

8-1-1988

Symbolic Analysis of Large-Scale Networks

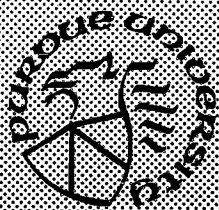
M. M. Hassoun
Purdue University

P. M. Lin
Purdue University

Follow this and additional works at: <https://docs.lib.purdue.edu/ecetr>

Hassoun, M. M. and Lin, P. M., "Symbolic Analysis of Large-Scale Networks" (1988). *Department of Electrical and Computer Engineering Technical Reports*. Paper 611.
<https://docs.lib.purdue.edu/ecetr/611>

This document has been made available through Purdue e-Pubs, a service of the Purdue University Libraries. Please contact epubs@purdue.edu for additional information.



Symbolic Analysis of Large-Scale Networks

**M. M. Hassoun
P. M. Lin**

**TR-EE 88-30
August 1988**

**School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907**

**Supported by National Science Foundation Grant
No. ESC-84-19841**

TABLE OF CONTENTS

	Page
LIST OF TABLES	v
LIST OF FIGURES	vii
ABSTRACT	x
CHAPTER 1 - INTRODUCTION	1
1.1 Introduction	1
1.2 The Network Approach	7
1.2.1 Circuit Partitioning	8
1.2.2 Terminal Block Analysis	8
1.2.3 Hierarchical Middle Block Analysis	10
1.3 Conventions	10
CHAPTER 2 - REVIEW: FLOWGRAPH ANALYSIS OF LARGE ELECTRONIC NETWORKS	13
2.1 Introduction	13
2.2 The Analysis Procedure	14
CHAPTER 3 - THE TERMINAL BLOCK ANALYSIS	19
3.1 Introduction	19
3.2 Overview	19
3.3 The Modified Nodal Analysis	22
3.4 The Reduced Modified Nodal Analysis Matrix (RMNA)	31
3.5 The Sparsity of the MNA Matrix	38
3.6 The Pivoting Problem.....	41

	Page
CHAPTER 4 - THE IDEAL OPERATIONAL AMPLIFIER IN THE MNA AND THE RMNA MATRICES	43
4.1 Introduction	43
4.2 The Nullator, The Norator and The Nullor	43
4.2 The Ideal Opamp Model	44
CHAPTER 5 - THE HIERARCHICAL MIDDLE BLOCK ANALYSIS	50
5.1 Introduction	50
5.2 Middle Block Analysis Model	50
CHAPTER 6 - THE GLOBAL ANALYSIS PICTURE AND THE EFFECTS OF NETWORK PARTITIONING	61
6.1 Introduction	61
6.2 The Global Model	61
6.3 The Effect of the Tearing Nodes on the Analysis	62
6.4 The Advantages of Network Partitioning	65
CHAPTER 7 - PARTITIONING PREPROCESSING	69
7.1 Introduction	69
7.2 Network Topology	69
7.2.1 Trees and Cotrees	70
7.2.2 Weighted Fundamental Loop Matrix	73
7.3 Finding a Tree and its Weighted Fundamental Loop Matrix	75
7.3.1 Finding an initial Tree T_i	75
7.3.2 Building the Weighted Fundamental Loop Matrix B	81
7.3.3 Finding an Optimal Tree T_o	81
7.4 Implementation Notes	84
CHAPTER 8 - PARTITIONING ALGORITHM	85
8.1 Introduction	85
8.2 The Loop Index and the Tearing Index	88
8.3 The Partitioning Criteria	89
8.4 The Global Algorithm	90
8.5 Execution Time Minimization	91

	Page
8.6 The Binary Partitioning Algorithm	95
8.6.1 General Discussion	95
8.6.2 Left Weighted Fundamental Loop Matrix B_l	96
8.6.3 Right Weighted Fundamental Loop Matrix B_r	98
8.6.4 The Binary Algorithm	103
 CHAPTER 9 - COMPARISONS AND EXAMPLES	 112
9.1 Comparison with the Flowgraph Method	112
9.2 General Example	123
 CHAPTER 10 - SUMMARY, CONCLUSIONS AND RECOMMENDATIONS	 131
10.1 Summary and Conclusions	131
10.2 Recommendations	132
 LIST OF REFERENCES	 135
 APPENDICES	
Appendix A: SCAPP Data Structures	141
Appendix B: SCAPP Users' Manual	152
 VITA	 158

LIST OF TABLES

Table	Page
3.1 The sparsity of the MNA matrix	40
9.1 The result of the flowgraph analysis	115
9.2 SCAPP result with manual partitioning	119
9.3 Results for example 9.1	121
9.4 Partitioning table for block 0 (entire circuit)	126
9.5 Binary partitioning table of block 2	127
9.6 Binary partitioning table of block 3	128
9.7 Results for example 9.2	130
Appendix	
Table	
A.1 <i>struct subckt</i> (the binary tree vertex)	141
A.2 <i>struct row</i> (the branch structure)	143
A.3 <i>struct ptr_llist</i> (pointers to branches)	144
A.4 <i>struct nlist</i> (the node structure)	145
A.5 <i>struct nodeptr</i> (Pointers to nodes)	146
A.6 <i>struct exp_llist</i> (expression pointer)	147
A.7 <i>struct symbol_tbl</i> (symbol table element)	148

Appendix Table	Page
A.8 <i>struct mna_ptr</i> (MNA row)	149
A.9 <i>struct mna_element</i> (MNA entry)	150
A.10 <i>struct temp_exp</i> (pointer to branch)	151

LIST OF FIGURES

Figure	Page
1.1 Resistive ladder network	6
1.2 A partitioning binary tree	9
1.3 n -terminal block	11
2.1 A circuit, its Coates graph and its k -connections	16
2.2 Hierarchical analysis	18
3.1 Circuit of example 3.1	24
3.2 Element stamps for the nodal analysis	26
3.3 Voltage source element stamp	28
3.4 VCVS element stamp	28
3.5 CCVS element stamp	30
3.6 CCCS element stamp	30
3.7 Circuit of example 3.2	32
3.8 Circuit of example 3.4	32
4.1 A nullator	45
4.2 A norator	45
4.3 A nullor	46
4.4 An opamp circuit	48

Figure	Page
5.1 Illustration of middle block analysis	51
5.2 A partitioned network	52
5.3 Circuit of example 5.1	58
6.1 Boardered block diagonal (BBD) matrix	63
7.1 A pseudo-branch for a 4-terminal block	71
7.2 A band-pass active filter and its corresponding graph	74
7.3 A local loop and a non-local loop	76
7.4 A linear tree and a star tree	78
7.5 A near-linear tree and a star-like tree	79
7.6 The tree finding algorithm	80
7.7 The tree optimization algorithm	83
8.1 Balanced and unbalanced binary trees	86
8.2 Partitioned graph of example 8.2	97
8.3 Left and right partitions of example 8.3	99
8.4 New tree and cotree of the right partition	104
8.5 Graph of example 8.5	109
8.6 Left and right partitions of example 8.5	110
9.1 Band pass filter [25]	113
9.2 Coates graph representation of example 9.1	114
9.3 Block interconnections for manual partitioning	117
9.4 A single block	117

Figure	Page
9.5 The binary tree model of example 9.1	118
9.6 Circuit of example 9.2	124
9.7 The binary tree model for example 9.2	124
9.8 Input to SCAPP for circuit of example 9.2	125
9.9 B matrix for block 0 (entire circuit)	126

ABSTRACT

A new approach to the problem of symbolic circuit analysis of large-scale circuits is presented in this report. The methodology has been implemented in a computer program called SCAPP (Symbolic Circuit Analysis Program with Partitioning). The method solves the problem by utilizing a hierarchical network approach and the *sequence of expressions* concept rather than a topological approach and the single expression idea which have dominated symbolic analysis in the past. The method employs further modifications to the modified nodal analysis (MNA) technique by allowing ideal opamps in the matrix formulation process and introducing the reduced modified analysis matrix (RMNA). The analysis algorithm is most efficient when network partitioning is used. A node-tearing binary partitioning algorithm based on the concepts of *loop index* and *tearing index* is also presented. The partitioning technique is very suitable for the hierarchical network analysis approach.

CHAPTER 1

INTRODUCTION

1.1 Introduction

The process of simulating the electrical behavior of a circuit using specially developed computer programs is referred to as circuit analysis or circuit simulation. The development of circuit analysis techniques and their computer implementation started taking a new role in the circuit design process during the sixties. The sixties were the turning point because they introduced the concept of integrated circuits (IC's). Before integrated circuits, circuits were built with discrete components on PC boards. The process of verifying the design was to physically build the circuits and test them as the design progressed; the time and financial cost was relatively small. However, the cost and time of building IC test circuits to verify the design grew considerably. Another mean was needed to verify the designs. This is where circuit simulation started realizing its importance.

The growth in the scale of integration from Small-Scale Integration (SSI) to Medium-Scale Integration (MSI) and Large-Scale Integration (LSI), resulted in the circuit size growth from tens of components to thousands of components. This lead to the development of what is referred to as second generation circuit simulation techniques [9,12] in the seventies [10]. Second generation programs that implement these techniques like SPICE2 [21] and ASTAP [36] are still the "standard" used for circuit analysis. With the arrival of the Very Large-Scale Integration (VLSI) era in the eighties with circuit sizes in the tens of thousands of components, third generation analysis techniques, namely relaxation techniques and mixed mode simulation techniques [37,38,39], emerged. These techniques try to make use of the computer hardware advancements in the form of multiprocessor and array processor machines. So they involve the idea of partitioning the problem into several smaller problems that could be distributed over several processors. The implementation of these algorithms in programs like SPLICE [38] and DIANA [39] resulted in much faster and more realistic simulation times for VLSI

circuits. However, they have not yet found wide commercial use because of convergence problems resulting from analog circuits with tight feedback loops.

The approaches in developing all of the above mentioned simulation techniques is a *numerical approach*. The circuit designer assigns numerical values to all the circuit components and then the circuit is simulated using the analysis program. For LSI and VLSI circuits the simulation times exceed 48 hours of continuous simulation. The numerical results produced are then analyzed and some circuit component values are adjusted and another simulation is run. A large number of iterations is needed to achieve the design goals. This occupied a great deal of time from the designers and the computers.

An alternative approach to the problem is *symbolic circuit analysis*. The aim here is to eliminate the iterations through the simulator from the design process. The idea is that some or all of the circuit components remain as symbols throughout the entire simulation. That is, no numerical values assigned to them. Numerical results can then be obtained by evaluating the results of the symbolic analysis at a specific numerical point for each symbol. So only one iteration is needed for the simulation itself and successive evaluations of the results replaces the need for any extra iterations through the simulator.

Symbolic circuit analysis is performed in the frequency domain where the results are in terms of the frequency variable s . The main goal of performing symbolic analysis on a network is to obtain a symbolic transfer function of the form

$$H(s, \mathbf{X}) = \frac{N(s, \mathbf{X})}{D(s, \mathbf{X})} \quad (1.1)$$

where

$$\mathbf{X} = [x_1, x_2, \dots, x_n] \quad n \leq n_{all} \quad (1.2)$$

The expression is a function of the complex frequency variable s , and the variables x_1 through x_n representing the variable network elements, where n is the number of variable network elements and n_{all} is the total number of network elements. *DC* analysis results are obtained by evaluating the resulting transfer function or functions at zero frequency. *AC* analysis can be performed by evaluating $|H(j\omega, \mathbf{X})|$ for different values of ω without the need to rerun any iterations through the simulator.

Several methods have been proposed to address the problem of symbolic circuit simulation. The early work was to produce a transfer function $H(s)$ with the frequency variable s being the only symbolic variable. Programs with these capabilities include: CORNAP [43] and NASAP [44].

The more general case is when some or all of the circuit elements are represented by symbolic variables. The methods developed for this type of analysis all fall under one of the following categories [14]:

1) The tree enumeration method:

Several programs have been produced based on this method [45,46]. The process is based on the concept of finding the determinant of the node admittance matrix [7] by finding the sum of all tree admittance products.

2) The signal flowgraph method:

The methods developed here are based on the idea developed by Mason [47,48] in the 1950's. Formulation of the signal flowgraph and then the evaluation of the gain formula associated with it (Mason's formula) is the basis for symbolic analysis using this method. This method is used in the publically available programs NASAP [44] and SNAP [17].

3) The interpolation method:

This method is best suited for when s is the only symbolic variable in the determinant. It requires the finding of the coefficient of the determinant's polynomial by evaluating it at different values of s . It has been deemed though [49] that using real values for s leads to ill-conditioned equations and leads to inaccurate solutions. Therefore it is best to use complex values for s . Also the Fast Fourier Transform is used to find the coefficients.

4) The parameter extraction method:

This method was introduced in 1973 by Alderson and Lin [2]. Other variations on the method were proposed later in [19,50,51]. The advantage of the method is that it is directly related to the basic determinant properties of widely used equation formulation methods like the modified nodal method [12] and the tableau method [9].

All of the methods previously proposed have severe network size limitations. The network size limits for the the most efficient of those methods are in the range of 50 nodes. In todays VLSI world these methods are not very useful. A single chip, as mentioned earlier, could contain thousands of components and in turn thousands of nodes which makes symbolic simulation of these chips impossible using these methods. There main problem is the

exponential growth of the number of terms involved in the expression for the transfer function, equation (1.1), as the network gets larger. The impedance function for a simple ladder network with only 100 resistors, figure (1.1), would produce 7.9×10^{20} terms for equation (1.1). A storage problem beyond the capabilities of any present or future computer. This problem is solved by utilizing the *sequence of expressions* concept.

The solution lay in a total departure from the traditional procedure of trying to state the transfer function as a single expression directly in terms of the network variables. Instead a *sequence of expressions* procedure is taken. The idea is to produce a succession of small expressions with a hierarchical dependency on each other. That is, the first expression is a function of the second which in turn is a function of the third expression, etc. The growth of the number of expressions in this case is simply linear, which is very reasonable compared to the impractical exponential growth of the conventional methods.

The advantage of having the transfer function stated in a single expression lays in the ability to gain insight to the relationship between the transfer function and the network elements by inspection [15]. For large expressions though, this is not possible, and the single expression loses that advantage. Example 1.1 fully illustrates the advantages that a sequence of expressions has.

Example 1.1

Consider the resistance ladder network in figure (1.1). The goal is to obtain the input impedance function of the network, ($Z_{in} = \frac{V_{in}}{I_{in}}$).

The single expression methods for up to 4 resistances would produce

$$Z_1 = \frac{R_1}{1} \quad (1.3)$$

$$Z_2 = \frac{(R_1 + R_2)}{1} \quad (1.4)$$

$$Z_3 = \frac{R_3 R_1 + R_3 R_2}{R_1 + R_2 + R_3} \quad (1.5)$$

$$Z_4 = \frac{R_4 R_1 + R_4 R_2 + R_4 R_3 R_1 + R_3 R_2}{R_1 + R_2 + R_3} \quad (1.6)$$

The number of terms in the numerator and denominator are given by the Fibonacci numbers satisfying the following difference equation [14]:

$$y(k+2) = y(k+1) + y(k) \quad k = 0, 1, 2, \dots \quad (1.7)$$

$$\text{with } y(0) = 0, y(1) = 1$$

An explicit solution to the above equation is :

$$y(n) = \frac{1}{\sqrt{5}} \left[\frac{1+\sqrt{5}}{2} \right]^n - \left[\frac{1-\sqrt{5}}{2} \right]^n \quad (1.8)$$

$$\sim 0.168 \times 1.618^n \quad \text{for large } n$$

The solution shows that the number of terms in Z_n increases exponentially with n . This is the best that can be achieved using any of the traditional methods [3,17]. The single expression transfer function has this inherent limitation.

Now using the sequence of expressions procedure the input impedance can be obtained from the following expressions:

$$Z_1 = R_1 \quad (1.9)$$

$$Z_2 = R_2 + Z_1 \quad (1.10)$$

$$Z_3 = \frac{R_3 Z_2}{R_3 + Z_2} \quad (1.11)$$

$$Z_4 = R_4 + Z_3 \quad (1.12)$$

$$Z_5 = \frac{R_5 Z_4}{R_5 + Z_4} \quad (1.13)$$

It is obvious for each additional resistance added the sequence of expressions will grow by one expression, either of the form $R_i + Z_{i-1}$ or $\frac{R_i Z_{i-1}}{R_i + Z_{i-1}}$. The number of terms in the sequence of expressions can be given by

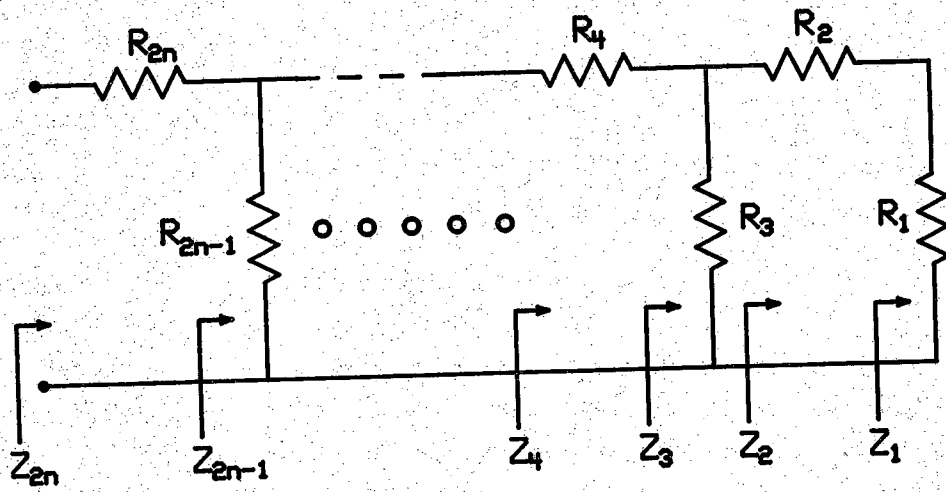


Figure 1.1 : Resistive ladder network

$$y(n) = \begin{cases} 2.5n-2 & , \text{ for } n \text{ even} \\ 2.5n-1.5 & , \text{ for } n \text{ odd} \end{cases} \quad (1.14)$$

which exhibits a linear growth with respect to n .

So, to find the input impedance of a 100 resistor ladder network the single expression methods would produce 7.9×10^{20} terms which requires unrealistically huge computer storage capabilities. On the other hand, the *sequence of expressions* method would produce only 248 terms, which is even within the scope of some desk calculator these days.

Another advantage of the *sequence of expressions* is the number of mathematical operations needed to evaluate the transfer function. To evaluate Z_9 for the above ladder network, the *single expression* methods would require 302 multiplications and 87 additions. The sequence of expressions method would only require 8 multiplications and 8 additions, a huge reduction in computer evaluation time. All this makes the concept of symbolic circuit simulation of large-scale networks very possible.

A topological analysis method for symbolic simulation of large-scale circuits has been proposed by Starzyk and Konczykowska in [25]. The analysis utilizes the *sequence of expressions* idea to obtain the transfer functions. A description of this method is presented in chapter 2, and comparisons are made with some of the results from this project in chapter 8.

1.2 The Network Approach

The symbolic analysis methodology proposed in this report is based on a hierarchical network partitioning and recombination approach to the problem. It is aimed at analyzing large circuits in the size range of thousands of nodes. The result of this project is the production of a Symbolic Circuit Analysis Program with Partitioning (SCAPP). This consisted of the development of the methodology and the algorithms for the analysis and implementing them in a computer program in C language [40]. One of the main issues that was kept in perspective is the utilization of multiprocessor computer systems. This required the design of parallel algorithms that will allow the analysis to be performed concurrently on several parts of the circuit. The process is divided into the following parts:

- 1) Circuit partitioning.
- 2) Subcircuit analysis, referred to as terminal block analysis.
- 3) Hierarchical analysis, referred to as middle block analysis.

The input to SCAPP is a description of the circuit elements by and element and their interconnections (the circuit topology). The elements allowed are resistors, capacitors, inductors, current and voltage controlled current and voltage sources, and ideal operational amplifiers (opamps). SCAPP also allows for user defined subcircuits.

Appendix B is the user's manual for SCAPP. It illustrated the usage and syntax for the input language to SCAPP.

1.2.1 Circuit Partitioning

After parsing the circuit elements, the automatic partitioning is optional. The user may specify their own partitions using the *.subckt* and *.ends* options, request SCAPP to perform the partitioning controlling the process through a collection of options using the *.options* command or choose to perform the analysis on the entire circuit without any partitioning. The partitioning algorithm, which is really a collection of algorithms that would result in the best partitioning suitable for the analysis, is a highly parallelizable heuristic binary process that is based on the concepts of *near-optimal tree* and *loop index* [29]. The process is modeled using a binary tree [1], figure (1.2). The leaves of the tree represent the final subcircuits and each non-leaf vertex represents a binary partitioning operation. The process attempts to minimize the number of nodes and the size of each partition. The development and implementation of the partitioning algorithms are fully explained in chapters 6 and 7.

1.2.2 Terminal Block Analysis

Each subcircuit (terminal block) produced by the partitioner has to be modeled as an n -terminal block, figure (1.3). Each block is generally characterized by an $(n+m \times n+m)$ matrix Y , where n is the number of tearing nodes for this block and m is the number of extra current variables needed in order to characterize controlled voltage sources plus the number of extra current variables requested by the analysis, if any. The Y matrix is a Reduced Modified Nodal Admittance (RMNA) matrix. This terminal block analysis process is totally independent for each subcircuit; it is not an iterative procedure. So each subcircuit may be processed in a different processor on a

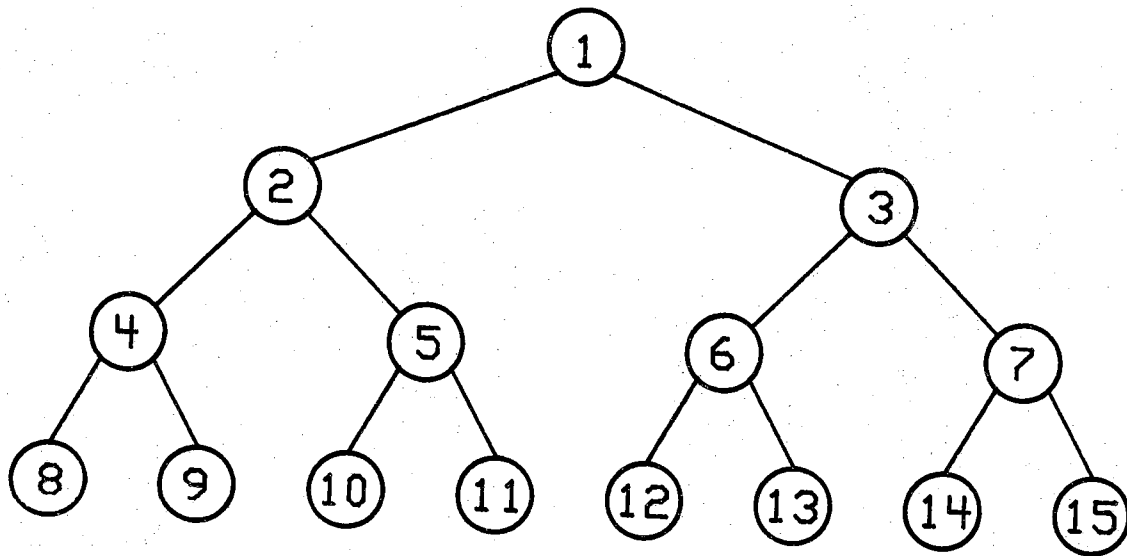


Figure 1.2 : A partitioning binary tree

multiprocessor machine. The result of the terminal block analysis is an RMNA matrix for each subcircuit. The theory and development of the RMNA matrix is presented in chapter 3 and the terminal block analysis implementation is explained in chapter 4.

1.2.3 Hierarchical Middle Block Analysis

In a physical sense this involves the upward hierarchical combination of the n -terminal blocks representing the subcircuits to produce the final result. In an analytical sense it is the binary combination of the RMNA matrices to produce one final RMNA matrix Y_o with dimensions $(n_o + m_o \times n_o + m_o)$, where n_o is the number of node voltages requested by the user, and m_o is the number of current variables requested by the user or irreducible by the analysis (section 3.6). The combination process traces the same but opposite path of the binary tree that models the partitioning, figure (1.1). It traces up the binary tree. Every time two subcircuits are combined, the leaves representing these subcircuits are deleted from the binary tree making their parent vertex a leaf itself. The process is continued until only the root vertex is left. Again the parallelism is very obvious in this analysis. However the scheduling of the processes must be handled with extreme care because of the dependencies generated by the structure of the partitioning. The middle block analysis process is presented in chapter 5.

1.3 Conventions

These are several conventions that are adopted throughout this report:

- 1) Both subtraction and division operations are counted as addition and multiplication operations, respectively, when calculating the total numbers of mathematical operations a certain method performs. This convention simplifies and makes clearer the comparisons between the different methods.
- 2) When comparing different *sequence of expressions* to each other, the measure of goodness is the number of multiplications and the number of additions in each sequence, the number of equations is not considered significant at all. This is justified because for computer evaluation of the expressions, an equation is simply an assignment operation which has minimal or no cost compared to an addition or a multiplication operation.
- 3) Since a modification of the nodal admittance matrix is used (chapter 3), all elements are represented by their admittance value.

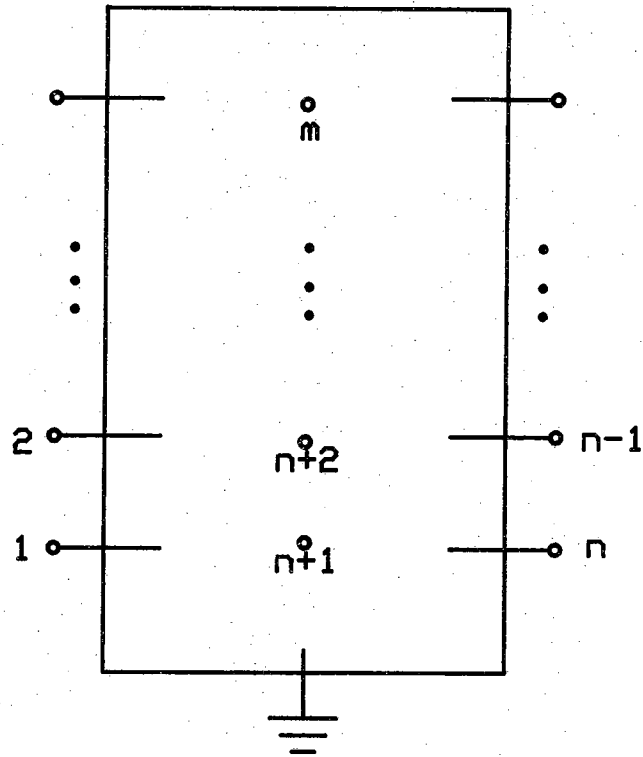


Figure 1.3 : n -terminal block

- 4) All resistors with an impedance value represented by the symbol R will have an admittance value represented by the symbol G . So it is always understood that $G = \frac{1}{R}$.
- 5) For the purpose of symbolic analysis, as mentioned earlier, the entire analysis is performed in the frequency domain with the aid of the complex frequency variable s . The Laplace transform representation of the value of any circuit element is therefore used in the equation formulation process. It is beyond the scope of this report to explain the conversion into the frequency domain. A detailed explanation of the conversion is presented in [35]. A resistor with impedance value R in the time domain has an admittance value of $\frac{1}{R}$ in the frequency domain, or, by the previous convention, simply G . The complex frequency variable s is not involved because a resistor is not a charge storage device. For charge storage devices; a capacitor with time domain impedance value of C has a frequency domain admittance value of sC and an inductor of time domain impedance value of L has a frequency domain admittance value of $\frac{1}{sL}$.

CHAPTER 2

REVIEW: FLOWGRAPH ANALYSIS OF LARGE ELECTRONIC NETWORKS

2.1 Introduction

Symbolic circuit simulation usage has been mostly limited to universities and research institutions. They have not yet found their way into the industry. This is mainly due the limitation that all existing symbolic circuit analysis software can only handle very small circuits. That is, networks in the range of 50 nodes.

A recent approach to the problem of symbolic simulation of large-scale circuits has been proposed by Starzyk and Konczykowska [25]. The method is based on the topological analysis of electrical networks. The basic idea is to represent a certain network, which is to be analyzed, by a flowgraph that fully describes the elements of the network (flowgraph branches) and their interconnections (flowgraph nodes). The result desired from a symbolic topological analysis is a rational function of a specific output variable to a specific input variable. That is,

$$H(s, x_1, x_2, \dots) = \frac{N(s, x_1, x_2, \dots)}{D(s, x_1, x_2, \dots)} \quad (2.1)$$

The expressions obtained above are functions of the complex frequency variable s , and the variables x_1 through x_n representing the values of the network elements, where n is the number of variable network elements. For large networks the above expression could become very complex, specially as n gets larger. Storing and evaluating such a complex expression using a computer would become highly impractical and in some cases even impossible. Starzyk and Konczykowska [25] solved the problem by decomposing the flowgraph into several smaller graphs and then performing an upward hierarchical graph analysis technique [13]. This would produce a *sequence of expressions* which would exhibit an upward hierarchical dependency.

For the construction of a flowgraph that represent a network, the Coates' flowgraph representation [8] was chosen by the Starzyk and Konczykowska, though their method could be modified to suit other flowgraph representations.

2.2 The Analysis Procedure

For the Coates' graph, the resulting transfer function is expressed as the ratio of the graph's 1-connection, which is dependent on the input variable and the output variable selected, to the graph's 0-connection, which is global to all the network's transfer functions (ie. independent of the I/O variables selected) [8]. A generalization of Coates' 1-connection and 0-connection is referred to as a k-connection [25]. To illustrate the concept of a k-connection the proper definition of a directed graph must be presented first [1].

Definition 2.1

A directed graph $G=(V,E)$ consists of a finite, nonempty set of vertices V and a set of edges E which are ordered pairs $e_i=(v_i,u_i)$ of vertices; v_i is called the tail and u_i the head of edge e_i . Each edge $e_i \in E$ has a weight w_i associated with it.

Definition 2.2 [25]

A k-connection (multiconnection) of a graph G with n nodes, is a subgraph p_j with n nodes, α node-disjoint directed edges, β isolated nodes and λ node-disjoint directed loops, where $k=\alpha+\beta$. The weight of a k-connection is given by

$$|p_j| = \prod_{e_i \in p_j} w_i \quad (2.2)$$

and the weight of a set of multiconnections P is given by

$$|P| = \sum_{p_j \in P} \text{sign } p_j \quad (2.3)$$

where

$$\text{sign } p = (-1)^{n+k+l} \text{ord}(v_1, \dots, v_k) \cdot \text{ord}(u_1, \dots, u_k) \quad (2.4)$$

$$\text{ord}(q_1, \dots, q_k) = \begin{cases} 1 & \text{when the number of} \\ & \text{permutations ordering the set is even} \\ -1 & \text{otherwise} \end{cases}$$

(l_p number of loops in multiconnection p).

The transfer function of a network with input variable v_1 and output variable v_2 is therefore,

$$H(s, x_1, x_2, \dots) = \frac{|P_{1\text{-connection}_{v_1 \rightarrow v_2}}|}{|P_{0\text{-connection}}|} \quad (2.5)$$

The $1\text{-connection}_{v_1 \rightarrow v_2}$ is the set of all 1-connections that include a directed path (a set of directed edges) that connect graph nodes v_1 to v_2 .

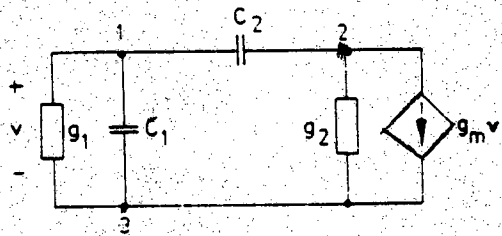
Example 2.1

As an example consider the circuit in figure (2.1). To find $\frac{V_2}{V_1}$, the $1\text{-connection}_{v_1 \rightarrow v_2}$ is needed. Only one such subgraph exists, figure (2.1b). However the 0-connection set has two members illustrated in figure (2.1c), so

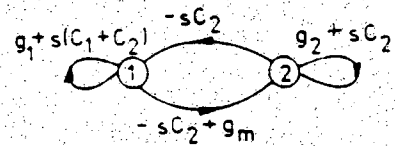
$$\begin{aligned} H(s, g_1, g_2, g_m, C_1, C_2) &= \frac{|P_{1\text{-connection}_{v_1 \rightarrow v_2}}|}{|P_{0\text{-connection}}|} \\ &= \frac{-(-sC_2 + g_m)}{(g_2 + sC_2)} \end{aligned} \quad (2.6)$$

The above described Coates' graph technique was utilized by Starzyk and Konczykowska to develop their topological analysis algorithm [25]. The main steps to the algorithm are :

- 1) Construction of the Coates' graph for the network.
- 2) Decomposition of the graph into several smaller subgraphs referred to as terminal blocks. There are several decomposition techniques that could be used at this stage. These terminal blocks must be of a size that would allow the Coates' graph analysis method to be applied without producing expressions for the k-connections that



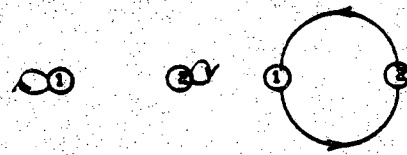
(a) A circuit



(b) Coates graph



(c) 1-connection



(d) 0-connections

Figure 2.1 : A circuit, its Coates graph and its k-connections

are too complex resulting in huge memory requirements and slow evaluation times. The problem of graph partitioning is an *NP-complete* problem [1]. Therefore finding an *optimal* algorithm to partition large graphs in reasonable time is impossible. However, there exists sub-optimal algorithms that would result in efficient enough partitions for this specific analysis purpose in reasonable time. One such efficient algorithm has been proposed in [24].

