CIRCUIT AND NETWORK ANALYSIS

DR. FRANKLIN H. BRANIN, JR.
IBM CORPORATION
POUGHKEEPSIE, NEW YORK

ABSTRACT

This paper treats the problem of programming a digital computer to formulate and solve the algebraic and differential equations characterizing linear and/or nonlinear networks. certain techniques are described which have been used successfully in an experimental program for analyzing transistor switching circuits. The techniques, predicated on a matrix formulation of network equations, include: (a) derivation of all the necessary topological and parameter matrices from input data which specify the network connections and parameters, (b) solution of the nonlinear d-c problem by an iteration procedure, and (c) solution of the transient problem by numerical integration of the nonlinear differential equations separately from the linear differential equations.

* Presented at the 1962 IRE International Convention, New York (March).

D-C AND TRANSIENT ANALYSIS OF NETWORKS USING A DIGITAL COMPUTER

Franklin H. Branin, Jr.
Development Laboratory, Data Systems Division
International Business Machines Corporation
Poughkeepsie, New York

Summary

An experimental program is described for computing the d-c and transient response of transistor switching circuits of arbitrary configuration and size (up to 20 transistors) using the IBM 704 computer. One important feature of the program which is discussed is its ability to compile all the necessary equations automatically from input data describing the circuit parameters and configuration. Another is the solution of the transient problem by numerical integration of the differential equations for the linear part of the circuit separately from those describing the transistors, the output from each set of equations being used periodically as input for the other set. Considerable increase in speed of integration is obtained in this manner.

The method of d-c analysis is based on a topological-matrix formulation of the linear part of the problem, and its solution by Kron's method, followed by an iterative procedure for satisfying certain nonlinear side conditions imposed by the transistors. Although the transient analysis also uses a matrix formulation of the required differential equations, it is not based on a topological approach. However, a generalized topological-matrix formulation of the transient problem is given in an appendix.

The nature of a serious theoretical limit on the rate of integration of the network equations, is discussed since it constitutes the principal computational barrier to a rapid solution of the transient problem. An outline of the more important programming procedures involved in the topological-matrix formulation is also given.

Certain shortcomings of the program, and pitfalls to be avoided are pointed out. In particular, the importance of being able to modify or replace the transistor equivalent circuit (network model) is emphasized.

Finally, the computed responses of a four-transistor switching circuit are displayed and shown to agree well with the observed responses.

Introduction

This paper is based on the experience gained in writing an experiental program for analyzing transistor switching circuits using the IBM 704 computer. This program, called TAP for "transistor analysis program",[1,2] was developed to provide circuit-design engineers with the ability to carry out "computational experiments" to aid in understanding, as well as designing, switching circuits.

Although this objective was reached, the program has become obsolete because it was restricted to the analysis of circuits containing a certain type of diffused base transistor which is of limited interest. Consequently, the program is not being maintained nor is it being made available for outside distribution. Nevertheless, the lessons learned from writing TAP are felt to be worth sharing for the benefit of those who may be interested in writing a similar program. In addition to describing the principal features of TAP, this paper will point out the major difficulties and pitfalls encountered; it will also suggest some improvements in technique which may prove helpful. Finally, some actual and potential applications of a digital computer program of this type will be described.

TAP is capable of performing the entire d-c and transient analysis of a multi-transistor switching circuit of arbitrary configuration and size up to 20 transistors. Its most valuable feature, from the user's point of view, is its ability to compile automatically all the necessary circuit equations using a simple input card format which specifies only the parameters and connections of the circuit components. By compiling these equations for him, TAP relieves the user (ostensibly an engineer rather than a programmer) of the tedious and error-prone chore of writing all the circuit equations himself and then setting up a program to solve them. At the same time, TAP makes it convenient to modify parameters or connections in the circuit simply by changing the appropriate input cards.

It is interesting to note that the ability to compile the circuit equations from simple input data has also been included in the DYANA program recently developed by the General Motors Research Laboratories for mechanical and electrical network analysis.[3] Since this feature is particularly helpful to the user, it is strongly recommended that any general purpose network analysis programs developed in future also include this compiling facility.

The d-c analysis portion of TAP, which provides the initial values of voltage and current needed in the transient problem, takes account of the nonlinearities due to the transistors. Fortunately, it has been found possible to view this d-c problem as one of solving a linear network problem subject to an appropriate set of nonlinear side conditions. The linear network is treated by Kron's method of "interconnecting solutions"[4,5,6] while the nonlinear side conditions are satisfied by using an iterative method of successive approximations.[2]

After the initial conditions of voltage and current are obtained, the transient response of the network, excited by a ramp-function input pulse of arbitrary amplitude and duration, is calculated by numerical integration of the differential equations describing the system. This part of the computation is the most time-consuming and represents the biggest bottleneck in the entire analysis. In the attempt to speed up the numerical integration process in TAP, the linear and nonlinear sets of differential equations are integrated separately, over short intervals of time, and only periodically rejoined so as to provide a meaningful solution. The reasons for and advantages of doing this will be described below.

One extremely important aspect of the analsis of networks containing nonlinear devices is that of developing suitable network models for such devices. A network model must, of course, represent the device in question to an acceptable degree of approximation. It should also be made as simple as possible to minimize the computational burden. Both of these desiderata are fairly well satisfied by the transistor model used in TAP. However, an addition consideration should be pointed out whose importance was not recognized until after TAP had been written: namely, the ability to change the network representation of the transistor, either in part or in toto, without extensively altering the program.

Since TAP was intended for analyzing only circuits containing a certain type of transistor for which an adequate model existed,[7] the ability to alter this model was not considered important. Although provision was made to vary each parameter of the model, its basic configuration and the character of its nonlinearities were fixed and were made an integral part of the program. Accordingly, once TAP had been tested and proved practical for its intended purpose, attempts to extend it to the analysis of circuits containing other nonlinear devices were frustrated by the amount of reprogramming required to alter or replace the transistor model. This shortcoming, unfortunately, forced the program into premature obsolescence.

Although a network analysis program necessitates the invention of network models for nonlinear devices, by its very nature it also provides the means for validating these models. This is another reason for setting up the program in such a way as to facilitate changing these models easily. No recommendations can be made as to how this may be accomplished, but the importance of doing so needs to be recognized.

In order to conserve space in this paper, only the essence of the mathematical and programming techniques used in TAP will be given. Reference to published material will be made for any mathematical details omitted; but programming details that should be apparent to those skilled in the art will be omitted entirely. Although TAP was programmed for the IBM 704 computer, the techniques used are amenable to any binary computer and, with suitable modification, to decimal machines as well.

The three main sections of the paper describe the compilation process, d-c analysis, and transient analysis. A generalized formulation of the linear transient network problem is described in Appendix I. This formulation is recommended in place of that actually used in TAP. Other appendices are included which describe certain procedures used in determining and using the topological matrices.

Compilation of Input Data

The scheme used in TAP for compiling the differential equations for the transient analysis of the linear part of the network is based on a matrix formulation[8] in which the coefficient matrices are determined by inspection of the network connections rather than by means of topological matrices. This formulation, using a combination of node voltages and mesh currents as variables, establishes a simultaneous system of first order differential equations, similar in form to that described by Bashkow[9]. The d-c analysis, which was incorporated into the program at a later date, makes use of a topological matrix formulation.[2,5,6]

A disadvantage arises from formulating the transient problem in terms of differential equations only since every node voltage or mesh current must be computed in terms of a corresponding differential equation, even though an algebraic equation might have been more appropriate. This presents no problem as far as mesh currents are concerned for these are introduced only when inductors are actually present. But since each node voltage requires a differential equation to describe it, there must be a capacitive path from each node to ground in order to define this differential equation. Hence, additional ("stray") capacitances must be inserted wherever the required capacitive path is absent in the original circuit.

Admittedly, these stray capacitances do exist in actual circuits and this is why they were included in the original formulation. However, their effect may be negligible in many cases and yet by their very presence, these capacitances may slow down the numerical integration process appreciably. Therefore, it is important to formulate the transient problem with sufficient generality to admit algebraic equations, when required, as well as differential equations.

A formulation, using a topological-matrix approach, is given in Appendix I. This formulation, which extends the recent work of Bashkow,[9] suffices for the d-c analysis as well and gives directly all the initial values required by the transient problem.

The input scheme developed for TAP permits arbitrary connections and parameters to be specified both for the linear part of the network and for the transistors. Only the transistor model is fixed in its configuration and nonlinear characteristics.[1] The input information required, therefore is the following: (1) type of circuit component, such as transistor, resistor, capacitor, or inductor; (2) serial number of each component (actually required only for transistors); (3) parameter (or parameters) pertinent to each component; (4) component connections as designated by node numbers; (5) voltage and/or current sources; (6) input pulse characteristics.

This information, punched into cards, is read into the computer and compiled either in matrices or in tables, some of which are later converted to matrix form. Since the cards specifying the linear part of the circuit are read in first, with R, L, and C cards intermixed, the compilation of the tables required for the d-c analysis goes on simultaneously with that of the coefficient matrices for the transient problem.

The information from the cards specifying the transistor parameters and connections is read in, and tabulated. Since the resistors in the transistor model are linear, these data are included in the tables for the d-c analysis. The transistors are simulated by current sources driving a linear circuit in both the d-c and transient analysis.

D-C Analysis

The tables required by the matrix formulation used in the d-c analysis[5,6] are the following: (1) branch resistances (RDATA); (2) branch connections (RCON) showing initial and final node numbers for each branch; (3) voltage sources (EDATA) and (4) current sources (IDATA) in each branch. (In TAP, the IDATA table is omitted since the only current sources encountered are due to transistors. In a more general program, however, the IDATA table should be included.)

These tables are not converted into a matrix form of the network equations, but rather into the appropriate "solution matrices" in terms of which the solution may be computed directly. Actually, the voltages are computed from the nodal solution matrix (inverse of the nodal admittance matrix) while the currents are computed from the mesh solution matrix (inverse of the mesh impedance matrix). To be sure, either solution matrix would suffice for computing all the voltages and currents but it was felt that round-off errors would be reduced by the method adopted.

To handle networks of arbitrary configuration, this program requires an unambiguous procedure for identifying and specifying just the right number of meshes. Such a procedure has been developed with the aid of the well-known topological concepts of tree, link, and basic mesh.

A tree is defined as any network structure devoid of closed paths. The network tree is a tree which includes all nodes of the network and a link is any branch of the tree-complement. When a single link is connected to the tree, it forms a unique closed path called a basic mesh; each basic mesh contains but one link. Hence, if all the network branches are classified as either tree-branches or links, the basic meshes may be unambiguously identified and they are just sufficient in number.[5,6]

In computing the nodal solution matrix, a modification of Kron's method called the link-at-a-time (or LAT) algorithm is used. This algorithm, which is explained in reference 5, is a constructive method for obtaining the nodal solution matrix for the complete network by adding a link at a time to the network tree and modifying the corresponding nodal solution matrix at each suc-

cessive step. Since the nodal solution matrix for the tree is obtained without matrix inversion and since the general formula for modification of this matrix at each step is the same, the algorithm is easy to use.

To minimize round-off errors at each successive step of this algorithm, it is desirable to chose the network tree of minimum total resistance.[5] Accordingly, the first step in the d-c analysis is to sort the RDATA table in order of increasing resistance. The RCON, EDATA and IDATA tables are then rearranged to conform to this new sequence of the network branches.

Next, starting with the branch of smallest resistance, the RCON table is examined to determine if each suceeding branch does or does not form a closed path with the then-defined tree. If a closed path is formed, the branch is classed as a link; if not, it is classed as a tree branch and added to the tree. When all the branches have been examined in this manner, the tree of minimum resistance is defined. Appendix II is a description of the tree-link sorting procedure.

The RDATA, RCON, EDATA and IDATA tables are rearranged to conform to this new classification with tree-branches in one group and links in another. The next step is to convert the RCON table into the appropriate topological matrices.

