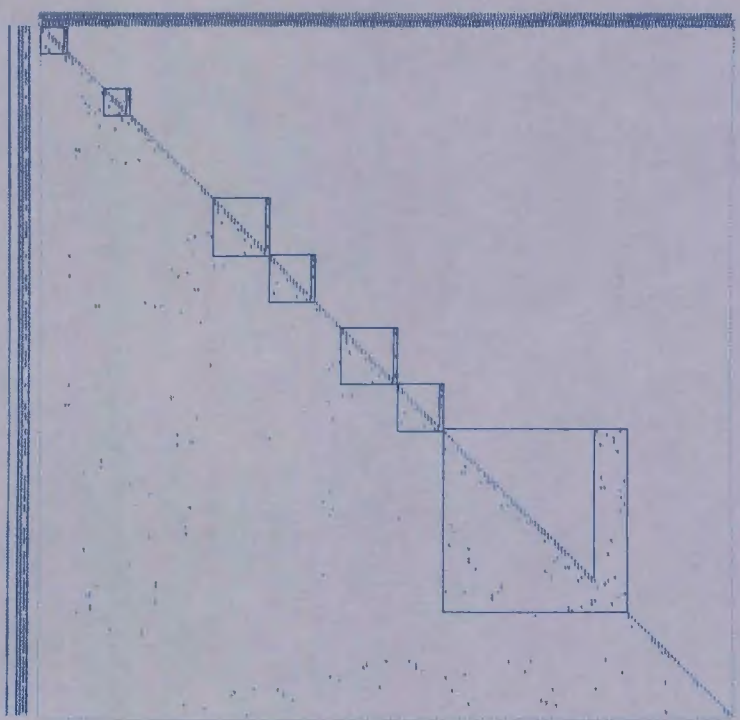
An interactive design and fault location tool for electronic circuits

AN INTERACTIVE DESIGN AND FAULT LOCATION TOOL FOR ELECTRONIC CIRCUITS



J.F.M. THEEUWEN

AN INTERACTIVE DESIGN AND FAULT LOCATION TOOL FOR ELECTRONIC CIRCUITS

AN INTERACTIVE DESIGN AND FAULT LOCATION TOOL FOR ELECTRONIC CIRCUITS

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR IN DE TECHNISCHE WETENSCHAPPEN AAN DE TECHNISCHE HOGESCHOOL EINDHOVEN, OP GEZAG VAN DE RECTOR MAGNIFICUS, PROF. DR. F.N. HOOGE, VOOR EEN COMMISSIE AANGEWEZEN DOOR HET COLLEGE VAN DEKANEN IN HET OPENBAAR TE VERDEDIGEN OP VRIJDAG 13 DECEMBER 1985 TE 14.00 UUR

DOOR

JOZEF FRANCISCUS MARIA THEEUWEN

GEBOREN TE GELEEN

Druk: Dissertatiedrukkerij Wibro, Helmond.

Dit proefschrift is goedgekeurd
door de promotoren:

Prof.Dr.-Ing. J.A.G. Jess

en

Prof.dr.ir. W.M.G. van Bokhoven

CIP-GEGEVENS KONINKLIJKE BIBLIOTHEEK, DEN HAAG

Theeuwen, Jozef Franciscus Maria

An interactive design and fault location tool for electronic circuits / Jozef Franciscus Maria Theeuwen. - [S.l. : s.n.]. - Fig.
Proefschrift Eindhoven. - Met lit. opg., reg.
ISBN 90-9001119-6
SISO 664.3 UDC 621.3.011.74 UGI 650
Trefw.: niet-lineaire netwerken; computer aided design.

CONTENTS

ABSTRACT

In this thesis a method for designing electrical nonlinear circuits, with a given topology is presented. The designer has to determine the topology of an electrical circuit, and he has to define the type of the elements in the circuit (a circuit element can for instance be a resistor, a transistor or a capacitor). Further he has to impose a number of constraints on the circuit. These constraints can prescribe values of parameters, but also voltages on nodes, currents through branches, impedances, gains and so on. The equations containing these constraints will be called "design constraint equations". The program is not only capable to handle biasing conditions of the circuit (DC behaviour), but also "small signal" behaviour, meaning the response of the circuit on small disturbances in the DC biasing point, and AC behaviour.

In the approach used, the design constraint equations are treated exactly in the same way as the equations describing the circuit structure, that is the Kirchhoff voltage and current law equations and the equations describing the behaviour of the elements (the branch constraint equations). In this way the program constructs a set of simultaneous nonlinear equations. Because it is possible that the designer imposed a number of constraints on the circuit, yielding an unsolvable system of equations, checks for solvability on the set of equations are necessary. If the designer formulated an unsolvable set of equations, these checks will reveal the source of the difficulties.

Generally the solution of a set of nonlinear equations is found by using an iterative method. One of the most popular methods is the Newton Raphson method. This method is usually applied to the whole set of equations. This however is not always necessary, especially not in sparse equation sets. By reordering the equations, large parts of the equation set can be solved by a non iterative method, saving a lot of computation time.

In this thesis we also propose a new "simulation after test" method. The basic elements of the method are already implemented in the

proposed interactive design system. This is true because the design method used, tackles conceptually the same problem as the fault location problem.

The design problem can be described as:
Given a number of design constraints, compute (all) the component values in the proposed circuit.

The fault location problem can be described as:
Given a number of measured responses of a circuit, compute (all) the component values of the circuit under test.

In addition to solving the mathematical equations associated with these two problems, also the problem to identify the most appropriate circuit entities to be measured is addressed in this thesis. We will present a constructive method to find a set of adequate measurements, taking sensitivities of components with respect to measurements into account.

1. Introduction

In electronic industry the computer has become an essential tool during the design, fabrication and testing of electronic circuits. Especially in the design of (very) large scale integrated ((V)LSI) circuits the computer has obtained a crucial role. In the last ten years many programs to verify steps during the design of a digital circuit have been developed. Examples are logic simulators, design rule checkers and circuit extractors. However also more powerful program packages have emerged. These packages help the designer during the creative steps in the development of an integrated circuit, or even take over these steps. Programs like these are optimisation programs for logic functions, floorplan programs, routers and cell generators. For certain classes of digital circuits there even exist totally automatic layout generators, often called "silicon compilers".

However in the analog field, the situation is different. There are a number of well known analog circuit simulators, such as for instance SPICE. With SPICE it is possible to simulate circuits containing about one hundred transistors, on transistor level. This program is widely used by circuit designers. SPICE has the capability to perform some parameter optimisation, but there are only a very few analog automatic synthesis tools available. For some special analog circuit families, there exist automatic circuit generation methods. Automatic filter design is an example of these. There are also programs which help the designer in developing special circuits like operational amplifiers [Nordholt]. These programs however can only be seen as a library of precompiled solutions for a large number of special cases.

In the literature some more general approaches to computer aided analog circuit design are known [Kozemchak], [Sussman], [de Kleer]. Kozemchak et. al. use a method of voltage forcing elements and current forcing elements. With these elements the user is able to define voltages and currents in the circuit. The program then computes values of elements in the circuit, in such a way that the imposed voltages and currents, are realized by the circuit.

Another approach is used by Sussman and Stallman. Here a concept of artificial intelligence is used. For each electrical rule, such as the Kirchhoff voltage law, and the Kirchhoff current law, a daemon can be created. Each time there are enough data for a daemon to compute an unknown variable in the circuit, this daemon is triggered. If no daemon can be triggered any more, and the circuit has not been solved yet, the designer has to add more information (impose restrictions on the circuit) or a symbolic variable is introduced, which is treated by the program as a known value. During the further analysis there possibly will come up an equation from which this symbolic variable can be computed. A program language like LISP is especially suited for an implementation of such a program.

The above mentioned methods have their strong and their weak points. In the approach used by Sussman et. al. it is doubtful whether this approach can solve sets of simultaneous nonlinear equations. A strong point in the method used by Kozemchak is, that the method is able to handle unequality constraints, like for instance a resistor whose value has to be between 1000 Ω and 2000 Ω. This however applies only for linear circuits.

In this thesis we present a method for designing electrical nonlinear circuits, with a given topology. The designer has to determine the topology of an electrical circuit, and he has to define the type of the elements in the circuit (a circuit element can for instance be a resistor, a transistor or a capacitor). Further he has to impose a number of constraints on the circuit. These constraints can prescribe values of parameters, but also voltages on nodes, currents through branches, impedances, gains and so on. The equations containing these constraints will be called "design constraint equations". The program is not only capable to handle biasing conditions of the circuit (DC behaviour), but also "small signal" behaviour, meaning the response of the circuit on small disturbances in the DC biasing point, and AC behaviour.

In the approach used, the design constraint equations are treated exactly in the same way as the equations describing the circuit structure, that is the Kirchhoff voltage and current law equations

and the equations describing the behaviour of the elements (the branch constraint equations). In this way the program constructs a set of simultaneous nonlinear equations. Because it is possible that the designer imposed a number of constraints on the circuit, yielding an unsolvable system of equations, checks for solvability on the set of equations are necessary. If the designer formulated an unsolvable set of equations, these checks will reveal the source of the difficulties.

Mostly the solution of a set of nonlinear equations is found by using an iterative method. One of the most popular methods is the Newton Raphson method. This method is usually applied to the whole set of equations. This however is not always necessary, especially not in sparse equation sets. By reordering the equations, large parts of the equation set can be solved by a non iterative method, saving a lot of computation time.

One very important difference between our approach and the optimisation approach used in SPICE should be made clear. In both programs iteration is used. If however parameter optimisation in SPICE is used, we can distinguish at least two levels of iterations nested into each other. The highest level is the iteration level which after each circuit evaluation generates a new, and hopefully better guess for the parameter to be optimised. The second iteration loop is the Newton Raphson iteration, trying to find a solution for a part of the circuit with the new estimated parameters. In our approach, there is only one iteration level. This is the iteration loop trying to find a solution for the system of nonlinear equations. If this solution is found, the computed circuit behaves in the way the designer has specified, and all the unknown parameters are determined.

Another crucial area in analog circuit design is testing of the circuit. Here we have to distinguish between two main testing areas:

1. Go-Nogo test.
   This kind of testing is essential during the production process of integrated circuits. After an integrated circuit has been produced, it has to be determined whether it works correctly or

not. For analog circuits this problem is mostly not too difficult. For filters and amplifiers for instance it is easy to check whether the transfer characteristics of the circuits are correct.

2. Another much more difficult kind of testing is "fault location". Now one tries to find out which elements in a circuit are correct and which elements are faulty. This problem is equivalent with determining all the parameter values in a circuit from a number of measurements.

   This kind of testing is important during the development of an integrated circuit. If a circuit doesn't work well, a designer wants to know which parameters in the circuit have wrong values. Also in the maintenance field this kind of testing is crucial.

   Because of the ever increasing complexity of integrated circuits, the task of fault location becomes very difficult. This because of the fact that the "accessibility" of a large integrated circuit is very bad. The number of elements in the circuit is large and the number of points where measurements can be performed is very small.

   The approaches used in fault location can be divided in two main groups:

   1. Simulation before test.
      In this method all the possible faulty circuits are simulated, and the results of these simulations are grouped in one or another way, constructing a so called "fault dictionary" [Lin]. If in the field, a faulty circuit is encountered its responses are compared with the results stored in the dictionary, this way trying to find the faulty element.
      The size of the dictionary and the simulation time will be in the order of $O(n^m)$, where n equals the number of elements in the circuit, and m is equal to the number of elements which are allowed to be faulty in one circuit

simultaneously. Mostly the assumption is made that only one element is faulty, thuswise reducing the size of the dictionary.

2. Simulation after test.
   In simulation after test we try to compute all the elements in the circuit. This is done by using the measurements made on the faulty circuit [Biernacki] [Bedrosian] [Salama] [Liu] [Duhamel] [Trick] [Lee] [Saeks]. Also now the assumption that only one element is faulty is often made. By simulation after test the required on line computation power is larger than in the "dictionary method".

In this thesis we propose a new "simulation after test" method. The basic elements of the method are already implemented in the proposed interactive design system. This is true because the design method used tackles conceptually the same problem as the fault location method.

The design problem can be described as:
Given a number of design constraints, compute (all) the component values in the proposed circuit.

The fault location problem can be described as:
Given a number of measured responses of a circuit, compute (all) the component values of the circuit under test.

In addition to solving the mathematical equations associated with these two problems, also the problem to identify the most appropriate circuit entities to be measured is addressed in this thesis. We will present a constructive method to find a set of adequate measurements, taking sensitivities of components with respect to measurements into account.

2. The formulation of the design problem

The formulation of a design problem requires the description of the associated circuit structure and the description of the design objectives. This design problem has to be stored in the program. We will call this the "internal design problem representation". On one hand it must be possible to generate this internal design problem representation from a description in terms of an input language used by the designer to specify the circuit and the design problem. On the other hand the internal design problem representation must be such that it is easy for the program to deal with it. The description of the network structure consists of the Kirchhoff voltage law equations, the Kirchhoff current law equations and the the branch constraint equations. These will constitute a "sparse tableau" [Hachtel].
In this chapter we will define all the equations necessary to describe the circuit. It will be done for the DC behaviour, the behaviour of the circuit linearised around a certain DC operation point (called the small signal behaviour) and for the behaviour of the circuit if it is excited by a sinusoidal source at a given frequency, called the AC behaviour.

2.1 The DC behaviour

In the sequel we will use the term "circuit variable" and "circuit parameter" (or simply : "variable" and "parameter"). By way of definition they will be assigned the following meaning:

1. A parameter is any entity characterising a circuit element such as resistances, gains or transimpedances.

2. A variable is any voltage or current in the circuit as well as any partial derivative of a voltage or current with respect to a parameter or variable. Also in case some transposed small signal system (the so called "adjoint system"; a more formal definition of this term will be given later) is included, the response entities of this system are considered as variables.

The parameters and variables may be "known" or "unknown".

The topology of a circuit can be represented by a directed graph. Each element in the circuit is represented by a branch. This does not mean that no multi port elements can be incorporated in the circuit. There is a possibility to define controlled voltage or current sources. The controlling variables can be voltages or currents. Multi terminal elements such as transistors can be modeled with a number of two terminal elements. For a bipolar transistor for instance the Ebers Moll model [Taub] can be used. Each node in the graph corresponds with a node in the electrical circuit. For the DC-description of the circuit with k nodes and l two terminal elements we can distinguish three types of variables:

1. k node voltages represented by the vector
   $\underline{n} = (n_1,\ldots,n_k)^t$
2. l branch currents represented by the vector
   $\underline{i} = (i_1,\ldots,i_l)^t$
3. l branch voltages represented by the vector
   $\underline{v} = (v_1,\ldots,v_l)^t$

Further we define $\underline{p} = (p_1,\ldots,p_l)^t$ as the vector of parameters, and $\underline{b} = (b_1,\ldots,b_l)^t$ as the vector of excitations, whose meaning will be explained in the next chapters. Most entries of this vector $\underline{b}$ will be equal to zero. For each element in the circuit we can define a Kirchoff voltage law equation

$$v_j = n_x - n_y \tag{2.1}$$

where x and y denote the nodes connected with the element,
and a branch constraint :

$$f_j(v_j , i_j , p_j) = 0 \tag{2.2}$$

A branch constraint equation is the equation describing the behaviour of the element and is mostly a relation between the current i through the element, the voltage v across the element and one parameter p describing the element. That each element is described by only one parameter is not essential at all. It is possible to introduce elements described by more than one parameter, this however is not done in this implementation, resulting in an equal number of elements

and parameters. For each node in the circuit we can formulate the Kirchhoff current law equation:

$$\sum_j i_j = 0 \qquad (2.3)$$

Where j enumerates all the branches incident with node n. The total number of unknowns is :

1	parameters
k-1	node voltages
1	branch voltages
1	branch currents
+	
total : $3l + k - 1$	unknowns.

The number of equations is :

1	Kirchoff voltage law equations.
1	branch constraint equations.
k-1	Kirchoff current law equations.
+	
total : $2l + k$-1	equations.

To obtain a solvable set of equations we need

$$3l + k\text{-}1 - (2l + k\text{-}1) = l$$

design constraint equations, which can be functions of $\underline{n}$, $\underline{i}$, $\underline{v}$ and $\underline{p}$. If we define $\underline{x} = (\underline{n}, \underline{i}, \underline{v})^t$, the system of all Kirchoff voltage law equations, Kirchoff current law equations and the branch constraint equations can be described with:

$$\underline{F}(\underline{x}, \underline{p}, \underline{b}) = \underline{0} \qquad (2.4)$$

$\underline{b}$ being the vector of excitations. $\underline{F}$ is called the "tableau operator". An example circuit is shown in figure (2.1)

The tableau operator $\underline{F}$, linearized around the operation point of the circuit, can be viewed as the product of the tableau matrix with the vector $\underline{x}$ of unknowns added to the vector $\underline{b}$.

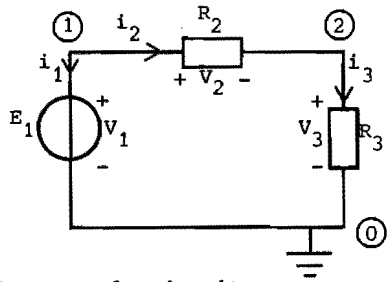


Figure 2.1. example circuit

$$\underline{F}(\underline{x}, \underline{p}, \underline{b}) = \begin{vmatrix} & \\ & \end{vmatrix} * \begin{vmatrix} \underline{x} \\ \underline{p} \end{vmatrix} + \underline{b} = \underline{0}$$

↑

tableau matrix

For the example in figure 2.1 $\underline{F}$ looks as follows:

$$\begin{array}{c|cccccccccc|c|c|c|c}
 & V_1 & N_1 & V_2 & N_2 & V_3 & I_3 & I_2 & I_1 & & & & \\
\hline
KC\ 1 & & & & & & 1 & 1 & & & V_1 & & 0 \\
KC\ 2 & & & & & 1 & -1 & & & & N_1 & & 0 \\
BC\ 3 & & & & 1 & -R_3 & & & & & V_2 & & 0 \\
KV\ 3 & & & 1 & -1 & & & & & * & N_2 & + & 0 & = \underline{0} \\
KV\ 2 & & 1 & -1 & -1 & & & & & & V_3 & & 0 \\
BC\ 2 & & & 1 & & & -R_2 & & & & I_3 & & 0 \\
KV\ 1 & -1 & 1 & & & & & & & & I_2 & & 0 \\
BC\ 1 & 1 & & & & & & & & & I_1 & & -E_1
\end{array}$$

KV: Kirchhoff voltage law equation.

KC: Kirchhoff current law equation.

BC: Branch constraint equation.

2.2 Small signal specification

The small signal behaviour of the circuit describes the response of the circuit to a "small" excitation. "Small" means that the circuit can be substituted by a linear circuit, obtained by linearisation around the DC bias point. The excitation can be a small disturbance

of a voltage, a current or parameter. The responding variables are elements of $\underline{x}$. If a small signal specification is used, it is possible to compute gains or sensitivities in the circuit.

To obtain the equations describing the small signal behaviour of the circuit we expand (2.4) into a Taylor series, around the point ($\underline{x}_0$, $\underline{p}_0$, $\underline{b}_0$), defining the DC biasing point of the circuit.

$$\underline{F}(\underline{x}_0,\underline{p}_0,\underline{b}_0) + \left.\frac{\partial \underline{F}}{\partial \underline{x}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{x} + \left.\frac{\partial \underline{F}}{\partial \underline{p}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{p} + \left.\frac{\partial \underline{F}}{\partial \underline{b}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{b} = \underline{0} \qquad (2.5)$$

with $\Delta\underline{b}$ the vector of excitations of the small signal systems in case of exciting voltages or currents.
These equations can be catenated to (2.4). The vectors $\Delta\underline{x}$ and $\Delta\underline{p}$ contain new variables, the small signal variables. The meaning of $\Delta\underline{b}$ will become clear in the next chapters. For each exciting quantity, we now will discuss the way it is modelled and described.

0.1 Exciting voltage

If a designer is interested in the gain of an amplifier, he wants to add a small perturbation to the input voltage source of the amplifier to compute the response of various voltages and currents in the circuit. Suppose v_j is the excitation voltage. We can model this as in Fig.2.2.

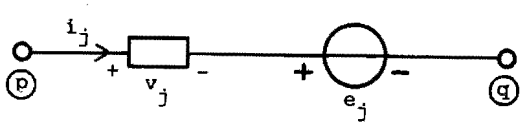


Figure 2.2. exciting voltage

In series with the element j we insert an exciting voltage source. By doing so the Kirchoff current law equations and the branch constraint equations will not change. The only equation which changes is the Kirchoff voltage law equations of branch j

$$v_j + e_j = n_p - n_q \qquad (2.6)$$

This can be incorporated in the description by making b_j equal to e_j. (All other elements of $\underline{b}$ are zero). Because no parameters are

exciting the system $\Delta\underline{p} = \underline{0}$ holds.

Also

$\underline{F}(\underline{x}_0, \underline{p}_0, \underline{b}_0) = \underline{0}$ holds because the circuit is in its DC biasing point.

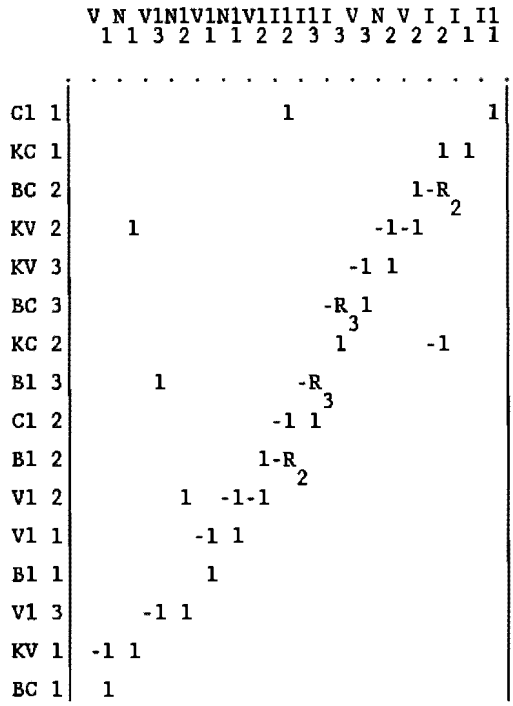
So in this case (2.5) will yield:

$$\left.\frac{\partial \underline{F}}{\partial \underline{x}}\right|_{\underline{x}_0, \underline{p}_0, \underline{b}_0} * \Delta\underline{x} + \left.\frac{\partial \underline{F}}{\partial \underline{b}}\right|_{\underline{x}_0, \underline{p}_0, \underline{b}_0} * \Delta\underline{b} = \underline{0} \qquad (2.7)$$

Notice that in the second term of (2.7) only the partial derivative of the Kirchoff voltage law equations of element "j" will result in a non zero value.

