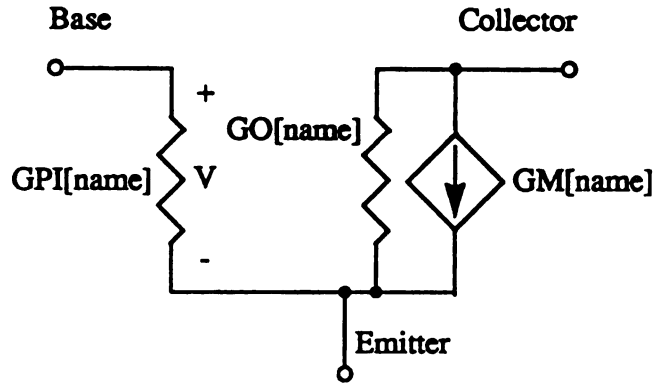


Figure 22. Low frequency hybrid pi model of a BJT

QH[name] < (collector) node > < (base) node > < (emitter) node >  
 [optional text, ignored by Sspice]

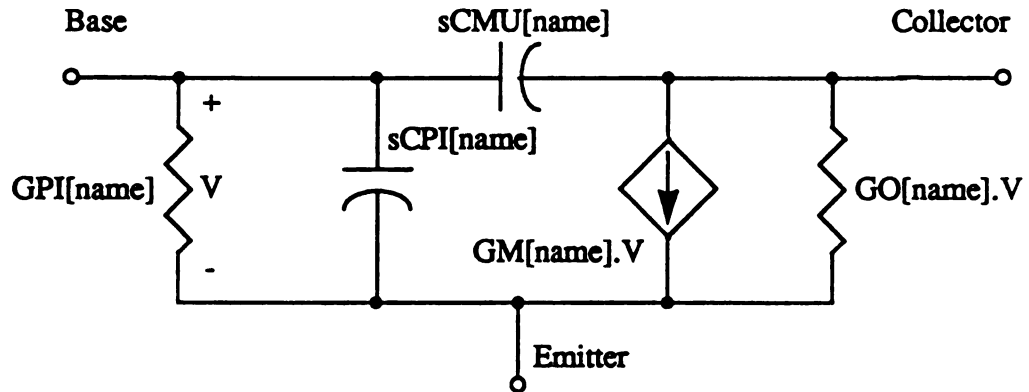


Figure 23. High frequency hybrid pi model of a BJT

J[name] < (drain) node > < (gate) node > < (source) node >  
 [optional text, ignored by Sspice]

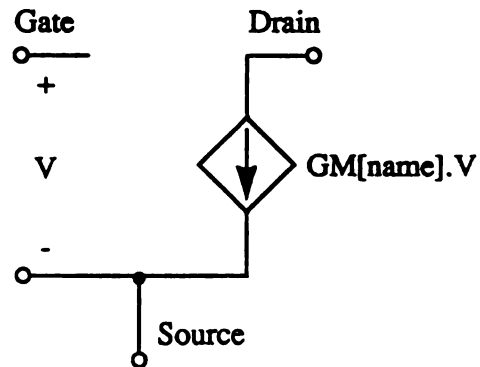


Figure 24. Lowest level JFET model

JL[name] < (drain) node > < (gate) node > < (source) node >  
 [optional text, ignored by Sspice]

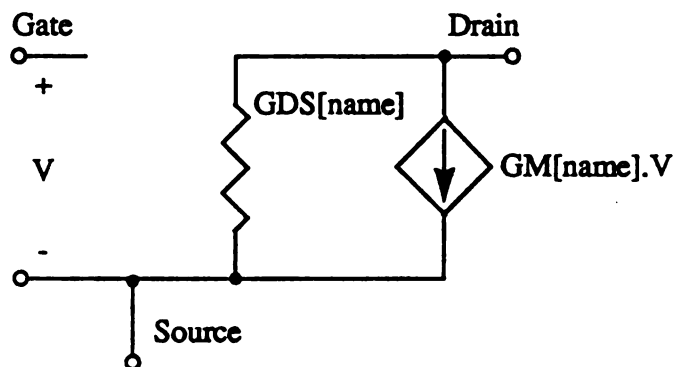


Figure 25. Low frequency JFET model

JH[name] < (drain) node > < (gate) node > < (source) node >  
 [optional text, ignored by Sspice]

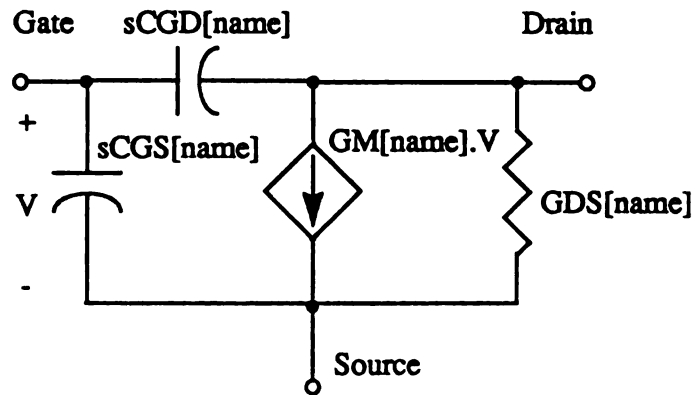


Figure 26. High frequency JFET model.

M[name] < (drain) node > < (gate) node > < (source) node > < (bulk) node >  
 [optional text, ignored by Sspice]

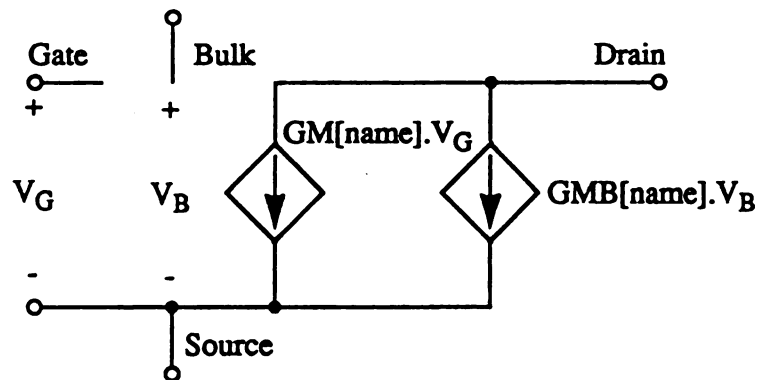


Figure 27. Lowest level MOSFET model.

