NBSIR 83-2794

On Generalizing the D-Algorithm

Center for Applied Mathematics
National Engineering Laboratory
U.S. Department of Commerce
National Bureau of Standards
Washington, D.C. 20234

Revised September 1983

Sponsored by:

Center for Applied Mathematics
Center for Electronics and Electrical Engineering
National Engineering Laboratory
U.S. Department of Commerce
Washington, D.C. 20234

NBSIR 83-2794

ON GENERALIZING THE D-ALGORITHM

J. Scott Provan*
Paul Domich

Center for Applied Mathematics
National Engineering Laboratory
U.S. Department of Commerce
National Bureau of Standards
Washington, D.C. 20234

Revised September 1983

Sponsored by:

Center for Applied Mathematics
Center for Electronics and Electrical Engineering
National Engineering Laboratory
U.S. Department of Commerce
National Bureau of Standards
Washington, D. C. 20234

*While performing this research in FY 1982, Provan was an NRC/NAS Postdoctoral Research Associate.

U.S. DEPARTMENT OF COMMERCE, Malcolm Baldrige, *Secretary*
NATIONAL BUREAU OF STANDARDS, Ernest Ambler, *Director*

ABSTRACT

We consider in this paper the d-algorithm of J. P. Roth, which tests for specific faulty behavior in an integrated circuit. We develop a formal and general mathematical description of the algorithm, which allows a large degree of flexibility and extension in its implementation. We discuss a subsequent FORTRAN coding of such an extended d-algorithm, along with some sample testing.

Table of Contents

1. Introduction

The advent of very large scale integrated circuits (VLSI) has intensified the interest in all aspects of testability, including test generation, self-testing, and testability measures, as well as the impact of these on design for testability and reliability of VLSI's. This report deals with a special aspect of test generation which is often referred to as "fault specific" test generation. By this we mean a procedure which generates — for a specific circuit and a suspected single malfunction or "fault" associated with that circuit — a test which detects the absence or presence of that malfunction in the circuit. The main tool for fault-specific test generation continues to be the d-algorithm. Developed by Roth in 1966 [19], the d-algorithm has been the target of continual development over the succeeding twenty years since its introduction.

This is a report on work of the Operations Research Division in developing a formal and general mathematical description of the d-algorithm, in designing a more flexible version based on this formalism, and in a subsequent FORTRAN implementation of this algorithm. The algorithmic scheme permits the handling of a very general type of circuit logic geared towards broader levels of circuit descriptions and analysis. It allows the coalescing of groups of components into a single gate, permits description of general fault logic, and allows handling of nonclassical and simultaneous types of faults. The flow of the algorithm itself can be brought partly under control of the user. This provides flexibility in testing specific types of circuits and generally allows for an improvement in the efficiency of the algorithm. Finally, the same algorithm which is used to generate test vectors for a circuit can be used to check a test vector efficiently in order to determine which faults it can detect. This feature is very useful when it is desired to find a set of test vectors which "cover" a given set of faults. Although the code as it stands is restricted essentially to digital nonsequential circuits, the tools developed up to this point show every indication of being applicable to fault detection in more general devices.

Section 1 gives a description of circuit faults and fault detecting algorithms, and provides a brief summary of work which has been done in the area. Section 2 describes the conventional d-algorithm in detail and develops general terminology used in Section 3. Section 3 describes the ways in which the description given in Section 2 can be generalized to include nonclassical faults and more general search techniques, and Section 4 describes aspects of the computer implementation of the generalized d-algorithm. The appendix describes a particular version of the d-algorithm which was implemented in FORTRAN at the Bureau of Standards, and includes some sample output.