Assume the term $\frac{\partial v_{03}}{\partial v_{01}}$

is used in a design constraint equation. In other words, one design objective of the network in fig 2.1. concerns the small signal behaviour. Combining the DC and small signal equations into one equation system yields (for the sample circuit of fig. 2.1) after some reordering the tableau matrix below.



KV: Kirchhoff voltage law equation.

KC: Kirchhoff current law equation.

BC: Branch constraint equation.

Vi: Kirchhoff voltage law equation of "delta system" *i*. (for the definition of the "delta system" see chapter 2.6).

Ci: Kirchhoff current law equation of "delta system" *i*.

Bi: Branch constraint equation of "delta system" *i*.

2.4 Exciting current

If i_j is the exciting current we model this as in Fig. 2.3. In parallel with element *j*, we insert a current source. This does not affect the Kirchoff voltage law equation and the branch constraint equations. Only the Kirchoff current law equations of node *p* and

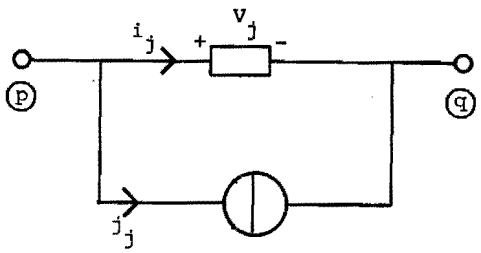


Figure 2.3. exciting current

node q will change.

node p : $\Sigma i_p + j_j = 0$

node q : $\Sigma i_q - j_j = 0$

Again we will make element b_j equal to j_j , and $\Delta\underline{p}$ will be zero, so again we obtain

$$\left.\frac{\partial \underline{F}}{\partial \underline{x}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{x} + \left.\frac{\partial \underline{F}}{\partial \underline{b}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{b} = \underline{0} \qquad (2.8)$$

$(\underline{F}(\underline{x}_0,\underline{p}_0,\underline{b}_0) = \underline{0})$

Only the partial derivatives of the Kirchoff current law equations of node p and node q will result in a nonzero value in the second term of equation 2.8.

2.5 Exciting parameters

If a designer wants to know how the circuit reacts on changes of the value of a parameter, for instance a resistor value, or the temperature, this can be modelled by an exciting parameter. If p_j is the exciting parameter of branch j, we make the j-th element of $\underline{p}$ equal to the exciting value.
So $\Delta\underline{p} \neq \underline{0}$, and $\Delta\underline{b} = \underline{0}$ thus (2.5) will result in :

$$\left.\frac{\partial \underline{F}}{\partial \underline{x}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{x} + \left.\frac{\partial \underline{F}}{\partial \underline{p}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{p} = \underline{0} \qquad (2.9)$$

$(\underline{F}(\underline{x}_0,\underline{p}_0,\underline{b}_0) = \underline{0})$

Parameter p_j appears in branch constraint equation j, so the partial derivative of this equation will yield a nonzero value. For instance

$$\frac{\partial}{\partial R}(v - i * R) = -i$$

or

$$\frac{\partial}{\partial T}(i-I_0(\exp(qv/(kT)-1))=I_0(qv/(kT^2))*\exp(qv/(kT))$$

So in the second term of equation (2.9) the branch constraint equations of parameter j will result in a non zero value.
If the term $\frac{\partial v_{03}}{\partial R_{02}}$

is used in a design constraint equation the tableau matrix will look as follows:

	V N V1	N1	V1	N1	V1	I1	I1	V N V	I	I	I	I1
	1 1 3	2	1	1	2	2	3	2 2 3	3	2	1	1
C1 1					1							1
KC 1										1	1	
KC 2									1	-1		
BC 3									1	-R_3		
KV 3								1	-1			
KV 2	1						-1	-1				
BC 2							1			-R_2		
B1 3		1				-R_3						
C1 2						-1	1					
B1 2					1-R_2					-1		
V1 2			-1		1-1							
V1 1				-1	1							
B1 1				1								
V1 3			-1	1								
KV 1	-1	1										
BC 1	1											

Equation B1 2 reads: V1 2 - R2 * I1 2 - I 2 = 0.

2.6 The implementation of the small signal system in the tableau

There are two ways to incorporate the small signal description, or incremental system [Desoer] of the circuit, in the tableau. The first method is what we will call the use of the "delta system". Consider "s" as a sensitivity of variable x_i for variations of y_j.

$$s = \frac{\Delta x_i}{\Delta y_j}$$

We can interpret Δx_i as a responding variable with Δy_j as exciting quantity, and add to the existing system of equations

$$\underline{F}(\underline{x}_0, \underline{p}_0, \underline{b}_0) = \underline{0}$$

the expression

$$\left.\frac{\partial \underline{F}}{\partial \underline{x}}\right|_{\underline{x}_0, \underline{p}_0, \underline{b}_0} * \Delta \underline{x}$$

together with

$$\left.\frac{\partial \underline{F}}{\partial \underline{p}}\right|_{\underline{x}_0, \underline{p}_0, \underline{b}_0} * \Delta \underline{p}$$

or

$$\left.\frac{\partial \underline{F}}{\partial \underline{b}}\right|_{\underline{x}_0, \underline{p}_0, \underline{b}_0} * \Delta \underline{b}$$

as an exciting source vector. This method is used in the examples in the previous paragraphs.
A second way to incorporate the small signal description is to use a "transposed small signal system" (sometimes called an "adjoint system") [Hachtel]. Assume we declare Δx_i to be an output variable. Then there are two possibilities :

a The exciting entity is a parameter,

b The exciting entity is a voltage or a current,

Ad a)
As derived in a previous chapter, the equation

$$\left.\frac{\partial F}{\partial \underline{x}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{x} + \left.\frac{\partial F}{\partial \underline{p}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{p} = \underline{0} \qquad (2.10)$$

holds.

Let $\underline{w}$ be a vector implicitly defined by :

(2.11)

$$\underline{w}^t * \left.\frac{\partial F}{\partial \underline{x}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} = \underline{e}_i^t \quad \text{with } \underline{e}_i^t = (0,0,\ldots,0,1,0,..,0)$$

$\uparrow$ entry i

or

$$\left(\frac{\partial F}{\partial \underline{x}}\right)^t \Big|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \underline{w} = \underline{e}_i \qquad (2.12)$$

So

$$\underline{w}^t * \left.\frac{\partial F}{\partial \underline{x}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{x} = \Delta x_i \qquad (2.13)$$

If we multiply (2.10) from the left side with $\underline{w}^t$ we obtain :

$$\Delta x_i + \underline{w}^t * \left.\frac{\partial F}{\partial \underline{p}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{p} = \underline{0} \qquad (2.14)$$

or

$$\left[\left.\frac{\partial F}{\partial \underline{p}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{p}\right]^t * \underline{w} = -\Delta x_i \qquad (2.15)$$

Because the quantities Δp_j are taken unequal to zero one at a time we obtain :

$$\left[\left.\frac{\partial F}{\partial \underline{p}}\right|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} \left|\begin{matrix}0\\ \vdots\\ 0\\ 1\\ 0\\ \vdots\\ 0\end{matrix}\right.\right]^t * \underline{w} = -\frac{\Delta x_i}{\Delta p_j} \qquad (2.16)$$

at place j

Now we have two sets of equations : (2.12) and (2.16). The advantage

of using an adjoint system is that if we want to know the sensitivities of one variable in respect to a number of parameters, only one adjoint system has to be considered. For each parameter, equation 2.16 (which really is only one equation) has to be catenated to the system (2.12) of equations.
(2.12) is a set of $2l + k - 1$ equations and can be catenated to the existing set of equations.
The matrix

$$\left[\frac{\partial \underline{F}}{\partial \underline{x}}\right]^t \Big|_{\underline{x}_0, \underline{p}_0, \underline{b}_0}$$

is a square matrix.
Consider

$$\frac{\partial f_r}{\partial x_s}$$

We can distinguish two cases:

1. f_r is a topological equation, i.e. a Kirchhoff voltage law equation or a Kirchhoff current law equation.

2. f_r is a branch constraint equation.

ad1. $\frac{\partial f_r}{\partial x_s} = \pm 1$ if x_s occurs in f_r

$\frac{\partial f_r}{\partial x_s} = 0$ if x_s does not occur in f_r.

ad2. The result depends on the type of the element described by f_r. For a resistor

$\frac{\partial f_r}{\partial v_s}$ results in + 1

$\frac{\partial f_r}{\partial i_s}$ results in - R_r

(2.16) is one equation, and can be written as :

$$-\frac{\Delta x_i}{\Delta p_j}\left[\frac{\partial f_1}{\partial p_j},\frac{\partial f_2}{\partial p_j},\ldots\ldots,\frac{\partial f_{2l+k-1}}{\partial p_j}\right] * \begin{bmatrix} w_1 \\ w_2 \\ \cdot \\ \cdot \\ w_{2l+k-1} \end{bmatrix} \qquad (2.17)$$

We normalise Δp_j to 1. The partial derivatives

$$\frac{\partial f_1}{\partial p_j},\frac{\partial f_2}{\partial p_j},\ldots\ldots,\frac{\partial f_{2l+k-1}}{\partial p_j}$$

can be evaluated and the row vector

$$\left[\frac{\partial f_1}{\partial p_j},\frac{\partial f_2}{\partial p_j},\ldots\ldots,\frac{\partial f_{2l+k-1}}{\partial p_j}\right]$$

can be catenated to the tableau.

Ad b)
As derived earlier we start with (2.8)

$$\frac{\partial \underline{F}}{\partial \underline{x}}\Big|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{x} + \frac{\partial \underline{F}}{\partial \underline{b}}\Big|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{b} = \underline{0}$$

From this equation again two systems of equations can be derived, which can be catenated to the existing set of equations.
Equation (2.15) will now be different:

$$\left[\frac{\partial \underline{F}}{\partial \underline{b}}\Big|_{\underline{x}_0,\underline{p}_0,\underline{b}_0} * \Delta\underline{b}\right]^t * \underline{w} = -\Delta x_i \qquad (2.18)$$

In the matrix

$$\frac{\partial \underline{F}}{\partial \underline{b}}\Big|_{\underline{x}_0,\underline{p}_0,\underline{b}_0}$$

the term $\frac{\partial f_r}{\partial b_s}$ will only result in a nonzero value in the Kirchhoff voltage law equation f_r of element s if b_s is a voltage, or in the Kirchhoff current law equation of the nodes in which b_s occurs if b_s

is a current. Thus we have two cases:

$$\left[\frac{\partial f_1}{\partial p_j}, \frac{\partial f_2}{\partial p_j}, \ldots\ldots, \frac{\partial f_{2l+k-1}}{\partial p_j}\right] * \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ \vdots \\ w_{2l+k-1} \end{bmatrix} = (0,0,..0,\underset{\uparrow}{1}_{r},0,...0) \qquad (2.19)$$

(here f_r is the Kirchhoff voltage law equation of element r)
or

$$\left[\frac{\partial f_1}{\partial p_j}, \frac{\partial f_2}{\partial p_j}, \ldots\ldots, \frac{\partial f_{2l+k-1}}{\partial p_j}\right] * \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ \vdots \\ w_{2l+k-1} \end{bmatrix} = (0,0,..0,\underset{\uparrow}{1}_{p},0,..,0,\underset{\uparrow}{1}_{q},0,..,0) \qquad (2.20)$$

(here f_p is the Kirchhoff current law equation of node p and f_q is the Kirchhoff current law equation of node q).

2.7 The AC description

To describe the AC behaviour of the circuit, we introduce a frequency ω and complex variables for all variables and parameters in the circuit [Hostetter]. For each frequency point we introduce a separate set of equations, describing the circuit at that frequency. For each frequency there is a possibility to incorporate a delta system or a transposed small signal system.

2.8 The circuit elements

The description of the circuit is totally general, so there are really no restrictions with respect to the type of circuit elements that can be used. In the implementation the following circuit elements have been included:

	DC	AC
Resistor	$v = R * i$	$v = R * i$
Diode	$i = I_0(\exp(qv/(kT)) - 1)$	$i = I_0 (\exp(qv/(kT))-1)$
Voltage source	$v = E$	$v = X$
Circuit source	$i = J$	$i = Y$
Capacitor	delete element	$i = (j\omega C) * v$
Inductor	short circuit	$v = j\omega L * i$
Dependent current source	$i = J * x$	$i = Y * x$
Dependent voltage	$v = E * x$	$v = X * x$

For both the pnp and the npn bipolar transistor we have the Ebers Moll model [Taub].The models are composed from diodes and dependent current sources, for instance the npn full model:

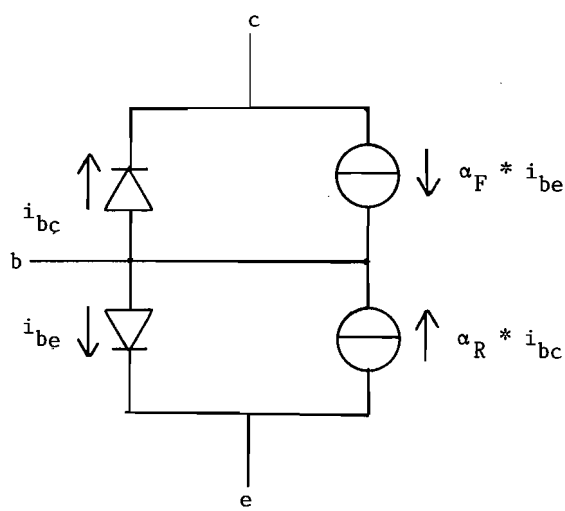


Figure 2.4. Static Ebers Moll model

2.9 Design constraints

To obtain a solvable set of equations, we need *1* design constraint equations. In these design constraint equations we can formulate relations between all unknowns in the circuit, not only with branch currents and voltages but also with circuit parameters. A resistance can be given a value by:

$R_2 = 3000\ \Omega$

A nonlinear resistor however, can for instance be defined as :

$R_2 = 3000\ \Omega - 10\ \Omega * i_2/A$,

If the temperature depends on the power consumption we obtain:

$T\ K = 300\ K + 100\ K * v_{(power\ supply)} * i_{(power\ supply)}/VA.$

3. Incidence matrix and matching

In chapter 2 we described all equations necessary to formulate the design problem of a circuit. The task of the program system is to determine the solution of this set of equations if there exists one. If no solution exists, the program has to point out, why the set of equations is not solvable. Here we have to distinguish two kinds of solvability

1. A set of equations can be unsolvable because there are no values for the unknown variables, satisfying all the equations, for instance:

$$\begin{aligned} x + y &= 4 \\ x + y &= 2 \end{aligned} \tag{3.1}$$

2. A set of equations can be structural unsolvable, for instance:

$$\begin{aligned} w + x + y + z &= 3 \\ w + x &= 4 \\ w + 3x &= 9 \\ w &= 8 \end{aligned} \tag{3.2}$$

   Now there are some unknown variables which are "over determined" and there are some variables for which no value can be computed at all.

One of the essential differences between the two kinds of unsolvability is that the structural unsolvability can be determined from the way the variables appear in the equations.
This can be seen in an incidence matrix. An incidence matrix is a matrix where a "1" appears in entry i,j if the variable, related with column j of the matrix, appears in the equation, related with a row i of the matrix (for a formal definition see chapter 3.1.) If we determine the incidence matrix from the equations in (3.2), this will look as follows:

	w	x	y	z
1	1	1	1	1
2	1	1		
3	1	1		
4	1			

Figure 3.1. incidence matrix

Form this incidence matrix we can see that w can be determined with equation number 4. Then however, equation number 2 and 3 are two equations with only one unknown variable, and equation number 1 is an equation with at least two unknown variables. Except for some special cases this set of equations will not be solvable.
From the incidence matrix of the equations in (3.1) we can not conclude that this set of equations is not solvable. The incidence matrix tells us that we have two equations with two unknown variables, and in general these equations will be solvable, except for some special cases.

During a design process of an analog circuit, the structural unsolvability occurs (roughly speaking), if a designer poses more than one constraint on a parameter. If for instance the design constraint equations of the circuit in figure (3.2) are:

$$E_1 = 5 \qquad (3.3)$$
$$i_3 = 0.001$$
$$v_1 * i_1 = -0.2$$

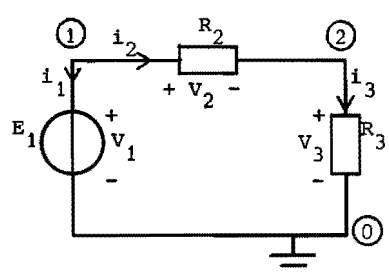


Figure 3.2. example circuit

the incidence matrix shown in fig.(3.3) can be constructed:

	I 0 3	I 0 2	I 0 1	V 0 1	N 0 1	E 0 1	N 0 2	R 0 2	V 0 3	R 0 3	V 0 2
BC 3	1								1	1	
KV 3							1		1		
BC 2		1						1			1
KV 2					1		1				1
BC 1				1		1					
KV 1				1	1						
DC 3			1	1							
KC 1		1	1								
KC 2	1	1									
DC 2	1										
DC 1						1					

Figure 3.3. incidence matrix of example circuit

If we look at the last seven equations we can see that these equations are incident with only 6 (the leftmost 6) variables, indicating a structural unsolvable set of equations.

The incidence matrix not only tells the designer which variables are "over determined" (one of the variables: i_3, i_2, i_1, v_1, n_1) , but also which variables can not be determined at all (E_1, n_2, R_2, v_3, R_3, v_2), giving the designer an indication how to reformulate his design problem.

To find out whether a set of equations is solvable, the notion of a "matching" is introduced (for a formal definition of a matching see chapter 3.1). A matching relates one variable to each equation, occurring in that equation. Each equation may only be related with one variable, and each variable may only be related with one equation. If it is possible to relate each variable to an equation we say that we have found a "complete matching", indicating that the set of equations is structurally solvable.

If a set of equations is structural solvable, the incidence matrix can tell us something about the way we have to solve the equations. If the incidence matrix for instance has the form of an upper triangular matrix, the set of equations can be solved by back substitution, firstly solving variable 1 with equation number n, then solving variable 2 with equation n-1 and so on (see fig. (3.4)).

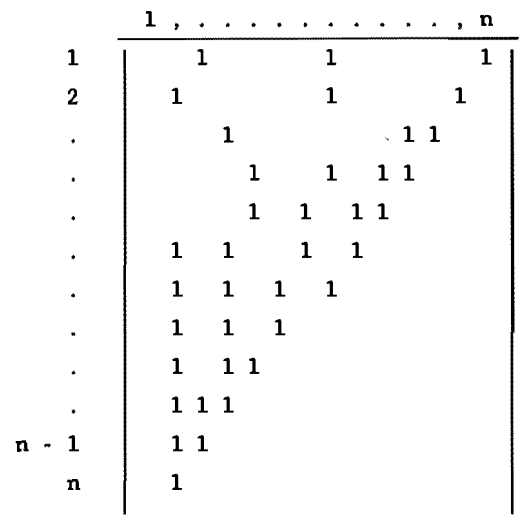


Figure 3.4. upper triangular form

Mostly however it is not possible to obtain a triangular form by row and column permutations and some equations have to be solved with an iteration scheme (or with elimination if all the equations are linear). In the majority of the applications the iteration is done with the full set of equations. This however is not always necessary. Mostly iteration is only needed in a small subset of the equations, while other parts of the equation set can be solved by back substitution, resulting in a much more effective way of solving the equations. The groups of equations which have to be solved by iteration are related with so called "strong components" (for an indication of what a strong component is see fig. (3.5)), to be found by manipulations with the incidence matrix. This will be explained in the next chapters.

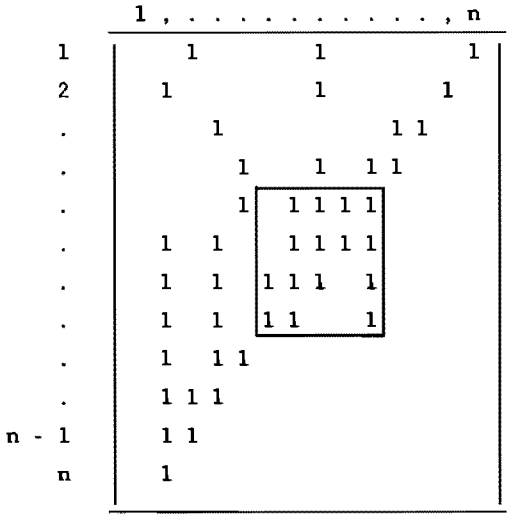


Figure 3.5. incidence matrix with strong component

3.1 Definitions

From the equations defined in chapter 2 we are able to derive a square incidence matrix with $3l + k - 1$ rows and columns.
This incidence matrix is defined as follows:

- Each equation is related to a row in the incidence matrix;
- Each variable and each parameter is related to a column in the incidence matrix;
- If a variable or a parameter j appears in equation i, entry i,j in the incidence matrix is '1';
- If a variable or a parameter j does not appear in equation i, entry i,j in the incidence matrix is '0'.

From the incidence matrix we can derive a graph G(V,B) called the incidence graph. V is the set of vertices, and B is the set of branches in the graph. For each row (equation) in the incidence matrix , there will be a vertex $v \in F \subset V$. Also for each column (variable) in the incidence matrix, there will be a vertex $v \in X \subset V$. F and X establish a partition on V, thus:

$$V = F \cup X. \tag{3.4}$$

and

$$F \cap X = \emptyset \tag{3.5}$$

A branch $b \in B$ between the vertices $v \in F$ and $w \in X$, $b[v,w]$, exists if the variable related to w is incident with the equation related to v.

Note: This graph is another graph as the graph derived from the circuit topology in chapter 2.1.

Because of this construction a branch always connects a vertex $v \in F$ with a vertex $w \in X$, resulting in a bipartite graph [Harary]. In such a graph a matching can be defined:

A matching M is a subset of the set of branches B such that there is no node which is incident with more than one branch in the matching M.

Determination of the (maximum) matching induces a direction on the branches from v to w where $v \in X$ (related to the variables) and $w \in F$ (related to the equations), if $b(v,w>$ is in the matching and from v to w where $v \in F$ and $w \in X$, if $b(v,w>$ is not in the matching. A directed path P $v \Rightarrow w$, from v to w in G is a sequence of directed branches leading from v to w.

A strong (sub)graph is a directed graph in which every node can be reached from every other node through a directed path. A strong component in a directed graph is a maximally[1] strong subgraph.