ML[name] < (drain) node > < (gate) node > < (source) node > < (bulk) node >  
 [optional text, ignored by Sspice]

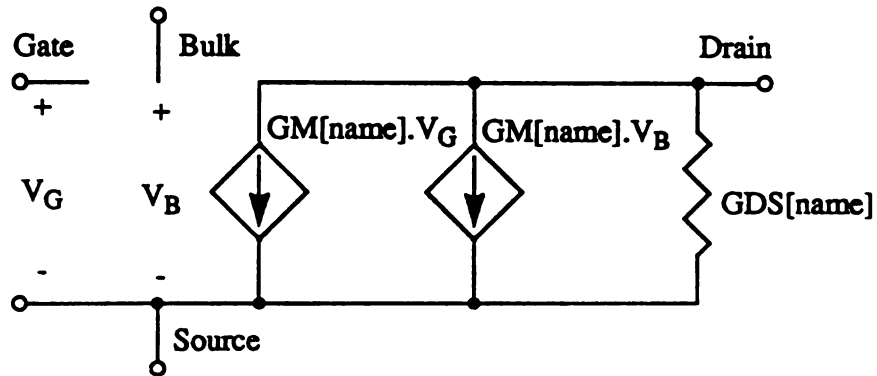


Figure 28. Low frequency MOSFET model

MH[name] < (drain) node > < (gate) node > < (source) node > < (bulk) node >  
 [optional text, ignored by Sspice]

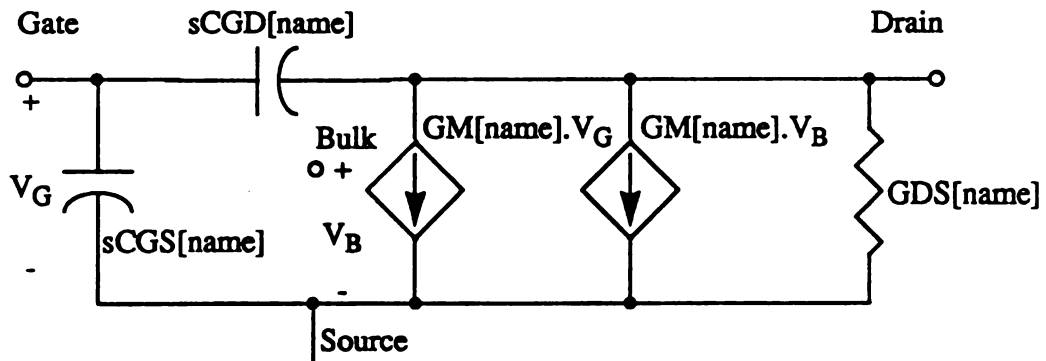


Figure 29. High frequency MOSFET model.

XOA[name] < (+ input) node > < (- input) node > < (output) node >  
 [optional text, ignored by Sspice]

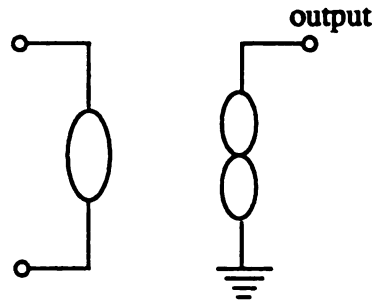


Figure 30. Ideal operational amplifier model

XOA[name] < (+ input) node > < (- input) node > < (output) node >  
 [optional text, ignored by Sspice]

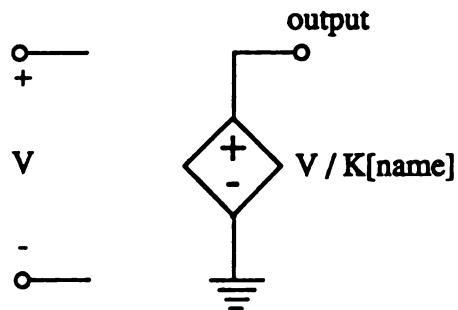


Figure 31. Non-ideal op amp model.

XOTA[name] < (+ input) node > < (- input) node > < (output) node >  
 [optional text, ignored by Sspice]

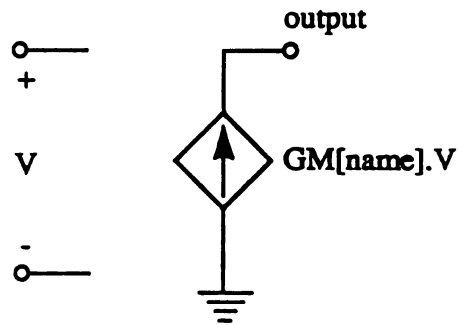


Figure 32. Operational transconductance amplifier model.

XFMR[name] <node > < node > <node > < node > [optional text ignored by Sspice]

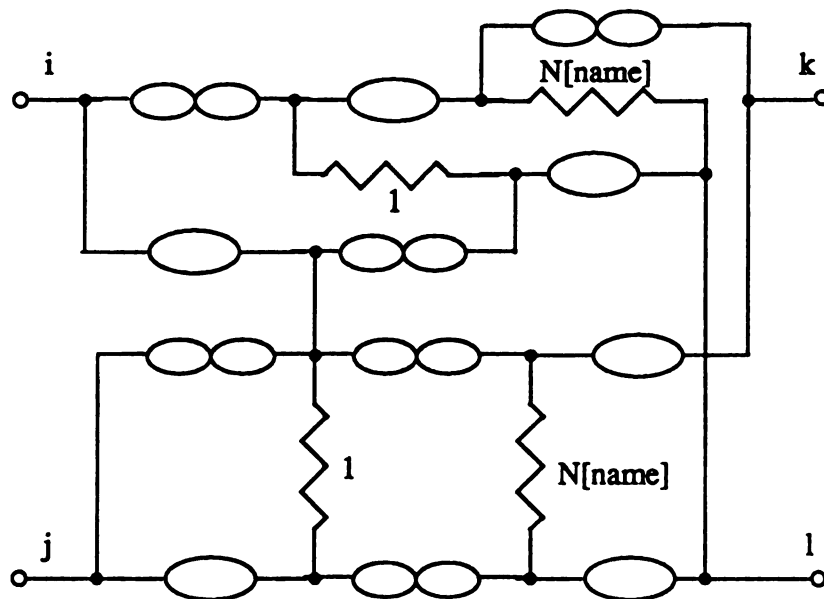


Figure 33. Transformer model with N:1 turns ratio.



## APPENDIX B

### Illustrative Example: State Variable Active Filter

Besides the basic analysis features of Sspice, there are other application specific features. At present these deal with the evaluation of active filters. The State Variable Active Filter is shown in Fig. B1. Because of the nature of symbolic analysis, Sspice needs some features not found in SPICE. One of these is the need to symbolically equate element values. In Fig. B1, several elements are equal. The Sspice input file is listed in Table B1 with  $R_5 = R_6 = R_7$ ,  $R_1 = R_2$ ,  $R_L = R_H$ , and  $C_1 = C_2$ .