Topological Matrices

Three topological matrices are required in the d-c analysis and all of them consist of the elements +1, -1, and/or 0 only. Using a binary computer such as the IBM 704, it is convenient to represent each such matrix element as a pair of binary digits (bits) in order to conserve storage -- and also to speed up certain parts of the computation. In TAP, the storage format used is 16-bit-pairs per word for each 16 elements of a column (or, sometimes, row) of a topological matrix. An alternative format, which has certain advantages, is to store the magnitude bits in one word and corresponding sign bits in an adjacent word.

By definition,[5,6] the elements of the branch-node matrix are:

a_{ij} = (+1, -1, 0) if the i-th branch is (positively, negatively, not) incident on the j-th node.

Now the i-th entry (word) of the RCON table corresponds to the i-th row of the branch-node matrix since both specify the initial (+) and final (-) node of the i-th branch. The RCON table is stored with the initial node number in the address field and the final node number in the decrement field of a single word. The task of converting this table to the bit-pair matrix format (stored column-wise) is therefore a straight-forward programming task that need not be elaborated.

Initially, the branch-node matrix includes the column corresponding to the datum or reference (ground) node, but later this column is deleted since it contains redundant information.[5,6] The matrix remaining after this deletion is then designated as the A matrix. The tree portion of this matrix, A_T, is a square matrix whose inverse may be obtained topologically rather than computationally. The inverse of A_T may be shown[5,6] to be identical with the transpose of a matrix, B_T, called the node-to-datum-path matrix of the tree. The elements of this matrix are defined as follows:

b_{ij} = (+1, -1, 0) if the i-th branch is (positively, negatively, not) included in the j-th node-to-datum path.

This B_T matrix is used in computing both the nodal solution matrix of the tree and the branch-mesh matrix described below. The procedure for determining the B_T matrix, by means of an exhaustive search of the network tree from the datum node outward, is explained in Appendix III.

The third topological matrix required in the d-c analysis is the branch-mesh matrix, designated C, whose elements are defined thus:

c_{ij} = (+1, -1, 0) if the i-th branch is (positively, negatively, not) included in the j-th basic mesh.

By adopting the convention that both the orientation and ordering of the basic meshes agree with those of the corresponding links, the link portion of the branch-mesh matrix, C_L, turns out to be a unit matrix and does not even need to be written. The tree portion C_T then, contains all the essential information about the branch mesh matrix.

Fortunately, C_T is readily computed from B_T and the transpose of A_L, the link portion of the branch-node matrix, by means of the equation[5,6]

$$C_T = -B_T A_L^t \qquad (1)$$

This computation amounts to taking the difference of two columns of B_T to get each column of C_T. Accordingly, it is easier to use the link portion of the RCON table than the A_L matrix for this purpose since the address and decrement of each RCON word tell directly which columns of B_T are to be added or subtracted. Thus, all the necessary

topological matrices may be obtained directly or indirectly from the RCON table.

Nodal Solution Matrix

The primitive impedance matrix, Z, and its inverse, the primitive admittance matrix, Y, characterize the branches of a network independently of their interconnections.[5,6] In the d-c problem, the RDATA table corresponds to a real diagonal Z matrix and its reciprocal to a real diagonal Y matrix. The nodal solution matrix, which is the inverse of the nodal admittance matrix A^tYA, is obtained by means of the LAT algorithm starting with the inverse of the nodal admittance matrix for the tree, $A_T^t Y_T A_T$. Using the relation $A_T^{-1} = B_T^t$ mentioned above, it is easily shown that

$$(A_T^t Y_T A_T)^{-1} = B_T^t Z_T B_T \qquad (2)$$

Hence, the nodal solution matrix for the tree may be calculated without any matrix inversion. Appendix IV describes how the triple matrix product $B_T^t Z_T B_T$ is computed, taking advantage of the diagonal nature of Z_T and using the compact storage format of B_T.

The LAT algorithm makes use of a recursion formula which modifies a given nodal solution matrix to take account of the addition of a single link. Assuming that the impedance of the j-th link is z_L^j and that this link is connected between the p-th and q-th nodes of the network, the modified nodal solution matrix, Z^j is given by the equation[5]

$$Z^j = Z^{j-1} - \frac{(Z^{j-1}_{\cdot p} - Z^{j-1}_{\cdot p})(Z^{j-1}_{\cdot p} - Z^{j-1}_{\cdot p})}{Z^{j-1}_{pp} + Z^{j-1}_{qq} - Z^{j-1}_{pq} - Z^{j-1}_{qp} + z^j_L} \qquad (3)$$

where $Z^{j-1}_{\cdot p}$ and $Z^{j-1}_{p\cdot}$ are the p-th column and p-th row of the previous nodal solution matrix, Z^{j-1}. Clearly $Z^0 = B_T^t Z_T B_T$.

Since all the Z^j are symmetric, only the diagonal and subdiagonal elements are computed and stored, thereby conserving both computation time and memory space. Furthermore, since the tree is chosen for minimum total resistance, the link resistances will be as large as possible, thereby minimizing the denominator of the correction term in Eq. (3) and reducing round-off errors.

Mesh Solution Matrix

Computation of the mesh-solution matrix is based on a recursion formula similar to Eq. (3). The corresponding algorithm adds one tree-branch at a time to the set of all links and modifies the mesh-solution matrix accordingly at each successive step. The starting point of this algorithm, of course, is the mesh solution matrix of the links which, fortunately, is diagonal. A discussion of this algorithm is given in reference 5.

Computation of the mesh and nodal solution matrices could, of course, have been done by the usual methods of matrix inversion and with less programming effort. However, since it was desired to evaluate the potentialities of Kron's method of interconnecting solutions, advantage was taken of the experimental nature of TAP to program and test the link-at-a-time and tree-branch-at-a-time algorithms in an actual situation. (The computational efficiency and applications of these algorithms are discussed in reference 5.)

Solution of the Nonlinear Problem

As previously stated, the nonlinearities introduced by the presence of transistors in the network are handled by imposing a set of nonlinear side conditions on the solution of the linear network problem. This is accomplished by representing the effect of the transistors on the linear network by means of current sources at the appropriate nodes and adjusting these current sources so that certain of the voltage responses satisfy the nonlinear side conditions.

The d-c portion of the equivalent circuit used in TAP to represent a nonsaturating, diffused base PNP transistor is shown in Fig. 1. The base and collector resistances, $R_{bb'}$ and $R_{cc'}$, as well as the leakage resistance R_c, are assumed to be constant and so are included as part of the linear network. The only nonlinearities, then, are due to the current sources αI_h and $(1-\alpha)I_h$ where α, the current gain factor, is assumed to be a nonlinear function of I_h. (See reference 1 for details.)

The current source I_h (hole current) is exponentially related to the emitter-base voltage V_{eb} by the diode equation

$$I_h = I_{es}\,(e^{\Lambda V_{eb'}} - 1) \qquad (4)$$

where I_{es} is the reverse saturation current and $\Lambda = q/kT = 1/.026$ volts at room temperature.

As far as the linear network is concerned, I_h can be chosen arbitrarily since this choice merely specifies the current sources αI_h and $(1-\alpha)\,I_h$. The resulting voltage responses of the linear network may then be calculated by the matrix equation[5,6]

$$e' = (A^t YA)^{-1} A^t (I - YE) \qquad (5)$$

where e' is the node-to-datum voltage vector and I and E are the current source and voltage source vectors (corresponding to the IDATA and EDATA tables.)

Since the only current sources considered in TAP are those due to transistor action, one may write the matrix relation

$$A^t I = \bar{\alpha} I_h \tag{6}$$

to define the equivalent nodal current sources depicted in Fig. 2. In Eq. (6), I_h represents the vector of I_h values for all transistors while $\bar{\alpha}$ represents a matrix whose only nonzero elements are -1, α, and $(1-\alpha)$ placed so as to assign current sources $-I_h$, αI_h and $(1-\alpha)\, I_h$ to the appropriate nodes of each transistor model.

The voltages V_{eb}' for each transistor may be computed from the node voltage relation $e_e' - e_b'$. In matrix form, this may be written as

$$V_{eb}' = He' \tag{7}$$

where H is a matrix whose only nonzero elements are +1 and -1 placed so as to perform the appropriate linear combination of the elements of e'.

Combining Eqs. (5), (6) and (7), we may write

$$V_{eb}' = \left[H(A^tYA)^{-1}\bar{\alpha}\right] I_h - H(A^tYA)^{-1} A^tYE \tag{8}$$

which expresses the vector V_{eb}' directly as a function of I_h. The trick now is to choose I_h so that both Eq. (8) and Eq. (5) are satisfied.

The nature of this problem is shown graphically in Fig. 3 where the diode curve represents Eq. (4) and where the load line represents the almost linear relation between V_{eb}' and I_h in Eq. (8) -- ignoring the slight dependency of $\bar{\alpha}$ on I_h. The two situations of major interest shown in Fig. 3 correspond to load lines for "on" and "off" transistors.

The iterative method of solving this problem may be explained as follows: For each transistor, an initial estimate $I_h^{(o)}$ is made. The diode curve is then approximated by a tangent line at the point $(I_h^{(o)}, V_{eb}^{(o)})$ and its intersection with the load line determined. This results in a new estimate $I_h^{(o)}$ as shown in Fig. 4. The process is then repeated until convergence is obtained.

The tangent-line approximation to the diode curve is given by the point-slope formula:

$$V_{eb}' = \left[\frac{1}{\Lambda(I_h^{(o)} + I_{es})}\right] I_h + V_{eb}^{(o)} - \left[\frac{1}{\Lambda(I_h^{(o)}+I_{es})}\right] I_h^{(o)} \tag{9}$$

This relation may be considered as a matrix equation describing the tangent lines for all transistors if the quantity $\left[1/\Lambda(I_h^{(o)} + I_{es})\right]$ is regarded as a diagonal matrix.

Now for transistors operating in the "off" region, close to the asymptotic limit of $-I_{es}$, the computation of $I_h + I_{es}$ may involve serious round-off error. As a result, the larger $1/\Lambda\,(I_h + I_{es})$ becomes, the more inaccurate it is. This error can be avoided, however, by making the change of variable,

$$J = I_h + I_{es} \tag{10}$$

and not calculating I_h explicitly.

Eliminating V_{eb}' from Eqs. (8) and (9) and introducing the variable J, we obtain the following equation for the iteration procedure:

$$\left[H(A^tYA)^{-1}\bar{\alpha}^{(n-1)} - \frac{1}{\Lambda J^{(n-1)}}\right] J^{(n)} = V_{eb}^{(n)} - \left[\frac{1}{\Lambda J^{(n-1)}}\right] + \left[H(A^tYA)^{-1}\bar{\alpha}^{(n-1)}\right] I_{es} + H(A^tYA)^{-1}A^tYE \tag{11}$$

where $\bar{\alpha}^{(n-1)} = \bar{\alpha}(I_h^{(n-1)})$ is assumed to be constant. In this expression, whatever accuracy is inherent in $J^{(n-1)}$ is retained in $1/\Lambda J^{(n-1)}$

One difficulty which arises in applying Eq. (11) is that an "out-of-bounds" intersection of the tangent-line and load line for an "off" transistor may occur, as depicted in Fig. 5. This problem is handled by testing each element of the $J^{(n)}$ vector for proper boundedness and replacing the out-of-bounds elements by the arbitrary value of $I_{es} \times 10^{-6}$.