1.1 Circuit Diagrams

Conventional versions of the d-algorithm address digital nonsequential (memory-less) circuits. The description of such a circuit involves two entities--lines, which carry 1-0 (high-low) signals between points of the circuit; and gates which process or generate signals on lines adjoining them. Each gate has input and output lines, along with an explicit description of the signals on the output lines of that gate which result from a specified set of signals on the input lines. Each line goes from the output of a single source gate to the inputs of one or more receiving gates, and indicates that the signal produced by the source gate is to be the corresponding input to the receiving gates. The entire circuit has special gates called input and output gates. A user of that circuit is able to control the circuit only through the sets of signals given to the input gates -called input vectors- and is able to observe the circuit only through signals -called output vectors- emanating from the output gates. Since the circuit has no memory, each gate description is independent of previous history of gate operation, and the gates and lines admit no "feedback," that is, no path of signals returns to a point previously visited. (This is what is meant by being nonsequential.) It follows that for any set of circuit inputs there is a well defined sequence for gate processing producing the unique set of circuit outputs which results from applying these inputs. It is possible to extend the description to include memory and feedback, although that problem is not addressed in this paper.

As an example circuit, we consider the "one-bit adder." It can be described simply as a circuit whose inputs are two one-bit binary numbers, and whose output is a two-bit binary number representing the sum of the inputs. One circuit which functions as a one-bit adder is shown in Figure 1.1. The gates are of five types and are interconnected with lines as shown. Input lines always enter on the left of the gate and output lines leave on the right. The gates marked "input" and "output" have no function other than to apply given inputs or to record the appropriate outputs. The remaining gates are processing gates and process inputs as their name indicates. Thus, a "not" gate simply reverses the input signal from 0 to 1 or from 1 to 0, the "or" gate produces a 1 output if at least one input has value 1, 0 otherwise, and the "and" gate produces a 1 output if both inputs have value 1, 0 otherwise. Of course, the output of these gates depends entirely on the values of the inputs. Further, it is not possible to travel progressively from gate to gate by traversing lines from output to input and return to a point previously visited. Thus, by processing the gates 1 through 5 in the order given, we obtain for any given set of inputs the unique set of outputs which corresponds to the two-bit sum of those inputs. We note, of course, that there may be many circuits which yield the same function as the circuit of Figure 1.1 but whose circuit description is essentially different. Figure 1.2 shows one of these circuits. This circuit is made up entirely of "nand" gates, whose function is exactly opposite that of the "and" gate. Clearly, both the gate description of the circuit and the actual physical layout of this circuit on a chip can affect the functioning of the circuit and the types of faults which can be expected to occur when operating such a circuit.

