ON THE IMPLEMENTATION OF AN ANALOG ATPG

by

CHIN-LONG WEY, B.S., M.S.

A DISSERTATION

IN

ELECTRICAL ENGINEERING

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for
the Degree of

DOCTOR OF PHILOSOPHY

ACKNOWLEDGEMENTS

I would like to express my sincere thanks to Paul Whitfield Horn Professor Richard E. Saeks for his expert guidance of this dissertation, Professors Kwong Shu Chao, John F. Walkup, Erich E. Kunhardt and Kazuo Nakajima for serving on my committee.

Thanks also to Mrs. Pansy Burtis for her fine work in typing this dissertation, and Mr. Henry Ford and Dr. Ashok Iyer for their suggestions. It is a pleasure acknowledging their help.

Finally a special thanks to my wife, Lih-Er, my parents, brothers, sisters and friends for their encouragement and support during my entire education.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Electronics design has become very sophisticated during the past quarter century. Graphical algorithms have been replaced by CAD (Computer-Aided Design), and features of design implementation can be studied by simulation, rather than requiring extensive breadboarding. Electronics maintenance, however, has changed very little during the same period. In fact, many industries have found that the life cycle maintenance costs for their electronics equipment exceed their capital investment. Consequently, it is becoming apparent that the new maintenance process, like the design process, must be automated.

Several formidable problems are faced in the maintenance of military electronics; the avionics and missiles are becoming far too complex for the typical military technician to maintain; the time required to test systems is becoming excessively large; system designs are changing too fast to keep maintenance documents current. Hence a more economical approach to maintenance is an actual necessity. Therefore, a multi-purpose automatic test equipment (ATE) which promises testing at computer speeds, fully automatic operation by low-skill operators, the virtual elimination of maintenance documents, and universal designs adaptable to any test problem through the flexibility of programming, has been investigated by several research groups.

Efforts at producing algorithms for automatic test program generation systems has concentrated mainly on digital circuits for which satisfactory solutions have been found. Several digital automatic test

program generation systems have been developed and widely used by both military and industrial communities. D-LASAR by Digitest, ATVG by General Electric, TGAS by the U. S. Navy, FAS/SDAP by Honeywell, LOGOS by Grumman, GLASH by Micro, SALT by IBM,[12] TESTAID-III by Hewlett-Packard,[36] etc., are some of the well known systems.

Several books[5,6,11,19,27,33] and articles[34,35] have discussed the fault detection and diagnosis in digital circuits. Typically, in the digital circuit, one assumes that all permanent component failures are either "stuck-at-zero," (s-a-0), or "stuck-at-one" (s-a-1).[17] Under this assumption, one hypothesizes some limit on the number of simultaneous faults and then simulates the responses of UUT (Unit Under Test) to a family of test vectors for each allowed combination of faults. The simulated responses are used to set up a fault dictionary which is stored in some bulk storage media such as disks and magnetic tapes. When the test is conducted the actual responses of UUT are compared with the responses in the fault dictionary to locate the failure. Of course, this approach is a kind of "brute force" search which requires one to simulate all possible responses to the various combinations of hypothesized faults. However, all these simulations need only be done once at the factory of a maintenance depot. The cost of simulation is therefore relatively cheap. Clearly, this approach is ideally suited for the maintenance environment. With the aid of some sophisticated software engineering, this apparently "brute force" approach to the fault diagnosis problem has slowly evolved into a workable concept.

program generation systems have been developed and widely used by both military and industrial communities. D-LASAR by Digitest, ATVG by General Electric, TGAS by the U. S. Navy, FAS/SDAP by Honeywell, LOGOS by Grumman, GLASH by Micro, SALT by IBM,[12] TESTAID-III by Hewlett-Packard,[36] etc., are some of the well known systems.

Several books[5,6,11,19,27,33] and articles[34,35] have discussed the fault detection and diagnosis in digital circuits. Typically, in the digital circuit, one assumes that all permanent component failures are either "stuck-at-zero," (s-a-0), or "stuck-at-one" (s-a-1).[17] Under this assumption, one hypothesizes some limit on the number of simultaneous faults and then simulates the responses of UUT (Unit Under Test) to a family of test vectors for each allowed combination of faults. The simulated responses are used to set up a fault dictionary which is stored in some bulk storage media such as disks and magnetic tapes. When the test is conducted the actual responses of UUT are compared with the responses in the fault dictionary to locate the failure. Of course, this approach is a kind of "brute force" search which requires one to simulate all possible responses to the various combinations of hypothesized faults. However, all these simulations need only be done once at the factory of a maintenance depot. The cost of simulation is therefore relatively cheap. Clearly, this approach is ideally suited for the maintenance environment. With the aid of some sophisticated software engineering, this apparently "brute force" approach to the fault diagnosis problem has slowly evolved into a workable concept.

Unfortunately, the above described success in the digital world has not been paralleled by progress in the analog world. The difficulty arises from a number of characteristics of the analog problem which are not encountered in digital circuits, namely,

(1) Analog systems have a continuum of possible failures. These failures may range from short circuit to open circuit,

(2) A good component may be "in tolerance" but not nominal,

(3) Complex feedback structures are encountered,

(4) Simulation is slow and costly because analog systems are frequently nonlinear,

(5) Post-fault component characteristics may not be known, and

(6) A fault in one component may induce an apparent fault in another.

Items (5) and (6) imply that the kind of "brute force" fault simulation algorithm associated with the digital problem will not be applicable to the analog or hybrid case.

A number of academic researchers have proposed a variety of analog fault diagnosis algorithms.[8,28,29] Conceptually, these algorithms can be subdivided into three classes;[31]

(i) simulation-before test,

(ii) simulation-after-test with a single test vector, and

(iii) simulation-after-test with multiple test vectors.

The first is commonly employed in digital testing and is characterized by minimal on-line computational requirements, but the high cost of analog circuit simulation coupled with the large number of potential fault modes limits the applicability of this algorithm. Typically, the

simulation-after-test technique attempts to model the analog fault diagnosis problem as a nonlinear equation in which the internal variables or component parameters are computed in terms of the test data. In this case, where sufficiently many test points are available, only a single test vector is required and the problem reduces to the solution of a linear equation.[32,38] Therefore, the on-line computational requirements are moderate. However, the test point requirements grow linearly with circuit complexity. To reduce the test point requirements one may consider using multiple test vectors to increase the number of equations obtained from a given set of test points. However, the on-line computation required to solve these complex sets of nonlinear equations (even for linear systems) is extremely expensive.

Comparing the above three techniques, the simulation-after-test, with single test vector, seems to be the closest to the "ideal" algorithm.[31] The remaining question is how to reduce the number of test points so that this algorithm can be made more applicable. An algorithm based on the simulation-after-test, with single test vector, was presented to reduce the number of test points.[41,42]

The Self-Testing Algorithm

A bound on the maximum number of simultaneous failures is used to reduce the test point requirements while still retaining the computational simplicity inherent in a single test vector algorithm. It is reasonable to assume that, at most, two or three components have failed simultaneously in a given circuit with several hundred IC's and/or discrete components. In fact, rather than solving a set of simultaneous

equations in n-space, the solution to our fault diagnosis problem actually lies in a two- or three-dimensional submanifold which should yield a considerable reduction in test point requirements. Unfortunately, we do not know which two or three have failed and a further search is still required. Fortunately, with the aid of an appropriate decision algorithm the required search can be implemented quite simply.

Conceptually, the components (individual chips, discrete components or subsystems) are subdivided into two groups at each step of the test algorithm. At each step we assume that one group is composed of good components and we use the known characteristics of these components, together with the test data, to determine whether or not the remaining components are good. In effect the first group of components is testing the second, hence the "self-test" algorithm. Of course, if the testers are actually good, then the resultant test results for the remaining components will be reliable. On the other hand, if any one of the testers is faulty the test data on the remaining components will be unreliable. Consequently, we repeat the process at the next step of the test algorithm with a different subdivision of components.

Of course, the number of components which may be tested at any one step is dependent on the number of test points available, while the number of steps required is determined by the number of components which may be tested at any one step and the bounds on the maximum number of simultaneous failures. Therefore, this procedure yields a natural set of tradeoffs between the number of test points, simultaneous failures and steps required by the algorithm.

Overview of Automatic Testing Programming

The Purpose of this overview is to introduce the Automatic Test Equipment (ATE) which provides test data to our test program. A collection of papers relating to hardware, software, and management aspects of automatic test equipment was edited by Liguori.[20] In Knowles' book[18] he introduced the automatic test systems and its applications. He gave a review of the elements which comprise automatic test systems, a survey of those factors which affect the choice of test systems, and a discussion on the planning studies which should precede any decision to adopt ATE. Another book written by Healy[16] concentrates on the automatic testing of the digital integrated circuits. Several articles published in IEEE Sprectrum also survey the ATE systems.[9,13,24,37]

A test is a process which is not only performed to obtain information about the performance of a component or device, it is also allowed to detect, locate, or identify faults. The electronic component or device which is to be tested is called a unit under test (UUT). Fault detection is a procedure for evidencing the presence of faults in a system, which is performed either during quality control or during maintenance. Fault Location determines the faulty element after detection of a fault. Fault Diagnosis or identification determines the causes of a fault.

There are essentially two purposes for testing: First, to determine whether or not the UUT is bad (functional testing), and then to find out which element is faulty and needs to be repaired (fault isolation). Testing may be performed manually or automatically.[24] Manual testing is

usually performed by connecting individual pieces of test equipment, including measurement devices, special-purpose signal generators, power supplies, decade boxes, and a collection of clip leads. The technician must plug in, set up, and connect all of this equipment to the UUT to make the tests he feels are pertinent by the help of some sort of manual or set of instructions.

Automatic testing is a test procedure which is performed with the aid of a computer. There are power supplies, stimuli, measurement devices, and a switching system to allow the equipment to be arranged in desirable configurations. The instruction manual is replaced by a program file which instructs the computer to carry out test instructions in the proper sequence, judge test results, and/or perform calculations. The test results are either written by a line printer or displayed on a CRT terminal.

A block diagram of a typical ATE system is shown in Figure 1.1. The switch unit is an equipment which is used to connect the device UUT/ATE interface to the test system and to vary the connections of the device terminals. Examples are multiplexers, relay trees, scanner, and so forth. The stinulus unit is a device which generates stimuli, such as power supplies, oscillators, synthesizes, function generators, waveform generators, and D/A converters, among others. For use in automatic test such sources are often required to be programmable; that is, all of their functions should be controllable by electric signals instead of manual control.[24] The measurement unit is an instrument which quantifies the response of a UUT to stimuli. The response of a UUT may be a

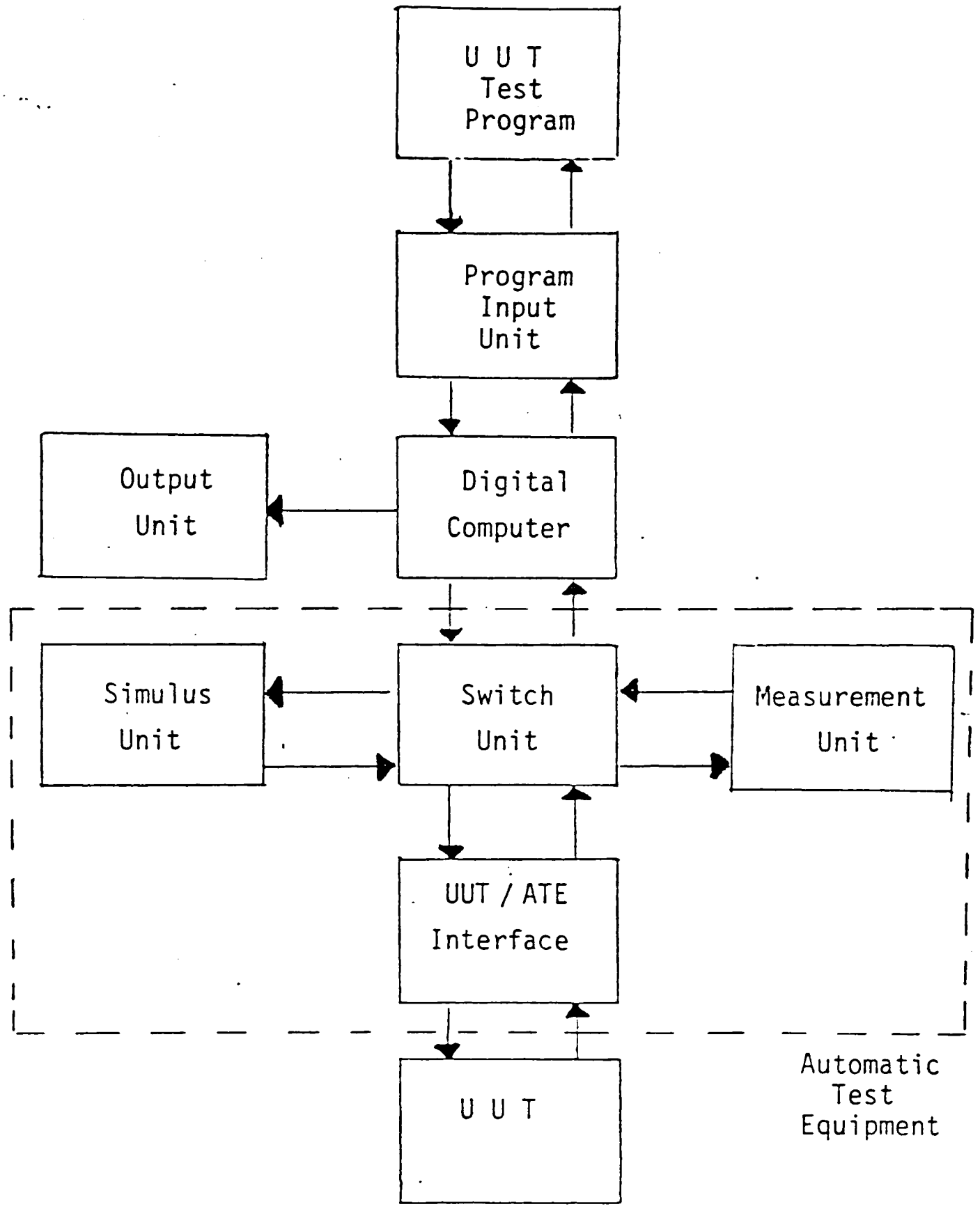


Figure 1.1. Simplified Block Diagram of Typical ATE

directly measurable quantity such as a voltage, or it may be a derived quantity from physical measurements such as a resistance. Therefore, these units include voltmeters, current meters, phase meters, impedance bridges, frequency counters, A/D converters, etc. UUT/ATE interface provides the connection of the UUT to the ATE which has the incompatible connecting points. Usually, an adapter is used to define items that interface the UUT to the switch unit. It may be test boards, fixtures, or sockets that contact the leads or terminals of a component. Output unit is a device which is used to display the test results. It may be a CRT terminal or line printer, or may transfer this test data to the host computer which provides a fault diagnosis.

The testing starts after connecting the UUT to the test equipment through an interface. The computer output, a synthesis of an electrical signal, is converted by means of a digital-to-analog (D/A) converter into voltage or current levels. These signals are applied to the UUT via a switching unit and interfaces. The test results are routed via the switching unit to a sampling instrument. The measured quantity is converted to digital representation, and this time series is analyzed by the computer. The computer output of the results can be presented in any form most suitable to the users' display or print-out requirements.

Organization

The purpose of this dissertation is to present an Analog Automatic Test Program Generation, AATPG, for both linear and nonlinear circuits based on the self-testing algorithm. In Chapter 2 a Component Connection

Model, CCM, is described. This model is used to formulate our test algorithm for both linear and nonlinear circuits. The simulation model is used to test one set of components under the assumption that the remaining components are good. Based on this model the software development of AATPGs for both linear and nonlinear circuits are discussed in Chapter 3. Each code is subdivided into off-line and on-line components. The off-line component is used by the test system designer to input nominal system specification to generate the test program and data base which is used by an on-line component. To verify the software's ability to locate and detect the faulty component(s), Program Verification is used. Similarly, the test program is validated by the measurement of the actual UUT. The on-line component is used in the implementation of the actual test. In order to run the actual test in a fully automatic mode, several supporting algorithms and decision algorithms are discussed in Chapter 4. Examples for both linear and nonlinear circuits are presented in Chapter 5. The conclusions follow in Chapter 6.

CHAPTER 2

THE SIMULATION MODEL

Although our test algorithm can be formulated in terms of any of the standard circuit or system models, for the purpose of this exposition we will assume a component connection for the circuit or system under test.[7] The component connection model naturally divides the system into two sets of equations: Component equations characterized by (block) composite component model (for linear case) or by decoupled state model (for nonlinear case), and the Connection equations characterized by coupled linear algebraic equations. Equations (2.1) and (2.20) are the component equations of linear and nonlinear systems, respectively, while the equations (2.2) and (2.3) are the connection equations. The connection matrix L, equations (2.2) and (2.3), characterizes the connection of components in the system. At each step of the algorithm, a "pseudo circuit" is generated and formulated by the equations (2.9) through (2.15) with a new connection matrix K. The data base which is used by the on-line component is computed by equation (2.16), matrix M for linear case while a SPICE code, based on the equations (2.23) through (2.26), is generated for the nonlinear case.

Linear Case

In the linear case the UUT is represented by a composite component model characterizing its components and/or subsystems together with an algebraic connection equation as follows:

$$b = Za \tag{2.1}$$

and

$$b^2 = Z^2 a^2 \tag{2.5}$$

Here, Z^1, a^1 and b^1 are the vectors of group "1" transfer functions, component input and output variables; and similarly for Z^2, a^2 and b^2. To retain notational compatibility with equations (2.4) and (2.5) we re-order and partition the connection equations of (2.2) and (2.3) to be conformable with (2.4) and (2.5) as follows:

$$a^1 = L_{11}^{11} b^1 + L_{11}^{12} b^2 + L_{12}^{1} u \tag{2.6}$$

$$a^2 = L_{11}^{21} b^1 + L_{11}^{22} b^2 + L_{12}^{2} u \tag{2.7}$$

$$y = L_{21}^{1} b^1 + L_{21}^{2} b^2 + L_{22} u \tag{2.8}$$

Unlike the commonly encountered circuit analysis problem, in which one desires to simulate the output responses y of a given circuit, in our application the vector y is obtained by the test engineer measuring the responses at various test points. The test responses y are, therefore, known for the purpose of our application. Given equations (2.4) through (2.8), our goal is to compute the group "2" component variables, a^2 and b^2. To this end we assume that L_{21}^{2} admits a left inverse, which, in turn, determines the allowable component subdivisions. Under this assumption one may then formulate a component connection model for a "pseudo circuit" composed of the group "1" components with external input vector $u^p = \text{col}(u,y)$ and external vector $y^p = \text{col}(a^2,b^2)$ in the form,

$$b^1 = Z^1 a^1 \tag{2.9}$$

and

$$a^1 = K_{11}\, b^1 + K_{12}\, u^p \tag{2.10}$$

$$y^p = K_{21}\, b^1 + K_{22}\, u^p \tag{2.11}$$

where K is the connection matrix of the pseudo circuit. Some algebraic manipulation of equations (2.6) through (2.8), together with the assumption that $[L_{21}^2]^{-L}$ exists, will yield

$$K_{11} = [L_{11}^{11} - L_{11}^{12}\, [L_{21}^2]^{-L}\, L_{21}^1] \tag{2.12}$$

$$K_{12} = [L_{12}^1 - L_{11}^{12}\, [L_{21}^2]^{-L}\, L_{22} \mid L_{11}^{12}\, [L_{21}^2]^{-L}\,] \tag{2.13}$$

$$K_{21} = \left[\begin{array}{c} L_{11}^{21} - L_{11}^{22}\, [L_{21}^2]^{-L}\, L_{21}^1 \\ \hline -\, [L_{21}^2]^{-L}\, L_{21}^1 \end{array}\right] \tag{2.14}$$

$$K_{22} = \left[\begin{array}{c|c} L_{12}^2 - L_{11}^{22}\, [L_{21}^2]^{-L}\, L_{22} & L_{11}^{22}\, [L_{21}^2]^{-L} \\ \hline -\, [L_{21}^2]^{-L}\, L_{22} & [L_{21}^2]^{-L} \end{array}\right] \tag{2.15}$$

For each pseudo circuit, substituting equation (2.9) into equations (2.10) and (2.11), the equations with the transfer function matrix M are shown as follows:

$$a^1 = (K_{11}\, Z^1)\, b^1 + K_{12}\, u^p$$

$$y^p = (K_{21}\, Z^1)\, b^1 + K_{22}\, u^p$$

and

$$y^p = M\, u^p$$

where

$$M = K_{21}\, Z^1\, (I - K_{11}\, Z^1)^{-1}\, K_{12} + K_{22}\ . \tag{2.16}$$

Specifically,

$$a^2 = M_{11}\, u + M_{12}\, y \tag{2.17}$$

$$b^2 = M_{21}\, u + M_{22}\, y \tag{2.18}$$

Since the matrix M is independent of the test data and computed in terms of the nominal values of the group "1" components, it may be computed off-line and stored in a data base to be retrieved at the time a test is conducted. Furthermore, since only a single test vector is required, single frequency testing can be employed. In this case M need only be computed at a single frequency. The only on-line computation required for the fault diagnosis of a linear system is the matrix/vector multiplication indicated by equations (2.17) and (2.18) together with the computation of

$$\hat{b}^2 = Z^2\, a^2 \tag{2.19}$$

to determine which, if any, of the group "2" components are bad.

Nonlinear Case

For the nonlinear case one may formulate an identical algorithm in which the component characteristics are represented by a set of decoupled state models, together with an algebraic equation as follows:

$$\left.\begin{aligned} \dot{x}_i &= f_i(x_i, a_i) \\ b_i &= g_i(x_i, a_i) \end{aligned}\right\} \quad ; \; x_i(0) = 0 \; ; \; i = 1,2,..n \tag{2.20}$$

and

$$a = L_{11}\, b + L_{12}\, u \tag{2.2}$$

$$y = L_{21}\, b + L_{22}\, u \tag{2.3}$$

Here, x_i's are the component state variables. The component equation (2.20) is modeled in the time domain.

Similarly, as in the linear case, the components are subdivided into two groups. The variables in (2.20) are then partitioned as

$$\dot{x}^1 = f^1(x^1,a^1)$$
$$; \quad x^1(0) = 0 \tag{2.21}$$
$$b^1 = g^1(x^1,a^1)$$

and

$$\dot{x}^2 = f^2(x^2,a^2)$$
$$; \quad x^2(0) = 0 \tag{2.22}$$
$$b^2 = g^2(x^2,a^2)$$

The connection equations (2.2) and (2.3) are partitioned as the equations (2.5) through (2.7).

For each component subdivision, a pseudo circuit is generated in the form,

$$\dot{x}^1 = f^1(x^1,a^1)$$
$$; \quad x^1(0) = 0 \tag{2.23}$$
$$b^1 = g^1(x^1,a^1)$$

$$a^1 = K_{11}\, b^1 + K_{12}\, u^p \tag{2.24}$$

$$y^p = K_{21}\, b^1 + K_{22}\, u^p \tag{2.25}$$

where the connection matrix K of the pseudo circuit is derived in the equations (2.12) through (2.15). Since in our test algorithm both u and y are known, the above equations can be solved via any standard circuit analysis code, SPICE,[25] NAP2,[26] etc., to compute $y^p = \mathrm{Col}(a^2,b^2)$. Once the values a^2 and b^2 are computed, with the computation of

$$\dot{x}^2 = f^2(x^2, a^2)$$
$$\hat{b}^2 = g^2(x^2, a^2) \qquad ; \; x^2(0) = 0 \tag{2.26}$$

to determine which, if any, of the groups two components are faulty.

However, the above test results are dependent on our assumption that the group "1" components are not faulty, they are not immediately applicable. A decision algorithm is required to cope with this ambiguity problem so that the actual fault(s) can be precisely identified. Following the philosophy initiated by Preparata, Metze, and Chein[2,14,30] in their study of self-testing computer networks, if one assumes a bound on the maximum number of components, it is possible to determine the actual fault(s) from an analysis of the test results obtained at the various steps of the algorithm. To this end we have derived the complete theory required to locate a single fault, together with Boolean algebraic and heuristic methods, which is applicable to the multiple fault case.[39,40,42]

CHAPTER 3

SOFTWARE DEVELOPMENT

The AATPG code for both linear and nonlinear circuits is subdivided into off-line and on-line components. The former, corresponding to the test system design stage, is used by test system designer to input nominal system specifications to generate a data base which is used by the on-line component. To implement the actual test, the field engineer invokes the on-line component input data describing the UUT: the assumed maximum number of simultaneous failures, the type of decision algorithm to be employed, and the source of the test data. The actual test can then be run in a fully automatic mode or interactively.