A second difficulty is that Eq. (11) yields inaccurate J values for the "off" transistors. These values may be improved by the method of successive substitutions depicted in Fig. 6. After each iteration, the elements of the J vector, corresponding to each "off" transistor, are tested for boundedness, replaced by $I_{es} \times 10^{-6}$ if necessary, and then substituted into the equation

$$V_{eb}' = \left[H(A^tYA)^{-1}\bar{\alpha}\right] J - \left[H(A^tYA)^{-1}\bar{\alpha}\right] I_{es} - H(A^tYA)^{-1} A^tYE \tag{12}$$

The J values are then recomputed using the expression

$$J \quad I_{cs}\,(e^{\Delta V_{eb'}}) \tag{13}$$

This process of successive substitutions converges if the slope of the load line is greater than that of the diode curve at their intersection point and if the starting point of the process is close enough to this intersection; otherwise, the process diverges. In order to guard against incipient divergence, the values of $\Delta V_{eb'}$ are monitored and if a value in excess of 4.0 is detected, the process of successive substitutions is terminated. Otherwise, three iterations are made.

The entire procedure of successive approximations is repeated until the fractional change in the length of the J vector is 10^{-7}, with a maximum of thirty iterations being allowed. Although no attempt has been made to establish theoretically the conditions under which convergence will be assured, it is believed that only those circuits which involve large positive feedback would be likely to cause trouble. In such cases, it may be necessary to resort to the transient analysis procedure, using whatever initial conditions are obtained from the d-c analysis, and allow the integration to proceed with no input pulse until a steady state is reached.

In the normal case, after convergence has been attained, all the voltages and currents of the network are computed, completing the d-c analysis. Not all of these data are required as initial conditions for the transient analysis, however, and so only the desired voltages and currents are selected.

Transient Analysis

After the initial conditions have been obtained from the d-c analysis, they are verified by running the numerical integration for a short time without any input pulse. When a satisfactory steady state has been reached, the input pulse is initiated. The transient computation may then be continued as long as desired. At suitable intervals during this computation, the entire set of voltages and currents in the network is printed out. It would be preferrable to display certain portions of this information graphically, either on a printer or cathode ray tube, but this feature was not included in TAP.

The solution of the transient problem, as mentioned above, is based on integrating the linear differential equations of the network separately from the nonlinear differential equations describing the transistors. To explain the practical importance of this modus operandi, a brief outline of the basic theory of the numerical integration is needed first.

The numerical integration of the single differential equation

$$\frac{dx}{dt} = f(x,t) \tag{14}$$

may be effected by means of the generalized expression

$$x_n = \sum_{j=1}^{N} a_j x_{n-j} + h \sum_{j=0}^{M-1} b_j (dx/dt)_{n-j} \tag{15}$$

where $x_n = x(t_o + nh)$ and $h = \Delta t$ is the integration interval. If $b_o = 0$, the integration formula is called a "predictor"; if $b_o \neq 0$, it is called a "corrector." Usually, a combined predictor-corrector scheme is employed with the coefficient a_j and b_j selected to give the accuracy desired.

In the special case of a constant coefficient, linear differential equation, where $f(x,t) = gx$, Eq. (15) becomes

$$(1 - hgb_o)\, x_n = (a_1 + hgb_1)\, x_{n-1} \tag{16}$$

$$+ (a_2 + hgb_2)\, x_{n-2} + \ldots + (a_r + hgb_r) x_{n-r}$$

where at least one of the coefficients a_r or b_r is nonzero. This finite difference equation, of order r, may be shown[10] to have the general solution

$$x_n = \sum_{j=1}^{r} c_j p_j^n \tag{17}$$

where the coefficients c_j are determined by the initial values of $x_o, x_1, \ldots x_{r-1}$ and where $p_1, p_2, \ldots p_r$ are their roots of the characteristic equation

$$(1-hgb_o)p^r - (a_1 - hgb_1)\, p^{r-1} - (a_2 - hgb_2)p^{r-2}$$

$$\ldots - (a_r - hgb_r) = 0 \tag{18}$$

obtained by substituting the particular solution $x_n = p^n x_o$ into Eq. (16).

One of these roots, say p_1, will generate the principal solution to the difference equation. The other r-1 roots will generate parasitic solutions which arise because the order of the finite difference equation is (r-1) greater than that of the differential equation being approximated. Accordingly, if any one of the parasitic roots is greater in magnitude than unity, then the corresponding term $c_j p_j^n$ in Eq. (17) increases without bound as n increases, thereby vitiating the desired solution.[10] This situation, called numerical

instability, arises if the integration interval is made too large.

Similar considerations apply to the numerical integration of a system of differential equations. For example, the system of equations describing a linear, constant parameter RLC network (see Appendix I) are of the form

$$P\dot{X} + Q X = F(t) \tag{19}$$

where P and Q are matrices, $\dot{X} = dX/dt$, X is a vector of voltages and currents, and F(t) is a vector of voltage and/or current sources, some of which may be time-dependent. By inverting the matrix P, we may write,

$$\dot{X} = -P^{-1} Q X + P^{-1}F(t) = S X + G(t) \tag{20}$$

The vector counterpart of Eq. (15) for numerical integration of Eq. (20), in predictor-corrector form, is:

$$\text{predictor:}\quad X_n = \sum_{j=1}^{N} a_j \overline{X}_{n-j} + h\sum_{j=1}^{M} b_j \dot{X}_{n-j} \tag{21}$$

$$\text{corrector:}\quad X_n = \sum_{j=1}^{T} c_j X_{n-j} + h \sum_{j=0}^{W-1} d_j \dot{X}_{n-j} \tag{22}$$

Here, $\overline{X}_n$, the corrected solution vector, is obtained by using the predicted derivative vector, $\dot{X}_n = SX_n + G(t_n)$.

To avoid numerical instability in the use of Eqs. (21) and (22), the integration interval h must be made less in magnitude than the reciprocal of the largest eigenvalue λmax of the matrix S in Eq. (20). This eigenvalue corresponds to the largest natural frequency, and its reciprocal to the smallest natural time constant τ_{min} of the network. Ironically, this eigenvalue, through its exponential function, $e^{-\lambda max t}$, contributes least to the (analytical) solution of Eq. (20) and yet it forces the numerical integration to proceed at a rate determined by the condition,

$$h < 0.25\,\tau_{min} \tag{23}$$

The permissible maximum value of h is in this range but depends somewhat on the actual choice of coefficients a_j, b_j, c_j and d_j in Eqs. (21) and (22).[11,12]

Eqs. (21) and (22) may also be used to approximate the solution of the differential equations describing nonlinear or time-varying networks, since the use of numerical integration implies that the network is linear and time-invariant at every particular time step, t_n, even though its parameters change from one time step to the next. Accordingly, the matrices P and Q will, in general, need to be recomputed and Eq. (19) either solved for X_n or converted to the form of Eq. (20 by inverting P at each time step. This will change the matrix S and its eigenvalues. Hence, the integration interval, h, may need to be adjusted from time to time in order to prevent instability, when τ_{min} decreases, or to permit faster integration when τ_{min} increases.

In the circuits handled by TAP, the smallest time constant is due primarily to the parameters in the transistor model. This time constant, at best, is about 1/10 that of the linear part of the network considered by itself. Therefore, by integrating the equations for the linear network separately from those pertaining to the transistors, a significant increase in speed of integration is obtained. This increase is due not only to the larger integration interval permitted but also to the fact that the P matrix for the linear system need be inverted but once.

The integration scheme used for the linear network in TAP is based on a modified Euler predictor - corrector formula:

$$\text{predictor:}\quad X_n = \overline{X}_{n-1} + h_L \dot{X}_{n-1} \tag{24}$$

$$\text{corrector:}\quad \overline{X}_n = \overline{X}_{n-1} + \frac{h_L}{2}(\dot{X}_n + \dot{X}_{n-1}) \tag{25}$$

with $\dot{X}_n = SX_n + G(t_n)$ and with h_L held constant. The vector $G(t_n)$, which is updated at each time step, contains current source terms that describe the effect of each transistor. The values of these current sources are obtained from the solution of the nonlinear differential equations describing each transistor's behavior.

These equations, two for each transistor, are based on the equivalent circuit shown in Fig. 7 where $R_{bb'}$, $R_{c'}$, $R_{cc'}$ and the current sources αI_h and $I_h(1-\alpha)I_h$ are identical to their counterparts in Fig. 1. The collector-base capacitance C_{tc} is assumed constant but the emitter-base capacitance C_e is assumed to be the sum of the two nonlinear capacitance

$$C_{se} = \frac{\Lambda (I_h + I_{es})}{2\pi f_{\alpha co}} \tag{26}$$

and

$$C_{te} = \frac{K}{(V_o - V_{eb'})^n} \tag{27}$$

where $f_{\alpha co}$ is the common-base cutoff frequency, K is a proportionality constant computed by the

program, V_o is the contact potential, and n is a constant dependent on the grading of the junction. All the basic parameters of this model, except K, are specified on the input cards for each transistor and tabulated during the compilation process. It should be pointed out that C_e is defined as a differential or small-signal, capacitance dQ/dV and not as a static capacitance Q/V. This definition is preferable from the standpoint of measuring C_e but care must be exercised in using small-signal capacitances, as discussed in Appendix I.

The nonlinear differential equations for each transistor are integrated separately using a modified Adam's predictor-corrector formula:

$$\text{predictor: } X_n = \bar{X}_{n-1} + (h_N/24)(55\dot{X}_{n-1} - 59\dot{X}_{n-2} + 37\dot{X}_{n-3} - 9\dot{X}_{n-4}) \quad (28)$$

$$\text{corrector: } \bar{X}_n = \bar{X}_{n-1} + (h_N/24)(9\dot{X}_n + 19\dot{X}_{n-1} - 5\dot{X}_{n-2} + \dot{X}_{n-3}) \quad (29)$$

The integration interval h_N is continuously monitored by comparing the difference between predicted and corrected values of the solution. When this difference increases beyond a certain limit, the integration interval is halved; when this difference decreases sufficiently, the interval is doubled. In this way, since each transistor is treated independently of the others, the integration proceeds at close to the maximum safe rate for each transistor instead of at the rate of the slowest.

The transistor equations require as input data the response voltages at each of the nodes of the linear network to which a transistor terminal is connected. These voltages are computed at each integration interval h_L of the linear system of equations and they are supplied to the nonlinear equations as driving forces. These driving forces are assumed constant over the next integration interval h_L while the nonlinear equations are integrated using the variable interval h_N. The integration process for each transistor is carried along, with h_N being adjusted enroute, until a period exactly equal to h_L has been covered. The terminal currents (emitter, base, and collector) computed at the end of this period are then supplied to the $G(t_n)$ vector for the linear system for its next integration step. In this way, the integration of the linear and nonlinear equations is carried out alternately with both systems being joined at each interval h_L.

Results and Applications

As an indication of the adequacy of TAP in predicting the transient behavior of a transistor switching circuit, the results of the analysis of the circuit shown in Fig. 8 are displayed in Figs. 9-16 together with the observed responses.[1] These results were obtained in about 10 minutes of computing time on the IBM 704. The close agreement between the computed and observed results is really a testimony to the faithfulness of the transistor model since the analytical and computational techniques are in themselves quite dependable.

The potential value of a network analysis program as an experimental tool is indicated by Figs. 17-22 which show the different responses resulting from the variation of a single parameter in the network of Fig. 8. Admittedly, some of these variations can be explored with the actual hardware. But the variation of transistor parameters, such as $f_{\alpha co}$ and V_o, cannot be achieved on demand in any practical sense. Evidently then, a network analysis program such as TAP offers the design engineer a direct means of studying the behavior of circuits and/or devices in intimate detail either for the purpose of increasing his understanding or for helping him to optimize circuit performance.

Beyond the obvious electrical applications of a network analysis program, there are many possible applications to nonelectrical problems. The DYANA program previously mentioned is already taking advantage of this fact by solving mechanical as well as electrical problems. But this is barely scratching the surface of a vast and fertile field. Actually, a significant portion of theoretical physics is amenable to a network approach.[5,13] Indeed, network models exist for so many different physical systems as to force the conclusion that a general purpose network analysis program is capable of converting a digital computer into a versatile and powerful analog machine.

Conclusions and Remarks

An experimental program has been written which is capable of automatically formulating and solving both the d-c and transient analysis problems relating to transistor switching circuits of arbitrary configuration. The program yields computed results which are in reasonable agreement with observations, a fact which proves the adequacy of the transistor model as well as that of the program itself.