We agree that each node can reach itself, so it is possible to have strong components consisting of only one node.

3.2 Canonical form of the incidence matrix

When looking at the incidence matrix, we can find two important properties in the set of equations. Firstly we can determine whether the set of equations is structural solvable or not. Secondly we can find out in which order the equations have to be solved to minimise the number of equations participating in an iteration process. In order to determine this information we shall have to transform the

1. A strong subgraph is maximal if it is not properly contained in any other strong subgraph.

incidence matrix into a canonical pattern like the one in Fig. (3.6) [Dulmage, 1958] [Johnson] [Dulmage, 1963] by row and column permutations.

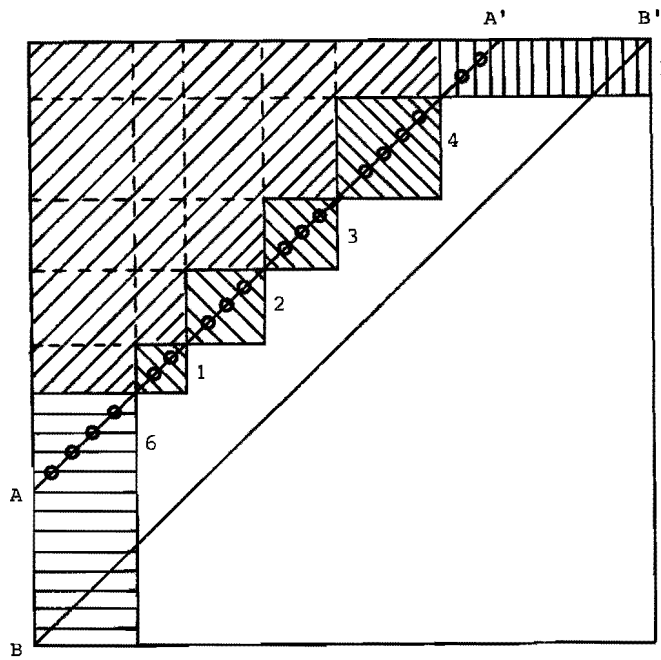


Figure 3.6. canonical pattern

Only the shaded areas in the pattern may contain nonzero entries, while white areas contain only zero entries. Define a " 45-degree line" in the matrix A as the entries: $A_{i,n-i+1-k}$

where: n is the dimension of the matrix.

and: $k = 0,\ldots,n - 1$

and: $i = 1,\ldots,n - k$

For k equal to 0 the "45-degree" line is the diagonal of the matrix. The authors mentioned above have proved that the pattern can be made to have the following properties:

1. The 45-degree line A - A' has all 1-entries. These entries correspond with the matching with maximum cardinality in the graph related to the incidence matrix.

2. There is no possibility to shift the line A - A' towards the line B - B' (decrease k, while maintaining the 45-degree slope)

by row and columns permutations, such that the new line also has all 1-entries.(so the matching has maximum cardinality).

3. The square arrays (like the shaded ones 1,2,3,4) are "irreducible" in the sense that it is impossible to break them down into smaller square arrays by row and column permutations only.

Under these circumstances the irreducible arrays are unique in the sense that the rows and columns involved per array are uniquely determined. This is true although the set of entries along the line A - A' and the sequence of the irreducible arrays along this line are not unique. The arrays 5 and 6 are called the "tails" of the scheme. In particular 5 is the "horizontal" and 6 the "vertical" tail. If the scheme is square and A - A' is the diagonal, there are no tails, and the "maximum matching" is called a "complete matching". If we can find a complete matching we can conclude that the system of equations is structurally solvable.

As stated in the introduction the order of solving the equations is a kind of back substitution. The equations $f_p, f_{p-1}, \ldots, f_k$ can be solved in that way (fig 3.7).

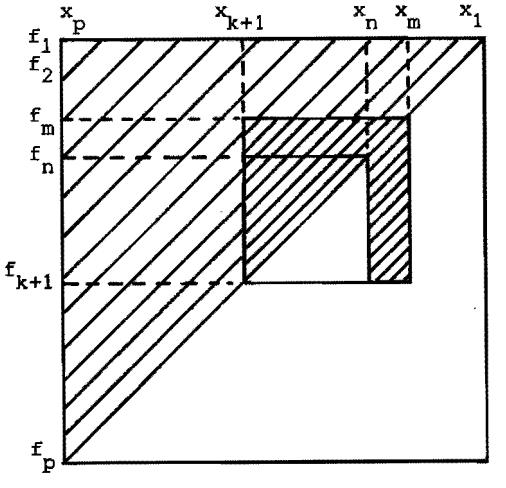


Figure 3.7. matrix with strong component

If an irreducible array containing more than one equation is encountered however, this way of solving the equations is not

possible any more. The equations associated with an irreducible array have to be solved simultaneously.

3.3 Determination of the maximum matching

In this paragraph it will be explained how the matching and the irreducible arrays can be determined.
To be able to explain how this can be done some definitions are necessary.

1. A free node is a node that is not incident with any branch in the matching M.
2. A path is a set of branches alternating in M and B-M where the end node of a branch is the same node as the start node of the next branch. (In a branch b(v,w> is v the start node and w the end node).
3. An augmenting path is a path starting in a free node in F and ending in a free node in X.
4. Two paths, P_1 and P_2, are disjoint if P_1 and P_2 have no node in common.
5. The number of branches in a matching M is denoted by $|M|$.
6. The number of branches in a path P is denoted by $|P|$.
7. Each bipartite graph has a maximum matching M_{max}, with the property that there exists no other matching M with $|M| > |M_{max}|$.
8. There may be more than one maximum matching.

Now we shall state a number of properties, for their proofs we refer to [Hopcroft].

- For each matching M_i with $|M_i| < |M_{max}|$ holds that there exists at least one augmenting path relative to M_i.
- If M is a matching and P is an augmenting path relative to M, then $M \oplus P$ is a matching and $|M \oplus P| = |M| + 1$.
  With $\oplus$ we denote the "EXCLUSIVE OR" operation, so $M \oplus P$ yields the branches which are in M and not in P together with the

branches which are in P and not in M (see fig 3.8).

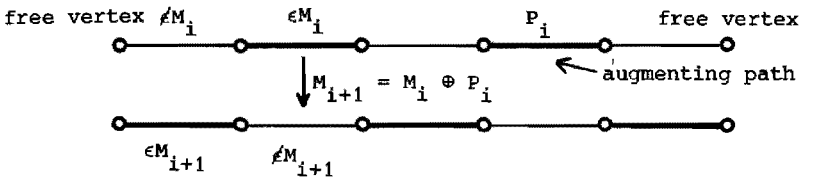


Figure 3.8. generation of a new matching

- An augmenting path P is called a shortest augmenting path if the cardinality of P is the least among the augmenting paths.

- Let M and N be matchings. If $|M| = r$, $|N| = s$, and $s > r$, then $M \oplus N$ contains at least s - r vertex-disjoint augmenting paths relative to M.

- Let M be a matching, P a shortest augmenting path relative to M and P' an augmenting path relative to $M \oplus P$ then

$$|P'| \geq |P| + |P \cap P'| \qquad (3.6)$$

- These features reveal an algorithm to determine a maximum matching [Hopcroft]. The algorithm can be described as :

1. $M = \emptyset$

2. Let $l(M)$ be the length of the shortest augmenting path of M. Find a maximal set of paths = $\{P_1, P_2, \ldots, P_t\}$ with the properties that :

   1. for each i P_i is an augmenting path relative to M and $|P_i| = l(M)$,

   2. the P_i are disjoint,

3. Halt if no path exists.

4. $M = M \oplus P_1 \oplus P_2 \ldots\ldots \oplus P_t$
   go to 2.

In [Hopcroft] it is shown that the number of times that step 2) has to be performed is bounded by $2*(\sqrt{(|M_{max}|)} + 2)$.

The above mentioned algorithm indicates that it is essential to have

an effective method to find a set of the shortest augmenting paths. This will be described in the next paragraph.

3.4 Determination of a maximal set of shortest augmenting paths

Until now the bipartiteness of the graph has not been considered. This property is very helpful if we have to compute a maximal set of shortest augmenting paths. From the undirected incidence graph G(V,B) we determine a set of directed branches as follows: Assume M to be a matching,

1. if a branch b(v,w) ∈ M with v ∈ X and w ∈ F, we derive from branch b a directed branch b(v,w> pointing from v to w.

2. If a branch b(v,w) ∈ B-M with w ∈ X and v ∈ F we derive from b a branch b(v,w>, pointing from v to w.

Now we can start to construct a search graph in which we can find a maximal set of shortest augmenting paths. This graph is divided into levels of nodes and is constructed applying the following rules :

1. Level 1 contains all free nodes f ∈ F.

2. Level(i+1) of the graph is obtained by adding the directed branches from v to w, b(v,w>, to the graph where v ∈ level (i).

3. When a new node is already a member of the constructed graph, it is not inserted into the graph. Thus
   level(i+1) =
   $\{ w \mid b(v,w> \in B \wedge v \in \text{level } (i) \wedge w \notin \text{level}(j) \wedge j \leq i \}$.

4. We end the graph construction process if we have finished the construction of a new level, and a free node x ∈ X is a member of that level.

Because of the way of construction, the levels consist alternately of nodes f ∈ F and nodes x ∈ X. (Fig. 3.9) We also can see that a path in the search graph consists alternately of branches b(v,w> ∈ M, and branches b(v,w> ∈ B-M.

Searching for the maximal set of vertex disjoint shortest augmenting paths is done by "Depth First Search" (DFS) [Tarjan] in the constructed search graph.

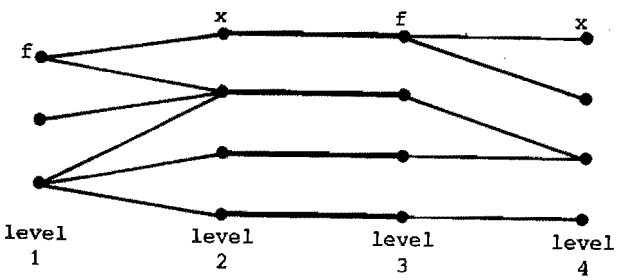


Figure 3.9. search graph

A shortest augmenting path related to the matching M is found by searching for a path from a free node $f \in F$ (located in the first level) to a free $x \in X$ (located in the last level). Each shortest augmenting path will start in level 1.

A node will only be added to a path in construction, if this node isn't already visited by the search.

The search for a path ends in one of the following two ways:

1. We end in a free $x \in X$, so we found a shortest augmenting path.
2. The DFS returns to the free starting node $f \in F$. Then there is no shortest augmenting path starting in f.

We stop the search, if we have tried to construct a path from each (free) node in level 1.

3.5 Determination of strong components in a directed graph

By determining a maximum matching we have found out, whether the set of equations is structural solvable or not. A matching however does not indicate in which order the equations have to be solved. This can be done by searching for strong components. Those components will impose a partial ordering on the equations.

In the previous paragraphs a description is given how a maximum matching in an incidence matrix can be found. During the determination of the matching we have coupled equations to variables. In the sequel we shall assume that the matching is complete. (Searching for strong components makes only sense if the system of equations is solvable).

An algorithm to find the strongly connected components is described in [Tarjan]. The algorithm is based on DFS and traverses all branches in the graph once, causing the time complexity to be linear with the number of branches in the graph. By performing a DFS on a directed graph, we determine a set of trees in the graph, called a forest. Each tree can consist of a set of subtrees. During the depth first search the nodes are numbered from low to high in the order they are visited. For each branch in the graph there are three possibilities:

1. Branch b(v,w> is a new branch of the tree; node w is not visited yet.

2. Branch b(v,w> is a branch pointing from v to a lower numbered node w within the same subtree. Branch b is called a frond and is responsible for a cycle in the graph, so v and w will be in the same strong component.

3. Branch b(v,w> is a branch pointing from v to a lower numbered node w, which is in another subtree. Branch b is called a cross-link and determines a partial ordering on the derived strong components. (see below).

During the depth first search each node gets a "lowlink" value initially equal to the number of the node. The lowlink value of v is the number of the node with the smallest lowlink number reachable from v by traversing zero or more tree arcs and at most one frond or cross link. This lowlink value can be determined for each node in the graph by searching the graph once with Depth First Search [Tarjan]. If the search is ready, all the nodes with the same lowlink value are member of a strong component.

Theorem 1

If the strong components are determined it is possible to rearrange them in such a way that the area below that covered by the strong components, contains only "0" entries. (Fig. 3.10) Remark:

If in figure (3.10) the entry marked with a "*" contains a "1", there exists a cycle (indicated by the arrows), so the rows and columns incident with this cycle have to be in one strong component. Obviously this is not true, so the entry marked with a "*" has to contain a "0".

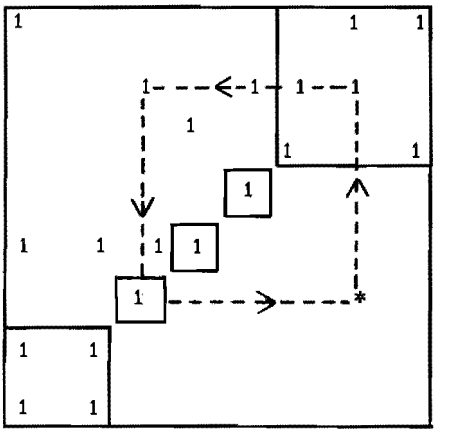


Figure 3.10. ordering of strong components

Proof:

Consider a part of an incidence matrix consisting of a number of strong components as shown in figure (3.11)

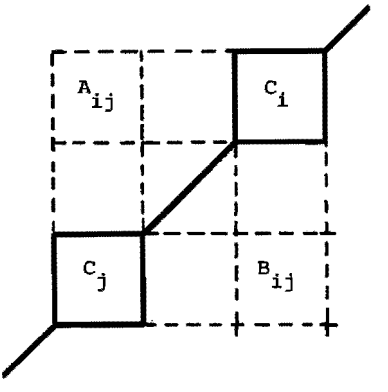


Figure 3.11. relations between two strong components

With the two strong components C_i and C_j two areas A_{ij} and B_{ij} are related. Now there are four possibilities:

1. Both A_{ij} and B_{ij} contain "0" entries.
   In this case,no crosslink between the two strong components exists and there is no order relation between the strong

components C_i and C_j.

2. Only A_{ij} contains "1" entries.
   Now there exists a crosslink pointing from C_i to C_j. If B_{ij} is allowed to contain only "0" entries, strong component C_i has to be above C_j.

3. Only B_{ij} contains "1" entries.
   Now there exists a crosslink from C_j to C_i. To maintain the "0" entries in the lower part of the matrix, the two strong components have to be swapped.

4. Both A_{ij} and B_{ij} contain "1" entries.
   Now there apparently exists a cycle, indicating that C_i and C_j are part of a larger strong component. This is contradictive to the assumption that C_i and C_j are strong components, so this situation can not occur.

(End of proof).

From these considerations we can conclude that if a crosslink from a strong component C_i to C_j exists, this crosslink imposes an ordering on the strong components. In practice this means that the equations related with strong component C_j have to be solved before the equations of strong component C_i can be solved.

3.6 Determination of essential variables

As we saw before, solving the system of equations can partly be done by back substitution. If, during this back substitution a strong component is encountered, all equations belonging to this strong component must be solved simultaneously, for instance by using a Newton Raphson iteration scheme. If the strong component incorporates $n + m$ variables and $n + m$ equations, the Jacobian matrix in the Newton Raphson iteration will generally be a sparse matrix with $n + m$ rows and $n + m$ columns. To obtain a new guess for the solution we need to solve a set of linear equations. To start up the iteration, initial values for all $n + m$ variables are needed.
So it is advantageous to reduce the number of unknowns during the iteration. By row and column permutations it is possible to derive a Bordered Lower Triangular Form (BLTF) of the incidence matrix related

to a strong component. (See Fig. 3.12) [Cheung] [Trouborst, 1979] [Smith].

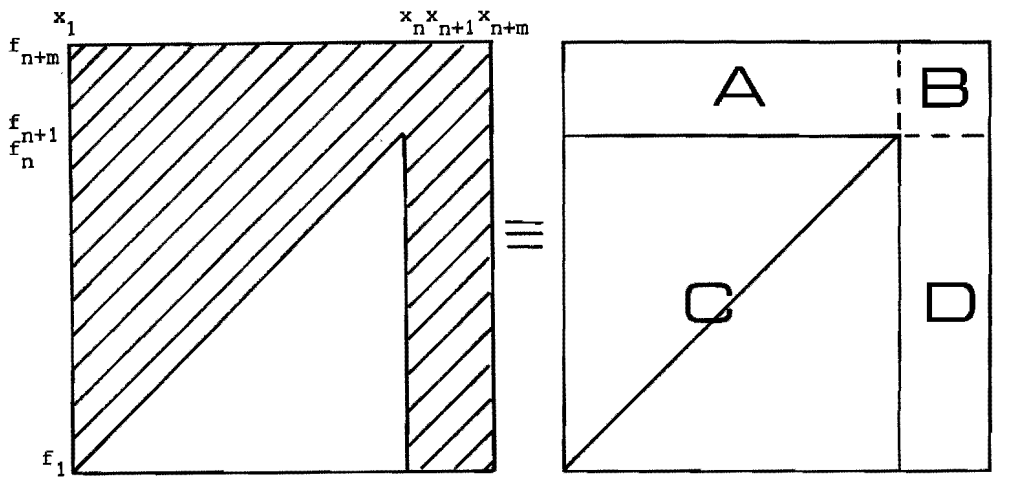


Figure 3.12. structure of a strong component

With such a BLTF it becomes possible to set up a Newton Raphson iteration in m variables and m equations. This can be achieved by expressing the variables $x_1,\ldots,x_n$, in terms of $x_{n+1},\ldots,x_{n+m}$, with the equations $f_1,\ldots,f_n$. In the following paragraph we shall give an algorithm to determine the BLTF.

3.7 The minimal essential set algorithm

The algorithm for determining the BLTF of a strong component is based on the minimal essential set algorithm described in [Trouborst, 1979][Trouborst, 1981]. To explain this algorithm we need some definitions:

1. A bordered upper triangular matrix can be divided into four submatrices, satisfying the following conditions :

   - Matrix B and C are square matrices;

   - $C_{ij} = 1$ if $i = j$,
     $C_{ij} = 0$ if $j > i$,
     So C has a upper triangular form.

- The off diagonal matrices A and D contain at least one nonzero entry.

2. An essential variable is a variable that is related to the columns of matrix B. Suppose the matrix M has the form shown in Fig. 3.13

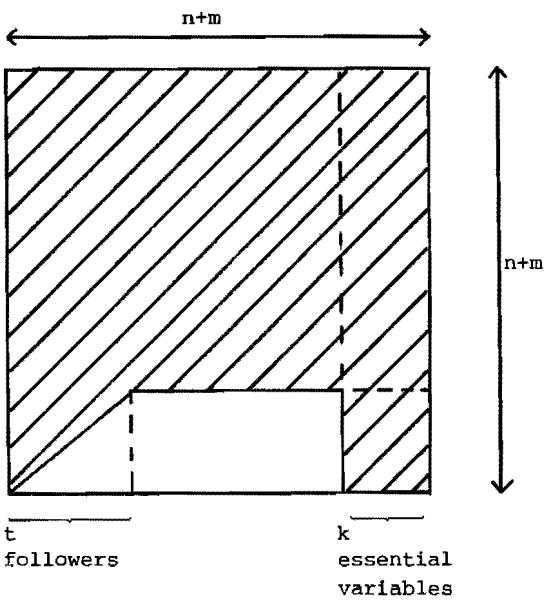


Figure 3.13. strong component

$M_{ij} = 0 \quad i = 1,2,\ldots,t \leq n + m - k \qquad i < j \leq m + n - k$

$M_{ii} \neq 0 \quad i = 1,2..,t$

3. The first t variables are called the followers.

4. The last k variables are, in accordance with 2, called essential variables.

5. All other variables are called non-followers.

6. The set of followers, united with the set of essential variables, is called the "train".

The algorithm described here, will try to find a set of essential variables with a cardinality as small as possible, such that there will be no nonfollowers any more and the matrix form in Fig. 3.12 will be obtained. Some important properties of this algorithm are :

1. The algorithm does not stick to the already found matching and in the submatrix B even zero valued diagonal elements may exist. This feature influences the cardinality of the set of essential variables extremely [Donald].

2. Within a subset of the equations, namely those that are linear, elimination steps can be included to reduce the cardinality of the set of essential variables.

3. The cardinality of the essential set is minimal but not minimum. The problem to find a maximum essential set has been shown to be NP-complete [Karp].

The algorithm picks from the set of nonfollowers a variable that, when added to the set of essential variables, results in a train as large as possible. This variable is obtained by trying all nonfollowers. When found, it will be added to the set of essential variables and, if the set of nonfollowers is not yet empty, the algorithm will start again to find the next essential variable. When adding a variable to the set of essential variables, we can determine the train as follows : Search an equation that contains only one nonfollower. If such an equation exists, add this variable to the train. Repeat this until there are no more equations with only one nonfollower.

The algorithm described so far searches a minimal essential set without elimination in the set of linear equations. The equations in the tableau can be divided into two classes:

1. The nonlinear equations and the linear equations with coefficients not equal to ±1 or 0.

2. The linear equations with coefficients equal to ±1 or 0.

The following equations belong to this second group :

- The Kirchoff voltage law equations;
- The Kirchoff current law equations;
- The equations in the small signal systems, derived from the Kirchoff voltage law equations and the Kirchhoff current law equations;

- The equations in the AC systems, derived from the Kirchhoff voltage law equations and the Kirchhoff current law equation.

Because of the fact that these equations are Kirchoff voltage and current law equations, we know that, if we subtract two of these equations, the variables appearing in both equations will be cancelled. This gives us the possibility to eliminate variables, and as a consequence entries in the incidence matrix, without knowing the values of these variables. The elimination can be performed during the search for those variables which, if added to the set of essential variables, result in the longest train. This can be achieved by a Gauss Jordan elimination [Hildebrand], performed only with the equations in class 2. These equations can be captured in the matrix formula in Fig. 3.14.

$M \cdot \underline{x} = \underline{y}$.

Where M comprises all equations of class 2.