As illustrated in Figure 3.1, a circuit description and test objectives are given to the off-line component to generate the test program. Necessary changes are indicated if the resultant test does not satisfy all of the requirements. If the design is satisfactory, the off-line component will generate the necessary data for the on-line test program and the data base. In the test package, the greatest part of the required computation is carried out by the off-line component with the "pseudo-internal test data" being obtained from the test measurements via a simple on-line matrix/vector multiplication of equations (2.17) and (2.18). To the contrary, in the nonlinear code SPICE is used to evaluate the "pseudo-internal test data" via the on-line simulation of an appropriate pseudo circuit.

In our implementation, to use the actual measured test data with these AATPGs, a HP 9825A is used to control special purpose ATE

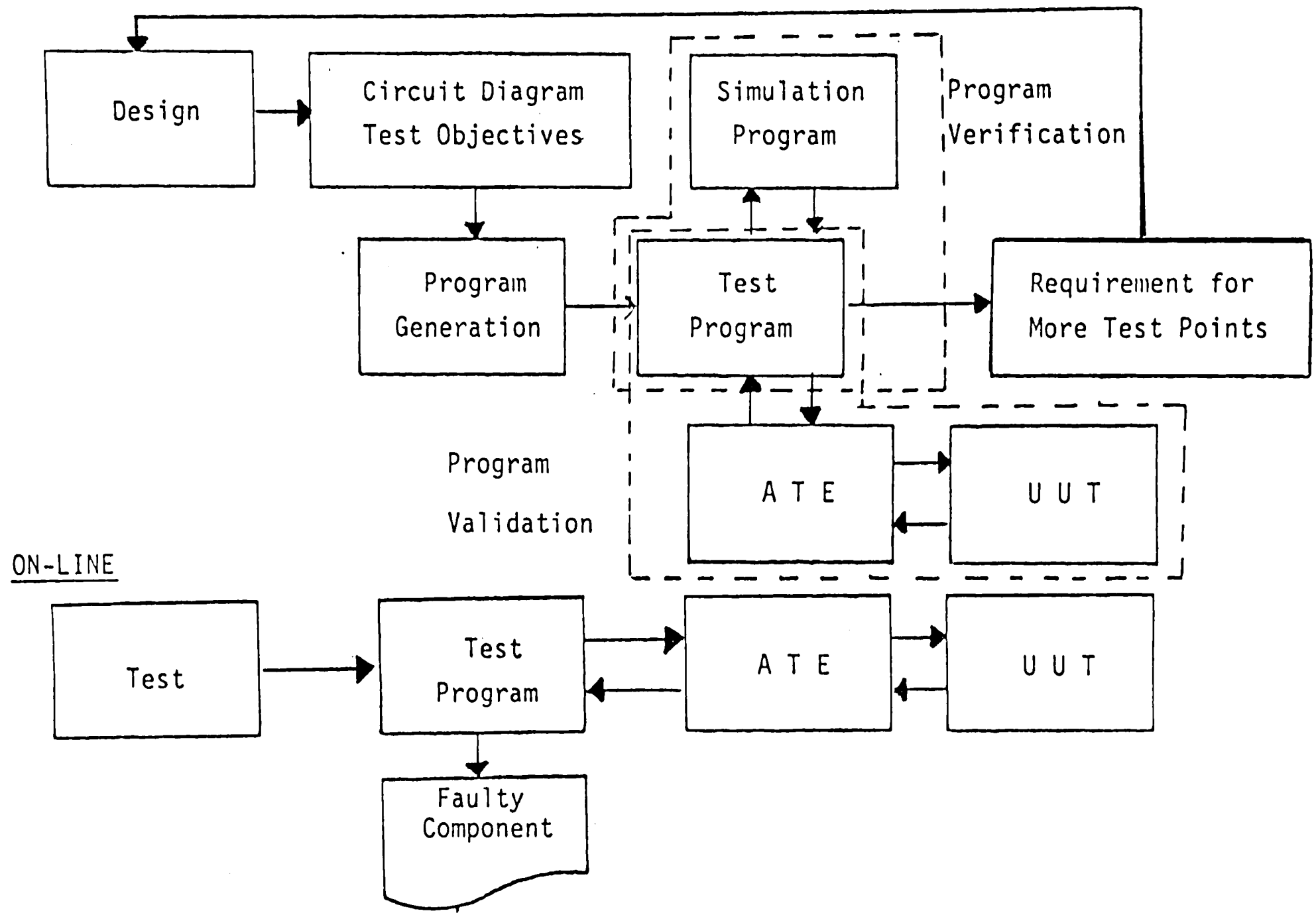


Figure 3.1. Design and Test of a Unit Under Test (UUT)

(Automatic Test Equipment) which generates test signals, and stores the measured test results. After some necessary calculations, the data is transferred to the host (VAX 11/780) where the on-line component of the ATPG takes over.

Both the off-line and on-line components have user-oriented interfaces to simplify the process of generating a new test program. The AATPG has been implemented on a VAX 11/780 in FORTRAN 77 and DCL (Dec Command Language).[39,40] In the linear code the user specifies the circuit in terms of certain standard elements, while in the nonlinear code standard SPICE circuit models are employed. The input syntax is a free-format style.

Linear Case

Off-line Component: Design Stage

The objective of the off-line component is to generate the test program and the appropriate data for the test program, verify and validate the test program so that the faulty components can be actually detected and located.

Test Program Generation: As illustrated in Figure 3.2, the input requirements in the test program generation are Circuit Description, Input Frequency, and Accessible Test Terminals.

Circuit Description: The Component Connection Model is used to simulate the UUT under nominal and faulty conditions. In the case of a linear circuit, the UUT components are characterized by a composite component model and the component equations are modeled in frequency domain. The circuit description consists of two steps, namely:

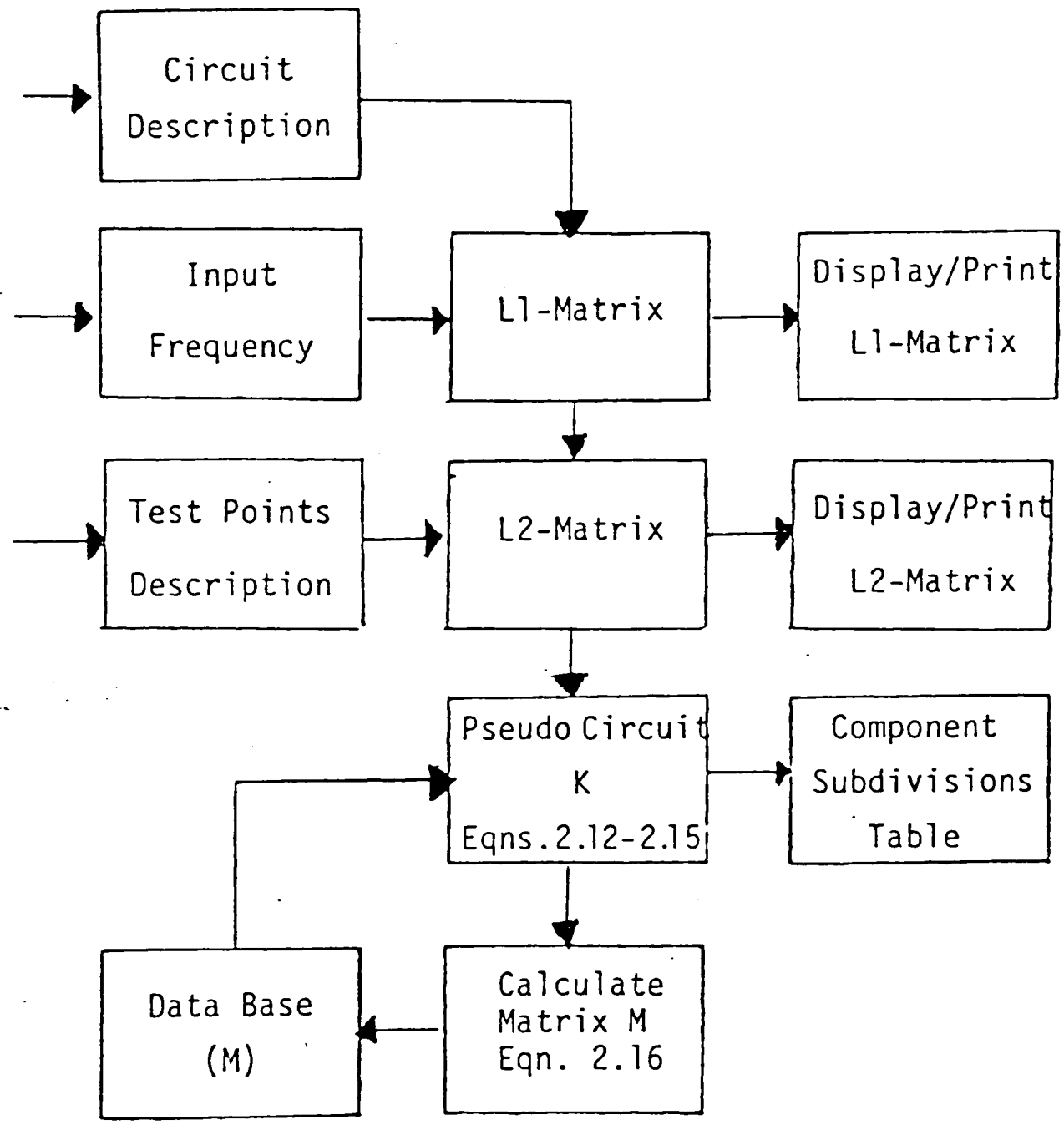


Figure 3.2. Test Program Generation - Linear Case

(1) Component Description,

(2) Source Description.

In the former, the user is required to input the component type (the component types currently available in this package are resistors, inductors, capacitors, transistors, op-amps, and transformers), give a unique name to each component (where the first letter identifies the component type), define the value of the appropriate elements by the input of a numerical constant, and indicate the current flow direction by specifying the nodes. Nodes must be nonnegative integers, but need not be numbered sequentially. The ground node must be numbered zero. In the source description, the independent sources are assumed to be located in series/parallel with a component, therefore, the branch which contains the source will be specified as well as the orientation of this branch.

Input Frequency: The component equations for our linear circuits are modeled in the frequency domain. Since a single test vector is required, single frequency testing can be employed. With this single frequency, the component transfer matrix Z is generated by computing the impedance or admittance for each one port component, such as a resistor, capacitor, or inductor. The square matrix Z contains all zeros everywhere except the diagonal blocks (where the block size depends on the number of ports of each component). For the two-port components, such as transistor, the hybrid pi parameters are used to characterize the component, and the dimension of the diagonal block is two rows by two columns with the h-parameters or the transformed parameters depending

upon the entries selected in the vector a. Together with the data generated in the above description, the first part of the connection is generated as shown in equation (2.2).

Accessible Test Terminals: The accessible Test Terminals are used to generate the second part of the connection matrix. In our test package the user is required to input manually the test point locations where the test points may be current measurements through components or voltage measurements across any two nodes. However, the test points may not be an entry of the vector a. With the assumption that the matrix $[L_{21}^{2}]^{-L}$ exists, a component subdivisions table is generated, where the elements in each subdivision are the group "2" components. As discussed in the previous section, given a subdivision, a "pseudo circuit" with the connection matrix K is created by computing equations (2.12) through (2.15), and the data base, M-matrix, is derived by using equation (2.16).

At the end of the test program generation process, the following data files are created; the connection matrix L, the component transfer matrix Z and the external input vector u, the component subdivisions table, the data base (M-matrix), and the test program.

Program Verification: The generated test program is tested in software, as shown in Figure 3.3, to verify its ability to detect and locate faults. To verify the test program, three tests are performed on each circuit, namely:

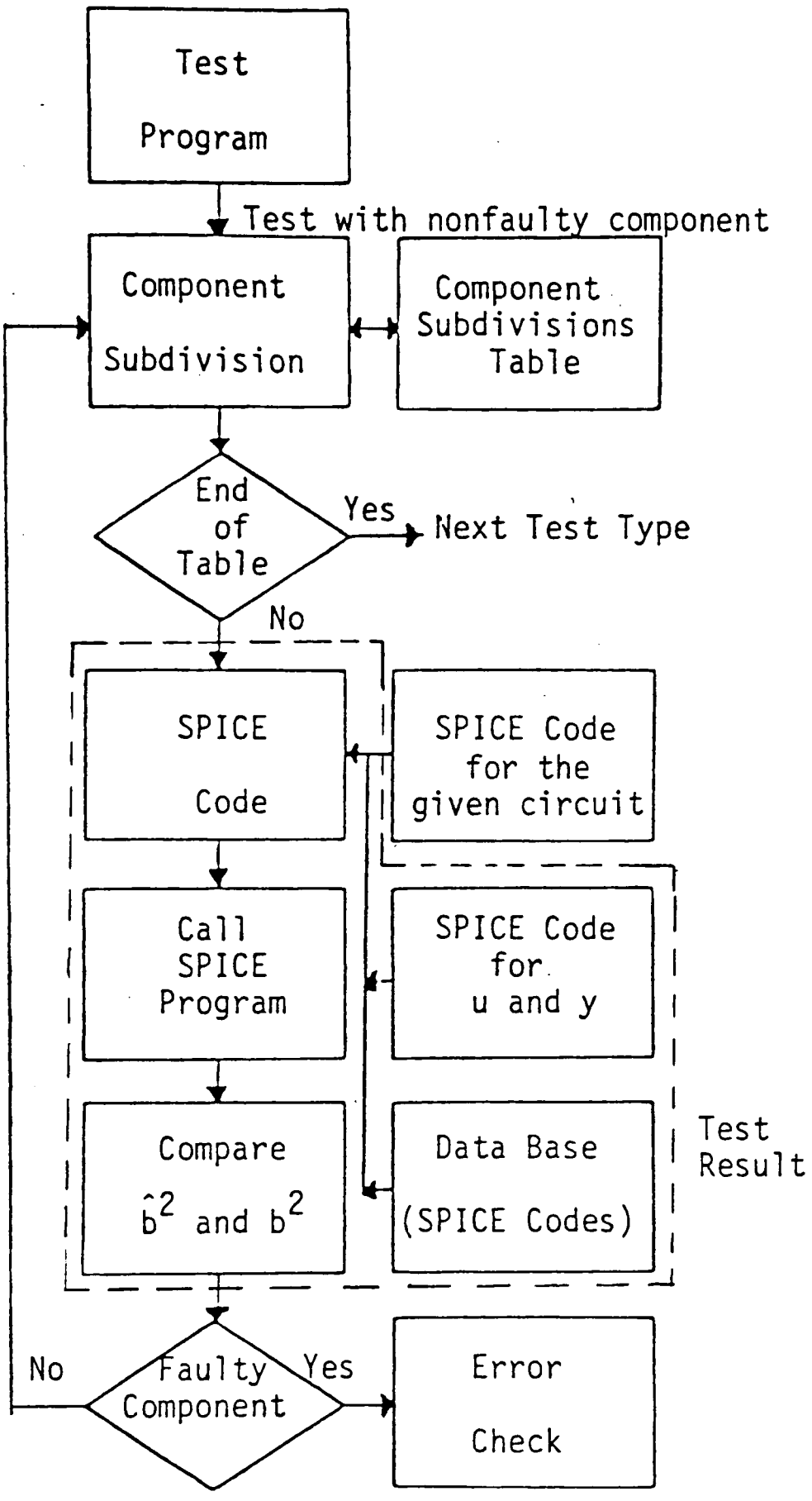


Figure 3.3. Program Verification - Linear Case

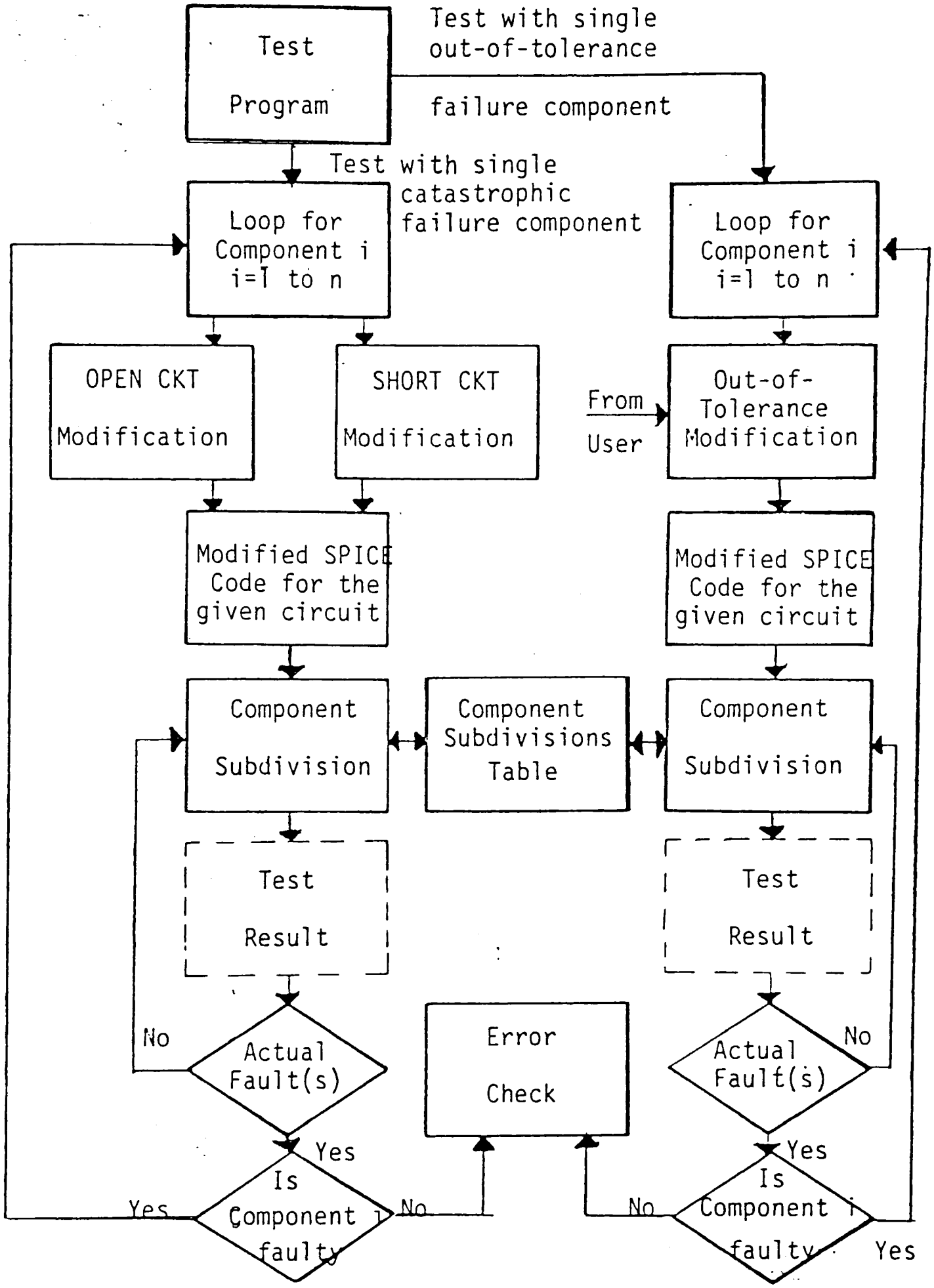


Figure 3.3. (Continued)

(1) A test with nonfaulty components,

(2) Tests with single catastrophic (open and short circuits) failures, and

(3) Tests with a single out-of-tolerance failure.

The first test verifies the correctness of the test program, while the remaining tests are performed to check whether or not the selected test points can actually locate and detect the faults. If the test is not satisfactory, the design engineer may change the test points and repeat the process of program generation. The source of test data for program verification is a simulation program in which the test data, y, is calculated by the following equation.

$$y = [L_{21} Z (I - L_{11} Z)^{-1} L_{12} + L_{22}] u \qquad (3.1)$$

In the first test, since no faulty component is assumed, all the group "1" components are good so that the test results should be reliable. If one of the test results is found to be bad in an arbitrary subdivision, which contradicts our nonfaulty component assumption, the process will be terminated with an error message. An error check routine is then loaded to check for correctness of the Circuit and Test Point Descriptions. If no error is detected, the program generation process is repeated with new test points. If no faulty component is detected in all possible subdivisions, the catastrophic failure test will be executed. In this test, components, taken one at a time, are modeled as open circuits. Using the calculated y of equation (3.1), the test results are obtained, and the actual faulty component will be located by the exact single fault algorithm. The details of the decision algorithms will be discussed later.

After the test with the open circuits for each component is executed successfully, a similar test where the components are short circuited is performed. If either one of the above tests fails the error check routine will be loaded. If all the above tests are processed successfully, similar tests for out-of-tolerance failures are executed. Once all tests are completed successfully, the test program is assumed correct.

Following the successful test program verification, the test program is validated by the measurement of the actual UUT, as shown in Figure 3.4.

On-line Component: Test Stage

The implementation of the actual test on a UUT is as illustrated in Figure 3.5. The maximum number of simultaneous failures allowed is specified by the user; the data files (matrix Z and vector u) and the test program are loaded and the host computer will send an instruction to the HP 9825A controller to conduct the test measurement. After the ATE is instructed to generate stimuli for the UUT, the measured test data will be stored and transferred to the host computer. When the data transfer is completed, the test program will compute the test result and identify the faulty component(s) with the aid of the decision algorithm.