The principal features of this program are: (1) its ability to formulate the network problem automatically on the basis of simple input data specifying the network parameters and configuration; (2) its use of topological-matrix methods for handling part of the formulation and analysis; (3) its faster solution of the transient problem by separately integrating the linear and nonlinear sets of differential equations.

The main failing of the program is the difficulty of altering or replacing the transistor equivalent circuit. Another disadvantage is the printing, rather than plotting, of the computed responses. Both of these disadvantages have been eliminated from a more recent program for circuit analysis of nonlinear systems (PE CANS) developed for the IBM 7090 computer by Beaudette and Honkanen.[14] This program compiles the equations for a network including arbitrary nonlinear elements. Hence, it can handle equivalent circuits for a variety of nonlinear devices.

A. F. Malmberg, at Los Alamos Scientific Laboratory, has also written a network analysis program for the MANIAC II computer.[15] This program is based on the topological-matrix formulation described in Appendix I and uses a network model capable of describing saturating transistors.

It has been amply demonstrated, therefore, that it is quite feasible to program a digital computer to both formulate and solve the algebraic and/or differential equations of an arbitrary network -- including at least certain types of nonlinear device. It now remains to refine the techniques described here and to develop new ones so that the full potentialities of a general purpose network analysis program can be realized. Clearly, the practical utility of such a program will depend almost as much on its input/output facilities as on its speed. Accordingly, due attention must be given to such user-oriented features as input format (including original network specifications and modifications thereto) and output display.

The central difficulty, however, is still that of solving the transient problem. Much can be gained by refining the techniques of programming predictor-corrector formulas. But what is really needed is a genuine analytical breakthrough which will lead to an orders-of-magnitude increase in speed. Such a breakthrough, it would appear, cannot possibly come unless some way over, around, or through the minimum time-constant barrier can be found. This is a frontier which offers the greatest challenge and most promising rewards.

Acknowledgements

The TAP program represents the combined efforts of several people whose contributions deserve to be recognized. The originator and director of the project was R. J. Domenico. The transient analysis program, developed primarily by Mrs. Nancy G. Brooks with the assistance of H. von Horn and E. J. Skiko, was based on the earlier work of G. L. Lasher and J. C. Morgan. The numerical integration program, used in the nonlinear transient analysis, was written by M. K. Haynes. The d-c analysis was developed by the present author and implemented by him with the assistance of H. S. Long and J. T. Urbain.

Appendix I

Formulation of the Transient Network Problem

The following topological-matrix formulation of the linear transient network problem leads to a simultaneous system of algebraic and first order differential equations similar to that previously described by Bashkow.[9] The present formulation, however, avoids the introduction of the extraneous capacitors and inductors which Bashkow's derivation requires. Moreover, it is in a form which is suitable for programming on a digital computer by an extension of the techniques described elsewhere in this paper. The terminology and notation to be used are essentially the same as in previous work of the author.[5,6]

Instead of employing either the mesh method or the node method of analysis, the present formulation of the transient problem makes use of a combination of these two methods. Actually, the tree method,[5,6] rather than the node method, is combined with the mesh method. This combination, which is also implicit in Bashkow's formulation, is made necessary by the requirement to establish first order differential equations rather than integro-differential equations to characterize the reactive elements of the network. A formal description of this combined method of analysis will be given first. The necessary extension to the transient problem then follows easily.

It is assumed that the network branches are first divided into two categories: admittances, designated by the subscript y, and impedances, designated by the subscript z. It is also assumed that there is no coupling between any admittance branch and any impedance branch although branches within the same category may be coupled arbitrarily with one another. Ohm's law, instead of being written either in the admittance form $J = YV$ or in the impedance form $V = ZJ$, is now written in the mixed form

$$\begin{bmatrix} J_y \\ V_z \end{bmatrix} = \begin{bmatrix} Y_y & 0 \\ 0 & Z_z \end{bmatrix} \begin{bmatrix} V_y \\ J_z \end{bmatrix} \tag{30}$$

where Y_y and Z_z are the primitive admittance and primitive impedance matrices, and where J and V are the coil current and coil voltage vectors.[5,6] Using the relations J = I + i and V = E + e, where I and E are the current and voltage source vectors while i and e are the branch current and branch voltage (response) vectors, Eq. (30) may also be written as follows:

$$\begin{bmatrix} I_y + i_y \\ E_z + e_z \end{bmatrix} = \begin{bmatrix} Y_y & 0 \\ 0 & Z_z \end{bmatrix} \begin{bmatrix} E_y + e_y \\ I_z + i_z \end{bmatrix} \tag{31}$$

Disregarding the question of ordering the branches, it is now assumed that the admittance branches are classified as either tree-branches or links, using the procedure outlined in Appendix II. Then, with the resulting admittance tree as a starting point, the impedance branches are similarly classified. The network tree obtained in this fashion will, of course, contain both admittance and impedance branches. However, since all the admittances will have been subjected first to the tree-link sorting procedure, all the basic meshes defined by admittance links will necessarily include only admittance tree-branches.

On the other hand, the basic meshes defined by the impedance links may include both admittance and impedance tree-branches. As a consequence, the C_T matrix contains one null submatrix. For if the matrices B_T and A_T are partitioned into submatrices thus,

$$B_T = \begin{bmatrix} B_{Ty} \\ B_{Ty} \end{bmatrix} \tag{32}$$

and

$$A_L = \begin{bmatrix} A_{Ly} \\ A_{Ly} \end{bmatrix} \tag{33}$$

it follows from Eq. (1) that

$$C_T = \begin{bmatrix} C_{Tyy} & C_{Tyz} \\ 0 & C_{Tzz} \end{bmatrix} = \begin{bmatrix} B_{Ty} A^t_{Ly} & -B_{Ty} A^t_{Lz} \\ 0 & -B_{Tz} A^t_{Lz} \end{bmatrix} \tag{34}$$

since $C_{Tzy} = -B_{Tz} A^t_{Ly} = 0$, as explained above.

In accord with the tree method of analysis, the branch voltages e for the entire network are expressed as a linear combination of the tree branch voltages e_T using the relation $e = De_T$, where D is the basic cut-set matrix for the entire network.[5,6] At the same time, in keeping with the mesh method, the branch currents i are expressed as a linear combination of the link currents i_L (which, by convention, are identical with the mesh currents,) using the relation $i = Ci_L$.

These relations, together with the four-way classification of branches described above, lead to the expressions

$$\begin{bmatrix} e_{Ty} \\ e_{Tz} \\ e_{Ly} \\ e_{Lz} \end{bmatrix} = \begin{bmatrix} U_{Ty} & 0 \\ 0 & U_{Tz} \\ -C^t_{Tyy} & 0 \\ -C^t_{Tyz} & -C^t_{Tzz} \end{bmatrix} \begin{bmatrix} e_{Ty} \\ e_{Tz} \end{bmatrix} \tag{35}$$

and

$$\begin{bmatrix} i_{Ty} \\ i_{Tz} \\ i_{Ly} \\ i_{Lz} \end{bmatrix} = \begin{bmatrix} C_{Tyy} & C_{Tyz} \\ 0 & C_{Tzz} \\ U_{Ly} & 0 \\ 0 & U_{Lz} \end{bmatrix} \begin{bmatrix} i_{Ly} \\ i_{Lz} \end{bmatrix} \tag{36}$$

where use has been made of the fact that for the basic cut-set matrix, $D_T = U_T$ (a unit matrix) and $D_L = -C^t_T$.[5,6]

Next, Eq. (31) is rearranged to give

$$\begin{bmatrix} (I_y - Y_y E_y) \\ (E_z - Z_z I_z) \end{bmatrix} = \begin{bmatrix} Y_y & 0 \\ 0 & Z_z \end{bmatrix} \begin{bmatrix} e_y \\ i_z \end{bmatrix} - \begin{bmatrix} i_y \\ e_z \end{bmatrix} \tag{37}$$

where e_y includes both subvectors e_{Ty} and e_{Ly}, i_z includes both i_{Tz} and i_{Lz}, etc. It now becomes necessary to introduce the admittance cut-set matrix,

$$D_y = \begin{bmatrix} U_{Ty} \\ -C^t_{Tyy} \end{bmatrix} \tag{38}$$

and the impedance mesh matrix,

$$C_z = \begin{bmatrix} C_{Tzz} \\ U_{Lz} \end{bmatrix} \tag{39}$$

to extract from Eq. (35) the expression $e_y = D_y e_{Ty}$, or

$$e_y = \begin{bmatrix} e_{Ty} \\ e_{Ly} \end{bmatrix} = \begin{bmatrix} U_{Ty} \\ -C^t_{Tyy} \end{bmatrix} e_{Ty} \tag{40}$$

and from Eq. (36) the expression $i_z = C_z i_{Lz}$, or

$$i_z = \begin{bmatrix} i_{Tz} \\ i_{Lz} \end{bmatrix} = \begin{bmatrix} C_{Tzz} \\ U_{Lz} \end{bmatrix} i_{Lz} \tag{41}$$

Then, considering only the first column of D in Eq. (35) and using the fundamental relation $D^t i = 0$, it is easily shown that

$$D_y^t i_y = \begin{bmatrix} U_{Ty} & -C_{Tyy} \end{bmatrix} \begin{bmatrix} i_{Ty} \\ i_{Ly} \end{bmatrix} = -C_{Tyz} i_{Lz} \tag{42}$$

Similarly, from the second column of C in Eq. (36) and the equation $C^t e = 0$, it follows that

$$C_z^t e_z = \begin{bmatrix} C_{Tzz}^t & U_{Lz} \end{bmatrix} \begin{bmatrix} e_{Tz} \\ e_{Lz} \end{bmatrix} = -C_{Tyz}^t e_{Ty} \tag{43}$$

Finally, premultiplication of the first row of Eq. (37) by D_y^t and of the second row by C_z^t, followed by substitution of $D_y e_{Ty}$ in place of e_y and of $C_z i_{Lz}$ in place of i_z, yields the result,

$$\begin{bmatrix} D_y^t(I_y - Y_y E_y) \\ C_z^t(E_z - Z_z I_z) \end{bmatrix} = \begin{bmatrix} D_y^t Y_y D_y & 0 \\ 0 & C_z^t Z_z C_z \end{bmatrix} \begin{bmatrix} e_{Ty} \\ i_{Lz} \end{bmatrix} - \begin{bmatrix} D_y^t i_y \\ C_z^t e_z \end{bmatrix} \tag{44}$$

which, together with Eqs. (42) and (43) may be condensed to the desired expression,

$$\begin{bmatrix} D_y^t(I_y - Y_y E_y) \\ C_z^t(E_z - Z_z I_z) \end{bmatrix} = \begin{bmatrix} D_y^t Y_y D_y & -C_{Tyz} \\ C_{Tyz}^t & C_z^t Z_z C_z \end{bmatrix} \begin{bmatrix} e_{Ty} \\ i_{Lz} \end{bmatrix} \tag{45}$$

Thus, Eq. (45) amounts to a tree analysis of the admittance branches alone followed by a mesh analysis of the impedance branches alone, the resulting two sets of equations being coupled together by the submatrix C_{Tyz}. But this submatrix denotes those admittance tree branches which belong to basic meshes defined by impedance links. Hence it follows that the corresponding admittance-tree-branch voltages and impedance-link currents will exhibit a reciprocal interaction.