- 3) Analysis of the terminal blocks. This step is basically finding the Coates' graph multiconnections for all the terminal blocks.
- 4) Analysis of the middle blocks. A middle block is the union of two blocks which could be any combination of terminal and middle blocks. The process of analyzing a middle block consists of combining the k -connections obtained in step 2. This is where the method gets its upward hierarchical analysis label. The best way to represent this combinational analysis technique is by the use of a binary tree. Figure (2.2) illustrated a binary tree for a graph decomposed into five subgraphs. Terminal blocks 8 and 9 are combined to produce middle block 4, then middle block 4 and terminal block 7 are combined to produce middle block 3. The process of analyzing a middle block consists of combining the k -connections for its two descendants to find the k -connections for the subgraph represented by the union of these two block. The process is continued until block 1 is reached. At that point the 0-connection and the 1-connection expressions are found, and the transfer function would then be the ratio of the two.

Chapter 8 illustrates a detailed example of the above method. It compares the results of the analysis to the network approach method developed in this project.

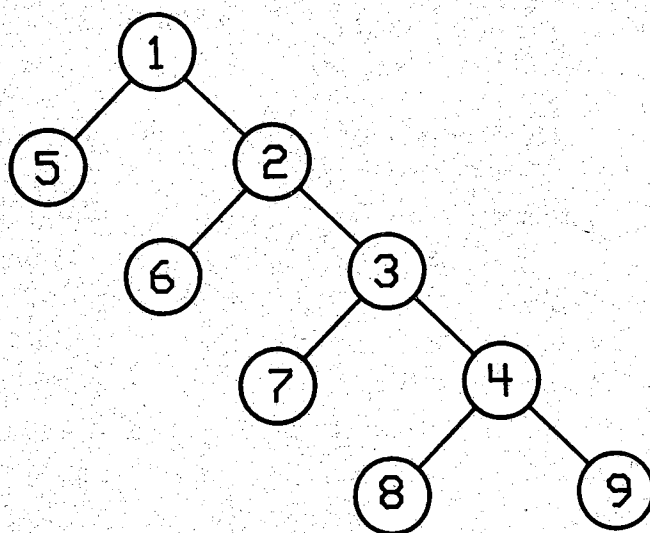


Figure 2.2 : Hierarchical analysis

CHAPTER 3

THE TERMINAL BLOCK ANALYSIS

3.1 Introduction

Terminal block analysis is the analysis of a circuit represented by an n -terminal block in order to characterize its electrical behavior. For the symbolic analysis methodology presented in this report and implemented in SCAPP, the characterization is done in terms of the block's terminal node voltages and some extra branch currents. This chapter presents the terminal block analysis methodology.

3.2 Overview

The heart of any circuit simulator, *symbolic* or *numeric*, is the circuit analysis methodology used to obtain the mathematical equations that characterize the behavior of the system. The circuit must satisfy these equations. These equations are of three kinds, Kirchoff's Current Law (KCL), Kirchoff's Voltage Law (KVL), and the Branch Relationship equations (BR) [7]. These laws are defined as follows :

KCL: *The algebraic sum of currents traversing each cutset of the network must always equal zero.* Always a set of linear equations. Cutset analysis [7,21] makes use of this law. Considering a special case of cutsets: an isolated node, would enable the restatement of KCL in its more commonly known form, which is:

The algebraic sum of currents leaving any node of the network must always equal to zero. Nodal analysis [7] makes use of this special case of the KCL.

KVL: *The algebraic sum of voltages around each loop of the network must always equal zero.* Always a set of linear equations. Loop analysis [7,21] makes use of this law.

BR — For independent branches, *a branch relationship defines the branch current in terms of the branch voltages.* The most common independent branches are resistors, capacitors and inductors. For the case of

dependent branches, *the branch current or voltage is defined in terms of other branches' currents or voltages*. The four common types of dependent branches are the four types of dependent sources: voltage controlled voltage source (VCVS), voltage controlled current source (VCCS), current controlled voltage source (CCVS) and current controlled current source (CCCS). A linear element would produce a BR equation that is linear, and a nonlinear element would produce a nonlinear BR equation. The scope of this report is only concerned with linear elements, therefore, the discussion herein will be limited to that subset of elements. If a nonlinear element is to be included in the circuit, a collection of linear elements could be used to closely model the behavior of that element over a certain frequency range [7]. BR equations are not used on their own to formulate a system of equations to characterize the network. They are however used in conjunction with KCL or KVL equations to produce a more complete set of equations to characterize the network. Hybrid analysis [7] uses a large set of BR equations in its formulation.

A system of equations to solve for the circuit variables: a collection of node voltages and branch currents, does not need to include all of the above equations. Several analysis methods are based on these equations or a subset thereof. Mathematically speaking, an equal number of circuit unknowns and linearly independent equations is required in order for a solution for the system to exist.

The most common analysis methods used in circuit simulators are:

- 1) A modification of nodal analysis [7,12] as used in the numeric circuit simulators SPICE2 [21], and SLIC [52], and in the formulation of the symbolic equations in [2]. Nodal analysis [7] uses KCL to produce the set of equations that characterize the circuit. Its advantage is its relatively small number of equations which results in a smaller matrix which is very desirable in computer implementations. Its limitation however, is that it only allows for node-to-datum voltages as variables. No branch current variables are allowed. Also the only type of independent power sources it allows are independent current sources. However, voltage sources are handled by using their Norton equivalent [35]. A further limitation to this method is that only voltage controlled current sources are allowed, none of the other dependent sources are allowed in the analysis. To overcome these limitations, the Modified Nodal Analysis (MNA) method was

proposed in [12]. It allows branch current as variables in the analysis, which in turn led to the ability to include voltage sources and all four types of controlled sources in the analysis. The MNA method is described in more detail in the next section because it is used in SCAPP after further modification to it. These modifications are necessary in order for the analysis to accept ideal opamps in the analysis, and to produce an n -terminal block characterization matrix. This matrix is referred to as the Reduced MNA (RMNA) matrix.

- 2) Hybrid analysis [7] is used in the numeric simulators ASTAP [36] and ECAP2 [53]. The method requires the selection of a network tree (section 6.2) and its associated cotree. It uses cutset analysis (KCL) and loop analysis (KVL) to formulate the set of equations. The equations produced solve for the branch voltages and branch currents. The method is able to accommodate voltage sources and all four types of dependent sources. It utilizes a much larger matrix than the nodal analysis methods; however, it is a very sparse matrix (lots of zero entries). The main drawback is that the behavior of the equations is highly dependent on the tree selected. A bad tree could easily result in an ill-conditioned set of equations. Some methods of tree selection can guarantee a well-conditioned set of hybrid equations [21]. Therefore special rules must be observed in the tree selection process. That constitutes a large additional overhead on the analysis.
- 3) Sparse tableau analysis [9] has been used in symbolic analysis to employ the parameter extraction method (section 1.2). It uses the entire set of circuit variables: node voltages, branch voltages, branch currents, for the symbolic formulation in addition to capacitor charges and inductor fluxes in the case of numeric simulations. This results in a huge set of equations and in turn a very large matrix but a very sparse one. This method uses all the above stated equation formulation methods, KCL, KVL and the BR. The entire system of equations is solved for all the circuit variables. The disadvantage of the method is its inherent problem of ill-conditioning [21].

It has been found that the sparse tableau method only produces a very minor improvement in the number of mathematical operations over the nodal analysis methods despite its large overhead, stemming from a large matrix and the large and rigid set of variables that have to be solved for. The MNA method gives the choice of what branch current variables to be solved for and equations are added accordingly. The tableau method

would always have a fixed size matrix and produce unneeded information to the user.

For the purpose of symbolic analysis, as mentioned earlier in chapter 1, the entire analysis is performed in the frequency domain with the aid of the complex frequency variable s . The Laplace transform representation of the value of any circuit element is therefore used in the formulation. Also, since a modification of the nodal admittance matrix is used, all elements are represented by their admittance values. It is beyond the scope of this report to explain the conversion into the frequency domain. A detailed explanation of the conversion is presented in [35]. A resistor with impedance value R in the time domain has an admittance value of $\frac{1}{R}$ in the frequency domain, or, by the convention of section 1.3, G . The complex frequency variable s is not involved because a resistor is not a charge storage device. For charge storage devices, a capacitor with time domain impedance value of C has a frequency domain admittance value of sC and an inductor of time domain impedance value of L has a frequency domain admittance value of $\frac{1}{sL}$.

3.3 The Modified Nodal Analysis

The initial step of MNA is to formulate the nodal admittance matrix \mathbf{Y} [7] from the circuit. The circuit variables considered here are all the node-to-datum voltages, referred to simply as node voltages; there are n_v of them. They are included in the variable vector \mathbf{V} . So \mathbf{V} has the dimensions of $(n_v \times 1)$. The vector \mathbf{J} , also of dimension $(n_v \times 1)$, represents the values of all independent current sources in the circuit. The i th entry of \mathbf{J} represents a current source entering node i . The nodal linear system of equations can be represented in the following matrix form:

$$\mathbf{YV} = \mathbf{J} \quad (3.1)$$

Row i of \mathbf{Y} represents the KCL equation at node i . \mathbf{Y} is constructed by writing KCL equations at each node except for the datum node. The i th equation then would state that the sum of all currents leaving node i is equal to zero. The equations are then put into the matrix form of equation (3.1). The following example illustrates the process.

Example 3.1

Consider the circuit in figure (3.1). Collecting the node voltages would produce the following \mathbf{V} :

$$\mathbf{V} = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} \quad (3.2)$$

Considering all the current sources in the circuit, the vector \mathbf{J} becomes:

$$\mathbf{J} = \begin{bmatrix} J_1 \\ J_4 \\ 0 \end{bmatrix} \quad (3.3)$$

Notice the last entry of \mathbf{J} is a zero because no independent current sources are connected to it.

Now writing KCL equations at the three non-datum nodes produces:

$$G_2 v_1 + G_3(v_1 - v_2) + \frac{1}{sL_7}(v_1 - v_3) = J_1 \quad (3.4)$$

$$G_5 v_2 + G_3(v_2 - v_1) + sC_6(v_2 - v_3) = J_4 \quad (3.5)$$

$$sC_6(v_3 - v_2) + \frac{1}{sL_7}(v_3 - v_1) + g_8(V_{R_3}) = 0 \quad (3.6)$$

Substituting $V_{R_3} = v_1 - v_2$ in the above three equations and rearranging their variables results in:

$$(G_2 + G_3 + \frac{1}{sL_7})v_1 - G_3 v_2 - \frac{1}{sL_7} v_3 = J_1 \quad (3.7)$$

$$-G_3 v_1 + (G_3 + G_5 + sC_6)v_2 - sC_6 v_3 = J_4 \quad (3.8)$$

$$(g_8 - \frac{1}{sL_7})v_1 - (g_8 + sC_6)v_2 + (\frac{1}{sL_7} + sC_6)v_3 = 0 \quad (3.9)$$

Now realizing the form of equation (3.1) would yield the nodal admittance matrix of this circuit \mathbf{Y} .

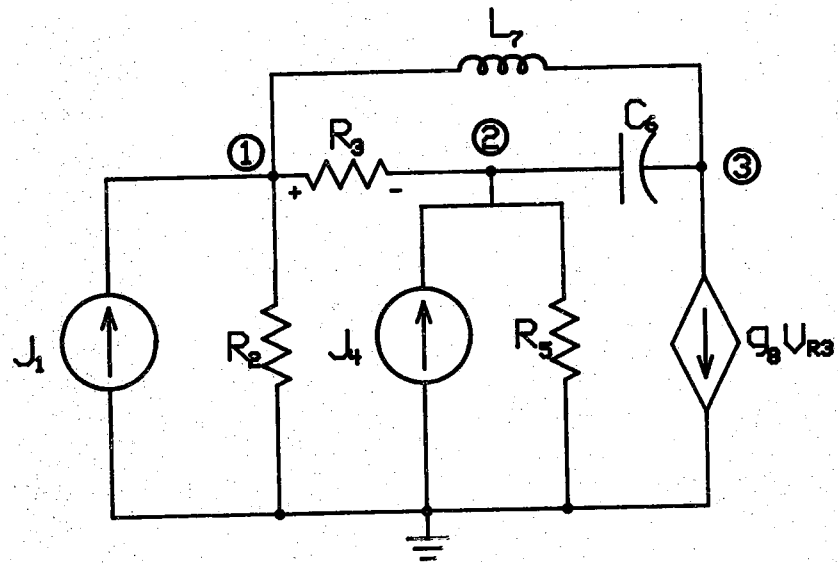


Figure 3.1 : Circuit of example 3.1

$$\begin{bmatrix} G_2+G_3+\frac{1}{sL_7} & -G_3 & -\frac{1}{sL_7} \\ -G_3 & G_3+G_5+sC_6 & -sC_6 \\ g_8-\frac{1}{sL_7} & -(g_8+sC_6) & sC_6+\frac{1}{sL_7} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} J_1 \\ J_4 \\ 0 \end{bmatrix} \quad (3.10)$$

An automatic technique to construct the nodal admittance matrix is the element stamp method [12]. The method constitutes going through each branch of the circuit and adding its contribution to the nodal admittance matrix in the appropriate positions. It is a very nice and easy way to illustrate the impact of each element on the matrix. Figure (3.2) shows the element stamp for any conductance (G , sC or $\frac{1}{sL}$), for an independent current source and for a voltage controlled current source (VCCS). These are the only types of elements allowed by the nodal analysis. Resolving example 3.1 using the element stamps would produce exactly the same \mathbf{Y} matrix, \mathbf{V} vector and \mathbf{J} vector as in equation (3.10). In the computer implementation for this project, the element stamp method was not used. A more efficient modification of it was used because of the sparse matrix storage programming technique implemented (appendix A). The technique requires building the matrix row by row. An element stamp could require accessing up to three rows to enter its contribution into the matrix. This would severely impede the efficiency of the program. The reason element stamps continue to be discussed here is because of their great illustration power of the concept of building the admittance matrix.

The modified nodal analysis technique introduced in [12] expands on the above nodal analysis in order to readily include independent voltage sources and the three other types of controlled sources: VCVS, CCCS and C CVS. This is done by introducing some branch currents as variables into the system of equations which in turn allows for the introduction of any branch current as an extra system variable. Each extra current variable introduced would need an extra equation to solve for it. The extra equations are obtained from the branch relationship (BR) equations for the branches whose currents are the extra variables. The effect on the nodal admittance matrix is the deletion of any contribution in it due to the branches whose currents have been declared as variables. This matrix is referred to as \mathbf{Y}_n . The addition of extra variables and a corresponding number of equations to the system results in the need to append extra rows and extra columns to \mathbf{Y}_n . The augmented \mathbf{Y}_m

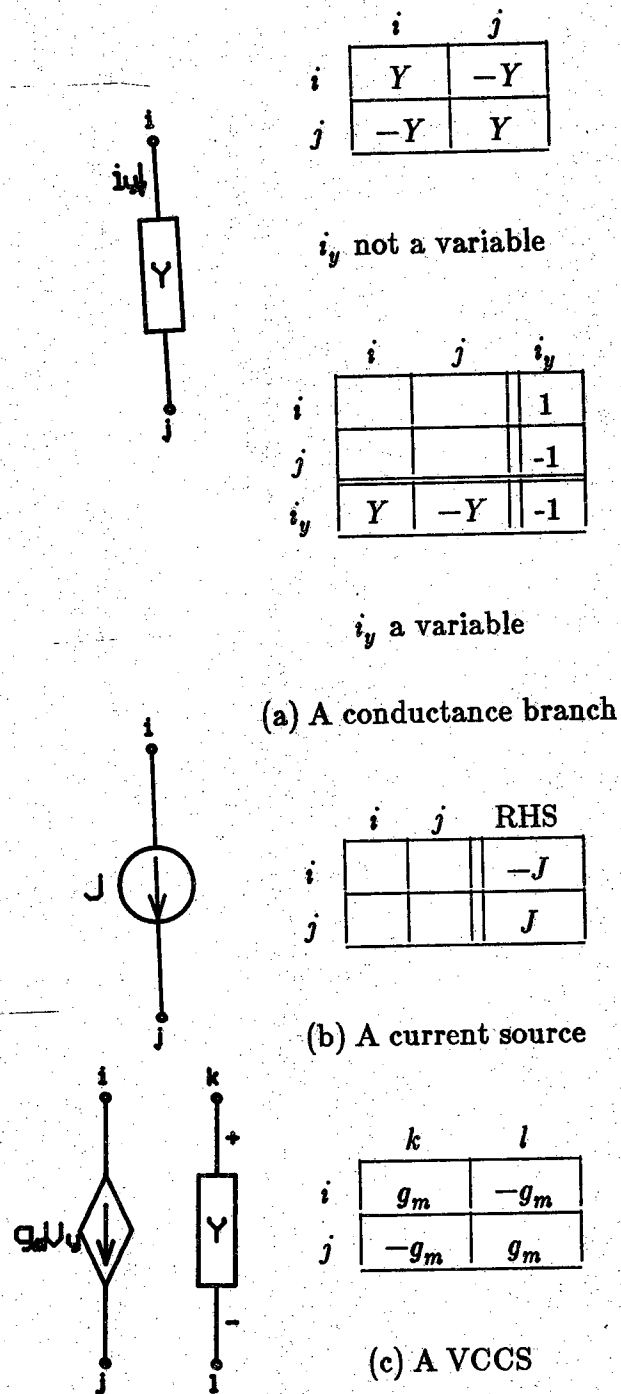


Figure 3.2 : Element stamps for the nodal analysis

matrix is referred to as the MNA matrix. The new system of equations, in matrix form, is:

$$\begin{bmatrix} \mathbf{Y}_n & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix} \begin{bmatrix} \mathbf{V} \\ \mathbf{I} \end{bmatrix} = \begin{bmatrix} \mathbf{J} \\ \mathbf{E} \end{bmatrix} \quad (3.11)$$

where \mathbf{I} is a vector of size n_i whose elements are the extra branch current variables introduced, \mathbf{E} is the independent voltage source values, and \mathbf{C} and \mathbf{D} correspond to BR equations for the branches whose currents are in \mathbf{I} .

According to [12] the MNA current variables should include all branch currents of independent voltage sources, controlled voltage sources and all controlling currents. In this project however, with some extra manipulations for CCVS's and CCCS's, the number of extra current variables required by the MNA could be reduced, in turn reducing the size of the MNA matrix. Each of the new elements that the MNA allows over the nodal analysis is examined in the following steps. These elements, as mentioned earlier, are:

- 1) *Independent voltage source* (figure 3.3). $i_v \in \mathbf{I}$ is the extra current variable and the extra equation is the BR provided by the independent voltage source, which is :

$$v_1 - v_2 = V \quad (3.12)$$

So V becomes an entry in \mathbf{E} . Also i_v will appear in the KCL equations at nodes 1 and 2. The element stamp for an independent voltage source is illustrated in figure (3.3).

- 2) *Voltage controlled voltage source*, VCVS (figure 3.4). $i_{vcvs} \in \mathbf{I}$ is the extra current variable and the extra equation is the BR provided by the VCVS, which is :

$$v_1 - v_2 = \mu V_y \quad (3.13)$$

Since $V_y = v_3 - v_4$, equation (3.13) becomes

$$v_1 - v_2 - \mu v_3 + \mu v_4 = 0 \quad (3.14)$$

The element stamp for a VCVS is illustrated in figure (3.4).

- 3) *Current controlled voltage source*, CCVS (figure 3.5). The MNA proposed in [12] would require that both the dependent voltage source current i_{ccvs} and the controlling current i_y be included as variables. That is however unnecessary. Recognizing that $i_y = Y(v_3 - v_4)$ would enable the elimination of i_y from being an extra variable, therefore reducing the size

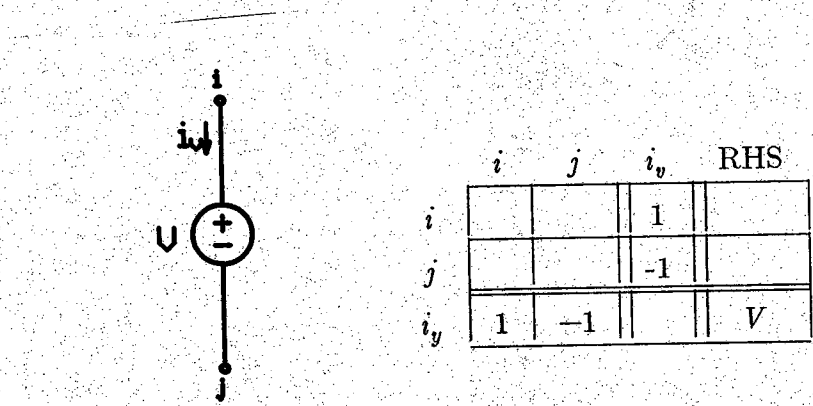


Figure 3.3 : Voltage source element stamp

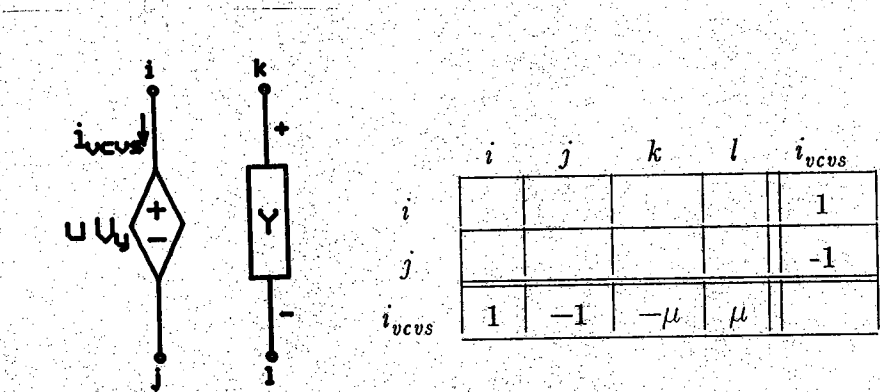


Figure 3.4 : VCVS element stamp

of the MNA matrix. So, $i_{ccvs} \in \mathbf{I}$ is the only extra current variable and the extra equation is the BR provided by the CCVS and the controlling branch, which is :

$$v_1 - v_2 = r i_y \quad (3.15)$$

Since $i_y = Y(v_3 - v_4)$, equation (3.15) becomes

$$v_1 - v_2 - r Y v_3 + r Y v_4 = 0 \quad (3.16)$$

The element stamp for a CCVS is illustrated in figure (3.5).

- 4) *Current controlled current source, CCCS* (figure 3.6). The MNA proposed in [12] would require that the controlling current i_y be included as variables. That is however unnecessary. Recognizing that $i_y = Y(v_3 - v_4)$ would enable the elimination of i_y from being an extra variable, therefore introducing no extra variables to the MNA matrix. So the CCCS's current can be expressed as

$$i_{cccs} = \beta Y(v_3 - v_4) \quad (3.17)$$

and immediately entered in the MNA matrix in that form. The element stamp for a CCCS is illustrated in figure (3.6).

Example 3.2

As an example of a MNA matrix formulation, consider the circuit of figure (3.7). The extra current variables are the branch current of the independent voltage source (branch 1) and the CCVS (branch 5). They are referred to as i_1 and i_5 respectively. Using element stamps, the MNA system of equations becomes:

$$\begin{array}{l} 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ x1 \\ x2 \end{array} \begin{bmatrix} G_2 & -G_2 & 0 & 0 & 0 & 1 & 0 \\ -G_2 & G_2 + G_3 + G_4 + \frac{1}{sL_8} & -G_4 & -\frac{1}{sL_8} & 0 & 0 & 0 \\ \beta_9 G_2 & -G_4 - \beta_9 G_2 & G_4 + sC_7 & -sC_7 & 0 & 0 & 1 \\ -\beta_9 G_2 & -\frac{1}{sL_8} + \beta_9 G_2 & -sC_7 & G_4 + sC_7 + \frac{1}{sL_8} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & G_{10} & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -r_5 G_3 & 1 & 0 & -1 & 0 & 0 \end{bmatrix}$$

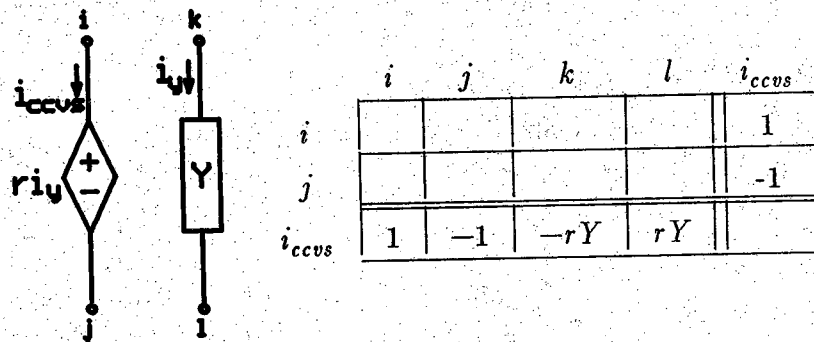


Figure 3.5 : CCVS element stamp

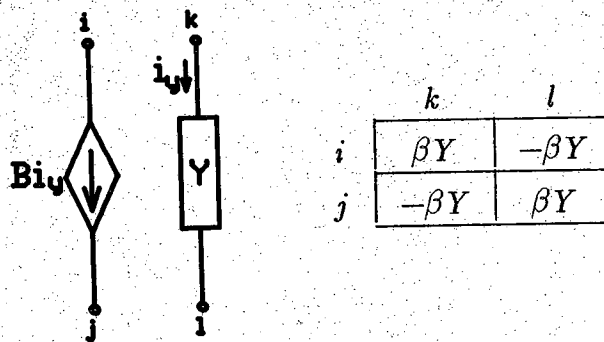


Figure 3.6 : CCCS element stamp

$$\begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \\ i_1 \\ i_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ J_{11} \\ 0 \\ V_1 \\ 0 \end{bmatrix} \quad (3.18)$$

3.4 The Reduced Modified Nodal Analysis Matrix (RMNA)

The goal here is to characterize a network in terms of a subset of the set of circuit variables. The MNA approach explained in the previous section characterizes the network in terms of all the circuit variables: node voltages and some branch currents. For the hierarchical analysis process in SCAPP the goal of the analysis is to obtain the transfer functions of some circuit variables with respect to others. These variables are the subset of circuit variables that the network is to be characterized in terms of. The options available in SCAPP enable the analysis to be performed on the whole circuit without partitioning it. In which case the analysis reduces to simply a process of the terminal block analysis of one big n -terminal block. The terminal nodes will be the nodes whose voltages are involved in the transfer functions requested. For the case where partitioning is used, user defined or automatic, each terminal block is modeled as an n -terminal block (figure 1.3), where the terminals of the block represent the tearing nodes associated with that specific block in addition to any nodes requested by the analysis that are part of this block. So the desire here is to produce a matrix that characterizes the circuit in terms of these tearing nodes and the extra current variables requested by the analysis. These variables are referred to as external variables.

This is done by additional modification to the MNA matrix to produce the Reduced Modified Analysis Matrix (RMNA) \mathbf{Y}_R that characterizes the n -terminal block.

The modification to be performed is the suppression of all node voltages and current variables that are local to the terminal block. They are referred to as internal variables. The local (non-tearing) nodes are referred to as internal nodes and the local currents (not requested by the analysis) as internal currents. The system of equations that describes the terminal block in terms of the external variables can be written as,

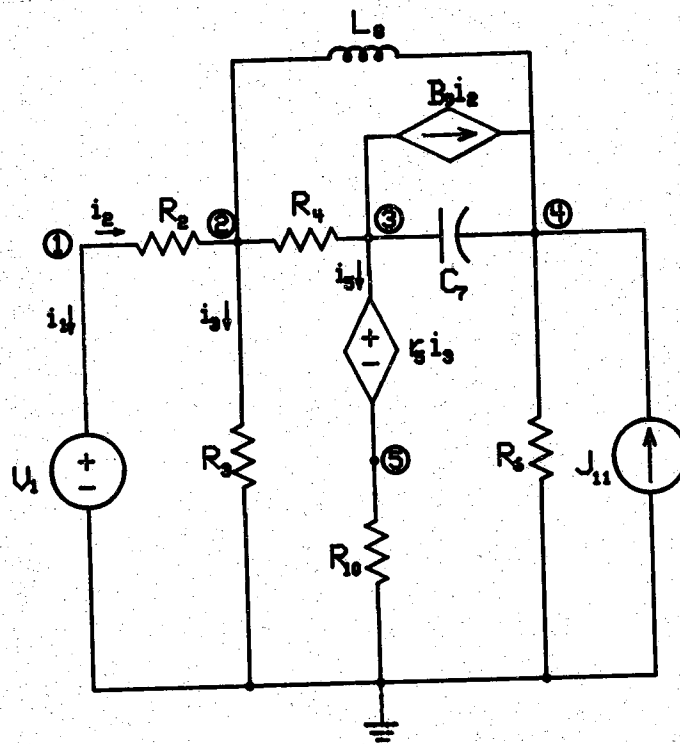


Figure 3.7 : Circuit of example 3.2

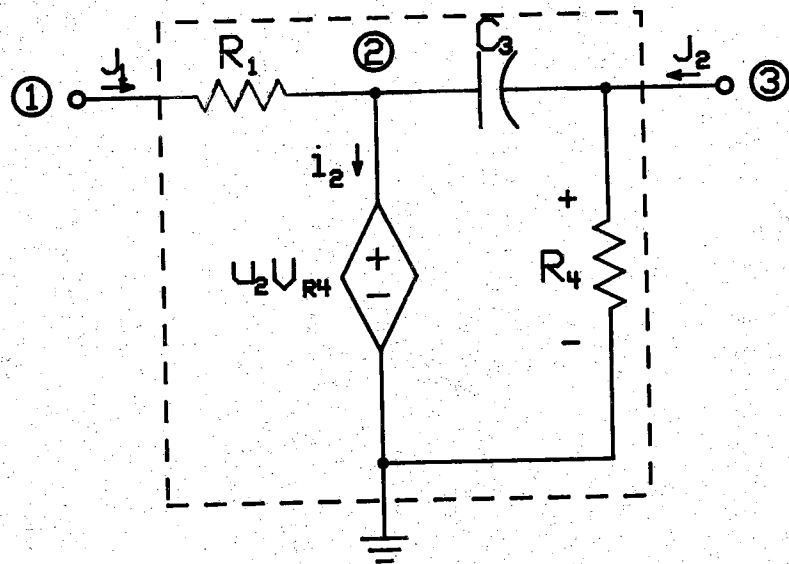


Figure 3.8 : Circuit of example 3.4

$$\begin{bmatrix} \mathbf{Y}_{R_n} & \mathbf{B}_R \\ \mathbf{C}_R & \mathbf{D}_R \end{bmatrix} \begin{bmatrix} \mathbf{V}_e \\ \mathbf{I}_e \end{bmatrix} = \begin{bmatrix} \mathbf{J}_R \\ \mathbf{E}_R \end{bmatrix} + \begin{bmatrix} \mathbf{J}_e \\ \mathbf{0} \end{bmatrix} \quad (3.18)$$

Where \mathbf{Y}_{R_n} is the reduced node admittance submatrix, \mathbf{B}_R is the reduced contributions of the external currents to KCL equations, \mathbf{C}_R and \mathbf{D}_R represent the reduced BR equations, \mathbf{V}_e is the vector of external node variables and \mathbf{I}_e is the vector of external current variables. The i th entry of the \mathbf{J}_e vector represents the currents entering the n -terminal block through the i th terminal, or in other words entering the i th node from another block or a power source. The \mathbf{J}_R and the \mathbf{E}_R vectors represent the contributions of the independent current and voltage sources, respectively, internal to the block.

The process of suppressing an internal node or an internal branch current in mathematical terms means solving for the variable in terms of the other system variables. The goal here was to find a suitable method for computer implementation. A process to solve for a variable in terms of the other system variables can be done by using a single step of the Gauss elimination method [31]. Each step of the Gauss elimination method consists of reducing a system of n linear equations in n unknowns to a system of $n-1$ linear equations in $n-1$ unknowns by using one of the equations to eliminate one of the unknowns from the remaining $n-1$ equations. The best way to illustrate the method is to consider an example.

Example 3.3

Consider the following system of three linear equations and three unknowns:

$$\begin{aligned} (1) \quad & b_{11}x_1 + b_{12}x_2 + b_{13}x_3 = l_1 \\ (2) \quad & b_{21}x_1 + b_{22}x_2 + b_{23}x_3 = l_2 \\ (3) \quad & b_{31}x_1 + b_{32}x_2 + b_{33}x_3 = l_3 \end{aligned} \quad (3.19)$$

The general matrix form for a linear system of equations can be written as:

$$\mathbf{BX} = \mathbf{L} \Leftrightarrow \quad (3.20)$$

Therefore equations (3.19) can be rewritten as follows:

$$\begin{matrix} 1 \\ 2 \\ 3 \end{matrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \\ b_{31} & b_{32} & b_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} l_1 \\ l_2 \\ l_3 \end{bmatrix} \quad (3.21)$$

Now the process of eliminating equation x_2 from the system requires the use of one of the three equations to do so. Without any loss of generality and in order to maintain a certain symmetry in the process, the second equation is chosen. The process is equivalent to eliminating the second row and second column of \mathbf{B} and the second entry in both \mathbf{X} and \mathbf{L} .