Nonlinear Case

Off-line Component: Design Stage

In the nonlinear case, the objective of the off-line component is to generate the test program, test data for test program use, and SPICE codes. These SPICE codes are generated for the given circuit, test inputs u and test data y, and pseudo circuits with equations (2.23) through

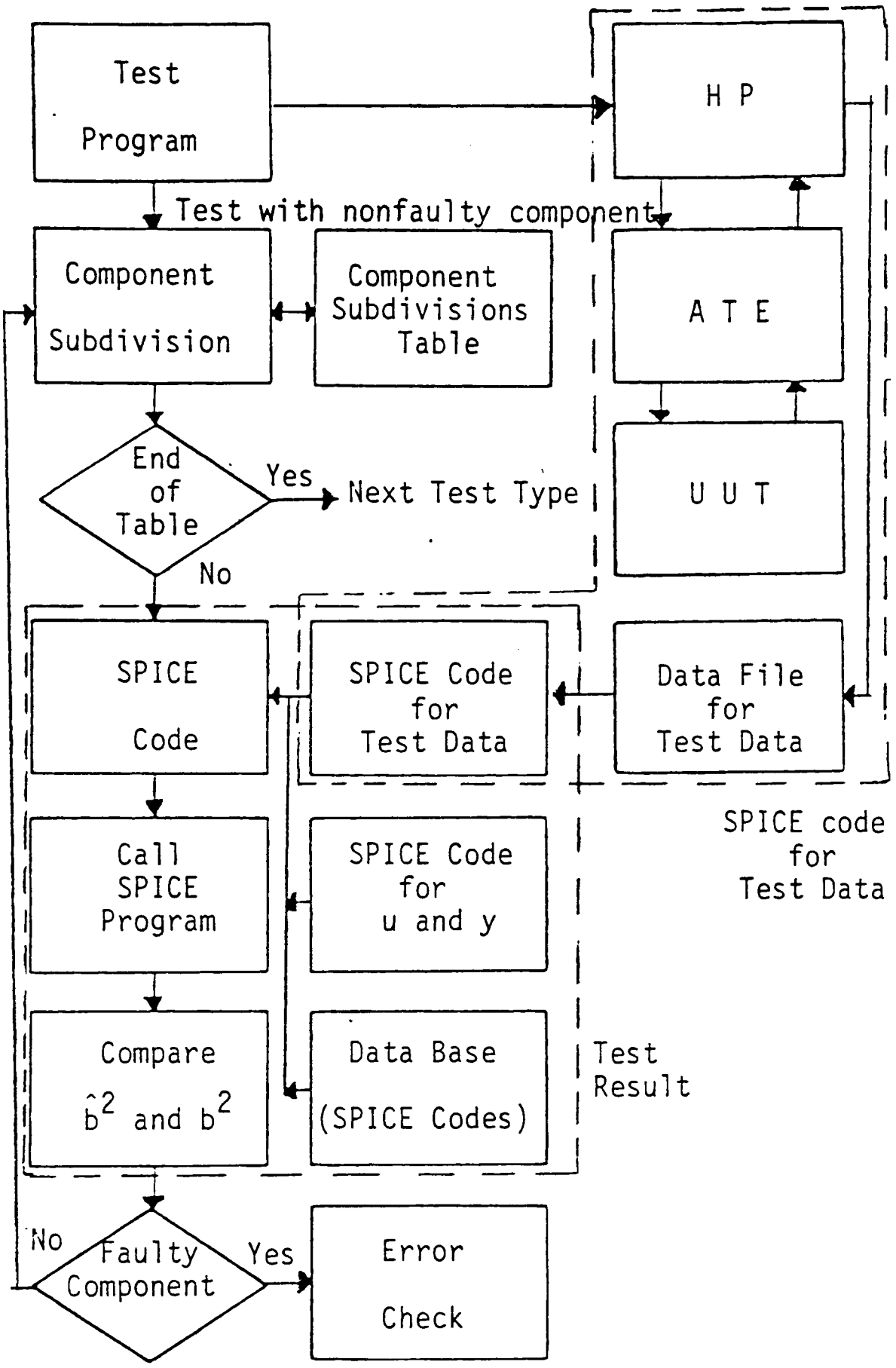


Figure 3.4. Program Validation - Linear Case

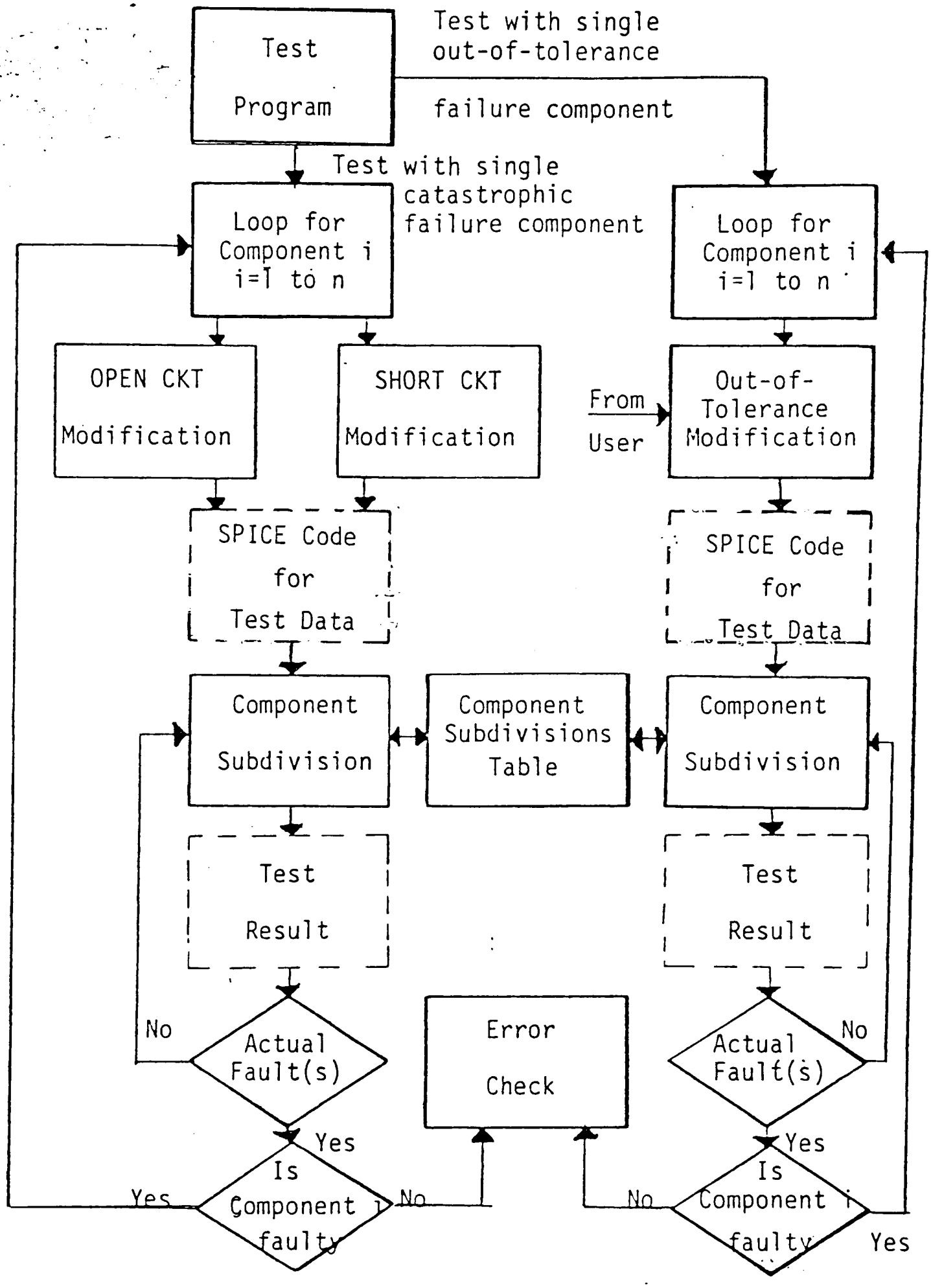


Figure 3.4. (Continued)

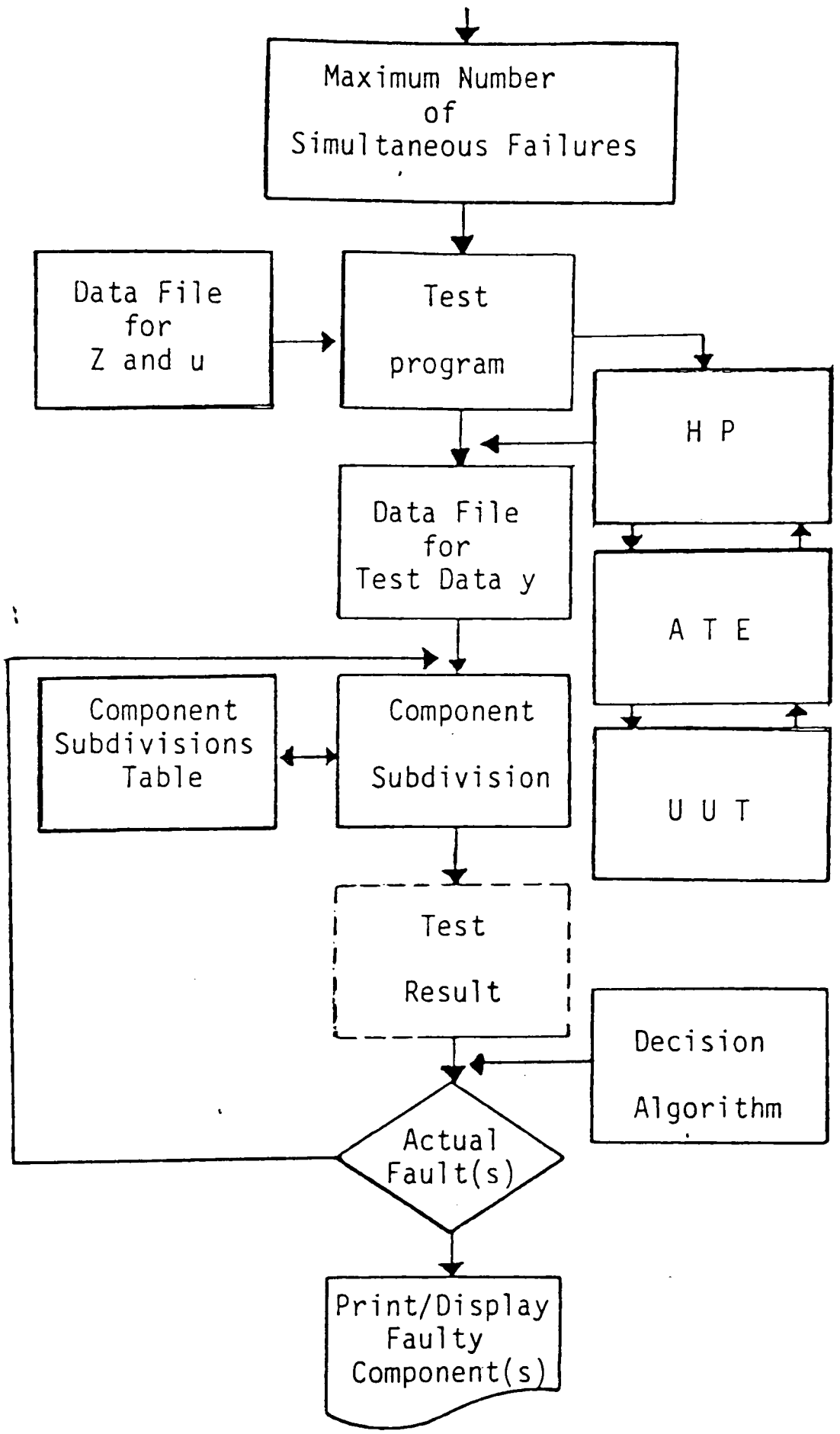


Figure 3.5. On-Line Component - Linear Case

(2.26). Similar ways to verify and validate the test program in the linear case are also employed here.

As illustrated in Figure 3.6, the input requirements in the test program generation are Circuit Description, Time Steps for Transient Analysis, and Accessible Test Terminals.

Circuit Description: In the SPICE program, each component in the circuit is specified by a component card that contains the component name, the circuit nodes to which the component is connected, and the values of the parameters that determine the electrical characteristics of the component. The first letter of the component name specifies the component type. Nodes must be nonnegative integers but need not be numbered sequentially. The ground node must be numbered zero. To describe the circuit analysis to SPICE, which is the same as the component description discussed in the linear case, the user is required to input the component type (any SPICE accepted component type) with a unique name, define the component value by any number field accepted by SPICE, and specify the nodes. For the semiconductor components, such as diodes, BJTs, JFETs, and MOSFETs, user needs to specify only the pertinent model parameter values. The model for the BJT is based on the integral charge model of Gummel and Poon; however, if Gummel-Poon parameters are not specified, the model reduces to the simpler Ebers-Moll model. In either case, charge storage effects, ohmic resistances, and a current-dependent output conductance may be included. The diode model can be used for either junction diodes or schottky barrier diodes. The JFET model is based on the FET model of Shichman and Hodges. The model for

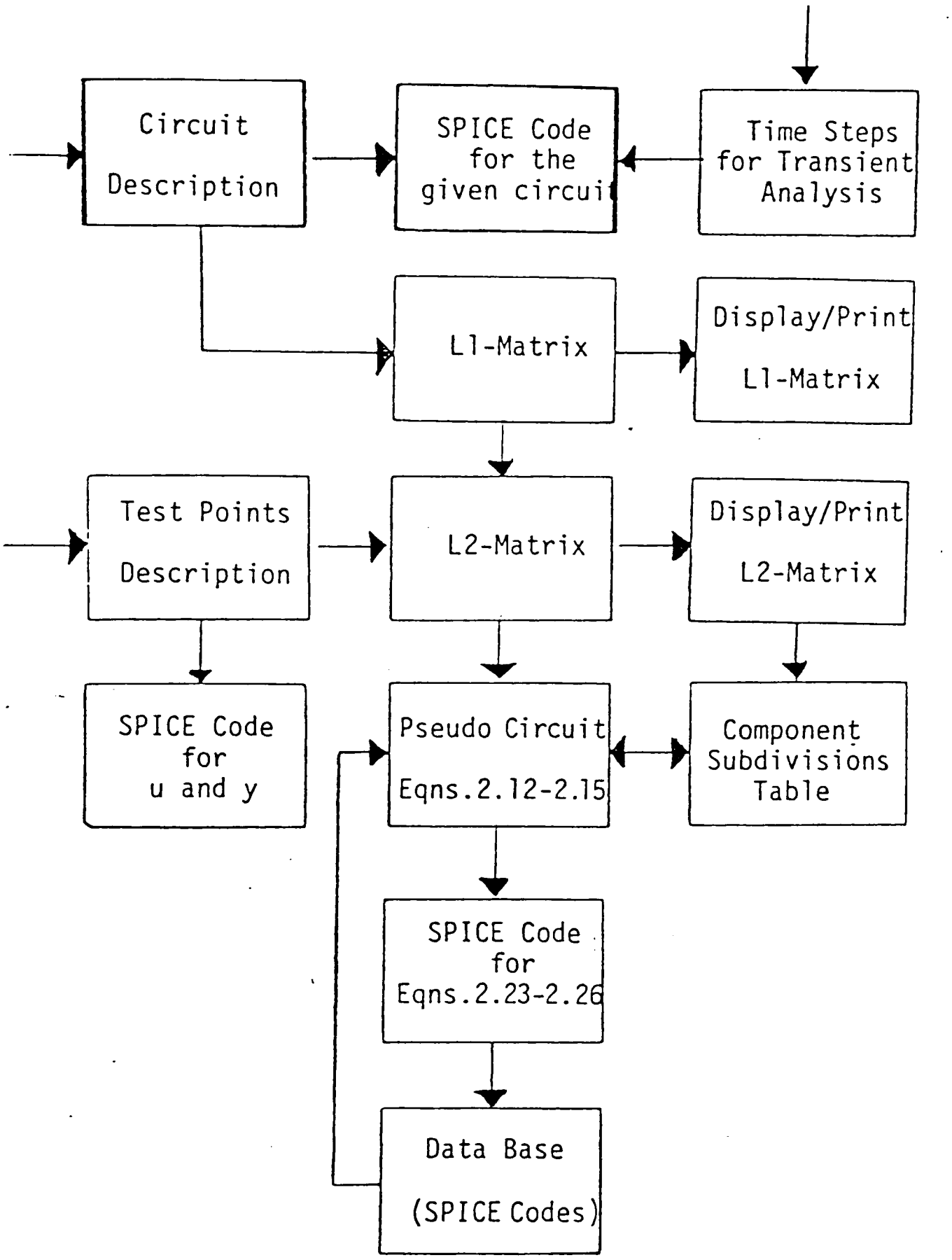


Figure 3.6. Test Program Generation - Nonlinear Case

MOSFET is based on the Frohman-Grove model; however, channel-length modulation, subthreshold condition, and some short-channel effects are included.[25] In the source description, any independent source in SPICE can be assigned a time-dependent value for transient analysis, and the source value can be a constant or independent source function (Pulse, Exponential, Sinusoidal, or Piecewise Linear). Therefore, the user may input the source value by a constant, or any above function.

Time Steps for Transient Analysis: The transient analysis portion of SPICE computes the transient output variables as a function of time over a user-specified time interval.

Accessible Test Terminals: When the accessible test terminals are specified, the L2-matrix, a table consisting of all possible component subdivisions, and a SPICE code for the test inputs u and test data y are generated. As discussed in the linear case, the connection matrix K of a pseudo circuit is created by computing the equations (2.12) through (2.15). For each subdivision, based on this generated K-matrix and equations (2.23) through (2.26), a SPICE code is generated and stored in a data base which will be used by the on-line component.

At the end of the program generation process, the following data files are generated: SPICE code for the given circuit, SPICE code for u and y, and data base (SPICE codes for the equations (2.23) through (2.26)).

Program Verification: In the linear case, three tests are used to verify the software's ability to locate the faulty component. Instead of using the test data y by computing equation (3.1) in the linear case, here, as indicated in Figure 3.7, we will modify the SPICE code for the

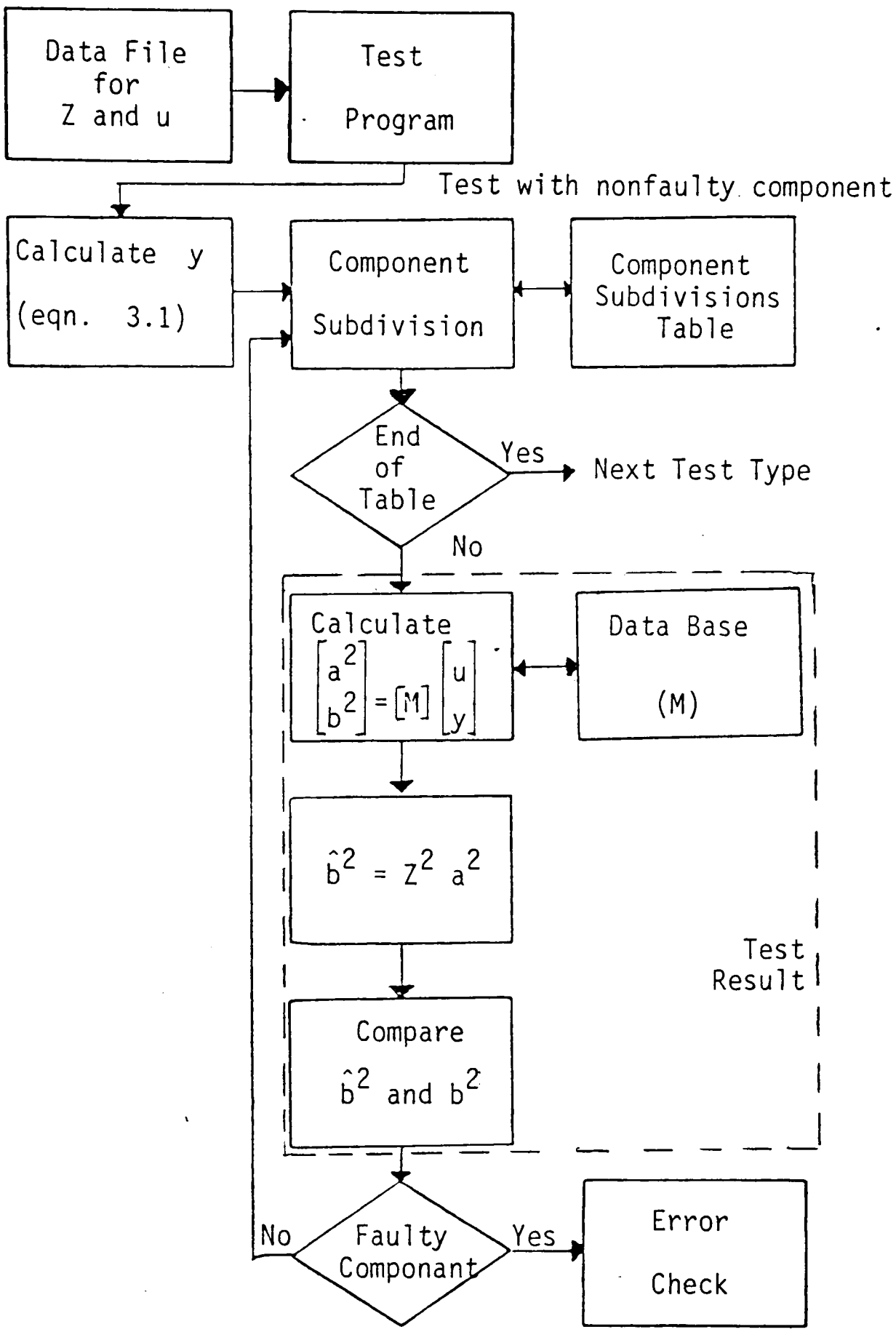


Figure 3.7. Program Verification - Nonlinear Case

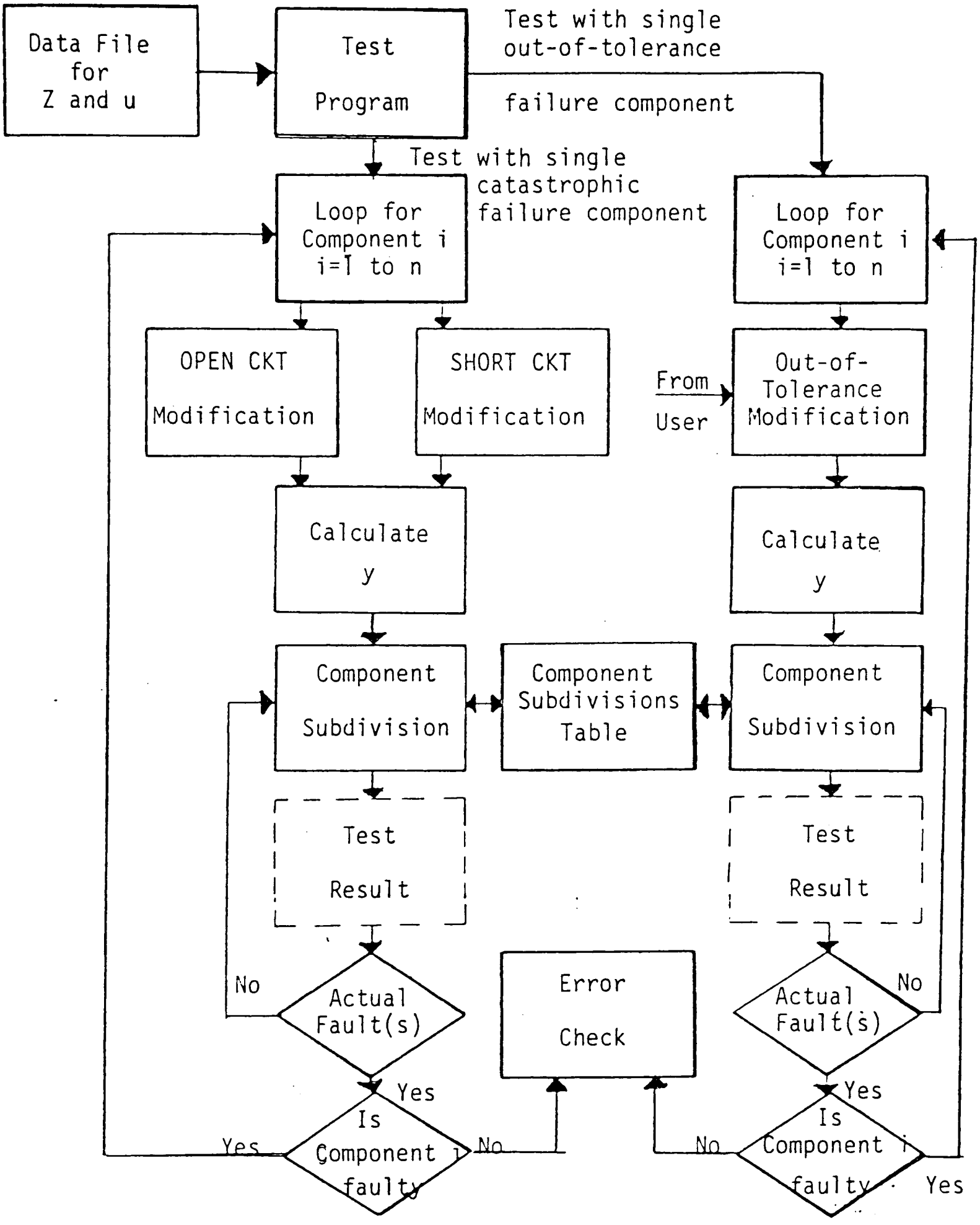


Figure 3.7. (Continued)

given circuit by assigning a simulated faulty value. Concatenating this code, the code for u and y, and the code for any subdivision, as an input file of the SPICE program, the output file stores the values of b^2 and $\hat{b}^2$. The test results are then obtained by comparing these two values.

Following the successful test program verification, the test program is validated as shown in Figure 3.8, by measurement of the actual UUT.

On-line Component: Test Stage

To implement the actual test on a UUT, as illustrated in Figure 3.9, the user inputs the maximum number of simultaneous failures allowed. The host computer loads the test program and sends instructions to command the HP 9825A controller conducting the test measurement. A data file is created to store the test data which is transferred from controller. A SPICE code for these test data is generated to replace the code for the given circuit when the SPICE program is run for the test results.