The following partition is possible:

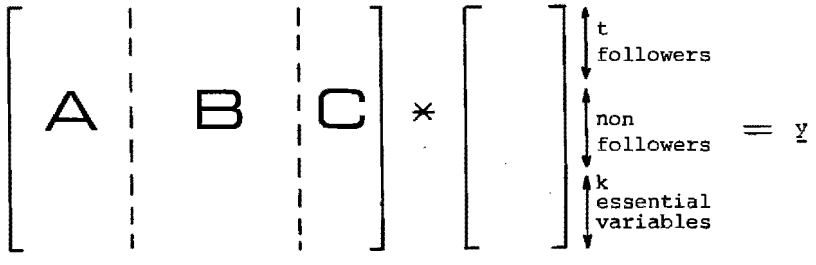


Figure 3.14. incidence matrix during Gauss-Jordan elimination

Matrix A is related to all followers. Matrix B belongs to the nonfollowers and matrix C belongs to the essential variables found at the current state. The Gauss Jordan elimination is executed with pivots in matrix B. The pivot for the Gauss-Jordan elimination is chosen in such a way that an identity matrix as large as possible is created. (See fig. 3.15.) In case a row occurs in the matrix B_r, in which only nonzero entries appear, an equation with only one nonfollower has been found. At the moment this situation is detected, the Gauss Jordan elimination will be stopped and the newly found follower will be added to the train [Trouborst, 1981].

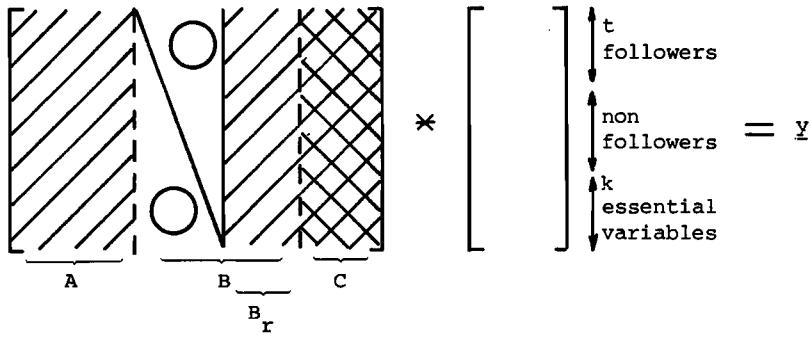


Figure 3.15. incidence matrix after Gauss-Jordan elimination

3.8 Evaluation of the minimal essential set algorithm

To evaluate the algorithm a number of tests has been performed. The tests are done with matrices derived from electrical circuits. The Gauss Jordan elimination, can be switched on by an option. To get insight into the performance of the algorithm, a second algorithm, which searches for a minimum set of essential variables is implemented. This algorithm does not change the already found matching and tries all the possible sets of essential variables, together with a branch and bound technique. This is of course only possible for small strong components. The results of the comparison are given in table 3.1.

maintaining the matching			not maintaining the matching			
size of component	number of ess. var.	CPU time (sec)	with elimination		without elimination	
			number of ess. var.	CPU time (sec)	number of ess. var	CPU time (sec)
3	1	0.14	1	0.20	1	0.20
6	1	0.12	1	0.22	1	0.24
15	1	0.14	1	0.54	1	0.50
16	2	0.16	1	0.64	1	0.53
31	3	0.28	2	3.54	2	2.46
55	6	>600	3	23.58	3	11.60
68	5	>600	3	52.92	3	16.46
77	-	-	8	296.18	8	56.44

TABLE 3.1.

From these results we can conclude that :

1. Maintaining the matching enlarges the cardinality of the minimal essential set;

2. Elimination does not reduce the minimal essential set significantly and uses a lot of computation time. Because of this result elimination has been taken out of the program.

As described earlier the reason for searching a minimal set of essential variables in a strong component is to obtain a smaller Jacobian matrix during the Newton Raphson iteration. During the search for essential variables, the equations are split into two groups : the essential equations and the non-essential equations. (Fig. 3.16)

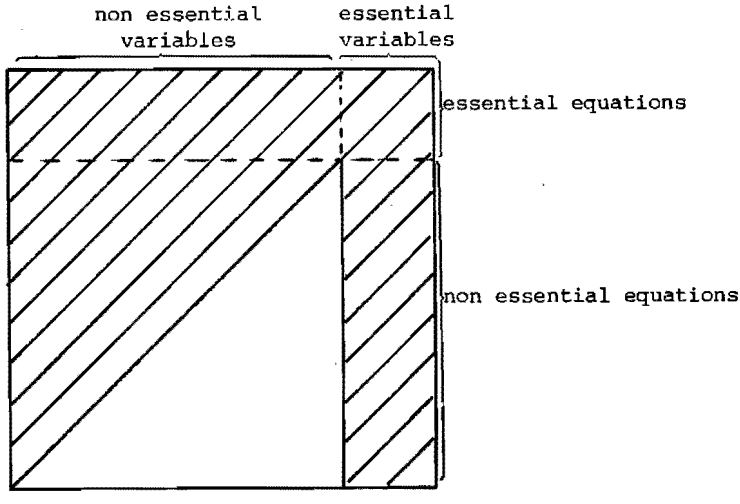


Figure 3.16. strong component with essential variables

To reduce the dimension of the Jacobian matrix it is necessary to write the non-essential variables in terms of the essential variables. This, however, is only possible if in the non-essential equations the matching variable can be written explicitly. For all Kirchoff equations this is the case. For some branch constraint equations and for the design constraint equations, this is not necessarily true. To be able to control the way how the essential variables are determined, and which variable will become the matching variable in a certain equation, a number of options are built into the algorithm:

1. A variable in a given equation can be forced to be the matching variable in that equation.

2. A variable in a given equation can be forced not to be the matching variable in that equation.

3. A certain variable can be forced to be an essential variable

If the designer uses one of these options it is possible that the resulting minimal essential set becomes larger.

3.9 Determining the matching in a set of equations with AC systems

If we want to incorporate the behaviour of a circuit at a number of different frequencies, we need a complete set of Kirchoff voltage law equations, Kirchoff current law equations and branch constraint equations for every frequency point. Moreover a new set of variables is needed for each frequency point. However, the incidence matrices for all frequency points are identical. The overall structure of the incidence matrix is given in Fig. 3.17

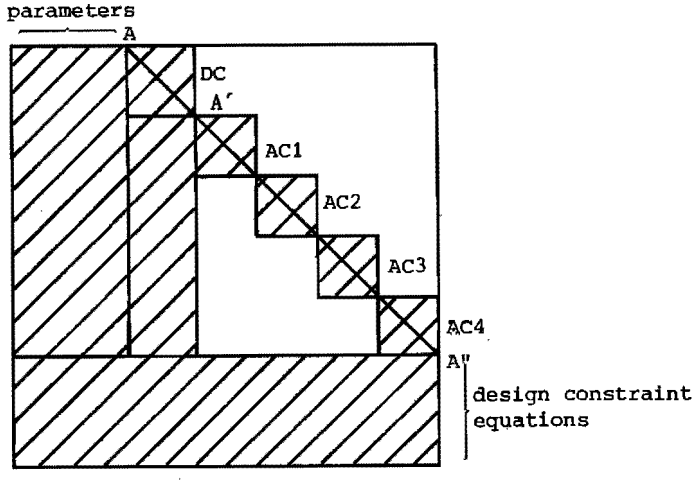


Figure 3.17. overall structure of the incidence matrix

The algorithm of Hopcroft and Karp determines a maximum matching by searching for the shortest augmenting paths, in a search graph constructed from a previously determined non-maximum matching and augment the matching with each shortest augmenting path. For the algorithm it is not essential at all how this non maximum matching

will be determined.
Because of the fact that the incidence matrices of all different AC-systems are equal, we are able to determine the matching of all AC systems by doing so for only one AC system and using this matching for all other systems.
Also for the system of equations describing the DC behaviour, a matching can be determined separately. In this way a smaller matching, given by line A-A' in Fig. 3.17 can already be found. By the algorithm of Hopcroft and Karp the construction of a maximum matching can be continued.
The above described method results in the following algorithm:

STEP 1 Find a maximum matching in the DC system by using the concept of Hopcroft and Karp (augmenting paths). The DC system is the set of equations and variables describing the DC operation point of the circuit. The vertices associated with the design constraint equations and the parameters are controlled to remain free. The result of these restrictions is that we can assume the parameter values to be known. Because of the fact that the DC system in this situation can be seen as a description of an existing circuit with known parameter values, we can assume that the DC system is a set of equations with a full rank Jacobian; so the set of equations is almost always solvable. Because of this we are certain that we can find a matching covering the whole DC system.

STEP 2 Find a maximum matching in one AC system. This is the set of equations and variables describing the AC operation of the circuit. The same restrictions as those for the DC system hold.

STEP 3 Catenate the AC system in the way displayed in Fig. 3.17. Now we have already found a matching represented by the line A-A" (Fig. 3.17).

STEP 4 Augment the matching with the aid of shortest augmenting paths.

In this way we reduce the time necessary to find a maximum matching. The time complexity of the algorithm of Hopcroft and Karp is $O((m+n)\sqrt{s})$ [Hopcroft], with

1. n = number of vertices in the bipartite graph associated with the incidence matrix;

2. m = number of edges in the bipartite graph associated with the incidence matrix;

3. s = number of edges of the maximum matching

A maximum on the number of times a set of disjoint shortest augmenting paths has to be found, is $O(s)$. The time needed to find a set of disjoint shortest augmenting paths is $O(m+n)$. If the matching covers the whole system of variables and equations (a complete matching) $s = n/2$ holds.

Definitions :

1. p = number of parameters;

2. v = number of equations in the DC system plus the design constraint equations;

3. e = number of entries in the DC system plus the design constraint equations;

4. x = number of frequencies we need (= number of AC systems) For each AC system we are allowed to introduce one design constraint equation. This design constraint can prescribe a frequency, but also another variable in the system, leaving the the frequency to be determined.

If we want to find the maximum matching without taking into account the resemblance between the AC systems we find :

$$n = 2(v - p + x(v - p) + p + x) - 2(v + x + x(v - p))$$

DC system

AC system

$$m = e + x(e - p) + x$$

DC system + design constraint.

AC system

The time needed to compute the maximum matching is asymptotically

$$T = C(m + n)\sqrt{s}$$

$$T \simeq C(n + m)\sqrt{(n/2)} = C(2(v + x + x(v - p) + e + x + x(e - p))\sqrt{(v + x + x(v - p))}$$

If we compute the time needed to obtain the maximum matching, taking into account the resemblance in the structure of the AC systems, we obtain :

STEP 1) $n' = 2(v - p)$

$m' = e - p$

$T' = C(2(v - p) + e - p)\ \sqrt{(v - p)}$

STEP 2) $t'' = T'$

STEP 4) $n''' = n$

$m''' = m$

$s = p + x$ (we only need to extend the matching with $p+x$ rows and columns)

$T''' = (n + m)(p + x)$

The obtained gain in execution time is

$$G = \frac{T}{T' + T'' + T'''}$$

$$G = \frac{2\,v + e + x(3 + 2\,v - 30 + e)\sqrt{(v + x(v - p) + x)}}{2(2(v - p) + e - p)(v - p) + (2\,v + e + x(3 + 2\,v - 3\,p - e)\sqrt{(p + x)}}$$

For large x (many AC systems) we find:

$$G \cong \frac{\sqrt{(v + x(v - p) + x)}}{\sqrt{(p+x)}} \cong \frac{\sqrt{(x(v - p) + x)}}{\sqrt{(x)}} = v\text{-}p+1$$

The gain obtained for a circuit with $v = 150$, $p = 30$ and $e = 554$ (uA-725 opamp) is plotted in Fig. 3.18.

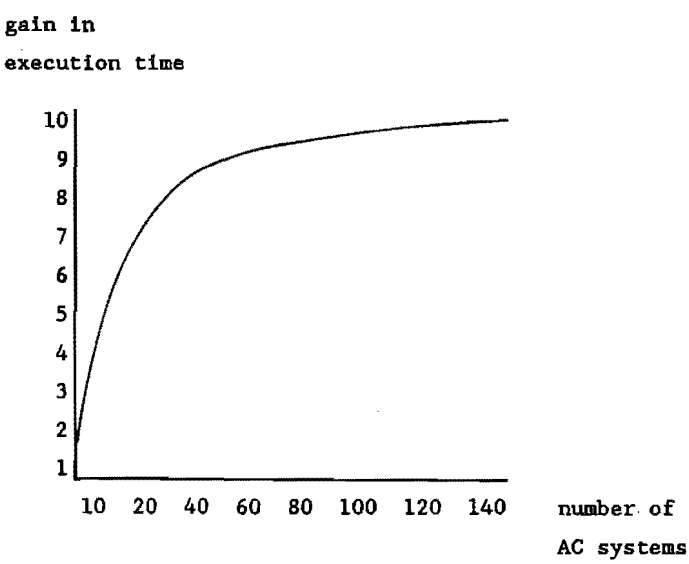


Figure 3.18. Gain in execution time versus the number of AC systems

4. Manipulating and solving the equations

The determination of the strong components and permutation of the incidence matrix into a canonical form imposes an ordering on the equations, determining in which order the equations have to be solved. The resulting incidence matrix indicates also which variable in an equation is the unknown variable (all the other variables are already solved by previous equations) and has to be written explicitly. This is the variable being in the matching. However this is not true for the essential equations (see Chapter 3). There are two different ways to generate the equations :

1. Generate the equations directly in the appropriate form; this means with the matching variable made explicit.

2. Generate the equations in a standard form and write them in the correct form afterwards.

The second method is chosen for two reasons:

1. It is much easier to write a procedure generating the equations in a standard form.

2. A procedure being capable of writing an explicit formula for a variable, is needed anyway, because the design constraint equations are in general not given in the appropriate form.

4.1 Writing a variable explicitly

To write a variable explicitly a formula manipulator has been written. This formula manipulator is able to handle a set of formulas appropriate for most of the equations occurring in a description of an electrical circuit.
In appendix A a BNF definition of the used equations is given. The "terms" (see appendix A) are important in the process of writing a variable explicitly.
In equation + v_6 = + i_6 * R_6 (4.1)
are + v_6 and + i_6 * R_6 the terms.
In equation + exp(+ q * v_5 / k / T) = + I_0 * q * v_{15} / k / T
are + exp(+ q * v_5 / k / t) (4.2)

and

$+ I_0 * q * v_{15} / k / T$

the terms.

The algorithm works as follows:

1. Scan the formula and search for the terms. When a term is encountered determine whether the variable, to be written, explicitly occurs in that term or not. If not, place this term with the appropriate sign in a string called B (in case the term occurred on the left side of the = sign, the sign changes). If so, substitute a "1" for the variable to be written explicitly (divide this term by that variable or multiply it with that variable depending on whether the variable is in the numeration or the denominator) and place the result with the appropriate sign in a string called A (when the term comes from the left hand side of the = sign, the sign does not change). Repeat this step until all terms have been placed in string A or string B.

2. The variable to be written explicitly can now be expressed by:

   $+x = (\text{string A}) / (\text{string B})$ (4.3)

   if x occurred in the numerator of a term and as

   $+x = (\text{string B}) / (\text{string A})$ (4.4)

   if x occurred in the denominator of a term.

This limited formula manipulator is able to handle most of the equations occurring on the design system. However, for some of the formulas something more has to be done.

1. $\exp(qv_5/(kT)) = i_{15}/(I_0\ (qv_{15})/(kT))$ (4.5)

   When v_5 has to be written explicitly, this equation is firstly written as :

   $qv_5/(kT) = \ln\ (i_{15}\ (I_o/qv_{15}/(kT)))$ (4.6)

Then it can be handled by the formula manipulator.

2. $x^n + r = - p$ (4.7)

When x has to be written explicitly, the power is initially ignored. Thereafter the 1/n-th power of the result is taken.

After these two extensions it may still be possible that a variable cannot be written explicitly as for instance the variable T in equation 4.5. In case this equation occurs in the strong component of dimension one, the only way to solve it is to iterate to the correct solution. If, however, this equation occurs in a strong component with a dimension larger than one, one can force this equation to be an essential equation. Because in the set of essential equations no variable has to be written explicitly the strong component can be solved. It is also possible to control the algorithm determining the essential variables in such a way that T will not be the matching variable in that equation.

4.2 Suffix notation

To be able to compute the equations efficiently, it will be necessary to write them in a suffix (reverse Polish or postfix) notation [McKeeman]. This can be done with a simple algorithm, whose flow diagram is shown in fig. 4.1. STACK is a last in first out array. The arrays STACK, INPUT and OUTPUT can contain variables, constants and operators. When the algorithm starts, the formula is stored in INPUT. After the algorithm the result can be found in the array OUTPUT. Priorities are assigned to the variables, constants and the operators. These priorities are listed below. "A inv" means -A. This operation is generated if the STACK or OUTPUT is empty and the top of the input stack is "-". To compute a formula written in a suffix notation, the following algorithm can be used.

1. Search for the first operator from the left side.
2. Search for the one or two corresponding operands.
3. Perform the operation.
4. Replace the operator and the operands by the result from 3).

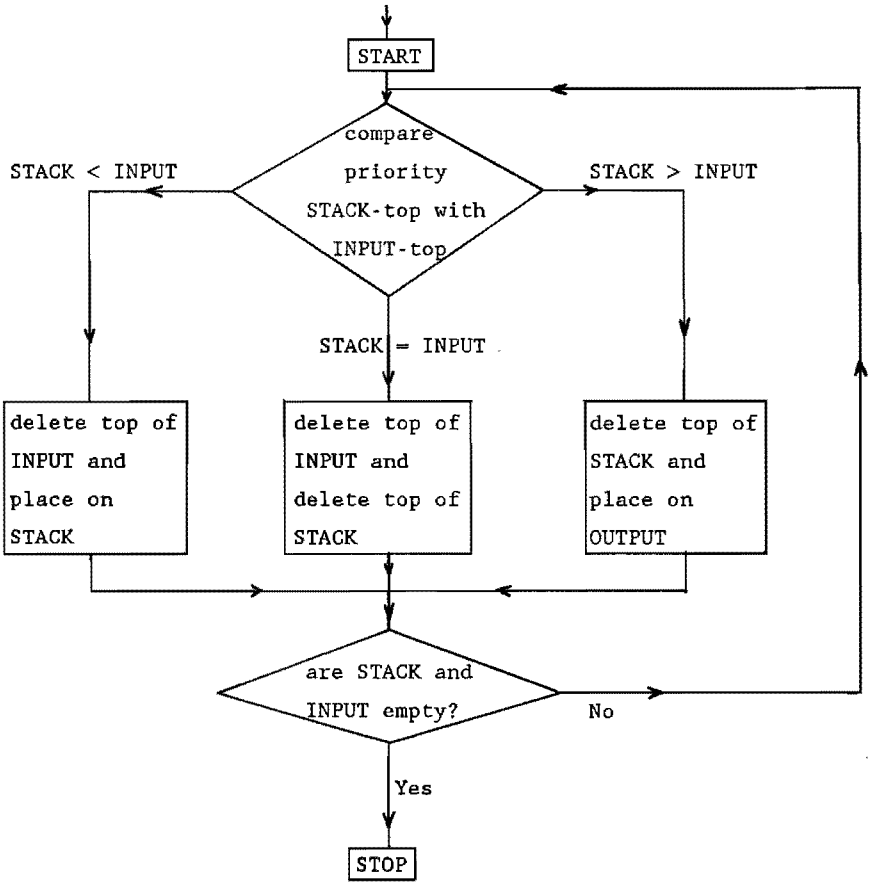


Figure 4.1. flow diagram

operator or operand	STACK priority	INPUT priority
+, -, inv	2	1
*, /, ↑	4	3
variable	6	5
constant	6	5
(	0	7
)	-	0
exp, ln	6	7

Table 4.1.