First x_2 must be eliminated from the first equation. This is done by multiplying the second equation by $\frac{b_{12}}{b_{22}}$ and subtracting it from the first equation. The next step is to eliminate x_2 from the third equation. This is done by multiplying the second equation by $\frac{b_{32}}{b_{22}}$ and subtracting it from the third equation. The result is:

$$(1) \quad (b_{11} - \frac{b_{12}}{b_{22}}b_{21})x_1 + (b_{13} - \frac{b_{12}}{b_{22}}b_{23})x_3 = l_1 - \frac{b_{12}}{b_{22}}l_2 \quad (3.22)$$

$$(2) \quad (b_{31} - \frac{b_{32}}{b_{22}}b_{21})x_1 + (b_{33} - \frac{b_{32}}{b_{22}}b_{23})x_3 = l_3 - \frac{b_{32}}{b_{22}}l_2$$

What has happened here is that the second equation was used to express x_2 in terms of x_1 and x_3 . In matrix form the process of suppressing the second row and second column of \mathbf{B} and the second entry in both \mathbf{X} and \mathbf{L} to produce \mathbf{B}_R , \mathbf{X}_R and \mathbf{L}_R , respectively, can be expressed as:

$$\mathbf{B}_R = \mathbf{B}_{cr} - \frac{1}{b_{22}}\mathbf{C}_2\mathbf{R}_2 \quad (3.23)$$

$$\mathbf{L}_R = \mathbf{L}_{cr} - \frac{1}{b_{22}}\mathbf{C}_2\mathbf{R}_2 \quad (3.24)$$

where \mathbf{C}_2 is the second column of \mathbf{B} with b_{22} removed, \mathbf{R}_2 is the second row of \mathbf{B} with b_{22} removed, \mathbf{B}_{cr} is \mathbf{B} with the second column and the second row removed and \mathbf{L}_{cr} is \mathbf{L} with the second entry removed. \mathbf{X}_R is simply \mathbf{X} with x_2 removed since it has been suppressed by equations (3.23-3.24).

The new system of linear equations can be written in general terms as:

$$\mathbf{B}_R \mathbf{X}_R = \mathbf{L}_R \Leftrightarrow \quad (3.25)$$

and for this specific example as:

$$\begin{matrix} 1 \\ 3 \end{matrix} \begin{bmatrix} b_{11} - \frac{b_{12}}{b_{22}} b_{21} & b_{31} - \frac{b_{32}}{b_{22}} b_{21} \\ b_{31} - \frac{b_{12}}{b_{22}} b_{23} & b_{33} - \frac{b_{32}}{b_{22}} b_{23} \end{bmatrix} \begin{bmatrix} x_1 \\ x_3 \end{bmatrix} = \begin{bmatrix} l_1 - \frac{b_{12}}{b_{22}} l_2 \\ l_3 - \frac{b_{32}}{b_{22}} l_2 \end{bmatrix} \quad (3.26)$$

The above procedure can be generalized for any system of linear equations in the form of equation (3.20) to be reduced to the form of equation (3.25) where the variable x_j is to be suppressed. The suppression equations become:

$$\mathbf{B}_R = \mathbf{B}_{cr} - \frac{1}{b_{jj}} \mathbf{C}_j \mathbf{R}_j \quad (3.27)$$

$$\mathbf{L}_R = \mathbf{L}_{cr} - \frac{1}{b_{jj}} \mathbf{C}_j l_j \quad (3.28)$$

where \mathbf{C}_j is the j th column of \mathbf{B} with b_{jj} removed, \mathbf{R}_j is the j th row of \mathbf{B} with b_{jj} removed, \mathbf{B}_{cr} is \mathbf{B} with the j th column and the j th row removed and \mathbf{L}_{cr} is \mathbf{L} with the j th entry removed. \mathbf{X}_R is simply \mathbf{X} with x_j removed since it has been suppressed by equations (3.27-3.28).

The effect of equation (3.27) on each element of \mathbf{B} , or in other words the effect of reducing the variable x_j on any element of \mathbf{B} , can be expressed as:

$$b_{pqR} = b_{pq} - \frac{b_{pj}}{b_{jj}} b_{jq} \quad p \neq j \text{ and } q \neq j \quad (3.29)$$

The effect on the members of \mathbf{L} can be expressed as:

$$l_{pR} = l_p - \frac{b_{pj}}{b_{jj}} l_j \quad p \neq j \quad (3.30)$$

The matrix entry b_{jj} is referred to as *the pivot*, the entry b_{pj} is referred to as the *column pivot* and the entry b_{jq} is referred to as the *row pivot*. Each *column pivot* is unique for each row in \mathbf{B} and each *row pivot* is unique for each column in \mathbf{B} and

A convention has been adopted in order to simplify the analysis operation and to make equations (3.28 and 3.30) unnecessary. The convention is that all independent sources must remain external to the n -terminal block. The nodes

they are connected to always remain as tearing nodes and therefore they are never suppressed throughout the analysis. Direct numeric or symbolic substitutions for their values is then possible at the end of the analysis as will be evident at the end of the middle block analysis (chapter 5). This convention results in the fact that the \mathbf{J}_R and \mathbf{E}_R in equation (3.18) are both $\mathbf{0}$. So, since the entries in \mathbf{J}_e in equation (3.18) are zeros except at the locations corresponding to the external nodes, the entry l_j in equations (3.28 and 3.30), where j is always an internal node, is always 0. This all means that the entire reduction operation now has no effect on the right hand side of the system of equations at all.

For the computer implementation of the method, where the matrix is operated on row by row, and in order to minimize the number of mathematical operations for the process, the first operation performed is the division of the *column pivot by the pivot*; that is, $\frac{b_{pj}}{b_{jj}}$ is performed first and stored in a new variable. This new variable is then used in equations (3.29-3.30). This saving might seem trivial for small size matrices, but when dealing with matrices of dimensions in the range of 1000x1000, the prospect of saving 999,000 (999x1000) multiplications becomes quite appealing.

The Gauss elimination step explained above is applied directly to the MNA matrix developed for the n -terminal block. The result is the RMNA matrix characterizing the n -terminal block in terms of its external variables only. All the internal variables have been suppressed by the successive application of the Gauss elimination step. A simple example will serve to illustrate and easily verify the formulation of the RMNA matrix.

Example 3.4

Consider the circuit in figure (3.8). Treating it as a 3-terminal block, nodes 1 and 3 become the external nodes. The assumption is that all current variables are to remain internal; in this case i_2 . Using element stamps the MNA matrix for this circuit is expressed as:

$$\begin{array}{c}
\begin{array}{cccc}
& v_1 & v_2 & v_3 & i_2 \\
1 & \left[\begin{array}{cccc}
G_1 & -G_1 & 0 & 0 \\
-G_1 & G_1+sC_3 & -sC_3 & 1 \\
0 & -sC_3 & sC_3+G_4 & 0 \\
0 & 1 & -\mu_2 & 0
\end{array} \right] \\
2 \\
3 \\
x1
\end{array}
\end{array} \quad (3.31)$$

To produce the RMNA matrix in terms of v_1 and v_3 only, all the other variables must be suppressed. The anticipated results is a (2×2) RMNA matrix.

Notice that an attempt to reduce the current variable i_2 first would cause a problem. The reason being *the pivot* is zero. Therefore it is deferred to the end. The reason is fully explained in the next section (section 3.6).

So the first step is to suppress the internal node 2. The *pivot* is G_1+sC_3 . Performing equation (3.27) the MNA matrix of equation (3.31) yields:

$$\begin{array}{c}
\begin{bmatrix} G_1 & 0 & 0 \\ 0 & sC_3+G_4 & 0 \\ 0 & -\mu_2 & 0 \end{bmatrix} - \frac{1}{G_1+sC_3} \begin{bmatrix} -G_1 \\ -sC_3 \\ 1 \end{bmatrix} \begin{bmatrix} -G_1 & -sC_3 & 1 \end{bmatrix} \\
= \\
\begin{array}{c}
\begin{array}{ccc}
v_1 & v_3 & i_2 \\
1 & \left[\begin{array}{ccc}
\frac{G_1 s C_3}{G_1+sC_3} & -\frac{G_1 s C_3}{G_1+sC_3} & \frac{G_1}{G_1+sC_3} \\
\frac{G_1 s C_3}{G_1+sC_3} & \frac{G_1 s C_3}{G_1+sC_3}+G_4 & \frac{s C_3}{G_1+sC_3} \\
\frac{G_1}{G_1+sC_3} & -\mu_2+\frac{s C_3}{G_1+sC_3} & -\frac{1}{G_1+sC_3}
\end{array} \right] \\
3 \\
x1
\end{array}
\end{array} \quad (3.32)$$

Notice what happened here, the suppression of node 2 which is one of the nodes of the branch whose current is a variable produced a fill in *the pivot* position for row x_1 . This now allows for the suppression of the internal current variable i_2 . Equation (3.27) is applied again to the matrix of equation (3.32) and results in:

$$\begin{aligned}
& \begin{bmatrix} \frac{G_1 s C_3}{G_1 + s C_3} & -\frac{G_1 s C_3}{G_1 + s C_3} \\ -\frac{G_1 s C_3}{G_1 + s C_3} & \frac{G_1 s C_3}{G_1 + s C_3} + G_4 \end{bmatrix} \\
& + (G_1 + s C_3) \begin{bmatrix} \frac{G_1}{G_1 + s C_3} \\ \frac{s C_3}{G_1 + s C_3} \end{bmatrix} \begin{bmatrix} \frac{G_1}{G_1 + s C_3} & -\mu_2 + \frac{s C_3}{G_1 + s C_3} \end{bmatrix} \quad (3.33)
\end{aligned}$$

After canceling some terms and some mathematical manipulation of the matrix of equation (3.33), the resulting RMNA system characterizing the 3-terminal block becomes:

$$\begin{bmatrix} 1 \\ 3 \end{bmatrix} \begin{bmatrix} G_1 & -G_1 \mu_2 \\ 0 & s C_3 + G_4 - s C_3 \mu_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_3 \end{bmatrix} = \begin{bmatrix} J_1 \\ J_2 \end{bmatrix} \quad (3.34)$$

where J_1 and J_2 are the currents entering terminals 1 and 3 from the external world.

3.5 The Sparsity of the MNA Matrix

The Gauss elimination method is appealing for the case of the MNA matrix because of the high sparsity of the matrix in the real circuits world. Making use of sparse matrix techniques enables SCAPP to only perform the reduction on the nonzero elements of each row of the MNA matrix. The number of nonzero entries in each row representing the KCL equations at the circuit nodes is strictly dependent on the number of elements connected to that node. For practical circuits each node will have at least a conductance branch incident on it. The contributions of each type of element to the entries of \mathbf{Y}_n and \mathbf{B} of the MNA matrix (equation 3.11) is illustrated herein.

- Each conductance incident on the node will contribute at least one entry at the pivot (the row entry corresponding to the node voltage variable) and another entry if its other node is not grounded, unless its current is a variable and then only one entry is contributed to the row. This is readily seen from the element stamps of figure (3.2). So b conductance branches incident on a node will contribute $b+1$ entries to its corresponding row.
- All types of voltage sources will contribute only one entry in the column corresponding to their extra current variable, figures (3.3-3.5).

— Current sources will contribute a maximum of two entries, figures (3.2,3.5).

By inspecting a collection of integrated and discrete, digital and analog practical circuits, it has been found that a node has an average of about four branches connected to it, only one of which is usually a controlled source. Therefore the worst case number of entries contributed to a KCL row in the MNA matrix is estimated at about eight nonzero entries.

The number of nonzero entries in the rows corresponding to the BR equations for the extra current variables has an upper limit of four entries. This can be readily seen from the element stamps of figures (3.2-3.6). The contribution of different elements to the entries of **C** and **D** of the MNA matrix (equation 3.11) is illustrated in the following list:

- For the case of conductance branches the BR row would have a maximum of three entries (two if one of its nodes is grounded), figure 3.2.
- For the case of an independent voltage source the BR row would have a maximum of two entries (one if one of its nodes is grounded), figure (3.3).
- For the cases of a VCVS and a C CVS the BR row would have a maximum of four entries (three if either the element or its controlling branch has a grounded node and two if both have a grounded node each), figures (3.4-3.5).

Therefore the worst case number for BR rows is four entries in each row.

Now assuming that the average number of entries per row of the MNA matrix is eight (worst case is four for the BR part so this is somewhat of an overestimate but will clearly illustrate the point nonetheless) and considering a circuit with 50 nodes, the minimum dimension of the MNA matrix is 50×50 , which is 2500 matrix entries. This is a minimum because the dimension of the matrix could be even larger because of the extra BR rows due to the extra current variables. Therefore 2500 entries is an underestimate of the size of the matrix. At an average of eight nonzero entries per row, the number of nonzero entries in the matrix will be $50 \times 8 = 400$. Which says that the MNA matrix for a 50 node circuit will, on the average, be only 16% full. A very sparse matrix. Table (3.1) illustrates the sparsity of MNA matrices as a function of the number of nodes in the matrix.

All the numbers in the last column of table (3.1) are an overestimate because of the assumptions made earlier. The MNA matrix is usually larger than the second column indicates because of the extra BR equations, and the number of entries in them has a maximum of four rather than an average of eight as in the rest of the matrix. So MNA matrices will even be sparser than

Table 3.1 : The sparsity of the MNA matrix

Number of Nodes	Number of Matrix Entries	Average Number of Nonzero Entries	% fullness
10	100	80	80%
20	400	160	40%
30	900	240	26.7%
40	1600	320	20%
50	2500	400	16%
100	10000	800	8%
500	250000	4000	1.6%
1000	1000000	8000	.8%

the table indicates but nonetheless it gives an excellent feel of how sparse the matrix becomes as the size of the circuit grows. The reason is the size of the matrix grows quadratically with the size of the circuit, as indicated in column two of table (3.1), and the number of nonzero entries in the matrix only grows linearly, as indicated by the third column.

3.6 The Pivoting Problem

The process of reducing the MNA matrix using the Gauss elimination steps is very dependent on *the pivot* element. Equation (3.27) illustrates that quite clearly. A zero entry in the pivot position would produce a division by zero which make equation (3.27) invalid and the variable in question irreducible. Mathematically speaking, if a variable x_i is to be suppressed, any of the system equations where the coefficient of x_i is not zero may be used to reduce it. In other words, the process consists of suppressing the i th column of the MNA matrix \mathbf{Y}_m using any of the rows of \mathbf{Y}_m with a nonzero entry in i th column position. That nonzero entry is *the pivot*. In this project the choice to use the i th row to suppress the variable x_i is not arbitrary. There are several reasons for the choice. They are:

- 1) In general all real world practical circuits are well behaved. A well behaved circuit guarantees that any variable in the circuit may be suppressed if only the i th row is used to suppress the i th column (see definition 3.1 and rest of section for explanation).
- 2) The process of randomly selecting a row to eliminate a certain variable could cause a change in the right hand side vector if one of the equations corresponding to an external node is chosen. This has the undesirable effect of introducing mathematical operations to the \mathbf{J}_e vector and some nonzero terms in the $\mathbf{0}$ vector of equation (3.18) because l_j in equations (3.28 and 3.30) is no longer zero. This would destroy the middle block analysis initial step of reconnecting n -terminal blocks by reconnecting their corresponding tearing nodes (chapter 5).
- 3) The much higher efficiency of the computer implementation of the symmetric (non-random) process of suppressing the i th column and the i th row.

Definition 3.1

A well behaved circuit is one for which every node has at least one conductance element connected to it.

The claim in (1) above states that a well behaved circuit will have an MNA matrix that has a nonzero diagonal or that a nonzero fill could be produced on any point on the diagonal by a proper order of variable elimination.

All the connection possibilities to a node will be considered here to support the above claim.

- 1) The node i has at least one conductance whose branch current is not a system variable connected to it. The element stamp of figure (3.2) clearly shows that a nonzero entry in the pivot position will always be produced.
- 2) The node i has only one conductance whose branch current is a system variable connected to it. The element stamp of figure (3.2) indicates that the row corresponding to that node will have a zero pivot. However, notice that the BR equation will have a nonzero pivot. Utilizing equation (3.29) shows that eliminating the BR equation first will produce a fill at the pivot of the i th equation. Node i may be reduced now. So the case where the branch current is an external variable may force one or both the voltage nodes of that branch to be carried along in the analysis as external variables.
- 3) The node i has a controlled voltage source connected to it (in addition to at least one conductance by definition 3.1). From the element stamps of figures (3.4 and 3.5), the BR equations for the current variable through the voltage source will have a zero pivot. However, utilizing equation (3.29) and case 1 or case 2 above (whichever applies) to eliminate node i will produce a fill at the pivot for the BR row. The branch current variable may now be eliminated. The case where both nodes of the controlled voltage source are external nodes forces the branch variable to be carried along as a variable throughout the entire analysis.
- 4) The node i has a controlled current source connected to it (in addition to at least one conductance by definition 3.1). From the element stamps of figures (3.2 and 3.6) this is simply a special case of cases 1 or 2 above.

CHAPTER 4

THE IDEAL OPERATIONAL AMPLIFIER IN THE MNA AND THE RMNA MATRICES

4.1 Introduction

Neither the nodal analysis, MNA, or RMNA approaches detailed in chapter 3 account for ideal operational amplifiers (opamps) in their equation formulation. The way opamps are usually handled in those analyses is by modeling each opamp by a voltage controlled voltage source with a large gain in order to attempt to characterize the infinite gain of the ideal opamp.

The ideal opamp is a 4-terminal device for which one of the terminals is always assumed to be grounded. Since opamps are used extensively in the building of a large number of analog circuits, specially analog filters, where symbolic circuit simulators have found large applications in finding transfer functions for the filters, it was deemed a necessity for SCAPP to handle ideal opamps. This required the expansion of the element set of the MNA approach and the RMNA matrix to include ideal opamps.

4.2 The Nullator, The Norator and The Nullor

Before the characterization of the ideal opamp is attempted the concepts of the nullator, the norator and the nullor are explored [4]. They are not real elements. They are tools to introduce some mathematical constraints into a circuit. They are used as an aid to the development of insight into the behavior of ideal devices like the ideal opamp.

The symbol for the nullator is illustrated in figure (4.1). A nullator is defined as follows:

Definition 4.1

A nullator is a two terminal element defined by the constraints

$$v_1 - v_2 = 0 \tag{4.1}$$

$$i = 0 \quad (4.2)$$

The symbol for the norator is illustrated in figure (4.2). A norator is defined as follows:

Definition 4.2

A norator is a two terminal element for which the voltage and current are not constrained. That is

$$v_1 - v_2 = \text{arbitrary} \quad (4.3)$$

$$i = \text{arbitrary} \quad (4.4)$$

A norator in a circuit introduces freedom from some constraints on the nodes it is connected to.

The combination of a nullator and a norator to produce a 4-terminal block as shown in figure (4.3) is referred to as a nullor. The equations characterizing this 4-terminal block are represented then by equation (4.1 through 4.4).

4.2 The Ideal Opamp Model

The two characteristics of an ideal opamp are:

- 1) An arbitrary current that the output node supplies, and
- 2) The zero voltage differential between its input nodes.

This can be modeled using a nullor (figure 4.3), with terminal 4 of the nullor grounded.

The rules for writing the KCL equations and finding the MNA matrix \mathbf{Y}_{op} for a circuit with ideal opamps is done by the following procedure:

- 1) Remove the nullor (the opamp) from the network leaving n nodes (plus the reference node).
- 2) Write the MNA matrix \mathbf{Y}_m for this network.
- 3) For the nullator between nodes p and q , delete column q of \mathbf{Y}_m and add column q to column p . If q is the reference node, simply delete column q and the q th entry in the voltage variable vector \mathbf{V} .
- 4) For the norator between nodes l and k , delete row l of \mathbf{Y}_m and add row l to row k . If k is the reference node, simply delete row l and the l th entry in the right hand side current vector \mathbf{J} .

The result is \mathbf{Y}_{op} .



Figure 4.1 : A nullator



Figure 4.2 : A norator

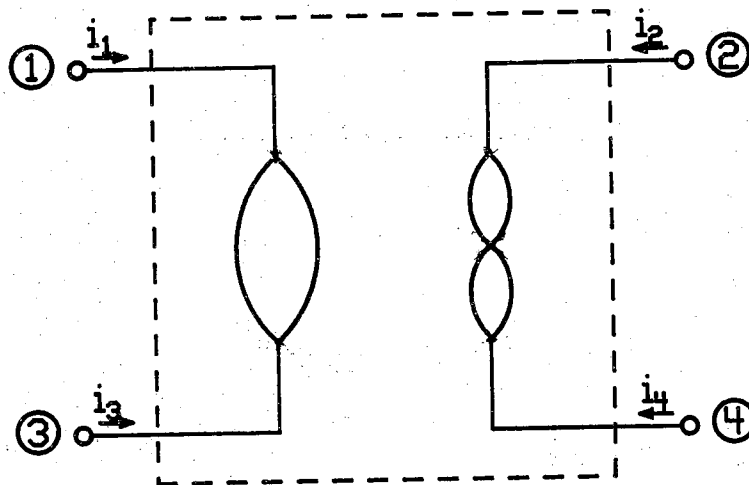


Figure 4.3 : A nullor

Example 4.1

Consider the 4-terminal block and its nullor model in figure (4.4). Following the above procedure, first the Y_m matrix is built:

$$\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{ccccc} & 1 & 2 & 3 & 4 & 5 \\ \begin{bmatrix} y_1+y_2 & -y_2 & 0 & -y_1 & 0 \\ -y_2 & y_2+y_3 & 0 & 0 & -y_3 \\ 0 & 0 & 0 & 0 & 0 \\ -y_1 & 0 & 0 & y_1 & 0 \\ 0 & -y_3 & 0 & 0 & y_3 \end{bmatrix} & \begin{bmatrix} v_1 \\ v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} & = & \begin{bmatrix} 0 \\ 0 \\ i_3 \\ i_4 \\ i_5 \end{bmatrix} \end{array} \quad (4.5)$$

By rule 3 above now column 1 is deleted and added to column 3 as follows:

$$\begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{ccccc} & 2 & 3 & 4 & 5 \\ \begin{bmatrix} -y_2 & y_1+y_2 & -y_1 & 0 \\ y_2+y_3 & -y_2 & 0 & -y_3 \\ 0 & 0 & 0 & 0 \\ 0 & -y_1 & y_1 & 0 \\ -y_3 & 0 & 0 & y_3 \end{bmatrix} & \begin{bmatrix} v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} & = & \begin{bmatrix} 0 \\ 0 \\ i_3 \\ i_4 \\ i_5 \end{bmatrix} \end{array} \quad (4.6)$$

By rule 4 deleting row 2, which is the KCL equation at the output of the opamp which is a norator terminal produces:

$$\begin{array}{c} 1 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{ccccc} & 2 & 3 & 4 & 5 \\ \begin{bmatrix} -y_2 & y_1+y_2 & -y_1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & -y_1 & y_1 & 0 \\ -y_3 & 0 & 0 & y_3 \end{bmatrix} & \begin{bmatrix} v_2 \\ v_3 \\ v_4 \\ v_5 \end{bmatrix} & = & \begin{bmatrix} 0 \\ i_3 \\ i_4 \\ i_5 \end{bmatrix} \end{array} \quad (4.7)$$

The result here is four equations in four unknowns: the node voltages. To produce the RMNA system of equations that characterize the 4-terminal block, that is, to complete the terminal block analysis, the internal node 2 must be suppressed. Since node 2 is the output terminal of an opamp, where the KCL equation has been deleted because of the arbitrary current entering that terminal, row 1 is used to reduce node 2 voltage variable, which always will be the KCL at the negative terminal of the opamp. The process of modeling the opamp using the nullor produces enough equations to solve for all the system variables. So, the output terminals of opamps are always special nodes. They are flagged in the analysis so that when they are to be suppressed,

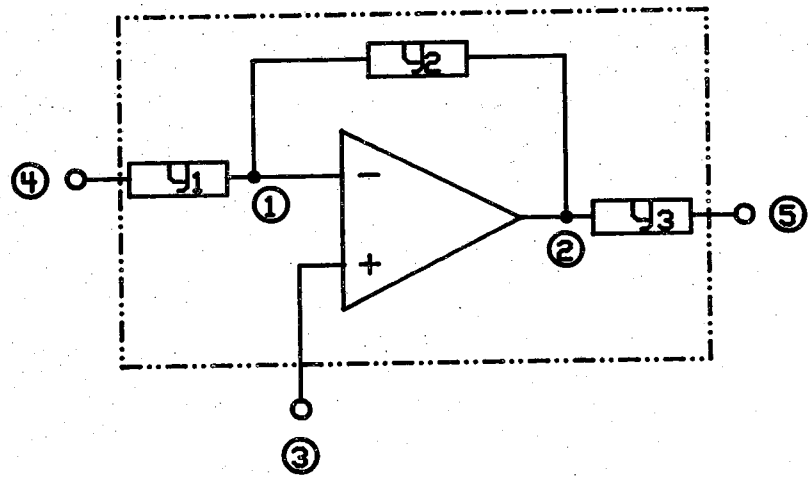


Figure 4.4 : An opamp circuit

the row corresponding to their negative input terminal is used to suppress them.

If the output terminal of the opamp is to remain a system variable throughout the analysis, then the row corresponding to the negative terminal of the opamp is carried along throughout the entire analysis.

Using equation (3.27) to reduce node 2 produces:

$$\begin{aligned}
 & \begin{matrix} & 3 & 4 & 5 \\ \begin{matrix} 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 \\ -y_1 & y_1 & 0 \\ 0 & 0 & y_3 \end{bmatrix} \end{matrix} - \frac{1}{-y_2} \begin{bmatrix} 0 \\ 0 \\ -y_3 \end{bmatrix} \begin{bmatrix} y_1+y_2 & -y_1 & 0 \end{bmatrix} \\
 & = \begin{matrix} & v_3 & v_4 & v_5 \\ \begin{matrix} 3 \\ 4 \\ 5 \end{matrix} & \begin{bmatrix} 0 & 0 & 0 \\ -y_1 & y_1 & 0 \\ -\frac{y_3}{y_2}(y_1+y_2) & \frac{y_3}{y_2}y_1 & y_3 \end{bmatrix} \end{matrix} \quad (4.8)
 \end{aligned}$$

The above RMNA matrix is the result of the block analysis of the 4-terminal block of the opamp circuit, figure 4.4. The hierarchical middle block analysis can then proceed normally if this terminal block is part of a larger circuit.

In the computer implementation in SCAPP of the opamp analysis, KCL equations at nodes corresponding to opamp output terminals are never built. That is, the rows corresponding to that output node are never included in the MNA or RMNA matrix at all. This saves on the overhead of building and then discarding that row. This is done by setting a special flag at every node that is an output of an opamp.

CHAPTER 5

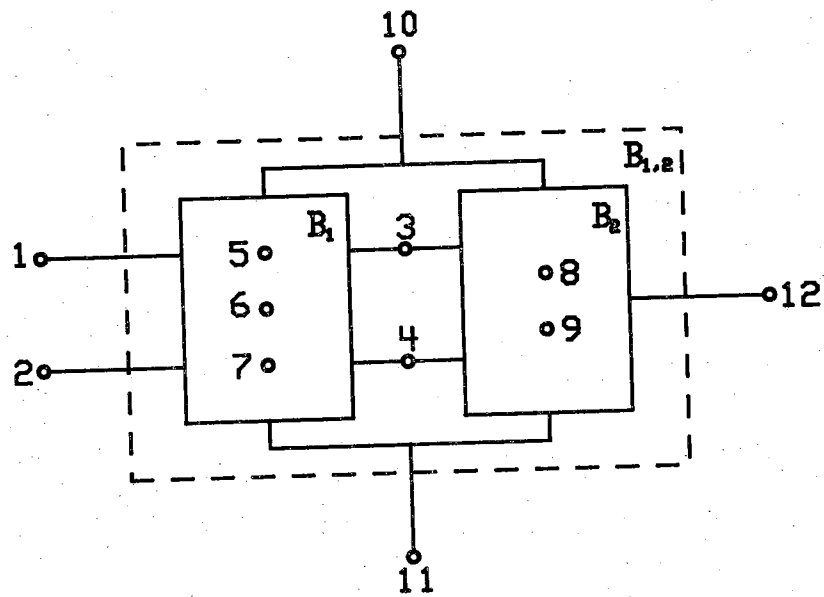
THE HIERARCHICAL MIDDLE BLOCK ANALYSIS

5.1 Introduction

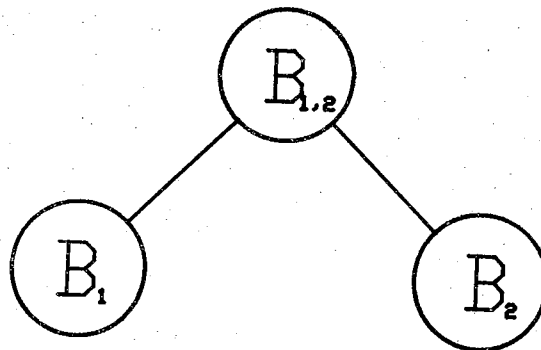
Middle block analysis is the process of combining the electrical characterization of two n -terminal blocks to produce a characterization for a new n -terminal block which physically is the interconnection of the initial two blocks. Figure (5.1) illustrates the idea of middle block analysis. Blocks \mathbf{B}_1 and \mathbf{B}_2 are both n -terminal blocks that will be characterized by two separate RMNA matrices, \mathbf{Y}_R^1 and \mathbf{Y}_R^2 , respectively. The new n -terminal block $\mathbf{B}_{1,2}$ that results from the interconnection of blocks \mathbf{B}_1 and \mathbf{B}_2 will be characterized by a new RMNA matrix $\mathbf{Y}_R^{1,2}$. The process of producing $\mathbf{Y}_R^{1,2}$ is the middle block analysis.

5.2 Middle Block Analysis Model

Consider the general interconnection of a group of n -terminal blocks shown in figure (5.2a). The n -terminal blocks represent the final subcircuits which are the result of the partitioning process, manual or automatic. Each one of these blocks is referred to as a *terminal* block. Figure (5.2b) shows the binary tree that models the binary partitioning process. The tree has p leaves, each corresponding to one of the terminal blocks. The partitioning is done utilizing a node tearing technique (chapter 8); therefore the terminal blocks share some common nodes between each other. These tearing nodes are referred to as *external* nodes. Appropriately, the nodes local to a block is referred to as *internal* nodes. Combining two of the terminal blocks as illustrated in figure (5.1) produces a new n -terminal block. This block is referred to as a *middle* block. After the characterization of the middle block has been achieved using the middle block analysis proposed herein, it behaves as a terminal block throughout the rest of the analysis. A better understanding of the concept can be achieved by studying the middle block analysis of the two terminal blocks in figure (5.1).

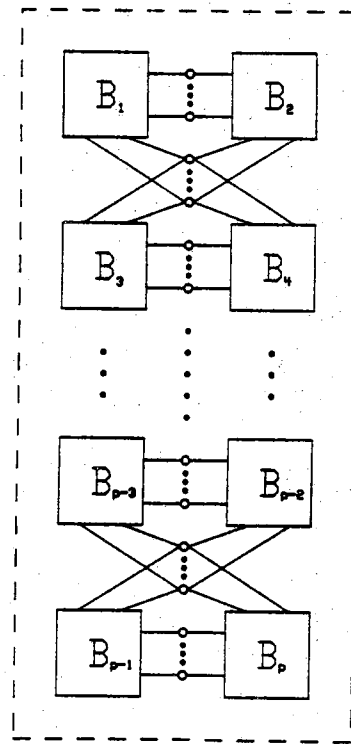


(a) A middle block

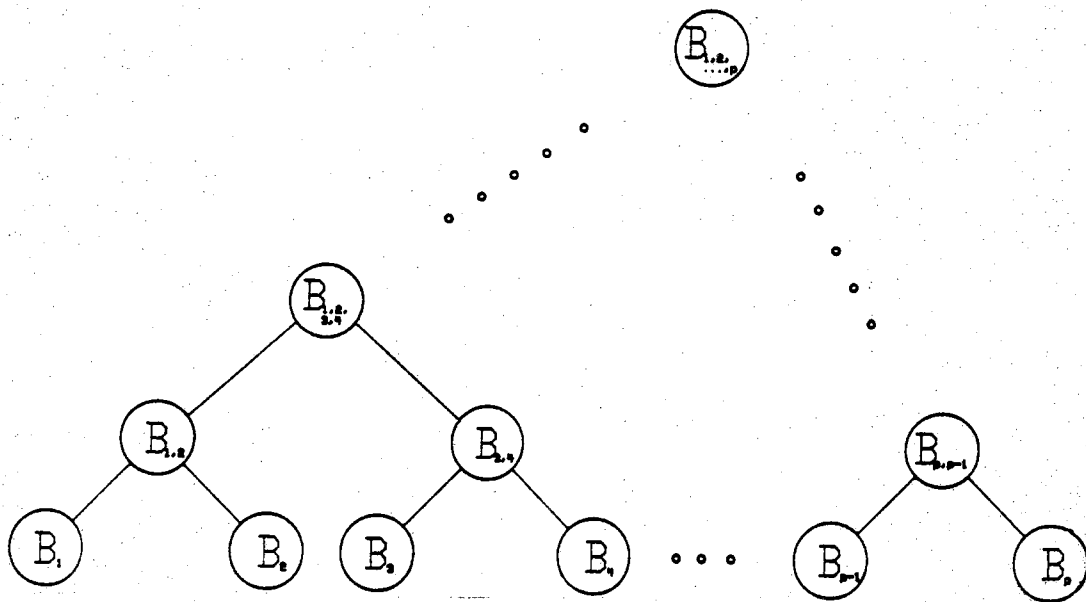


(b) Binary tree model

Figure 5.1 : Illustration of middle block analysis



(a) The network



(b) Binary tree model

Figure 5.2 : A partitioned network

In order to establish the terminology used in describing a block, *terminal* or *middle*, a typical n -terminal block is considered: block B_i . N^i is the set of all the nodes of block i . It is defined as follows:

$$N^i = N_I^i \cup N_E^i \quad (5.1)$$

where N_I^i is the set of *internal* nodes to the block, and N_E^i is the set of *external* nodes. There are n^i total nodes in N^i . n^i is defined as:

$$n^i = n_I^i + n_E^i \quad (5.2)$$

where n_I^i is the number of elements in N_I^i , and n_E^i is the number of elements in N_E^i . The complete set of tearing nodes for the entire circuit is TN . The tearing nodes between two or more blocks are always a subset of TN . This subset, $TN^{i,j,k,\dots} \subset TN$ is defined as follows:

$$TN^{i,j,k,\dots} = N_E^i \cap N_E^j \cap N_E^k \cap \dots \quad i \neq j \neq k \quad (5.3)$$

For the simple example of figure (5.1),

$$N^1 = \{1,2,3,4,5,6,7,10,11\} \quad n^1=9 \quad n_E^1=6 \quad n_I^1=3$$

$$N_E^1 = \{1,2,3,4,10,11\} \quad N_I^1 = \{5,6,7\}$$

$$N^2 = \{3,4,8,9,10,11,12\} \quad n^2=7 \quad n_E^2=5 \quad n_I^2=2$$

$$N_E^2 = \{3,4,10,11,12\} \quad N_I^2 = \{8,9\}$$

$$TN^{1,2} = \{3,4,10,11\}$$

$$TN = \{1,2,3,4,10,11,12\}$$

Notice that always $N_I^i \cap N_E^i = \emptyset$.