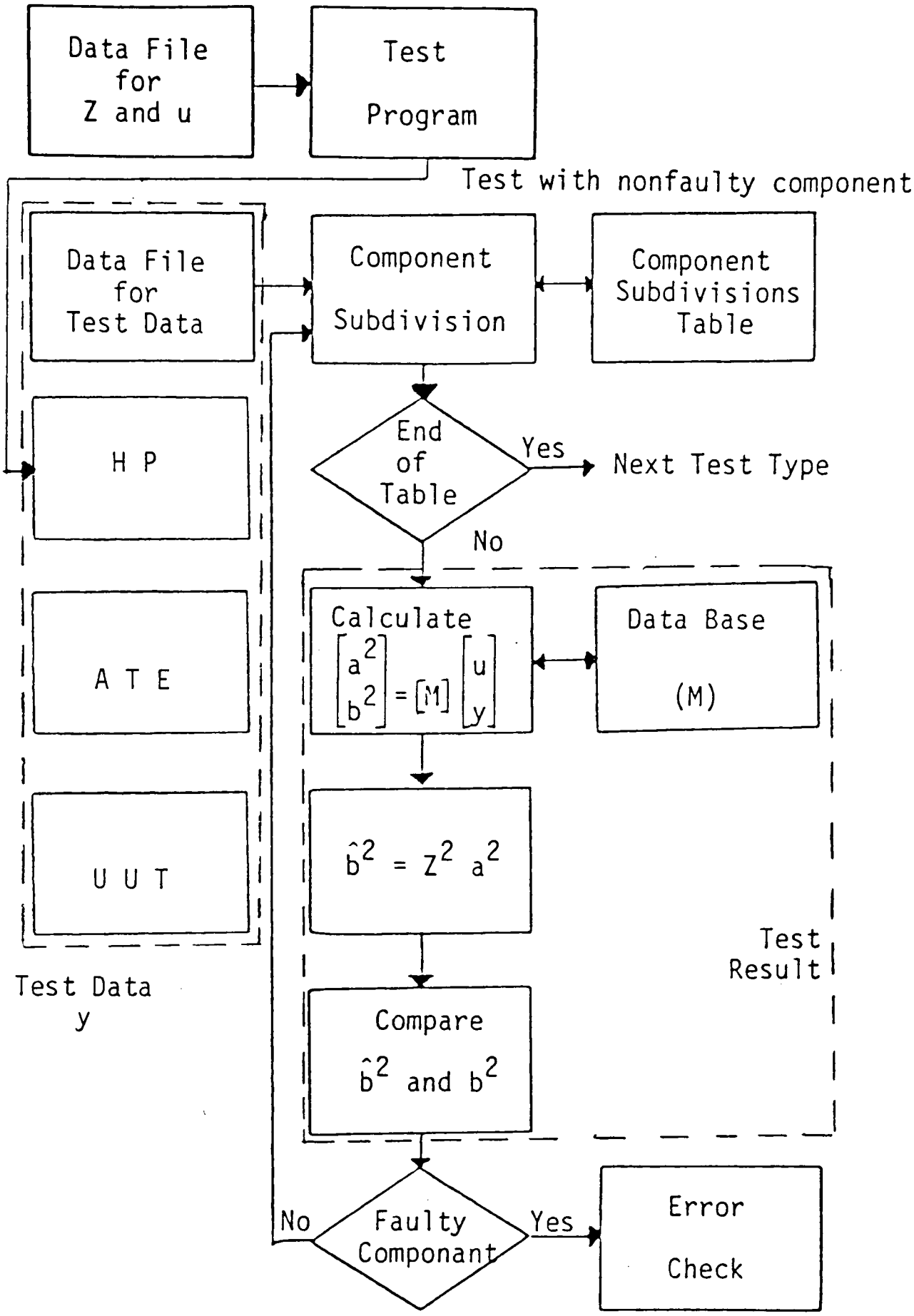


Figure 3.8. Program Validation - Nonlinear Case

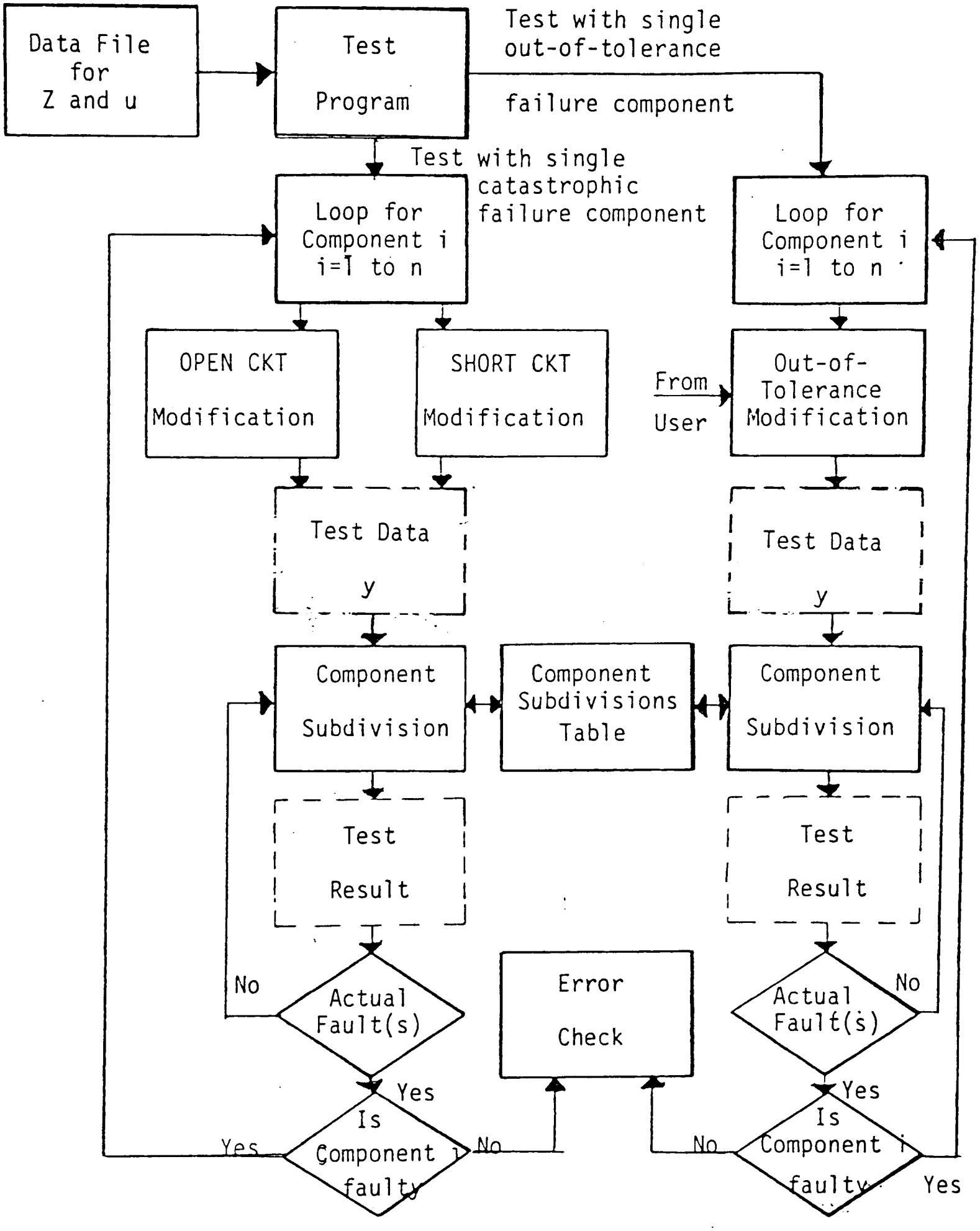


Figure 3.8. (Continued)

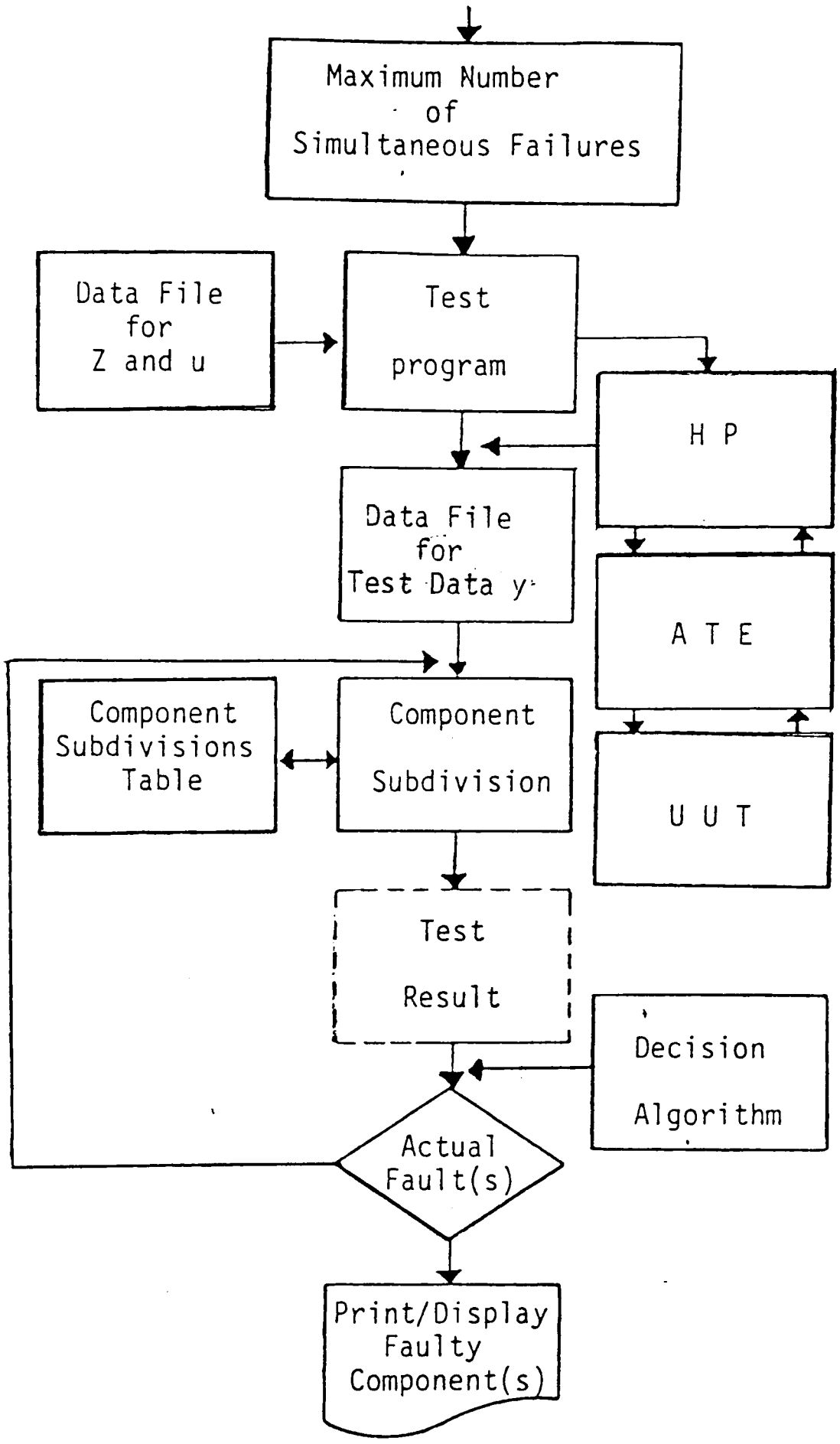


Figure 3.9. On-Line Component - Nonlinear Case

CHAPTER 4

ALGORITHMS

Along with the software developed in the previous chapter, it is desirable that the AATPG be run in a fully automatic mode, or interactively, in order to simplify the process of generating new test programs for both linear and nonlinear circuits. To this cause some supporting algorithms are discussed in the first section. In the second section, three decision algorithms are presented to make the decision for the choice of the component subdivision.

Supporting Algorithms

As described in the previous section, given a circuit, one can generate a data base by computing the matrix K together with some known parameters. The measured data and the stimuli will be used as the inputs to the test program while a test is conducted, and then a number of on-line simulations will be carried out. In this section, the following problems with algorithms are discussed:

(1) The generation of the connection matrix L with the users' inputs,

(2) The kind of subdivisions to be chosen, under the assumption that $[L_{21}^{2}]^{-L}$ exists,

(3) Based on the above subdivisions, how to set up the data base off-line so that the executing time of the on-line can be minimized, and

(4) How to decide a "good" component from the computed data.

L-Matrix: This commonly encountered geometric connection model is the linear graph used for electric networks and other bilateral components.[23] The linear graph, like the signal flow graph, is a directed graph composed of vertices and edges. Here each edge represents a single bilateral component or a part thereof. Associated with each edge is a pair of variables, V_i and I_i. For an electrical network V_i depicts the port voltage and I_i the port current. V_i is often generically termed an across variable and I_i a through variable. They are not a priori identified as the component input and output. For most components either may serve as the input with the other taken as the-output. The usual conservation laws constrain the variables V_i and I_i. In particular, the Kirchhoff voltage law constrains V_i, the Kirchhof currant law constrains I_i.

Let a graph have a specified tree containing r edges and the complementary co-tree containing (p-r) edges. (Here p is the number of edges in the graph.) The number r always equals "n-1" where n is the number of vertices in the graph.[7] Let B_f be the fundamental circuit matrix and S_f be the fundamental cut-set matrix, then

$$B_f = [X \mid I_{p-r}] \tag{4.1}$$

$$S_f = [I_p \mid D\ ] \tag{4.2}$$

where, X is a (p-r)xp matrix, I_{p-r} and I_p are identity matrices, and D is a px(p-r) matrix. It can be easily verified that

$$X = -D^t$$

therefore, the equation (4.1) can be written as

$$B_f = [-D^t \mid I_{p-r}]$$

Let $I = \mathrm{col}(I_t, I_c)$ and $V = \mathrm{col}(V_t, V_c)$ be the current and voltage vectors of the circuit respectively, where $I_t(V_t)$ and $I_c(V_c)$ are denoted as the currents (voltage) in the tree and co-tree edges, respectively. According to the Kirchhoff's current law and Kirchhoff's voltage law, we obtain the equations

$$0 = S_f\, I = I_t + D\, I_c$$

and (4.3)

$$0 = B_f\, V = -D^t V_t + V_c$$

therefore

$$\begin{bmatrix} I_t \\ V_c \end{bmatrix} = \begin{bmatrix} 0 & -D \\ D^t & 0 \end{bmatrix} \begin{bmatrix} V_t \\ I_c \end{bmatrix} \tag{4.4}$$

To obtain the matrix D, an incidence matrix A can be transformed into the form $\left[\begin{array}{c|c} I_n & D \\ \hline 0 & 0 \end{array}\right]$ by using a Gaussian elimination process.

In order to derive the L-matrix, equations (2.2) and (2.3), we will specify a tree in a linear graph modeling the topology of an electric network. Let $a = \mathrm{col}(I_t, V_c)$ and $b = \mathrm{col}(V_t, I_c)$ be our composite component input and output vectors. Inherently, tree edges correspond to components having the impedence models and co-tree edges correspond to components with admittance models. Usually, the choice of a tree depends on the choice of a model for the various components. Assuming that there is no intra-component coupling between components represented by tree edges and co-tree edges, the composite component model for the linear circuit becomes

$$\begin{bmatrix} V_t \\ I_c \end{bmatrix} = \left[\begin{array}{c|c} Z_t & 0 \\ \hline 0 & Y_c \end{array}\right] \begin{bmatrix} I_t \\ V_c \end{bmatrix} \tag{4.5}$$

where Z_t and Y_c are block diagonal matrices, the size of the diagonal block depends on the number of ports of the component. For one-port components: resistors, capacitors, and inductors; Z_t is the composite component impedance matrix for components identified with tree edges and Y_c is the composite component admittance matrix for components identified with co-tree edges. For two-port components such as the transistors, a two by two block is used to describe this component, the values are defined by the hybrid-pi parameters or appropriately transferred parameters, depending upon the elements of the transistor model employed.

With the choice of "a" and "b" the connection matrices follow directly from the equation (4.4) taking the form

$$L_{11} = \begin{bmatrix} 0 & -D \\ D^t & 0 \end{bmatrix} \tag{4.6}$$

However, for the network sources, the following assumptions will be made: each voltage source is connected inseries with an element which is not a source, and each current source is connected in parallel with an element which is not a source. Under these assumptions, the matrix L_{12} will be derived from the location of sources.

Suppose a voltage source E_s is located in the tree edge i,

$$V_{ti} = V'_{ti} + \omega E_s$$

where ω is defined as follows:

$$\omega = \begin{cases} 1 & \text{if the orientation of the source} \\ & \text{coincides with that of the edge,} \\ -1 & \text{otherwise.} \end{cases}$$

From the equations (4.3),(4.4), and (4.6) we obtain

$$V_{ck} = \sum_{j=1}^{n} L_{11}^{kj} V_{tj} \quad k = n+1, N+2,.., b$$

$$= \sum_{\substack{j=1 \\ j\neq i}}^{n} L_{11}^{kj} V_{tj} + L_{11}^{ki} V'_{ti} + \omega L_{11}^{ki} E_s \qquad (4.7)$$

Hence, the (k,s)-entry of the matrix L_{12} equals to ωL_{11}^{ki}.

Similarly, if a current source J_s is located in the co-tree edge i, the (k,s)-entry of L_{12} is also ωL_{11}^{ki}.

Suppose a voltage source E_s is located in the co-tree edge i,

$$V_{ci} = V'_{ci} + \omega E_s$$

and

$$V_{ck} = \sum_{j=1}^{n} L_{11}^{kj} V_{tj} \qquad k = n+1, n+2,..b$$

which implies

$$V'_{ck} = \sum_{j=1}^{n} L_{11}^{kj} V_{tj} - \omega E_s \qquad (4.8)$$

therefore, the (k,s)-entry of L_{12} equals $-\omega$. Similarly, the (k,s)-entry of L_{12} for a current source J_s located in the tree edge i is also $-\omega$.

Since the vector y is the system responses measured at the various test points, the matrices L_{21} and L_{22} in the equation (2.3), therefore, depend on the selection of the test points. The test points can be selected to measure the current or voltage of any edge or the voltage across any two nodes. In the former case, i.e., the current flow through k-th co-tree edge or the voltage across the k-th tree edge, the response y_j is then an element of vector a. Therefore, the jth rows of the

matrices L_{21} and L_{22} are just the k-th rows of L_{11} and L_{12}, respectively. In the latter case the responses y_j will be written in terms of vectors b and y, and the coefficients form the jth rows of L_{11} and L_{22}, respectively.

ALGORITHM I: (L-Matrix)

```
Step 1.  Generate the incidence matrix A.
Step 2.  Obtain matrix D from matrix A by Gaussian
         elimination process.
Step 3.  Generate L11-matrix from the equation (4.5).
Step 4.  Let e be the number of voltage sources and
           j be the number of current sources
         L12(*,*) = 0.
Step 5.  IF e = 0 THEN go to step 6 (no voltage source)
           DO i=1 TO e
             IF the voltage source is in the tree edge
             (assume they are k-th edge and jth  source)
           THEN
             DO m=p-r TO p
               IF L11(m,k)≠0 THEN L12(m,j)=ω*L11(m,k)
                End of loop
           ELSE  (in the co-tree edge)
               L12(k,j)= -ω
           End If
           End of Loop
Step 6.  IF j = o THEN go to step 7 (no current source)
           DO i=1 TO j
             IF the current source is in the tree edge
             THEN
                DO m=1 TO r
                  IF L11(m,k)≠0 THEN L12(m,j)=ω*L11(m,k)
                  End of loop
             ELSE
                  L12(k,j)= -ω
             End If
             End of Loop
Step 7.  (Generate matrices L21 and L22)
         (assume the number of test points is m)
         DO i=1 TO m
         (assume yi is the i-th system response)
           IF yi is selected from the entry of "a"
           (let it be the k-th entry of vector "a")
           THEN
             L21(i,*)=L11(k,*) and L22(i,*)=L12(k,*)
```

```
ELSE  (y_j is specified by user)
(let P(t) be the specified value corresponding to
the nonzero column t)
  L2 (i,t)=P(t) for all nonzero t
End If
End of Loop
```

Component Subdivisions Table: The component subdivisions table is derived from the allowable component subdivisions which satisfy the assumption that $[L_{21}^2]^{-L}$ exists. Consider an n components circuit with m test points. The L_{21} matrix is then a m by n matrix. For the software implementation, m x m matrices are constructed by selecting all possible combinations of m columns from the n columns in the matrix L_{21}, and checking whether or not the matrices are invertible. The subdivisions are recorded into the following table if and only if the matrices are invertible.

Subdivision number	Group 1 Component	Group 2 Component		Group m Component
(1)	*	*	...	*
(2)	*	*	...	*
:	:	:	...	:
:	:	:	...	:
(C)	*	*	...	*

ALGORITHM II: (Component Subdivisions Table)

```
Step 1.  (Matrix L21 is an m by b matrix)
         pick any m columns of matrix L21 to form a matrix
Step 2.  IF matrix is invertible
         THEN
           record the number of these m's column.
         ELSE
           pick next m columns to form a matrix.
Step 3.  Repeat 2 until all combinations have been chosen.
```

Data Base: Given any component subdivision, a pseudo circuit with connection matrix K is created by computing the equations (2.12) through

(2.15). The data base for the linear case is generated by computing matrix M, equation (2.16), and stored in the data files.

In the nonlinear case, the data base is SPICE codes for equations (2.23) through (2.26). The SPICE code is generated as follows:

Consider the equations (2.23) through (2.26),

$$\dot{x}^1 = f^1 (x^1, a^1)$$
$$b^1 = g^1 (x^1, a^1) \quad ; \quad x^1(0) = 0 \tag{2.23}$$

$$a^1 = K_{11}\, b^1 + K_{12}\, u^p \tag{2.24}$$

$$y^p = K_{21}\, b^1 + K_{22}\, u^p \tag{2.25}$$

$$\dot{x}^2 = f^2 (x^2, a^2)$$
$$\hat{b}^2 = g^2 (x^2, a^2) \quad ; \quad x^2(0) = 0 \tag{2.26}$$

where u^p = col (u,y) and y^p = col(a^2, b^2). Our goal is to compute the values of b^2 and $\hat{b}^2$ from the above equations. Solving the values b^1 from equations (2.23) and (2.24), plugging b^1 into equation (2.25) for y^p, i.e., a^2 and b^2, and substituting the values a^2 into (2.26) to solve $\hat{b}^2$.

Mathematically, the equation (2.24) shows that the element a_i, i-th element of a^1, is the sum of the products of the i-th row of K_{11} and b^1, and the products of the i-th row of K_{12} and u^p. Physically, suppose that a_i is a one-port component, if the element a_i is a voltage measurement, the b_i is then a current measurement. Therefore, the voltage a_i is the sum of the measured voltages of b^1 and u^p where the corresponding terms of the i-th rows of K_{11} and K_{12} are not zero. For example, if

$$b^1 = \text{col}\,[V_{b1},\ I_{b2},\ V_{b3},\ I_{b4},\ V_{b5}]$$

$$u^p = \text{col}\,[V_{u1},\ V_{u2},\ I_{u3},\ V_{u4}],$$

i-th row of K_{11} = [1, 0, -1, 0, 0] and

i-th row of K_{12} = [1, 0, 0, -1]

then

$$V_{ai} = V_{b1} - V_{b3} + V_{u1} - V_{u4} \tag{4.9}$$

i.e., the voltage measurement at component a_i is the sum of the measured voltages, V_{b1}, $-V_{b3}$, V_{u1}, and $-V_{u4}$. In order to solve the equation (4.9) with SPICE program, a cirduit type of description is used to generate the SPICE code. Here, the voltage controlled sources connected in series are used to indicate the sum of measured voltages,

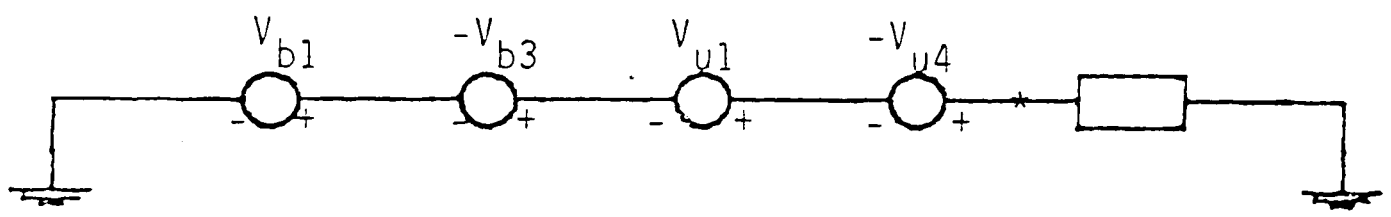


Figure 4.1. Controlled Sources

and the voltage measured at node * is then the value of V_{ai}. Obviously, once the values a^1 are known, the component equation (2.23) will give the values b^1. In circuit specification, the above box will be filled by the component. Given the values a^1, evidently the values b^1 may be

derived from their characteristics. For example, if the component is a resistor, connecting the resistor in series with the above sources as shown in Figure 4.2. Since a_i is a voltage measurement, the current Ia_i is the current that flows through the resistor.