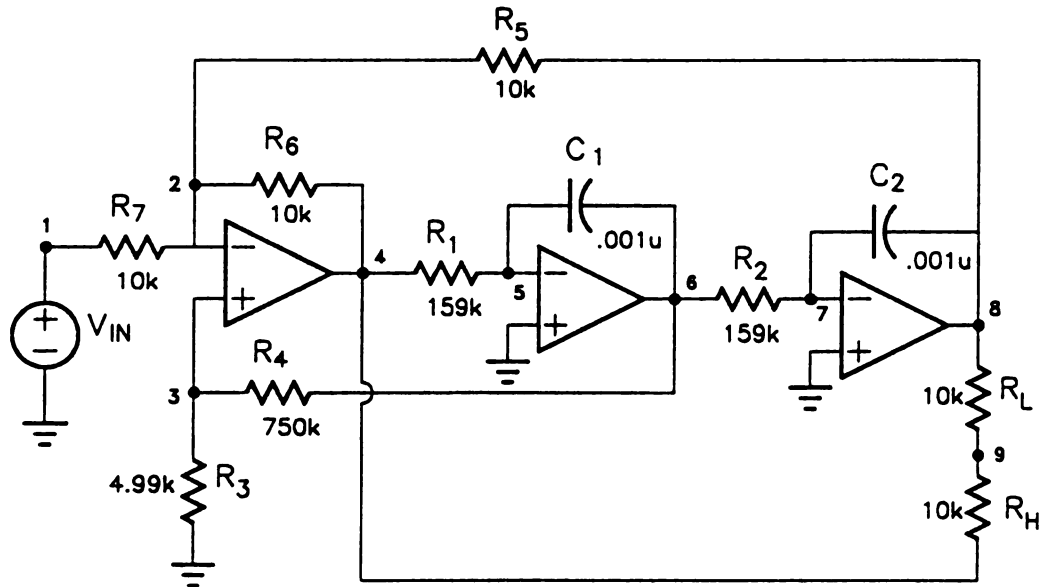


Figure B1. State variable active filter

Table B1. Sspice Input File SVAF.CIR

```

State Variable Active Filter
VIN 1 0 AC 1
R5 1 2 10K
R5 2 4 10K
R3 3 0 4.99K
R4 3 6 750K
R5 2 8 10K
XOA1 3 2 4
R1 4 5 159K
C1 5 6 0.001U
XOA2 0 5 6
R1 6 7 159K
C1 7 8 0.001U
XOA3 0 7 8
RL 8 9 10K
RL 9 4 10K
.END

```

In the first pass, we will treat the op-amp as ideal and find the filter functions. The following are the prompts seen by the user. The user responses are shown in bold and the output is listed in Table B2:

#### sspice

Sspice - Symbolic SPICE Circuit Analyzer

Version 1.0, 1 June 1990

(C) Copyright 1990 by Michigan State University

Unauthorized copying of this program is prohibited

INPUT FILE NAME [.cir] : **svaf**

Is XOA1 an ideal op-amp ? (y/n) : **y**

Is XOA2 an ideal op-amp ? (y/n) : **y**

Is XOA3 an ideal op-amp ? (y/n) : **y**

OUTPUT FILENAME [svaf.det] : **(hit return)**

Would you like the determinant string sorted according to orders of some variable? (y/n) : **n**

Would you like a numerical evaluation of the results? (y/n) : **y**

Do you want to save the parameter values for future invocations of the program? (y/n) : **y**

Would you like to discard some terms from the answer if their relative magnitude falls below a certain threshold? (y/n) : **n**

Would you like to see the numerator terms? (y/n) : **y**

Would you like to prepare a file to check and solve for any second order filter functions? (y/n) : **y**

FILTER FUNCTION FILE NAME [svaf.fun] : **(hit return)**

Do you want the numerator terms for V1? (y/n) : **n**

Do you want the numerator terms for V2? (y/n) : **n**

Do you want the numerator terms for V4? (y/n) : **y**

Do you want the numerator terms for V6? (y/n) : y  
 Do you want the numerator terms for V8? (y/n) : y  
 Do you want the numerator terms for V9? (y/n) : y

Table B2. Sspice Output File SVAFFUN

State Variable Active Filter

SECOND ORDER FILTER PARAMETERS:

Qo is:

$$\frac{\text{SQRT}( + 4*G4*G4 + 8*G4*G3 + 4*G3*G3 )}{( + 6*1)*( + G4)}$$

= 50.4337

Wo\*\*2 is:

$$\frac{( + G1*G1 )}{( + C1*C1)}$$

fo = 1000.97Hz

\*\*\*\*\*

There exists a HIGH PASS filter at : V4

HIGH PASS GAIN (Hhp) is:

$$\frac{( + 1 )}{- 1}$$

= -1

\*\*\*\*\*

There exists a BAND PASS filter at : V6

BAND PASS GAIN (Hbp) is:

$$\frac{( 1 + 1)*(G4)}{( + G4)*( + 6*1)}$$

= 50.4337

\*\*\*\*\*

There exists a LOW PASS filter at : V8

LOW PASS GAIN (Hlp) is:

$$\frac{( + 1 )}{- 1}$$

= -1

\*\*\*\*\*

Table B2. (cont'd.).

There exists a BAND STOP filter at : V9

BAND STOP GAIN (Hbs) is:

$$\frac{( + G4 + G3)}{- 2*G4 - 2*G3}$$

$$= -0.5$$

BAND STOP FREQUENCY (Wz\*\*2) is:

$$\frac{( + G1*G1)}{( + C1*C1)}$$

$$fz = 1000.97\text{Hz}$$

\*\*\*\*\*

Next we can find the approximate error in the design poles due to gain-bandwidth-product. Run the same input file but this time answer the prompt to indicate non-ideal op-amps. The following are the prompts seen by the user. The user responses are shown in bold and the output is listed in Table B3:

#### sspice

Sspice - Symbolic SPICE Circuit Analyzer  
Version 1.0, 1 June 1990  
(C) Copyright 1990 by Michigan State University  
Unauthorized copying of this program is prohibited

INPUT FILE NAME [.cir] : svaf

Is XOAl an ideal op-amp ? (y/n) : n

Is XOAl an ideal op-amp ? (y/n) : n

Is XOAl an ideal op-amp ? (y/n) : n

OUTPUT FILENAME [svaf.det] : (hit return)

Would you like the determinant string sorted according to orders of some variable? (y/n) : n

Would you like a numerical evaluation of the results? (y/n) : y

Do you want the parameter values to be read from the existing .DAT file? (y/n) : y

Would you like to discard some terms from the answer if their relative magnitude falls below a certain threshold? (y/n) : n

Would you like to prepare a file to solve for second order pole shifting due to gain-bandwidth-product (GBP in Hz)? (y/n) : y

ERROR ANALYSIS FILE NAME [svaf.err] : (hit return)