The process of performing a symbolic analysis on the network in figure (5.2) follows the following steps:

- 1) Partitioning the network into the terminal blocks B_i , $1 < i < p$, and obtaining the binary tree model illustrated in figure (5.2b) (chapters 7 and 8).
- 2) Performing the terminal block analysis on each block B_i (on each leaf of the binary tree) and obtaining a RMNA matrix Y_R^i characterizing each one (chapter 3).

- 3) Successively performing the middle block analysis in a hierarchical binary fashion by traversing the partitioning binary tree upwards starting with the leaves until the vertex is reached. The final result is one RMNA matrix characterizing the entire network in terms of the defined input and output variables specified by the user.

5.3 Middle Block Analysis

After steps 1 and 2 are performed above, the network in figure (5.2a) is characterized by p RMNA matrices in terms of the variables that are the members of the set \mathbf{TN} in addition to the extra current variables requested from the analysis. Consider the connection of any two terminal blocks, namely blocks \mathbf{B}_i and \mathbf{B}_j . The RMNA equations describing each are:

$$\begin{bmatrix} \mathbf{Y}_R^i \end{bmatrix} \cdot \begin{bmatrix} \mathbf{V}_e^i \\ \mathbf{I}_e^i \end{bmatrix} = \begin{bmatrix} \mathbf{J}_e^i \\ \mathbf{0} \end{bmatrix} \quad (5.4)$$

$$\begin{bmatrix} \mathbf{Y}_R^j \end{bmatrix} \cdot \begin{bmatrix} \mathbf{V}_e^j \\ \mathbf{I}_e^j \end{bmatrix} = \begin{bmatrix} \mathbf{J}_e^j \\ \mathbf{0} \end{bmatrix} \quad (5.5)$$

The two blocks share the tearing nodes in the set $\mathbf{TN}^{i,j} = \mathbf{TN}^i \cap \mathbf{TN}^j$. Also $\mathbf{N}^{i,j} = \mathbf{N}_E^i \cup \mathbf{N}_E^j$.

The following middle analysis steps are followed in order to produce the RMNA matrix corresponding to a middle block:

- 1) Combine the vector of external variables for the two blocks by letting

$$\mathbf{V}^{i,j} = \mathbf{V}_e^i \cup \mathbf{V}_e^j \quad (5.6)$$

$$\mathbf{I}^{i,j} = \mathbf{I}_e^i \cup \mathbf{I}_e^j \quad (5.7)$$

where the vectors \mathbf{V} and \mathbf{I} are the node voltages and current variables of the middle block $\mathbf{B}_{i,j}$, respectively.

- 2) Combine the RMNA matrices to produce a temporary intermediate matrix \mathbf{Y}_{R_t} such that:

$$\mathbf{Y}_{R_t} = \begin{bmatrix} \mathbf{Y}_R^i & \mathbf{0} \\ \mathbf{0} & \mathbf{Y}_R^j \end{bmatrix} \quad (5.8)$$

and also to produce the following temporary right hand side vector \mathbf{RHS}_t

$$\mathbf{RHS}_t = \begin{bmatrix} \mathbf{J}_e^i \\ \mathbf{0} \\ \mathbf{J}_e^j \\ \mathbf{0} \end{bmatrix} \quad (5.9)$$

- 3) Add the columns and rows of \mathbf{Y}_{R_t} that corresponds to the tearing node voltage variables shared between the two blocks, that is, the nodes that are members of the set $\mathbf{TN}^{i,j}$. Mathematically, adding the columns in this step reflects the fact that the voltage at a tearing node is equal in all blocks. The concept of adding the rows is not as simple. Each row corresponding to a tearing node represents the sum of the currents entering the block from that node. The corresponding entry in the **RHS** vector symbolizes the current entering that node from the rest of the network (a more detailed explanation of this concept in section 6.1). A tearing node shared between the two blocks falls under one of three categories:

- 3.1) The tearing node is only an element of \mathbf{N}_E^i and \mathbf{N}_E^j . So it is a local tearing node for block $\mathbf{B}_{i,j}$ and therefore is an internal node to $\mathbf{B}_{i,j}$ and a member of the set $\mathbf{N}_I^{i,j}$.
- 3.2) The tearing node is also an element of \mathbf{N}_E^k $k \neq i$ and $k \neq j$. That is, it is shared by more than the blocks \mathbf{B}_i and \mathbf{B}_j . This dictates that it must remain as a tearing node, ie. its row and column not suppressed. It is therefore an external node to $\mathbf{B}_{i,j}$ and a member of the set $\mathbf{N}_E^{i,j}$.
- 3.3) The tearing node voltage variable is requested by the analysis in which case it becomes a member of $\mathbf{N}_E^{i,j}$.

So when adding the rows of a node corresponding to 3.1 above the entry in \mathbf{RHS}_t will become zero. This is because the current entering block \mathbf{B}_i from the rest of the network through the tearing node (only \mathbf{B}_j in this case) is equal to the negative of the current entering block \mathbf{B}_j from the rest of the network (only \mathbf{B}_i in this case).

However, when adding the rows of a node corresponding to 3.2 and 3.3 above, the entry in $\mathbf{J}_e^{1,2}$ becomes simply a symbol indicating the current entering that node from the rest of the network (outside $\mathbf{B}_{i,j}$).

- 4) Suppress all the internal variables to block $\mathbf{B}_{i,j}$ in order to obtain the RMNA matrix characterizing the block in terms of its external variables. After step 3 is completed the sets $\mathbf{N}_I^{i,j}$ and $\mathbf{N}_E^{i,j}$ become completely

defined and the intermediate RMNA system of equations can be written as:

$$\begin{bmatrix} \mathbf{Y}_{R_t}^{i,j} \\ \mathbf{I}^{i,j} \end{bmatrix} = \mathbf{RHS}_t \quad (5.10)$$

The suppression of the tearing nodes shared between the two blocks and corresponding to case 3.1 above will reduce the $\mathbf{V}^{i,j}$ vector to the external voltage variable vector $\mathbf{V}_e^{i,j}$. To understand the entries of $\mathbf{I}^{i,j}$ branch current variable reduction a look at how a current variable reaches a middle block analysis stage is considered. A branch current variable is always an internal variable unless it has been requested by the analysis or has had a pivoting problem and was unable to be reduced earlier (see section 3.6), in which case it has been labeled as an *external* variable. Conceptually though it can never be an *external* variable. Blocks do not share branch currents, they share node voltages; because the partitioning is a node tearing technique. Therefore, an attempt to reduce a current variable that has not been requested by the analysis must be made. The current variables considered here all have a pivot problem. The pivot problem might have been resolved by the creation of a fill at that position because of the suppression of the now internal nodes that control the pivot of the current variable. If not, the variable is not suppressed and its row and column are carried along further up the middle block analysis binary tree. So $\mathbf{I}_e^{i,j} = \mathbf{I}^{i,j}$ unless the suppression of one or more "healed" variables was successful, in which case $\mathbf{I}_e^{i,j} \subset \mathbf{I}^{i,j}$.

The final result of the middle block analysis of blocks \mathbf{B}_i and \mathbf{B}_j is the RMNA matrix characterizing middle block $\mathbf{B}_{i,j}$ in terms of its external variables. The RMNA system of equations is written as:

$$\begin{bmatrix} \mathbf{Y}_{R}^{i,j} \\ \mathbf{I}_e^{i,j} \end{bmatrix} = \begin{bmatrix} \mathbf{J}_e^{i,j} \\ \mathbf{0} \end{bmatrix} \quad (5.11)$$

The current variables are grouped at the bottom of the variable vector strictly for cosmetic reasons and has no effect whatsoever on the analysis.

An example will serve to illustrate the above middle block analysis steps.

Example 5.1

The circuit in figure (5.3) is partitioned into two terminal blocks: \mathbf{B}_1 and \mathbf{B}_2 . The binary tree in figure (5.1b) can be used to model the analysis. The

terminal block analysis of each block was performed in example 3.4. The result is the following RMNA matrices:

$$\mathbf{Y}_R^1 = \begin{matrix} & v_1 & & v_3 \\ 1 & \left[\begin{array}{cc} G_1 & -G_1\mu_2 \\ 0 & sC_3+G_4-sC_3\mu_2 \end{array} \right] & & \end{matrix} \quad (5.12)$$

$$\mathbf{Y}_R^2 = \begin{matrix} & v_3 & & v_5 \\ 3 & \left[\begin{array}{cc} G_5 & -G_5\mu_6 \\ 0 & sC_7+G_8-sC_7\mu_6 \end{array} \right] & & \end{matrix} \quad (5.13)$$

Notice that $\mathbf{TN}^{i,j} = \{3\}$ and node 3 is local to the middle block $\mathbf{B}_{1,2}$. Therefore $\mathbf{N}_I^{1,2} = \{3\}$ and $\mathbf{N}_E^{1,2} = \{1,5\}$. Concatenating \mathbf{Y}_R^1 and \mathbf{Y}_R^2 and adding the rows and columns corresponding to the tearing node results in:

$$\mathbf{Y}_{R_t} = \begin{matrix} & v_1 & & v_3 & & v_5 \\ 1 & \left[\begin{array}{ccc} G_1 & -G_1\mu_2 & 0 \\ 0 & sC_3+G_4-sC_3\mu_2+G_5 & -G_5\mu_6 \\ 0 & 0 & sC_7+G_8-sC_7\mu_6 \end{array} \right] & & \end{matrix} \quad (5.14)$$

The final step of the middle block analysis is to delete the row and column corresponding to the internal variable v_3 . The result is the following RMNA matrix characterizing the middle block $\mathbf{B}_{1,2}$:

$$\mathbf{Y}_R^{1,2} = \begin{matrix} & v_1 & & v_5 \\ 1 & \left[\begin{array}{cc} G_1 & -\frac{G_1\mu_2}{sC_3+G_4-sC_3\mu_2+G_5}G_5\mu_6 \\ 0 & sC_7+G_8-sC_7\mu_6 \end{array} \right] & & \end{matrix} \quad (5.15)$$

5.4 The Hierarchical Analysis

The process of successively applying the above middle block analysis up the binary tree of figure (5.2b) at each non-leaf vertex (middle block) will produce one final RMNA matrix characterizing the network in terms of the variables requested by the analysis. The terminal block analysis is performed on the leaves of the tree and p RMNA matrices are produced. At the second level of the binary tree, the middle block analysis is performed on pairs of terminal blocks. The process will produce $\frac{p}{2}$ RMNA matrices representing the $\frac{p}{2}$ middle blocks. As illustrated earlier, physically a middle block is a

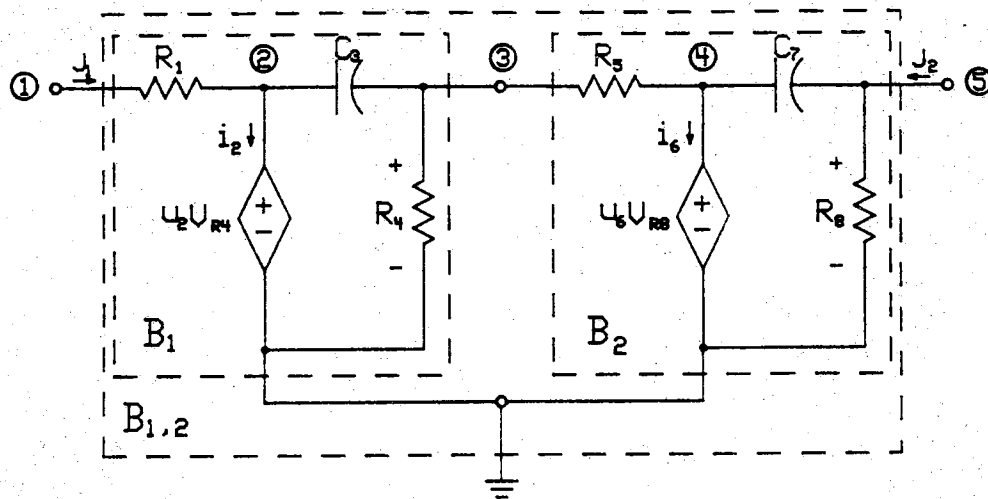


Figure 5.3 : Circuit of example 5.1

subcircuit. It is the result of interconnecting two closely coupled smaller subcircuit. So terminating the hierarchical analysis at this point will yield a characterization of the $\frac{p}{2}$ subcircuits represented by the middle block. At this point the leaves are no longer of any use to the process. They may be deleted after the middle block analysis at their parent vertex has been performed. The $\frac{p}{2}$ middle blocks and their RMNA characterizations become the leaves of the new reduced binary tree. They can be treated as terminal blocks and the middle block analysis repeated at the next level up. The process is a recursive one. It can be illustrated by the following function:

```

analyze(parent) /* Recursively Manage the Terminal and Middle Block
                  Analyses */

{
  if (no left_child) { /* If there is a left child then a
                        right one must exist */
    analyze(left_child);
    analyze(right_child);
    middle(parent); /* Perform middle block analysis */
  }
  else { /* A leaf has been reached */
    terminal(parent); /* Perform terminal block analysis */
  }
}

```

The analysis process is started by a call to the recursive function *analyze(root)* at the top of the binary tree. The function will traverse the binary tree in an *postorder* fashion [1]. The functions *terminal()* and *middle()* return RMNA matrices to *analyze()*.

The structure of the binary tree is dictated by the partitioning (chapter 8). So depending on the partitioning, the structure of the binary tree will range between a balanced binary tree and a totally unbalanced binary tree (figure 8.1). A balanced binary tree is a highly desirable structure and is essential for the case where the hierarchical analysis is performed on a multiprocessor computer.

The above algorithm is very parallelizable. Each terminal block analysis is a totally independent process. Each middle block analysis is only dependent on its children. Examining figure (5.1b) will show that block $\mathbf{B}_{1,2,3,4}$ has to wait for blocks $\mathbf{B}_{1,2}$ and $\mathbf{B}_{3,4}$ to finish before it can be processed, which in turn requires all four blocks \mathbf{B}_1 , \mathbf{B}_2 , \mathbf{B}_3 and \mathbf{B}_4 to finish. This is the hierarchical dependency of the process. The vertices on the same level are always totally independent processes. They may be analyzed concurrently.

A couple of big examples of illustrating the hierarchical middle analysis process are presented in chapter 9.

CHAPTER 6

THE GLOBAL ANALYSIS PICTURE AND THE EFFECTS OF NETWORK PARTITIONING

6.1 Introduction

The goal in this chapter is to obtain a model that globally explains the symbolic analysis process proposed in this report. This model will be used to illustrate the advantages of partitioning. For the case where the network is partitioned, the symbolic analysis process consists of first the terminal block analysis of each terminal block, followed by the hierarchical middle block analysis. The case where no partitioning is performed, the entire network is considered as one large terminal block and the terminal block analysis is performed only once to obtain the solution.

6.2 The Global Model

The model is developed after two steps have already been performed on the network. They are:

- 1) The network has been partitioned into the terminal blocks B_i , $1 < i < p$, figure (5.2).
- 2) The first step of the terminal block analysis has been performed which is finding the MNA matrix characterization for each terminal block. No suppression of any internal variables is performed.

So now the network is characterized by p MNA matrices. TN represents the global set of tearing nodes and several of the terminal block MNA matrices will have rows corresponding to the KCL equations at these nodes. The collection of equations from the different MNA matrices corresponding to a tearing node j do not constitute a mathematical redundancy. The system of equations characterizing each block B_i is:

$$\begin{bmatrix} Y^i \end{bmatrix} \cdot \begin{bmatrix} V_e^i \\ I_e^i \end{bmatrix} = \begin{bmatrix} J_e^i \\ 0 \end{bmatrix} \quad (6.1)$$

where the vector V_e^i contains the tearing nodes associated with B_i , the

vector \mathbf{I}_e^i contains a subset of the current variables requested by the analysis and are local to block \mathbf{B}_i , and the vector \mathbf{J}_e^i represents the currents entering \mathbf{B}_i .

Obviously the entries of \mathbf{J}_e^i are zero except for the entries corresponding to the tearing nodes because they are the only contact \mathbf{B}_i has with the external world. So, the equation in (eq. 6.1) corresponding to the tearing node j represents the fact that the sum of all the currents leaving node j into block \mathbf{B}_i is equal to the sum of the currents entering the node from the rest of the network. Therefore, adding all the rows corresponding to j from all the MNA matrices will yield the fact that the sum of all the currents entering node j is equal to zero.

Turning again to the network in figure (5.2a) and collecting all the rows from each \mathbf{Y}_R^i representing the tearing nodes and adding the corresponding ones together would enable the network to be modeled using the boarded block diagonal (**BBD**) matrix in figure (6.1). Notice that in order to do this, the added rows and the rows corresponding to the current variables requested by the analysis have been grouped together at the bottom of the matrix (submatrices **BB**, the bottom border). Also the variable vector has been rearranged so that the tearing node variables and \mathbf{I}_e are grouped at the bottom of the vector (submatrices **RB**, the right boarder). Each of the diagonal blocks **B** represent the KCL equations at the internal nodes of each terminal block in addition to the BR equations corresponding to the current variables local to that block and not requested by the analysis.

6.3 The Effect of the Tearing Nodes on the Analysis

The **BBD** matrix in figure (6.1) is the MNA matrix representing the network in terms of all its variables. The symbolic analysis methodology is to suppress all the variables internal to the network. This corresponds to suppressing the boarder blocks and all the diagonal blocks in addition to the rows and columns of **BP** that correspond to variables that have not been requested by the analysis.

In order to reduce the the boarder blocks **BB** and **RB**, and the diagonal block **B**, a global reduction equation that is equivalent to equation (3.27) is used to update the values of the matrix. The equation is:

$$= \begin{bmatrix} \mathbf{B} & \mathbf{RB} \\ \mathbf{BB} & \mathbf{BP} \end{bmatrix}$$

Figure 6.1 : Boardered block diagonal (**BBD**) matrix

$$\mathbf{BBD}_{-i} = \mathbf{BBD}_{-i} - \begin{bmatrix} \mathbf{0} \\ \mathbf{BB}_i \end{bmatrix} \cdot \mathbf{B}_i^{-1} \begin{bmatrix} \mathbf{0} & \mathbf{RB}_i \end{bmatrix} \quad 1 < i < p \quad (6.2)$$

where, \mathbf{BBD}_{-i} is the \mathbf{BBD} matrix with \mathbf{B}_l , \mathbf{BB}_l and \mathbf{RB}_l $1 < l \leq i$, deleted. It can be written as:

$$\mathbf{BBD}_{-i} = \begin{bmatrix} \mathbf{B}_{i+1} & & & & & \mathbf{RB}_{i+1} \\ & \mathbf{B}_{i+2} & & & & \mathbf{RB}_{i+2} \\ & & \bullet & & & \bullet \\ & & & \bullet & & \bullet \\ & & & & \bullet & \bullet \\ & & & & & \mathbf{B}_p & \mathbf{RB}_p \\ \mathbf{BB}_{i+1} & \mathbf{BB}_{i+2} & \bullet & \bullet & \bullet & \mathbf{BB}_p & \mathbf{BP} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{-i} & \mathbf{RB}_{-i} \\ \mathbf{BB}_{-i} & \mathbf{BP} \end{bmatrix} \quad (6.3)$$

Multiplying out equation (6.2) and rewriting it in terms of the \mathbf{BBD} submatrices results in:

$$\begin{bmatrix} \mathbf{B}_{-i} & \mathbf{RB}_{-i} \\ \mathbf{BB}_{-i} & \mathbf{BP} \end{bmatrix} = \begin{bmatrix} \mathbf{B}_{-i} & \mathbf{RB}_{-i} \\ \mathbf{BB}_{-i} & \mathbf{BP} \end{bmatrix} - \begin{bmatrix} \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{BB}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{RB}_i \end{bmatrix} \quad 1 < i < p \quad (6.4)$$

Because of the structure of the \mathbf{BBD} matrix, the only part of it that is updated at each i is the submatrix \mathbf{BP} . The rest of the matrix is never affected by the process, as equation (6.4) illustrates. So the process can be further compacted into the equation:

$$\mathbf{BP} = \mathbf{BP} - \mathbf{BB}_i \cdot \mathbf{B}_i^{-1} \cdot \mathbf{RB}_i \quad (6.5)$$

The rest of the analysis consists of reducing the \mathbf{RB} matrix itself until the only rows and columns left are the ones corresponding to the variables requested by the analysis. Successive application of equation (3.27) can be used for this purpose. Since the tearing nodes set is a very small subset of the total network nodes, equation (6.5) is the bottleneck of the process.

Notice from equation (6.5) that the zero structure of the \mathbf{BBD} remains intact. That is, the process does not create fills in the zero areas of the matrix; therefore not creating the need for any extra mathematical operations to handle these fills. Because of the usually small ratio of number of tearing nodes to the total number of network nodes, these zero areas are large compared to the nonzero parts of the matrix. Creating fills in these areas would lead to a great increase in the number of mathematical operations needed to reduce the matrix. This is illustrated in the next section.

Equation (6.5) shows that the number of matrix entries manipulated has a direct relationship to the size of **BP**. **BP** grows *quadratically* with the total number of circuit tearing nodes and the number of current variables requested by the analysis. However, the current variables have a negligible effect compared to the tearing nodes for two reasons:

- 1) For large circuits, the number of current variables requested by the analysis is usually small compared to the number of tearing nodes.
- 2) These branch current variables are local to a terminal block and are not shared by other blocks. Therefore, for each current variable, only one **BB_i** block and one **RB_i** block will have nonzero entries corresponding to the row and column of the variable. This simply says that out of the p successive applications of equation (6.5) only one time will the entries corresponding to the current variable in **BP** be manipulated.

Since the number of tearing nodes has a direct effect on the dimensions and therefore the size of **BP**, it is obvious from the structure of the **BBD** matrix in figure (6.1) that the number of tearing nodes also affect the sizes of the **BB_i** and **RB_i** blocks. Thus having a greater effect on the number of computations equation (6.5) has to perform.

Conclusion

Minimizing the number of tearing nodes in the partitioning process will have approximately a *quadratic* effect on increasing the efficiency of the analysis.

6.4 The Advantages of Network Partitioning

From the previous discussion, partitioning the network has a great effect on the shape of the **BBD** MNA matrix characterizing the network. That in turn effects the efficiency of the symbolic analysis process. In order to explore the advantages that network partitioning provides to the analysis over the process of analyzing the entire circuit as a terminal block, the MNA matrix for an unpartitioned network is considered. The symbol **UP** is used to indicate a MNA matrix corresponding to an unpartitioned network. Some points about **UP** are:

- 1) The dimensions of **UP** are the same as for the partitioned case because partitioning has no effect on the number of network variables.
- 2) The **UP** matrix has no specific zero-nonzero structure as for the **BBD**

matrix. The arrangement of the rows is highly dependent on the order in which the KCL and the BR equations were constructed. So it is a random function of the process of numbering the nodes of the network.

- 3) The critical order of variable reduction for the UP matrix is a random process. Selecting the wrong order rows and corresponding columns to suppress first would result in a crippling explosion in the number of mathematical reduction operations.

The reasoning behind 3 above can be easily seen by considering the UP matrix. The UP matrix and the BBD matrix are equivalent except for the ordering of rows and columns. Assume that the random order of selection required the suppression of submatrix BP first. The equation for this reduction operation would be:

$$\mathbf{B} = \mathbf{B} - \mathbf{RB} \cdot \mathbf{BP}^{-1} \cdot \mathbf{BB} \quad (6.6)$$

This displays the worst case choice of order. Notice that \mathbf{B} has a very sparse structure. The matrix $\mathbf{RB} \cdot \mathbf{BP}^{-1} \cdot \mathbf{BB}$ does not have a sparse structure at all. The result is, the updated \mathbf{B} matrix will completely lose its sparse structure. Fills will be created in all the zero patterns. Proceeding with the process by reducing the rest of the rows and columns of \mathbf{B} will require the manipulation, not only of the \mathbf{B}_i diagonal blocks, but also of all the extra fills that have been created. The explosion in the number of mathematical operations equation (5.27) has to perform is quite evident. Equation (6.5) only would have needed to operate on the diagonal blocks of \mathbf{B} . Assuming that the dimensions of each \mathbf{B}_i block is $l_i \times l_i$, the number of terms manipulated by equation (6.5) is:

$$\sum_{i=1}^p l_i^2 \quad (6.7)$$

For the case when equation (6.6) is used first, the fills created in \mathbf{B} would require the manipulation of the entire \mathbf{B} structure. The number of terms to be manipulated is given by:

$$\left[\sum_{i=1}^p l_i \right]^2 = \sum_{i=1}^p l_i^2 + 2 \sum_{i=1}^p \left[l_i \sum_{j=i+1}^p l_j \right] \quad (6.8)$$

From the previous discussion the fact that \mathbf{B} has large zero patterns indicates that $l_i \ll \sum_{i=1}^p l_i$, which makes the ratio of equation (6.8) to (6.7) a large one. Consider the case where 10 even partitions exist and $l_i = 10$ $1 < i < p$. The ratio of equation (6.8) to (6.7) is 10. 9000 more terms have to be considered

in the reduction process. This illustrates one of the advantages of partitioning the network.

One footnote on the above analysis should be mentioned. The fact of the matter is that not all of BP needs to be suppressed. A small subset of variables requested for the analysis must remain for the final result. The fills in the zero structure of the matrix however is the same as deduced above.

An example to illustrate the above is the network in example 9.2 (figure 9.6). A random selection of the reduction order resulted in 287 multiplication operations in order to reduce the matrix. The partitioned network required 88 multiplication. A savings of about 77%. A more selective choice of the order of suppression, that is, trying to suppress the densest rows and corresponding columns of the matrix last required 102 multiplications. Still the partitioned network had a savings of 35%.

So one way of dealing with the reduction of UP would be to sort its rows so that the densest ones are reduced last. This is at best an $O(n \log n)$ algorithm [1]. This still will not achieve the BBD matrix structure of the partitioned network and equation (6.5) cannot be realized because B will not have a diagonal block structure. The previous example illustrates the 35% difference still between a somewhat ordered unpartitioned matrix and the BBD matrix. A way of realizing the BBD matrix would be to explore the patterns of zeros in the matrix. This is an $O(n^2)$ process since every entry in the matrix has to be checked, where n is the number of system variables. So a partitioning process of $O(n^2)$ or less would be much more advantageous because of the other advantages of partitioning listed in this section.

Another advantage is in the time complexity of the implementation of the algorithm. Performing the reduction procedure to produce the RMNA matrices is an $O(n^2)$ process, where n is the number of system variables. Partitioning the network into p parts would mean that the process becomes an $O(p * (\frac{n}{p})^2)$ which is the same as $O(\frac{n^2}{p})$. A reduction in the time complexity by a factor of p . These order of complexities are obtained under the assumption that no sparse matrix techniques are used. The purpose here is merely to illustrate the advantages of network partitioning. Sparse matrix techniques are employed however in the computer implementation of the analysis.

A third advantage of partitioning is the multiprocessor application of the analysis algorithm. Each terminal block analysis and each middle block

analysis are completely independent of each other. Consider the binary tree model for the network in figure (5.2). The tree structure is defined by the partitioning. It is very obvious that all the terminal block analyses at the leaves of the tree can be performed concurrently with no restrictions on the scheduling of these events once the terminal blocks have been reached. For the middle block analyses at the non-leaf vertices, the scheduling becomes more critical. Each vertex has to wait until both its children have been analyzed. The task however is not a complicated one at all. Chapter 5 has more details on the concurrency of the hierarchical middle analysis process.

A fourth advantage of partitioning is the ability to build libraries containing symbolic RMNA matrices for basic building blocks. This is similar to a standard cell design system. The terminal block analysis for all the occurrences of this special block does not have to be performed. The library will contain the RMNA matrix representing the block. Only simple symbolic or numeric substitutions need to be performed for each call to the block. Even for the case where a block is not in the library but has several occurrences in the circuit, only one terminal block analysis need to be performed.

CHAPTER 7

PARTITIONING PREPROCESSING

7.1 Introduction

The complete partitioning process is composed of several algorithms that combine together to produce the partitioning algorithm. The first three algorithms are described in this chapter. They are considered preprocessing algorithms that take the raw input data from the input to SCAPP and produces an input that is suitable for the partitioning algorithm. The reason these algorithms are grouped here, is that they are highly related to the subject of the network topology, a branch of graph theory. The partitioning algorithm itself is introduced in chapter 8.

An overview of the subject of network topology and its application to this project is first presented and then the preprocessing algorithms are presented. These algorithms are :

- 1) *find_tree* : Finds the initial tree and cotree from the raw input data.
- 2) *build_B* : Builds the initial weighted fundamental loop matrix \mathbf{B} given the initial tree and cotree.
- 3) *opt_tree* : Produces an optimal or near-optimal \mathbf{B}_o from an initial \mathbf{B} . \mathbf{B}_o is a matrix that is as sparse as possible.

7.2 Network Topology

Network topology is a very involved branch of graph theory that deals with the interconnection properties of lumped networks. A few essential concepts of network topology are used here and are slightly modified to suit the purpose of this partitioning technique. This is done mainly to add generality to the partitioning algorithm so it is able to accept user-defined partitions (ie. user-defined subcircuits). These subcircuits are modeled in the algorithm as multi-terminal black boxes and referred to as n-terminal blocks. Transistors are modeled as 3 or 4-terminal blocks unless they are replaced by their lumped circuit model before starting the partitioning procedure.

The topology of a circuit to be partitioned can be represented by a graph G . The graph G is extracted from the circuit by replacing each two-terminal element by a line segment, called a branch, and replacing each n -terminal block by an n -terminal box called a pseudo-branch, figure (7.1). This pseudo-branch, as will be illustrated later in this section, is treated as a regular branch in the tree finding and weighted fundamental loop matrix construction processes (section 7.3). The difference between a branch and a pseudo-branch is that the latter may have more than two nodes associated with it. An n -terminal block will have exactly n nodes associated with its pseudo-branch.

The resulting graph G will contain all the interconnection information needed for the partitioning algorithm.

In the following discussions the graph G is assumed to be connected. That is, a path can always be found connecting any two nodes of G . This is justifiable because in order for a circuit to produce a disconnected graph, it (the circuit) would have to be composed of more than one totally independent (disconnected) subcircuits. These subcircuits are considered natural partitions and the partitioning algorithm will recognize that.

7.2.1 Trees and Cotrees

The partitioning process involves finding a tree for the constructed graph G . A tree is defined as follows:

Definition 7.1 [7]

A tree is a connected subgraph of graph G that contains all the nodes of G and no loops.

Each tree has associated with it a cotree. The cotree contains the remaining branches of G that are not included in the selected tree. The cotree branches are referred to as links. Pseudo-branches are not allowed to be links. This is justified later in this section.

One of the main properties of a tree and its associated cotree is that, each link together with a unique subset of the tree branches form a loop. That loop will be called the fundamental loop for that link [7]. For the case of a pseudo-branch, a loop is completed by entering into any terminal of that branch and exiting from any other terminal. Example 7.1 illustrated this property.

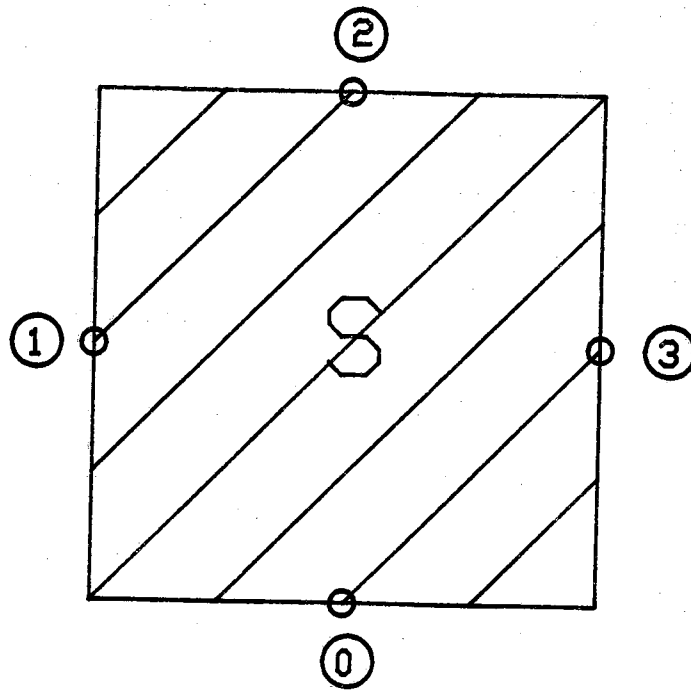


Figure 7.1 : A pseudo-branch for a 4-terminal block

As a convention, all touching user-defined subcircuits, that is, having common nodes between them, are considered as one pseudo-branch. This will minimize the entries of the weighted fundamental loop matrix (section 7.2.2).