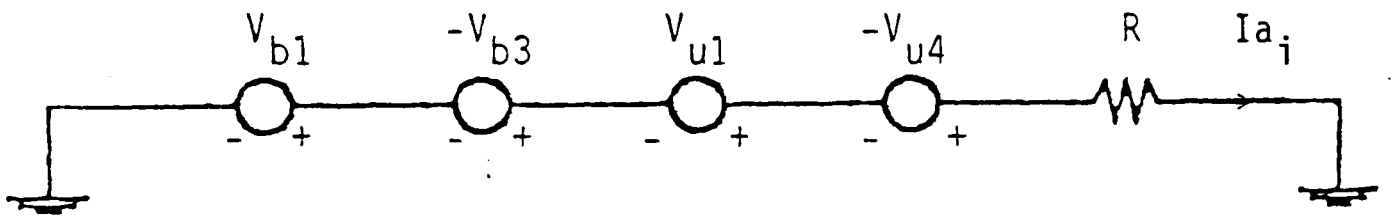


Figure 4.2. Controlled Sources with Component

Similarly, the current controlled sources connected in parallel are used to describe the current measurement case.

After the SPICE code for equation (2.24) is generated properly, consider the equation (2.25) with the partitioned matrices,

$$a^2 = K_{21}^1 b^1 + K_{22}^1 u^p \tag{4.10}$$

$$b^2 = K_{21}^2 b^1 + K_{22}^2 u^p \tag{4.11}$$

The SPICE code for equation (4.11) is generated the same as that for equation (2.24), except that the box in Figure 4.1 is replaced by a zero-valued voltage source if the element of b^2 is a current measurement, or by a resistor with resistance 1 for a voltage measurement. Similarly, the SPICE code for equation (4.11) can be generated in the same way,

however, what we are interested in is the values $\hat{b}^2$. Therefore, the box in Figure 4.1 is filled by the component with equation (2.26) to compute the value $\hat{b}^2$.

ALGORITHM III: (Data Base)

Step 1. (Linear Case)
(Let C be the number of all possible subdivisions)
DO i=1 TO C
Compute K-matrix (Equations (2.12) through (2.15)).
Compute M-matrix (Equation (2.16)).
Store M to a data file named TE00**.DT, **=i.
End of Loop.

Step 1. (Nonlinear Case)
Let N be the number of components
M be the number of test points,
NM=N-M
Step 2. (For equations (2.23) and (2.24))
DO i=1 TO NM
IF a_i^1 is a voltage measurement
THEN
Connecting the controlled sources, which have nonzero elements in K_{11} and K_{12}, in series.
ELSE
Connecting the controlled sources, which have nonzero elements in K_{11} and K_{12} in parallel.
Connecting a zero-valued voltage source in series.
(Remark: In SPICE, a zero-valued voltage source is used to measure the current)
End if
Connecting the group "1" component i in series.
End of Loop
Step 3. (For equations (4.10) and (2.26))
DO i=1 to M
If a_i^2 is a voltage measurement
THEN
Connecting the controlled sources, which have nonzero elements in K_{11} and K_{12}, in series.

```
        ELSE
          Connecting the controlled sources, which
            have nonzero elements in K11 and K12,
            in parallel.
          Connecting a zero-valued voltage source in
            series.
          End if
        Connecting the group "2" component i in the series.
        End of Loop
Step 4. (For equation (4.11))
        DO i=1 TO M
             2
        IF b_i is a voltage measurement
        THEN
          Connecting the controlled sources, which
            have nonzero elements in K11  and K12,
            in the series.
          Connecting a resistor with resistance 1.
        ELSE
          Connecting the controlled sources, which
            have nonzero elements in K11 and K12,
            in parallel.
          Connecting a zero-valued voltage source in
            series
          End If
        End of Loop
```

Test Results: In the linear case, only matrix-vector multiplications, equations (2.17) through (2.19), are required to evaluate b^2 and $\hat{b}^2$ while the SPICE program is executed for the nonlinear case. Use of this computed data to determine test outcome (either "good" or "bad" for each group "2" component) may be obtained by comparing b^2 and $\hat{b}^2$. If b_i^2 is equal to $\hat{b}_i^2$, then we say that the test outcome for the group "2" component i is "good"; otherwise, the component is "bad". In a more realistic environment, instead of requiring that b_i^2 and $\hat{b}_i^2$ be equal one may say that a component is "good" if b_i^2 is sufficiently close to $\hat{b}_i^2$ in some reasonable sense. In this way one may compensate for numerical errors and tolerance.[10] Moreover, b_i^2 and $\hat{b}_i^2$ are

not necessarily scalars, they may be vectors, depending upon the component type with which one deals. For instance, a two-port component may require a two-tuple vector to represent its input/output characteristics.

ALGORITHM IV: (Test Results)

```
Step 1.  (Linear Case)
         Let Z2 be the transfer function matrix for group "2"
           UY=col(u,y)
           M be the matrix for the data base.
Step 2.  Choose a component subdivision.
Step 3.  Retrieve the matrix M for this subdivision from
         the data base.
Step 4.  (Compute a^2 and b^2)
         DO i=1 TO m
         DO j=1 TO s (s is the dimension of UY)
           A2(i) = A2(i) + M(i,j)* UY(j)
           B2(i) = B2(i) + M(m+i,j)*UY(j)
           End of Loop j
         End of Loop i
Step 5.  (Compute b̂^2)
         DO i=1 TO m
           B2W(i) = A2(i)* Z2(i,i)
           End of Loop
Step 6.  (Compare b^2 with b̂^2)
         (Let ε be the tolerance)
         DO i=1 TO m
           VALUE = |B2W(i)-B2(i)|/|B2(i)|
           IF VALUE < = ε THEN
             RESULT(i) = 0
           ELSE
             RESULT(i) = 1
             End If
           End of Loop

Step 1.  (Nonlinear Case)
         Let SOURCE.DT be a data file which contains the
           SPICE code for test inputs and test data.
         Let SPICE.DT be a data file which contains either
           (1) Modified SPICE code for the given circuit, or
           (2) SPICE code for test data.
Step 2.  Retrieve the data base, assume i-th subdivision is
         used.
Step 3.  Concatenating the file SPICE.DT, SOURCE.DT and TE00*.DT
         (where *=i) as an input file of SPICE program.
Step 4.  The output values b^2 and b̂^2 are stored in a data file.
```

Once the test outcomes have been obtained, the algorithm reduces to a combinatorial "self-testing problem" in which one locates the actual failure. In other words, one may complete the test algorithm by implementing an appropriate decision algorithm.

Decision Algorithms

Three decision algorithms, with their software implementation are presented:

(1) Exact Algorithm,

(2) Heuristic Algorithm, and

(3) Boolean Expression Algorithm.

The first algorithm is employed to locate single failures, while the remaining two algorithms are used to identify multiple failures.

Exact Algorithm: In the single failure case, we assume that at most one component is faulty. As discussed in 42, we summarize all possible test results obtained from a given step of the algorithm, together with the conclusions as follows:

Test Result (1 2 3 . . . m)	Conclusions
0 0 0 . . . 0	all group "2" components are good
1 0 0 . . . 0	all group "2" except 1 are good
1 1 0 . . . 0	all group "2" components are good
: : : : : : :	: :
: : : : : : :	: :
1 1 1 . . 1 0	: :
1 1 1 . . 1 1	all group "2" components are good

In the first case we conclude that all group "2" components are good. If a group two component were actually faulty then our test results are incorrect, which would only happen if one of the group "1" components

was faulty. This would imply that the system has two faulty components, contradicting our assumption that, at most, one component is faulty. In case two, the same argument we used above will guarantee that the components which test good, say 2 through m, are good, and we have no information about x. It may be faulty or, alternatively, the test results may be due to a faulty group "1" component. In the remaining cases we have the same conclusion as in the first case. Since, under our assumption of a single failure, it is impossible for two or more group "2" components to be faulty, these test results imply that at least one of the group "1" components is bad. However, since we have assumed that there is, at most, one faulty component, and the group "1" component is the only faulty component, then the group "2" components are all good.

Consistent with the above arguments, at each step of the test algorithm, either all, or all but one, of the group "2" components are found to be good. If we choose our subdivision so that good components are included in group "1" the test results obtained at that step will be reliable, thereby, allowing us to accurately determine the faulty components in group "2."

From the above algorithm, one may be interested in the problem of how many steps are required to locate the faulty component(s): For single failure case, in each step of the algorithm, one concludes that all, or all but one, group "2" components are good. Thus, one may select a minimum collection of subdivisions which covers all components, i.e.,

Let N = {1,2,..,n} be a set of n components,

$S = \{\#1, \#2, ..\#C\}$ be a collection of all possible subdivisions,

where C is the number of subdivisions,

then

$$B = \min \{T \subset S \mid N \subset T\} \tag{4.12}$$

is the minimum subcollection of S which covers N.

The subcollection B is not unique. If $B = \{B_1, B_2, .., B_t\}$, where B_i is in S, then, t is the minimum steps needed to locate the faulty component.

Recall that all, or all but one, components are known to be good at each step of the algorithm. After completing the test simulations with the above indicated subdivisions, a component which is the only one with test results "1" in a subdivision, may be faulty. If there exists only one such component, eventually, the component is located as faulty. However, if more than than one such components exist more steps are needed to identify the faulty components.

Let $P = \{P_1, P_2, .., P_s\} \subset B$, where P_i is the subdivision with the pattern that contains all "0" but one "1",

$R = \{R_1, R_2, .., R_s\}$, $R_i \in P_i$, where R_i is the component with test results "1", all R_i may not be distinct, and

Let $R' = \{r_1, r_2, .., r_{s'}\} = R$, where all r_i are distinct.

In the next steps, a subdivision with more than two components of R' is selected to simulate the test results, and the components which are known to be good will be excluded from R'. Therefore, the set R' is getting smaller. Repeating the above process until no such subdivision can be chosen, the algorithm is then terminated. Here, the number of times of the above processes plus t, the minimum steps needed in

equation (4.12), is the maximum steps needed to locate the faulty component, and the components remaining in the set R' is the ambiguity set.

ALGORITHM V: (Exact Algorithm for single failure case)

Step 1. (Off-line job)
Select the minimum subcollection of the subdivisions which covers all components. $B = \{B_1, B_2, .., B_t\}$.

Step 2. (On-line)
Simulate each subdivision B_i, $i = 1,2,..t$.

Step 3. Let $P = \{P_1, P_2, .., P_s\} \subset B$, where P_i are the subdivisions with pattern (all "0" but one "1").

Step 4. Let $R = \{R_1, R_2, .., R_s\}$, $R_i \in P_i$, are the components with test result "1"

Step 5. IF s'=1 THen STOP [the component r_1 is faulty].

Step 6. (More than one subdivisions with the above pattern)
Find a subdivision that contains more than one component in R'.

Step 7. IF no such subdivision THEN GOTO Step 8.
ELSE
Simulate the test result
R' <— R' excludes the simulated good components.
GOTO Step 6.

Step 8. R' is the ambiguity set, and the components in R' are all possible faulty components.

From the above algorithm, the step 1, $B = \{B_1, B_2, .., B_t\}$, can be derived off-line, and each B_i subdivision is simulated independently. Therefore, they can be computed by multiple processors, i.e., the parallel processors can be used to reduce the executing time. Thus the number t is the maximum number of processors needed.

To handle the multiple failure, following Liu, the problem can be greatly simplified if an "analog heuristic" is adopted. The effect will be that two independent analog failures will never cancel.[20] Needless to say, this is an inherently analog heuristic, since two binary failures have a fifty-fifty chance of canceling one another. In the analog case, however, two independent failures are highly unlikely to

cancel one another (as long as one works with reasonably small tolerances). Based on the above argument, we may assume that a component is definitely good if the test result shows "0." Therefore, if a t-diagnosable system is assumed, that is, the number of faulty components does not exceed t, all possible test results with the conclusions are shown as follows:

Test Result	Conclusions
(1 2 3 .. t . . .. m)	
0 0 0 .. 0 0	all group "2" components are good
1 0 0 .. 0 : : :: 0	components 2 thru m are good
: : : :: : : : :: :	:
: : : :: : : : :: :	:
1 1 1 .. 1 0 0 .. 0	components t+1 thru m are good
1 1 1 .. 1 1 0 .. 0	components t+2 thru m are good, and at least one faulty component in group "1"
: : : :: : : : :: :	:
1 1 1 1	at least one faulty component in group "1"

In the first t+1 cases the results show that the component with test result "0" will be good. In the (t+2)-th case we claim that the components (t+2) through m are good and at least one faulty component in group "1." If none of the group "1" components were actually faulty then the test results obtained at this step are reliable. The system would then have t+1 faulty components contradicting our assumption to the effect that, at most, t components are faulty. The remaining cases are the same as above. Unfortunately, the above argument gives the information that at least one of the group "1" components is faulty, but the identification of the faulty component in group "1" is still not available.

Heuristic Algorithm: Based upon the "Analog heuristic" the Heuristic algorithm was presented as a multifailure decision algorithm.[41,42] In practice, the Heuristic algorithm is used with a coupling table.

The coupling table is designed to detect whether or not a faulty group "1" component will effect the test results on a group "2" component. Two components are called "coupled" if they are functions of each other. In order to set up this coupling table, we first look for the relationship between the group "2" components and the group "1" components.

Consider the equations (2.10) and (2.11) with the constant matrix K. One can rewrite the equation (2.10) as follows:

$$\begin{bmatrix} a_1^1 \\ a_2^1 \\ \vdots \\ a_{n-m}^1 \end{bmatrix} = [K_{11}] \begin{bmatrix} b_1^1 \\ b_2^1 \\ \vdots \\ b_{n-m}^1 \end{bmatrix} + \text{constant term}$$

and then

$$a_i^1 = \sum_{k=1}^{n-m} K_{11}^{ik} b_k^1 + \text{constant term}$$

If K_{11}^{ik} is not zero, then the group "1" component k effects the group "1" component i. Thus the (n-m) equations give the relationships between the group "1" components. The equation (2.11) can be written as

$$a_i^2 = \sum_{k=1}^{n-m} K_{21}^{ik} b_k^1 + \text{constant term}$$

$$b_i^2 = \sum_{k=1}^{n-m} K_{21}^{tk} b_k^1 + \text{constant term, } t = m + i.$$

If K_{21}^{ik} or K_{21}^{tk} is not zero, then the group "1" component k effects the group "2" component i, and the components i and k are therefore coupled.

ALGORITHM VI: (Coupling Table)

Step 1: (Generate the matrix A for the relationship between the group "1" components)
(By a transitive closure algorithm [1]

$$A = I \oplus E^1 \oplus E^2 \oplus \ldots \oplus E^k$$

where E^k gives an explicit accounting of all the vertices jointed by paths of length k, $\oplus$ is the Boolean "or" expression, and A_{ij} is defined as

$$A_{ij} = 1 \quad \text{if i effects j}$$
$$= 0 \quad \text{otherwise} \qquad)$$

Let E be the matrix K_{11} in the equation (2.12). (K_{11} is a m by m matrix and K_{21} is a 2m by m matrix)
Initialize A(*,*)=0, F(i,i)=1 for i=1,2,..m.
DO i=1 TO m
  F <— F * E (i.e., $F=E^i$)
  IF F(j,k)=1 THEN A(j,k)=1 for all j and k
  End of loop

Step 2: (Generate the matrix B for the relationship between the group "2" components and the group "1" components)
(Let B_1 and B_2 be two Boolean matrices which are defined as follows:

$$B_1^{ij} = 0 \quad \text{if} \quad K_{21}^{ij} = 0; \qquad i=1,2,..m. \quad j=1,2,..,n-m.$$
$$= 1 \quad \text{if} \quad K_{21}^{ij} \neq 0$$

and

$$B_2^{ij} = 0 \quad \text{if} \quad K_{21}^{ij} = 0; \qquad i=m+1,m+2,..2m.$$
$$= 1 \quad \text{if} \quad K_{21}^{ij} \neq 0; \qquad j=1,2,..n-m.$$

therefore,

$$B = B_1 \oplus B_2 \qquad)$$

(define the matrices B_1 and B_2)
DO i=1 TO m
  DO j=1 TO n-m
    IF $K_{21}(i,j) = 0$ THEN $B_1(i,j) = 0$

```
                ELSE B1(i,j) = 1
                IF K21(i+m,j)=0 THEN B2(i,j)=0
                ELSE B2(i,j)=1
                End of Loop j
            End of Loop i

        (define B = B1 ⊕ B2)
        DO i=1  TO m
            DO j=1 TO n-m
                IF B1(i,j)=1 OR B2(i,j)=1 THEN B(i,j)=1
                ELSE B(i,j)=0
                End of Loop j and i
```

Step 3. (Generate a matrix C — for coupling table)
(Let C = B * A, where * is the Boolean multiplication, and then matrix C is the desired coupling table)

Interestingly, our heuristic can be carried a step further than indicated above since, under our heuristic, a bad group "1" component would normally yield erroneous test results. An exception would, however, occur if some of the group "1" components are totally decoupled from some of the group "2" components. Consider the coupling table and the simulation results are shown in the following table:

Table 4.1. Coupling Table with the Test Results

"2"	"1"	#4	#5	#7	#8	#9	#10
1	#1	1	1	1	1	1	1
0	#2	0	1	1	1	1	1
1	#3	1	1	1	1	1	1
0	#6	0	1	1	1	1	1

The test results show that #2 and #6 are good in the test. Our heuristic implies that the components #5, #7, #8, #9 and #10, which are coupled by components #2 and #6, are also good. There are no information from #1 and #3. Therefore, all components are good except #1, #3

and #4 are unknown. Moreover, in the single failure case, the test results show that the components #1 and #3 are also good, and a conclusion will be immediately obtained in this simulation result — the component #4 is faulty.

ALGORITHM VII: (Heuristic Algorithm)

```
Step 1.  Input t, the maximum number of simultaneous failures
         FLAG(j)= .false., j=1,2,..,n.
Step 2.  Choose a component subdivision
Step 3.  Call subroutines to derive the simulation results.
Step 4.  Let GOOD(j) be the good components, j=1,2,..,s.
Step 5.  Retrieve the coupling table TABLE(m,n-m).
         DO j=1 TO s
            Let x be the row of the component GOOD(j)
            DO k=1 TO n-m
              IF TABLE(x,k)=1 THEN
                record this component as good
              End of Loop k
            End of Loop j
Step 6.  IF t > 1 GOTO Step 9.
         (For single failure case)
            All or all but one components in group "2" are known
               to be good.
Step 7.  FLAG(GOOD(j))=.true., j=1,2,..,s
Step 8.  To have more simulations ? if so, go to step 2.
         Otherwise, end of this algorithm.
```

Boolean Algorithm: In this algorithm, a Boolean expression is derived from each step of the test algorithm, which includes all possible fault patterns associated with the test data. The actual fault(s) can be located by multiplying the Boolean expressions associated with several steps of the algorithm or equivalently comparing the fault patterns obtained from each test step and excluding the impossible fault patterns.

Consider the case where group "1" contains five components, a, b, c, d and e, group "2" contains three components, x, y and z. Suppose that the test results is indicated as follows:

"2" \ "1"	a b c d e
0 x	
1 y	
0 z	

The group "1" components are assumed to be all good so that the test is reliable; that is, the components a, b, c, d and e are good. The test results show that x and z are good and y is possibly faulty (where the test result "0" means "good" and "1" means "fault"). Therefore, the Boolean form for this possible test result can be expressed as follows:

$$\bar{a}\ \bar{b}\ \bar{c}\ \bar{d}\ \bar{e}\ \bar{x}\ y\ \bar{z}$$

Here, the letter a indicates the component "a" is bad, and $\bar{a}$ means the component "a" is good. However, the test may be unreliable if one of the group "1" components is bad. In that case the remaining components could be either good or bad, and those components are thus defined as "don't care" (with the notation "ϕ"). If the group "1" component a is assumed to be bad, for instance, the possible pattern is that a is bad and the remaining components are "don't care," and the expression for this pattern is $a(= a\phi\phi\phi\phi\phi\phi\phi)$. Therefore, the completed results for this test will be expressed by a Boolean form as follows:

$$T_1 = a + b + c + d + e + \bar{a}\ \bar{b}\ \bar{c}\ \bar{d}\ \bar{e}\ \bar{x}\ y\ \bar{z} \qquad (4.13)$$

Our goal is to combine the information derived from various test results so that the actual fault(s) can be fully identified.

Consider the test results for another subdivision in this example, where components a, b, c, x and z are in group "1" and components d, e and y are in group "2." The test result is assumed to be

"2" \	"1"	a	b	c	x	z
1	d					
1	e					
0	y					

The Boolean expression for this test is

$$T_2 = a + b + c + x + z + \bar{a}\,\bar{b}\,\bar{c}\,d\,e\,\bar{x}\,\bar{y}\,\bar{z} \qquad (4.14)$$

If we combine these two results, the intersection of the two expressions, T_1 and T_2, is derived by directly applying the multiplication rule of two logical functions, and thus

$$\begin{aligned} T &= T_1 * T_2 \\ &= (a + b + c + d + e + \bar{a}\,\bar{b}\,\bar{c}\,\bar{d}\,\bar{e}\,\bar{x}\,y\,\bar{z}) * \\ &\qquad (a + b + c + x + z + \bar{a}\,\bar{b}\,\bar{c}\,d\,e\,\bar{x}\,\bar{y}\,\bar{z}) \\ &= a + b + c + dx + dz + ex + ez + \overline{abcdexyz} \end{aligned}$$

The number of possible faulty patterns is really reduced in this resultant expression (there were 249 faulty patterns for each, and now it becomes 241 patterns). Each test step generates a temendous number of possible fault patterns. This is almost useless unless we can combine the fault patterns from the various test steps and eliminate patterns which do not appear in each step and/or pattern.

The interesting problems are: how to implement this symbolic Boolean expression to a software program? And how to accelerate the speed of convergence?

A tabulated expression may be readily implemented in software. One can tabulate all possible fault patterns corresponding to the test results in (4.13) as follows (where ϕ denotes the "don't care" terms):

a	b	c	d	e	x	y	z
1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	ϕ	1	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	ϕ	ϕ	1	ϕ	ϕ	ϕ
0	0	0	0	0	0	1	0

Recall for the expression of equation (4.13), the first term a means the component "a" is bad and the remaining components are "don't care." Therefore, as a tabulated expression, the bad component "a" is denoted by "1," and the remaining components are denoted by "ϕ." The following four terms have similar expressions, as above. For the last term, in equation (4.13), the group "1" components are all good, i.e., the components a, b, c, d, and e are good, denoted by "0," and the test results for x, y and z will be copied under them.

ALGORITHM VIII. (Tabulated Expression)

```
Step 0.  Let GR1(i) be the group "1" components, i=1,2,..,n-m.
             GR2(i) be the group "2" components, i=1,2,..,m.
             RESULT(i) be the test results, i=1,2,..,m.
             TABLE(n,n-m+1) be the tabulated expression.
Step 1.  (If one of group "1" components is faulty)
         DO i=1 TO n-m
             Initialize TABLE(m,i)="φ", j=1,2,..,n.
             TABLE(GR1(i),i)="1"
             End of Loop
Step 2.  (Case of the group "1" components are all good)
         DO i=1 TO n-m
             TABLE(i,n-m+1)="0"
             End of Loop
Step 3.  (Copy the test results)
         DO i=1 TO m
             TABLE(n-m+i,n-m+1)=RESULT(i)
             End of Loop
```

Similarly, following this algorithm, the tabulated expression for equation (4.14) can be written as follows:

a	b	c	d	e	x	y	z
1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	ϕ	ϕ	ϕ	1	ϕ	ϕ
ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	1
0	0	0	1	1	0	0	0

The following rules will be used to compute the intersection of any two Boolean expressions,

Rule 1: Let the Boolean set B = {0, 1, ϕ}, the
(1) x * x = x
(2) x * ϕ = ϕ * x = x
where x = 0, 1, or ϕ.
(3) 0 * 1 = 1 * 0 = null (impossible pattern)

Rule 2: (A + B + C) * (A + B + D) = A + B + CD

ALGORITHM IX: (Intersection of two Boolean expressions)

Step 0. Let $T_1(n,t_1)$ and $T_2(n,t_2)$ be the two inputs expressions, $T_3(n,t_3)$ be the output expression. NUM be the number of the common terms of T_1 and T_2. NUM is initialized by 0.

Step 1. (Searching for the common terms)
DO i=1 TO t_1
DO j=1 TO t_2
Compare $T_1(k,i)$ with $T_2(k,j)$, k=1,2,..,n
If they have the common terms then record them.
End of loop j
End of Loop i

Step 2. (Reorder the expressions of T_1 and T_2 by placing the common terms to the beginning terms, and copy this common terms to the first "NUM" terms of T_3)
IF NUM=0 THEN GOTO Step 3
ELSE
Reorder T_1 and T_2.
Copy first "NUM" terms to T_3.