Do you want the GBP's equated in the ERROR ANALYSIS FILE? (y/n) : y

Would you like to see the numerator terms? (y/n) : n

What is the GBP (in Hz) of XOAl ? 1meg

Table B3. Sspice Output File SVAF.ERR

State Variable Active Filter

\*\*\*\*\*  
 \* Qo and fo \*  
 \*\*\*\*\*

Qo is:

$$\frac{\text{SQRT}( + 4*G4*G4 + 8*G4*G3 + 4*G3*G3 )}{( + 6*1 ) * ( + G4 )}$$

$$= 50.4337$$

(2\*PI\*Fo)\*\*2 is:

$$\frac{( + G1*G1 )}{( + C1*C1 )}$$

fo = 1000.97 Hz

\*\*\*\*\*  
 \* DQo/Qo = (D2-D0) foQo - Dfo/fo \*  
 \*\*\*\*\*

D2-D0:           where Ki = 1/GBP1

The numerator is:

+K TIMES

$$+ 8*G4*G4 + 28*G4*G3 + 20*G3*G3$$

The denominator is:

$$+ 4*G4*G4 + 8*G4*G3 + 4*G3*G3$$

\*\*\*\*\*  
 \* Dfo/fo = (D2-D1) fo/(2Qo) \*  
 \*\*\*\*\*

D2-D1 :           where Ki = 1/GBP1

The numerator is:

+K TIMES

$$+ 40*G4*G4 + 32*G4*G3 - 8*G3*G3$$

The denominator is:

$$( + G4 ) * ( + 12*G4 + 12*G3 )$$

\*\*\*\*\*  
 \* NUMERICAL EVALUATION \*  
 \*\*\*\*\*

Dfo/fo = -0.00096128

DQo/Qo = 0.252375

## APPENDIX C

### Illustrative Example: CMOS Op-Amp

This example illustrates a common problem in electronic circuit design that of the need to approximate. A CMOS op-amp is shown in Fig. C1. Sspice will be used to find the poles, zeros and low frequency gain of the open loop transfer function of the configuration shown in Fig. C2. Since there is a dominant pole capacitor and load capacitor in Figs. C1 and C2, a low frequency model for the MOSFETs will keep the order of the equations at two. The PSpice/Sspice input file is listed in Table C1.

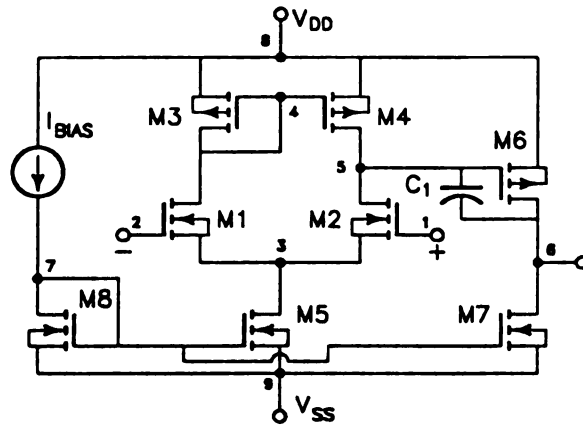


Figure C1. CMOS op-amp

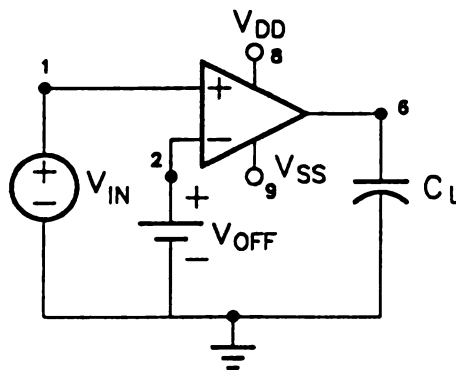


Figure C2. Open loop gain circuit

Table C1. PSpice/Sspice Input File

```

Allen and Holberg CMOS Op-Amp
ML1 4 2 3 3 NMOS1 W=43U L=10U AD=0.3N AS=0.3N PD=50U PS=50U
ML2 5 1 3 3 NMOS1 W=43U L=10U AD=0.3N AS=0.3N PD=50U PS=50U
ML3 4 4 8 8 PMOS1 W=10U L=10U AD=0.3N AS=0.3N PD=20U PS=20U
ML4 5 4 8 8 PMOS1 W=10U L=10U AD=0.3N AS=0.3N PD=20U PS=20U
ML5 3 7 9 9 NMOS1 W=38U L=10U AD=0.3N AS=0.3N PD=40U PS=40U
ML6 6 5 8 8 PMOS1 W=344U L=10U AD=1.3N AS=1.3N PD=350U PS=350U
ML7 6 7 9 9 NMOS1 W=652U L=10U AD=2.3N AS=2.3N PD=660U PS=660U
ML8 7 7 9 9 NMOS1 W=38U L=10U AD=0.3N AS=0.3N PD=40U PS=40U
C1 5 6 4.4P
IBIAS 8 7 8.8U
CL 6 0 20P
VDD 8 0 5
VSS 9 0 -5
VIN 1 0 AC 1
VOFF 2 0 0.1M
.MODEL NMOS1 NMOS (VTO=1 KP=17U GAMMA=1.3 LAMBDA=0.01 PHI=0.7
+ PB=0.8 MJ=0.5 MJSW=0.3 CGBO=200P CGSO=350P CGDO=350P CJ=300U
+ CJSW=500P LD=0.8U TOX=80N)
.MODEL PMOS1 PMOS (VTO=-1 KP=8U GAMMA=0.6 LAMBDA=0.02 PHI=0.6
+ PB=0.5 MJ=0.5 MJSW=0.25 CGBO=200P CGSO=350P CGDO=350P CJ=150U
+ CJSW=400P LD=0.8U TOX=80N)
.OP
.DC VIN -5M 5M 50U
.AC DEC 20 1 10MEG
.PROBE
.END

```

In order for Sspice to approximate the analysis of this circuit, the user must supply information about the biasing point. This is found by running PSpice and reading the output file. A partial output file is given in Table C2.