5. If there are operators left, go to 1)

6. Stop.

4.3 Solving the strong components

As already has been shown in Chapter 3 the rows and columns of a strong component can be rearranged in such a way that the incidence matrix of that strong component looks like the one in Fig. 3.12. The diagonal of the submatrix C consists of entries equal to one and the lower triangular part of submatrix C consists of only "0" entries.
We define $\underline{x} = (x_1, x_2, \ldots, x_n)^t$

$$\underline{x}_e = (x_{e_1}, x_{e_2}, \ldots, x_{e_m})^t.$$

$\underline{x}_e$ is the vector of essential variables.
$\underline{x}$ is the vector of all the non essential variables in a strong component.
The equations $f_1, f_2, \ldots, f_n$ can be written as :

$$x_i = f_i \ (x_1, x_2, \ldots, x_{i-1}, \underline{x}_e) \quad i = 1,2,\ldots,n \tag{4.8}$$

the equations f_{n+1} until f_{n+m} read :

$$f_i \ (\underline{x}, \underline{x}_e) = 0 \qquad i = n+1,\ldots,n+m. \tag{4.9}$$

The strong component must be solved by computing the equations $f_{n+1}, \ldots, f_{n+m}$ simultaneously.
To be able to do so, the variables of $\underline{x}$ have to be eliminated, resulting in :

$$\underline{F}(\underline{x}_e) = \underline{0} \text{ with} \tag{4.10}$$

$$\underline{F} = (F_{n+1}, F_{n+2}, \ldots, F_{n+m}) .$$

This can be obtained by using the equations $f_1, \ldots f_n$.

$$x_1 = f_1 \ (\underline{x}_e) = g_1(\underline{x}_e) \tag{4.11}$$

$$x_2 = f_2 \ (x_1, \underline{x}_e) = f_2 \ (g_1 \ (\underline{x}_e), \underline{x}_e) = g_2 \ (\underline{x}_e)$$

$$x_3 = f_3 \ (x_1, x_2, \underline{x}_e) = f_3 \ (g_1 \ (\underline{x}_e), g_2 \ (\underline{x}_e), \underline{x}_e) = g_3 \ (\underline{x}_e)$$

.
.
.

$$x_n = f_n\,(g_1\,(\underline{x}_e\,),\ g_2\,(\underline{x}_e\,),\ldots,\ g_{n-1}\,(\underline{x}_e\,),\ \underline{x}_e\,) = g_n\,(\underline{x}_e\,)$$

$$0 = f_{n+1} = (g_1(\underline{x}_e\,),\ g_2\,(\underline{x}_e\,),\ \ldots,\ g_n\,(\underline{x}_e\,),\ \underline{x}_e\,) = F_{n+1}\,(\underline{x}_e\,)$$

.
.
.
.

$$0 = f_{n+m} = (g_1(\underline{x}_e\,),\ g_2\,(\underline{x}_e\,),\ \ldots,\ g_n\,(\underline{x}_e\,),\ \underline{x}_e\,) = F_{n+m}(\underline{x}_e\,)$$

The set of equations $\underline{F}(\underline{x}_e\,) = \underline{0}$ can be solved by a Newton Raphson iteration scheme [Hildebrand]:

$$\underline{x}_e^k = \underline{x}_e^{k-1} - \left(J\Big|_{\underline{x}_e^{k-1}}\right)^{-1} * F(\,\underline{x}_e^{k-1}) \tag{4.12}$$

where $\underline{x}_e^k$ is the vector of computed values for the essential variables after k iterations and

$$J\Big|_{\underline{x}_e^{k-1}} = \begin{pmatrix} \dfrac{\partial \underline{F}_{n+1}}{\partial \underline{x}_{e_1}}\Big|_{\underline{x}_e^{k-1}} & \dfrac{\partial \underline{F}_{n+1}}{\partial \underline{x}_{e_2}}\Big|_{\underline{x}_e^{k-1}} & \cdots & \dfrac{\partial \underline{F}_{n+1}}{\partial \underline{x}_{e_m}}\Big|_{\underline{x}_e^{k-1}} \\ \dfrac{\partial \underline{F}_{n+2}}{\partial \underline{x}_{e_1}}\Big|_{\underline{x}_e^{k-1}} & \dfrac{\partial \underline{F}_{n+2}}{\partial \underline{x}_{e_2}}\Big|_{\underline{x}_e^{k-1}} & \cdots & \dfrac{\partial \underline{F}_{n+2}}{\partial \underline{x}_{e_m}}\Big|_{\underline{x}_e^{k-1}} \\ \vdots & & & \\ \dfrac{\partial \underline{F}_{n+m}}{\partial \underline{x}_{e_1}}\Big|_{\underline{x}_e^{k-1}} & \dfrac{\partial \underline{F}_{n+m}}{\partial \underline{x}_{e_2}}\Big|_{\underline{x}_e^{k-1}} & \cdots & \dfrac{\partial \underline{F}_{n+m}}{\partial \underline{x}_{e_m}}\Big|_{\underline{x}_e^{k-1}} \end{pmatrix} \tag{4.13}$$

Because the equations to be solved are known, we are analytically able to determine the derivatives of F. Thus there is no need to apply approximations. However the equations $F_{n+1},\ldots,F_{n+m}$ will be very complicated, and the computation of the derivatives will be lengthy. A better approach to compute the Jacobian is the following: first compute the derivatives of each non essential variable with respect to the essential variables. Hereafter compute the Jacobian

using the essential equations. The order in which the derivatives have to be established can be generated by the following algorithm:

```
FOR i := 1 UNTIL n DO
BEGIN
 FOR j := 1 UNTIL m DO
 BEGIN
```

$$\frac{\partial x_i}{\partial x_{e_j}} = \sum_{k=1}^{i-1} \frac{\partial f_i(x_1, x_2, \ldots, x_{i-1}, \underline{x}_e)}{\partial x_k} \frac{\partial x_k}{\partial x_{e_j}} + \frac{\partial f_i(x_1, x_2, \ldots, x_{i-1}, \underline{x}_e)}{\partial x_{e_j}}$$

```
 END;
END;
FOR i := n+1 UNTIL m+n DO
BEGIN
 FOR j := 1 UNTIL m DO
 BEGIN
```

$$\frac{\partial F_i(\underline{x}_e)}{\partial x_{e_j}} = \sum_{k=1}^{n} \frac{\partial f_i(x_1, x_2, \ldots, x_{i-1}, \underline{x}_e)}{\partial x_k} \frac{\partial x_k}{\partial x_{e_j}} + \frac{\partial f_i(x_1, x_2, \ldots, x_{i-1}, \underline{x}_e)}{\partial x_{e_j}}$$

```
  END;
END;
```

Before the derivative of an equation can be determined, this equation has to be written in suffix notation (Chapter 4.2). Hereafter the determination of the derivative implies repeatedly applying a number of basic rules. These basic rules are listed below, where (A)' means the derivative of A. A or B may represent more complex expressions.

(constant)'=> 0
(variable)'=> derivative of the variable , or zero. If the

variable is no member of the strong component
its derivative will be zero.

```
(A B *)'   =>  A  B' * B' A * +
(A B +)    =>  A' B' +
(A B -)    =>  A' B' -
(A B /)    =>  A' B / A B' * B 2 ↑ / -
(A exp)    =>  A' A exp *
(A ln)'    =>  A' A /
(A inv)'   =>  A' inv
(A B ↑)'   =>  A' B A B 1 - ↑ * *
```

(A B ↑) means A^B where B has to be a constant.

If a derivative of a constant or a variable results in a zero value, there is a possibility to reduce the length of the resulting expressions. For this simplification we can formulate the following rules :

```
A 0 + =>  A
0 A + =>  A
0 A - =>  A inv
A 0 - =>  A
A 0 * =>  0
0 A * =>  0
0 A / =>  0
0 inv =>  0
A 0 ↑ =>  1
0 exp =>  1
```

If the result of the reduction rule equals zero perhaps a second reduction rule can be applied.
The resulting algorithm to determine a derivative of an equation is shown below.

1. Search for the rightmost term or operator whose derivative has to be determined.

2. Search for the corresponding operand(s).

3. Apply the appropriate basic rule. Apply, if possible, one or more reduction rules.

4. If there are subexpressions whose derivative have to be determined go to step 1).

5. Stop.

4.4 The Newton Raphson method

To solve the equations belonging to a strong component a Newton Raphson scheme is used. For each iteration, equation (4.12) has to be solved. This equation can also be written as :

$$J\Big|_{\underline{x}^{k-1}} (\underline{x}_e^{k-1} - \underline{x}_e^{k}) = \underline{F} (\underline{x}_e^{k-1}) \qquad (4.14)$$

This set of linear equations can be solved with LU decomposition [Hildebrand]. During the LU decomposition, partial pivoting is used [Jennings]. This means that the largest element in a column is chosen as a pivot. Only the rows will be permuted.
As with most Newton Raphson iteration schemes in electrical engineering, the method is only appropriate if the initial guess of the solution is close enough to the real solution and the second derivative of the function is continuous in the neighbourhood of the solution [Kantorovich]. For equation systems resulting from descriptions of electrical circuits, these conditions mostly do not hold (mainly because the initial guess for the solution is not close enough to the solution). This also applies for our case. To improve the global convergence of the method a number of strategies are possible. Some of these are implemented. The extensions resulted in a significant improvement of the global convergence of the Newton Raphson method.

4.4.1 *Matching variables and convergence*

It appears - as is also pointed out in [Kevorkian] - that the convergence of the Newton Raphson method can be enhanced by pushing the right variables into the matching.

Let f be a function of

$$\underline{x} = (x_1 , x_2 , \ldots, x_n)^t \quad \text{and } f(\underline{x}) = 0 \text{ holds.} \tag{4.15}$$

From this equation a number of equations can be derived

$$x_i = f(x_1 , x_2,..,x_{i-1} ,x_{i+1} , \ldots,x_n) \text{ for } i = 1 ,\ldots, n \tag{4.16}$$

For convergence purposes it is advantageous to take variable x_i as a matching variable such that the relative change in x_i with respect to variations in x_j $(j \neq i)$ is as small as possible.

For the Kirchoff current law equation and the Kirchoff voltage law equation, these relative sensitivities are all +1 or -1. So for those equations it does not matter which variable is a matching variable. For branch constraint equations, however, this is different, especially for diodes (which are also part of the Ebers Moll transistor models). For a diode the branch constraint equation reads

$$i = I_0 \, (\exp(qv/(kT)) - 1) \tag{4.17}$$

$$\frac{\partial i}{\partial v} = q/(kT) \, I_0 \exp(qv/(kT)) \tag{4.18}$$

but also the formulation

$$v=kT/q \ln ((i+I_0/I_0)) \tag{4.19}$$

is possible yielding

$$\frac{\partial v}{\partial i} = \frac{kT*I_0}{q(i + I_0)} \tag{4.20}$$

For a conducting diode (4.20) yields the smallest value so (4.19) has to be used. For a nonconducting diode (4.18) results in the smallest value, so (4.17) is the appropriate equation.
Mostly the designer knows roughly the biasing configuration of his circuit. During the determination of the essential variables in the strong components he will be able to control the search of the matching.

4.4.2 *Maximum stepsize*

It often occurs that the steps taken by the Newton Raphson iteration method are very large. This may result in an intermediate solution

which is far from the initial guess and often also far from the solution. To circumvent this pitfall, the notion of a maximal step size is introduced. If the Newton Raphson method wants to take a step larger than the maximal step size, the maximal step size in the newly computed direction is taken as increment to the old intermediate solution rather than the value computed by the Newton Raphson method.

4.4.3 *Global nonincreasing function*

The objective of the Newton Raphson method is to find the vector $\underline{x}$ such, that $\underline{F}(\underline{x}) = \underline{0}$. If for an iteration step k+1 holds that

$$||\underline{F}(\underline{x}^{k+1})||^2 > ||\underline{F}(\underline{x}^{k})||^2 \tag{4.21}$$

it is likely that the k+1-th solution is not closer to the solution than the k-th solution. Nasrollah proposes a method to improve the global convergence of the Newton Raphson method by modifying the method to:

$$\underline{x}^{k+1} = \underline{x}^{k} - \lambda_k \, (J\Big|_{\underline{x}^k})^{-1} * \underline{F}(\underline{x}^{k}) \tag{4.22}$$

where $0 < \lambda < 1$ holds, and λ_k is chosen such that

$$||\underline{F}(\underline{x}^{k+1})||^2 < ||\underline{F}(\underline{x}^{k})||^2$$

(Nasrollah]

This results in the following algorithm.

Starting from an initial guess $\underline{x}_0$ set $\lambda_k = 1$ and $j = 0$.

Step I : Compute z^k from the set of linear operations
$J(x^k) \, . \, \underline{z}^k = F(\underline{x}^k)$.
Then compute $\underline{x}^{k+1}$ from
$\underline{x}^k - \underline{x}^{k+1} = \lambda_k \, \underline{z}^k$.
Then if

$$||\underline{F}(\underline{x}^{k+1})||^2 < ||\underline{F}(\underline{x}^{k})||^2$$

we accept the point $\underline{x}^{k+1}$ as the next iterate set $\lambda_k = 1$ and $j = 0$, then repeat.

Step II : If, on the other hand

$$||\underline{F}(\underline{x}^{k+1})||^2 > ||\underline{F}(\underline{x}^{k})||^2$$

we let $j = j+1$.
Then set $\lambda_k = (1/2)^j$,
go to step I and continue.

By this measure the algorithm cannot diverge any more; the algorithm, however, can still get stuck in a local minimum.

By the above described methods the convergence of the Newton Raphson method is increased significantly, but it is still possible that the program does not give the correct solution of a design problem. This happens often with not correctly stated design problems or in circuits with high gain factors. In Chapter 7 we shall give some ways to define design problems correctly.

5. Fault Location

5.1 Introduction

One of the largest problems in electrical engineering these days is testing of integrated circuits. Because of the continuous enhancements in processing facilities of integrated circuits there is the possibility to produce smaller features on silicon. Thus it is also possible to compose larger integrated circuits. This leads to a steady growth of the complexity of the circuits that can be integrated. However, the number of ports through which the circuit is accessible has not grown equally fast. Because of this it is getting more and more difficult to test integrated circuits. This is not only true for digital circuits but also for analog circuits. For digital circuits a number of approaches have been found to tackle these problems, for instance the scan path techniques and the self testing techniques. For analog circuits, however, those techniques are not applicable.

In the testing field two kinds of problems can be distinguished. The first kind is the go/nogo test. This test is necessary at the end of the production process because the manufacturer wants to check whether a circuit is working within the desired specifications or not. For an analog circuit this test may be relatively easy. In case of for instance an operational amplifier this means measurement of the output response to a given set of input signals. There is, however, a much more difficult test which can be seen as a diagnostic test. This test is necessary during the development of an integrated circuit . The designer does this test to determine which circuit element value causes a circuit not to work properly. This problem can also be stated in another way, which reveals the relation between the testing problem and the design problem, as described in the previous chapters: The designer wants to know the measurements he has to make in order to be able to compute the value of the parameters in the circuit. These measurements, can be looked upon as constraints imposed on the circuit. As one can see the problem is equivalent to the design problem where a designer wants to know which values of a

circuit element must be chosen to obtain a desired response. The problem left is how to couple measured values to electrical components such that if instead of the component value the measured voltage is taken as known, the system of equations is a solvable system. It will be explained below how to do this.

5.2 Manipulations in the incidence matrix

The incidence matrix gives us a powerful tool to find out which equations will in principle be needed to compute an electrical component. Consider the electrical circuit to be tested with all electrical parameters at their nominal value. From this circuit we can derive the Kirchoff voltage law equations, the Kirchoff current law equations and the branch constraint equations describing the circuit elements. By adding to this set of equations design constraints, assigning a constant value for each parameter in the circuit, we shall obtain a solvable set of equations. This is true because the equations obtained, describe an existing electrical circuit and solving this set of equations is equivalent to simulating this circuit. Because the set of equations is solvable we know that there has to be a complete matching in the incidence matrix which is derived from it.

Definition: Let G(V,B) be the directed graph derived from the incidence matrix with a complete matching.
A parameter p is reachable from a variable x if it is possible to construct a directed path in the graph G(V,B) from the vertex x, to the vertex p.

With this definition of reachability we are able to find out which variable values are controlled by a certain parameter or, the other way around, with which variable it is possible to find out about the value of a parameter. Notice that here the order in which the equations are solved, is opposite to the direction of a path.
As an example consider fig. 5.1.

As an example in fig 5.1 the node r_3 is reachable by the node voltage n_2 because there exists a path in the directed graph from n_2 to r_3,

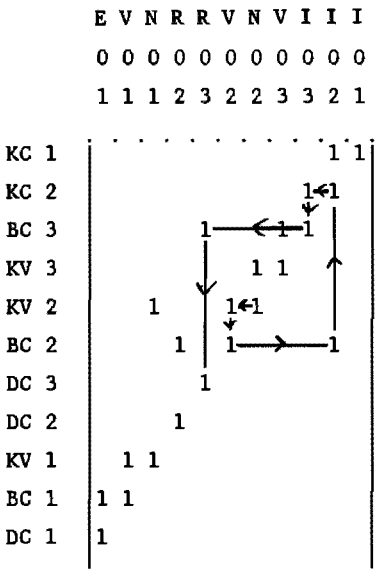


Figure 5.1. incidence matrix of circuit in Fig. 2.1

but computation of n_2 if r_3 is known, is in the opposite order. With the previous definition we are able to state the following :

Theorem 2

Assume that there is a complete matching M, a parameter p is reachable from a variable v and there exists an equation of the form p = constant, then, if the design constraint p = constant is replaced by v = constant, it is possible to find a complete matching M^* in the newly defined set of equations.

Proof :

Consider an existing path P from the variable v to the parameter p. Such a path exists because p is reachable from v. The path comprises matching and nonmatching branches alternatingly. The arcs of the matching are denoted by bold lines.

The matching arc at the end of the path represents the incidence of p in the design constraint p = constant. The new matching can be constructed by the following equation :

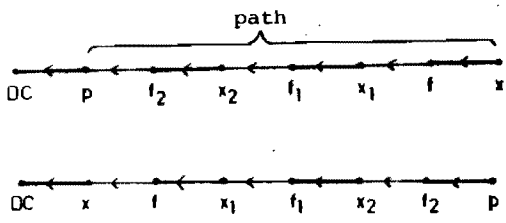


Figure 5.2. generation of a new matching

$M^* = M \oplus P = (M \cup P)/M \cap P.$

The matching arc at the end of the path represents the incidence of v on the design constraint v = constant.
(End of proof).

By applying theorem 1 we are able to identify a measurable voltage or current to compute the value of the parameter. If we do so repeatedly with the members of a set of unknown parameters, we may be able to find a set of measurable circuit variables so as to compute the values of the unknown parameters.

5.3 Sensitivity matrix

Experiments with some practical circuits have shown that the approach described previously does not work always. It appears that often parameter values are computed inaccurately or incorrectly.
The reasons for this are twofold :

1. Reachability from a parameter by a variable means that the value of the parameter depends on the value of that variable. However, numerically this dependence may be very small or even zero by way of compensation, such as in balanced input stages. So the value of the sensitivity $\frac{\partial v}{\partial p}$ is of great interest and has to be large in absolute value.

2. The sensitivities of one parameter relative to measured variables are (nearly) a linear combination of the sensitivities of a set of other unknown parameters. That this can cause a faulty computation of the parameter values can be explained as follows :

If the sensitivities of parameter p_{k+1} are linear combinations of the sensitivities of the parameters $p_1,\ldots,p_k$ then we can not detect whether a disturbance in the measurements is caused by a faulty parameter p_{k+1} or by a linear combination of the faulty parameters $p_1,\ldots,p_k$.
To solve these problems we introduce the following definitions :

$v_1,v_2,\ldots,v_n$ voltages and currents which are measurable.
$p_1,p_2,\ldots,p_m$ parameter values which have to be checked.

$$G = \begin{bmatrix} \frac{\partial v_1}{\partial p_1} & \frac{\partial v_2}{\partial p_1} & \cdots & \frac{\partial v_n}{\partial p_1} \\ \frac{\partial v_1}{\partial p_2} & \frac{\partial v_2}{\partial p_2} & \cdots & \frac{\partial v_n}{\partial p_2} \\ \vdots & & & \\ \frac{\partial v_1}{\partial p_m} & \frac{\partial v_2}{\partial p_m} & \cdots & \frac{\partial v_n}{\partial p_m} \end{bmatrix}$$

This matrix will be called the "sensitivity matrix". The values in this matrix can easily be computed with the aid of a (transposed) small signal system [Hachtel]. To make sure that the two situations mentioned above are not the reason of failure, we must choose such a set of parameters and measurements that a submatrix J of G, belonging to that set of parameters and measurements, is well conditioned and the sensitivities are not too small by values. We identify such a well conditioned submatrix by a technique derived from full pivoting on Gaussian elimination [Hildebrand]. The algorithm to perform the selection of parameters and measurements is as follows :

1. Assume that already k parameters and measurements are selected.

2. Search for the largest absolute value among the coefficients

$k < i \leq m$

g_{ij}

$k < j \leq n$

in G such that the parameter associated with row i is reachable from the measurable quantity associated with column j. This defines a pivot $\hat{g}_{ij}$

3. Rearrange the rows and columns such that entry $\hat{g}_{ij}$ appears at place $k+1,k+1$.

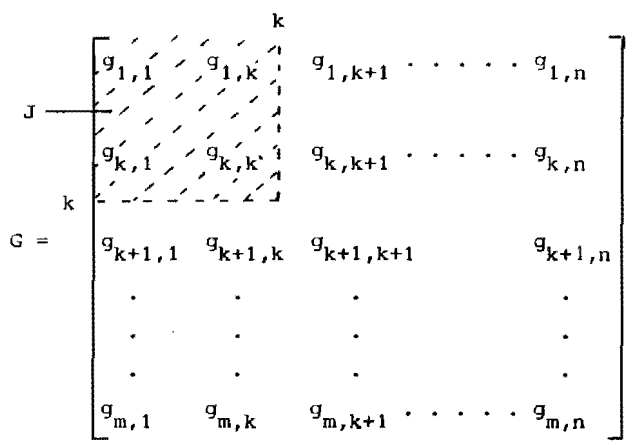


Figure 5.3. structure of the sensitivity matrix

4. Perform a Gaussian elimination step in G.

$$g_{ij} = - \frac{g_{k+1,j}}{g_{k+1,k+1}} * g_{i,k+1}$$

for all i,j such that:

$$k+2 \leq i \leq m$$
$$k+2 \leq j \leq n$$

5. Repeat these steps until no parameters or measurements are left, or until no appropriate pivot can be found any more.

5.4 Determination of the computations to be made

In general, and especially in IC design there will not be enough points to measure voltages and currents. If so, perhaps the same measurements can be made at different supply voltages or at different frequencies. Mostly, however, this will not result in as many measurements as there are parameters. Assume that only one parameter is faulty (an assumption that is often made in these analyses). Now we try to locate the faulty parameter by doing several computations, each time selecting a different set of parameters to be tested. For any such set we assemble an associated set of measurable quantities according to the above procedure. Let the sets of parameters and measured quantities be $P^{(i)}$ and $V^{(i)}$ respectively, where $i = 1,2,...$ identify the various computations.

Because of tolerances on the nominal values of all the parameters, no parameter will be computed to have the exact value. To decide whether a parameter is correct or not we have to take it's tolerance into account.

From any computation the result may be:

1. Only one parameter is off it's tolerance region.

2. More than one parameter is off it's tolerance region.

In the second case, assuming the result of the computation is unique none of the tested parameters can be the faulty one. Thus we select an alternate set $P^{(i)}$ from the parameters not yet tested, and continue.

In the first case the parameter with the non nominal value may be the faulty one. However, any of the not yet tested parameters could be the faulty one as well.

Eventually more and more parameters will be excluded to be the faulty one. A final decision as to which one of the remaining parameters is actually faulty can possibly be obtained by inserting those parameters simultaneously into some test set $P^{(i)}$. Either we are able to narrow down the set of possible faulty parameters to one by a sequence of computations, this way deciding the game. Alternatively we may get stuck with all badly conditioned computations. In that case the faulty parameter can not be identified by the measured

quantities because of numerical reasons.
If it is not possible to identify the faulty parameter, it is also possible that there are more faulty parameters. The same procedure as described above, can be followed with the assumption that there are two faulty parameters, in this way trying to identify two faulty parameters. An example can be found in chapter 7.

6. Implementation Aspects

6.1 The Modular Structure

The program has been written in the Fortran IV programming language and has been set up as an interactive program. Possible commands are for instance INP(UT) causing the program to read the input description of the circuit. If there exist no syntax errors in the input the incidence matrix for the equations derived from that description will be generated. DEC results in the determination of the matching together with the strong components. For a complete set of commands and their purpose: See Appendix B.

Each command invokes a separate and self contained part of the program. The results of such a step can afterwards be evaluated by the user. He then can decide to proceed with the next command or to redefine the design problem or to rerun some previous commands with other options. The data needed for each step are always read-in from files stored on disk. The results will be written back to the disks, mostly on other files. Because of this it is not necessary to perform all computations in one session and it is also possible to rerun a part of the program.

The relation between the different parts of the program and the data files is shown in Fig. 6.1.