Step 3. (Find the intersection)
TERMS=NUM
DO i=NUM+1 TO t_1
DO j=NUM+1 TO t_2
TERMS=TERMS+1
DO k=1 TO n
IF $T_1(k,i)$ = "ϕ" THEN
$T_3(k,TERMS)=T_2(k,j)$

```
ELSE
  IF (T1(k,i)="1" AND T2(k,j)="0") OR
     (T1(k,i)="0" AND T2(k,j)≠"1") THEN
     T3(k,TERMS=T1(k,i)
  ELSE
     TERMS=TERMS-1
  End of Loop k
End of Loop j and i
```

Following the above algorithm, the tabulated expression of the intersection of T_1 and T_2 can be expressed as follows:

T = T_1 * T_2

row	a	b	c	d	e	x	y	z
(1)	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
(2)	ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
(3)	ϕ	ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ
(4)	ϕ	ϕ	ϕ	1	ϕ	ϕ	ϕ	ϕ
(5)	ϕ	ϕ	ϕ	ϕ	1	ϕ	ϕ	ϕ
(6)	0	0	0	0	0	0	1	0

*

a	b	c	d	e	x	y	z
1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	ϕ	1	ϕ	1	ϕ	ϕ
ϕ	ϕ	ϕ	1	ϕ	ϕ	ϕ	1
0	0	0	1	1	0	0	0

=

a	b	c	d	e	x	y	z	operation
1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	common term
ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	common term
ϕ	ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ	common term
ϕ	ϕ	ϕ	1	ϕ	1	ϕ	ϕ	$T_1(4)*T_2(4)$
ϕ	ϕ	ϕ	1	ϕ	ϕ	ϕ	1	$T_1(4)*T_2(5)$
0	0	0	1	1	0	0	0	$T_1(4)*T_2(6)$
ϕ	ϕ	ϕ	ϕ	1	1	ϕ	ϕ	$T_1(5)*T_2(4)$
ϕ	ϕ	ϕ	ϕ	1	ϕ	ϕ	1	$T_1(5)*T_2(5)$
0	0	0	1	1	0	0	0	$T_1(5)*T_2(6)$
null								$T_1(6)*T_2(4)$, 0*1 for x
null								$T_1(6)*T_2(5)$, 0*1 for z
null								$T_1(6)*T_2(6)$, 0*1 for d

After deleting the duplicated terms and rearranging their order, the reduced table becomes:

a	b	c	d	e	x	y	z
1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	1	ϕ	ϕ	ϕ	ϕ	ϕ
ϕ	ϕ	ϕ	1	ϕ	1	ϕ	ϕ
ϕ	ϕ	ϕ	1	ϕ	ϕ	ϕ	1
ϕ	ϕ	ϕ	ϕ	1	1	ϕ	ϕ
ϕ	ϕ	ϕ	ϕ	1	ϕ	ϕ	1
0	0	0	1	1	0	0	0

The above algorithm is called Regular Boolean algorithm. To accelerate the speed of convergence one may specify the maximum number of simultaneous failures. Once this number has been specified the impossible patterns will be reduced. For example, if a single failure is assumed, recall from the first term of (4.13), a, the component "a" is bad and the remaining components are "don't care," either good or bad. Since at most one faulty component is allowed, the remaining components would not be bad, therefore, the component "a" is the only bad component in this possible pattern. It is the same as the remaining terms in equation (4.13). The tabulated expression will be rewritten as follows:

a	b	c	d	e	x	y	z
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0
0	0	0	1	0	0	0	0
0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	0

Similarly, the tabulated expression for (4.14) can be simplified by replacing "ϕ" by "0."

The intersection of these two expressions, as shown above, will be reduced by deleting those terms which contain more than one failure, that is, the items 4, 5, 6, 7, and 8 are deleted, and by replacing all "ϕ" by "0" in items 1, 2, and 3. The tabulated expression of this intersection is thus shown as follows:

a	b	c	d	e	x	y	z
1	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0
0	0	1	0	0	0	0	0

which shows that the components a, b and c are possibly "faulty." Similarly, for any multiple failure case, say t failures, the impossible fault patterns with more than t faulty components will be eliminated.

ALGORITHM X: (Regular Boolean Algorithm)

Step 1. Retrieve the component subdivisions table and input t. (the maximum number of simultaneous failures)
Step 2. Choose a subdivision.
Step 3. Call subroutines to derive the test results and the tabulated expression.
Step 4. Search the pattern which contains more than t's "1" and delete the impossible patterns.
Step 5. If the subdivision is the first one, go to step 2, otherwise do the next step.
Step 6. Call a subroutine to compute the product of this expression and the previous one.
Step 7. Search for the impossible patterns and delete them.
Step 8. Repeat the above steps until the actual faulty component is determined.

Since the Regular Boolean algorithm often requires a great number of steps to accelerate the speed of convergence, two additional algorithms are presented: Boolean Exact Algorithm and Boolean Heuristic Algorithm.

Boolean Exact Algorithm: This algorithm is developed by applying the concept of the Exact algorithm into the Regular Boolean algorithm. Conceptually, recall from the discussion of the Exact algorithm, the test result obtained from a given step of the algorithm will indicate all of the good components in group "2," and a set of all possible fault patterns will be generated by this test result (by Regular Boolean algorithm). However, some of these patterns will be eliminated when one or more components, which were known to be good, were predicted as bad by the pattern. "Don't care" values in the pattern will be replaced with good when a component is known to be good.

For example, consider the resultant expressions T_1 and T_2, since the components x and z are known to be good in the first subdivision (T_1) and then all ϕ in columns x and z are replace by "0" to rewrite the tabulated expression for T_1.

a	b	c	d	e	x	y	z
1	ϕ	ϕ	ϕ	ϕ	0	ϕ	0
ϕ	1	ϕ	ϕ	ϕ	0	ϕ	0
ϕ	ϕ	1	ϕ	ϕ	0	ϕ	0
ϕ	ϕ	ϕ	1	ϕ	0	ϕ	0
ϕ	ϕ	ϕ	ϕ	1	0	ϕ	0
0	0	0	0	0	0	1	0

Similarly, the component y is known to be good in the second subdivision (T_2). Therefore, the impossible patterns in T_1 will be eliminated again for the good component y, that is, the last row will be deleted because of contradicting that component y is known to be good, and all ϕ in the column y of the remaining rows will be replaced by "0." Since the components x, y, and z are known to be good, as a result, all "ϕ" in the columns x, y, and z in T_2 are replace by "0." This heuristic reduces and simplifies the possible patterns for both T_1 and T_2.

T_1:

a	b	c	d	e	x	y	z
1	ϕ	ϕ	ϕ	ϕ	∩	0	0
ϕ	1	ϕ	ϕ	ϕ	0	0	0
ϕ	ϕ	1	ϕ	ϕ	0	0	0
ϕ	ϕ	ϕ	1	ϕ	0	0	0
ϕ	ϕ	ϕ	ϕ	1	0	0	0

T_2:

a	b	c	d	e	x	y	z
1	ϕ	ϕ	ϕ	ϕ	0	0	0
ϕ	1	ϕ	ϕ	ϕ	0	0	0
ϕ	ϕ	1	ϕ	ϕ	0	0	0
0	0	0	1	1	0	0	0

After multiplying these two Boolean expressions, the reduced table for the intersection is,

a	b	c	d	e	x	y	z
1	ϕ	ϕ	ϕ	ϕ	0	0	0
ϕ	1	ϕ	ϕ	ϕ	0	0	0
ϕ	ϕ	1	ϕ	ϕ	0	0	0
0	0	0	1	1	0	0	0

The tremendous number of possible fault patterns is then greatly reduced. Moreover, if the maximum number of simultaneous failures is specified, the resultant expression will be much more applicable.

ALGORITHM XI: (Boolean Exact Algorithm)

Step 1. Retrieve the component subdivisions table and input t. (The maximum number of simultaneous failures)
FLAG(j)=.false., j=1,2,..,n.
Step 2. Choose a subdivision.
Step 3. Call subroutines to derive the test results and the tabulated expression.
Step 4. Search the pattern which contains more that t's "1", and delete the impossible patterns.
Step 5. Let GOOD(j) be the good components, j=1,2,..,s.
(For single failure case, all or all but one components in group "2" are known to be good.
For multiple failure case, the component with test result "0" is indicated to be good).
FLAG(GOOD(j))=.true., j=1,2,..,s.
Step 6. If this subdivision is the first one, go to Step 2.
Step 7. (Eliminate the impossible patters for both expressions).
Delete the patterns which predict the good components as bad, and replace all "ϕ" to be "0" for the good components.
Step 8. Call a subroutine to compute the product of this expression and the previous one.
Search for the impossible patterns and delete them.
Step 9. Repeat the above steps until the actual faulty components are determined.

Boolean Heuristic Algorithm: Because of the advantages discussed in the previous algorithms, where the heuristic algorithm was applied to the Regular Boolean algorithm, the heuristic algorithm can be carried a step further than indicated above. Most of the good components will be determined in few steps using the help of the coupling table.

ALGORITHM XII: (Heuristic Boolean Algorithm)

Step 1. Retrieve the component subdivisions table and input t. (The maximum number of simultaneous failures)
FLAG(j)=.false., j=1,2,..,n.
Step 2. Choose a subdivision.

Step 3. Call subroutines to derive the test results and the tabulated expression.

Step 4. Search the pattern which contains more than t's "1", and delete the impossible patterns

Step 5. Let GOOD(j) be the good components, j=1,2,..,s.
Retrieve the coupling table TABLE(m,n-m)
DO j=1 TO s
  Let x be the row of the component GOOD(j)
  DO k=1 TO n-m
    IF TABLE(x,k)=1 THEN
       Record this component as good
    End of Loop k
  End of Loop j

Step 6. FLAG(GOOD(j))=true., j=1,2,..,s

Step 7. If this subdivision is the first one, go to Step 2.
(Eliminate the impossible patterns for both expressions)
Delete the patterns which predict the good components as bad, and replace all "ϕ" to be "0" for the good components.

Step 8. Call a subroutine to compute the product of this expression and the previous one.
Search for the impossible patterns and delete them.

Step 9. Repeat the above steps until the actual faulty components are determined.

CHAPTER 5

EXAMPLES

In order to implement the algorithms discussed in the previous chapters, two examples, with details, are presented in this chapter. In the first section a linear circuit a BJT small signal amplifier circuit with beta-independent bias is used to describe the procedure of a linear circuit package. The second example is a power supply circuit[3] which demonstrates the nonlinear circuit package.

Linear Case

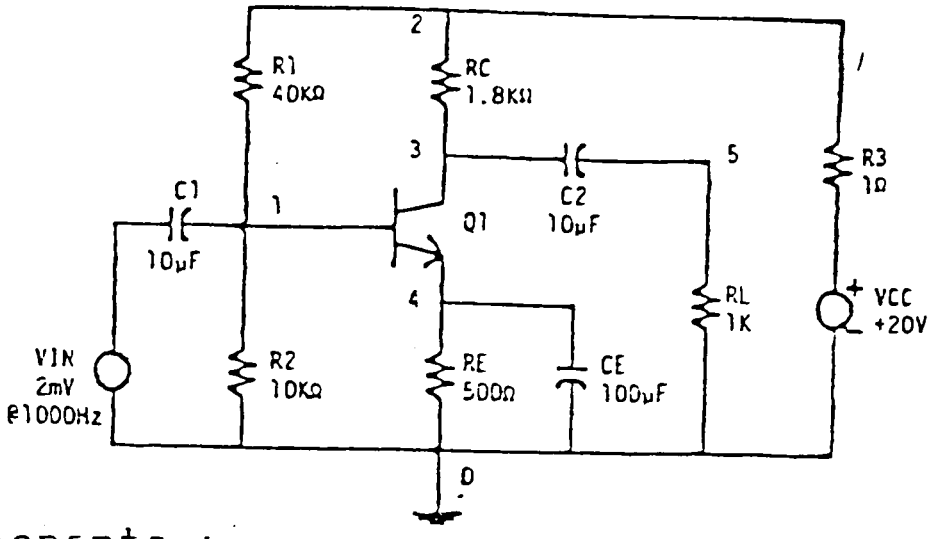
Consider a linear circuit, a BJT small signal amplifier circuit with beta-independent bias, as shown on the data sheet in Table 5.1. The data sheet contains information about the three inputs in the test program generation process. The user will enter the above input data by responding to the prompt shown on the terminal under "off-line."

L-Matrix: After the circuit description is entered, an incidence matrix for this circuit is created as follows:

$$A = \begin{bmatrix} 1 & 0 & -1 & 0 & 0 & 0 & -1 & -1 & 0 & -1 & -1 \\ -1 & -1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & -1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 1 & 0 \end{bmatrix}$$

The matrix D, in the equation (4.4) is calculated by calling a subroutine DMXGEN to transfer the matrix A to the following form

Table 5.1. Data Sheet - Linear Circuit



Number of components : 11.
Number of Nodes : 6

Component Description

	1	2	3	4	5,6	7	8	9	10
(1)	C	R	R	R	Q	R	C	C	R
(2)	C1	R1	R2	RC	Q1	RE	CE	C2	RL
(3)	1.0E-05	40000	10000	1800	2000 0 240 1.0E-05	500	1.0E-04	1.0E-05	1000
(4)	0	2	1	2	3	4	4	3	5
	1	1	0	3	1 4	0	0	5	0

		(1)	(2)	(3)	(4)
component	abbr	notation	component value	node	
resistor	R	Rn	resistance	(+), (−)	
capacitor	C	Cn	capacitance	(+), (−)	
inductor	L	Ln	inductance	(+), (−)	
op-amp	U	Un	voltage gain	input(+), (−), output	
transistor	Q	Qn	hybrid-pi (Hie, Hre, Hfe, Hoe)	(C), (B), (E)	
transformer	X	Xn	transformation ratio	primary	(+), (−)
				secondary	(+), (−)

Table 5.1. (Continued)

Number of components :
Number of Nodes :

Component Description

	(1)	(2)	(3)	(4)
component	abbr	notation	component value	node
resistor	R	Rn	resistance	(+), (−)
capacitor	C	Cn	capacitance	(+), (−)
inductor	L	Ln	inductance	(+), (−)
op-amp	O	On	voltage gain	input(+), (−), output
transistor	Q	Qn	hybrid-pi (Hie, Hre, Hfe, Hoe)	(C), (B), (E)
transformer	X	Xn	transformation ratio	primary (+), (−) secondary (+), (−)

Table 5.1. (Continued)

Source Description

Number of Voltage Sources : 2
Number of Current Sources : 0

branch number	n	1	11		
value	(**)	2.0E-03	10		
notation	xxxxx	VIN	VCC		
coincide ? (*)	(Y/N)	N	Y		

(*) Does the orientation of this source coincide with that of the branch ?
(**) any numerical expression.

Input Frequency

Frequency w = 1000

Test Points

Number of Test Points : 4

	notation	(1)	entry	(2)	columns	values
	xxxxx	Y/N	n	n	*,*,*,*	*,*,*
1	IC1	Y	1			
2	IR1	Y	2			
3	V45	N		3	1,5,10	-1,-1,-1
4	V13	N		2	2,4	1,-1
5						
6						

(1) from the entries of a ? (Y/N)
(2) number of nonzero elements

$$A = \left[\begin{array}{rrrrr|rrrrrr} 1 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & -1 & -1 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & -1 & -1 & 0 & 1 & 0 & 0 \\ \hline 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

and the matrix D is

$$D = \begin{bmatrix} -1 & -1 & -1 & 0 & -1 & -1 \\ 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & -1 & -1 & 0 & 0 \\ -1 & -1 & 0 & 1 & 0 & 0 \end{bmatrix}$$

By the equation (4.6) the L_{11} matrix is then derived as

$$L_{11} = \begin{bmatrix} 0 & -D \\ D^t & 0 \end{bmatrix} = \left[\begin{array}{rrrrr|rrrrrr} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 \\ \hline -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

The order of the components was C1, R1, R2, RC, Q1(1), Q1(2), RE, CE, C2, RL, and R3. After the transformation is performed, the order is rearranged to be C1, R1, RL, RC, Q1(1), CE, RE, C2, Q1(2), R3, and R2, and the component inputs vector a and component outputs vector b are shown as follows:

$$a = \begin{bmatrix} IC1 \\ IR1 \\ IRL \\ IRC \\ IBQ1 \\ VCE \\ VRE \\ VC2 \\ VCEQ1 \\ VR3 \\ VR2 \end{bmatrix} \qquad b = \begin{bmatrix} VC1 \\ VR1 \\ VRL \\ VRC \\ VBEQ1 \\ ICE \\ IRE \\ IC2 \\ ICQ1 \\ IR3 \\ IR2 \end{bmatrix}$$

where the first five components are in the tree edges, and the remaining components are in the co-tree edges.

As illustrated in the data sheet, information for source description shows that two voltage sources are located in the branches 1 and 11, respectively. Since in branch 1, the capacitor C1 is located in the tree edge, and the orientation of the voltage source is opposite to that of this branch, equation (4.7) therefore implies that the first column of L_{12} is the same as the row corresponding to C1 in the matrix L_{11}. Another voltage source, VCC, is located in the co-tree edge R3 with the same orientation. By equation (4.8), all elements of the second column of L_{12} are zero except the position which corresponds to that of C1, i.e., the 10th row and second column is -1. The L_1-matrix is then generated as follows:

$$L_1 = [\ L_{11} \mid L_{12}\ ]$$

$$= \left[\begin{array}{ccccccccccc|cc} 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{array}\right]$$

For convenience, we would like to change the component order back to the original component order. A matrix transformation, where the L_1-matrix and the component input/output are reordered, is shown.

$$a = L_{11} * b + L_{12} * u$$

where

$$L_1 = \left[\begin{array}{cccccccccc|cc}
0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \\
-1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & -1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\
0 & 1 & 0 & -1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
-1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
-1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
-1 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
-1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & -1
\end{array}\right]$$

$$a = \begin{bmatrix} IC1 \\ IR1 \\ VR2 \\ IRC \\ IBQ1 \\ VCEQ1 \\ VRE \\ VCE \\ VC2 \\ IRL \\ VR3 \end{bmatrix} \qquad b = \begin{bmatrix} VC1 \\ VR1 \\ IR2 \\ VRC \\ VBEQ1 \\ ICQ1 \\ IRE \\ ICE \\ IC2 \\ VRL \\ IR3 \end{bmatrix} \qquad u = \begin{bmatrix} VIN \\ \\ VCC \end{bmatrix}$$

The connection matrix L_1 can be easily checked for correctness by using Kirchoff's laws.

For L_2-matrix a test point description is input. The following four test points are selected: IC1, IR1, V45, and V13. The first two are current measurements, which are selected from the entries of a, and the remaining are voltage measurements, which are input by the users' specifications. Therefore, the first two rows of L_2-matrix duplicate the first two rows of L_1-matrix. In the third row, since

V45 = -VC1 -VBEQ1 -VRL, all elements in this row are 0 but the positions 1, 5, and 10 are -1. Similarly, the fourth row contains all 0 except for the 2nd and 4th columns which are 1 and -1, respectively. That is

$$L_2 = [\ L_{21} \quad L_{22}\ ]$$

$$= \left[\begin{array}{ccccccccccc|cc} 0 & 0 & 1 & 0 & 0 & 0 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & -1 & 0 & -1 & 0 & 0 \\ -1 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}\right]$$

The component connection model for this linear circuit can be expressed as follows:

$$\begin{bmatrix} a \\ y \end{bmatrix} = [\ L\ ] \begin{bmatrix} b \\ u \end{bmatrix}$$

where

$$L = \left[\begin{array}{c|c} L_{11} & L_{12} \\ \hline L_{21} & L_{22} \end{array}\right], \quad y = \begin{bmatrix} y1 \\ y2 \\ y3 \\ y4 \end{bmatrix} = \begin{bmatrix} IC1 \\ IR1 \\ V45 \\ V13 \end{bmatrix}$$

The Component Transfer Matrix Z: When the single frequency is input, the matrix Z is generated and stored in a data file named ZU.DT. The component transfer matrix in this example is formed as follows:

$$Z = \begin{bmatrix} Z_{C1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & Z_{R1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & Y_{R2} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & Z_{RC} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Z_{h1} & Z_{h2} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & Z_{h3} & Z_{h4} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & Y_{RE} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y_{CE} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y_{C2} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Z_{R1} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & Y_{R3} \end{bmatrix}$$

For the on-port components: resistors, capacitors, and inductors, the corresponding element in the matrix Z is nothing but the impedance or admittance, depending on whether a current or voltage appears in the vector a. Consider the first component, C1, where the current measurement is in the vector a and voltage measurement in b, therefore, the first diagonal block is the impedance of the component C1. Similarly, the impedance/admittance of the remaining one-port components will be located in the corresponding diagonal block. For the two-port components, such as transistors, a two by two block is used to describe this component, the values are defined by the hybrid-pi parameters or appropriate transformed parameters, depending upon the elements of the transistor model employed.

Recall that two pairs of transistor nodes are used to describe the incidence matrix. To define them as component inputs/outputs, there are four possible combinations which appear in the vectors a and b;

	a	b		a	b
(1)	V_{be}	I_b	(2)	V_{be}	I_b
	I_c	V_{ce}		V_{ce}	I_c
(3)	I_b	V_{be}	(4)	I_b	V_{be}
	I_c	V_{ce}		V_{ce}	I_c

When one of the above combinations is selected, the following calculation will be performed:

Let the matrix Z_{qi} be a 2x2 block of the component transfer matrix with the respect to the transistor, where Z_{qi} is written as

$$z_{qi} = \begin{bmatrix} z_1 & z_2 \\ z_3 & z_4 \end{bmatrix}$$

For the combination (1),

$$z_1 = H_{oe} / (H_{ie} * H_{oe} - H_{re} * H_{fe})$$
$$z_2 = -H_{re} / (H_{ie} * H_{oe} - H_{re} * H_{fe})$$
$$z_3 = -H_{fe} / (H_{ie} * H_{oe} - H_{re} * H_{fe})$$
$$z_4 = H_{ie} / (H_{ie} * H_{oe} - H_{re} * H_{fe})$$

For the combination (2),

$$z_1 = 1.0 / H_{ie}$$
$$z_2 = -H_{re} / H_{ie}$$
$$z_3 = H_{fe} / H_{ie}$$
$$z_4 = H_{oe} - H_{fe} * H_{re} / H_{ie}$$

For the combination (3),

$$z_1 = H_{ie} - H_{re} * H_{fe} / H_{oe}$$
$$z_2 = H_{re} / H_{oe}$$
$$z_3 = -H_{fe} / H_{oe}$$
$$z_4 = 1.0 / H_{oe}$$

For the combination (4),

$$z_1 = H_{ie}$$
$$z_2 = H_{re}$$
$$z_3 = H_{fe}$$
$$z_4 = H_{oe}$$

If one of the above combinations is computable, the combination is then selected as elements of the vectors a and b.