Table C2. PSpice Output File Bias Point Values

\*\*\*\* MOSFETS

NAME	ML1	ML2	ML3	ML4	ML5
MODEL	NMOS1	NMOS1	PMOS1	PMOS1	NMOS1
VBS	0.00E+00	0.00E+00	0.00E+00	0.00E+00	0.00E+00
VTH	1.00E+00	1.00E+00	-1.00E+00	-1.00E+00	1.00E+00
VDSAT	3.15E-01	3.15E-01	-9.53E-01	-9.53E-01	4.75E-01
GM	2.86E-05	2.86E-05	9.43E-06	9.43E-06	3.79E-05
GDS	4.31E-08	4.31E-08	8.66E-08	8.66E-08	8.67E-08
GMB	2.22E-05	2.22E-05	3.65E-06	3.65E-06	2.94E-05

NAME	ML6	ML7	ML8
MODEL	PMOS1	NMOS1	NMOS1
VBS	0.00E+00	0.00E+00	0.00E+00
VTH	-1.00E+00	1.00E+00	1.00E+00
VDSAT	-9.31E-01	4.75E-01	4.75E-01
GM	3.36E-04	6.58E-04	3.71E-05
GDS	2.84E-06	1.49E-06	8.67E-08
GMB	1.30E-04	5.11E-04	2.88E-05

The following are the prompts seen by the user. User responses are in bold:

### **sspice**

Sspice - Symbolic SPICE Circuit Analyzer  
Version 1.0, 1 June 1990  
(C) Copyright 1990 by Michigan State University  
Unauthorized copying of this program is prohibited

INPUT FILE NAME [.cir] : **cmos**

OUTPUT FILENAME [cmos.det] : **(hit return)**

Would you like the determinant string sorted according to orders of some variable? (y/n) : **n**

Would you like a numerical evaluation of the results? (y/n) : **y**

Do you want to save the parameter values for future invocations of the program? (y/n) : **y**

Would you like to discard some terms from the answer if their relative magnitude falls below a certain threshold? (y/n) : **n**

Would you like to see the numerator terms? (y/n) : **y**

Would you like to prepare a file to check and solve for any second order filter functions? (y/n) : **n**

What is the numerical value of GM8 ? **3.71e-5**

What is the numerical value of GM3 ? **9.43u**

What is the numerical value of GM2 ? **2.86e-5**

What is the numerical value of GM1 ? **2.86e-5**

What is the numerical value of RDS5 ? **11.53meg**

What is the numerical value of RDS2 ? **23.2meg**

What is the numerical value of RDS1 ? **23.2meg**

What is the numerical value of RDS3 ? **11.55meg**

What is the numerical value of RDS8 ? **11.53meg**

What is the numerical value of GM4 ? **9.43u**

What is the numerical value of RDS4 ? **11.55meg**

What is the numerical value of GM6 ? **3.36e-4**

What is the numerical value of RDS7 ? **671.1k**

What is the numerical value of RDS6 ? **352.1k**

Do you want the numerator terms for V1? (y/n) : **n**

Do you want the numerator terms for V3? (y/n) : **n**

Do you want the numerator terms for V4? (y/n) : **n**

Do you want the numerator terms for V5? (y/n) : **n**

Do you want the numerator terms for V6? (y/n) : **y**

Do you want the numerator terms for V7? (y/n) : **n**



The output file "cmos.det" is approximately four pages long. A partial listing is given in Table C3.

Table C3. Sspice Output File CMOS.DET

Allen and Holberg CMOS Op-Amp

[0 ]	[-GM2	Y 1,2	-GDS1	-GDS2	0	GM5	][V1 ]
[0 ]	[0	-GM1-GDS1	Y 2,3	0	0	0	][V3 ]
[0 ]	[-GM2	-GM2-GDS2	GM4	Y 3,4	-sC1	0	][V4 ]
[0 ]	[0	0	0	-sC1+GM6	Y 4,5	GM7	][V5 ]
[0 ]	[0	0	0	0	0	GM8+GDS8	][V6 ]
[1 ]	[1	0	0	0	0	0	][V7 ]

Y( 1,2 ) = +GM2+GM1+GDS5+GDS2+GDS1

Y( 2,3 ) = +GM3+GDS3+GDS1

Y( 3,4 ) = +sC1+GDS4+GDS2

Y( 4,5 ) = +sC1+sC1+GDS7+GDS6

\*Ignore nodes 10 and higher if present. They are used for internal numbering.

\*\*\*\*\*DENOMINATOR ANALYSIS\*\*\*\*\*

TERMS SORTED ACCORDING TO POWERS OF s

s\*\*2 terms:

- sCL\*sC1\*GM8\*GM3\*GM2 - sCL\*sC1\*GM8\*GM3\*GM1  
 - sCL\*sC1\*GM8\*GM3\*GDS5 - sCL\*sC1\*GM8\*GM3\*GDS2  
 - sCL\*sC1\*GM8\*GM3\*GDS1 - sCL\*sC1\*GM8\*GM2\*GDS3  
 - sCL\*sC1\*GM8\*GM2\*GDS1 - sCL\*sC1\*GM8\*GM1\*GDS3  
 - sCL\*sC1\*GM8\*GDS5\*GDS3 - sCL\*sC1\*GM8\*GDS5\*GDS1  
 - sCL\*sC1\*GM8\*GDS3\*GDS2 - sCL\*sC1\*GM8\*GDS3\*GDS1  
 - sCL\*sC1\*GM8\*GDS2\*GDS1 - sCL\*sC1\*GM3\*GM2\*GDS8  
 - sCL\*sC1\*GM3\*GM1\*GDS8 - sCL\*sC1\*GM3\*GDS8\*GDS5  
 - sCL\*sC1\*GM3\*GDS8\*GDS2 - sCL\*sC1\*GM3\*GDS8\*GDS1  
 - sCL\*sC1\*GM2\*GDS8\*GDS3 - sCL\*sC1\*GM2\*GDS8\*GDS1

.

.

130 lines not shown

\*\*\*\*\*

NUMERICAL VALUE OF ABOVE SYMBOLIC RESULT

- 1.79078e-036 \* s\*\*2 - 3.05371e-029 \* s\*\*1 - 1.14017e-026 \* s\*\*0

\*\*\*\*\*

\*\*\*\*\*NUMERATOR ANALYSIS\*\*\*\*\*

Numerator for V6 is :

TERMS SORTED ACCORDING TO POWERS OF s

s\*\*1 terms:

+ sC1\*GM8\*GM4\*GM2\*GM1 + sC1\*GM8\*GM4\*GM2\*GDS1  
 + sC1\*GM8\*GM3\*GM2\*GM1 + sC1\*GM8\*GM3\*GM2\*GDS5  
 + sC1\*GM8\*GM3\*GM2\*GDS1 + sC1\*GM8\*GM2\*GM1\*GDS3

.

.