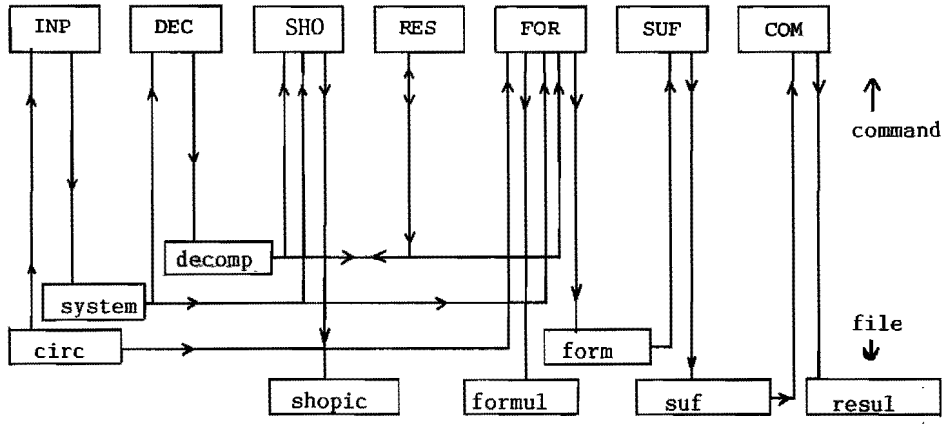


Figure 6.1. Relation between the program and the files

6.2 Storage of the different intermediate results

Because there is a variety of data and algorithms, not all data can be stored in the same way. While defining a data structure, not only memory space has to be considered. Also very important is the data structure to be such that the algorithm can run efficiently on it.
Next we shall describe some important data structures used in the program.

6.2.1 *The incidence matrix*

The incidence matrix is not implemented as a matrix. This would be inappropriate for two reasons :

1. The incidence matrix is sparse, so in terms of memory, this would be an inefficient realization.

2. The algorithm to find a matching and the strong components does not run on a matrix but needs a graph representation of the structure.

The storage of the incidence matrix is done rowwise, or formulated in another way, equation after equation. For each equation a list of incident variables is generated. This is an effective way of storage in terms of memory and also with respect to the algorithms which work on it.
A close examination of the matching algorithm reveals that in order to find the shortest augmenting paths we must know which variables are incident with a given function to generate the arcs pointing from an uneven numbered level to an even numbered level. The arcs of the even to the uneven numbered levels can be found in the matching which is stored in another array. The variables and the equations are numbered in the order they are generated.
The storage of the incidence matrix is done in two arrays.

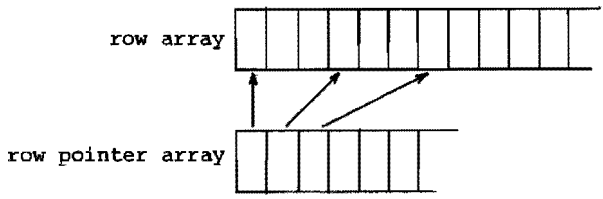


Figure 6.2. storage structure of the incidence matrix

The incident variables of equation *i* are stored in Row Array at the places Row Pointer Array [*i*] until Row Pointer Array [*i*+1].

The names of the variables and equations are stored in two arrays, where the name of equation *i* is stored in the *i*-th position of one such array. The names of the variables are stored in the same way in the other array.

The matching is stored in another array named the Match Array. For this purpose the variables are renumbered. The new number of a variable is equal to the old number plus the total number of equations. If variable *i* is matched with equation *j* Match Array [*i*] is equal to *j*, and Match Array [*j*] is equal to *i*.

The strong components are stored in two arrays (Fig. 6.3)

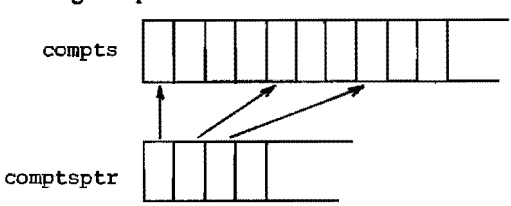


Figure 6.3. storing structure for the strong components

The information about strong component *i* is stored in array compts[comptsptr[*i*]] until compts[comptsptr[*i*+1]]. In strong components the number of the equation together with the number of the matching variable of that equation are stored pairwise. The order of the pairs of numbers is such that it indicates the order in which the equations in the strong components have to be solved.

6.2.2 *The equations*

Each equation is stored in a separate array. Entities to be represented in the equations are :

- Variables, derivatives of variables and parameters;
- operators like +, -, /, exp;
- constants.

The values of all these entities - except for the operators - are stored in an array called VALUE. All these entities are mapped into integers in the following way :

1. $1 \leq i \leq 500$ is the value of variable number i.
2. $500 < i < 520$, the integer i represents an operator for instance

   501 +
   502 -
   515 exp

3. $600 \leq i < 890$ represents a constant value. This value is stored in VALUE (i).
4. $890 \leq i \leq 990$ The jacobian of the Newton Raphson iteration is stored here.

To be able to perform the Newton Raphson iteration, the Jacobian matrix has to be computed. This matrix is stored row by row in the array VALUE, starting at entry 890. Because the highest entry to store an element of the Jacobian matrix in the array value is 989, the maximum number of iteration equations is 10 so the strong components may contain no more than 10 essential variables each.
During the Newton Raphson iteration process the derivative of each non essential variable with respect to each essential variable has to be computed and stored. The derivatives with respect to the first essential variable are stored in the array VALUE in the range from 1001-1500.
For the derivative with respect to the second essential variable the values are stored on the range 1501-2000 and so on.

6.2.3 *Input language*

The definition of the circuit and the design rules is described in an input file. The file consists of two parts. In the first part the

circuit is defined. The language for this part is a SPICE-like language. Each line defines an electrical element in the circuit. The second part consists of the design rules. For a detailed description of the input language see Appendix C.

6.2.4 *The output formats*

There are essentially three places where output can be obtained from the program. Firstly it is possible to get pictures from the incidence matrix. In an interactive way a part of the incidence matrix can be defined. The defined part of the incidence matrix can then be displayed on the terminal screen or stored in a file.
The second output possibility is a printout of the generated equations which have to be solved.
Finally the computed values of all variables can be printed out.
In all these ways of output the equations and variables are represented by names. The naming conventions are defined in Appendix D.

7. Design strategy and examples.

As explained in chapter 3, the computation time for solving the set of equations largely depends on the size of the strong components. In strong components containing more than one equation, a Newton Raphson iteration has to be performed. So it makes sense to formulate the design problem in such a way that only small strong components will result.

As can be expected, there is a close relation between strong components and the circuit. If for instance the circuit incorporates an amplifier with a feedback loop, the equations describing this part of the circuit will be one large strong components. If the feedback is taken away, this strong component will fall apart into a number of smaller strong components.

Taking away the feedback, does not mean that the circuit topology has to be changed. If with a design constraint equation for instance the voltage of a node in the feedback circuit is defined, there will (electrically) be no feedback any more, resulting in a number of smaller strong components.

An indication for which variables have to be fixed by a design constraint equation to split up a strong component is given by its essential variables. This is true because if the essential variables are taken out of the strong component (which is done by incorporating them into a design constraint equation) all the other equations can be solved by back substitution. It should be noted here that in general there is more than one set of candidates for being essential variables.(The algorithm to determine the minimal essential set chooses one of these).

In circuits with high gain factors, it is important that there does not exist a strong component describing this part of the circuit. By the use of design constraint equations, that strong component has to be split up into a number of smaller strong components.

Because of the fact that the design constraint equations derived from the measurements during fault location, will fix voltages and currents in the circuit, the strong components will be small, so the calculation will be performed efficiently.

In the next sections we will give two examples illustrating the capabilities of the described design and test-generation package. The first example will be about the design of an amplifier, the second will be an example of the generation of test measurements to test a circuit.

7.1 design of an amplifier

The circuit to be designed is depicted in fig. 7.1.

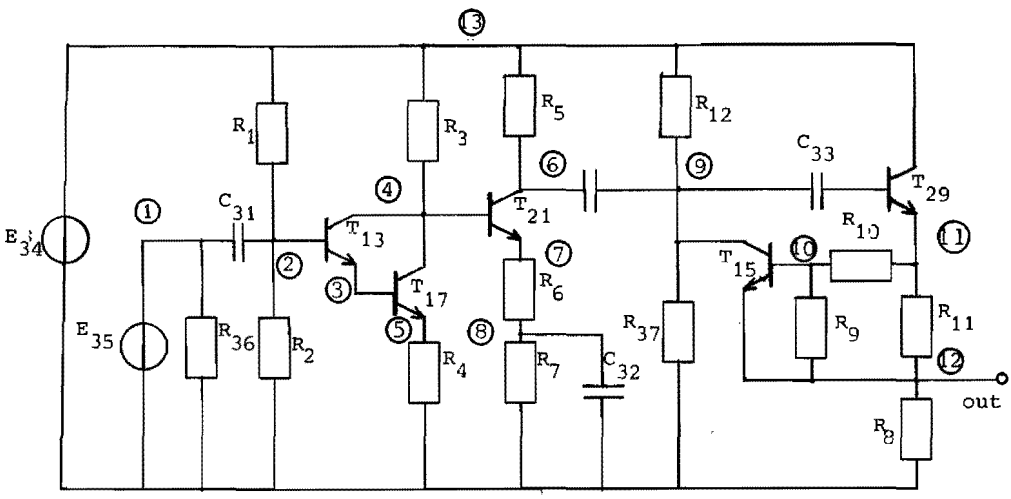


Figure 7.1. The circuit to be designed

The topology of this circuit can be described in the file circ.dat.

```
#example
ex
0
01R1302
02R0200
03R1304
04R0500
05R1306
06R0708
07R0800
08R1200
09R1012
10R1011
```

```
11R1112
12R1309
13T040302NH
17T040503NH
21T060704NH
25T091210NH
29T131109NH
31C0102
32C0800
33C0609
34E1300
35E0100
36R0100
37R0900
```

As we can see that "half Ebers Moll" models are incorporated for the transistors. This means that the transistors are correctly modeled if they are not operating in saturation mode. This is true for this example.

The design constraint equations can be added to the description of the circuit.

```
DC01(R01)=+R01=+370000
DC02(R02)=+R02=+100000
DC03(N04)=+N04=+3.9
DC04(R04)=+R04=+1800
DC06(R06)=+R06=+630
DC08(R08)=+R08=+2200
DC09(V25)=+V25=+0.3
DC10(R09)=+R09=+1000
DC11(I29)=+I29=+0.004
DC12(R12)=+R12=+39000
DC13(R11)=+R11=+470
DC14(T)=+T=+300
DC15(A14)=+A14=+0.99
DC16(A18)=+A18=+0.99
DC17(A22)=+A22=+0.99
DC18(A26)=+A26=+0.99
DC19(A30)=+A30=+0.99
```

DC20(E34)=+E34=+15

DC21(E35)=+E35=+0.0

DC22(X35)=+X35=+1.0

DC27(W01)=+W01=+10000

DC23(C31)=+C31=+0.0000002

DC24(C32)=+C32=+0.0001

DC25(C33)=+C33=+0.000001

DC26(R36)=+R36=+10000

DC28(X34)=+X34=+0.0

DC29(W02)=+W02=-10000

DC30(1N12,2N12)=+1N12*2N12=+1600

DC31(R07)=+R07=+33000

A number of comments can be made here:

1. Design constraint equation DC03 defines the DC operation point of the first stage of the amplifier. Resistor R03 has to be computed to obtain the desired operation point.

2. With design constraint equation DC09 we want to express that the transistor in the current limitating circuit must not be active during normal operation of the circuit (transistor 25 has to be in cut-off mode). Resistor R10 is the designable parameter.

3. Design constraint DC11 establishes the operating point of the output stage of the circuit. (R37 is the designable parameter).

4. Design constraint DC16 - DC19 define the α's of the transistors.

5. DC27, DC29 and DC30 express a constraint on the gain of the circuit. There is a small problem here. If a designer imposes a gain on an amplifier, he mostly is only interested in the absolute value of the gain, and not in the real and imaginary part. The modulus of a complex number can be found, by multiplying it with its complex conjugate. The complex conjugate, can be computed by defining a negative frequency. Design constraint equation DC30 defines a gain of 40. (The AC signal source has an amplitude of 1, X35).

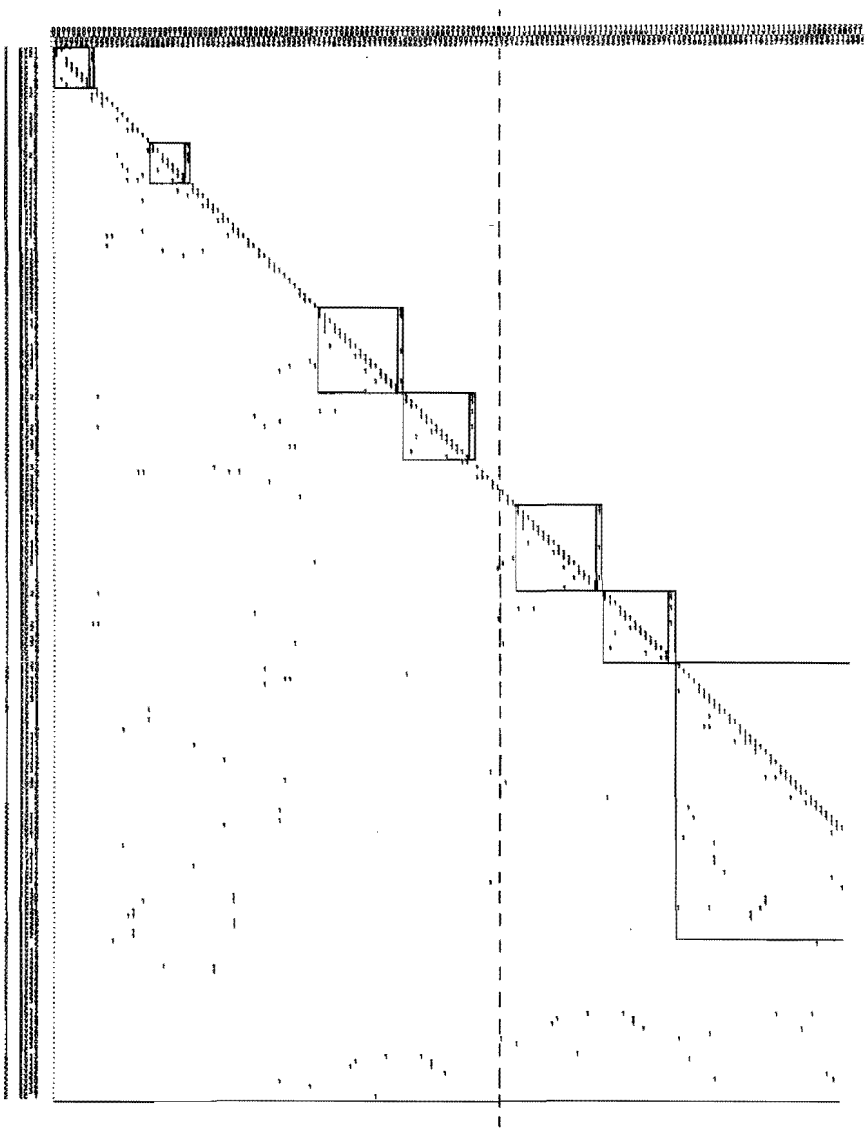
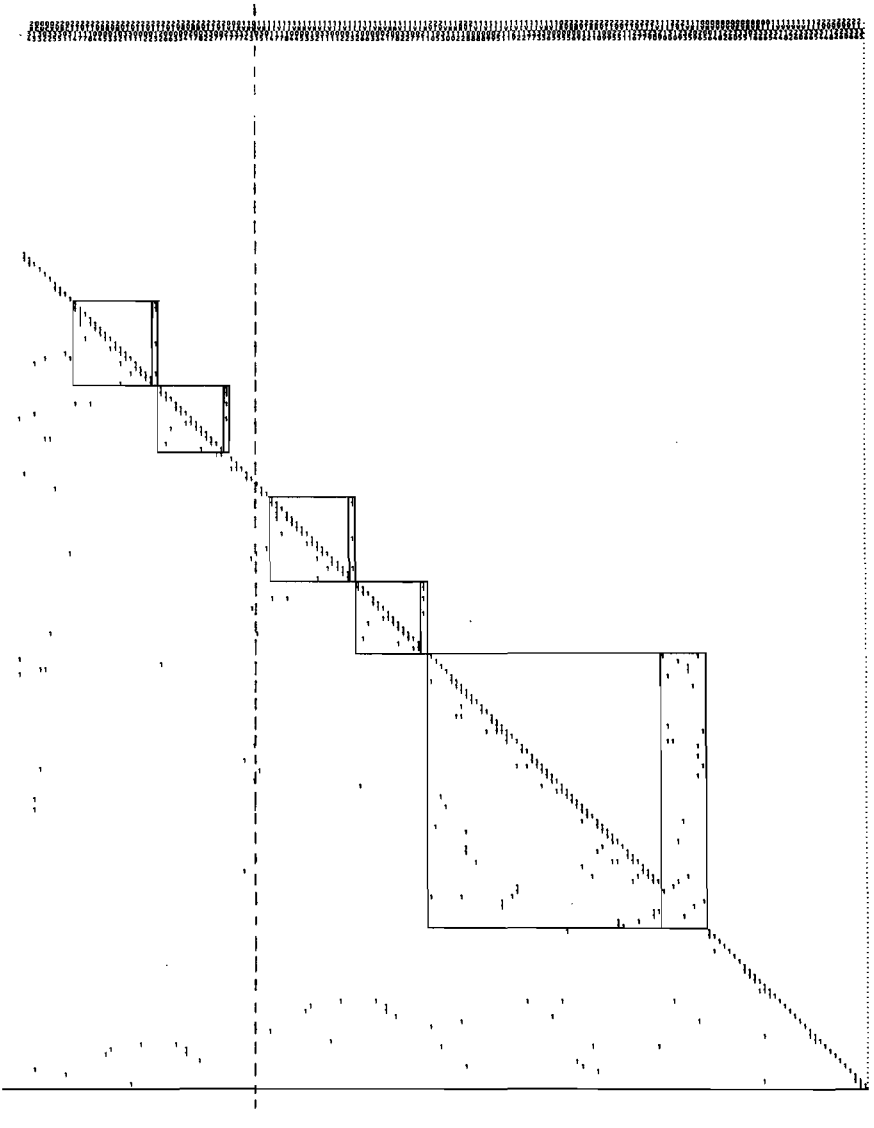