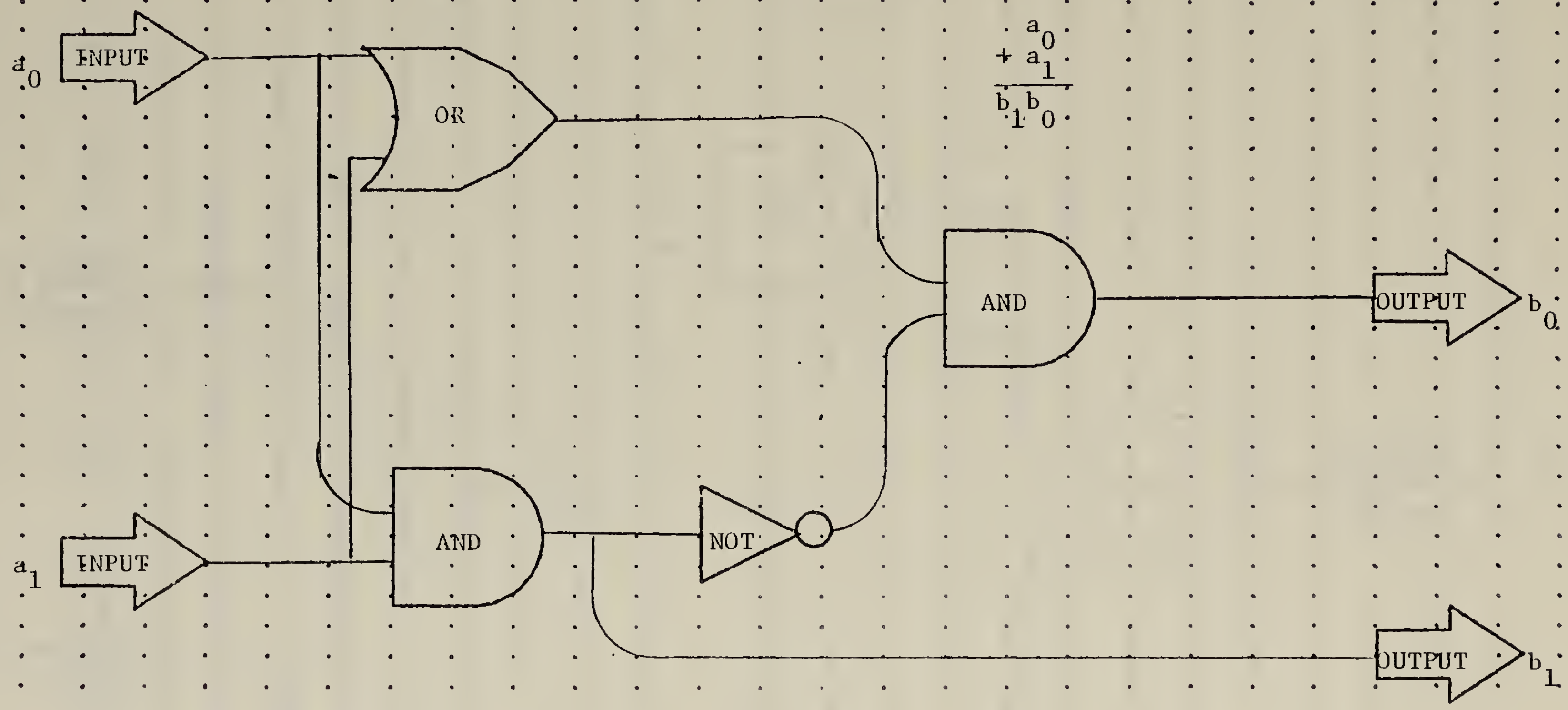


Figure 1.1: A realization of the 1-bit adder

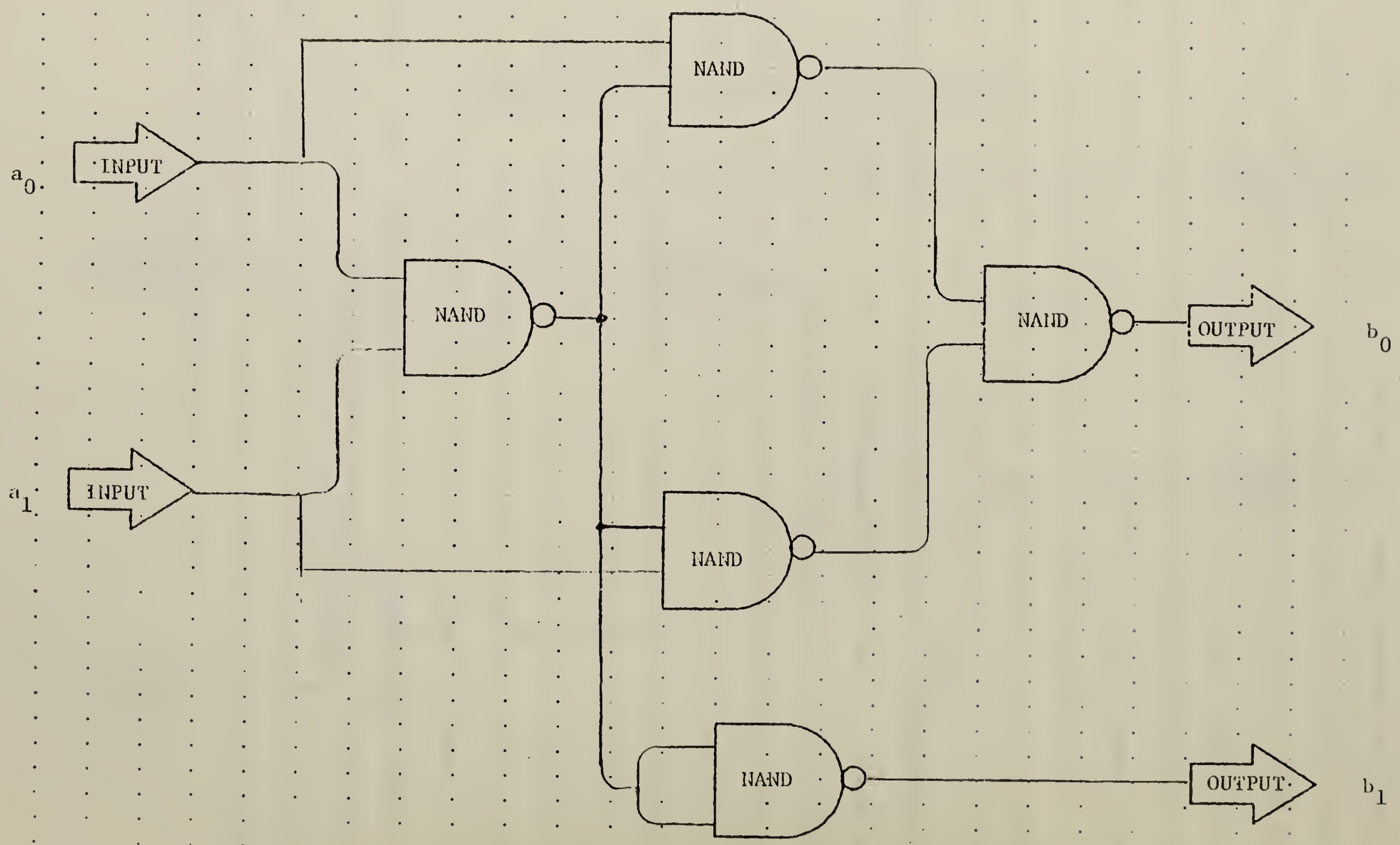


Figure 1.2: An all "NAND" gate realization of the 1-bit adder

1.2 Faults and Test Vectors

A *fault* is any change in the internal structure of a circuit which affects the normal functioning of that circuit. In order that we have a demonstrable method of detecting faults in a circuit, we insist that a fault, when it occurs, has a well-defined and consistent functional effect on the gates and lines associated with it. Thus, for example, "intermittent" faults--that is, faults which occur at random times in circuit operation--are not considered here. One can, in fact, think of a fault as being a "pseudogate" in the circuit which processes lines and/or replaces gates in the system. This pseudogate, unlike normal gates, functions in one of two "modes"--the first corresponding to the normal functioning of the circuit and the second corresponding to the functioning of the circuit when the fault occurs. The mode in which the pseudogate functions, of course, is unknown to the user, except when it causes a discrepency in the outputs from the normal function of the circuit.

As examples, we describe two types of faults which occur naturally in connection with physical malformation of circuits, namely, the *stuck-at* faults and the *cross-wire* faults. The classified faults previously studied are the stuck-at faults which correspond to breaks or shorts in a line and have the effect of producing a constant signal at the terminal of the line regardless of the signal impulse at the beginning of the line. We, therefore, can speak of *stuck-at-1* faults and *stuck-at-0* faults. They can be described by placing on the line a pseudogate with one input and one output, which under normal operating mode simply passes the proper signal through and under fault mode outputs a constant 1, or respectively an 0, regardless of the input (see Figure 1.3).

A cross-wire fault, (or bridging fault) as its name implies, occurs when two wires in close proximity inadvertently relay signals to each other. A cross-wire "and" fault occurs when a low (0) signal dominates a high (1) signal. The result can be described by placing on the two lines a pseudogate with two inputs and two outputs which under normal operating mode simply passes the respective signals through, and under fault mode outputs to both lines the signal corresponding to the "and" of the inputs (see Figure 1.4). A cross-wire "or" fault occurs when the high signal dominates the low signal, and the corresponding pseudo-gate described above acts in fault mode as an "or" gate.

The types of faults described above represent some of the standard types of faults which might occur on a circuit. They are, furthermore, "local" faults in the sense that they affect a relatively small number of signals in a small area of the circuit. One can imagine more complicated types of faults. A simultaneous fault, for example, consists of several individual faults occurring simultaneously in different parts of the circuit, and this often requires substantially different test finding procedures than simply testing for the faults individually. Other faults may involve certain complex types of gate malfunctions which have been observed only empirically and are described primarily by example. More general classes of faults will be discussed later in the paper.