15 lines not shown

\*\*\*\*\*

NUMERICAL VALUE OF ABOVE SYMBOLIC RESULT

+ 2.54343e-030 \* s\*\*1 - 1.94226e-022 \* s\*\*0

\*\*\*\*\*

Clearly the full symbolic denominator is so too complicated to be useful. This symbolic "explosion" is a central problem in symbolic analysis. Since we know the values of the transistor parameters (or even just the approximate values), we can simplify the determinants. The same input file is again run but this time with a magnitude threshold of 5% or 0.05. The following are the prompts seen by the user. The user responses are shown in bold:

#### **s Spice**

S Spice - Symbolic SPICE Circuit Analyzer

Version 1.0, 1 June 1990

(C) Copyright 1990 by Michigan State University

Unauthorized copying of this program is prohibited

INPUT FILE NAME [.cir] : **cmos**

OUTPUT FILENAME [cmos.det] : **(hit return)**

Would you like the determinant string sorted according to orders of some variable? (y/n) : **n**

Would you like a numerical evaluation of the results? (y/n) : **y**

Do you want the parameter values to be read from the existing .DAT file? (y/n) : **y**

Would you like to discard some terms from the answer if their relative magnitude falls below a certain threshold? (y/n) : **y**

Please enter the threshold factor (between 0 and 1) : **.05**

Would you like to see the numerator terms? (y/n) : **y**

Would you like to prepare a file to check and solve for any second order filter functions? (y/n) : **n**

Do you want the numerator terms for V1? (y/n) : **n**

Do you want the numerator terms for V3? (y/n) : **n**

Do you want the numerator terms for V4? (y/n) : **n**

Do you want the numerator terms for V5? (y/n) : **n**

Do you want the numerator terms for V6? (y/n) : **y**

Do you want the numerator terms for V7? (y/n) : **n**

Note that S Spice looked for the "cmos.dat" file using the same path as that given for "cmos.cir" and the user was able to avoid having to again supply element values. The user could also edit this file as appropriate in other examples. The approximated results found in "cmos.det" are completely listed in Table C4.

Table C4. Sspice Approximated Output File CMOS.DET

Allen and Holberg CMOS Op-Amp

[0 ]	[-GM2	Y 1,2	-GDS1	-GDS2	0	GM5	][V1 ]
[0 ]	[0	-GM1-GDS1	Y 2,3	0	0	0	][V3 ]
[0 ]	[-GM2	-GM2-GDS2	GM4	Y 3,4	-sC1	0	][V4 ]
[0 ]	[0	0	0	-sC1+GM6	Y 4,5	GM7	][V5 ]
[0 ]	[0	0	0	0	0	GM8+GDS8	][V6 ]
[1 ]	[1	0	0	0	0	0	][V7 ]

Y( 1,2 ) = +GM2+GM1+GDS5+GDS2+GDS1

Y( 2,3 ) = +GM3+GDS3+GDS1

Y( 3,4 ) = +sC1+GDS4+GDS2

Y( 4,5 ) = +sCL+sC1+GDS7+GDS6

\*Ignore nodes 10 and higher if present. They are used for internal numbering.

RESULTS APPROXIMATED USING A THRESHOLD MAGNITUDE OF: 0.05

\*\*\*\*\*DENOMINATOR ANALYSIS\*\*\*\*\*

TERMS SORTED ACCORDING TO POWERS OF s

s\*\*2 terms:

- sCL\*sC1\*GM8\*GM3\*GM2 - sCL\*sC1\*GM8\*GM3\*GM1

s\*\*1 terms:

- sC1\*GM8\*GM6\*GM3\*GM2 - sC1\*GM8\*GM6\*GM3\*GM1

s\*\*0 terms:

- GM8\*GM4\*GM1\*GDS7\*GDS2 - GM8\*GM4\*GM1\*GDS6\*GDS2  
 - GM8\*GM3\*GM2\*GDS7\*GDS4 - GM8\*GM3\*GM2\*GDS6\*GDS4  
 - GM8\*GM3\*GM1\*GDS7\*GDS4 - GM8\*GM3\*GM1\*GDS7\*GDS2  
 - GM8\*GM3\*GM1\*GDS6\*GDS4 - GM8\*GM3\*GM1\*GDS6\*GDS2

\*\*\*\*\*

NUMERICAL VALUE OF ABOVE SYMBOLIC RESULT

- 1.76102e-036 \* s\*\*2 - 2.95851e-029 \* s\*\*1 - 1.12376e-026 \* s\*\*0

\*\*\*\*\*

\*\*\*\*\*NUMERATOR ANALYSIS\*\*\*\*\*

Numerator for V6 is :

TERMS SORTED ACCORDING TO POWERS OF s

s\*\*1 terms:

+ sC1\*GM8\*GM4\*GM2\*GM1 + sC1\*GM8\*GM3\*GM2\*GM1

s\*\*0 terms:

- GM8\*GM6\*GM4\*GM2\*GM1 - GM8\*GM6\*GM3\*GM2\*GM1

\*\*\*\*\*

NUMERICAL VALUE OF ABOVE SYMBOLIC RESULT

+ 2.51826e-030 \* s\*\*1 - 1.92303e-022 \* s\*\*0

\*\*\*\*\*

The denominator determinant went from 140 lines to 6 lines with a 5% approximation.

To see the overall effect on accuracy in this example we can compare the numerical values for each power of s. The exact value of the denominator is:

$$- 1.79078\text{e-}036 * s^{**2} - 3.05371\text{e-}029 * s^{**1} - 1.14017\text{e-}026 * s^{**0}$$

while the approximated value of the denominator is:

$$- 1.76102\text{e-}036 * s^{**2} - 2.95851\text{e-}029 * s^{**1} - 1.12376\text{e-}026 * s^{**0}$$

Similarly, the exact value of the numerator is:

$$+ 2.54343\text{e-}030 * s^{**1} - 1.94226\text{e-}022 * s^{**0}$$

while the approximated value of the denominator is:

$$+ 2.51826\text{e-}030 * s^{**1} - 1.92303\text{e-}022 * s^{**0}$$

The approximated transfer function magnitude and angle is plot with the numeric PSpice results in Figs. C3 and C4. The approximation breaks down at frequencies above 10 MHz where the transistor capacitance begins to take effect.