Component Subdivisions Table: The component subdivisions table is generated by the assumption that $[L_{21}^{2}]^{-L}$ exists. Therefore, 34 subdivisions are generated as follows:

Subdivision Number	Component Number				Subdivision Number	Component Number			
(1)	1	2	3	9	(18)	2	8	10	11
(2)	1	2	3	11	(19)	3	4	9	10
(3)	1	2	7	9	(20)	3	4	10	11
(4)	1	2	7	11	(21)	4	7	9	10
(5)	1	2	8	9	(22)	4	7	10	11
(6)	1	2	8	11	(23)	4	8	9	10
(7)	1	3	4	9	(24)	4	8	10	11
(8)	1	3	4	11	(25)	2	3	5	6
(9)	1	4	7	9	(26)	2	5	6	7
(10)	1	4	7	11	(27)	2	5	6	8
(11)	1	4	8	9	(28)	2	5	6	9
(12)	1	4	8	11	(29)	2	5	6	11
(13)	2	3	9	10	(30)	3	4	5	6
(14)	2	3	10	11	(31)	4	5	6	7
(15)	2	7	9	10	(32)	4	5	6	8
(16)	2	7	10	11	(33)	4	5	6	9
(17)	2	8	9	10	(34)	4	5	6	11

(where the component pair 5 and 6 is a two-port component)

Pseudo Circuit and Data Base: To generate the data base, assume the first component division is chosen, i.e., the group 1 components are #4, (#5, #6), #7, #8, #10, and #11, and the components #1, #2, #3, and #9 are contained in the group 2. By equations (2.12-2.15), the K-matrix is computed as follows:

$$\left[\begin{array}{c} a^1 \\ \hline a^2 \\ b^2 \end{array}\right] \quad \left[\begin{array}{c|c} K_{11} & K_{12} \\ \hline K_{21} & K_{22} \end{array}\right] \left[\begin{array}{c} b^1 \\ u \\ y \end{array}\right]$$

IRC	0	0	0	0	0	0	-1	0	0	0	-1	0	0	VRC
IBQ1	0	0	-1	1	1	0	0	0	0	0	0	0	0	VBEQ1
VCEQ1	0	1	0	0	0	0	0	0	0	0	0	0	1	ICQ1
VRE	0	0	0	0	0	1	0	1	0	0	0	1	0	IRE
VCE	0	0	0	0	0	1	0	1	0	0	0	1	0	ICE
IRL	0	0	-1	0	0	0	-1	0	0	0	-1	0	0	VRL
VR3	1	1	0	0	0	1	0	1	-1	0	0	1	1	IR3
IC1	0	0	0	0	0	0	0	0	0	1	0	0	0	u1
IR1	0	0	0	0	0	0	0	0	0	0	1	0	0	u2
VR2	0	1	0	0	0	1	0	1	0	0	0	1	0	y1
VC2	0	1	0	0	0	0	0	1	0	0	0	1	1	y2
														y3
VC1	0	-1	0	0	0	-1	0	0	0	0	0	-1	0	y4
VR1	1	0	0	0	0	0	0	0	0	0	0	0	1	
IR2	0	0	1	-1	-1	0	0	0	0	1	1	0	0	
IC2	0	0	-1	0	0	0	-1	0	0	0	-1	0	0	

The connection of the pseudo circuit can be verified by the test that they satisfy the Kirchoff's laws.

For the data base, the matrix M is derived from equations (2.17) and (2.18) and a data file for each matrix M is created with the name TE00**.DT where ** is the subdivision number.

After the test generation process is completed, consider the program verification with the test in which the impedance of the first component, C1, is changed to 1% of the nominal value, i.e., change the value from -j*5.000E+04 to -j*5.000E+02. The values of y are then computed as,

$$y = \begin{bmatrix} -(0.246291D-03) - j(0.348101D-05) \\ (0.249843D-03) + j(0.309305D-06) \\ -(0.302768D+01) - j(0.414982D+00) \\ (0.305966D+01) + j(0.116744D+00) \end{bmatrix}$$

Suppose also that the fifth subdivision is chosen, i.e., group 2 contains the components #1, #2, #8, and #9, then the data base M-matrix in a file named TE0005.DT will be retrieved. With the known values u and y, a^2 and b^2 are computed from the equations (2.17) and (2.18). We then substitute

the computed value of a^2 into the component transfer matrix for the group "2" to obtain the corresponding value for $\hat{b}^2$. Comparing the computed values b^2 and $\hat{b}^2$, if the difference of each component is less than a given tolerance, the result is defined to be good ("0"), otherwise it is bad ("1"). The result of this simulation is illustrated as follows:

Components	$\hat{b}^2$	b^2	Result
#1	2.492637D-02	2.492638D-04	1
#2	9.994136D+00	9.994136D+00	0
#8	1.756785D-04	1.756785D-04	0
#9	3.486035D-03	3.486035D-03	0

From this subdivision, we conclude that the test results for components #2, #8, and #9 are 0 and for component #1 is 1. With aid of the decision algorithm, the actual faulty components will then be determined.

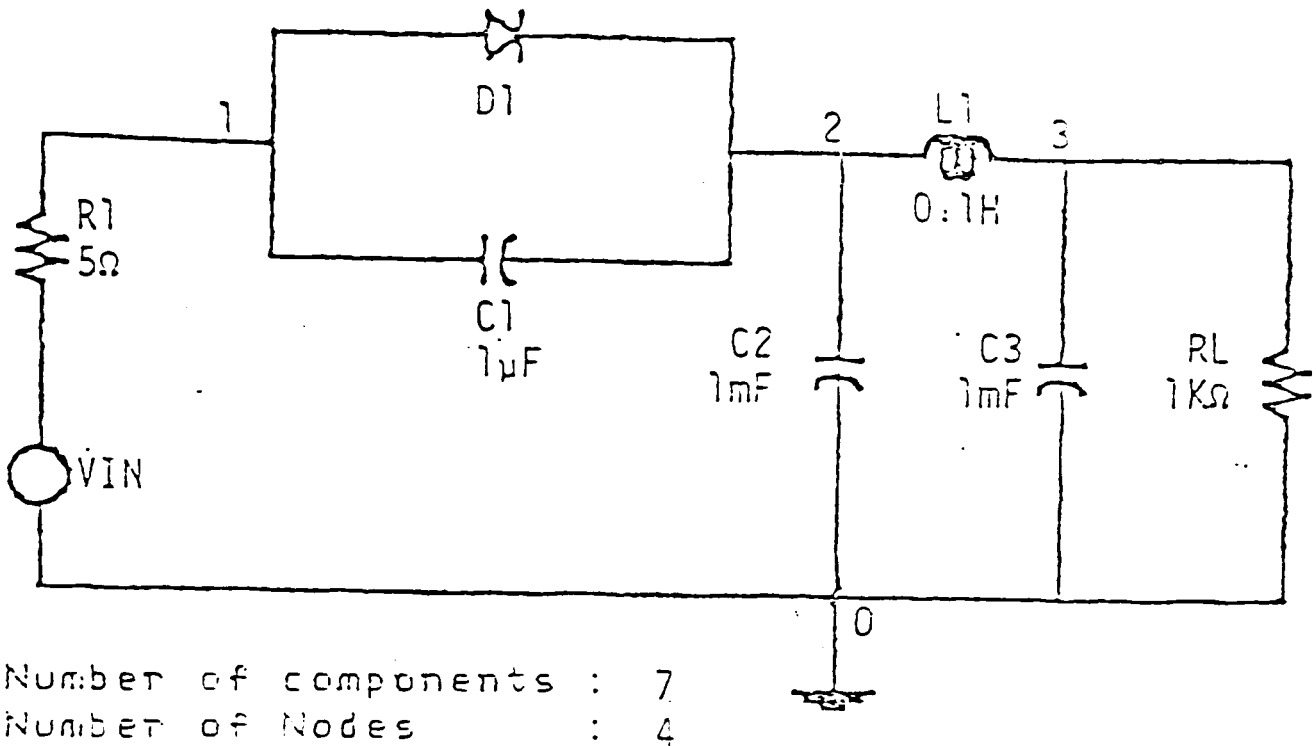
Nonlinear Case

Consider a nonlinear circuit, power supply circuit, as per the data sheet in Table 5.2. After user enters the number of components, the number of nodes, the circuit description, and the test point description, the connection matrix L and vectors a, b, u and y are automatically generated in the form

$$a = \begin{bmatrix} IR1 \\ VC1 \\ ID1 \\ VC2 \\ IL1 \\ VC3 \\ VRL \end{bmatrix} \quad b = \begin{bmatrix} VR1 \\ IC1 \\ VD1 \\ IC2 \\ IL1 \\ IC3 \\ IRL \end{bmatrix} \quad u = [u_1] = [VIN]$$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} ID1 \\ IL1 \\ VRL \end{bmatrix}$$

Table 5.2. Data Sheet - Nonlinear Circuit



Number of components : 7
Number of Nodes : 4

Component Description

	1	2	3	4	5	6	7		
(1)	R	C	D	C	L	C	R		
(2)	R1	C1	D1	C2	L1	C3	RL		
(3)	5	1U	DM1	1M	0.1	1M	1K		
	0	1	1	2	2	3	3		
(4)	1	2	2	0	3	0	0		

	(1)	(2)	(3)	(4)
component	abbr	notation	component value	node
resistor	R	Rn	resistance	(+), (-)
capacitor	C	Cn	capacitance	(+), (-)
inductor	L	Ln	inductance	(+), (-)
diode	D	Dn	model name	(+), (-)
transistor	G	Gn	model name	(C), (B), (E)

Model Parameters:

	Model Name	Model parameters
1.	DM1	D IS=1.0E-06 N=0.97

Table 5.2. (Continue)

Number of Voltage Sources : 1
Number of Current Sources : 0

branch number	n	1			-
value	(**)	SIN(0 10 60)			
notation	xxxxx	VIN			
coincide ? (*)	(Y/N)	N			

(*) Does the orientation of this source coincide with that of the branch ?
(**) any numerical expression.

SPICE Parameters

TRANsient : Time step : 10M
Final time : 200M
(option) starting time :

Control Cards (options) :

1.

2.

Test Points

Number of Test Points : 3

	notation	(1)	entry	(2)	columns	values
	xxxxx	Y/N	n	n	*, *, *, *	*, *, *
1	ID1	Y	3			
2	IL1	Y	5			
3	VRL	Y	7			
4						

(1) from the entries of a ? (Y/N)
(2) number of nonzero elements

$$L = \left[\begin{array}{c|c} L_{11} & L_{12} \\ \hline L_{21} & L_{22} \end{array}\right] = \left[\begin{array}{ccccccc|c} 0 & 0 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & 1 & 0 & 1 & 1 & 0 \\ -1 & 0 & -1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 & 0 & 1 \\ -1 & 0 & -1 & 0 & -1 & 0 & 0 & 1 \\ \hline 0 & -1 & 0 & 1 & 0 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 \\ -1 & 0 & -1 & 0 & -1 & 0 & 0 & 1 \end{array}\right]$$

SPICE Code for the Circuit: Based on the circuit description, a SPICE code is generated as shown below, several new nodes are assigned due to the performance of current measurements. The generated SPICE code is stored in a data file named MAIN.DT. The circuit diagram, with new nodes, is shown in Figure 5.1.

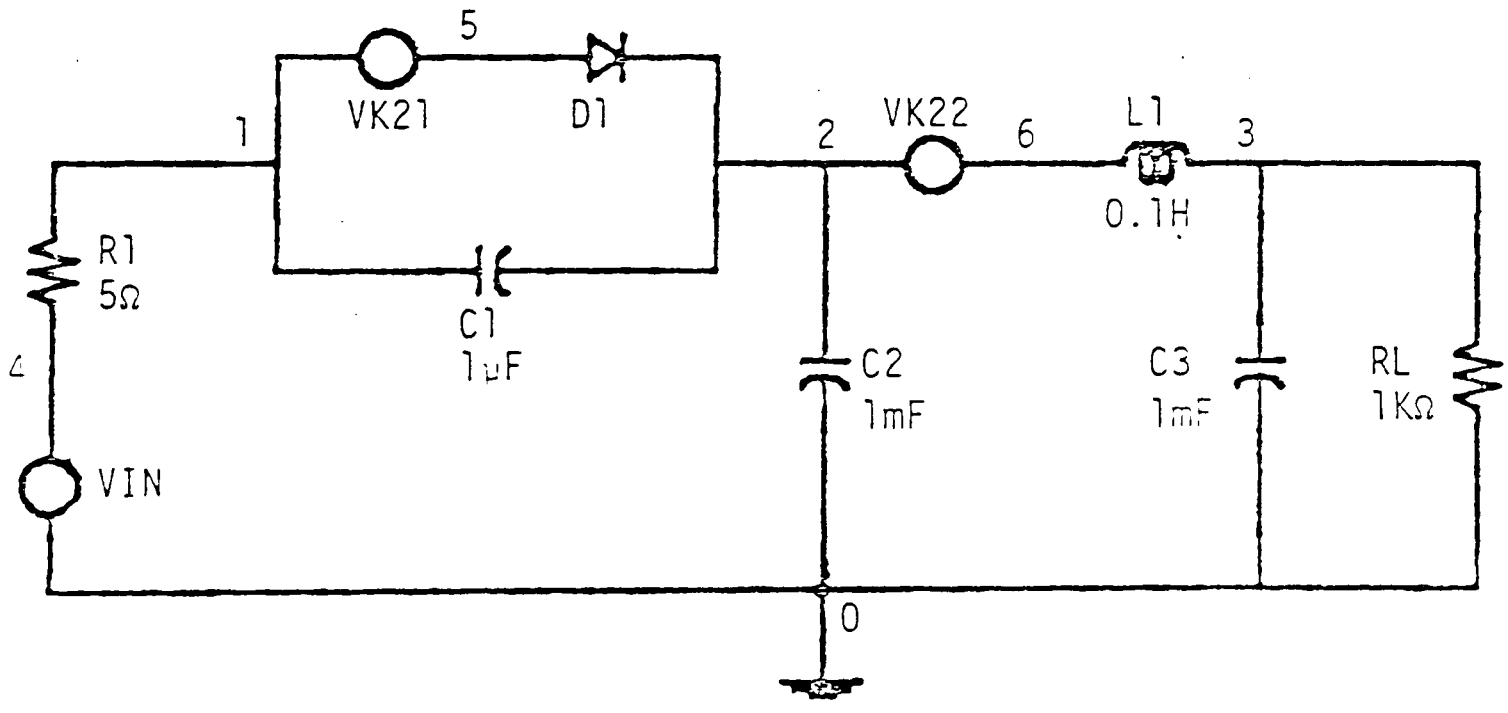


Figure 5.1. Power Supply Circuit

The SPICE code is,

```
SPICE CODE
.MODEL DM1 D IS=1.0E-06 N=0.97
+
VIN      4     0     SIN(0 10 60)
VK21     1     5     0
VK22     2     6     0
R1       4     1     5
C1       1     2     1U
D1       5     2     DM1
C2       2     0     1M
L1       6     3     0.1
C3       3     0     1M
RL       3     0     1K
.TRAN 10M 200M
```

Recall that the test points are ID1, IL1 and VRL. In order to measure the current flow through D1, a zero-valued voltage source, VK21, is added and a new node, node 5, is assigned to the circuit diagram. The current flows through the voltage source VK21, I(VK21), is then equal to ID1. Similarly, a zero-valued voltage source and node 6 are added to the circuit diagram for IL1.

SPICE Code for the External Input of "Pseudo Circuit": The external input of pseudo circuit

$$U^p = \text{col}\,[u \mid y] = \text{col}\,[\text{VIN} \mid \text{ID1 IL1 VRL}]$$

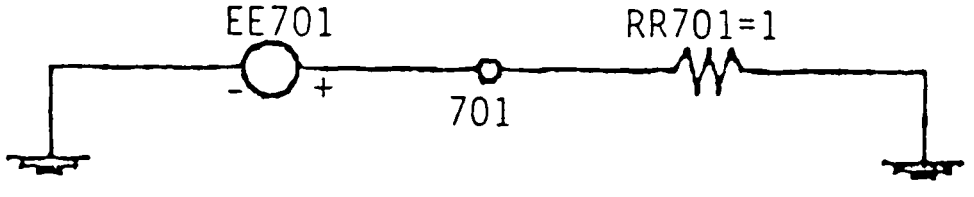
That is,

u_1 = VIN = V(4,0) --- the voltage at node 5.
y_1 = ID1 = I(VK21) --- the current flows through VK21.
y_2 = IL1 = I(VK22) --- the current flows through VK22.
y_3 = VRL = V(3,0) --- the voltage at node 3.

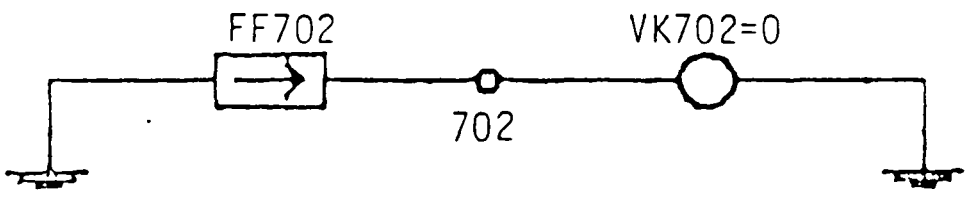
Therefore, the SPICE code is generated and stored in a file named SOURCE.DT,

```
EE701 701    0    4    0    1
RR701 701    0    1
FF702   0  702 VK21    1
VK702 702    0    0
FF703   0  703 VK22    1
VK703 703    0    0
EE704 704    0    3    0    1
RR704 704    0    1
```

The first two cards describe the voltage measurement for VIN. A voltage controlled voltage source, EE701, which is controlled by node 5, VIN, and a resistor, with resistance 1, are connected in series. The voltage at node 701 is, therefore, equal VIN.



For the current measurement, cards 3 and 4, a current controlled current source, FF702, which is controlled by I(VK21), (=ID1), and zero-valued voltage source is connected in the series so that the current flowing through the zero-valued source is the same as ID1.



Component Subdivisions table: 12 possible component subdivisions are generated as follows:

Subdivision Number	Component Number		
(1)	1	2	6
(2)	1	2	7
(3)	1	4	6
(4)	1	4	7
(5)	2	3	6
(6)	2	3	7
(7)	2	5	6
(8)	2	5	7
(9)	3	4	6
(10)	3	4	7
(11)	4	5	6
(12)	4	5	7

Pseudo Circuit and Data Base: If the first subdivision is chosen, i.e., the components #3, #4, #5 and #7 test the components #1, #2 and #6. By the equations (2.12) through (2.15), the K-matrix is calculated as follows:

$$\begin{bmatrix} ID1 \\ VC2 \\ IL1 \\ VRL \\ \hline IR1 \\ VC1 \\ VC3 \\ \hline VR1 \\ IC1 \\ IC3 \end{bmatrix} = \left[\begin{array}{cccc|cccc} 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline 0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ \hline -1 & 0 & -1 & 0 & 1 & 0 & 0 & -1 \\ 0 & 1 & 0 & 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 1 & 0 \end{array}\right] \begin{bmatrix} VD1 \\ IC2 \\ VL1 \\ IRL \\ \hline u_1 \\ y_1 \\ y_2 \\ y_3 \end{bmatrix} \qquad (5.1)$$

where u_1 = VIN, y_1 = ID1, y_2 = IL1, and y_3 = VRL.

A SPICE code for the equations (2.23) through (2.26) is generated as follows:

```
****    1
FF101     0 101 VK702    1
VK101   101 102      0
D1      102   0 DM1
****    2
EE101   103   0 POLY(2) 106 0 704 0 0 1 1
VK102   103 104      0
CC101   104   0     1M
****    3
FF102     0 105 VK703    1
VK103   105 106      0
LL101   106   0    0.1
****    4
EE102   107   0    704    0    1
VK104   107 108      0
RR101   108   0     1K
****    1
FF103     0 109 POLY(2) VK102 VK703 0 1 1
VK105   109 110      0
RR102   110   0      5
****    2
EE103   111   0    102    0    1
VK106   111 112      0
CC102   112   0     1U
****    3
EE104   113   0    704    0    1
VK107   113 114      0
CC103   114   0     1M
****    1
EE105   115   0 POLY(4) 102 0 106 0 701 0 704 0 0 -1 -1 1 1
RR108   115   0      1
****    2
FF104     0 116 POLY(3) VK102 VK702 VK703 0 1 -1 1
VK109   116   0      0
****    3
FF105     0 117 POLY(2) VK104 VK703 0 -1 1
VK110   117   0      0
.PRINT TRAN  V(115),V(110)
.PRINT TRAN  I(VK109),I(VK106)
.PRINT TRAN  I(VK110),I(VK107)
.END
```

As shown in equation (5.1) the first row shows that

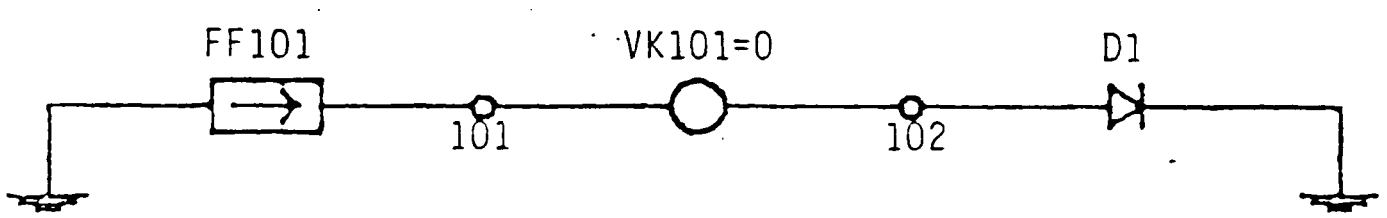
$$ID1 = y_1$$

i.e., the current flow through diode D1 equals to the external input y_1.

Define a controlled current source FF101 which is controlled by the current source y_1 (=I(VK702)), therefore, the code is generated as follows:

```
FF101   0 101 VK702     1
VK101 102   0
```

As shown in the following circuit, if the current flow through the diode is known, then the voltage across the diode can be computed.



Therefore, the voltage at node 102 is VD1.

For the voltage case, consider the second row of the matrix in equation (5.1),

$$VC2 = VL1 + y_3$$

The voltage across C2 equals to the voltage across inductor L1 and the external input y_3. The voltage source, EE101, is controlled by the sum of the voltages at nodes 106 and 704.

```
EE101  103   0 POLY(2) 106 0 704 0 0 1 1
```

which is equivalent to

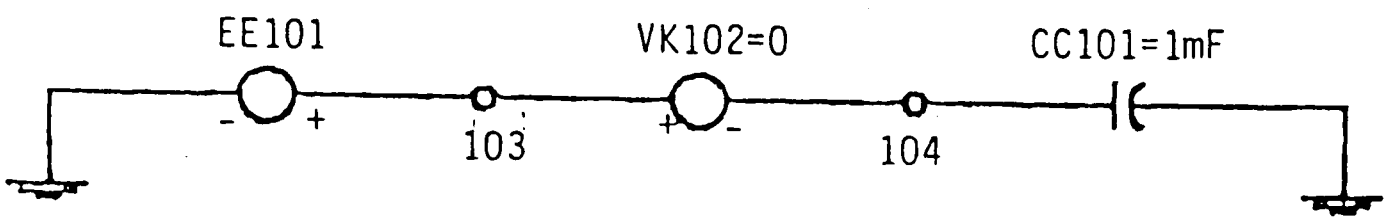
$$V(103,0) = V(106,0) + V(704,0)$$

(where POLY(2) is the number of controlled nodes, the last three numbers are the coefficients of the polynomial.)

To calculate VC2 in the entry of b^1. Two cards are specified,

```
VK102 103 104     0
CC101 104   0    1M
```

the equivalent circuit is



When the voltage is calculated, the voltage at node 103 is known, therefore, the current flows through the capacitor, IC1, is represented by I(VK102).

To compute b^2 and $\hat{b}^2$, consider the first element of a^2,

$$IR1 = IC2 + y_2,$$

The equivalent SPICE code is

```
EE103   0 109 POLY(2) VK102 VK703 0 1 1
VK105 109 110     0
```

gives the value of IR1 = I(VK105), and

```
RR102 110   0     5
```

Using this value of IR1 and the component values R1, the voltage VR1 which is an element of $\hat{b}^2$ is computed. The first element of $\hat{b}^2$ is to measure the voltage across node 110, V(110).

Consider the first element of b^2,

$$VR1 = -VD1 - VL1 + u_1 - y_3$$

The equivalent SPICE code is

```
EE105 115   0 POLY(4) 102 0 106 0 701 0 704 0 0 -1 -1 1 1
RR108 115   0     1
```

These cards compute the voltage of R1, VR1=V(115).

To compare these two values a print control card is used,

.PRINT TRAN V(115), V(110)

in which the values of b_1^2 and $\hat{b}_1^2$ are stored in the output file.

Each component subdivision creates a pseudo circuit, and each pseudo circuit generates a SPICE code as above, which is stored in a data file named TE00**.DT (where ** is the subdivision number).

Test Results: When the on-line component is conducted, either a simulation program or the ATE interface is used to obtain the test data. If the simulation program is used, user will specify the simulated faulty component, it may be open or short circuit, or out-of tolerance. A data file named SI00**.DT is used to store the SPICE code for the circuit with simulated faulty component (where ** is the faulty component number).

Suppose that the component 1, R1, is faulty, and it is simulated as an open circuit, the SPICE code for the first component was

```
R1      5   1   5
```

and now is changed to be

```
IR1     5   1   0
```

The SPICE code is stored in SI0001.DT.

If the test data is obtained from the measurement of the actual UUT, a data file is created in the host computer to collect the test data which transfers from the controller. A SPICE code is then generated by using the voltage controlled courses to simulate the test data, and stored in a data file named SI0001.DT.