Now in order to guarantee that only first order differential (as well as algebraic) equations will result from the application of Eq. (45) to the transient problem, all capacitors must be classified as admittances and all inductors as impedances. Resistors, however, may be put into either category. The matrices Y_y and Z_z, then, contain both algebraic and differential operators and may be written thus:

$$Y_y = G_y + \frac{d}{dt} K_y \tag{46}$$

and

$$Z_z = R_z + \frac{d}{dt} L_z \tag{47}$$

where the symbols G, K, R and L denote conductance, capacitance, resistance and inductance matrices. (K is used for capacitance because C has already been used to designate a topological matrix.) Hence, for the most general case of time-varying capacitances and inductances, Eq. (30) becomes

$$\begin{bmatrix} J_y \\ V_z \end{bmatrix} = \begin{bmatrix} K_y & 0 \\ 0 & L_z \end{bmatrix} \begin{bmatrix} \dot{V}_y \\ \dot{J}_z \end{bmatrix} + \begin{bmatrix} G_y + \dot{K}_y & 0 \\ 0 & R_z + \dot{L}_z \end{bmatrix} \begin{bmatrix} V_y \\ J_z \end{bmatrix} \tag{48}$$

All admittances having zero capacitance and all impedances having zero inductance will, of course, give rise to zero entries in the K_y and L_z matrices; they will also generate algebraic rather than differential equations. Therefore, to gather these zeros into null matrices and group the algebraic equations together, it is convenient to classify the network branches as follows:

(1) admittances with nonzero capacitance,
(2) admittances with conductance only,
(3) impedances with nonzero inductance,
(4) impedances with resistance only.

These four classes will be denoted by the subscripts k, g, l, and r respectively and it will be assumed that these classes are always in the order shown above. Accordingly, Eq. (48) becomes

$$\begin{bmatrix} J_{yk} \\ J_{yg} \\ V_{zl} \\ V_{zr} \end{bmatrix} = \begin{bmatrix} K_{yk} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & L_{zl} & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} \dot{V}_{yk} \\ \dot{V}_{yg} \\ \dot{J}_{kl} \\ \dot{J}_{zr} \end{bmatrix} + \begin{bmatrix} G_{yk} + \dot{K}_{yk} & 0 & 0 & 0 \\ 0 & G_{yg} & 0 & 0 \\ 0 & 0 & R_{zl} + \dot{L}_{zl} & 0 \\ 0 & 0 & 0 & R_{zr} \end{bmatrix} \begin{bmatrix} V_{yk} \\ V_{yg} \\ J_{zl} \\ J_{zr} \end{bmatrix} \tag{49}$$

where the variables with the subscripts yg and zr are involved in purely algebraic equations. It should be noted that all capacitative branches may have nonzero conductance and that all inductive branches may have nonzero resistance. Hence, these particular conductive and/or resistive elements need not be treated as separate branches. This does not preclude their being treated as separate branches, however, if there is some reason for doing so.

If the network branches, ordered by class as shown above (but arbitrarily ordered within each class), are subjected to a tree-link sort, then a sequence of expressions similar to Eqs. (32) to (45), but with twice as many branch categories will result. In particular, it follows that

$$B_T = \begin{bmatrix} B_{Tk} \\ B_{Tg} \\ B_{Tl} \\ B_{Tr} \end{bmatrix} \tag{50}$$

$$A_L = \begin{bmatrix} A_{Lk} \\ A_{Lg} \\ A_{Ll} \\ A_{Lr} \end{bmatrix} \tag{51}$$

and

$$C_T = -B_T A_L^t = \begin{bmatrix} C_{Tkk} & C_{Tkg} & C_{Tkl} & C_{Tkr} \\ 0 & C_{Tgg} & C_{Tgl} & C_{Tgr} \\ 0 & 0 & C_{Tll} & C_{Tlr} \\ 0 & 0 & 0 & C_{Trr} \end{bmatrix} \tag{52}$$

where the submatrices of C_T are defined in the obvious way. Eqs. (38) and (39) now become

$$D_y = \begin{bmatrix} U_{Tk} & 0 \\ 0 & U_{Tg} \\ -C_{Tkk}^t & 0 \\ -C_{Tkg}^t & -C_{Tgg}^t \end{bmatrix} \tag{53}$$

and

$$C_z = \begin{bmatrix} C_{Tll} & C_{Tlr} \\ 0 & C_{Trr} \\ U_{Ll} & 0 \\ 0 & U_{Lr} \end{bmatrix} \tag{54}$$

while the submatrix C_{Tyz} of Eqs. (42), (43) and (45) becomes

$$C_{Tyz} = \begin{bmatrix} C_{Tkl} & C_{Tkr} \\ C_{Tgl} & C_{Tgr} \end{bmatrix} \tag{55}$$

Next, the vectors e and i are written in partitioned form (as row vectors) as:

$$e = (e_{Tk}\ e_{Tg}\ e_{Tl}\ e_{Tr}\ e_{Lk}\ e_{Lg}\ e_{Ll}\ e_{Lr}) \tag{56}$$

$$i = (i_{Tk}\ i_{Tg}\ i_{Tl}\ i_{Tr}\ i_{Lk}\ i_{Lg}\ i_{Ll}\ i_{Lr}) \tag{57}$$

while the Y_y and Z_z matrices are partitioned thus:

$$Y_y = \begin{bmatrix} G_{Tk} + \frac{d}{dt}K_{Tk} & 0 & 0 & 0 \\ 0 & G_{Tg} & 0 & 0 \\ 0 & 0 & G_{Lk} + \frac{d}{dt}K_{Lk} & 0 \\ 0 & 0 & 0 & G_{Lg} \end{bmatrix} \tag{58}$$

and

$$Z_z = \begin{bmatrix} R_{Tl} + \frac{d}{dt}L_{Tl} & 0 & \frac{d}{dt}L_{TLl} & 0 \\ 0 & R_{Tr} & 0 & 0 \\ \frac{d}{dt}L_{LTl} & 0 & R_{Ll} + \frac{d}{dt}L_{Ll} & 0 \\ 0 & 0 & 0 & R_{Lr} \end{bmatrix} \tag{59}$$

where the matrices L_{TLl} and L_{LTl} allow for inductive coupling (if any) between tree-branches and links.

Finally, with all these relations substituted into Eq. (45), it can be shown that

$$\begin{bmatrix} I'_{Tk} \\ I'_{Tg} \\ E'_{Ll} \\ E'_{Lr} \end{bmatrix} = \begin{bmatrix} K'_{Tkk} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & L'_{Lll} & L'_{Llr} \\ 0 & 0 & L'_{Lrl} & L'_{Lrr} \end{bmatrix} \begin{bmatrix} \dot{e}_{Tk} \\ \dot{e}_{Tg} \\ \dot{i}_{Ll} \\ \dot{i}_{Lr} \end{bmatrix} + \begin{bmatrix} G'_{kk}+\dot{K}'_{kk} & G'_{kg} & -C_{Tkl} & -C_{Tkr} \\ G'_{gk} & G'_{gg} & -C_{Tgl} & -C_{Tgr} \\ C^t_{Tkl} & C^t_{Tgl} & R'_{ll}+\dot{L}'_{ll} & R'_{lr}+\dot{L}'_{lr} \\ C^t_{Tkr} & C^t_{Tgr} & R'_{rl}+\dot{L}'_{rl} & R'_{rr}+\dot{L}'_{rr} \end{bmatrix} \begin{bmatrix} e_{Tk} \\ e_{Tg} \\ i_{Ll} \\ i_{Lr} \end{bmatrix} \quad (60)$$

where the following relations apply:

$$\begin{bmatrix} I'_{Tk} \\ I'_{Tg} \end{bmatrix} = \begin{bmatrix} U_{Tk} & 0 & -C_{Tkk} & -C_{Tkg} \\ 0 & U_{Tg} & 0 & -C_{Tgg} \end{bmatrix} \left\{ \begin{bmatrix} I_{Tk} \\ I_{Tg} \\ I_{Lk} \\ I_{Lg} \end{bmatrix} - \begin{bmatrix} (G_{Tk}+\dot{K}_{Tk})E_{Tk} \\ G_{Tg}\,E_{Tg} \\ (G_{Lk}+\dot{K}_{Lk})E_{Lk} \\ G_{Lg}\,E_{Lg} \end{bmatrix} - \begin{bmatrix} K_{Tk}\dot{E}_{Tk} \\ 0 \\ K_{Lk}\dot{E}_{Lk} \\ 0 \end{bmatrix} \right\} \quad (61)$$

$$\begin{bmatrix} E'_{Ll} \\ E'_{Lr} \end{bmatrix} = \begin{bmatrix} C^t_{Tll} & 0 & U_{Ll} & 0 \\ C^t_{Tlr} & C^t_{Trr} & 0 & U_{Lr} \end{bmatrix} \left\{ \begin{bmatrix} E_{Tl} \\ E_{Tr} \\ E_{Ll} \\ E_{Lr} \end{bmatrix} - \begin{bmatrix} (R_{Tl}+\dot{L}_{Tl})I_{Tl} \\ R_{Tr}I_{Tr} \\ (R_{Ll}+\dot{L}_{Ll})I_{Ll} \\ R_{Lr}I_{Lr} \end{bmatrix} - \begin{bmatrix} L_{Tl}\dot{I}_{Tl}+L_{TLl}\dot{I}_{Ll} \\ 0 \\ L_{Ll}\dot{I}_{Ll}+L_{LTl}\dot{I}_{Tl} \\ 0 \end{bmatrix} \right\} \quad (62)$$

$$K'_{Tkk} = K_{Tk} + C_{Tkk}\,K_{Lk}\,C^t_{Tkk} \quad (63)$$

$$L'_{Lll} = L_{Ll} + C^t_{Tll}\,L_{Tl}\,C_{Tll} + C^t_{Tll}\,L_{TLl} + L_{LTl}\,C_{Tll} \quad (64)$$

$$L'_{Llr} = C^t_{Tll}\,L_{Tl}\,C_{Tlr} + L_{LTl}\,C_{Tlr} \quad (65)$$

$$L'_{Lrl} = C^t_{Tlr}\,L_{Tl}\,C_{Tll} + C^t_{Tlr}\,L_{TLl} \quad (66)$$

$$L'_{Lrr} = C^t_{Tlr}\,L_{Tl}\,C_{Tlr} \quad (67)$$

$$G'_{kk} = G_{Tk} + C_{Tkk}\,G_{Lk}\,C^t_{Tkk} + C_{Tkg}\,G_{Lg}\,C^t_{Tkg} \quad (68)$$

$$G'_{kg} = C_{Tkg}\,G_{Lg}\,C^t_{Tgg} \quad (69)$$

$$G'_{gk} = C_{Tgg}\,G_{Lg}\,C^t_{Tkg} \quad (70)$$

$$G'_{gg} = G_{Tg} + C_{Tgg}\,G_{Lg}\,C^t_{Tgg} \quad (71)$$

$$R'_{ll} = R_{Ll} + C^t_{Tll}\,R_{Tl}\,C_{Tll} \quad (72)$$

$$R'_{lr} = C_{Tll}\,R_{Tl}\,C_{Tlr} \quad (73)$$

$$R'_{rl} = C^t_{Tlr}\,R_{Tl}\,C_{Tll} \quad (74)$$

$$R'_{ll} = C'_{Tlr}\,R_{Tl}\,C_{Tlr} \quad (75)$$

From Eqs. (60), (65), (66) and (67), it is apparent that unless $C_{Tlr} = 0$, a set of differential equations will arise from those purely resistive network branches which are classed as impedances. In other words, if any of the link currents i_{Lr} pass through inductive impedances belonging to the network tree (and this is what the matrix C_{Tlr} specifies) these link currents must be determined as the solution of differential rather than algebraic equations. Therefore, unless some specific reason exists for not doing so, all purely resistive branches of a network should be treated as conductances. This will not only reduce the number of differential equations to be solved but will also result in considerable simplification of Eqs. (49) et seq.

If this is done it can be shown that the remaining algebraic equations in Eq. (60) have the solution

$$e_{Tg} = (G'_{gg})^{-1}\left[I'_{Tg} - G'_{gk}e_{Tk} + C_{Tgl}\,i_{Ll} + C_{Tgr}\,i_{Lr}\right] \quad (76)$$

Hence, if this expression is substituted in Eq. (60), a set of differential equations identical in form to Eq. (19) is obtained. The total number of differ-

ential equations derived in this way corresponds to the degree of complexity of the network.