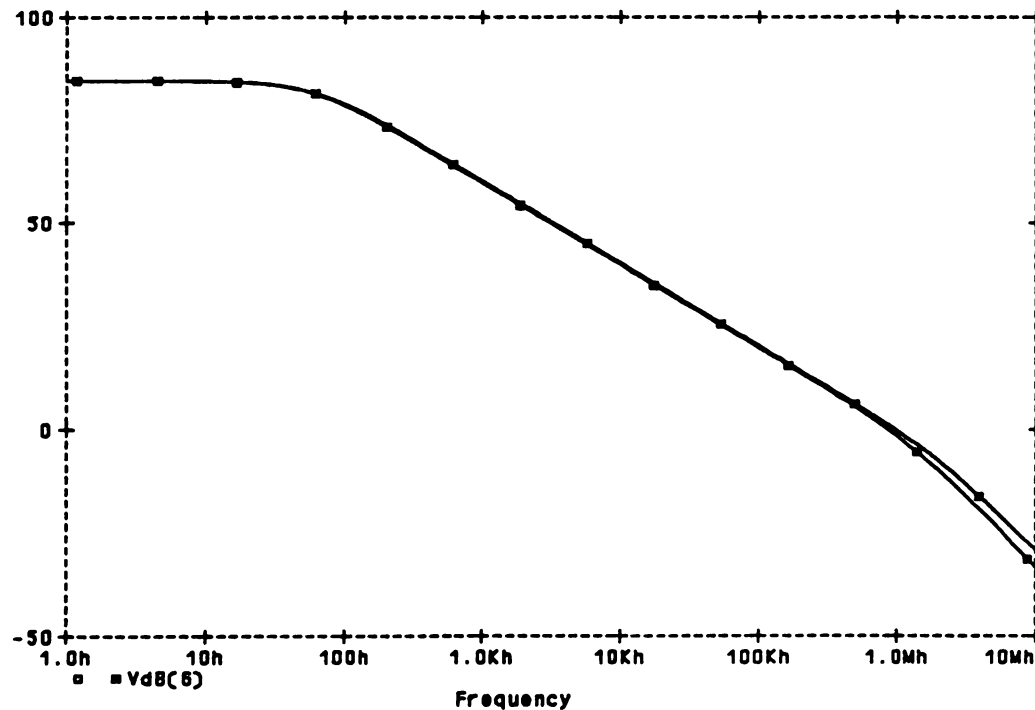


Figure C3. Open loop magnitude comparison of PSpice and Sspice

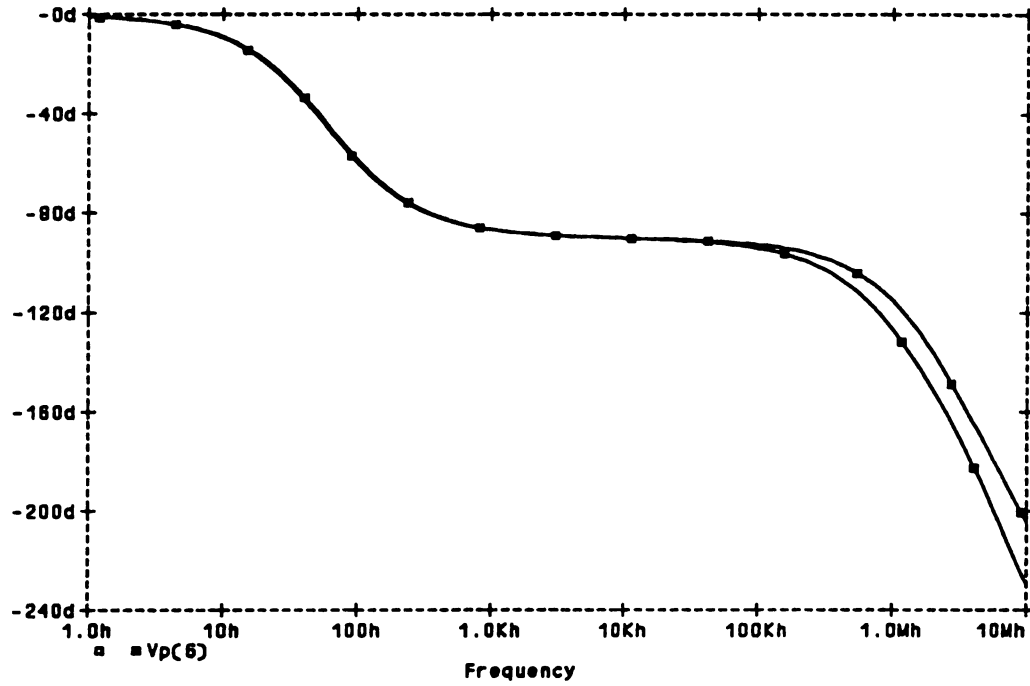


Figure C4. Open loop phase response comparison of PSpice and Sspice

Using the symbolic results of Table C4, the user may wish to further simplify or manipulate the results. Using  $GM1=GM2$  and  $GM3=GM4$  and canceling common factors that result from this, we have

$$\frac{V_6}{V_1} = - \frac{s C_1 G M_1 - G M_6 G M_1}{s^2 C_L C_1 + s C_1 G M_6 + (G D S_7 + G D S_6) (G D S_3 + G D S_1)} \quad (1)$$

Factoring the denominator using the approximated numeric results has two real roots several decades apart. Thus our transfer function is as follows where  $p_2 \gg p_1$ :

$$\frac{V_6}{V_1} = \frac{K(s + z_1)}{(s + p_1)(s + p_2)} \approx \frac{K(s + z_1)}{s^2 + p_2 s + p_1 p_2} \quad (2)$$

Solving for the roots using the approximation in Eqn. 2, we find

$$z_1 = -\frac{GM_6}{C_1} = -2\pi 12.15 \text{ MHz}$$

$$p_2 = \frac{GM_6}{C_L} = 2\pi 2.67 \text{ MHz}$$

$$\begin{aligned} p_1 &= \frac{(GDS_7 + GDS_6)(GDS_3 + GDS_1)}{C_1 GM_6} \\ &= \frac{1}{C_1 GM_6 (RDS_7 \parallel RDS_6)(RDS_3 \parallel RDS_1)} = 2\pi 60.46 \text{ Hz} \end{aligned}$$

The low frequency gain can be found by running the input file with lines containing  $C_1$  and  $C_L$  removed or taking the limit as  $s$  approaches zero in the transfer function of Eqn.

1. Doing this we find,

$$\frac{V_6}{V_1}|_{LP} = \frac{GM_6 GM_1}{(GDS_7 + GDS_6)(GDS_3 + GDS_1)} = 17,065 \Rightarrow 84.6 \text{ dB} \quad (3)$$

## APPENDIX D

### Interactive structure of Sspice

Flow of control in interactive parts of Sspice (modules I and II) is presented below. Besides offering greater flexibility and control to the user, the interactive approach results in a shorter turnaround time in a Unix context. The program also prompts for values of circuit parameters whose numerical values were not specified in the input file if numerical evaluation of the symbolic result is desired (not shown).