As mentioned above pseudo-branches are only allowed to be tree branches. Because, by observation, if a pseudo-branch was allowed to be a link it could produce more than one fundamental loop corresponding to that link. This could cause, because of the heuristic nature of the partitioning algorithm, a disconnected subcircuit as a terminal block (section 3.2), and that is highly undesirable for the symbolic analysis of that terminal block. This is almost guaranteed to happen if the pseudo-branch is deeply embedded inside the circuit.

To show that pseudo-branches could always be chosen as tree branches, the following argument is given: Since a tree has no loops, and no touching pseudo-branches exist in the circuit, as stated above in this section, then starting the tree finding process by picking the pseudo-branches as tree branches will never create a loop. Therefore pseudo-branches can always be chosen as tree branches.

Now definition 7.2 may be introduced.

Definition 7.2

For a graph G with b total branches of which b_s are pseudo-branches and n nodes of which n_s are pseudo-branches terminals, there are

$$b_t = n - n_s + 2b_s - 1 \quad (7.1)$$

tree branches. This of course implies that there are

$$b_l = b - b_t \quad (7.2)$$

links and hence, $b - b_t$ fundamental loops.

Equation (7.1) is derived by observing that to span the nodes of G to form a tree, the following number of branches are needed :

- 1) b_s pseudo-branches to span n_s nodes,
- 2) $n - n_s - 1$ branches to span $n - n_s$ remaining nodes [7],

and

- 3) b_s branches to connect the subgraphes formed in 1) and 2) above.

Adding these components would produce equation (7.1).

7.2.2 Weighted Fundamental Loop Matrix

The input data to needed for the partitioning algorithm is the tree, cotree and all the fundamental loops. This information could be compacted into matrix form. This matrix is called the weighted fundamental loop matrix **B**. The idea is a generalization of the network theory's fundamental loop matrix [7,26]. Here, the **B** is defined as follows :

Definition 7.3

The weighted fundamental loop matrix **B** for a graph G and a chosen tree T with b_t tree branches and b_l links is a $b_l \times b_t$ matrix

$$B = [b_{ij}] \quad (7.3)$$

where

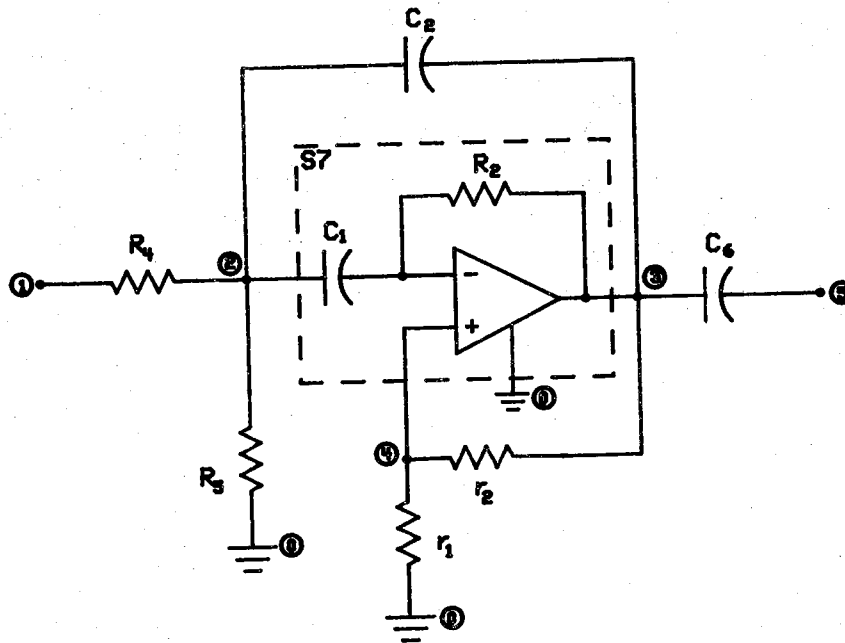
$$b_{ij} = \begin{cases} 1 & \text{if tree branch } j \text{ is in the fundamental} \\ & \text{loop of link } i \\ 2 & \text{if tree branch } j \text{ is in the fundamental} \\ & \text{loop of link } i \text{ and } j \text{ is a pseudo-branch} \\ 0 & \text{otherwise} \end{cases} \quad (7.4)$$

Example 7.1

Figure (7.2a) is a band-pass active filter circuit. The opamp amplifier circuit enclosed in a box and illustrated in figure (4.3) is considered a 4-terminal block. The graph representation of the circuit with the 4-terminal block considered a pseudo-branch and the chosen tree highlighted, is displayed in figure (7.2b). The pseudo-branch S_7 is chosen as a tree branch in addition to branches R_4 , C_6 and R_8 . The weighted fundamental loop matrix is then

$$\mathbf{B} = \begin{matrix} & R_4 & C_6 & S_7 & R_8 \\ \begin{matrix} C_2 \\ r_2 \\ r_1 \\ R_5 \end{matrix} & \begin{bmatrix} 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 2 & 1 \\ 0 & 0 & 2 & 1 \end{bmatrix} \end{matrix}$$

So a pseudo-branch has a higher weight in **B** than a regular branch. This characteristic will help isolate user-defined subcircuits as separate entities in



(a) A band-pass filter circuit

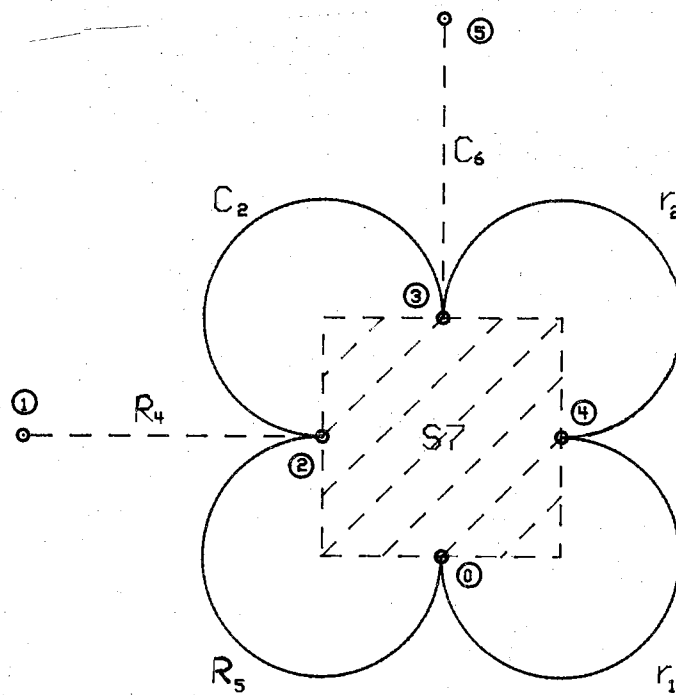
(b) The corresponding graph G

Figure 7.2 : A band-pass active filter and its corresponding graph

the partitioning algorithm. Although, again because of heuristic nature of the algorithm, this is not guaranteed for all circuits. A final partition (ie. a terminal block) might consist of the user-defined subcircuit and a few regular branches attached to it. In which case, an extra partitioning step is performed where this terminal block is split into two terminal blocks, one that consists of the user-defined subcircuit and the other of the rest of the regular branches attached to it. The hierarchical symbolic analysis process then starts at these new terminal blocks.

7.3 Finding a Tree and its Weighted Fundamental Loop Matrix

7.3.1 Finding an initial Tree T_i

A preprocessing step for the partitioning technique is finding a tree for the circuit in question. The choice of a tree is very crucial and has a great effect on the speed and efficiency of the partitioning. A good tree, is a tree that produces a sparse weighted fundamental loop matrix, or in other words, produces local loops. Intuitively a local loop refers to a small loop that, in a planar since, engulfs a minimum number of branches. Figure (7.3) illustrate the concept.

Now the notion of the best tree possible for the purpose of this report, referred to as the optimum tree, may be defined as follows:

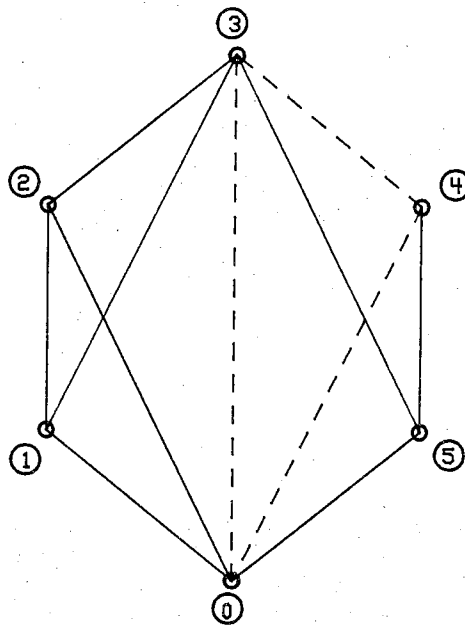
Definition 7.4

An optimum tree T_o , is a tree that produces the sparsest possible weighted fundamental loop matrix B_o .

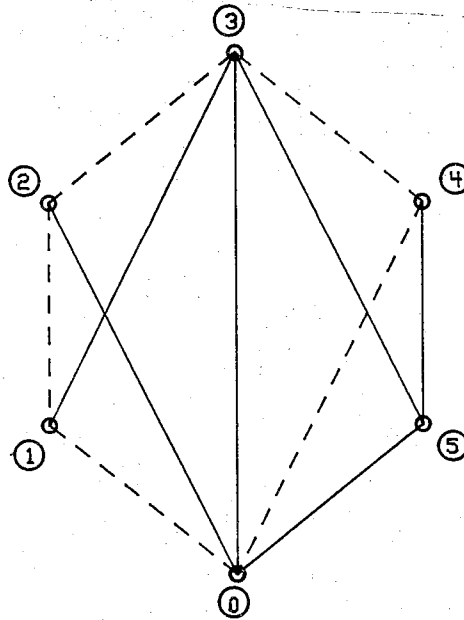
The process of finding an optimum tree involves two steps:

- 1) Finding an initial tree T_i ,
- and
- 2) performing an iterative scheme to optimize T_i until an optimum tree is found.

The choice of the initial tree T_i has a great effect on the speed of the second step. Starting with a near-optimal tree reduces the number of iterations needed in the optimization algorithm. The best T_i would be a star tree (in this case T_i would be the optimum tree). The worst choice would be a linear tree [26].



(a) A local loop



(b) A non-local loop

Figure 7.3 : A local loop and a non-local loop

Definition 7.5

A linear tree is a tree that has only two incident branches on every node in the graph G except for two node called the tip nodes. In other words, a linear tree is a path between the tip nodes (figure (7.4a)).

Definition 7.6

A star tree is a tree which all its branches have one common node (figure 7.4b).

The existence of either a linear or a star tree for any practical circuit is rare. So a reference herein to a linear tree indicates a tree that is as linear as possible. This is illustrated in figure (7.5a). Also, a reference to a star tree indicates a tree that is as star-like as possible. This is illustrated in figure (7.5b).

The advantage of starting with a star tree is illustrated in section 7.4.

The recursive algorithm to find a star-like tree T_i is illustrated in figure (7.6).

If a star-tree exists, the algorithm is guaranteed to find it, otherwise, a star-like tree is found. The resulting tree in this case will not be the optimum tree.

The linear time complexity of the algorithm [1] can be found by solving the recurrence equation that represents the execution time of the algorithm in as a function of the input number of nodes. Let $T(n)$ be the maximum number of times *loop1* is executed, $b_{n_{ave}}$ be the average number of branches incident on each node and n the number of nodes in the circuit. The recurrence equation for $T(n)$ can be written as

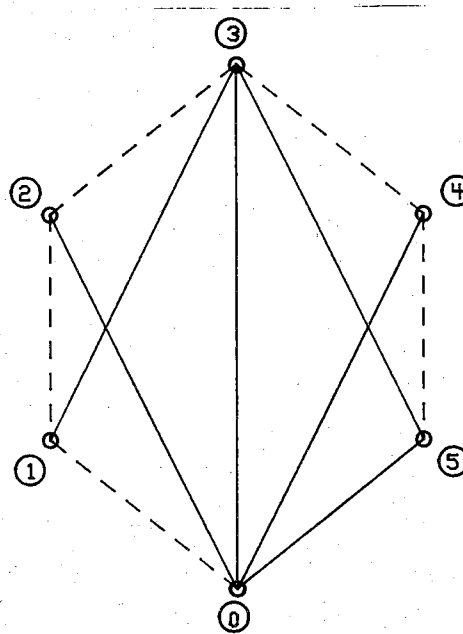
$$T(n) = \begin{cases} b_{n_{ave}} & n=2 \\ T(n-1) + b_{n_{ave}} & n>2 \end{cases} \quad (7.5)$$

which when solved produces

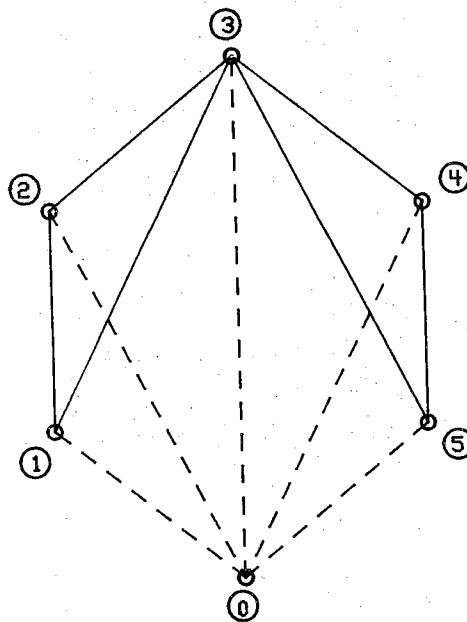
$$T(n) = nb_{n_{ave}} - b_{n_{ave}} \quad (7.6)$$

So the time complexity of the initial tree finding algorithm is on the order of $nb_{n_{ave}}$, which is written as $O(nb_{n_{ave}})$.

From the discussion in section 3.5, the average number of branches incident on each node for real world circuits is a constant (approximately 4). So the time complexity of *find_tree* becomes a linear figure. That is, $O(n)$.

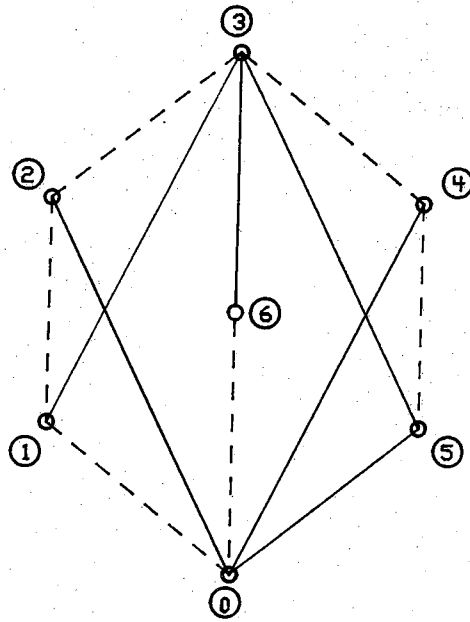


(a) A linear tree

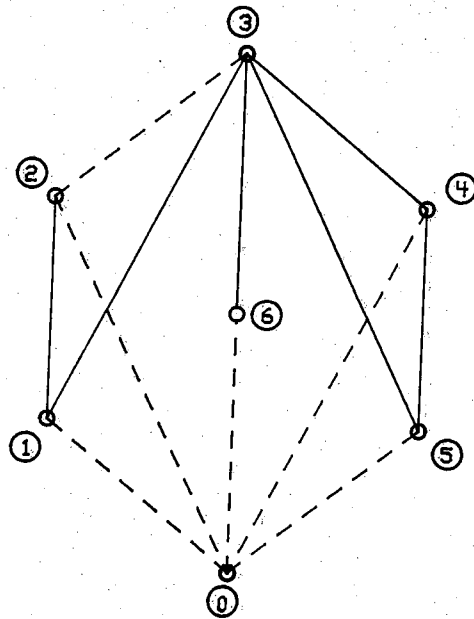


(b) A star tree

Figure 7.4 : A linear tree and a star tree



(a) A near-linear tree



(b) A star-like tree

Figure 7.5 : A near-linear tree and a star-like tree

```

procedure find_tree(ground_node)
{
    traverse(ground_node) ;
    return  $\mathbf{T}_i$  ;
}
procedure traverse(n)
{
    if (nothing in n) return ;
    /* indicates empty queue, which means all
       nodes have been traversed (ie. tree complete). */

    foreach branch b connected to n not yet in  $\mathbf{T}_i$  {

        if other_node of b never queued {
            add b to  $\mathbf{T}_i$  ;
            enqueue(other_node) ;
        }
    }
    traverse(dequeue()) ;
}

```

Figure 7.6 : The tree finding algorithm

7.3.2 Building the Weighted Fundamental Loop Matrix \mathbf{B}

It is known that every link along with a collection of tree branches constitute a fundamental loop. So there are b_l fundamental loops for a graph G with b_l links.

The procedure *build_B* constructs \mathbf{B} by processing each link and searching through the tree until a loop is found. Note that not every tree branch must be in a fundamental loop, but every link has to be. R_4 and C_6 in Example 6.1 are tree branches not in a fundamental loop, that is why the columns corresponding to them in \mathbf{B} are zero columns.

When considering real circuits, the entire circuit usually is revolved around the ground node. A very dense node. They are characterized by the existence of local loops. The complexity of an exhaustive algorithm to find all the loops of a general graph is exponential which is disastrous for the case of large circuits. However, the unique property of circuits, having local loops, reduces the complexity of building the \mathbf{B} matrix to a linear process. It has been observed (see section 8.5) that each of these local loops has an upper limit average of 6 branches in them, a very small number compared to the total number of branches in the circuit. Because of this property the search for the loops of \mathbf{B} is conducted in the local area of the link. So on the average only a constant number of nodes are visited in order to find a local loop. This produces an average complexity figure for *build_B* when dealing with electrical circuits rather than general graphs of $O(b_l)$, which is linear.

7.3.3 Finding and Optimal Tree T_o

An algorithm to find the optimum tree T_o from an initial tree T_i is adapted from a method proposed by Barbay and Zobrist [34]. Their method guarantees finding what they define as an optimum tree (definition 7.7).

Definition 7.7 [33]

The Barbay and Zobrist optimum tree is defined as the tree which results in a flowgraph with the minimum number of loops found.

This definition is more constrained than the optimum tree definition herein (definition 7.4). Note that the flowgraph mentioned above and the graph G are two entirely different entities. The latter is a representation of the connectivity of the circuit versus the former is a representation of the

electrical characteristics of the circuit [7,26], (ie Kirchoff's Voltage and Current Laws [35]). For a detailed description of Barbay and Zobrist's algorithm see [33,34].

The goal here is to reach a tree that would minimize the number of non-zero entries in the weighted fundamental loop matrix **B**. An exhaustive optimization algorithm for T_i has a high time complexity. For the partitioning algorithm and the symbolic analysis presented in this report, a near-optimal tree is sufficient to produce, for most cases, an optimum partition. For all the cases tested only a near-optimum tree was needed to produce the same solutions as the optimum tree. The reduction in the order of complexity of the algorithm is substantial the reason being that this optimization algorithm is the bottleneck of the partitioning process as a whole. The order of complexity of the algorithm will be presented later.

Let qs be the number of non-zero entries in **B**; the optimization procedure is illustrated in figure (7.7).

The worst case linear time complexity of the optimization procedure, to within a constant, can be expressed as a function of the number of links b_l (ie. number of fundamental loops, or number of rows in **B**), and the average number of tree branches per fundamental loop $b_{t_{ave}}$. Let $T(n)$ be the total linear computation time needed to perform the optimization. Assuming that it takes constant time to perform steps (1), (2) and (3), $T(n)$ can be expressed as follows:

$$T(n) = c_1(b_l - 1)b_{t_{ave}}b_l + c_2 + c_3 + c_4b_l \quad (7.7)$$

This is because it takes c_1 units to perform step (1) which is executed $b_l - 1$ times for every link other than l , this loop in turn is executed $b_{t_{ave}}$ for every branch in every fundamental loop, and this outer loop is called from `opt_tree` b_l times for every link in the tree (worst case). The extra overhead takes a constant time plus a b_l loop to update the tree. Hence equation (7.7).

The order of complexity of the optimization algorithm can be then given as $O(b_l^2 b_{t_{ave}})$. This is a worst case figure for finding the optimal tree. However to find a near optimal tree, the loop in procedure `opt_tree` is only run for 1 or 2 times, that is at worst a constant number of times. That was found to be more than sufficient for the purposes of the partitioning process and the symbolic analysis. This reduces the order of complexity to a quadratic figure rather than a cubic one. It is given as $O(b_l b_{t_{ave}})$. Also, from the discussion in section 8.5.4, for practical circuits the value of $b_{t_{ave}}$ has a constant ceiling of 4,

```

procedure opt_tree {

    foreach link l while qs changing {
        /* ie. This loop stops when no change in qs is observed */

        link_branch(l) ;

    }
}

procedure link_branch {

    foreach branch b in loop l {
        /* Each link completes a unique fundamental loop */

        temporarily switch l with b ;
        foreach link other than l {

            calculate effect of switch on qs ; (1)
            /* The switch reconfigures the loops that intersect
               b */
        }
    }

    permanently switch l with b that produce (2)
        the minimum qs ;

    update qs ; (3)
    update B ; (4)
}

```

Figure 7.7 : The tree optimization algorithm

which makes it a constant independent of the size of the circuit. This reduces the complexity of *opt_tree* here to a simple $O(b_l)$.

7.4 Implementation Notes

For the partitioning implementation in SCAPP, it is sufficient to run *opt_tree* only once. Using a near optimal tree rather than an optimal one was found to have very little effect on the partitioning algorithm and the symbolic analysis methodology.

Another note is that in SCAPP the way user-defined subcircuits have been handled is by completely deleting the pseudo-branch corresponding to them in G and considering their terminals as tearing nodes. This process could result in a disconnected graph, but the partitioning algorithm, as already mentioned, and is capable of recognizing such natural partitions. This was implemented instead of allowing the pseudo-branches in the trees because of the simplicity of programming the handling of a regular fundamental loop matrix (with only ones and zeros, no weights). Also, the flexibility of the symbolic analysis process does restrict the contents of the partitions (like requiring a controlled branch to be in the same subcircuit as its controlling branch).

CHAPTER 8

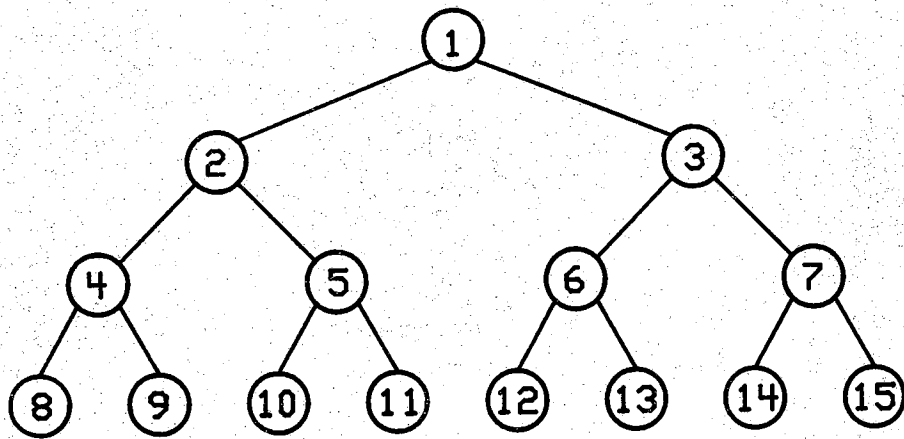
PARTITIONING ALGORITHM

8.1 Introduction

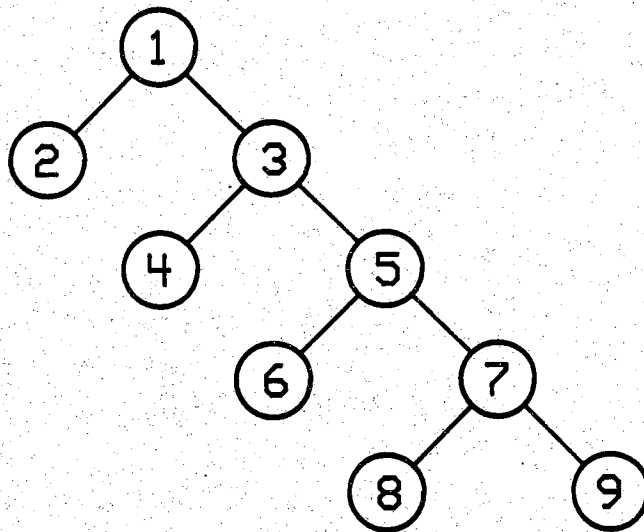
The circuit information needed for this part of the partitioning process is the tree, cotree and the fundamental loops of that circuit. This information is represented in full in the weighted fundamental loop matrix \mathbf{B} (section 7.2). The process is based on the ordered permutation of the rows of \mathbf{B} using the concept of *loop index* [29] and the binary partitioning of the matrix based on the concept of *tearing index* (section 8.2). There are two numbers associated with every i th fundamental loop, (i th row of \mathbf{B}), the *loop index* li_i and the *tearing index* ti_i . li_i is a measure of the coupling between the first i fundamental loops and the rest of the circuit, and ti_i is a measure of the number of nodes shared by specific groups of fundamental loops. The *loop indices* and the *tearing indecies* of all the fundamental loops are represented by the vectors \mathbf{LI} and \mathbf{TI} , respectively. They both are of dimensions $(b_l \times 1)$, where b_l is the number of rows in \mathbf{B} (the number links or fundamental loops in the circuit).

The partitioning algorithm has a recursive binary structure. The circuit entered is always partitioned into two parts, and then each partition is again re-entered to the algorithm to further partition each part into two new partitions for a total of four subcircuits. The process can then be repeated on any or all of these four subcircuits to further partition the circuit. The process continues until a set criterion is met by each partition, like a minimum number of branches, a minimum number of tearing nodes, a total number of partitions or a combination of the above criteria.

The process of partitioning a circuit into p subcircuits can be represented by the aid of a binary tree [1], figure (8.1). Each vertex of a binary tree can have only two children, a left child and a right child. So letting each vertex correspond to a single binary partitioning step, makes it clear that each vertex of the binary tree must either have both its children or none at all. The former case indicates that a partitioning has been performed at that vertex and the latter represents a final partition (subcircuit). A vertex with no



(a) Balanced binary tree



(b) Unbalanced binary tree

Figure 8.1 : Balanced and unbalanced binary trees

children is referred to as a leaf. So the binary tree representing the process of partitioning a circuit into p subcircuits should have p leaves. The number of times a binary partitioning has been performed is readily seen to be the number of non-leaf vertices in the binary tree, or in other words, the number of parents in the tree. This number of parent vertices v_p in a binary tree is given by

$$v_p = p - 1 \quad (8.1)$$

Equation (8.1) can easily be shown to be true by the following Mathematical Induction process [31]:

- 1) For the case where $p=2$ the circuit needs only to be partitioned once to produce 2 subcircuits. So for this case the number of parents in the binary tree is $p-1$.
- 2) Assume equation (8.1) is true for p , that is $v_p = p-1$ is true.
- 3) For $p+1$ partitions we have

$$v_{p+1} = (p+1) - 1 = p \quad (8.2)$$

In order to produce $p+1$ partitions from a binary tree that represents p partitions ($v_p = p-1$), one of the leaves has to be partitioned one more time to produce two new partitions. This turns that leaf into a parent vertex with two children. This increases the number of partitions to $p+1$ and decreases the number of parents by one to $p-1+1=p$.

Hence equation (8.1) by Mathematical Induction [31].

An advantage to this binary structure is that it makes the partitioning algorithm highly parallelizable. The reason being is that the data dependency in the partitioning binary tree is an upward hierarchical one. This means that the vertices (partitioning calls) on the same level of a binary tree are totally independent of each other. They depend only on data generated by their direct parents which is a level above them in the hierarchy and not on data generated on the same or a lower hierarchy level. Therefore, all vertices on the same or a lower level may be visited (partitioned) at the same time, ie. in parallel.

The parallelization concept for this partitioning algorithm is very simple. A special purpose multiprocessor is not needed at all. The software can easily be ported to a general purpose multiprocessor, like ELXSI [32], with the introduction of very few inter-processor communication variables.

8.2 The Loop Index and the Tearing Index

The permutation of the rows of \mathbf{B} is based on the loop index concept introduced by Rutkowski in [29]. The binary partitioning of \mathbf{B} is based on the tearing index concept proposed in this report.

Each circuit to be partitioned has associated with it the numeric vector \mathbf{l}_i , the loop index vector. It has been explained in chapter 7 that each circuit is presented to the partitioning algorithm as a weighted fundamental loop matrix \mathbf{B} (section 7.2). Each row of \mathbf{B} corresponds to a cotree link, and also represents a fundamental loop in the circuit. There are b_l of them. The columns of \mathbf{B} correspond to the b_t tree branches of the circuit. Now the loop index of row i ($1 < i < b_l$) of \mathbf{B} , l_i , may be defined as follows:

Definition 8.1 [29]

The loop index l_i of the i th row of \mathbf{B} is the number of nonzero entries in rows from the $(i+1)$ th to the b_l th and in columns of \mathbf{B} with a minimum of one nonzero entry in rows from the 1st to the i th.

Example 8.1 illustrates the computation of l_i .

The tearing index t_i for loop i is defined as follows:

Definition 8.2

The tearing index t_i of the i th row of \mathbf{B} is the number of nodes shared between the first through i th fundamental loops, and the rest of the circuit.

So the loop index basically says that if the first i rows of \mathbf{B} were joined together to form a subcircuit, the subcircuit will have t_i tearing nodes. Thus to minimize the number of tearing nodes, the binary partitioning of \mathbf{B} is performed at the row with a minimum tearing index.

Example 8.1 illustrates the computation of l_i .

Example 8.1

The graph of figure (8.5) and the \mathbf{B} matrix from example 8.5 is used to illustrate the calculation of l_i and t_i . The permuted matrix \mathbf{B}_p is:

$$\begin{array}{rcl}
 & & \begin{array}{c} 1 \quad 6 \quad 7 \quad 9 \quad 11 \\ \left[\begin{array}{ccccc} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{array} \right] \end{array} \\
 \mathbf{B} = \begin{array}{l} \text{row 1} \\ \text{row 2} \\ \text{row 3} \\ \text{row 4} \\ \text{row 5} \\ \text{row 6} \end{array} & \begin{array}{l} 2 \\ 3 \\ 4 \\ 5 \\ 8 \\ 10 \end{array} &
 \end{array}$$

To calculate li_1 , notice that the only two entries in row 1 are in columns 1 and 9. The number of entries in the $i+1$ st through b_i row, rows 2 through 6 in this case, in columns 1 and 9 is two. Therefore $li_i=2$. For li_2 , the entries in columns 1, 9 and 11 and rows 3 through 6 must be counted. The result is $li_2=4$. The rest of the loop indices are calculated in the same way (see example 8.5 for the complete list of the loop indices).

Now in order to calculate ti_1 a look figure (8.5) shows that loop 1 will have three nodes to share with the rest of the graph, nodes 1,2 and 3. ti_i is also three. It is true that node 4 has been added to the subcircuit by adding row 1 to row 2 but notice that node 3 is no longer a tearing node. It is only part of the first two loops. So $ti_i=3$. The rest of the tearing indices are tabulated in example 8.5.

8.3 The Partitioning Criteria

The partitioning criteria is obtained from the requirements of the hierarchical symbolic network analysis. The goal is to produce partitions that are heuristically suitable for that analysis. There are two main requirements involved here. The first requirement is that each partition be small enough so that the terminal block analysis of each partition, modeled by reducing the diagonal blocks of **BBD** in figure (6.1), is optimized. The second is that the number of tearing nodes at each binary partitioning point be small enough so that the hierarchical middle block analysis process, modeled by reducing the **BP** of **BBD**, is also optimized (chapter 6). This produces the set of criteria that are used in the complete partitioning process.

The first criterion that the algorithm uses to control the partitioning process is represented by the variable b_m , which is the maximum number of branches that a partition may have. So b_m defines the stopping point for the complete partitioning process. However, this is not enough, since the terminal block analysis is also dependent on the number of tearing nodes associated

with each terminal block (final partition), and the middle block analysis is completely dependent on the number of tearing nodes associated with each binary partitioning step (at every parent vertex of the partitioning binary tree). The number of tearing nodes, represented by the variable n_{tear} , is used as the second criterion to perform each binary partitioning step. The goal, of course, is to minimize n_{tear} in order to produce lightly coupled partitions with a minimum number of tearing nodes between them. This will result in a minimal size **BP** block of the **BBD** matrix (see chapter 6).

8.4 The Global Algorithm

The steps of the recursive routine *new_part* is illustrated below.