However, if some of the purely resistive branches are treated as impedances, then more than this number of differential equations may be obtained, as explained above. But this larger number cannot also be equal to the degree of complexity of the network. Accordingly, the true degree of complexity of any capacitive and/or inductive network must be equal to the sum of the number of capacitive tree-branches plus the number of inductive links identified by the foregoing tree-link sorting procedure.

One precaution in using Eq. (60) should be pointed out. This equation is based on the definitions of static capacitance as charge/voltage and static inductance as flux/current. These definitions, in turn, are responsible for the terms $\dot{K}_y V_y + K_y \dot{V}_y$ and $\dot{L}_z J_z + L_z \dot{J}_z$ in Eq. (48). But when differential (small-signal) capacitances and/or inductances are involved, as in the case of C_e defined by Eqs. (26) and (27), the quantities $\dot{K}_y V_y$ and $\dot{L}_z J_z$ must be deleted from Eq. (48). This follows from the definitions of differential capacitance $\bar{K}$ and differential inductance $\bar{L}$ according to the relations

$$J = \dot{Q} = \frac{dQ}{dV} \dot{V} = \bar{K} \dot{V} \qquad (77)$$

and

$$V = \dot{\phi} = \frac{d\phi}{dJ} \dot{J} = \bar{L} \dot{J} \qquad (78)$$

Accordingly, all terms involving $\dot{K}$ or $\dot{L}$, with whatever subscripts, must be eliminated from Eqs. (60), (61) and (62) when differential capacitances and/or inductances are involved.

Appendix II

Tree-Link Sorting Procedure

The object of this sorting procedure is to classify each branch of the network as either a tree-branch or a link and then to rearrange the tables RCON, RDATA, EDATA, and IDATA accordingly. It is assumed that these tables are already in some desired sequence, such as in order of increasing resistance, and that a set of tree branches is to be selected from as near the beginning of this sequence as possible. (This will result in choosing the tree of minimum total resistance if the original ordering is that of increasing resistance.)

Starting with the first branch of the sequence, the network tree is constructed stepwise by adding only those branches which do not form a closed path with the partial tree already constructed. This partial tree, as well as the complete network tree, may consist of several disjoint subtrees. Hence, a branch will be classed as a link if both of its nodes are already connected by the branches of one of these subtrees – or if its initial and final nodes are identical, a trivial case which must nevertheless be handled.

Each branch is examined by comparing its initial node number (+N) and its final node number (-N), obtained from the RCON table, against a master list (MLIST) of nodes contained in the partial tree and also against the individual node list (TLIST) for each subtree. Each such node list consists of a string of bits indicating the presence (1) or absence (0) of the node corresponding to a given bit position in the string.

The following criteria form the logical basis of the tree-link sorting procedure:

(1) If +N = -N, the branch is a link.

(2) If either +N or -N or both are absent from the MLIST, the branch is a tree branch.

(3) If both +N and -N are present in MLIST,

  (a) The branch is a link if both +N and -N are also in the same TLIST.
  (b) The branch is a tree-branch (joining two previously disjoint subtrees) if +N and -N are in different TLISTS.

As each branch is examined, the appropriate node lists are updated. When the classification as tree-branch or link has been made for the I-th branch, its index (I) is stored in the next available location of TREE or LINK, as appropriate. The two resulting sequences of index numbers are then used for rearranging the tables RCON, RDATA, EDATA, and IDATA after the tree-link sort has been completed.

In the following description of the tree-link sorting procedure, the symbols I, J, K, L and M are indices, hence +N(I) and -N(I) represent the initial and final node numbers of the I-th branch and are obtained from the address and decrement of the I-th word of the RCON table. IMAX is the total number of branches in the network.

1. Set I = J = K = 1, clear MLIST and all TLISTS.
2. If +N(I) = -N(I), go to 16.
3. If +N(I) is absent from MLIST, go to 11.
4. If -N(I) is absent from MLIST, go to 15.
5. Find L for which +N(I) is present in TLIST(L) and save L.
6. If -N(I) is present in TLIST(L), go to 16.

7. Find M for which -N(I) is present in TLIST(M) and save M.
8. If M<1., go to 10.
9. Add TLIST(M) to TLIST(L), clear TLIST(M), and go to 19.
10. Add TLIST(L) to TLIST(M), clear TLIST(L), and go to 19.
11. Add +N(I) to MLIST.
12. If -N(I) is absent from MLIST, go to 14.
13. Find L for which -N(I) is present in TLIST(L), add +N(I) to TLIST(L), and go to 19.
14. Add -N(I) to MLIST, find lowest L for which TLIST(L) is clear, add +N(I) and -N(I) to TLIST(L), and go to 19.
15. Add -N(I) to MLIST, find L for which +N(I) is present in TLIST(L), add -N(I) to TLIST(L), and go to 19.
16. Set LINK(J) = I.
17. If I = IMAX, go to 22.
18. Set I = I + 1, J = J + 1, and go to 2.
19. Set TREE(K) = I.
20. If I = IMAX, go to 22.
21. Set I = I + 1, K = K + 1, and go to 2.
22. Rearrange RCON, RDATA, EDATA and IDATA according to the index numbers in TREE and LINK.

Appendix III

Determination of the B_T Matrix

The node-to-datum-path matrix, B_T, is determined by an exhaustive search of the network tree. Starting at the datum node, and proceeding always along the branch of lowest serial number connected to each node encountered en route, a path is traced out until it terminates at some particular node of the tree. As the path is traced, a path record (PR) is kept in +1, -1, 0 format showing the branches traversed and their orientations relative to this path in the sense of a node-to-datum traversal.

When the path terminates at some node J, the path record is stored in the J-th column of the B_T matrix. The branch leading to node J is then retraced, its entry in PR deleted, and this branch removed from the tree. Next, the outward path is continued, if possible, again taking the branch of lowest serial number at each successive node until the path terminates once more at, say, node K. The PR is stored in the K-th column of B_T, the branch leading to node K retraced, its entry in PR deleted, and the branch removed from the tree. By repeating this procedure until all branches of the tree have been exhausted, the entire B_T matrix may be determined.

In actual practice, both the RCON table and the branch-node matrix are used alternately in tracing out these datum-to-node paths. Since the branch-node matrix is stored columnwise, it provides the simplest means of determining the branch of lowest serial number connected to a particular node. The RCON table, however, is more convenient for finding the number of the node at the far end of a given branch, wherever this is required.

During the search procedure described below, the branch-node matrix, which must include the datum column, is destroyed. Since each entry of this matrix, designated A(I, J), consists of a bit-pair, there are four possible values of each bit-pair of which only three are required for the quantities +1, -1, 0. The fourth value, designated -0, is required to designate the "access" branch leading to each successive node of the path being traced. It is this access branch which must be identified whenever it is necessary to retrace and delete a branch from the tree. Deletion of a branch, after retracing it, is accomplished simply by substituting 0 in place of the +1 or -1 value of the appropriate A(I, J).

In the following description of the procedure for determining the B_T matrix, the symbols I and J represent the row (branch) and column (node) indices. (J = 0 designates the datum node.) As in Appendix II, the symbols +N(I) and -N(I) represent the initial and final node numbers of the I-th branch and are obtained from the I-th word of the RCON table. IMAX is the total number of branches in the tree.

1. Set I = 1, J = 0.
2. If A(I, J) = ±1, go to 5.
3. If I = IMAX, go to 7.
4. Set I = I + 1 and go to 2.
5. If A(I, J) = +1, set J = -N(I), PR(I) = -1, A(I, J) = -0, I = 1, and go to 2.
6. Set J = +N(I), PR(I) = +1, A(I, J) = -0, I = 1, and go to 2.
7. Transfer PR to column J of B_T matrix and set I = 1.
8. If A(I, J) = -0, go to 11.
9. If I = IMAX, go to 15.
10. Set I = I + 1 and go to 8.
11. If PR(I) = +1, go to 13.
12. Set J = +N(I) and go to 14.
13. Set J = -N(I).
14. Set PR(I) = 0, A(I, J) = 0, I = I + 1, and go to 2.
15. If J = 0, stop. Otherwise, an error has occurred.

Appendix IV

Computation of $B_T^t Z_T B_T$

By taking advantage of the diagonal nature of the matrix Z_T and the compact storage format of the matrix B_T, a very efficient program can be developed for computing the triple matrix product $B_T^t Z_T B_T$. Since Z_T is diagonal, it follows that the ij-th term of this product is given by the expression

$$(B_T^t Z_T B_T)_{ij} = \sum_{k=1}^{p} (b_{ki} b_{kj}) z_{kk} \qquad (79)$$

where p is the number of tree-branches, b_{ki} and/or b_{kj} are elements of B_T, and z_{kk} are the diagonal elements of Z_T. Since this product is symmetric, only the diagonal (i=j) and subdiagonal (i>j) terms need be computed.

Let the i-th and j-th columns of B_T be designated by $B_{.i}$ and $B_{.j}$ and let the product $(b_{ki} b_{kj})$ for all values of k be represented by the expression

$$B(i,j) = B_{.i} \otimes B_{.j} \qquad (80)$$

where the special operator $\otimes$ signifies multiplication of the corresponding elements of $B_{.i}$ and $B_{.j}$ and where the elements of the p-vector B(i,j) are +1, -1, or 0. Next define a vector Z(T) comprised of the diagonal elements of Z_T. Then, the result defined by Eq. (79) is identical with the scalar product of the two vector B(i,j) and Z(T).

The advantage of using this peculiar method for evaluating Eq. (79) is the fact that the compact storage format of $B_{.i}$ and $B_{.j}$ allows the vector B(i,j) to be computed many elements at a time. Moreover, this computation may be effected by means of logical operations (rather than arithmetic operations) on the bit-pair equivalents of the elements +1, -1 and 0 in $B_{.i}$ and $B_{.j}$, the result being the bit-pair representation of B(i,j). Thus, the calculation of B(i,j) may be carried out very rapidly.

The subsequent computation of the scalar product of B(i,j) and Z(T) involves searching B(i,j) for its nonzero elements and then adding or subtracting the corresponding elements of Z(T). The programming details of this task, however, need not be discussed.

The code actually developed for calculating B(i,j) on the IBM 704 computer will not be described. Instead, an equivalent and much simpler scheme will be outlined to illustrate the principles of the computation. It is assumed that p = 6 and that the machine word length is 6 bits since only 6 different combinations of the elements +1, -1, 0 are encountered. To show this, let $B_{.i}$ and $B_{.j}$, written as row vectors, be

$$(B_{.i}) = (1 \quad 1 \quad 1 \quad 0 \quad 0 \quad -1) \qquad (81)$$

$$(B_{.j}) = (1 \quad 0 \quad -1 \quad 0 \quad -1 \quad -1) \qquad (82)$$

Hence B(i,j), also written as a row vector, is

$$B(i,j) = (1 \quad 0 \quad -1 \quad 0 \quad 0 \quad 1) \qquad (83)$$

These three vectors may be represented in machine code by using the bits of one word to indicate the magnitudes (M) and bits of another word to indicate the signs (S) of successive elements:

$$(B_{.i}) = \begin{matrix} 1\,1\,1\,0\,0\,1 & M(I) \\ 0\,0\,0\,0\,0\,1 & S(I) \end{matrix} \qquad (84)$$

$$(B_{.j}) = \begin{matrix} 1\,0\,1\,0\,1\,1 & M(J) \\ 0\,0\,1\,0\,1\,1 & S(J) \end{matrix} \qquad (85)$$

$$B(i,j) = \begin{matrix} 1\,0\,1\,0\,0\,1 & M(I,J) \\ 0\,0\,1\,0\,0\,0 & S(I,J) \end{matrix} \qquad (86)$$

The logical "AND" operation, then, suffices to convert M(I) and M(J) into M(I, J):

$$\begin{matrix} & 1\,1\,1\,0\,0\,1 & M(I) \\ \text{"AND"} & 1\,0\,1\,0\,1\,1 & M(J) \\ & \hline 1\,0\,1\,0\,0\,1 & M(I,J) \end{matrix}$$

or,

$$M(I,J) = M(I) \text{ "AND" } M(J). \qquad (87)$$

Two steps are needed, however, for the calculation of S(I,J). First, S(I) and S(J) are combined using the "exclusive or" operation:

$$\begin{matrix} & 0\,0\,0\,0\,0\,1 & S(I) \\ \text{"EXOR"} & 0\,0\,1\,0\,1\,1 & S(J) \\ & \hline 0\,0\,1\,0\,1\,0 & \text{RESULT} \end{matrix}$$

Then, this result is combined with M(I,J) using the "AND" operation:

$$\begin{matrix} & 0\,0\,1\,0\,1\,0 & \text{RESULT} \\ \text{"AND"} & 0\,0\,1\,0\,0\,1 & M(I,J) \\ & \hline 0\,0\,1\,0\,0\,0 & S(I,J) \end{matrix}$$

Hence, it follows that

$$S(I,J) = \left[S(I) \text{ "EXOR" } S(J)\right] \text{ "AND" } M(I,J) \qquad (88)$$

Since this procedure treats the magnitude bits separately from the sign bits, it is much more efficient than the procedure actually used in TAP.