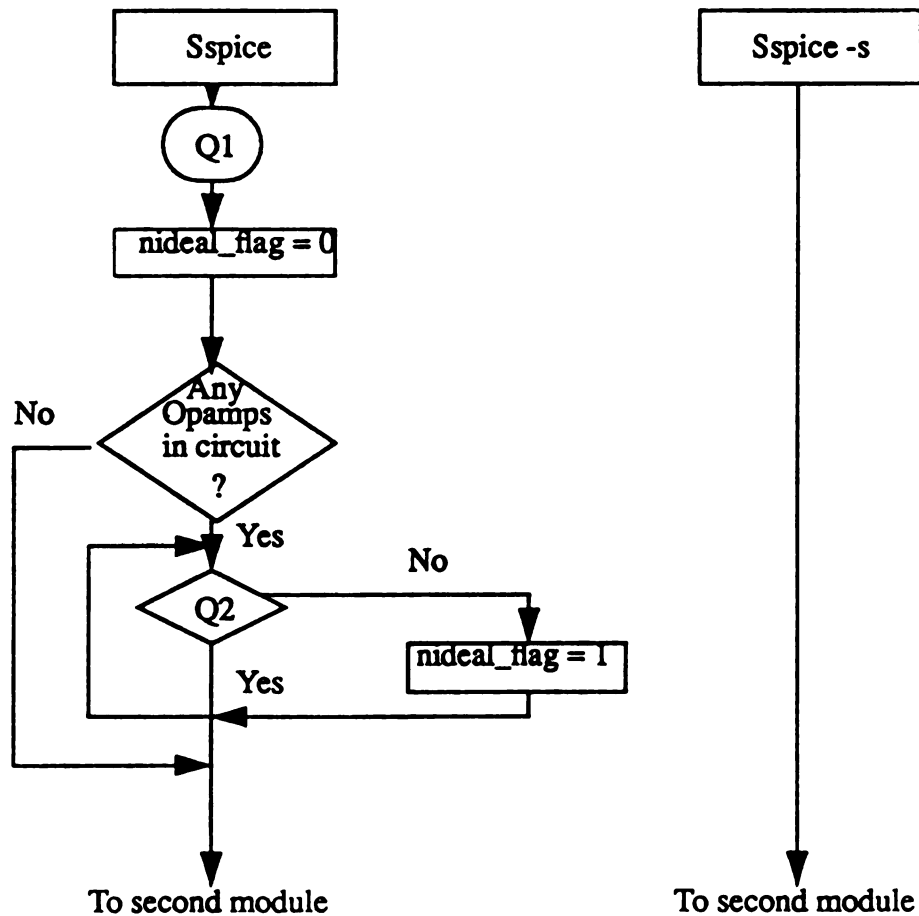


Figure D1. Interactive structure of Module I

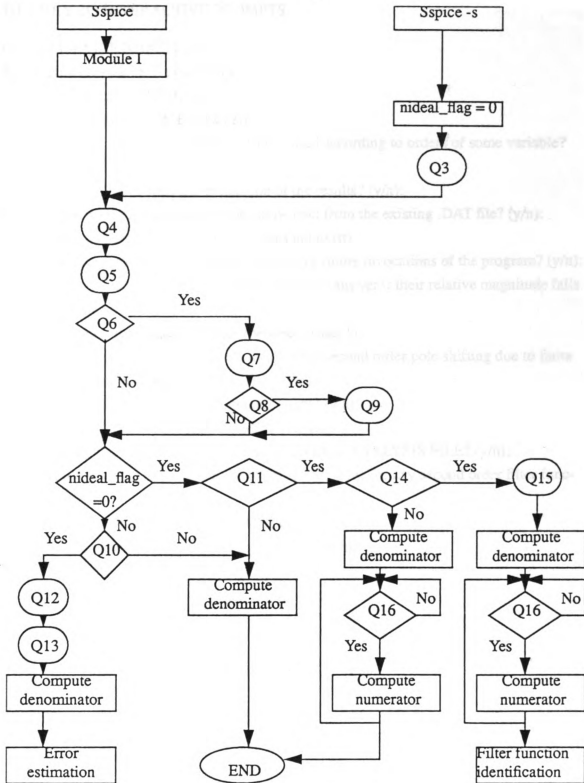


Figure D2. Interactive structure of Module II



## D1.1 KEY TO INTERACTIVE PROMPTS

Q1: INPUT FILE NAME [.cir]:

Q2: Is XOAx<sub>xxx</sub> an ideal opamp? (y/n):

Q3: INPUT FILE NAME [.mtx]:

Q4: OUTPUT FILE NAME [xxx.det]:

Q5: Would you like the determinant string sorted according to orders of some variable?  
(y/n):

Q6: Would you like numerical evaluation of the results? (y/n):

Q7: Do you want the parameter values to be read from the existing .DAT file? (y/n):  
OR (in the event that the .DAT file does not exist)

Do you want to save the parameter values for future invocations of the program? (y/n):

Q8: Would you like to discard some terms from the answer if their relative magnitude falls  
below a certain threshold? (y/n):

Q9: Please enter the threshold factor (between 0 and 1):

Q10: Would you like to prepare a file to solve for second order pole shifting due to finite  
gain-bandwidth-product (GBP in Hz)? (y/n):

Q11: Would you like to see the numerator terms? (y/n):

Q12: ERROR ANALYSIS FILE NAME [xxx.err]:

Q13: Do you want the GBP's equated in the ERROR ANALYSIS FILE? (y/n):

Q14: Would you like to prepare a file to check and solve for any second order filter func-  
tions? (y/n):

Q15: FILTER FUNCTION FILE NAME [xxx.fun]:

Q16: Do you want the numerator terms for V<sub>xx</sub>? (y/n):

## **BIBLIOGRAPHY**

## BIBLIOGRAPHY

1. G.Wilson, Y.Bedri and P.Bowron, "RC-Active networks with reduced sensitivity to amplifier gain-bandwidth-product," IEEE Trans. on Circuits and Systems, CAS-21, pp. 618-626, September 1974.
2. J.A. Svoboda, G.M. Wierzba and T.Reynolds, "Op-amp relocation, the complementary and stability," Proc. 29th Midwest Symposium on Circuits and Systems, August, 1986.
3. G.M. Wierzba and J.A. Svoboda, "An op-amp relocated bandpass filter with zero center frequency sensitivity to the gain-bandwidth-product," Proc. 29th Midwest Symposium on Circuits and Systems, pp. 28-32, August, 1986.
4. G.M. Wierzba and J.A.Svoboda, "A comparison of circuits generated by op-amp relocation," Proc. 23rd Midwest Symposium on Circuits and Systems, pp. 800-804, August 1980.
5. L.T.Bruton, RC-Active Circuits Theory and Design, Prentice Hall, 1980.
6. G.M. Wierzba, "Op-amp relocation: A topological active network synthesis," IEEE Trans. on Circuits and Systems, CAS-33, pp 469-475, May 1986.
7. G.M.Wierzba, A.Srivastava, V.Joshi, K.V.Noren and J.A.Svoboda, "Sspice: A symbolic SPICE program for linear active circuits," Invited, 32nd Midwest Symposium on Circuits and Systems, pp 1197-1201, May 1989.
8. A.Srivastava and G.M. Wierzba, "Symbolic approximation of analog circuits using Sspice.", Invited, 33rd Midwest Symposium on Circuits and Systems, August 1990.

9. P. Sannuti and N.N. Puri, "Symbolic network analysis-An algebraic formulation," **IEEE Trans. on Circuits and Systems**, CAS-27, pp 679-687, August 1980.
10. V. Joshi, "SLAP : Symbolic Linear Analysis Program," MS thesis, Michigan State University, E. Lansing MI, 1987.

MICHIGAN STATE UNIV. LIBRARIES



31293008952883