Figure 7.2. Incidence matrix



The design problem formulated in this way results in a set of equations with a complete matching, indicating a solvable set of equations.
The incidence matrix after determination of the essential variables in the strong components is shown in figure 7.2. As we can see there exist 7 strong components.
The equations to be solved are listed below:

```
ODC20: +0E34=(+15)                  OBC34: +0V34=(+0E34)
OKV34: +0N13=(-0V34)*(-1)           ODC1 : +0R1=(+370000)
ODC14: +0T13=(+300)                 ODC4 : +0R4=(+1800)
ODC16: +0A18=(+0.99)                ODC15: +0A14=(+0.99)
ODC2 : +0R2=(+100000)               OKC2 : +0=(+0I1-0I2-0I13+0I14)
OKC5 : +0I4=(+0I17)                 OBC4 : +0V4=(+0R4*0I4)
OKV4 : +0N5=(-0V4)*(-1)             OBC17: +0V17=(+LN(+0I17/IO+1))
                                           /(+Q*1/K/0T13)
OKV17: +0N3=(-0V17-0N5)*(-1)        OBC18: +0I18=(+0A18*0I17)
OKC3 : +0I13=(-0I17+0I18)*(-1)      OBC13: +0V13=(+LN(+0I13/IO+1))
                                           /(+Q*1/K/0T13)
OKV13: +0N2=(-0V13-0N3)*(-1)        OKV1 : +0V1=(+0N13-0N2)
OBC1 : +0I1=(-0V1)/(-0R1*1)         OKV2 : +0V2=(+0N2)
OBC2 : +0I2=(-0V2)/(-0R2*1)         OBC14: +0I14=(+0A14*0I13)
ODC3 : +0N4=(+3.9)                  ODC31: +0R7=(+33000)
ODC6 : +0R6=(+630)                  OBC6 : +0=(-0V6+0R6*0I6)
OBC21: +0V21=(+LN(+0I21/IO+1))
/(+Q*1/K/0T13)                      OKV21: +0N7=(-0V21+0N4)
OKC7 : +0I6=(+0I21)                 OKC8 : +0I7=(+0I6)
OBC7 : +0V7=(+0R7*0I7)              OKV7 : +0N8=(-0V7)*(-1)
OKV6 : +0V6=(+0N7-0N8)              ODC17: +0A22=(+0.99)
OBC22: +0I22=(+0A22*0I21)           OKC4 : +0I3=(-0I14-0I18-0I21+
                                           0I22)*(-1)
OKC6 : +0I5=(-0I22)*(-1)            ODC9 : +0V25=(+0.3)
ODC10: +0R9=(+1000)                 OBC25: +0I25=(-EX(+Q*0V25/K/
                                           0T13)+1)/(-1/IO)
ODC18: +0A26=(+0.99)                OBC26: +0I26=(+0A26*0I25)
ODC11: +0I29=(+0.004)               ODC8 : +0R8=(+2200)
OKC10: +0=(-0I9-0I10-0I25+0I26      OKV8 : +0V8=(+0N12)
```

```
0BC8 : +0I8=(-0V8)/(-0R8*1)
0KV9 : +0V9=(+0N10-0N12)
0KC12: +0I11=(-0I8+0I9+0I25)*
(-1)
0DC13: +0R11=(+470)
0KV11: +0N11=(-0V11-0N12)*(-1)

0KV29: +0N9=(-0V29-0N11)*(-1)
0DC12: +0R12=(+39000)
0DC19: +0A30=(+0.99)
0KC13: +0I34=(-0I1-0I3-0I5-
0I12-0I30)
0BC3 : +0R3=(-0V3)/(-1*0I3)
0BC10: +0R10=(-0V10)/(-1*0I10)
2BC34: +2V34=(+0X34)
0DC25: +0C33=(+0.000001)
0DC24: +0C32=(+0.0001)
2BC35: +2V35=(+0X35)
0DC23: +0C31=(+0.0000002)
2BC14: +2I14=(+0A14/(+1-0A14)*
2I13)
2BC17: +2V17=(-2I17)/(-1*Q*
0I17/K/0T13)

2KC5 : +2I4=(+2I17+2I18)
2KV4 : +2N5=(-2V4)*(-1)
2BC13: +2V13=(-2I13)/(-1*Q*
0I13/K/0T13)
2KV31: +2V31=(+2N1-2N2)
2KV1 : +2V1=(+2N13-2N2)
2KC2 : +2I2=(+2I1-2I13+2I31)
2BC7 : +0=(-2V7+0R7*2I7)

2KC7 : +2I6=(+2I21+2I22)
2KC4 : +2I3=(-2I14-2I18-2I21)*
(-1)
2KV3 : +2N4=(-2V3+2N13)

0KV25: +0N10=(-0V25-0N12)*(-1)
0BC9 : +0I9=(-0V9)/(-0R9*1)

0KC11: +0I10=(-0I11+0I29)*(-1)
0BC11: +0V11=(+0R11*0I11)
0BC29: +0V29=(+LN(+0I29/IO+1))
/(+Q*1/K/0T13)
0KV12: +0V12=(+0N13-0N9)
0BC12: +0I12=(-0V12)/(-0R12*1)
0BC30: +0I30=(+0A30*0I29)

0KV3 : +0V3=(+0N13-0N4)
0KV10: +0V10=(+0N10-0N11)
0DC28: +0X34=(+0.0)
2KV34: +2N13=(-2V34)*(-1)
0DC29: +0W2=(-10000)
0DC22: +0X35=(+1.0)
2KV35: +2N1=(-2V35)*(-1)
2KV2 : +0=(-2V2+2N2)

2KC3 : +2I17=(+2I13+2I14)

2BC18: +2I18=(+0A18/(+1-
0A18)*2I17)
2BC4 : +2V4=(+0R4*2I4)
2KV17: +2N3=(-2V17-2N5)*(-1)

2KV13: +2N2=(-2V13-2N3)*(-1)
2BC31: +2I31=(+0W2*0C31*2V31)
2BC1 : +2I1=(-2V1)/(-0R1*1)
2BC2 : +2V2=(+0R2*2I2)
2BC22: +2I22=(+0A22/(+1-0A22)
*2I21)
2BC6 : +2V6=(+0R6*2I6)

2BC3 : +2V3=(+0R3*2I3)
2BC21: +2V21=(-2I21)/(-1*Q*
```

```
                                        0I21/K/0T13)
2KV21: +2N7=(-2V21+2N4)          2KV6 : +2N8=(-2V6+2N7)
2KV32: +2V32=(+2N8)              2BC32: +2I32=(+0W2*0C32*2V32)
2KC8 : +2I7=(+2I6-2I32)          2KV7 : +2V7=(+2N8)
0KV37: +0V37=(+0N9)              0KC9 : +0I37=(+0I12-0I26-0I29
                                        +0I30)
0BC37: +0R37=(-0V37)/(-1*0I37)   1BC34: +1V34=(+0X34)
1KV34: +1N13=(-1V34)*(-1)        0DC27: +0W1=(+10000)
1BC35: +1V35=(+0X35)             1KV35: +1N1=(-1V35)*(-1)
1KV2 : +0=(-1V2+1N2)             1BC14: +1I14=(+0A14/(+1-0A14)
                                        *1I13)
1KC3 : +1I17=(+1I13+1I14)        1BC17: +1V17=(-1I17)/(-1*Q*0I17
                                        /K/0T13)
1BC18: +1I18=(+0A18/(+1-0A18)*
1I17)                            1KC5 : +1I4=(+1I17+1I18)
1BC4 : +1V4=(+0R4*1I4)           1KV4 : +1N5=(-1V4)*(-1)
1KV17: +1N3=(-1V17-1N5)*(-1)     1BC13: +1V13=(-1I13)/(-1*Q*
                                        0I13/K/0T13)
1KV13: +1N2=(-1V13-1N3)*(-1)     1KV31: +1V31=(+1N1-1N2)
1BC31: +1I31=(+0W1*0C31*1V31)    1KV1 : +1V1=(+1N13-1N2)
1BC1 : +1I1=(-1V1)/(-0R1*1)      1KC2 : +1I2=(+1I1-1I13+1I31)
1BC2 : +1V2=(+0R2*1I2)           1BC7 : +0=(-1V7+0R7*1I7)
1BC22: +1I22=(+0A22/(+1-0A22)*
1I21)                            1KC7 : +1I6=(+1I21+1I22)
1BC6 : +1V6=(+0R6*1I6)           1KC4 : +1I3=(-1I14-1I18-1I21) *(-1)
1BC3 : +1V3=(+0R3*1I3)           1KV3 : +1N4=(-1V3+1N13)
1BC21: +1V21=(-1I21)/(-1*Q*
0I21/K/0T13)                     1KV21: +1N7=(-1V21+1N4)
1KV6 : +1N8=(-1V6+1N7)           1KV32: +1V32=(+1N8)
1BC32: +1I32=(+0W1*0C32*1V32)    1KC8 : +1I7=(+1I6-1I32)
1KV7 : +1V7=(+1N8)               1BC30: +0=(-1I30+0A30/(+1-0A30)
                                        *1I29)
1BC29: +0=(-1I29+1V29*Q*
0I29/K/0T13)                     1KV11: +0=(-1V11+1N11-1N12)
1BC26: +0=(-1I26+0A26/(+1-
0A26)*1I25)                      1BC25: +0=(-1I25+1V25*Q*0I25 /K/0T13)
2BC30: +0=(-2I30+0A30/(+1-
```

```
0A30)*2I29)                          2BC29: +0=(-2I29+2V29*Q*0I29
                                            /K/0T13)
2BC26: +0=(-2I26+0A26/(+1-
0A26)*2I25)                          2BC25: +0=(-2I25+2V25*Q*0I25/K/0T13)
1KV29: +1N11=(-1V29+1N9)             2BC10: +2V10=(+0R10*2I10)
2KC6 : +2I5=(-2I22-2I33)*(-1)        2BC33: +2V33=(-2I33)/(-0W2*0C33*1)
1BC10: +1V10=(+0R10*1I10)            1KV10: +1N10=(-1V10-1N11)*(-1)
1KV25: +1N12=(-1V25+1N10)            0DC30: +2N12=(+1600)/(+1N12*1)
2KV8 : +2V8=(+2N12)                  2BC8 : +2I8=(-2V8)/(-0R8*1)
1KV8 : +1V8=(+1N12)                  1BC8 :  +1I8=(-1V8)/(-0R8*1)
1KV9 :+1V9=(+1N10-1N12)              1BC9 : +1I9=(-1V9)/(-0R9*1)
1KC10: +1I25=(-1I9-1I10)             1KC12: +1I11=(-1I8+1I9+1I25+1I26)
                                                  *(-1)
1BC11: +1V11=(+0R11*1I11)            1KC11: +1I29=(+1I10-1I11+1I30)*(-1)
1KV12: +1V12=(+1N13-1N9)             1BC12: +1I12=(-1V12)/(-0R12*1)
1KV37: +1V37=(+1N9)                  1BC37: +1I37=(-1V37)/(-0R37*1)
1KC9 : +1I33=(+1I12-1I26-1I29
-1I37)*(-1)                          1BC33: +1V33=(-1I33)/(-0W1*0C33*1)
1KV33: +1N6=(-1V33-1N9)*(-1)         1KV5 : +1V5=(+1N13-1N6)
1KC6 : +1I5=(-1I22-1I33)*(-1)        1BC5 : +0R5=(-1V5)/(-1*1I5)
2BC5 : +2V5=(+0R5*2I5)               2KV5 : +2N6=(-2V5+2N13)
2KV33: +2N9=(-2V33+2N6)              2KV12: +2V12=(+2N13-2N9)
2BC12: +2I12=(-2V12)/(-0R12*1)       2KV29: +2N11=(-2V29+2N9)
2KV10: +2N10=(-2V10-2N11)*(-1)       2KV9 : +2V9=(+2N10-2N12)
2BC9 : +2I9=(-2V9)/(-0R9*1)          2KC10: +2I25=(-2I9-2I10)
2KV25: +2V25=(+2N10-2N12)            2KV11: +2V11=(+2N11-2N12)
2BC11: +2I11=(-2V11)/(-0R11*1)       2KC12: +2I26=(-2I8+2I9+2I11+2I25)
                                                  *(-1)
2KV37: +2V37=(+2N9)                  2BC37: +2I37=(-2V37)/(-0R37*1)
2KC9 : +2I29=(+2I12-2I26+2I33-
2I37)                                2KC11: +2I30=(+2I10-2I11+2I29)*(-1)
0BC5 : +0V5=(+0R5*0I5)               0KV5 : +0N6=(-0V5+0N13)
0KV14: +0V14=(+0N4-0N2)              0KV18: +0V18=(+0N4-0N3)
0KV22: +0V22=(+0N6-0N4)              0KV26: +0V26=(+0N9-0N10)
0KV30: +0V30=(+0N13-0N9              0DC21: +0E35=(+0.0)
0BC35: +0V35=(+0E35)                 0KV35: +0N1=(-0V35)*(-1)
0KV36: +0V36=(+0N1)                  0DC26: +0R36=(+10000)
```

```
0BC36: +0I36=(-0V36)/(-0R36*1)  0KC1 : +0I35=(-0I36)
1KC13: +1I34=(-1I1-1I3-1I5-
1I12-1I30)                      1KV14: +1V14=(+1N4-1N3)
1KV18:  +1V18=(+1N4-1N5)        1KV22: +1V22=(+1N6-1N7)
1KV26: +1V26=(+1N9-1N12)        1KV30: +1V30=(+1N13-1N11)
1KV36: +1V36=(+1N1)             1BC36: +1I36=(-1V36)/(-0R36*1)
1KC1 : +1I35=(-1I31-1I36)       2KC13: +2I34=(-2I1-2I3-2I5-2I12-2I30)
2KV14: +2V14=(+2N4-2N3)         2KV18: +2V18=(+2N4-2N5)
2KV22: +2V22=(+2N6-2N7)         2KV26: +2V26=(+2N9-2N12)
2KV30: +2V30=(+2N13-2N11)       2KV36: +2V36=(+2N1)
2BC36: +2I36=(-2V36)/(-0R36*1) 2KC1 : +2I35=(-2I31-2I36)
```

The results of the solution of the equations are listed below. The values are represented by two numbers, the first number is the real part and the second number is the imaginary part of the value.

```
0N 13    .15000E+02 0.          0N  2    .31813E+01 0.
0V  1    .11819E+02 0.          0I  1    .31942E-04 0.
0V  2    .31813E+01 0.          0I  2    .31813E-04 0.
0N  4    .39000E+01 0.          0V  3    .11100E+02 0.
0I  3    .12979E-02 0.          0N  5    .23346E+01 0.
0V  4    .23346E+01 0.          0I  4    .12970E-02 0.
0N  6    .85532E+01  .11880E-03 0V  5    .64468E+01 -.11880E-03
0I  5    .10252E-03 0.          0N  7    .34825E+01 0.
0N  8    .34172E+01 0.          0V  6    .65264E-01 0.
0I  6    .10355E-03 0.          0V  7    .34172E+01 0.
0I  7    .10355E-03 0.          0N 12    .88024E+01 0.
0V  8    .88024E+01 0.          0I  8    .40011E-02 0.
0N 10    .91024E+01 0.          0V  9    .30000E+00 0.
0I  9    .30000E-03 0.          0N 11    .10541E+02 0.
0V 10   -.14390E+01 0.          0I 10   -.30001E-03 0.
0V 11    .17390E+01 0.          0I 11    .37000E-02 0.
0N  9    .11053E+02 0.          0V 12    .39467E+01 0.
0I 12    .10120E-03 0.          0N  3    .28175E+01 0.
0V 13    .36381E+00 0.          0I 13    .12970E-04 0.
0V 14    .71872E+00 0.          0I 14    .12840E-04 0.
0V 17    .48284E+00 0.          0I 17    .12970E-02 0.
```

OV 18	.10825E+01	0.	OI 18	.12840E-02	0.
OV 21	.41751E+00	0.	OI 21	.10355E-03	0.
OV 22	.46532E+01	.11880E-03	OI 22	.10252E-03	0.
OV 25	.30000E+00	0.	OI 25	.10983E-05	0.
OV 26	.19509E+01	0.	OI 26	.10873E-05	0.
OV 29	.51195E+00	0.	OI 29	.40000E-02	0.
OV 30	.39467E+01	0.	OI 30	.39600E-02	0.
OV 34	.15000E+02	0.	OI 34	-.54936E-02	0.
ON 1	0.	0.	OV 35	0.	0.
OI 35	0.	0.	OV 36	0.	0.
OI 36	0.	0.	OV 37	.11053E+02	0.
OI 37	.60109E-04	0.	IN 13	0.	0.
IN 2	.99996E+00	.63789E-02	IV 1	-.99996E+00	-.63789E-02
II 1	-.27026E-05	-.17240E-07	IV 2	.99996E+00	.63789E-02
II 2	.99996E-05	.63789E-07	IN 4	-.41838E+01	-.25901E-01
IV 3	.41838E+01	.25901E-01	II 3	.48921E-03	.30286E-05
IN 5	.99974E+00	.63775E-02	IV 4	.99974E+00	.63775E-02
II 4	.55541E-03	.35430E-05	IN 6	.46060E+02	-.15872E+00
IV 5	-.46060E+02	.15872E+00	II 5	-.73244E-03	.25104E-05
IN 7	-.41673E+01	-.25773E-01	IN 8	-.51611E-04	.66146E-02
IV 6	-.41672E+01	-.32387E-01	II 6	-.66146E-02	-.51408E-04
IV 7	-.51611E-04	.66146E-02	II 7	.40933E-08	.20251E-06
IN 12	.39998E+02	.36728E+00	IV 8	.39998E+02	.36728E+00
II 8	.18181E-01	.16694E-03	IN 10	.41008E+02	.37654E+00
IV 9	.10092E+01	.92665E-02	II 9	.10092E-02	.92665E-05
IN 11	.46054E+02	.42288E+00	IV 10	-.50462E+01	-.46335E-01
II 10	-.10521E-02	-.96625E-05	IV 11	.60553E+01	.55602E-01
II 11	.12884E-01	.11830E-03	IN 9	.46055E+02	.42298E+00
IV 12	-.46055E+02	-.42289E+00	II 12	-.11809E-02	-.10843E-04
IN 3	.99985E+00	.63782E-02	IV 13	.11068E-03	.70606E-06
II 13	.55541E-07	.35430E-09	IV 14	-.51836E+01	-.32279E-01
II 14	.54986E-05	.35076E-07	IV 17	.11068E-03	.70606E-06
II 17	.55541E-05	.35430E-07	IV 18	-.51835E+01	-.32278E-01
II 18	.54986E-03	.35076E-05	IV 21	-.16510E-01	-.12832E-03
II 21	-.66146E-04	-.51408E-06	IV 22	.50227E+02	-.13294E+00
II 22	-.65485E-02	-.50894E-04	IV 25	.10092E+01	.92686E-02
II 25	.42882E-04	.39376E-06	IV 26	.60561E+01	.55705E-01

1I 26	.42453E-02	.38991E-04	1V 29	.90049E-03	.82705E-05
1I 29	.13936E-03	.12796E-05	1V 30	-.46054E+02	-.42288E+00
1I 30	.13796E-01	.12671E-03	1N 1	.10000E+01	0.
1V 31	.40692E-04	-.63789E-02	1I 31	.12758E-04	.81383E-07
1V 32	-.51611E-04	.66146E-02	1I 32	-.66146E-02	-.51611E-04
1V 33	.53405E-02	-.58161E+00	1I 33	.58161E-02	.53405E-04
1V 34	0.	0.	1I 34	-.12370E-01	-.12139E-03
1V 35	.10000E+01	0.	1I 35	-.11276E-03	-.81383E-07
1V 36	.10000E+01	0.	1I 36	.10000E-03	0.
1V 37	.46055E+02	.42289E+00	1I 37	.25045E-03	.22997E-05
0R 1	.37000E+06	0.	0R 2	.10000E+06	0.
0R 3	.85522E+04	0.	0R 4	.18000E+04	0.
0R 5	.62886E+05	-.11588E+01	0R 6	.63000E+03	0.
0R 7	.33000E+05	0.	0R 8	.22000E+04	0.
0R 9	.10000E+04	0.	0R 10	.47965E+04	0.
0R 11	.47000E+03	0.	0R 12	.39000E+05	0.
0T 13	.30000E+03	0.	0A 14	.99000E+00	0.
0A 18	.99000E+00	0.	0A 22	.99000E+00	0.
0A 26	.99000E+00	0.	0A 30	.99000E+00	0.
0E 34	.15000E+02	0.	0E 35	0.	0.
0R 36	.10000E+05	0.	0R 37	.18389E+06	0.
0C 31	.20000E-06	0.	0W 1	0.	.10000E+05
0C 32	.10000E-03	0.	0C 33	.10000E-05	0.
0X 34	0.	0.	0X 35	.10000E+01	0.
0W 2	0.	-.10000E+05	2N 13	0.	0.
2N 2	.99996E+00	-.63789E-02	2V 1	-.99996E+00	.63789E-02
2I 1	-.27026E-05	.17240E-07	2V 2	.99996E+00	-.63789E-02
2I 2	.99996E-05	-.63789E-07	2N 4	-.41838E+01	.25901E-01
2V 3	.41838E+01	-.25901E-01	2I 3	.48921E-03	-.30286E-05
2N 5	.99974E+00	-.63775E-02	2V 4	.99974E+00	-.63775E-02
2I 4	.55541E-03	-.35430E-05	2N 6	.46062E+02	.15691E+00
2V 5	-.46062E+02	-.15691E+00	2I 5	-.73247E-03	-.25086E-05
2N 7	-.41673E+01	.25773E-01	2N 8	-.51660E-04	-.66147E-02
2V 6	-.41672E+01	.32387E-01	2I 6	-.66146E-02	.51408E-04
2V 7	-.51660E-04	-.66147E-02	2I 7	.36813E-07	-.25145E-06
2N 12	.39998E+02	-.36728E+00	2V 8	.39998E+02	-.36728E+00
2I 8	.18181E-01	-.16694E-03	2N 10	.41009E+02	-.37836E+00

2V 9	.10111E+01	-.11079E-01	2I 9	.10111E-02	-.11079E-04
2N 11	.46056E+02	-.42469E+00	2V 10	-.50461E+01	.46334E-01
2I 10	-.10521E-02	.96625E-05	2V 11	.60573E+01	-.57413E-01
2I 11	.12888E-01	-.12216E-03	2N 9	.46056E+02	-.42470E+00
2V 12	-.46056E+02	.42470E+00	2I 12	-.11809E-02	.10890E-04
2N 3	.99985E+00	-.63782E-02	2V 13	.11068E-03	-.70606E-06
2I 13	.55541E-07	-.35430E-09	2V 14	-.51836E+01	.32279E-01
2I 14	.54986E-05	-.35076E-07	2V 17	.11068E-03	-.70606E-06
2I 17	.55541E-05	-.35430E-07	2V 18	-.51835E+01	.32278E-01
2I 18	.54986E-03	-.35076E-05	2V 21	-.16510E-01	.12832E-03
2I 21	-.66146E-04	.51408E-06	2V 22	.50229E+02	.13113E+00
2I 22	-.65485E-02	.50894E-04	2V 25	.10111E+01	-.11079E-01
2I 25	.40905E-04	.14193E-05	2V 26	.60582E+01	-.57421E-01
2I 26	.42411E-02	-.35128E-04	2V 29	.90049E-03	-.82705E-05
2I 29	.14350E-03	-.50756E-05	2V 30	-.46056E+02	.42469E+00
2I 30	.13796E-01	-.12674E-03	2N 1	.10000E+01	0.
2V 31	.40692E-04	.63789E-02	2I 31	.12758E-04	-.81383E-07
2V 32	-.51660E-04	-.66147E-02	2I 32	-.66147E-02	.51660E-04
2V 33	.53403E-02	.58160E+00	2I 33	.58160E-02	-.53417E-04
2V 34	0.	0.	2I 34	-.12369E-01	.12137E-03
2V 35	.10000E+01	0.	2I 35	-.11276E-03	.81383E-07
2V 36	.10000E+01	0.	2I 36	.10000E-03	0.
2V 37	.46056E+02	-.42470E+00	2I 37	.25046E-03	-.23095E-05

7.2 Fault location in an amplifier

To demonstrate fault location we use the circuit shown in Fig. 7.3. The parameters which can be faulty are:

R_2, R_3, R_4, R_5, R_6, R_7, A_{10}, A_{14}.

The measurable quantities are the nodal voltages:

N_2, N_4, N_5, N_6,

and the source currents: I_1, I_8.

The matrix G is shown in Fig. 7.4. If we want to compute all the parameters with the same relative accuracy and we assume that all the variables can be measured with the same relative accuracy, the sensitivities are all normalized to values 1 for parameters, voltages and currents. This normalization does not affect the linear dependence of the rows and columns.

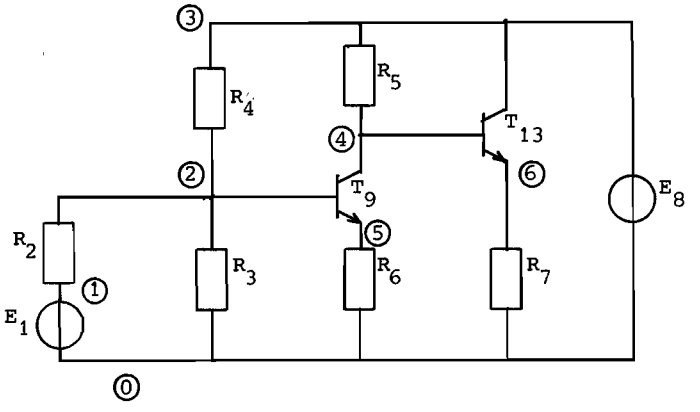


Figure 7.3. circuit for fault location

	N_2	N_4	N_5	N_6	I_1	I_8
R_2	-9.5E-3	6.9E-2	-6.2E-2	7.5E-2	-0.5	5.4E-2
R_3	4.1E-1	-3.0	2.7	-3.2	-21	-2.3
R_4	-4.2E-1	3.0	-2.7	3.3	21	2.3
R_5	0	-1.2	0	-1.3	0	-1.0
R_6	8.0E-3	7.2E-1	3.6E-1	7.7E-1	-4.1E-1	5.0E-1
R_7	0	7.2E-2	0	8.2E-2	0	-7.5E-1
A_{10}	1.1	-9.4	7.4	-10	-5.9	-7.3
A_{14}	0	7.2	0	7.6	0	6.4

Figure 7.4. The sensitivity matrix

Because of the fact that there are eight parameters and six measurable voltages and currents, at least two computations are to be made to detect the faulty parameter (we assume that only one parameter is faulty). So in each computation we need to incorporate only four unknown parameters.

By searching for the best conditioned submatrix in G we find:

- parameters : A_{10}, A_{14}, R_{07}, R_{06}.
-measurements: I_{01}, N_{06}, I_{08}, N_{05}.

For finding the parameters and measurements for the second computation we first delete the rows in G associated with the parameters:

A_{10}, A_{14}, R_{07}, R_{06}.
For the second computation we find:

- parameters : R_{04}, R_{05}, R_{02}, R_{03}
-measurements: I_{01}, N_{06}, N_{05}, I_{08}.