Assume the 5th subdivision is chosen, then the SPICE program is executed with an input file which concatenates the data files as follows:

SI0001.DT
SOURCE.DT
TE0005.DT

The values b^2 and $\hat{b}^2$ are stored in the output file. Similar to the linear case, with the comparison of these two values and the aid of the decision algorithm, one will be able to identify the faulty component(s).

The test tesults of the program verification for the Power Supply Circuit are summarized in Table 5.3.

In addition to the Power Supply example, a couple more examples are presented. They are the Astable Multivibrator[25] in Figure 5.2 and the Oscillator[25] in Figure 5.3. The SPICE codes[25] for both circuits are shown in Figure 5.4, and their test results are also summarized in Table 5.3.

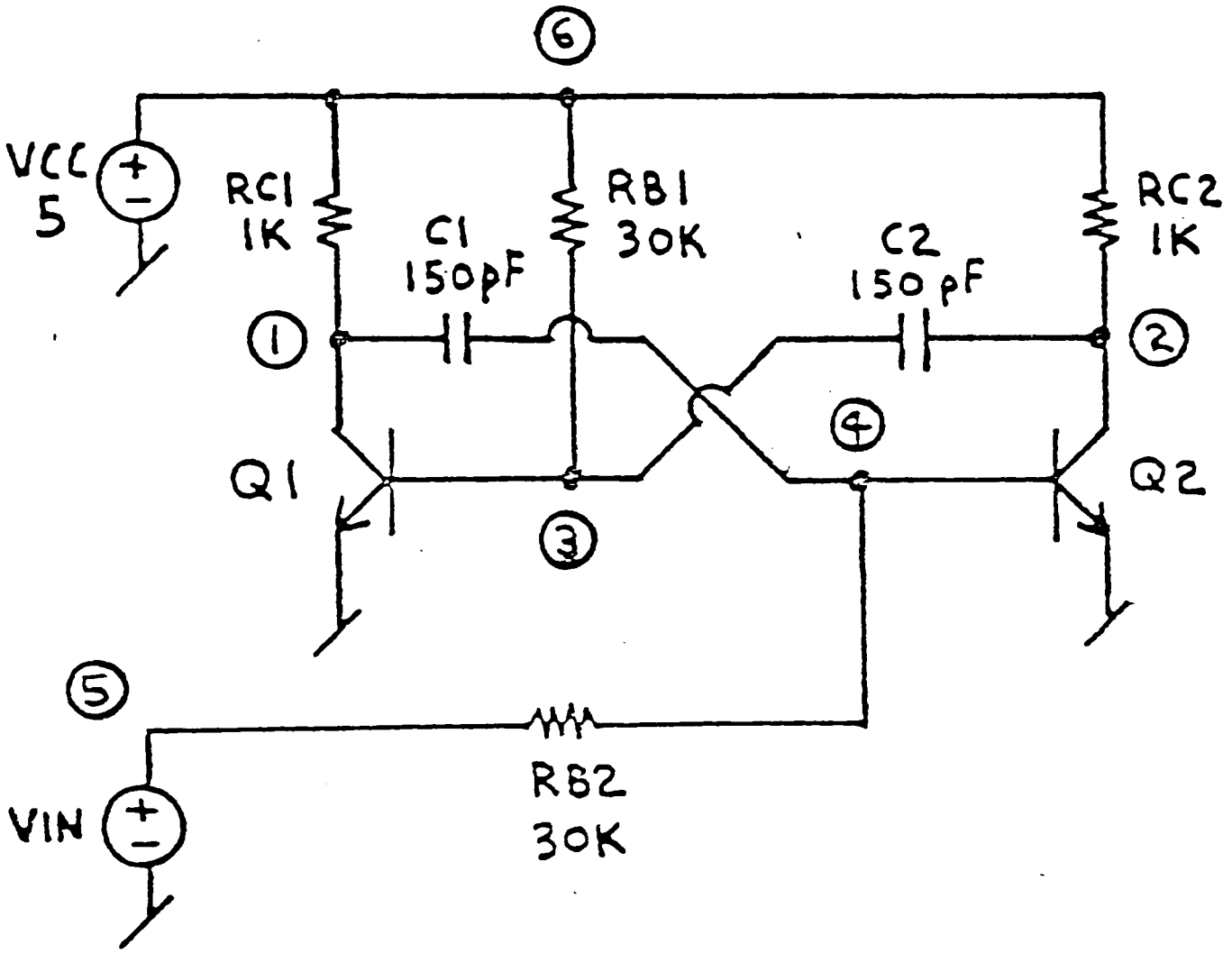


Figure 5.2. Astable Multivibrator

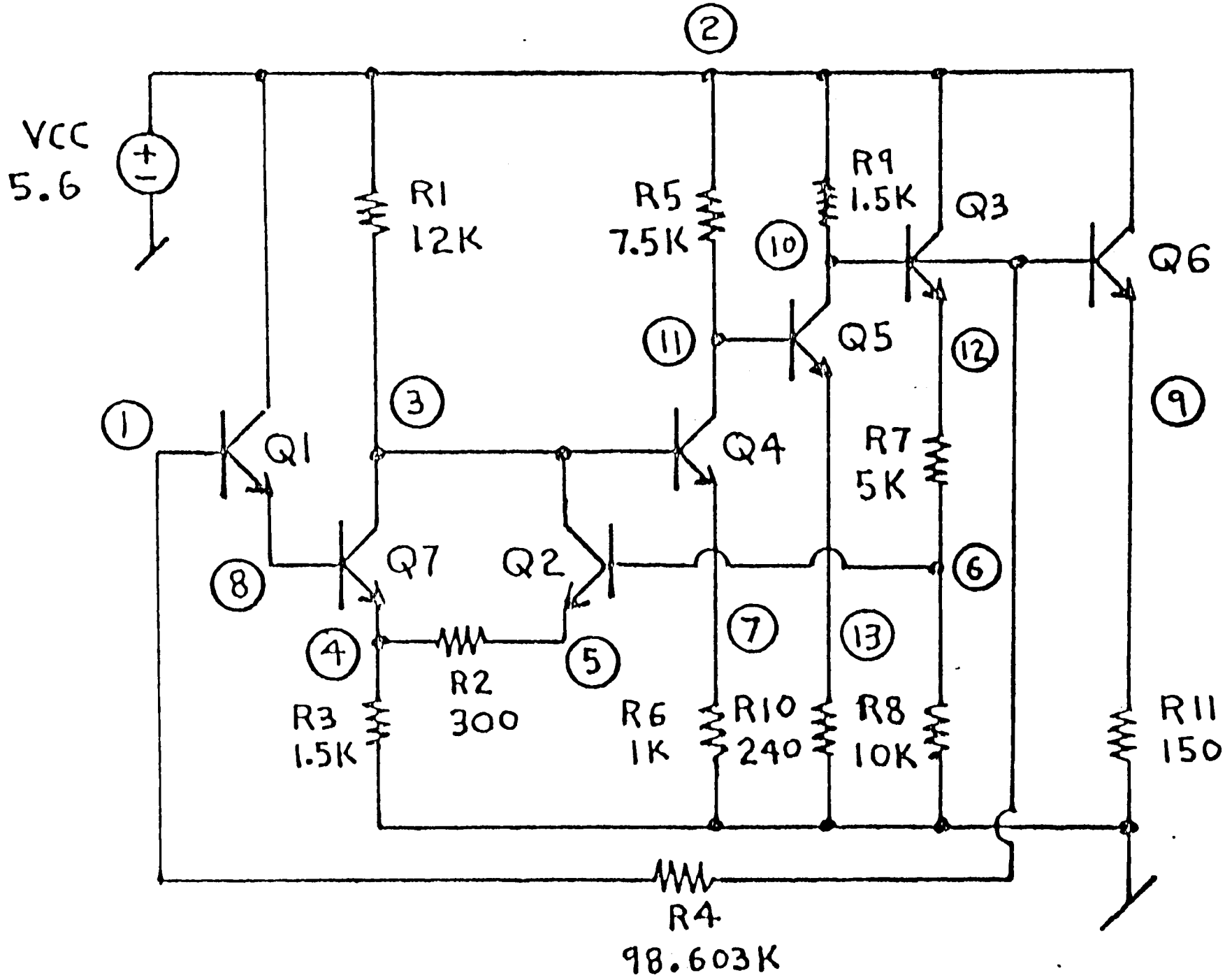


Figure 5.3. Oscillator

```
ASTABLE CKT - A SIMPLE ASTABLE MULTIVIBRATOR
.TRAN 0.1US 10US
VIN 5 0 PULSE(0 5 0 1US 1US 100US 100US)
VCC 6 0 5.0
RC1 6 1 1K
RC2 6 2 1K
RB1 6 3 30K
RB2 5 4 30K
C1 1 4 150PF
C2 2 3 150PF
Q1 1 3 0 QSTD
Q2 2 4 0 QSTD
.OUTPUT V1 1 0 PRINT TRAN PLOT TRAN
.OUTPUT V2 2 0 PRINT TRAN PLOT TRAN
.OUTPUT V3 3 0 PRINT TRAN PLOT TRAN
.OUTPUT V4 4 0 PRINT TRAN PLOT TRAN
.MODEL QSTD NPN(IS=1E-16 BF=50 BR=0.1 RB=50 RC=10 TF=0.12NS
+  TR=5NS CJE=0.4PF PE=0.8 ME=0.4 CJC=0.5PF PC=0.8 MC=0.333
+  CCS=1PF VA=50)
.END


OSC CKT - 1KHZ OSCILLATOR
VCC 2 0 5.6
Q1 2 1 8 Q1
Q2 3 6 5 Q1
Q3 2 10 12 Q1
Q4 11 3 7 Q1
Q5 10 11 13 Q1
Q6 2 10 9 Q1
Q7 3 8 4 Q1
R1 2 3 12K
R2 4 5 300
R3 4 0 1.5K
R4 10 1 98.603K
R5 2 11 7.5K
R6 7 0 1K
R7 12 6 5K
R8 6 0 10K
R9 2 10 1.5K
R10 13 0 240
R11 9 0 150
.MODEL Q1 NPN(BF=60 BR=0.205 IS=1.21E-15)
.END
```

Figure 5.4. SPICE Codes for ASTABLE and OSC CKTs

Table 5.3. Test Results - Nonlinear Circuits

(a) Power Supply

	1	2	3	4	5	6	7
OPEN	3	3					1
SHORT	3	2			1		1
out-of-tolerance	3	4					

(b) Astable Multivibrator

	1	2	3	4	5	6	7	8	9
OPEN	7	4							
SHORT	7	4							
out-of-tolerance	11								

(c) Oscillator

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
OPEN		6	1	1		1			3	2	2		6	1	3				
SHORT	3	7	1			1			3	1	1		6	1	2				
out-of-tolerance	8	7	1	2		2	1	2				1	2						

CHAPTER 6

CONCLUSIONS

An analog Automatic Test Program Generator, AATPG, for both linear and nonlinear circuits, based on the self-test algorithm, has been presented. The AATPG code was divided into off-line and on-line components. In the former, the test engineer inputs the system specifications to generate the test program and data base. The test program was also verified and validated before the actual on-line test was performed. The test is run in a fully automatic mode.

Basically, the algorithm is unique in its ability to test linear and nonlinear subsystems or models of arbitrary size. Actually, as shown in Chapter 1, an IC chip or a subsystem can be considered as an individual component in the test process. Thus one can use the algorithm to test modern electronic circuits.

Although the on-line computational requirements for the test algorithm do not compare with a simulation-before-test algorithm, they can be kept within reasonable bounds. One can limit the on-line computation, for instance, by restricting the number of algorithm steps. Therefore, the proposed AATPG code permits one to trade-off between on-line computational requirements and test points. A formula was derived in Chapter 4, to define the maximum number of algorithm steps required. This number depends on the structure of the matrix L_2 which relates to the choice of the test points. As the number of test points increase, the steps in the algorithm decreases. We believe the testability

discussed in the Appendix can be used to pick up a set of "good" test points. An alternate way to reduce the computational requirements using parallel processing was presented in Chapter 4. The minimum number of algorithm steps required shows that these subdivisions are chosen independently; therefore, those steps can be carried out simultaneously.

To identify the faulty component(s) from the test results, three decision algorithms were presented. The next algorithm subdivision can be chosen automatically or manually for the single failure case by user's choice. But the exact algorithm for the multiple failure case is not available. However, it is noteworthy that the underlying combinatorial decision problem is quite similar to the t-diagnosability problem usually associated with self-testing computer networks, wherein the multiple fault has been resolved.[14,15,30]

Another major open question with respect to the performance of the algorithm is robustness, i.e., its sensitivity intolerance deviations from nominal of the "good" components.

Finally, a big ambiguity set shown in Table 5.3 (c) is due to the performance of the simulation program. Since, in our algorithm, the test data and the test results are simulated from a pseudo circuit, the simulation may not be handled by the conventional circuit package. Therefore, either accurate simulation models or new circuit packages are needed.

REFERENCES

1. Aho, A.V., Hopcroft, J.E. and Ullman, J.D., The Design and Analysis of Computer Algorithm, Addison-Wesley, 1974, pp. 199-200.

2. Amin, T., unpublished notes, Bell Laboratories, 1980.

3. Aprille, T.J. and Trick, T.N., "Steady-State Analysis of Nonlinear Circuits with Periodic Inputs", Proc. IEEE, Vol. 60, pp. 108-114, 1972.

4. Boylestad, R. and Nashelksy, L., Electricity, Electronics, and Electromagnetics, Prentice-Hall, Englewood Cliffs, N.J., 1977, p. 290.

5. Breuer, M.A., Editor, Design Automation of Digital Systems, Vol. 1, Theory and Techniques, Prentice-Hall, Englewood Cliffs, N.J., 1972.

6. Chang, H.Y., Manning, E.G. and Metze, G., Fault Diagnosis of Digital Systems, Wiley, New York, N.Y., 1970.

7. DeCarlo, R.A. and Saeks, R., Interconnected Dynamical Systems, Marcel Dekker, New York, 1981.

8. Duhamal, P. and Rault, J.C., "Automatic test generation techniques for analog circuits and systems: a review", IEEE Trans. on Circuits Syst., Vol. CAS 26, pp. 411-440, July 1979.

9. Eleccion, M., "Automatic Test Equipment: Hardware and Software", IEEE Spectrum, June 1976, pp. 60-64.

10. El-Turkey, F.M., and Vlach, J., "Calculation of element values from node voltage measurements", 1980 Int. Symp. Circuits Syst. Proc., pp. 170-172.

11. Friedman, A.D. and Menon, P.R., Fault Detection in Digital Circuits, Prentice-Hall, Englewood Cliffs, N.J., 1970.

12. Greenbaum, J.R., "Computer-aided fault analysis - today, tomorrow, or never", in Rational Fault Analysis (ed. R. Saeks and S.R. Liberty), Marcel Dekker, New York, pp. 96-111, (1977).

13. Greenspan, A.M., "Automatic Test Systems Dedicated or Integrated", in Automatic Test Equipment: Hardware, Software and Management, (ed. F. Liguori), IEEE Press, New York, N.Y. 1974.

14. Hakimi, L.S., "Fault analysis in digital systems - A graph theoretic approach", in Rational Fault Analysis (ed. R. Saeks and S.R. Liberty), Marcel Dekker, New York, pp. 1-12, (1977).

15. Hakimi, S.L. and Nakajima, K., "On a theory of t-diagnosable analog systems", IEEE Trans. Circuits and Syst., to appear.

16. Healy, J.T., Automatic Testing and Evaluation of Digital Integrated Circuits, Prentice-Hall, Reston, Virginia, 1981.

17. Hayes, J.P., "Modeling faults in digital circuits", in Rational Fault Analysis (ed. R. Saeks and S.R. Liberty), Marcel Dekker, New York, pp. 78-95, (1977).

18. Knowles, R., Automatic Testing: Systems and Applications, McGraw-Hill (UK), 1976.

19. Lee, S.C., Digital Circuits and Logic Design, Prentice-Hall, Englewood Cliffs, N.J., 1976.

20. Liguori, F., Editor, Automatic Test Equipment: Hardware, Software and Management, IEEE Press, New York, 1974.

21. Lin, C.S., Huang, Z.F., and Liu, R.-W., "Fault Diagnosis of Linear Analog Networks: A Theory and its Application", Proc. IEEE Int. Symp. Circuits and Syst., pp. 1090-1093, May 1983.

22. Liu, R.-W., unpublished notes, Univ. of Notre Dame, 1980.

23. Mayeda, W., Graph Theory, Wiley, New York, 1972.

24. McAleer, H.T., "A Look at Automatic Testing", in Automatic Test Equipment: Hardware, Software and Management (ed. F. Liguori), IEEE Press, New York, N.Y., 1974.

25. Nagel, L.W., SPICE2: A computer program to simulate semiconductor circuits, Univ. of California, Berkeley, 1975.

26. NAP2: A Nonlinear Analysis Program for Electric Circuits, Version 2, Technical Univ. of Denmark, Lyngby, Denmark, Dec. 1976.

27. Peatman, J.B., The Design of Digital Systems, McGraw-Hill, 1972.

28. Plice, W.A., "A survey of analog fault diagnosis", presented at the Workshop on Analog Fault Diagnosis, Univ. of Notre Dame, Notre Dame, IN., May 1981.

29. Plice, W.A., "Automatic generation of fault isolation tests for analog circuit boards: a survey", presented at ATEX East '78, Boston, Sept. 1978, pp. 26-28.

30. Preparata, F.P., Metze, G., and Chien, R.T., "On the connection assignment problem of diagnosible systems", IEEE Trans. Electronic Computers, Vol. EC-16, pp. 448-454, (1967).

31. Saeks, R., "Criteria for analog fault diagnosis", in Nonlinear Fault Analysis, Texas Tech Univ., Lubbock, TX pp. 19-28.

32. Saeks, R., Singh, S.P., and Liu, R.W., "Fault isolation via components simulation", IEEE Trans. Circuit Theory, Vol. CT-19, pp. 634-640, Nov. 1972.

33. Sellers, F.F., Hsiao, M.Y., and Bearson, L.W., Error Detecting Logic, McGraw-Hill, New York, N.Y., 1968.

34. Special Issue on Fault-Tolerant Computing, IEEE Trans. Computers, Vol. C-20, Nov. 1971.

35. Special Issue on Fault-Tolerant Computing, IEEE Trans. Computers, Vol. C-22, Mar. 1973.

36. TESTAID-III Logic Simulator, Hewlett-Packard, Palo Alto, California, Jan. 1977.

37. To, K., and Tullos, R.E., "Automatic Testing Systems", IEEE Spectrum, September 1974, pp. 44-53.

38. Trick, T.N., Mayeda, W., and Sakla, A.A., "Calculation of parameter values from node voltage measurements", IEEE Trans. Circuits Syst., Vol. CAS-26, pp. 466-474, July 1979.

39. Wey, C.L., AATPG-Linear Circuit Version, User Manual, Texas Tech University, 1982.

40. Wey, C.L., AATPG-Nonlinear Circuit Version, User Manual, Texas Tech University, 1983.

41. Wu, C.-c., Ph.D. Dissertation, Texas Tech Univ., 1981.

42. Wu, C.-c., Nakajima, K., Wey, C.L. and Saeks, R., "Analog Fault Diagnosis with Failure Bounds", IEEE Trans. Circuits Syst., Vol. CAS-29, May 1982, pp. 277-284.

APPENDIX

Let B be a m by n matrix, n > m,

Definition:[21]

The global column-rank of B is said to be k if every combination of k columns of B is linearly independent, and some combination of (k+1) columns of B is linearly dependent.

Definition:[30]

A system is t-diagnosable if, given the results of all tests, one can identify the faulty units provided that the number of faulty units does not exceed t.

In the single failure case, we assume that at most one component is faulty. All possible test results obtained from a given step of an algorithm are summarized, and together the conclusions are summarized as follows:

Test Result (1 2 3 . . m)	Conclusions
0 0 0 . . 0	all group "2" components are good
1 0 0 . . 0	all group "2" except #1 are good
1 1 0 . . 0	all group "2" components are good
. . . . .	
1 1 1 . . 1	all group "2" components are good

Consistent with the above arguments, at each step of test algorithm, either all or all but one of the group "2" components are found to be good , i.e., at most one faulty component is found in each step.

Lemma 1:

A system is 1-diagnosable if and only if every pair of elements appear in at least one group "2".

Proof:

If x and y are two possible faulty components, there exists a subdivision which contains x and y. As shown in the above note, in each step, at most one faulty component is found, implying that x and y cannot be faulty simultaneously, therefore, the system is 1-diagnosable.

Conversely, in the worst case, if x and y do not appear in the same group "2", and also, x and y are found to be possibly faulty, implying that two possible faulty components are in this system, contradicting the assumption of 1-diagnosable. □

Theorem 1: (For the case of all one-port components)

The system is 1-diagnosable if and only if the flobal column-rank of the matrix L_{21} is at least 2.

Proof:

If the system is 1-diagnosable, by Lemma 1, every pair of elements appears in at least one group "2". Let x and y be any pair of elements appearing in the group "2". In our algorithm, the group "2" components are formed by a combination of columns of L_{21}, in which the matrix is invertible. Therefore, any two columns of this matrix of the column combination must be linearly independent, i.e., the columns corresponding to x and y are linearly independent. The global column-rank is then at least 2.

Conversely, if the global column-rank of L_{21} is at least 2, i.e., any two columns of L_{21} matrix are linearly independent, then every

pair of elements can be selected in the same group "2", by Lemma 1, the system is 1-diagnosable.□

For the multiple-port component case, Let

$L_{21} = B = [M_1 \quad M_2 \quad \ldots \quad M_r]$ and

W_i = # of columns in M_i, i = 1,2,..,r.

where $\sum_{i=1}^{r} W_i = n$

Definition:

M_1 and M_2 are linearly independent, if

$M_1 * A_1 + M_2 * A_2 = \underline{0}$ then $A_1 = \underline{0}$ and $A_2 = 0$

where A_1 and A_2 are vectors.

Definition: (Generalized Global column-rank) ("L-rank")

Let B be subdivided into r partitioned columns, M_i, as shown above.

The L-rank of B is said to be k, if every combination of k partitioned columns of B is linearly independent.

Theorem 2: (General case)

The system is a 1-diagnosable, if and only if the L-rank of the matrix L_{21} is at least 2.

Note:

Since, if any one port of the multiple-port component is faulty, the group is faulty, hence, we can group the multiple ports as one. Therefore, the multiple-port problem is equivalent to the one port case.

In the k failure case, we assume that at most k components are faulty. Consider the case of k=2, the possible test results with the

conclusions, in each step of the algorithm, are summarized as follows:

Test Results (1 2 3 . . m)	
0 0 0 . . 0	all group "2" components are good.
1 0 0 . . 0	all group "2" components are good. except component #1.
1 1 0 . . 0	all group "2" except #1 and #2 are good.
1 1 1 . . 0	at least one of group "1" components are faulty.
: : : : : :	" "
1 1 1 . . 1	" "

We conclude that, at each step, no more than two faulty components are found in the group "2".

Similar to the arguments and proofs in the single failure case, we conclude the following lemma and theorems for multiple failure case.

Lemma 2:

A system is 2-diagnosable if and only if every triplet of elements appears in at least one group "2".

Theorem 3: (For the case of all one-port components)

The system is 2-diagnosable if and only if the global column-rank of the matrix L_{21} is at least 3.

Theorem 4: (General case)

The system is 2-diagnosable if and only if the L-rank of the matrix L_{21} is at least 3.

Similarly, consider an arbitrary integer k, i.e., k failure case. all the possible results with the conclusions are shown as follows:

Test Results (1 2 3 . k-1 k k+1 . m)	Conclusions
0 0 0 . 0 0 0 . 0	all group "2" components are good.
1 0 0 . 0 0 0 . 0	all group "2" except #1 are good.
: : : : : : : : :	" "
1 1 1 . 1 0 0 . 0	all group "2" except #1 through #(k-1) are good.
1 1 1 . 1 1 0 . 0	all group "2" except #1 through #k are good.
1 1 1 . 1 1 1 . 0	at least one of group "1" components are faulty.
: : : : : : : : :	" "
1 1 1 1	" "

We conclude that at each step, no more than k faulty components are found in the group "2".

Lemma 3:

A system is k-diagnosable if and only if every (k+1)-tuple of elements appears in at least one group "2".

Theorem 5: (For the case of all one-port components)

The system is k-diagnosable if and only if the global column-rank of the matrix L_{21} is at least (k+1).

Theorem 6: (General case)

The system is k-diagnosable if and only if the L-rank of the matrix L_{21} is at least (k+1).