- 1) Find a star-like tree using algorithm *find_tree* (section 7.3.1).
- 2) Build the weighted fundamental loop matrix **B** using routine *build_B* (section 7.3.2).
- 3) Call the binary recursive algorithm *new_part* to partition the circuit into p parts or until the each subcircuit meets the maximum size and tearing node criteria specified by the user or the network analysis requirements.

new_part(**B**) {

```

/* Find the near-optimal tree for this subcircuit (section 7.3.1). */
  opt_tree(B) ;
/* Partition this subcircuit into a left and a right subcircuits (section 8.6).
   This function checks the the branch size criterion. If the circuit is
   already small enough a partitioning will not occur and a flag is set,
   otherwise the function returns two weighted fundamental loop matrices
   Bleft and Bright */
  binary_part(B) ;
/* If the circuit has not been partitioned then quit this call of the routine
   */
  if (no partitions) return ;
/* Here a partitioning has happened so check if the number of parts
   criteria has been met or if further partitioning is needed */
  parts = parts - 1 ; /* Decrement the number of parts requested */
  if (parts = 0) return ; /* Done with the partitioning */
/* Push the left and the right subcircuits on to a queue */
  enqueue(Bleft) ;
  enqueue(Bright) ;
/* Partition the next subcircuit in the queue by calling this routine again
```

```

*/
if (queue not empty) new_part(dequeue(Bleft)) ;
if (queue not empty) new_part(dequeue(Bright)) ;
} /* End of new_part */

```

The linear average asymptotic time complexity [1] of *new_part* is $O(p \log p)$, where p is the number of parts the circuit is divided into. This expression was found by observing in section 7.3 that *opt_tree* on the average is $O(b_l)$ and it will be shown in section 8.6 that *binary_part* is on the average $O(b_l^2)$. So both steps 3.1 and 3.2 in the above algorithm are a function of the size of the subcircuit being partitioned (since, in general, $b_l = b - n - 1$, equation (8.6)) and are not a function of p . Therefore, the recursive equation from which the asymptotic time complexity of *new_part* was computed is

$$T(p) = f(b_l) + 2T\left(\frac{p}{2}\right) \quad (8.3)$$

since *new_part* calls itself twice in step 3.6. The solution to the above equation yields the asymptotic time complexity of the partitioning algorithm as a function of the number of partitions p , which is

$$O(p \log p) \quad (8.4)$$

8.5 Execution Time Minimization

As explained in the previous section, the routine *new_part* partitions each input circuit into two subcircuits, a left subcircuit and a right subcircuit. It then proceeds to call itself twice. The process eventually will invoke *new_part* on the left and the right subcircuits to further partition the circuit into smaller parts. So steps 3.1 and 3.2 will be performed on these subcircuits. The size of each partition has a great effect on the execution time of the algorithm because of the time needed to perform the actual partitioning which is $f(b_l)$ in equation (8.3) above. This describes the execution time of *opt_tree* and *binary_part* in steps 3.1 and 3.2, respectively. There is a nonlinear dependency of the time complexity for these steps on the size (branch and node numbers) of the subcircuit. It is therefore essential to have a feel for the best binary partition size that would yield a minimum asymptotic execution time for the algorithm.

From sections 7.3, 8.4 and 8.6, it can be seen that the asymptotic time complexity of $f(b_l)$ is given by

$$O(b_l^2) \quad (8.5)$$

where b_l is the number of links in the circuit and is given by equations (7.1 and 7.2). In order to simplify the argument here, but without any loss of generality, an assumption is made that no pseudo-branches exist in the circuit. In which case the number of links is given by

$$b_l = b - n - 1 \quad (8.6)$$

where b and n are the total number of branches and nodes in the circuit, respectively. So the asymptotic complexity of the algorithm is a function of both b and n .

Now to discuss partitioning the circuit into two parts, a generic outlook on the problem is taken. The partitioning problem can be generalized as a problem of size m that takes $m \log m$ time units to divide into two similar but smaller problems, a left problem and a right problem. Each left and right problem having size

$$\frac{m}{c} \quad \text{and} \quad \frac{(c-1)m}{c} \quad (8.7)$$

respectively, where c is a positive real integer. So the time for the further partitioning of the two problems into four even smaller problems is given by substituting the expressions in equation (7.7) into equation (8.5) which yields,

$$\begin{aligned} T(m) &= \left(\frac{m}{c}\right)^2 + \left(m - \frac{m}{c}\right)^2 \\ &= m^2 \left(1 + \frac{2}{c^2} + \frac{2}{c}\right) \end{aligned} \quad (8.8)$$

So the goal here is to find the integer c for which $T(m)$ is a minimum. Differentiating equation (8.8) with respect to c and setting the result to zero, produces

$$\begin{aligned} \frac{\partial T(m)}{\partial c} &= m^2 \left(-\frac{4}{c^3} + \frac{2}{c^2}\right) = 0 \\ \Rightarrow \quad (-4 + 2c) &= 0 \end{aligned} \quad (8.9)$$

The solution to equation (8.9) is $c=2$, which implies dividing the problem into two equal halves.

Going back to the partitioning problem, this seems to imply that the best partitioning is one that separates the fundamental loops of the circuit into two

equal parts of size $\frac{b_l}{2}$. This is not the case though. As illustrated in section 8.4, the right partition does not preserve its tree and cotree structure and the process guarantees that the number of fundamental loops will be less than the original $\frac{b_l}{c}$ loops that the partitioning point defined in \mathbf{B}_2 . Otherwise, that would indicate that the circuit had two sets of loops with no branch intersections between them, ie. two totally independent circuits. Also, simply dividing b_l in half ignores the fact that the number of nodes are not directly divided into two halves. There is the set of tearing nodes that are part of both the left and the right partitions. That is, although for the branches

$$b = b_{left} + b_{right} \quad (8.10)$$

for the nodes the split must account for the tearing nodes n_{tear}

$$n = n_{left} + n_{right} - 2n_{tear} \quad (8.11)$$

Since the time complexity of the algorithm is $O(b_l^2)$, it can be deduced from the previous argument that the best partition would be one that produces a left and a right partitions that have an equal number of fundamental loops, ie. cotree links. This requirement can be expressed as follows:

$$b_{l_{left}} = b_{l_{right}} = b_{part} \quad (8.12)$$

$$\Rightarrow b_{left} - n_{left} - 1 = b_{right} - n_{right} - 1 \quad (8.13)$$

From the expression for the number of links in a circuit, $b_l = b - n - 1$, and substituting equations (7.10) and (7.11) in that expression, the expression for b_l becomes

$$b_l = (b_{left} + b_{right}) - (n_{left} + n_{right} - 2n_{tear}) - 1 \quad (8.14)$$

which, after rearranging the terms, may be rewritten as

$$b_l = (b_{left} - n_{left} - 1) + (b_{right} - n_{right} - 1) + (2n_{tear} + 1) \quad (8.15)$$

which simply is

$$b_l = b_{l_{left}} + b_{l_{right}} + (2n_{tear} + 1) \quad (8.16)$$

Now making use of equation (8.12) produces

$$b_l = 2b_{part} + (2n_{tear} + 1) \quad (8.17)$$

Solving for b_{part} yields

$$b_{part} = \frac{b_l - 2n_{tear} - 1}{2} \quad (8.18)$$

Equation (8.18) above defines the size of each partition given the number of fundamental loops b_l in the parent circuit and the number of tearing nodes n_{tear} at the partitioning point. Therefore, the best partitioning point to heuristically optimize the execution time of the complete partitioning process, is one that has a local minimum for equation (8.18).

There are two execution cases to be considered when deciding on which criterion is more important in the partitioning process, minimizing the number of tearing nodes or balancing the sizes of the partitions. The conclusion is that always a combination of both should be considered. The number of tearing nodes is the first factor to be considered in order to minimize the middle block analysis, but the binary partitioning should not be allowed to be grossly uneven because that could extremely slow down the partitioning process. The two cases are a function of the hardware available for the running of the entire process. They are the sequential execution case and parallel execution case.

- 1) For parallel execution of the algorithm, the partitioning goal is to produce a binary tree structure that is as balanced as possible. Figure (8.1a) shows a balanced binary tree and figure (8.1b) shows an unbalanced binary tree. Producing a balanced binary tree would result in high utilization of the available processors with minimal idle processors at any certain time. The best situation is if each partitioning process would always produce equal left and right partitions, that is, an equal number of fundamental loops in each partition. This may be accomplished by emphasizing equation (8.18) as the main partitioning criterion, rather than the number of tearing nodes n_{tear} .

The balanced binary tree not only is helpful in the full utilization of the multiprocessor during partitioning but also during the hierarchical combination process of the middle block analysis described in chapter 5.

- 2) For sequential execution, the requirement on the partitioning to produce a balanced tree is not essential as in the parallel execution case. The number of tearing nodes is a more important criteria to use for the binary partitioning step in order to optimize the middle block analysis process. However, because of the nonlinear $O(b_l^2)$ time complexity of *binary-part*,

grossly uneven partitions can result in creating a bottleneck for the partitioning algorithm; thus nullifying the future advantage gained by the minimization of the tearing nodes.

8.6 The Binary Partitioning Algorithm

8.6.1 General Discussion

The algorithm will partition the input circuit into two parts referred to as the left subcircuit and the right subcircuit. The basic idea of the partitioning is to group the most tightly coupled loops together in the same partition. The rows of the fundamental loop matrix \mathbf{B} are permuted such that the first b_l rows have a local minimum number of intersections with the remaining $b_r = b_l - b_l$ rows. This also means that the first b_l fundamental loops have a local minimum number of shared branches with the remaining b_r fundamental loops. The new permuted fundamental loop matrix is

$$\mathbf{B}_p = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix} \quad (8.19)$$

where \mathbf{B}_1 and \mathbf{B}_2 are submatrices of dimensions $b_l \times b_t$ and $b_r \times b_t$, respectively. Hence, the left subcircuit will consist of the first b_l fundamental loops of the circuit, and the right subcircuit will consist of the remaining b_r fundamental loops excluding the tree branches that already have been selected in the left partition, or in other words, excluding the loop intersections with the first b_l fundamental loops.

Example 8.2

The \mathbf{B}_p matrix for a circuit represented by the graph in figure (8.2) is shown in equation (8.20). The partitioning illustrated in figures (8.2 and 8.3) would define \mathbf{B}_1 and \mathbf{B}_2 as illustrated also in equation (8.20) below.

$$\mathbf{B}_p = \begin{bmatrix} \mathbf{B}_1 \\ \mathbf{B}_2 \end{bmatrix} = \quad (8.20)$$

	2	6	12	9	11	13	14	17	18
8	0	0	0	1	1	0	0	0	0
15	0	0	0	0	1	0	1	0	0
16	0	0	0	0	0	1	1	0	0
19	0	0	0	0	0	1	1	0	1
20	0	0	0	0	0	0	0	1	1
1	1	1	0	0	0	0	0	0	0
3	1	0	1	0	0	0	0	0	0
4	1	0	1	0	0	0	0	1	0
10	0	0	1	0	0	1	0	0	0
7	0	1	1	0	0	1	0	0	0
5	0	1	1	1	1	1	0	0	0

Since the columns of \mathbf{B}_p represent the tree branches of the circuit, the above partitioning of \mathbf{B}_p does not yield fundamental loop submatrices. That is, \mathbf{B}_1 and \mathbf{B}_2 are not fundamental loop matrices for the left and right subcircuits, respectively. Their columns include the tree branches of the entire circuit. The first step to producing the fundamental loop matrices \mathbf{B}_l and \mathbf{B}_r for the left and right subcircuits, respectively, is to simply remove the zero columns from \mathbf{B}_1 and \mathbf{B}_2 , thus removing the tree branches that do not belong to that particular partition. Caution must be taken here though, because zero columns are allowed in \mathbf{B} . Section 8.6 explains the handling of such a special case. The consideration now is that in order for the submatrices to be fundamental loop matrices, all of their columns must represent the tree branches, their rows must represent the links, and their row entries must represent the fundamental loops of their respective subcircuits.

8.6.2 Left Weighted Fundamental Loop Matrix \mathbf{B}_l

Looking at the left subcircuit and \mathbf{B}_1 , \mathbf{B}_l can be obtained by simply removing zero columns from \mathbf{B}_1 , therefore eliminating the branches that do not belong to the left partition. The resulting matrix \mathbf{B}_l is a fundamental loop matrix. This is the case here because of the nature of the permutation process of the rows of \mathbf{B} and the process of choosing the first b_l rows for a left partition. They both dictate that the i th row must intersect the $(i+1)$ st row, where $1 < i < b_l - 1$. So the tree branches that are in the left subcircuit are a connected subgraph of the original tree. Also the nodes of the left partition are determined by the subset of tree branches in it, because the b_l links form

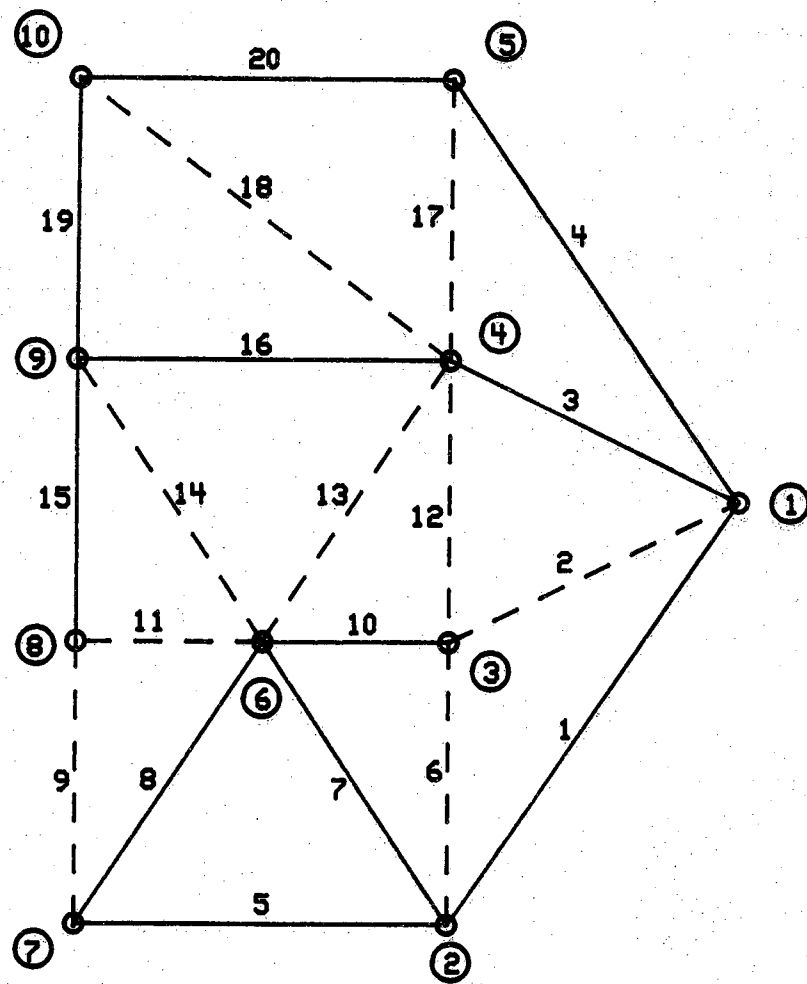


Figure 8.2 : Partitioned graph of example 8.2

complete loops with the tree branches of the left partition by the above choice of the left partition. These two facts indicate that the subset of the original tree branches included in the left partition form an acyclic connected subgraph of the left subcircuit that spans all the nodes in that subcircuit. Hence \mathbf{B}_l is a fundamental loop matrix for the left subcircuit.

Example 8.3

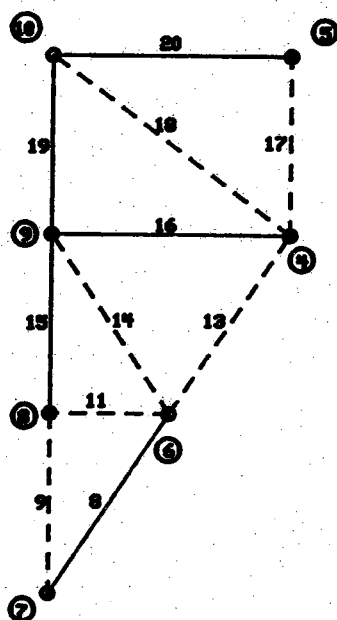
Again considering the graph in figure (8.2), the left partition is shown in figure (8.3a). Notice by inspection of that graph, the subgraph of the tree is also a spanning tree for the left partition. The weighted fundamental loop matrix for the left partition \mathbf{B}_l is obtained from \mathbf{B}_1 (equation 8.20) by simply deleting the zero columns from \mathbf{B}_1 which are the columns corresponding to tree branches 2, 6 and 12. This would produce

$$\mathbf{B}_l = \begin{matrix} & 9 & 11 & 13 & 14 & 17 & 18 \\ \begin{matrix} 8 \\ 15 \\ 16 \\ 19 \\ 20 \end{matrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 \end{bmatrix} \end{matrix} \quad (8.21)$$

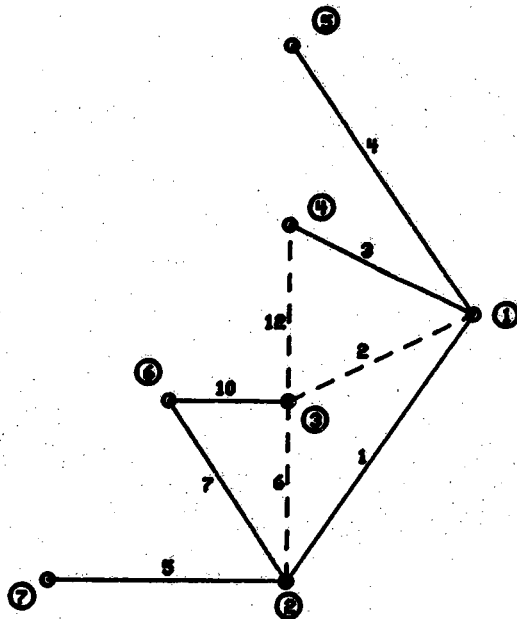
8.6.3 Right Weighted Fundamental Loop Matrix \mathbf{B}_r

Deleting the zero columns from \mathbf{B}_2 is necessary but not sufficient to construct the fundamental loop matrix \mathbf{B}_r for the right subcircuit. The columns corresponding to those tree branches that intersect the loops of the left partition must also be deleted, which opens one or more of the b_l fundamental loops originally defined in \mathbf{B}_2 . This destroys the fundamental loop property of this submatrix. So \mathbf{B}_r still needs to be constructed, that is, a new tree, cotree and fundamental loops for the right subcircuit must be constructed.

One way to find \mathbf{B}_r is to start from scratch, that is use the tree finding and optimization algorithms described in section 7.3. However, \mathbf{B}_2 has some partial information that may be used to construct \mathbf{B}_r . An algorithm has been devised that manipulates the broken loops of \mathbf{B}_2 in order to produce the fundamental loop matrix for the right subcircuit \mathbf{B}_r . The process is referred to as updating the right tree. This algorithm exhibits a substantial savings in execution time over the process of starting the search for a tree from scratch.



(a) Left partition



(b) Right partition

Figure 8.3 : Left and right partitions of example 8.3

The basic idea of the algorithm is to turn every link l_i for which the corresponding loop (referred to as loop L_i) intersects the left partition into a tree branch, and then update all the other loops that are affected by this transformation.

The algorithm, *right_update*, is illustrated in the following steps:

- 1) Form a list, d , that contains all the columns (tree branches) of B_2 that correspond to the branches already chosen to be in the left partition. The cost of constructing d is not included in the calculation of the complexity of this algorithm. The reason being that d is constructed earlier during the process of partitioning the circuit into a left and right subcircuits at a constant cost (not a function of the data set size).
- 2) For each fundamental loop L_i in B_2 , find the list $local_d = L_i \cap d$. $local_d$ will contains all branches in loop L_i that belong to the left partition.
- 3) If $local_d$ is empty, that is loop L_i does not intersect the left partition, then L_i is a fundamental loop in the right partition and the algorithm proceeds to process L_{i+1} (go to step 2 above).
- 4) If $local_d$ is not empty, that is loop L_i does intersect the left partition, then for each remaining loop L_j in B_2 ($j > i$),

- 4.1) If loop L_j and loop L_i have any common branches that are in $local_d$ (branches that belong to the left partition), or in other words if $L_j \cap L_i \cap local_d$ is not empty, then, update L_j as follows,

$$L_j = L_j \cup L_i - L_j \cap L_i + l_i \quad (8.22)$$

That is create a new L_j by deleting all the branches that the old L_j and L_i have in common from L_j and add to L_j the remainder of branches from L_i plus l_i which will become a tree branch in step 5 below. Example 8.4 illustrates this step.

- 4.2) If $L_j \cap L_i \cap local_d$ is empty, then turning link l_i into a tree branch will have no effect on loop L_j , therefore proceed to process L_{j+1} (go back to step 4.1).

- 5) Add link l_i to the tree branch list and delete it from the link list. In terms of B_2 this constitutes deleting row i and adding an extra column.

- 6) Delete all the columns corresponding to the branches in d . These branches belong to the left partition.

It can be seen from step 2 above, that the algorithm terminates when $i = b_l$. The result is a new fundamental loop matrix \mathbf{B}_r which defines a tree, cotree and the associated fundamental loops for the right partition. However, this new tree might not be an optimal one or near-optimal enough for the partitioning algorithm. In which case \mathbf{B}_r may be run through the tree optimization algorithm *opt_tree* illustrated in section 7.3. From experimental results, it has been found that running the optimization algorithm for only one iteration is usually sufficient to produce a near-optimal tree that is suitable for the partitioning algorithm.

The *right_update* algorithm can be better illustrated by the use of the following example.

Example 8.4

Partitioning a circuit represented by the graph in figure (8.2) in the fashion shown in figures (7.3 a and b) produces a \mathbf{B}_2 submatrix shown in equation (8.20). After deleting the zero columns from \mathbf{B}_2 the intermediate result becomes

$$\mathbf{B}_{temp} = \begin{matrix} & \begin{matrix} 2 & 6 & 12 & 9 & 11 & 13 & 17 \end{matrix} \\ \begin{matrix} 1 \\ 3 \\ 4 \\ 10 \\ 7 \\ 5 \end{matrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 \end{bmatrix} \end{matrix}$$

By inspection of figure (8.2) it can be seen that

$$d = \{ 9, 11, 13, 17 \}$$

Now applying the *right_update* algorithm to update the tree of the right partition results in:

- $i=1 \rightarrow l_1=1$
 $local_d=\{\emptyset\} \rightarrow$ no change in \mathbf{B}_{temp} .
- $i=2 \rightarrow l_2=3$
 $local_d=\{\emptyset\} \rightarrow$ no change in \mathbf{B}_{temp} .
- $i=3 \rightarrow l_3=4$
 $local_d=\{17\}$

Now step 4 is executed for l_j taking the values 10,7,5 respectively, but since non of them intersect with branch 17, no change occurs in these loops. Row 4 is now deleted and link 4 becomes a tree branch, which requires adding a new column to \mathbf{B}_{temp} which becomes,

$$\mathbf{B}_{temp} = \begin{matrix} & 2 & 6 & 12 & 9 & 11 & 13 & 17 & 4 \\ \begin{matrix} 1 \\ 3 \\ 10 \\ 7 \\ 5 \end{matrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \end{bmatrix} \end{matrix}$$

- $i=4 \rightarrow l_4=10$
 $local_d=\{13\}$

Since branch 13 intersects both L_5 and L_6 , both loops need to be updated using equation (8.22). That is:

$$\begin{aligned} L_5 &= L_4 \cup L_5 - L_4 \cap L_5 + l_4 \\ &= \{6, 12, 13\} - \{12, 13\} + \{10\} \\ &= \{6, 10\} \end{aligned}$$

and

$$\begin{aligned} L_6 &= L_4 \cup L_6 - L_4 \cap L_6 + l_4 \\ &= \{6, 9, 11, 12, 13\} - \{12, 13\} + \{10\} \\ &= \{6, 9, 10, 11\} \end{aligned}$$

So now \mathbf{B}_{temp} becomes,

$$\mathbf{B}_{temp} = \begin{matrix} & 2 & 6 & 12 & 9 & 11 & 13 & 17 & 4 & 10 \\ \begin{matrix} 1 \\ 3 \\ 7 \\ 5 \end{matrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 0 & 0 & 1 \end{bmatrix} \end{matrix}$$

• $i=5 \rightarrow l_5=7$

$local_d = \{\emptyset\} \rightarrow$ no change in \mathbf{B}_{temp} .

• $i=6 \rightarrow l_6=5$

$local_d = \{9, 11\}$

Now since no more loops are left after L_6 , row 6 is simply deleted and $l_6=5$ becomes a tree branch, which requires adding a new column to \mathbf{B}_{temp} . Also since $i=b_l=6$ here, the algorithm terminates after this step. After deleting the columns corresponding to the elements of d , \mathbf{B}_r becomes:

$$\mathbf{B}_r = \begin{matrix} & 2 & 6 & 12 & 4 & 10 & 5 \\ \begin{matrix} 1 \\ 3 \\ 7 \end{matrix} & \begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \end{bmatrix} \end{matrix}$$

The new tree and cotree are illustrated in figure (8.4).

8.6.4 The Binary Algorithm

The input to this part of the process is the weighted fundamental loop matrix

$$\mathbf{B} = [b_{ij}] \text{ where } (1 \leq i \leq b_t, 1 \leq j \leq b_l) \quad (8.23)$$

which includes the tree, cotree and fundamental loop information. Also, b_t and b_l signify the number of tree branches and cotree links in the circuit, respectively. The total number of branches in the circuit then is simply $b = b_l + b_t$.

The steps of the algorithm that is referred to as *binary-part* are listed herein.

1) Define and initialize the following vectors :

-- $(\mathbf{F} : f_j \in \mathbf{F}, 1 \leq j \leq b_l)$, where f_j is the number of nonzero entries in

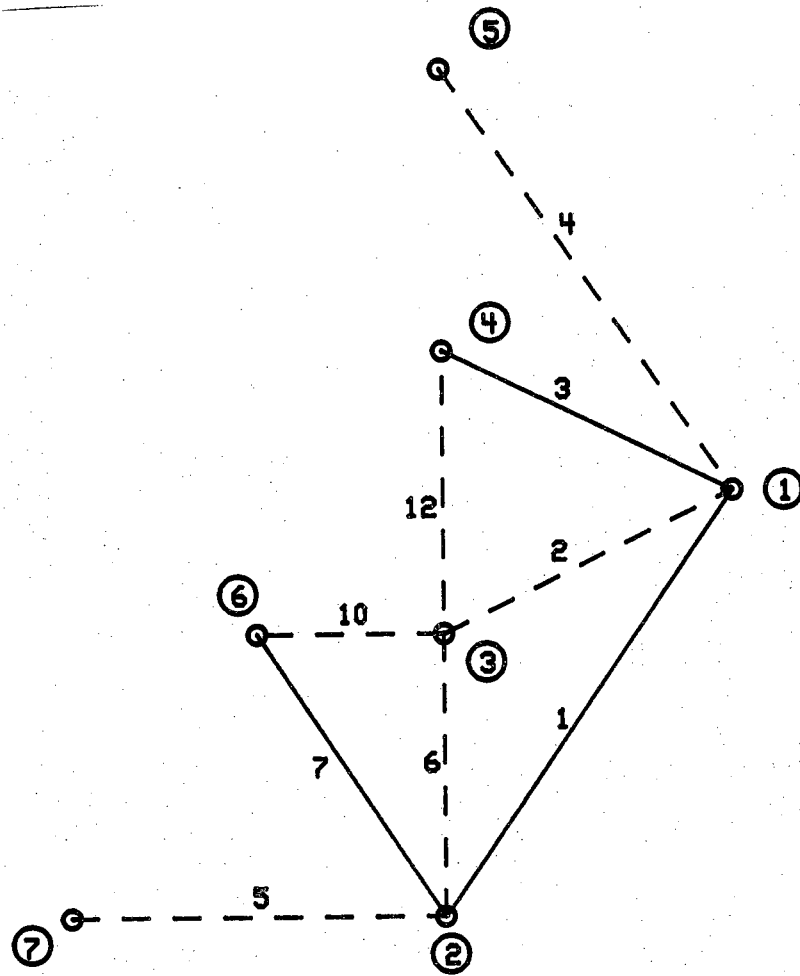


Figure 8.4 : New tree and cotree of the right partition

the j th column of bold B , that is

$$f_j = \sum_{k=1}^{b_i} b_{kj} \quad (8.24)$$

- (**D** : $d_j \in \mathbf{D}$, $1 \leq j \leq b_i$), where $d_j = 0$ if the j th tree branch has already been processed and 1 otherwise. Initialize every d_j to 1.
- (**DI** : $di_i \in \mathbf{D}$, $1 \leq i \leq b_l$), where di_i is the number of branches in the the 1st i fundamental loops of the circuit. In other words,

$$di_i = i + \sum_{k=1}^i \sum_{l=1}^{b_i} b_{kl} \quad (8.25)$$

- (**DN1** : $dn1_m \in \mathbf{DN1}$, $1 \leq m \leq n$), where n is the total number of nodes in the graph G and $dn1_m$ is 1 if node m is part of the first i fundamental loops, 0 otherwise. Initialize every $dn1_m$ to 0.
- (**DN2** : $dn2_m \in \mathbf{DN2}$, $1 \leq m \leq n$), where $dn2_m$ is the number of branches incident on node m and are not part of the first i fundamental loops. Initialize $dn2_m$ to the total number of branches incident on node m .

The significance of **F** and **D**, and **DN1** and **DN2** will be seen clearly in the calculation of **li** and **ti**, respectively.

2) For each fundamental loop L_i (ie. each row i of **B**) do the following :

- (2.1) for every loop L_p (p th row), where ($i \leq p \leq b_l$), which has an intersection with loops (rows) 1 through $i-1$, calculate the loop index li_p . For the case where no such intersection exist (which includes the initial case when $i=1$), li_p is calculated for all p where $i \leq p \leq b_l$. The equation used to calculate li_p is

$$li_p = li_{i-1} - \sum_{l=1}^{b_i} b_{pl} + \sum_{l=1}^{b_i} b_{pl} di_l f_l \quad (8.26)$$

- 2.2) Find the loop L_m (row m) with the minimum loop index li_m on the interval $i \leq m \leq b_l$.
- 2.3) Permute the m th row into the i th row of **B**.
- 2.4) Update **DN1** by setting $dn1_m = 1$ for all the nodes connected to the i th loop.
- 2.5) Update **DN2** by decrementing $dn2_m$ for every occurrence of node m in loop i . Note here that a maximum of two can be subtracted from any $dn2_m$ because a loop cannot have more than two

branches incident on one node.

2.6) Calculate ti_i using the following equation:

$$ti_i = \sum_{m=1}^n (dn1_m \bullet dn2_m) \quad (8.27)$$

where the " \bullet " is an and operation and is equal to 1 if both operands are nonzero and 0 otherwise.

2.7) Calculate di_i for the new i th row using

$$di_i = di_{i-1} + \sum_{l=1}^{b_i} b_{il} di_l + 1 \quad (8.28)$$

2.8) Update the \mathbf{D} vector by setting d_j to 0 for every nonzero entry b_{ij} in the newly permuted i th row.

2.9) Check if the $b_l - i$ loops still unpermuted have at least a set minimum number of branches that a partition may have b_{\min} , if so go to step 2 otherwise go to step 3. That is, if

$$b - di_i > b_{\min} \quad (8.29)$$

continue with the loop in step 2, else exit the loop.

3) The new permuted weighted fundamental loop matrix is now referred to as \mathbf{B}_p . Search for a starting loop (row) i_s in \mathbf{B}_p such that the first $i_s - 1$ loops have at least b_{\min} branches in them. That is until

$$di_{i_s} \geq b_{\min} \quad (8.30)$$

4) Search for a row i_{\min} which has the minimum *tearing index* ti_{\min} on the interval $\{i_s, i\}$. i_{\min} is the *partitioning point*.

So the first i_{\min} loops of \mathbf{B}_p represent the left partition of the circuit. The remaining $b_l - i_{\min}$ loops excluding the tree branches that are already part of the left partition, that is, the intersection between the 1st i_{\min} rows and the rest of \mathbf{B}_p , belong to the right partition.

A note is in order here. A fairly common special case is when one of the nodes of a tree branch is a dangling node. This means that the tree branch is not part of any fundamental loop, which results in a zero column corresponding to it in \mathbf{B} . So the partitioning process as stated above will always include these branches in the right subcircuit although its nondangling node is part of the left subcircuit. This is highly inefficient. An extra partitioning step is performed to handle these branches. A simple search through the tearing nodes of the left partition is conducted and a check is

performed if a branch with a dangling other node is connected to any of them; in which case, the branch is added to the left partition. It remains in the right partition otherwise.

Example 8.5

Consider the circuit represented by the graph in figure (8.5). Preprocessing the circuit yields the tree and cotrees highlighted in figure (8.5). The weighted fundamental loop matrix for this graph is then

$$\mathbf{B} = \begin{matrix} & & & & 1 & 6 & 7 & 9 & 11 \\ \begin{matrix} 2 \\ 3 \\ 4 \\ 5 \\ 8 \\ 10 \end{matrix} & \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 \end{array} \right] \end{matrix}$$

Performing the step 2 of *binary_tree* will yield the following partitioning table :

i	L	t_i	d_i
1	2	3	3
2	3	3	5
3	10	2	6
4	8	3	8
5	5	3	10
6	4	0	0

Step 2 went through 5 iteration. For $i=1$ and 2, no permutations were performed since they produced the minimum li at their respective position. For $i=3$ and 4 though, 2 permutations were performed. First exchanging the rows corresponding to $l_3=4$ and $l_6=10$, and second exchanging the rows corresponding to $l_4=5$ and $l_5=8$. For the last iteration, no exchange between $l_5=5$ and $l_6=4$ was performed. The resultant permuted \mathbf{B} matrix is

$$\mathbf{B}_p = \begin{matrix} & & & & 1 & 6 & 7 & 9 & 11 \\ \begin{matrix} 2 \\ 3 \\ 10 \\ 8 \\ 5 \\ 4 \end{matrix} & \left[\begin{array}{cccccc} 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 1 \end{array} \right] \end{matrix}$$

Performing steps 3 and 4 of the algorithm, the *partitioning point* is found to be the 3rd row corresponding to $l_4=10$. So the first three fundamental loops are the left partition. These include the branches $\{1,2,3,9,10,11\}$. The right partition consists of the rest of the loops less the tree branches already in the left partition. So the right partition includes branches include $\{4,5,6,7,8\}$. The partitioning nodes are node $\{1,4\}$. The two partitions are illustrated in figure (8.6).