(note that the measurements are the same as in the first case).
The nominal values for the parameters are:

R_{02} = 1000Ω R_{06} = 46Ω
R_{03} = 1200Ω R_{07} = 750Ω
R_{04} = 32000Ω A_{10} =0.99
R_{05} = 6400Ω A_{14} =0.99

In the first computation three parameters, R_{07}, A_{10} and A_{14} are computed to be faulty, so we must conclude that the assumption that R_{04}, R_{05}, R_{03} and R_{02} are correct is wrong. The second assumption reveals R_{03} as being the faulty component with a value of 1000Ω instead of 1200Ω.

8. CONCLUSIONS

In this thesis a new approach for interactive design for electronic circuits is presented. It is shown that the fault location problem is a closely related problem, and large parts of the proposed interactive design program can be used to solve this problem.
A number of observations can be made:

1. By treating design constraint equations in exactly the same way as the equations describing the circuit, an efficient way of determining element values in circuits is obtained.

2. By using the the same concept for the fault location problem, an easy way is found to handle multiple faults in a circuit, without excessive computation time.

3. It is valuable for a designer to have information about the solvability of the set of equations describing the design problem. With this information he is able to identify the parts of the circuit being "overdefined", and in this way having an indication how to change the formulation of the design problem.

4. The computation time for solving the set of equations describing the design problem is reduced by reordering the equations in such a way that large parts of the equation set can be solved by back substitution.

5. By using design constraint equations prescribing voltages and currents in the circuit, the equation set falls apart into a large number of small strong components, being advantageously for the computation time and convergence of the Newton Raphson method.

I am grateful for the fruitful discussions with Professor J.A.G. Jess, who also gave me the opportunity to write this thesis.

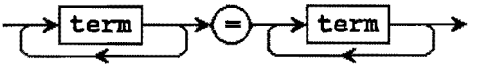
9. APPENDIX A

The syntax of an equation is presented in a BNF notation.

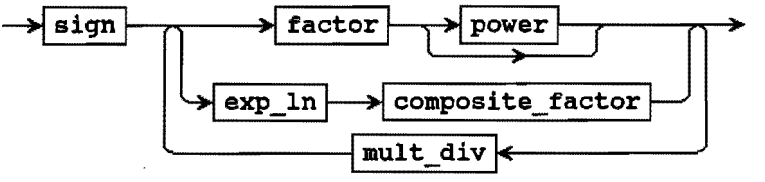
```
<equation>          ::= {<term> }+ "=" {<term> }+.

<term>              ::=  <sign> [ ((<factor> [<power>]) |
                                   <exp_ln> <composite_factor>)
                                   <mult_div>)+.

<factor>            ::=  <constant> |
                         <variable> |
                         <composite_factor>.

<composite_factor>::="(" [ <sign> [ (<variable> |
                                      <constant>
                                     ) [<power>]
                                        <mult_div> )+ )+")".

<power>             ::= "**" <sign> <constant>.

<sign>              ::=  "+" |
                         "-".

<mult_div>          ::=  "*" |
                         "/".

<exp_ln>            ::=  "exp"|
                         "ln".
```

Below this syntax is presented in a more visual way, which should clarify the meaning of the various meta symbols.

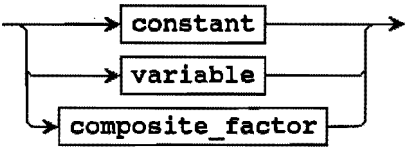
equation :



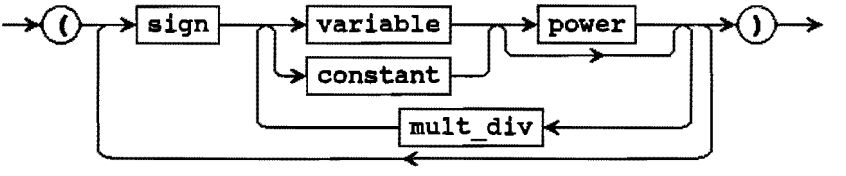
term :



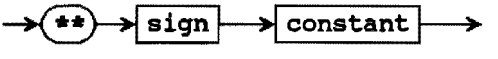
factor :



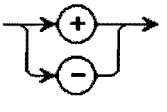
composite_factor :



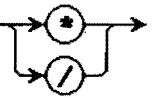
power :



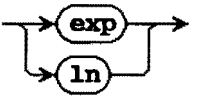
sign :



mult_div :



exp_ln :



10. APPENDIX B

INP: The description and the design constraint equations are read and checked for syntax errors. If there exist no syntax errors, the incidence matrix is generated.

DEC: The maximal matching is computed, and the strong components are determined.

SHO: With SHO it is possible to display the incidence matrix, or parts of the incidence matrix. It is also possible to obtain information about in which strong component a variable or equation occurs.

RES: The command RES calls the routines, necessary to compute the minimal essential set of a strong component.

FOR, GEN: FOR activates the formula manipulator. With GEN the formulas are generated in a standard form.

FOR, EXP: With EXP the matching variables are written explicitly.

BER, SUF: BER activates the part of the program to solve the equations. SUF converts the formulas to suffix notation, and generates the derivatives of the equations.

BER, COM: With COM the equations are actually solved.

BER, PRI: PRI prints the results.

11. APPENDIX C

The circuit and the design constraint equations are described in the file circ/dat.
The file consists of six parts:

1. A circuit name consisting of two characters.
2. The reference node (>-1, <51).
3. Lines describing the topology of the circuit.
4. A line starting with a "f", indicating the end of the description of the circuit.
5. Lines describing the design constraint equations.
6. A line starting with a "f", indication the end of the file.

Each line starting with a "#" is seen as comment.
In the topology part three kinds of lines are permitted:

1. Stringlength= 7.
   The first two characters give the branch number i of the element. $(0 < i < 100)$.
   The third character indicates the type of the branch:
   "r": resistor.
   "d": diode.
   "e": controlled voltage source.
   "j": controlled current source. Character four and five give the positive node p of the element. $(0 < p < 51)$
   Character six and seven give the negative node q of the element. $(0 < q < 51)$
2. Stringlength= 11. (dependent sources).
   The first seven characters describe an element of type "e" or "j".
   Character eight gives the type of the controlling quantity ("v" or "i" for voltage or current).
   Character nine and ten give the number p of the controlling quantity $(0 < p < 100)$.
   Character ten gives the the name of the parameter describing

the controlled source.

3. Stringlength = 11. Transistor.
   The first two characters give the branch number i of the transistor. $(0 < i < 100)$.
   The third character is a "t" for transistor.
   Character four and five give the node number c of the collector $(0 < c < 51)$.
   Character six and seven give the node number e of the emitter $(0 < e < 51)$.
   Character eight and nine give the node number b of the base $(0 < b < 51)$.
   Character ten indicates the type of the transistor ("p" for pnp and "n" for npn).
   Character eleven is a "h" or a "f". A "h" indicates a half Ebers Moll model, containing two branches. A "f" indicates a full Ebers Moll model, containing four branches.

The syntax of a design constraint equation is:
"DC" <number> "(" <variables incident with the design constraint equation> ")=" <design constraint equation>.

1. <number> consists of two characters.

2. The variables in the list of incident variables are separated by ",". The following variables may occur:

length	elements	description
1	t	temperature
3	v_k, i_k, n_k, p_k	voltage, current, node voltage or parameter.
7	"d" <x> "/t"	$x \in \{v_k, i_k, n_k\}$ sensitivity
9	"d" <x> "/d" <y>	$x \in \{v_k, i_k, n_k\}$ $y \in \{v_k, i_k, p_k\}$ sensitivity

3. The design constraint equation has to obey the syntax describe in appendix A:

12. APPENDIX D

Each variable name consists of four fields: variable name: <*a*><*n*><*m*><*ii*>

<*a*>: indicates the number of the AC system. If this number is zero it may be skipped. (one digit).

<*ii*>: indicates the branch or node number to which the equation or variable refers to. (two digits).

<*n*> <*m*>: For these two entries are a number of possibilities. They are listed below. For columns (variables) we have the following possibilities:

<*n*>	<*m*>	description
n		voltage of node number *ii*
v		voltage across element *ii*
i		current through element *ii*
<p>		p is a variable indicating the parameter of element *ii*
n	<*j*>	voltage of node *ii* in delta system *j*
v	<*j*>	voltage across element *ii* in delta system *j*
i	<*j*>	current through element *ii* in delta system *j*
<*j*>	v	transposed kirchhoff voltage law equation of element *ii* in adjoint system *j*.
<*j*>	c	transposed kirchhoff current law equation of element *ii* in adjoint system *j*.
<*j*>	b	transposed branch constraint equation of element *ii* in adjoint system *j*.
d	<*j*>	sensitivity in delta system *j*,the response variable is a branch voltage
e	<*j*>	sensitivity in delta system *j*,the response variable is a branch current
f	<*j*>	sensitivity in delta system *j*,the response variable is a node voltage
<*j*>	p	sensitivity in adjoint system, exiting

entity is a parameter

<j> q sensitivity in adjoint system, exiting entity is a voltage of a branch

<j> r sensitivity in adjoint system, exiting entity is a current through a branch

For the rows (equations) are the following possibilities:

<n>	<m>	description
k	v *ii*	kirchhoff voltage law equation of element *ii*
k	c	kirchhoff current law equation of node *ii*
b	c	branch constraint equation of element *ii*
d	c	design constraint equation of element *ii*
v	<j>	kirchhoff voltage law equation in deltasystem *j*
c	<j>	kirchhoff current law equation of node *ii* in delta system *j*
b	<j>	branch constraint equation of element *ii* in delta system *j*
<j>	n	transposed voltage of node *ii* in adjoint system *j*
<j>	v	transposed voltage of element *ii* in adjoint system *j*
<j>	i	transposed current through element *ii* in adjoint system *j*
d	<j>	row with sensitivities in delta system *j*, responding variable is the voltage of element *ii*
e	<j>	row with sensitivities in delta system *j*, responding variable is the current through element *ii*
f	<j>	row with sensitivities in delta system *j*, responding variable is the voltage of node *ii*
<j>	p	row with sensitivities in adjoint system *j*, exiting variable is parameter *ii*
<j>	q	row with sensitivities in adjoint system *j*,

exiting variable is the voltage of element *ii*

<*j*> r row with sensitivities in adjoint system *j*, exiting variable is the current through element *ii*

13. REFERENCES

Bedrosian, S.D. and J.H. Lee
GRAPH THEORETIC ASPECTS OF ANALOG FAULT DIAGNOSIS.
In: Proc. 17th Annual Allerton Conf. on Communication, Control, and Computing, Monticello, Ill., 10-12 Oct. 1979. Co-chairmen: J.B. Cruz, Jr. and F.P. Preparata.
Department of Electrical Engineering and The Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, 1979.
P. 164-171.

Biernacki, R.M. and J.W. Bandler
FAULT LOCATION OF ANALOG CIRCUITS.
In: Proc. 13th IEEE Int. Symp. on Circuits and Systems, Houston, Texas, 28-30 April 1980.
New York: IEEE, 1980. P. 1078-1081.

Cheung, L.K. and E.S. Kuh
THE BORDERED TRIANGULAR MATRIX AND MINIMUM ESSENTIAL SETS OF A DIGRAPH.
IEEE Trans. Circuits & Syst., Vol. CAS-21(1974), p. 633-639.

de Kleer, J. and G.J. Sussman
PROPAGATION OF CONSTRAINTS APPLIED TO CIRCUIT SYNTHESIS.
Int. J. Circuit Theory & Appl., Vol. 8(1980), p. 127-144.

Desoer, C.H. and E.S. Kuh
BASIC CIRCUIT THEORY.
New York: McGraw-Hill, 1969.

Donald, J. and J. Elwin, R. Hager, P. Salamon
A BAD EXAMPLE FOR THE MINIMUM FEEDBACK VERTEX SET PROBLEM.
IEEE Trans. Circuits & Syst., Vol. CAS-32(1985), p. 491-493.

Duhamel, P. and J.-C. Rault
AUTOMATIC TEST GENERATION TECHNIQUES FOR ANALOG CIRCUITS AND SYSTEMS: A review.
IEEE Trans. Circuits & Syst., Vol. CAS-26(1979), p. 411-440.

Dulmage, A.L. and N.S. Mendelsohn
COVERINGS ON BIPARTITE GRAPHS.
Can. J. Math., Vol. 10(1958), p. 517-534.

Dulmage, A.L. and N.S. Mendelsohn
TWO ALGORITHMS FOR BIPARTITE GRAPHS.
J. Soc. Ind. & Appl. Math., Vol. 11(1963), p. 183-194.

Hachtel, G.D. and R.K. Brayton, F.G. Gustavson
THE SPARSE TABLEAU APPROACH TO NETWORK ANALYSIS AND DESIGN.
IEEE Trans. Circuit Theory, Vol. CT-18(1971), p. 101-113.

Harary, F.
GRAPH THEORY.
Reading, Mass.: Addison-Wesley, 1969.
Addison-Wesley Series in Mathematics

Hildebrand, F.B.
INTRODUCTION TO NUMERICAL ANALYSIS.
New York: McGraw-Hill, 1956.
International Series in Pure and Applied Mathematics

Hopcroft, J.E. and R.M. Karp
AN $n^{5/2}$ ALGORITHM FOR MAXIMUM MATCHINGS IN BIPARTITE GRAPHS.
SIAM J. Comput., Vol. 2(1973), p. 225-231.

Hostetter, G.H.
FUNDAMENTALS OF NETWORK ANALYSIS.
New York: Harper & Row, 1980.

Jennings, A.
MATRIX COMPUTATION FOR ENGINEERS AND SCIENTISTS.
London: Wiley, 1977.

Johnson, D.M. and A.L. Dulmage, N.S. Mendelsohn
CONNECTIVITY AND REDUCEBILITY OF GRAPHS.
Can. J. Math., Vol. 14(1962), p. 529-539.

Kantorovich, L.V. and G.P. Akilov
NEWTON'S METHOD.
A chapter in: Vainberg, M.M., VARIATIONAL METHODS FOR THE STUDY OF NONLINEAR OPERATORS. Translated from Russian edition (Moscow, 1956) and supplemented by A. Feinstein.
San Francisco: Holden-Day, 1964.
Holden-Day Series in Mathematical Physics. P. 258-298.

Karp, R.M.
REDUCIBILITY AMONG COMBINATORIAL PROBLEMS.
In: Complexity of Computer Computations. Proc. Symp., Yorktown Heights, N.Y., 20-22 March 1972. Ed. by R.E. Miller et al.
The IBM Research Symposia Series.
New York: Plenum, 1972. P. 85-103.

Kevorkian, A.K.
ON BORDERED TRIANGULAR OR LOWER N FORMS OF AN IRREDUCIBLE MATRIX.
IEEE Trans. Circuits & Syst., Vol. CAS-23(1976), p. 621-624.

Kozemchak, E.B. and M.A. Murray-Lasso
COMPUTER-AIDED CIRCUIT DESIGN BY SINGULAR IMBEDDING.
Bell Syst. Tech. J., Vol. 48(1969), p. 275-315.

Lee, J.H. and S.D. Bedrosian
FAULT ISOLATION ALGORITHM FOR ANALOG ELECTRONIC SYSTEMS USING THE FUZZY CONCEPT.
IEEE Trans. Circuits & Syst., Vol. CAS-26(1979), p. 518-522.

Lin, P.M. and Y.S. Elcherif
ANALOGUE CIRCUITS FAULT DICTIONARY - NEW APPROACHES AND IMPLEMENTATION.
Int. J. Circuit Theory & Appl., Vol. 13(1985), p. 149-172.

Liu, Ruey-Wen and V. Visvanathan
SEQUENTIALLY LINEAR FAULT DIAGNOSIS. Part 1: Theory.
IEEE Trans. Circuits & Syst., Vol. CAS-26(1979), p. 490-495.

McKeeman, W.M. and J.J. Horning, D.B. Wortman
A COMPILER GENERATOR.
Englewood Cliffs, N.J.: Prentice-Hall, 1970.
Prentice-Hall Series in Automatic Computation

Navid, N. and A.N. Willson, Jr.
A THEORY AND AN ALGORITHM FOR ANALOG CIRCUIT FAULT DIAGNOSIS.
IEEE Trans. Circuits & Syst., Vol. CAS-26(1979), p. 440-457.

Nordholt, E.H.
THE DESIGN OF HIGH-PERFORMANCE NEGATIVE-FEEDBACK AMPLIFIERS.
Amsterdam: Elsevier, 1983.
Studies in Electrical and Electronic Engineering, Vol. 7.
Revised and reviewed version of the Ph.D. Thesis, Delft University of Technology, 1980.

Saeks, R. and S.R. Liberty (eds.)
RATIONAL FAULT ANALYSIS. Proc. Symp., Texas Tech University, Lubbock, Texas, 19-20 Aug. 1974.
New York: Marcel Dekker, 1977.
Electrical Engineering and Electronics, Vol. 1.

Salama, A.E. and J.A. Starzyk, J.W. Bandler
A UNIFIED DECOMPOSITION APPROACH FOR FAULT LOCATION IN LARGE ANALOG CIRCUITS.
IEEE Trans. Circuits & Syst., Vol. CAS-31(1984), p. 609-622.

Smith, Jr., G.W. and R.B. Walford
THE IDENTIFICATION OF A MINIMAL FEEDBACK VERTEX SET OF A DIRECTED GRAPH.
IEEE Trans. Circuits & Syst., Vol. CAS-22(1975), p. 9-15.

Sussman, G.J. and R.M. Stallman
HEURISTIC TECHNIQUES IN COMPUTER-AIDED CIRCUIT ANALYSIS.
IEEE Trans. Circuits & Syst., Vol. CAS-22(1975), p. 857-865.

Taub, H. and D. Schilling
DIGITAL INTEGRATED ELECTRONICS.
New York: McGraw-Hill, 1977.

Tarjan, R.
DEPTH-FIRST SEARCH AND LINEAR GRAPH ALGORITHMS.
SIAM J. Comput., Vol. 1(1972), p. 146-160.

Trick, T.N. and W. Mayeda, A.A. Sakla
CALCULATION OF PARAMETER VALUES FROM NODE VOLTAGE MEASUREMENTS.
IEEE Trans. Circuits & Syst., Vol. CAS-26(1979), p. 466-474.

Trouborst, P.M. and J.A.G. Jess
THE TRANSFORMATION TO A LOWER BORDERED TRIANGULAR FORM WITH THE APPLICATION OF ELIMINATION STEPS.
In: Proc. 12th Int. Symp. on Circuits and Systems, Tokyo, 17-19 July 1979.
New York: IEEE, 1979. P. 108-111.

Trouborst, P.M.
DYNAMIC SPARSE MATRIX CODE AS AN AUTOMATIC APPROACH TO MACROMODELING.
Ph.D. Thesis. Eindhoven University of Technology, 1981.

Een interactief ontwerp en fouten localisatie hulpmiddel voor electronische schakelingen

Het onderzoek bestaat uit het realiseren van een programma pakket, waarmee het mogelijk is om interactief een analoge niet lineaire electronische schakeling met een gegeven topologie, te analiseren en te ontwerpen. Hierbij worden de eisen die de ontwerper aan het systeem stelt op gelijke wijze behandeld als de vergelijkingen die het circuit beschrijven, waardoor er een set vergelijkingen ontstaat die geanaliseerd en opgelost moet worden. Hierbij wordt gebruik gemaakt van incidentie matrices en grafen, die afgeleid zijn van de vergelijkingen. In deze grafen worden "matchings" en "sterke componenten" bepaald. Matchings geven een indicatie over de oplosbaarheid van het stelsel vergelijkingen, en sterke componenten geven een leidraad voor de volgorde waarin de vergelijkingen opgelost moeten worden. Naast de bovengenoemde technieken worden er tevens formule manipulatie technieken en methoden voor het oplossen van een stelsel niet lineaire vergelijkingen (Newton Raphson) gebruikt.
Naast het gebruik als ontwerp tool, kan het systeem tevens gebruikt worden als fouten localisatie hulpmiddel. Beide problemen zijn namelijk conceptueel gelijk aan elkaar:

Het ontwerp probleem kan omschreven worden als:
Gegeven een aantal ontwerpeisen, bereken (alle) componentwaarden in de voorgestelde schakeling.

Het fouten localisatie probleem kan omschreven worden als:
Gegeven een aantal gemeten responsies van de schakeling, bereken (alle) componentwaarden van de te testen schakeling.

Naast het oplossen van de vergelijkingen behorende bij de boven omschreven problemen, kan het beschreven programma pakket ook een set adequate metingen berekenen, waarmee de schakeling getest kan worden. Hierbij wordt rekening gehouden met gevoeligheden van componenten ten opzichte van metingen.

Curriculum vitae Ir. J.F.M. Theeuwen.

Frans Theeuwen was born in Geleen, The Netherlands, in 1954. He received the M.Sc degree in electrical engineering in 1979 from the Eindhoven University of Technology, The Netherlands. He is currently working at the Eindhoven University of Technology in the computer-aided design group. His main interests are in functional design and fault location in electronic circuits, automatic layout generation and automatic design of logic circuits.

Stellingen

1. Een van de vaak onterecht aangehaalde oorzaken van een niet goed functionerende geautomatiseerde administratie is: "De computer doet het niet".

2. De scheiding tussen software en hardware onderhoudstechnici voor computer systemen, wordt steeds inadequater.

3. Lettend op de (neven)effecten van het loonbeleid voor ambtenaren, en het UHD beleid, zou men moeten concluderen dat de regering niet geinteresseerd is een goed universitair onderzoek en onderwijs.

4. Het gebruik van PLA's in IC ontwerpen wordt hoofdzakelijk veroorzaakt door een gebrek aan software om "random logic" op een "optimale manier" te implementeren.

5. Het zou een zegen voor het onderwijs in de informatica zijn als de programmeertaal BASIC, veel gebruikt in de home computer sfeer, in populariteit zou inboeten ten voordele van een krachtigere en leerzamere taal zoals bijvoorbeeld Pascal.

6. De nieuwe ontwikkelingen op het gebied van het testen van digitale schakelingen (signature analysis, build in test), zullen binnen enkele jaren veel test- en meetapparatuur overbodig maken.

7. Het invoeren van arbeidstijdverkorting heeft voor veel werknemers de consequentie dat hun werkzaamheden in minder tijd gedaan moeten worden, wat vaak resulteert in onbetaald overwerk.

8. Lettend op het aantal componenten van moderne mini-computers, moet men concluderen dat bedrijven veel geld verdienen aan onderhoudscontracten, die nog steeds 7 tot 15% per jaar van de nieuwprijs bedragen.