References

1. Nancy G. Brooks and H. S. Long, "A Program for Computing the Transient Response of Transistor Switching Circuits – PETAP," Technical Report 00. 700, IBM Development Laboratory, Poughkeepsie, New York (1959)
2. F. H. Branin, Jr., "D-C Analysis Portion of PE TAP – A Program for Analyzing Transistor Switching Circuits," Technical Report TR 00.701, IBM Development Laboratory, Poughkeepsie, New York (1960)
3. J. T. Olsztyn and T. J. Theodoroff, "GMR DYANA Programming Manuals I and II," General Motors Research Laboratories, Warren, Michigan (1959).
4. G. Kron, "A Set of Principles to Interconnect the Solutions of Physical Systems," J. Appl. Phys., 24, 965-980 (1953)
5. F. H. Branin, Jr., "The Relation Between Kron's Method and the Classical Methods of Network Analysis," IRE WESCON convention Record, Part 2, 3-28 (1959). Also available as Technical Report 00.686, IBM Development Laboratory, Poughkeepsie, New York, (1959)
6. F. H. Branin, Jr., "Machine Analysis of Networks," Technical Note 00.490, IBM Development Laboratory, Poughkeepsie, New York (1961)
7. R. J. Domenico, "Simulation of Transistor Switching Circuits on the IBM 704," Trans. IRE, Prof. group on Electronic Computers, EC-3, 242-247 (1957)
8. G. L. Lasher and J. C. Morgan, "A General Method of Predicting the Transient Response of a Nonlinear Circuit," IBM Research Report, RC-7 (1957)
9. T. R. Bashkow, "The A Matrix, New Network Description," Trans. IRE, Prof. Group on Circuit Theory, CT-4, 117-119 (1957).
10. F. B. Hildebrand, "Introduction to Numerical Analysis," McGraw-Hill, New York (1956), pp. 202-208
11. H. J. Gray, Jr., "Numerical Methods in Digital Real-Time Simulation," Quarterly of App. Math XII (2), 133-140 (1954)
12. H. M. Gurk, "The Use of Stability Charts in the Synthesis of Numerical Quadrature Formulae," Quarterly of Appl. Math, XIII (1), 73-78 (1955)
13. F. H. Branin, Jr., "An Abstract Mathematical Basis for Network Analogies and Its Significance in Physics and Engineering," AIEE preprint S-128, April 1961. Also available as Technical Report 00.781, IBM Development Laboratory, Poughkeepsie, New York (1961)
14. J. H. Beaudette and P. A. Honkanen," PE CANS, Circuit Analysis of Nonlinear Systems," (IBM 7090 Program writeup), IBM Development Laboratory, Poughkeepsie, New York (1961)
15. A. F. Malmberg, private communication.

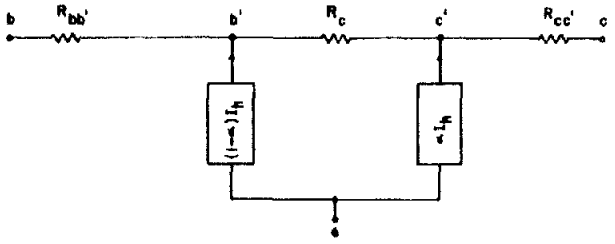
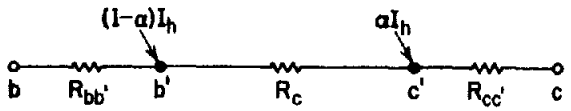


Fig. 1. Nonsaturating PNP Transistor Equivalent Circuit (D-C).



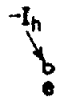


Fig. 2. Nonsaturating PNP Transistor Equivalent Circuit with Nodal Equivalent Current Sources.

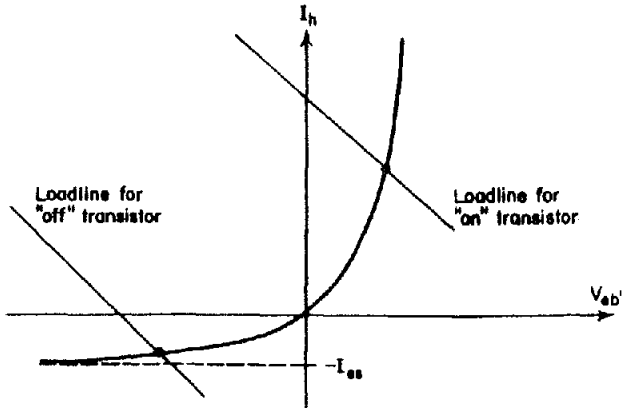


Fig. 3. The Nonlinear D-C Problem.

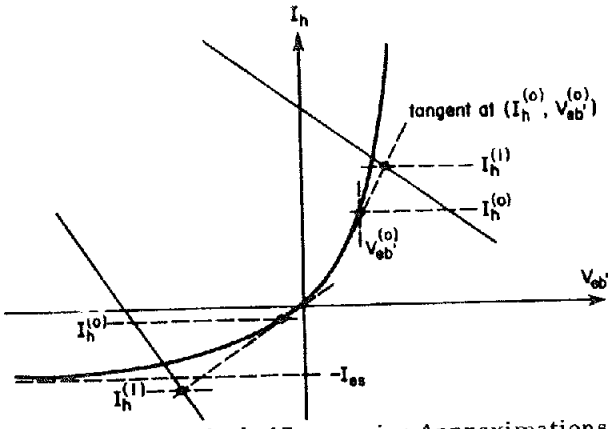


Fig. 4. The Method of Successive Approximations.

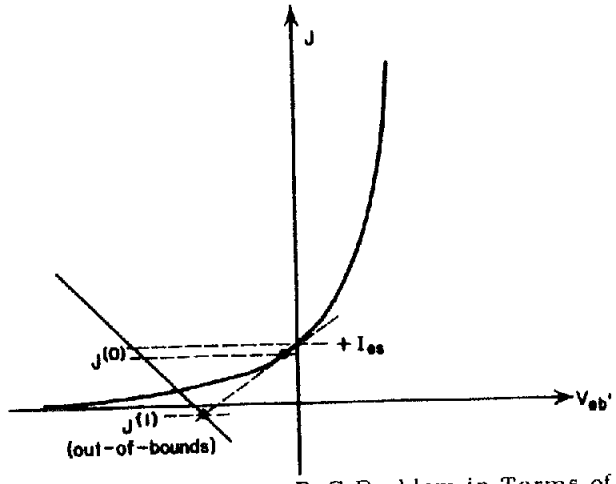


Fig. 5. The Nonlinear D-C Problem in Terms of J.

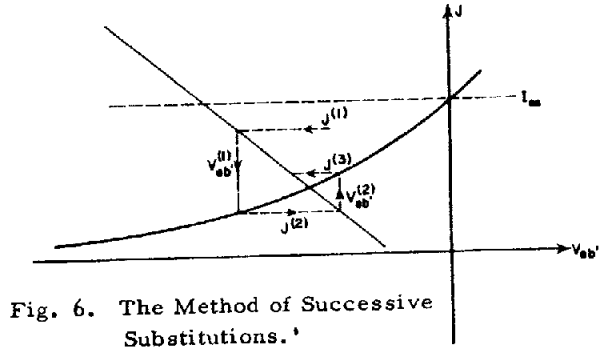


Fig. 6. The Method of Successive Substitutions.

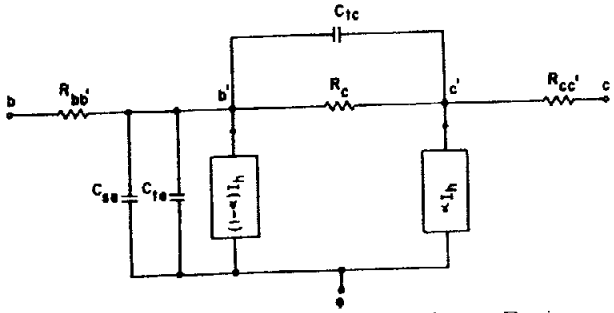


Fig. 7. Nonsaturating PNP Transistor Equivalent Circuit (Transient).

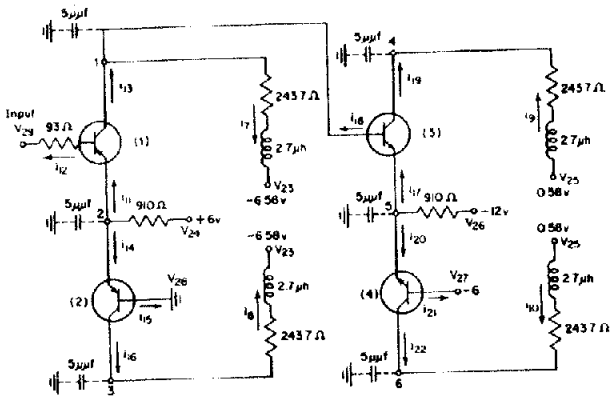


Fig. 8. Four-Transistor Switching Circuit.

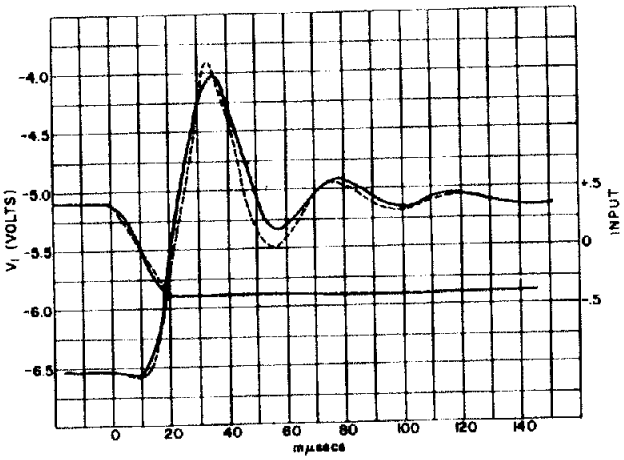


Fig. 9. Collector Voltage for Transistor 1, Negative Input Pulse.

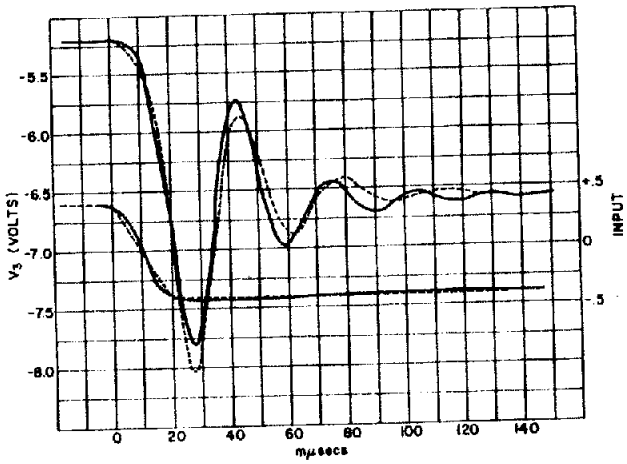


Fig. 10. Collector Voltage for Transistor 2, Negative Input Pulse.