The linear execution time complexity of *binary-part* can be computed by finding the approximate linear execution time T for the algorithm. Step 2 is the bottle neck of the process. Let $|L|$ be the average size of each loop, that is, the number of branches in each loop, which also corresponds to the number of nonzero entries in each row of \mathbf{B} . So the linear execution time of the algorithm to within a constant is given by the finite sum

$$T(b_l) = |L|_{ave} \cdot \sum_{q=1}^{b_l} (b_l - q) \quad (8.31)$$

which converges to [30]

$$T(b_l) = \frac{b_l(b_l+1)}{2} \cdot |L|_{ave} \quad (8.32)$$

From equation (8.32), it can be seen that *binary-part* is on the average

$$O(b_l^2 \cdot |L|_{ave}) \quad (8.33)$$

So the complexity of *binary-tree* is a function of the circuit size since, in general, $b_l = b - n - 1$, and the complexity expression of equation (8.33) can be rewritten as a function of the difference between the number of branches b and the number of nodes n .

Examining the problem from a practical point of view though, the practical and more realistic complexity of the algorithm can be obtained. First, examining $|L|_{ave}$; the average number of branches per fundamental

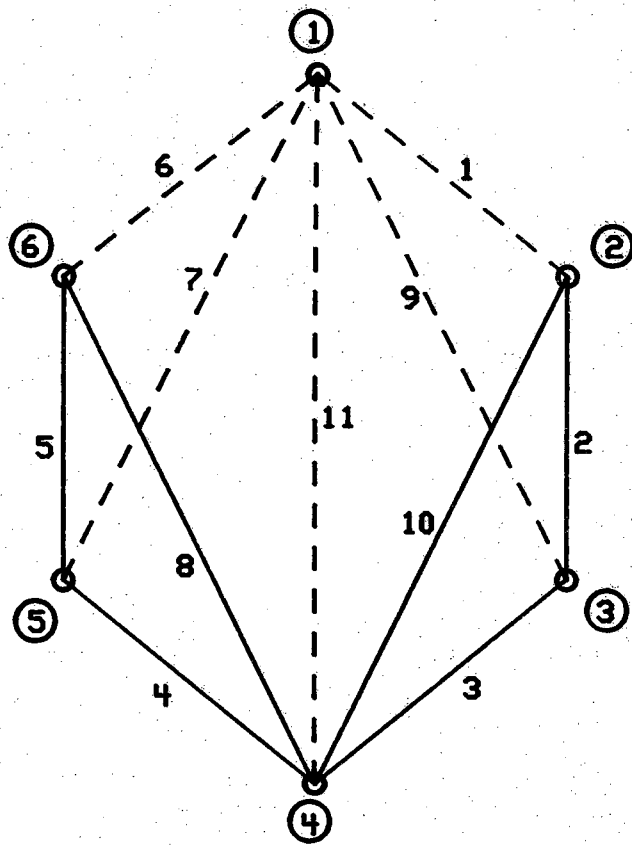


Figure 8.5 : Graph of example 8.5

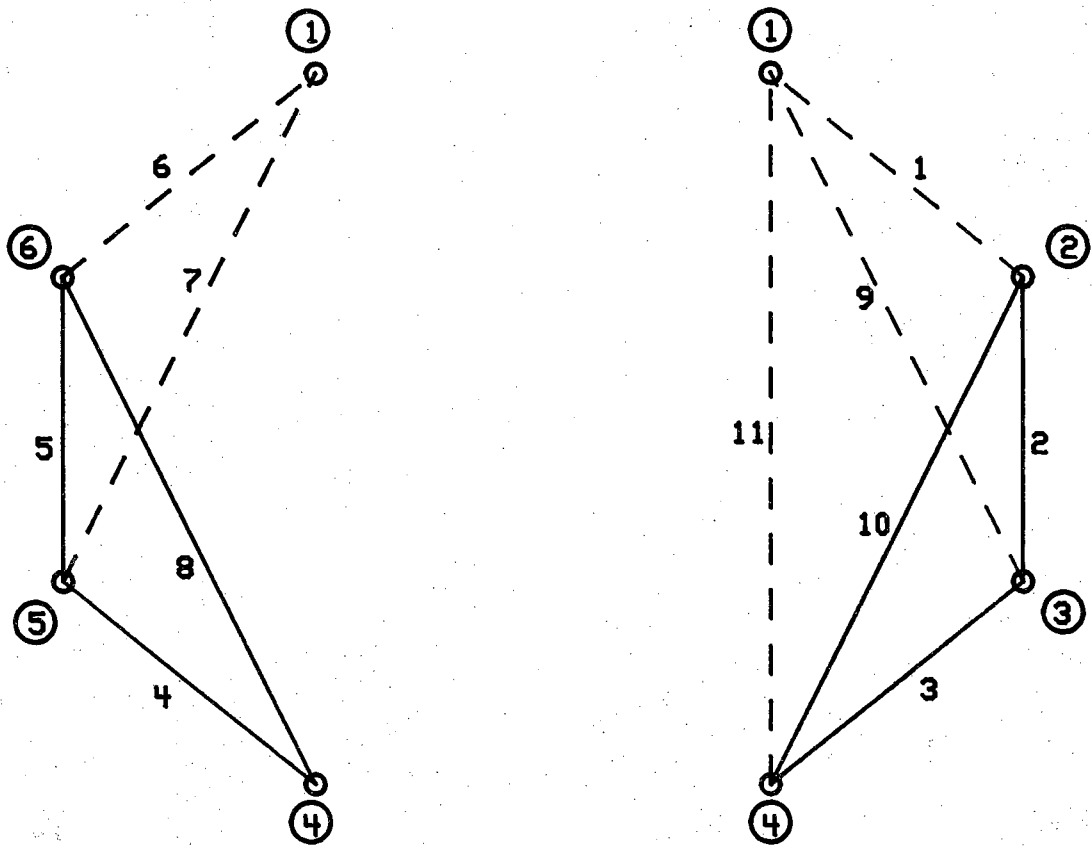


Figure 8.6 : Left and right partitions of example 8.5

loop, it is the goal of the preprocessing algorithm *opt_tree* to minimize the number of entries in the weighted fundamental loop matrix \mathbf{B} . This in turn minimizes the number of branches per fundamental loop $|L|_{ave}$. For general graphs, it has been found in [29] that an upper limit on $|L|_{ave}$ is six and tends to be independent of the size of the graph. This was achieved without the use of any tree optimization algorithms. In this project, it has the same observation was made, however because of the tree optimization algorithm used, the figure was found to be around five. So the practical time complexity of *binary_part* actually becomes

$$O(b_l^2). \quad (8.34)$$

CHAPTER 9

COMPARISONS AND EXAMPLES

9.1 Comparison with the Flowgraph Method

The aim of this section is to illustrate the advantages of the network method of symbolic analysis of large-scale circuits developed in this project and implemented in SCAPP, over the flowgraph method developed by Starzyk and Konczykowska in [25] and briefly explained in chapter 2. A Band-pass filter example simulated in [25] is used for this purpose. The comparison will show the savings in the number of mathematical operations that the network approach exhibits to evaluate the results of the analysis.

Example 9.1

The band-pass filter is shown in figure (9.1). The symbolic transfer function $\frac{v_2}{v_1}$ is the output requested from the symbolic analysis. The flowgraph analysis technique has been performed on the circuit in [25] using the Coates graph representation illustrated in figure (9.2) where one of the partitions of the flowgraph is shown. The partitioning was done manually and explores the repetitive nature of the flowgraph. The result of the analysis is the *sequence of expressions* shown in table (9.1). The final transfer function is given by:

$$\frac{v_2}{v_1} = \frac{F(24)}{F(25)} \quad (9.1)$$

The statistics on the analysis extracted from table (9.1) are listed in table (9.3).

The circuit was also simulated using SCAPP. It was run under three different conditions:

- 1) Manual partitioning performed on the circuit. This is done in order to perform a fair comparison with the flowgraph method. This manual partitioning explores the cascaded structure of the filter shown in figure

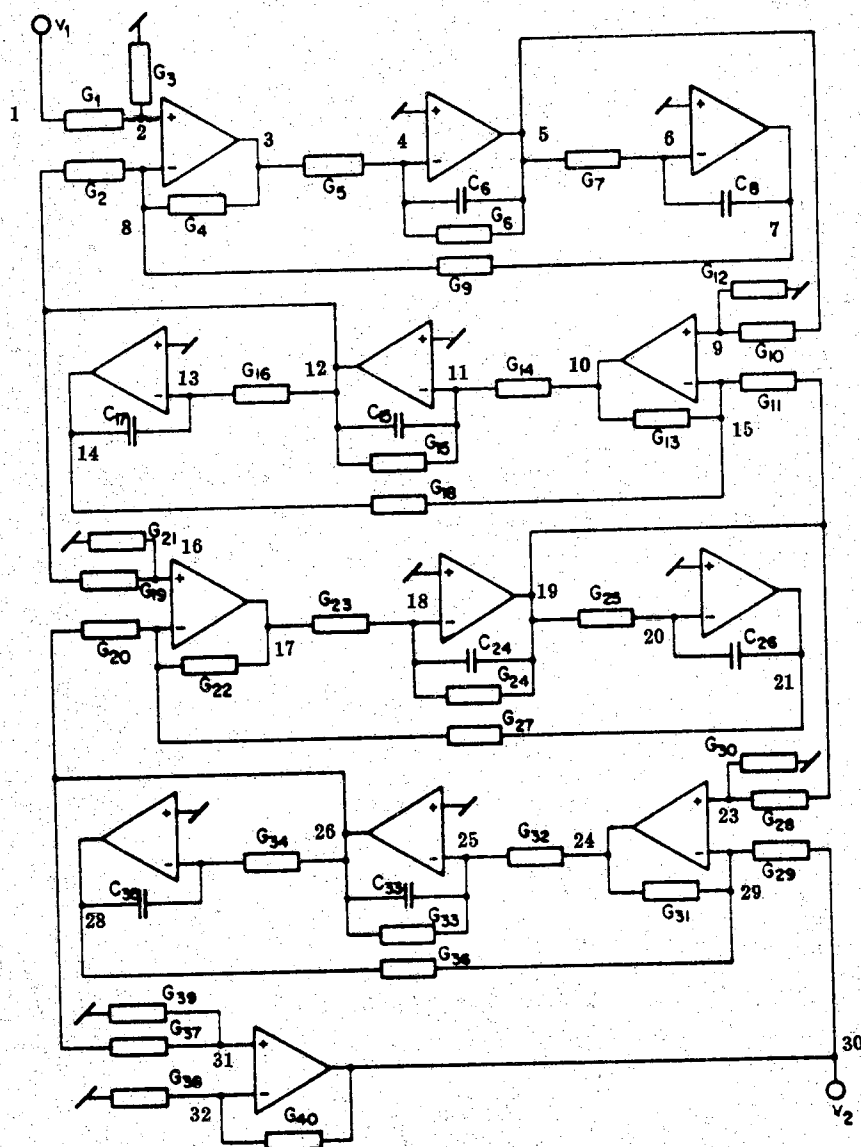


Figure 9.1 : Band-pass filter [25]

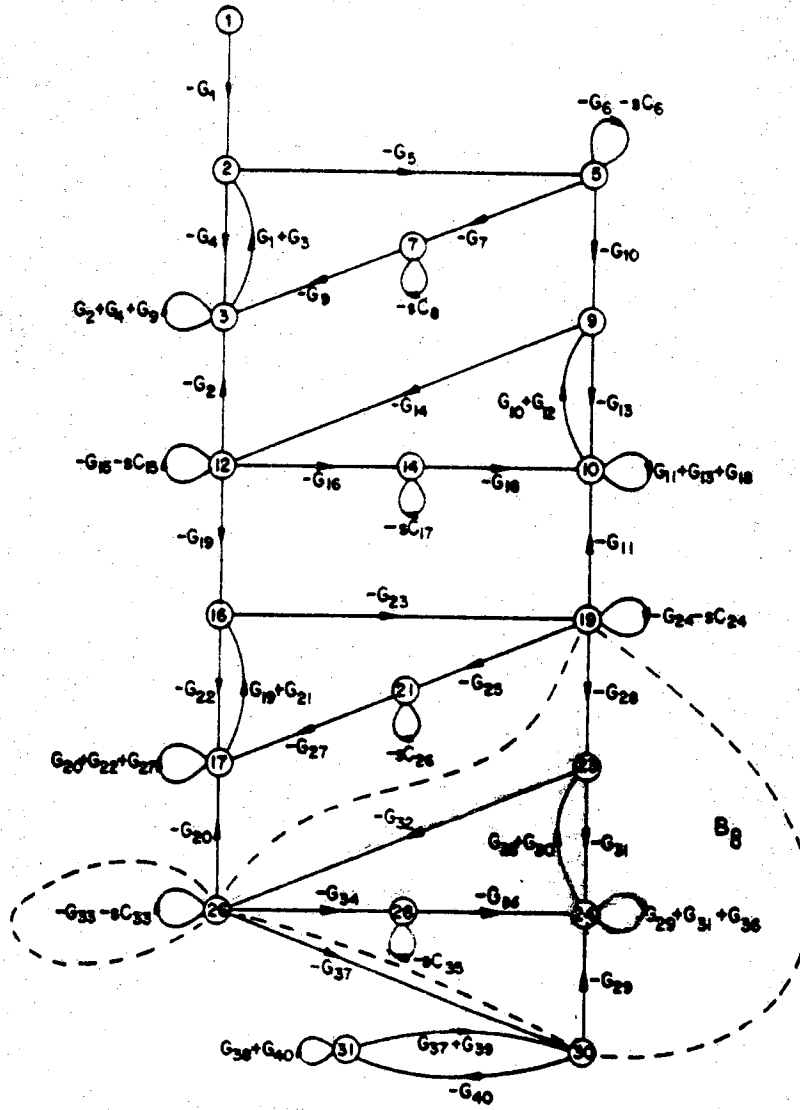


Figure 9.2 : Coates graph representation of example 9.1

Table 9.1 : The result of the flowgraph analysis

Block number	NB	B	E	Weight Function	
9	26,30	26	30	F(1)	$C_{37}(C_{38} + C_{40})$
		30	30	F(2)	$C_{40}(C_{37} + C_{39})$
8	19,26,30	30	26	F(3)	$C_{29}C_{32}C_{35}(C_{28} + C_{30})$
		26	26	F(4)	$(C_{28} + C_{30})[C_{32}C_{34}C_{36} + C_{31}C_{35}(C_{33} + C_{33})]$
		19	26	F(5)	$-C_{28}C_{32}C_{35}(C_{29} + C_{31} + C_{36})$
7	12,19,26	26	19	F(6)	$C_{20}C_{23}C_{26}(C_{19} + C_{21})$
		19	19	F(7)	$(C_{19} + C_{21})[C_{23}C_{25}C_{27} + C_{22}C_{26}(C_{24} + C_{24})]$
		12	19	F(8)	$C_{19}C_{23}C_{26}(C_{20} + C_{22} + C_{27})$
6	5,12,19	19	12	F(9)	$C_{11}C_{14}C_{17}(C_{10} + C_{12})$
		12	12	F(10)	$(C_{10} + C_{12})[C_{14}C_{16}C_{18} + C_{13}C_{17}(C_{15} + C_{15})]$
		5	12	F(11)	$C_{10}C_{14}C_{17}(C_{11} + C_{13} + C_{18})$
5	1,5,12	12	5	F(12)	$C_2C_3C_8(C_1 + C_3)$
		5	5	F(13)	$(C_1 + C_3)[C_5C_7C_9 + C_4C_8(C_6 + C_6)]$
		1	5	F(14)	$C_1C_3C_8(C_2 + C_4 + C_9)$
4	19,26,30	26,30	30,30	F(15)	$-F(3)F(1) + F(4)F(2)$
		19,26	26,30	F(16)	$F(5)F(1)$
		19,30	26,30	F(17)	$-F(5)F(2)$
3	12,19,30	12,19	19,30	F(18)	$F(8)F(16)$
		19,30	19,30	F(19)	$F(7)F(15) - F(6)F(17)$
		12,30	19,30	F(20)	$-F(8)F(15)$
2	5,12,30	5,12	12,30	F(21)	$F(11)F(18)$
		12,30	12,30	F(22)	$F(10)F(19) - F(9)F(20)$
		5,30	12,30	F(23)	$-F(11)F(19)$
1	1,30	1	30	F(24)	$F(14)F(21)$
		30	30	F(25)	$F(13)F(22) - F(12)F(23)$

(9.3). Figure (9.4) shows one of the subcircuits as a 4-terminal block and figure (9.5) shows the binary tree modeling the upward hierarchical analysis process. The resulting *sequence of expressions* is listed in table (9.2) and the statistics associated with it are listed in table (9.3).

- 2) Automatic partitioning is performed on the circuit. The resulting partitions are not the cascaded block of figure (9.3). The partitioner cannot recognize the repetitive structure of the circuits. However, its goal is to heuristically minimize the number of tearing nodes. The statistics from the resulting *sequence of expressions* are listed in table (9.3).
- 3) No partitioning performed on the circuit. SCAPP considers the whole circuit as one large 3-terminal block. The results are tabulated in table (9.3).

The transfer function requested from the analysis is given by:

$$\frac{v_2}{v_1} = -\frac{T0(32,1)}{T0(32,30)} \quad (9.2)$$

From table (9.3) the advantages of the network approach are obvious. To compare the *sequence of expressions*, the most important number to consider is the number of multiplications. Using manual partitioning, SCAPP has approximately a 24% advantage over the flowgraph approach. Also the advantage of using partitioning, automatic or manual, over not partitioning the circuit at all is very visible. The savings of manual partitioning is about 39% and of automatic partitioning is about 19% over the nonpartitioned case.

The advantages of the symbolic analysis methodology presented in this report over the flowgraph approach of [25] are summarized as follows:

- 1) The resulting *sequence of expressions* from SCAPP will have less mathematical operations than the flowgraph approach.
- 2) The results of the network approach have a division operation as the basic step for each block. Note that the results of the flowgraph approach only allows multiplications to be performed, no divisions. The problem for computer evaluation of the resulting expressions is the finite precision. The continuous division operations preserve the integrity of the numerical values of the functions. For instance, resistors with values in the mega ohm range will have conductances on the order of 10^{-6} mhos. The continuous multiplication of such numbers would result in severe underflow problems. One way to get around this would be by normalization, an added overhead to the flowgraph

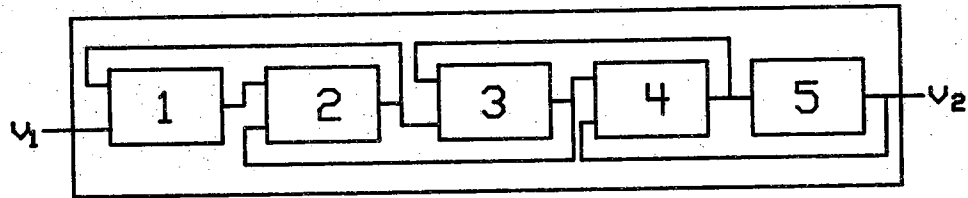


Figure 9.3 : Block interconnections for manual partitioning

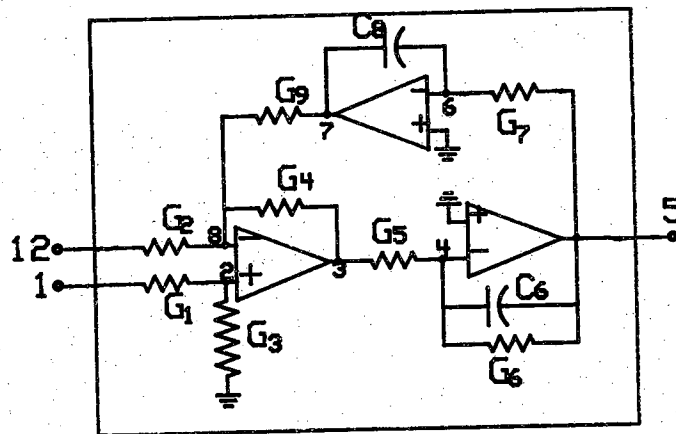


Figure 9.4 : A single block

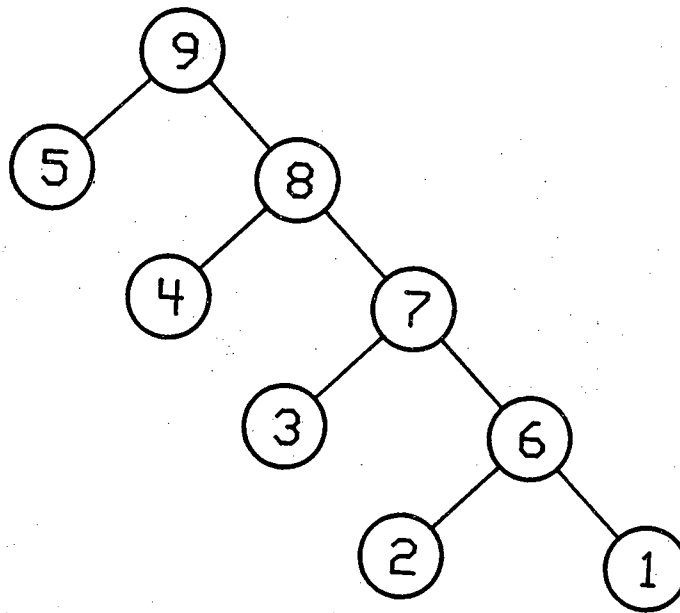


Figure 9.5 : The binary tree model of example 9.1

Table 9.2 : SCAPP result with manual partitioning

(a) Terminal blocks

Block Number	TN	Symbolic Function	
5	26,30	$P(1)$	$(G_{37}+G_{39})$
		$P1(32)$	$(G_{38}+G_{40})/(P(1))$
		$T1(32,26)$	$-P1(32)*(-G_{37})$
4	19,26,30	$P(3)$	$(G_{28}+G_{30})$
		$P3(29)$	$(G_{29}+G_{31}+G_{36})/(P(3))$
		$T3(29,19)$	$-P3(29)*(-G_{28})$
		$P3(25)$	$(-G_{32})/(-G_{31})$
		$T3(25,30)$	$-P3(25)*(-G_{29})$
		$T3(25,19)$	$-P3(25)*(T3(29,19))$
		$T3(25,28)$	$-P3(25)*(-G_{30})$
		$P3(25)$	$(T3(25,28))/(-sC_{35})$
		$T3(25,26)$	$(-G_{33}-sC_{33})-P3(25)*(-G_{34})$
3	12,19,26	$P(5)$	$(G_{19}+G_{21})$
		$P5(22)$	$(G_{20}+G_{22}+G_{27})/(P(5))$
		$T5(22,12)$	$-P5(22)*(-G_{19})$
		$P5(18)$	$(-G_{23})/(-G_{22})$
		$T5(18,12)$	$-P5(18)*(T5(22,12))$
		$T5(18,26)$	$-P5(18)*(-G_{20})$
		$T5(18,21)$	$-P5(18)*(-G_{27})$
		$P5(18)$	$(T5(18,21))/(-sC_{26})$
		$T5(18,19)$	$(-G_{24}-sC_{24})-P5(18)*(-G_{25})$
2	5,12,19	$P(7)$	$(G_{10}+G_{12})$
		$P7(15)$	$(G_{11}+G_{13}+G_{18})/(P(7))$
		$T7(15,5)$	$-P7(15)*(-G_{10})$
		$P7(11)$	$(-G_{14})/(-G_{13})$
		$T7(11,5)$	$-P7(11)*(T7(15,5))$
		$T7(11,19)$	$-P7(11)*(-G_{11})$
		$T7(11,14)$	$-P7(11)*(-G_{18})$
		$P7(11)$	$(T7(11,14))/(-sC_{17})$
		$T7(11,12)$	$(-G_{15}-sC_{15})-P7(11)*(-G_{16})$

Table 9.2 (continued)

Block Number	TN	Symbolic Function	
1	1,5,12	$P(8)$	$(G_1 + G_2)$
		$P8(1)$	$(-G_1)/(P(8))$
		$T8(1,1)$	$(G_1) - P8(1) * (-G_1)$
		$P8(8)$	$(G_2 + G_4 + G_6)/(P(8))$
		$T8(8,1)$	$-P8(8) * (-G_1)$
		$P8(4)$	$(-G_6)/(-G_4)$
		$T8(4,1)$	$-P8(4) * (T8(8,1))$
		$T8(4,12)$	$-P8(4) * (-G_2)$
		$T8(4,7)$	$-P8(4) * (-G_6)$
		$P8(4)$	$(T8(4,7))/(-sC_2)$
		$T8(4,5)$	$(-G_6 - sC_6) - P8(4) * (-G_7)$

(b) Middle blocks

Block Number	TN	Symbolic Function	
6	1,12,19	$P6(11)$	$(T7(11,5))/(T8(4,5))$
		$T6(11,1)$	$-P6(11) * (T8(4,1))$
		$T6(11,12)$	$(T7(11,12)) - P6(11) * (T8(4,12))$
7	1,19,26	$P4(18)$	$(T5(18,12))/(T6(11,12))$
		$T4(18,1)$	$-P4(18) * (T6(11,1))$
		$T4(18,19)$	$(T5(18,19)) - P4(18) * (T7(11,19))$
8	1,26,30	$P2(25)$	$(T3(25,19))/(T4(18,19))$
		$T2(25,1)$	$-P2(25) * (T4(18,1))$
		$T2(25,26)$	$(T3(25,26)) - P2(25) * (T5(18,26))$
9	1,30	$P0(32)$	$(T1(32,26))/(T2(25,26))$
		$T0(32,1)$	$-P0(32) * (T2(25,1))$
		$T0(32,30)$	$(-G_{40}) - P0(32) * (T3(25,30))$

Table 9.3 : Results for example 9.1.

Type of Analysis	Type of Partitioning	Number of mults	Number of adds	Number of eqs
Flowgraph	manual	63	30	25
SCAPP	manual	48	27	56
	automatic	64	31	73
	none	79	35	88

would be by normalization, an added overhead to the flowgraph method. The division operations in the network approach will keep the precision of the numbers at a set range through the evaluation process, thus eliminating any need for a normalization procedure.

- 3) All the intermediate results of SCAPP; that is, any terminal or middle block analysis result, has a physical meaning associated with a subcircuit. The equations represent the entries of the RMNA matrix characterizing the block. This is the case because the analysis is based on network partitioning. The flowgraph method however, is based on the partitioning of the Coates graph. The intermediate results do not directly correspond to a physical partition of the circuit. Mapping these intermediate results back to the circuit would require additional algorithmic manipulations.
- 4) For the network approach it is directly possible to build standard terminal block libraries for common circuit building blocks, opamps, amplifiers, NAND gates, NOR gates, etc. The flowgraph approach would need the exploration of the process of interconnecting Coates graph representations of interconnected subcircuits; an added overhead.

9.2 General Example

This section will explore the complete process of symbolically analyzing a circuit using SCAPP. The partitioning tables are all listed. Table 9.7 shows a comparison for the symbolic output between the partitioned case and the unpartitioned case.

Example 9.2

The circuit in figure (9.6) is to be analyzed using SCAPP. The resulting binary tree model that will result from the analysis is displayed in figure (9.7). The input deck for SCAPP is shown in figure (9.8). Notice that four partitions have been requested from the partitioner.

The first step of the analysis is to find a tree and to optimize it. The search in SCAPP for a star-like tree starts at the ground; the densest node in the circuit. The star-like tree found by *find_tree* is an optimum tree. So *opt_tree* was not needed in this case. Figure (9.6) highlights the tree by the dashed branches. The resulting fundamental loop matrix **B** is given in figure (9.9).

This matrix is actually **B** after the rows have been permuted based on their loop indices *li*. The partitioning table with *li*, *di* and *ti* computed for each row is listed in table (9.4).

Minimum *li* is at row c_{16} which becomes the *partitioning point* in **B**. The first three fundamental loops are then grouped together to form block **B**₁ and the rest of the loops belong to **B**₂. The two partitions are:

$$\mathbf{B}_1 = \{ r_9, r_{15}, c_{16}, r_8, c_{13}, r_{14} \}$$

$$\mathbf{B}_2 = \{ r_2, r_{18}, e_7, c_{19}, r_{21}, r_5, r_{11}, c_4, c_1, f_{10}, r_{12}, r_{17}, r_6, r_{20}, c_3 \}$$

The set of tearing nodes is:

$$TN^{1,2} = \{0,7\}$$

Figure (9.7) shows the position of **B**₁ and **B**₂ on the binary tree.

The left subcircuit **B**₁ cannot be partitioned any further because of the branch size limitation of $b_{\min}=4$. **B**₁ actually has 6 branches which cannot be partitioned into two parts with at least 4 branches in each.

Now the right subcircuit **B**₂ is entered to the binary partitioning process after its tree has been updated. The one iteration of *opt_tree* results in

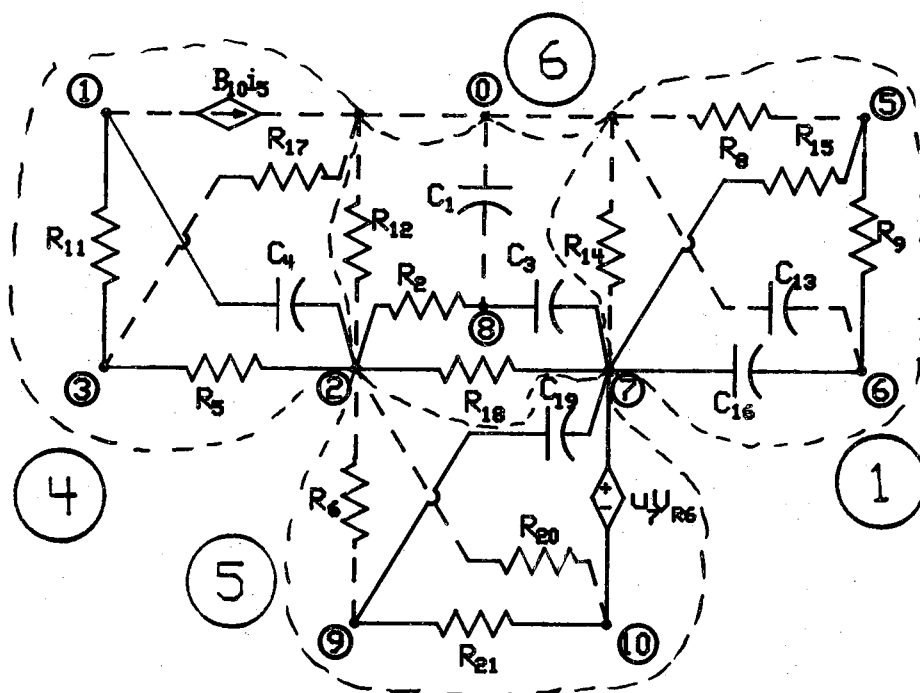


Figure 9.6 : Circuit of example 9.2

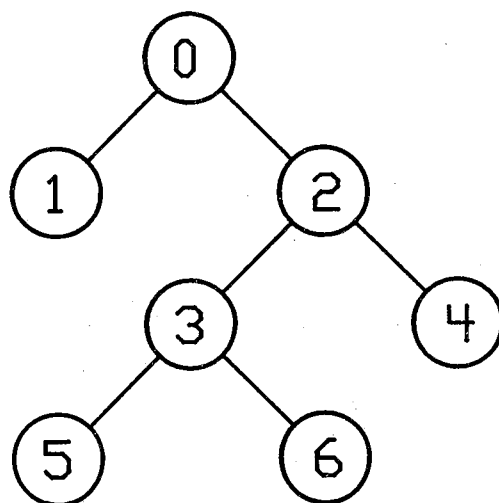


Figure 9.7 : The binary tree model for example 9.2

```

* Example 9.2 : A dense circuit
* Includes VCVS (branch 17)
* and CCCS (branch 10)
*
*

```

```

.opt bmin 4
.opt parts 4
.opt alltree 1
.opt auto 1

```

```

c1 0 8 VAR
r2 2 8 VAR
c3 7 8 VAR
c4 1 2 VAR

```

```

r5 2 3 VAR
r6 2 9 VAR
e7 7 10 2 9 VAR = u7
r8 0 5 VAR
r9 5 6 VAR
f10 1 0 r5 VAR = B10
r11 1 3 VAR
r12 0 2 VAR
c13 0 6 VAR
r14 0 7 VAR
r15 5 7 VAR
c16 6 7 VAR
r17 3 0 VAR
r18 2 7 VAR
c19 7 9 VAR
r20 2 10 VAR
r21 9 10 VAR
.output v8 v10

```

Figure 9.8 : Input to SCAPP for circuit of example 9.2

$$\begin{array}{l}
 r_9 \\
 r_{15} \\
 c_{16} \\
 c_3 \\
 r_2 \\
 r_{18} \\
 e_7 \\
 c_{19} \\
 r_{21} \\
 r_5 \\
 r_{11} \\
 c_4
 \end{array}
 \begin{bmatrix}
 c_1 & r_8 & f_{10} & r_{12} & c_{13} & r_{14} & r_{17} & r_6 & r_{20} \\
 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
 0 & 0 & 0 & 1 & 0 & 1 & 0 & 1 & 0 \\
 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\
 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0
 \end{bmatrix}$$

Figure 9.9 : **B** matrix for block 0 (entire circuit)

Table 9.4 : Partitioning table for block 0 (entire circuit)

i	L	li	di	ti
1	r_9	2	3	3
2	r_{15}	6	5	3
3	c_{16}	4	6	2
4	c_3	4	8	3
5	r_2	8	10	3
6	r_{18}	6	11	3
7	e_7	5	13	4
8	c_{19}	4	15	4
9	r_{21}	2	16	2
10	r_5	2	18	3
11	r_{11}	0	0	0
12	c_4	2	0	0

reducing the entries of the new **B** matrix from $qs=21$ to $qs=16$. The permuted **B** matrix for this partition is:

$$\begin{matrix} c_1 \\ c_3 \\ e_7 \\ c_{19} \\ r_{21} \\ r_5 \\ r_{11} \\ c_4 \end{matrix} \begin{bmatrix} r_2 & f_{10} & r_{12} & r_{17} & r_6 & r_{20} & r_{18} \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The partitioning table for this block becomes:

Table 9.5 : Binary partitioning table of block 2

i	L	li	di	ti
1	c_1	3	3	3
2	c_3	4	5	3
3	e_7	4	7	4
4	c_{19}	4	9	4
5	r_{21}	2	10	2
6	r_5	2	12	3
7	r_{11}	0	0	0
8	c_4	0	0	0

The *partitioning point* is r_{21} . The resulting partitions are:

$$B3 = \{c_1, c_3, e_7, c_{19}, r_{21}, r_2, r_{12}, r_6, r_{20}, r_{18}\}$$

$$B4 = \{r_{11}, c_4, f_{10}, r_{17}, r_5\}$$

The tearing node set between blocks 3 and is:

$$TN^{3,4} = \{0,2\}$$

The left partition is considered for yet another binary partitioning process.

The resulting **B** matrix and partitioning tables are:

$$\begin{matrix} c_1 \\ c_3 \\ e_7 \\ c_{19} \\ r_{21} \end{matrix} \begin{bmatrix} r_2 & r_{12} & r_6 & r_{20} & r_{18} \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \end{bmatrix}$$

Table 9.6 : Binary partitioning table of block 3

i	L	li	di	ti
1	c_1	1	3	2
2	c_3	2	5	2
3	e_7	2	7	3
4	c_{19}	0	0	0
5	r_{21}	0	0	0

The resulting blocks are:

$$B5 = \{c_1, c_3, r_2, r_{12}, r_{18}\}$$

$$B6 = \{c_{19}, r_{21}, r_6, r_{20}, e_7\}$$

The tearing node set is:

$$TN^{4,5} = \{2,7\}$$

At this point, further partitioning is not possible because of the subcircuit size limitation. The final partitioned circuit is illustrated in figure (9.6) and the terminal blocks are:

$$B1 = \{r_9, r_{15}, c_{16}, r_8, c_{13}, r_{14}\}$$

$$B4 = \{c_1, c_3, r_2, r_{12}, r_{18}\}$$

$$B5 = \{c_{19}, r_{21}, r_6, r_{20}, e_7\}$$

$$B6 = \{r_{11}, c_4, f_{10}, r_{17}, r_5\}$$

The final set of tearing nodes which must include the I/O variables is:

$$TN = \{0, 2, 7, 8, 10, \}$$

SCAPP now proceeds with the analysis of the partitioned circuit. It will perform a terminal block analysis on each terminal block and then trace up the binary tree for the hierarchical middle block analysis process. The statistics of the result are listed in table (9.7).

The circuit was run without partitioning. From the discussion in chapter 6 it is clear that the order in which the nodes are reduced in the terminal block analysis has an effect on the efficiency of the running and the results. The unpartitioned circuit was simulated using several different random ordering of the reduction. The two basic set of numbers the simulations resulted in are displayed in table (9.7). For the partitioned case, the order did not affect the analysis results.

The advantage of partitioning here is very clear, the partitioned case requires 77% less multiplications than the first random case and 35% less than the second case.

Table 9.7 : Results for example 9.2

Type of Partitioning	Type of Order	Reduction mults	Number of adds	Number of eqs
automatic	all	66	70	75
none	random	287	216	292
	random	102	89	108

CHAPTER 10

SUMMARY, CONCLUSIONS AND RECOMMENDATIONS

10.1 Summary and Conclusions

A new method for the symbolic analysis of large-scale circuits has been developed. The method has been implemented in a software package using the C programming language. The program, referred to as SCAPP, an abbreviation for Symbolic Circuit Analysis Program with Partitioning, is a unique user oriented tool for symbolic circuit analysis of large-scale networks. At the present time, the only other tools available for symbolic analysis are limited to a circuit size of about 50 nodes [3].

The hierarchical network analysis method proposed consists of three basic parts. They are:

- 1) Network partitioning: The partitioning algorithm is based on the process of finding an initial spanning tree for the network, building the weighted fundamental loop matrix \mathbf{B} and then finding a *near-optimal* spanning tree. After that, a hierarchical binary partitioning process is performed. It consists of permuting the rows of \mathbf{B} based on the concept of *loop index* to group tightly coupled fundamental loops together, and then partitioning the network using the new concept of *tearing index* to perform the actual binary partitioning. The goal of the process is to minimize the number of tearing nodes while considering the size of the resulting partitions. This in turn will optimize the symbolic analysis process. The concept of *pseudo-branches* was introduced in order to allow for user defined subcircuits to be present in the network and considered in the partitioning process.

The result was an efficient network partitioning process with a main goal of optimizing the hierarchical symbolic analysis by minimizing the number of tearing nodes while considering the size of the partitions. The process was found highly suitable for parallel implementation.

- 2) Terminal block analysis: A further modification to the modified nodal analysis technique has been introduced in order to allow ideal opamps in the symbolic networks. Also the concept of the reduced modified analysis (RMNA) was introduced to produce a symbolic matrix that characterizes the subcircuits in terms of their tearing nodes and extra current variables.

The RMNA reduction technique was proven to be very stable for practical electrical networks. Also, it was found that this terminal analysis methodology is able to perform the analysis on the entire network without partitioning. This allowed for the flexibility in using the partitioning. Another result of this terminal block analysis method is that it allows for the characterization and building of standard symbolic libraries for standard circuit building blocks. The partitioning is capable of handling these blocks as user defined subcircuits. This results in a more efficient analysis process. Each terminal block analysis result is a complete solution of the subcircuit in terms of its external variables. The terminal block analysis process is completely independent from any of the other analysis steps, middle or terminal. So, the parallel of the terminal blocks would result in a major speed-up in the complete symbolic analysis.

- 3) Hierarchical middle block analysis: A binary hierarchical combination process of terminal and middle blocks was proposed. The solutions are represented as RMNA matrices. The analysis proceeds up the binary tree model dictated by the partitioning.

The two criteria that control the efficiency of the process, in their order of importance, are: the number of tearing nodes, and secondly the size of the partitions. This is where the constraints on the partitioning technique arise.

Therefore, the main conclusion of this project is that large-scale networks can be analyzed symbolically. The presentation of the hierarchical network approach for the symbolic analysis of large-scale networks and its computer implementation in SCAPP proves the argument.

10.2 Recommendations

The main consideration for the symbolic analysis of large-scale networks is reducing a large problem into several smaller problems, solving them, and

then combining the solutions. In this project, this process corresponds to the partitioning, terminal block analysis and hierarchical middle block analysis, respectively. In the author's opinion, the most important consideration when attempting to analyze large problems is : the ability to easily combine the solutions of similar but smaller size problems to produce a solution to the large problem. The following recommendations are based on this consideration.

- 1) Exploring the possibility of using one of the existing topological approaches, like parameter extraction [3], to perform the terminal block analysis. Actually further modifications and improvements would have to be realized in the methods because of their exponential growth behavior with the growth in subcircuit sizes. Otherwise, a very powerful partitioner must be used such that the size of the each partition is the main criteria for partitioning. The partitioner would almost have to guarantee such optimum sizes. Since partitioning is of the class of *NP-complete* problems [1], guaranteeing such an outcome could be very costly. All practical partitioners are heuristic. Therefore, the analysis methodology should be flexible and efficient enough to handle occasional large partitions. Also, the recombination abilities of these methods must be studied in detail.
- 2) Exploring the possibility of using other network approaches for the terminal block analysis, like the sparse tableau method [9] or loop analysis [7]. Here the research must begin by the applicability of the methods to symbolic analysis. After that, the same considerations of the combination abilities of the method must be studied.
- 3) The exploration of different partitioning techniques, like ones based on branch tearing rather than node tearing, or a hybrid combination of both. This is highly related to the type of terminal block analysis used. For instance, a nodal analysis technique, like the one developed in this project, would conceptually need a node tearing partitioning technique. The restrictions on the partitioner are imposed by the analysis methodology. However, even within each class of partitioning, node tearing or branch tearing, there is a vast area of possibilities.
- 4) The exploration of the symbolic sensitivity analysis of large-scale

networks. This would involve finding the first and possibly higher order derivatives of variables involved in the *sequence of expressions*.

- 5) A practical application for SCAPP would be to build a standard library of symbolic RMNA matrices for standard circuit building blocks. Something similar to a standard cell library. This would save the cost of terminal block analysis for circuits utilizing these standard building blocks.
- 6) Extensively studying the roundoff error and the effects of the numerical evaluation of the *sequence of expressions*. This study should be performed with regard to the expression patterns produced by the symbolic analysis of electrical circuits and not any general expression sequence. The former will produce specific repetitive patterns, like the output of SCAPP versus the latter could have infinite possibilities. An attempt can be made to find some level of recurrence in the patterns.
- 7) And finally the extensive testing of SCAPP. As with any software product the bugs are endless. Extensive use of the program will encourage its use, make for the addition of new features to it and explore the application of symbolic circuit analysis in many areas.

LIST OF REFERENCES

LIST OF REFERENCES

- [1] A. V. Aho, et. al., The Design and Analysis of Computer Algorithms. Addison- Wesly, 1974.
- [2] G. E. Alderson, and P. M. Lin, "Computer Generation of Symbolic network functions-A new theory and implementation," IEEE Trans. on Circuit Theory, Vol. CT-20, pp. 48-56, Jan 1973.
- [3] G. E. Alderson," Semi-Numerical Analysis of Large Networks," Ph.D. Thesis, Purdue University, August 1971.
- [4] L. T. Bruton, RC-Active Circuits. Prentice Hall, Englewood Cliffs, NJ., 1980.
- [5] N. Balabanian, and T. A. Bickart, Electrical Network Theory. John Wiley & Sons, 1969.
- [6] W. K. Chen, "Topological Analysis for Active Networks," IEEE Trans. on Circuit Theory, Vol. CT-12, no. 1,pp. 85-91, Mar. 1965.
- [7] L. O. Chua, and P. M. Lin, Computer Aided Analysis of Electronic Circuits - Algorithms and Computational Techniques. Englewood Cliffs, NJ: Prentice-Hall, 1975.
- [8] C. L. Coates, "Flow graph Solutions of Linear Algebraic Equations," IRE Trans. on Circuit Theory, Vol. CT-6, pp. 170-187, 1959.
- [9] G. D. Hachtel, et. al., "The Sparse Tableau Approach to Network and Design," IEEE Trans. on Circuit Theory, Vol. CT-18, pp. 101-113, Jan 1971.

- [10] G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "A Survey of Third-Generation Simulation Techniques," Proc. IEEE, Vol. 69, pp. 1264-1280, Oct. 1981.
- [11] I. N. Hajj, and D. G. Saab, "Symbolic Logic Simulation of MOS Circuits," Proc. IEEE Int. Symposium on Circuits and Systems, May 1983, pp. 246-249.
- [12] C. Ho, A. E. Ruehli, and Brennan, "The Modified Nodal Approach to Network Analysis," IEEE Trans. on Circuits and Systems, Vol. CAS-25, pp. 504-509, June 1975.
- [13] A. Konczykowska, and J. A. Starzyk, "Computer Justification of Upward Topological Analysis of Signal-Flow Graphs," Proc. European Conf. on Circuit Theory Design, pp. 464-467, 1981.
- [14] P. M. Lin, "Symbolic Methods for Large-Scale Circuits," A research proposal for NSF, Purdue University.
- [15] P. M. Lin, "A Survey of Applications of Symbolic Network Functions," IEEE Trans. on Circuit Theory, Vol. CT-20, pp. 732-737, Nov. 1973.
- [16] P. M. Lin, "Computer Generation of Symbolic Network Functions: an Overview," in Computer-Aided Design, edited by J. Vlietstra and R. F. Wielinga, Amsterdam, The Netherlands: North Holland, 1973.
- [17] P. M. Lin, and G. E. Alderson, "SNAP - A Computer Program for Generating Symbolic Network Functions," School of EE, Purdue University, West Lafayette, IN., Rep. TR-EE 70-16, Aug. 1970.
- [18] B. J. Leon, and P. A. Wintz, Basic Linear Networks for Electrical and Electronics Engineers. Holt, Rinehart and Winston, Inc., 1970.
- [19] R. R. Mielke, "A New Signal Flowgraph Formulation of Symbolic Network Functions," IEEE Trans. on Circuits and Systems, Vol. CAS-25, pp. 334-340, June 1978.

- [20] S. K. Mitra, Analysis and Synthesis of Linear Active Networks. John Wiley & Sons, Inc., 1969.
- [21] L. W. Nagel, "SPICE2: A Computer Program to Simulate Semiconductor Circuits," Memo ERL-M520, Electronics Res. Lab., University of California at Berkeley, May 1975.
- [22] V. B. Rao, and T. N. Trick, "Network Partitioning and Ordering for MOS VLSI circuits," IEEE Trans. on Computer-Aided Design, Vol. CAD-6, number 1, pp. 128-144, Jan. 1987.
- [23] A. Sangiovanni-Vincentelli, L. Chen, and L. O. Chua, "A New Tearing Approach -- Node-Tearing Nodal Analysis," Proc. IEEE Int. Symposium on Circuits and Systems, pp. 143-147, 1977.
- [24] A. Sangiovanni-Vincentelli, L. K. Chen, and L. O. Chua, "An efficient heuristic cluster algorithm for tearing large-Scale Networks," IEEE Tran. on Circuits and Systems, Vol CAS-24, pp. 709-717, Dec. 1977.
- [25] J. A. Starzyk, and A. Konczykowska, "Flowgraph Analysis of Large Electronic Networks," IEEE Trans. on Circuits and Systems., Vol CAS-33, pp. 302-315, March 1986.
- [26] M. N. S. Swamy, and K. Thulasiraman, Graphs, Networks and Algorithms. John Wiley & Sons, 1981.
- [27] J. White, and A. L. Sangiovanni-Vincentelli, "Partitioning Algorithms and Parallel Implementation of Waveform Relaxation Algorithms For Circuit Simulation," Department of EECS, University of California at Berkeley, 1986.
- [28] F. F. Wu, "Solution of Large-Scale Networks by Tearing," IEEE Trans. on Circuits and Systems, Vol. CAS-23, No. 12, pp. 706-713, December 1987.
- [29] J. Rutkowski, "Heuristic Network Partitioning Algorithm Using the Concept of Loop Index," IEE Proceedings, Vol. 131, Pt. G, No. 5, pp. 203-208, October 1984.

- [30] E. W. Swokowski, *Calculus With Analytic Geometry*. Boston, MA: Prindle, Weber & Schmidt, 1978.
- [31] B. Noble, and J. W. Daniel, *Applied Linear Algebra*. Englewood Cliffs, NJ: Prentice-Hall Inc., 1977.
- [32] S. K. McGrogan, "Parallel Circuit Simulation Using SPICE on an ELXSI 6400," *ELXSI System 6400 Documentation*, August 1985.
- [33] J. E. Barbay, and G. W. Zobrist, "Distinguishing Characteristics of Optimum tree," 5th Allerton Conference on Circuits and Systems Theory, pp. 730-737, 1967.
- [34] J. E. Barbay, and G. W. Zobrist, "Optimum Tree Generation by Tree Transformation," 11th Midwest Symposium on Circuits Theory, pp. 463-471, 1968.
- [35] W. H. Hayt, Jr., and J. E. Kemmerly, *Engineering Circuit Analysis*. McGraw-Hill Book Co., 1978.
- [36] W. T. Weeks, A. J. Jiminez, G. W. Mahoney, D. Mehta, H. Qassemzadeh, and T. R. Scott, "Algorithms for ASTAP-A network analysis program," *IEEE Trans. on Circuit Theory*, Vol. CT-20, pp. 628-634, Nov. 1973.
- [37] M. Hassoun and P. M. Lin, "Performance Analysis of a Relaxation Method for Simulation of Large-Scale Circuits," *Proc. IEEE Int. Symposium on Circuits and Systems*, 1985.
- [38] J. E. Kleckner, "Advanced Mixed Mode Simulation Techniques," Memo ERL M84/48, Electronics Res. Lab., University of California at Berkeley, June 1984.
- [39] H. DeMan, et. al., "DIANA: Mixed Mode Simulator with a hardware description language for hierarchical design of VLSI," *Proc. IEEE ICC*, pp. 356-360, 1980.

- [40] B. W. Kernighan, and D. M. Ritchie, The C programming Language. Englewood Cliffs, NJ: Prentice-Hall, 1978.
- [41] J. K. Fidler, and J. I. sewell, "Symbolic Analysis for Computer-Aided Circuit Design-The Interpolative Approach," IEEE Trans. on Circuit Theory, Vol. CT-20, November 1973.
- [42] K. S. Yeung, "Symbolic Network Function Generation Via Discrete Fourier Transform," IEEE Trans. on Circuits and Systems, Vol. CAS-31, February 1984.
- [43] C. Pottle, CORNAP User Manual, School of Electrical Engineering, Cornell University, Ithaca, NY, 1968.
- [44] L. P. McNamee and H. Potash, "A User's and Programmer's Manual for NASAP," University of California at Los Angeles, Rep. 63-38, Aug. 1968.
- [45] D. A. Calahan, "Linear Network Analysis and Realization Digital Computer Programs, and Instruction Manual" University of Ill. Bull., Vol. 62, Feb. 1965.
- [46] J. O. McClanahan, and S. P. Chan, "Computer Analysis of General Linear Networks Using Digraphs," International Journal of Electronics, No. 22, pp. 153-191, 1972.
- [47] S. J. Mason, " Feedback Theory - Some Properties of Signal Flow Graphs," Proceedings of the IRE, Vol. 41, pp. 1144-1156, Sept. 1953.
- [48] S. J. Mason, " Feedback Theory - Further Properties of Signal Flow Graphs," Proceedings of the IRE, Vol. 44, pp. 920-926, July 1956.
- [49] K. Singhal and J. Vlach, "Generation of immittance functions in symbolic form for lumped distributed active networks," IEEE Tran. on Circuits and Systems, Vol CAS-21, pp. 57-67, Jan. 1974.

- [50] K. Singhal and J. Vlach, "Symbolic Analysis of Analog and Digital Circuits," IEEE Tran. on Circuits and Systems, Vol CAS-24, pp. 598-609, Nov. 1977.
- [51] P. Sannuti, and N. N. Puri, "Symbolic Network Analysis - An Algebraic Formulation," IEEE Tran. on Circuits and Systems, Vol CAS-27, pp. 679-687, Aug. 1980.
- [52] T. E. Idleman, F. S. Jenkins, W. J. McCalla, and D. O. Pederson, "SLIC--A Simulator for Linear Intergrated Circuits," IEEE Journal of Solid-State Circuits, Vol. SC-6, August 1971, pp. 188-204.
- [53] F. H. Branin, G. R. Hogsett, R. L. Lunde, and L. E. Kugel, "ECAP II---A New Electronic Circuit Analysis Program," IEEE Journal of Solid-State Circuits, Vol. SC-6, August 1971, pp. 146-165.
- [54] M. M. Hassoun, "A Study of a Semi-Direct Method for Computer Analysis of Large-Scale Circuits," Master's Thesis, Purdue University, December 1984.

APPENDECES

Appendix A: SCAPP Data Structures

A.1 Subcircuits

Each vertex of the binary tree modeling the process is a discription of the subcircuit that has been partitioned at that vertex or is still there (the case of a leaf). There are only one set of branches that are never duplicated, rather they are split when a partitioning occurs. The root of the tree is pointed to by the variable *ckt_vertex*.

Table A.1: *struct subckt* (the binary tree vertex)

Type	element	Discription
int	num	Subcircuit number
int	nsize	Number of nodes
int	bsize	Number of branches
struct nlist	*nodes	Singly linked list of node structures (only used in root subcircuit in ascending order)
struct nodeptr	*sub_nodes	Singly linked list of pointers to the node structures in the list above
int	num_tn	Number of tearing nodes (external variables)
struct nodeptr	*tearing_nodes	Singly linked list of pointers to the node structures pointed to by *nodes above
struct row	*branches	Linked list of all the circuit's branches (Only used for reading in main circuit)
int	row_knt	Number of links for this subcircuit
struct row	*rows	The linked list of links of this subcircuit
int	col_knt	The number of tree branches for this subcircuit
struct row	*cols	The tree branches of this subcircuit

Table A.1 (continued)

Type	element	Discription
int	i_perm	The partitioning point in the partitioning table
struct mna_ptr	*mna	Pointer to MNA matrix rows
		(The rows then point to the matrix entries)
struct subckt	*left	Pointer to left child (subcircuit)
struct subckt	*right	Pointer to right child

A.2 Branches

Each branch of the circuit is represented by the *struct branch* structure. The reason the word row is used here is because these branches also point to the row contents of the **B** matrix.

Table A.2: *struct row* (the branch structure)

Type	element	Discription
int	flag	Set if this is a tree branch
char	type	Type of element r,c,l,e,h,g,f,o
int	num	Branch internal number
char	*branch	Branch name
struct nlist	*node1	Pointer to structure of first node
struct nlist	*node2	Pointer to structure of second node
union	extra1	
struct nlist	*cnode1	For e,h,g, or f, pointer to first control node
struct nlist	*node3	For o, pointer to opamp third node
union	extra2	
struct nlist	*cnode2	For e,h,g, or f, pointer to second control node
struct mna_ptr	*mna	Not used
struct nlist	*cvar	Pointer to current variable structure
struct row	*cbranch	Pointer to control branch
int	f	Number of nonzero entries in this B row or column
struct ptr_llist	*loc	Pointer to the nonzero entries of B
struct exp_llist	*sym_value	Pointer to linked list of the symbolic
		expression of the value of the element
double	value	Numeric value of element, 0 if purely symbolic
struct row	*next	Next element of linked list

A.3 Pointers to Branches

This is the structure of the elements of the linked list pointing to the branches of circuit. It is used in building the **B** matrix.

Table A.3: *struct ptr_llist* (pointers to branches)

Type	element	Discription
struct row	*ptr	Pointer to branch structure
struct ptr_llist	*next	Next element of linked list

A.4 Nodes

Each node of the circuit is represented by this node structure. All these nodes are linked only in the root subcircuit. The rest of the subcircuits have pointers to their nodes in the list.

Table A.4: *struct nlist* (the node structure)

Type	element	Discription
int	num	Node internal number
int	real_num	Node real number
struct ptr_llist	*branches	Linked list of pointers to the branches
		incident on this node
struct ptr_llist	*last_branch	Pointer to the end of the above list
struct nlist	*op_node1	Flags that this is a - terminal of an opamp
		and points to + terminal
struct nlist	*op_node2	Flags that this is an output terminal of an
		opamp and points to - terminal
struct nlist	*op_node3	Flags that this is a + terminal of an opamp
		and points to - terminal
struct nlist	*op_out	Pointer to the output terminal of an opamp
		if this is a - terminal
struct nlist	*next	Next element of linked list

A.5 Pointers ot Nodes

This structure is used in the subcircuits to point to their nodes in the main linked list of nodes linked in the main subcircuit data structure (*ckt_vetrex* \rightarrow *nodes*).

Table A.5: *struct nodeptr* (Pointers to nodes)

Type	element	Discription
struct nlist	*ptr	Pointer to node structure
struct nodeptr	*next	Next element of linked list

A.6 Expression List

This structure is linked in each branch and points to the symbols involved in the expression for the branch.

Table A.6: *struct exp_llist* (expression pointer)

Type	element	Discription
struct symbol_tbl	*symbol	Pointer to symbol in symbol table
struct exp_llist	*next	Next element of linked list

A.7 Symbol Table Structure

The linked list of these structures starting at *sym_table* contain all the symbols of the circuit without duplication. They are referenced by the branches by pointers to these structures.

Table A.7: *struct symbol_tbl* (symbol table element)

Type	element	Description
char	*var	The symbol
struct symbol_tbl	*next	Next element of linked list

A.8 MNA Matrix Row Pointer

This is the structure of the elements of the linked list representing the rows of the MNA or the RMNA matrices.

Table A.8: *struct mna_ptr* (MNA row)

Type	element	Discription
struct nlist	*ptr	Pointer to the node this row corresponds to
struct mna_element	*row	Linked list of row nonzero elements
struct mna_ptr	*next	Next element of linked list

A.9 MNA Matrix entry

This structure represents a nonzero entry in the MNA or RMNA entry. It is linked the row structures of the MNA (section A.8).

Table A.9: *struct mna_element* (MNA entry)

Type	element	Discription
struct nlist	*ptr	Pointer to the node this entry corresponds to,
		it represents the column this element is in
struct temp_exp	*expression	Linked list of pointers to branches
		that contribute an entry to this element
struct mna_element	*next	Next element in linked list

A.10 Expression Pointer to Branch

This structure is used to build the linked list of pointers to branches that contribute to the an entry in the MNA or RMNA matrix. This is only true for the first time the expression is manipulated. After that an internal symbol is generated to represent the expression.

Table A.10: *struct temp_exp* (pointer to branch)

Type	element	Discription
struct row	*ptr	Pointer to the branch
char	*symbol	Symbol associated with this entry
struct temp_exp	*next	Next element of linked list

Appendix B: SCAPP Users' Manual

B.1 Program Usage

NAME

SCAPP (Symbolic Circuit Analysis Program with Partitioning)

SYNOPSIS

```
scapp file1 [file2]
```

DESCRIPTION

SCAPP is a general symbolic circuit analysis program for lumped linear circuits. It takes the input deck describing the circuit from *file1* and produces the output in *file2*, if specified; otherwise the output goes to standard out. The input file is a description of the circuit elements and their interconnections. The following pages of the manual illustrate the input syntax. The output is a transfer function in the form of a collection of equations (*sequence of expressions*) in the form of a C function. The argument of the function is the symbolic elements of the circuit. The program also has a circuit partitioner. The users may choose to partition the circuit manually using the *.subckt* statement, request the program to perform automatic partitioning, or perform the analysis with no partitioning at all.

BUGS

This is an initial test version of SCAPP. The *.subckt* statment requires a blank line before it. Further testing of the program is still needed.

AUTHOR

Marwan Hassoun

B.2 Hardware Implementation

The program was developed on a VAX 11/780 VM/Unix (4.3BSD) system.

B.3 Type of Analysis

SCAPP finds the symbolic transfer functions for one or more output variables with respect to one or more input variables. The variables currently are internal nodes and external currents. Internal branch currents will be accessible in the near future. The transfer function is in the form of a C function with the arguments of the function being the symbolic variables of the circuit and the complex frequency variable s . The output could be slightly modified to make it into a FORTRAN subroutine which is more convenient for complex number evaluation purposes.

B.4 Comment Statement

* This is a comment line

A comment line is any line that starts with an asterisk. The contents of the line is completely ignored by SCAPP. An input file may contain as many comment lines as desired.

B.5 .option Statement

.option [*opt1 value*] [*opt2 value*]

The user can control the partitioning process using the *.option* command. The options available are:

- 1) *bmin*: The limit on the number of branches in each final partition.
- 2) *maintree*: Number of tree optimization iterations to perform on the initial tree, default 1.
- 3) *subtree*: Number of tree optimization iterations to perform on the trees of the subcircuits, default 1.
- 4) *auto*: 1 if automatic partitioning is requested, 0 if manual or no partitioning is desired, default 1.
- 5) *parts*: Number of parts the circuit is to be partitioned into, default 2, unless *bmin* is specified in which case that will decide the number of partitions.
- 6) *noprint*: 0 if the partitioning information and the analysis matrices are requested to standard out, 1 if no partitioning and matrix

information are desired, default 0.

- 7) *delop*: 0 if the partitioning is to treat the opamp as a *pseudo-branch* in the partitioning, and 1 if it is to be simply deleted, default 1.

B.6 .output Statement

`.output [Vk] [Vl] [Vm]`

The final output expressions are to solve for the relationship between nodes k, l and m. The output in the RMNA matrix is in terms of these output node voltages. The final transfer functions are obtained by a simple extra user manipulation of the matrix. The released version will allow a specification of the form X_{out}/X_{in} , where X is a node voltage or a branch current.

B.7 Resistor Statement

`rname node1 node2 VAR [= var_name]` *or*
`rname node1 node2 VALUE = xx.xx`

This line describes a resistor connected between nodes node1 and node2. The *name* can be any collection of characters, up to 8 of them. Care must be taken though in using special characters that C or FORTRAN do not allow, otherwise a problem will arise in the later evaluation of the output expressions. This applies to all symbolic names. VAR specifies that this is a symbolic element. The user has an option to give this symbolic resistor a different name other than *rname*, otherwise *rname* will be used as the symbol for this resistor. Specifying VALUE and a numeric value following indicates that this is a numeric resistor with that certain value. The frequency domain admittance expression for the resistor will be $\frac{1}{rname}$ or $\frac{1}{var_name}$.

B.8 Capacitor Statement

`cname node1 node2 VAR [= var_name]` *or*
`cname node1 node2 VALUE = xx.xx`

This line describes a capacitor connected between nodes node1 and node2. The user has an option to give this symbolic capacitor a different name other than *cname*, otherwise *cname* will be used as the symbol for this

capacitor. Specifying VALUE and a numeric value following indicates that this is a numeric capacitor with that certain value. The frequency domain admittance expression for the capacitor will be $s*cname$ or $s*var_name$.

B.9 Inductor Statement

$lname$ node1 node2 VAR [= var_name] or
 $lname$ node1 node2 VALUE = xx.xx

This line describes an inductor connected between nodes node1 and node2. The user has an option to give this symbolic inductor a different name other than $lname$, otherwise $lname$ will be used as the symbol for this capacitor. Specifying VALUE and a numeric value following indicates that this is a numeric inductor with that certain value. The frequency domain admittance expression for the inductor will be $\frac{1}{s*lname}$ or $\frac{1}{s*var_name}$.

B.10 VCVS Statement

$ename$ node1 node2 cnode1 cnode2 VAR [= var_name] or
 $ename$ node1 node2 cnode1 cnode2 VALUE = xx.xx

This line describes a voltage controlled voltage source connected between nodes node1 and node2 and controlled by the voltage between cnode1 and cnode2. The user has an option to give the symbolic voltage gain of the VCVS a different name other than $ename$, otherwise $ename$ will be used as the symbol for this VCVS. Specifying VALUE and a numeric value following indicates that this is a numeric gain with that certain value.

B.11 CCVS Statement

$hname$ node1 node2 cbranch VAR [= var_name] or
 $hname$ node1 node2 cbranch VALUE = xx.xx

This line describes a current controlled voltage source connected between nodes node1 and node2 and controlled by the branch current of cbranch, where cbranch is a branch name. The user has an option to give the symbolic transresistance of the CCVS a different name other than $hname$, otherwise $hname$ will be used as the symbol for this CCVS. Specifying VALUE and a numeric value following indicates that this is a

numeric transresistance with that certain value. The direction of the controlling current is always assumed from node1 of the controlling branch to node2.

B.12 VCCS Statement

gname node1 node2 cnode1 cnode2 VAR [= *var_name*] *or*
gname node1 node2 cnode1 cnode2 VALUE = xx.xx

This line describes a voltage controlled current source connected between nodes node1 and node2 and controlled by the voltage between cnode1 and cnode2. The user has an option to give the symbolic transconductance of the VCCS a different name other than *gname*, otherwise *gname* will be used as the symbol for this VCCS. Specifying VALUE and a numeric value following indicates that this is a numeric transconductance with that certain value.

B.13 CCCS Statement

fname node1 node2 cbranch VAR [= *var_name*] *or*
fname node1 node2 cbranch VALUE = xx.xx

This line describes a current controlled current source connected between nodes node1 and node2 and controlled by the branch current of cbranch, where cbranch is a branch name. The user has an option to give the symbolic current gain of the CCCS a different name other than *fname*, otherwise *fname* will be used as the symbol for this CCCS. Specifying VALUE and a numeric value following indicates that this is a numeric current gain with that certain value. The direction of the controlling current is always assumed from node1 of the controlling branch to node2.

B.14 Ideal Opamp Statement

oname node1 node2 node3

This line describes an ideal opamp with node1 being the negative terminal, node2 being the output terminal and node3 being the positive terminal. The ideal opamp has no options associated with it, it has infinite gain and a zero current entering its input terminal. Also $v_+ = v_-$.

B.15 User Defined Subcircuit

```
.subckt node1 node2 .....  
.ends
```

All the lines between *.subckt* and *.ends* statements are considered as a user defined subcircuit and are not considered in the partitioning process. The tearing nodes of this subcircuit are node1, node2, node3, etc.

B.16 Illustrations

Consider the circuit in figure (9.6). The SCAPP input syntax is listed in figure (9.8). The partitioning results are shown in figure (9.6). A sample of the output of SCAPP is shown in table (9.2).

VITA

VITA

Marwan M. Nadim Hassoun was born in Kuwait on November 1st 1962. He received his BSEE degree with high honors from South Dakota State University in Brookings in 1983. In 1984 he received his MSEE from Purdue University while working as a graduate research assistant on an International Business Machines supported project. He spent the year of 1985 as a software development engineer for Hewlett-Packard Company in Cupertino, California. In 1986, he returned to Purdue to continue his studies toward the Ph.D. degree as a graduate research assistant on a project supported by the National Science Foundation. He also was a graduate teaching assistant at Purdue in 1988.

Mr. Hassoun is a member of Phi Kappa Phi, Tau Beta Pi and Eta Kappa Nu.