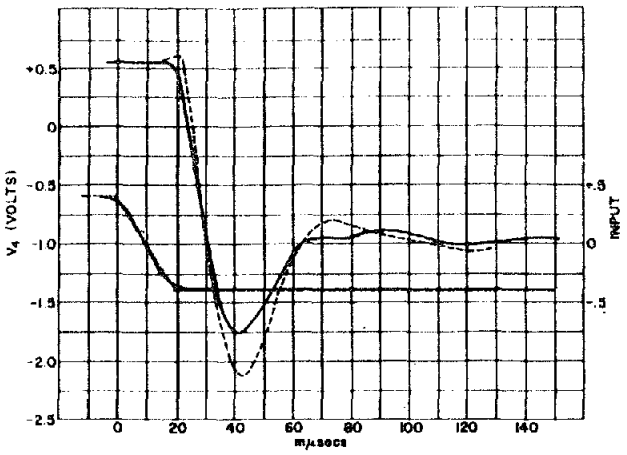


Fig. 11. Collector Voltage for Transistor 3, Negative Input Pulse.

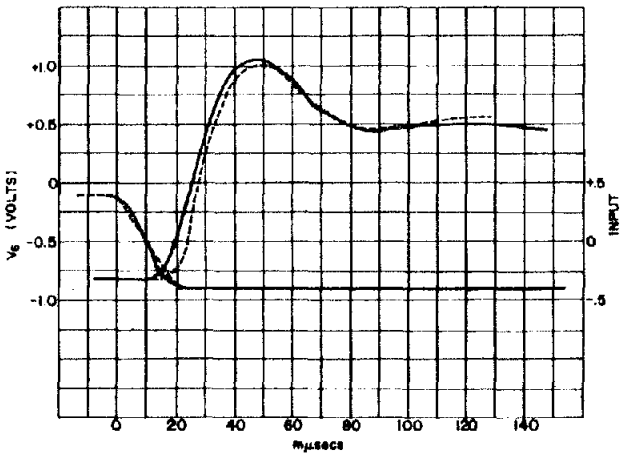


Fig. 12. Collector Voltage for Transistor 4, Negative Input Pulse.

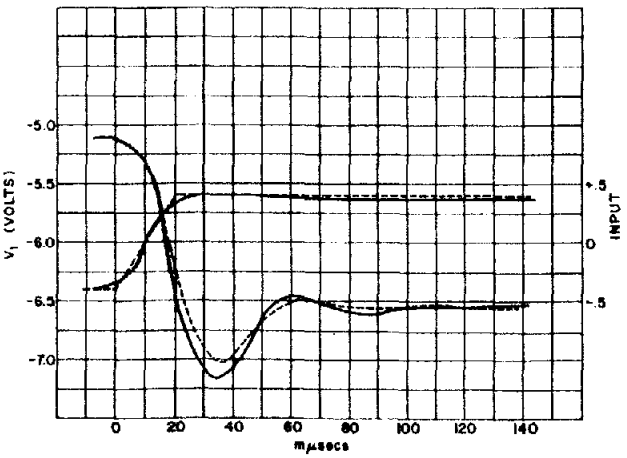


Fig. 13. Collector Voltage for Transistor 1, Positive Input Pulse.

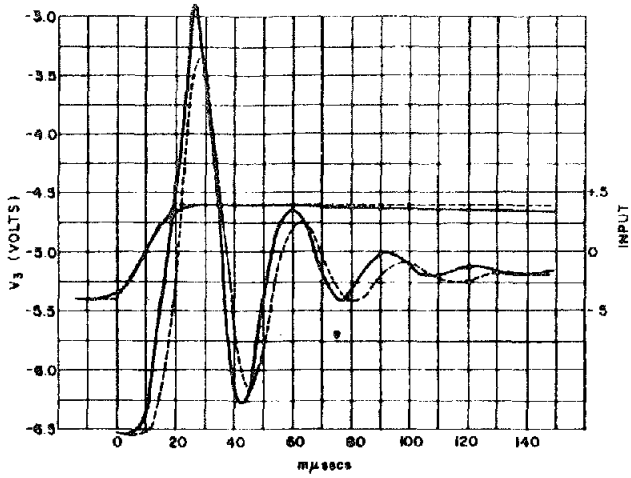


Fig. 14. Collector Voltage for Transistor 2, Positive Input Pulse.

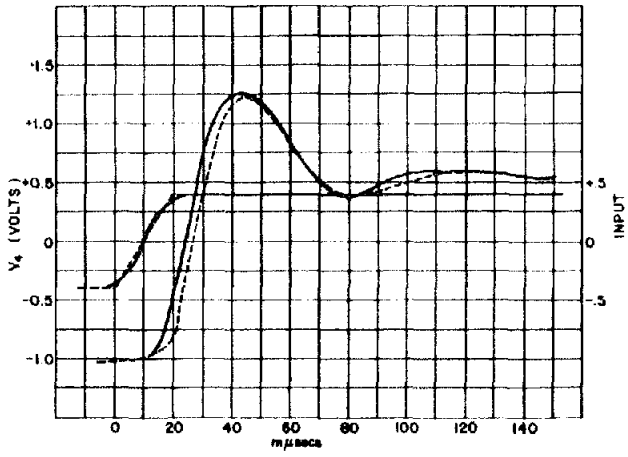


Fig. 15. Collector Voltage for Transistor 3, Positive Input Pulse.

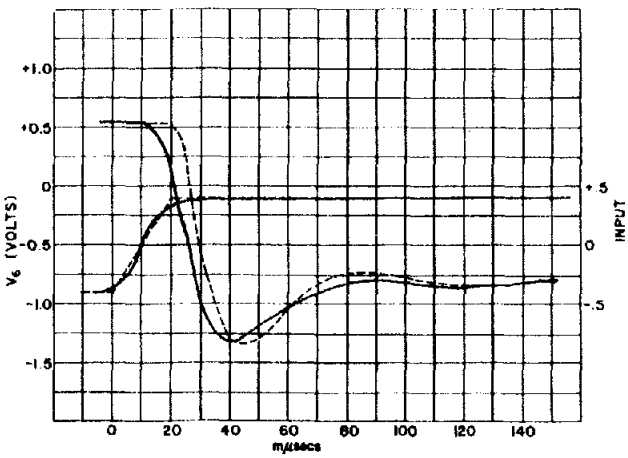


Fig. 16. Collector Voltage for Transistor 4, Positive Input Pulse.

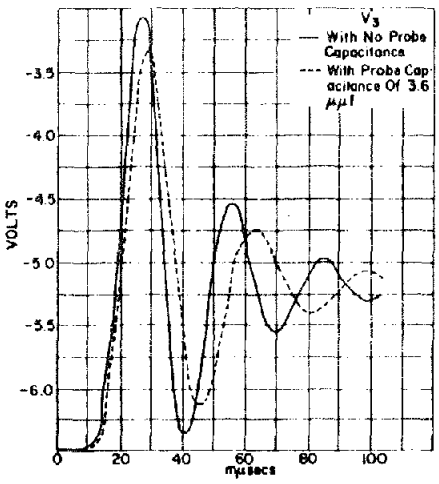


Fig. 17. Collector Voltage for Transistor 2, Positive Input Pulse (Showing Effect of Added Capacitance).

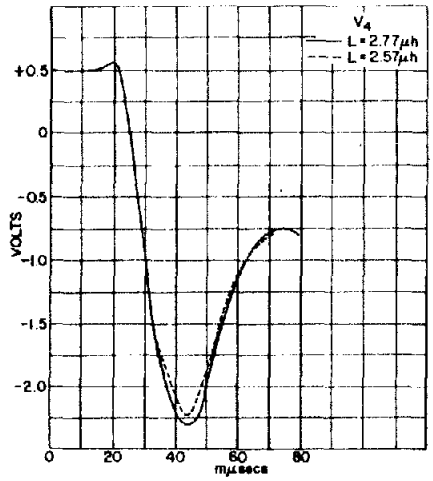


Fig. 18. Collector Voltage for Transistor 3, Negative Input Pulse (Showing Effect of Series Inductance).

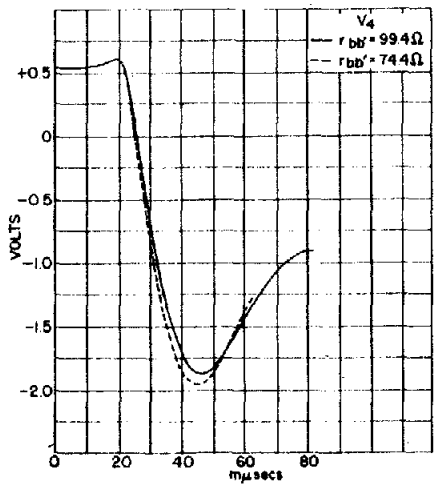


Fig. 19. Collector Voltage for Transistor 3, Negative Input Pulse (Showing Effect of Base Resistance).

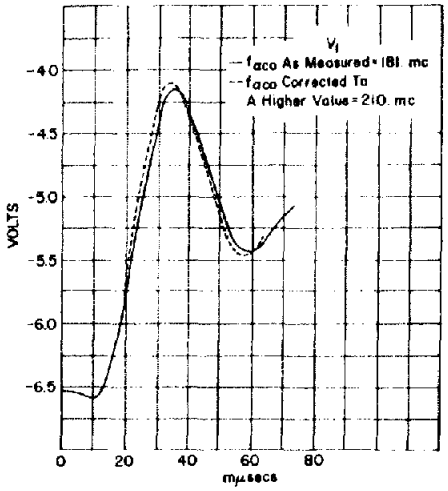


Fig. 20. Collector Voltage for Transistor 3, Negative Input Pulse (Showing Effect of Cutoff Frequency)

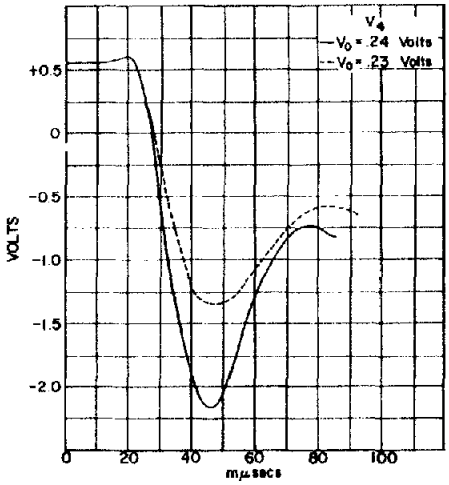


Fig. 21. Collector Voltage for Transistor 3, Negative Input Pulse (Showing Effect of Contact Potential).

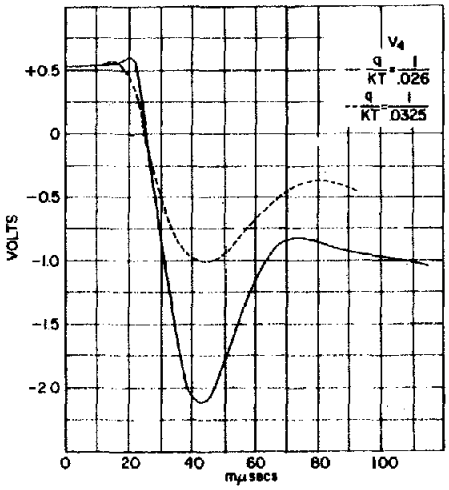


Fig. 22. Collector Voltage for Transistor 3, Negative Input Pulse (Showing Effect of Temperature).