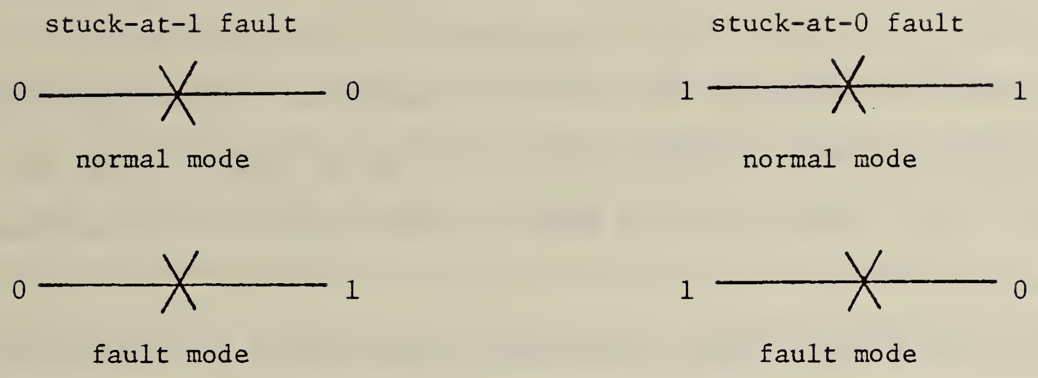


Figure 1.3: The stuck-at faults

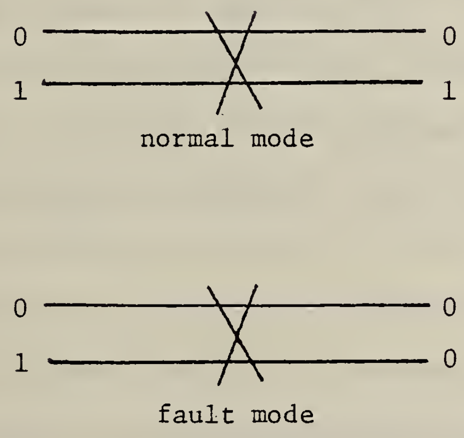


Figure 1.4: The cross-wire "and" fault

A <u>test vector</u> for a circuit and associated fault is an input vector whose output vector in the fault mode is <u>different</u> in at least one output from the output vector in the normal mode of circuit operation. A test vector, then, distinguishes the case when that fault occurs <u>and no other</u> from the case where no fault occurs. This is what is meant, in this report by the term "detects the fault."

The simultaneous fault concept allows one to distinguish the occurrence of several individual faults occurring <u>simultaneously</u> from the case where <u>none</u> of the faults occur.

A <u>testing scheme</u> for a circuit will consist of a <u>set</u> of test vectors applied to a circuit which will be able to detect any of a given set of faults. Since these faults may occur in any number of simultaneous combinations, the scheme should ideally be able to test not only each individual fault, but any subset of faults occurring simultaneously, and should further more be able to isolate which of the set of faults are occurring in any specific malfunction. The construction of such a testing scheme is beyond the scope of this paper. Henceforth we are interested in simply producing a test vector which will identify a given specific malfunction in the circuit, and no other. It will turn out that this problem alone is difficult enough for an entire report.

1.3 The d-Algorithm: Description and History

The general goal of the d-algorithm is: given the description of an integrated circuit and a fault which can be expected to occur on the circuit, find a test vector, for the circuit which detects the occurrence of this fault in terms of discrepancies in the outputs elicited from the circuit. Specifically, given a circuit description with a fault pseudogate (or set of simultaneous fault pseudo-gates), a given set of inputs detects the given fault if the outputs obtained when the pseudogate (or all pseudogates) are operating under fault mode differ from the corresponding outputs obtained when the pseudogate (or all pseudogates) are operating under normal mode.

It is clear that any digital nonsequential circuit can be tested completely by applying all possible inputs and observing the corresponding outputs for discrepencies between normal and fault mode operations. This is dependent only on the functional description of the circuit and detects all possible faults which affect the correct functional operation of the circuit. For large enough circuits, however, the number of such inputs would far exceed any reasonable amount of testing time alloted for the circuit. A method is needed for efficiently designing tests for certain "key" faults expected to occur in the circuits. One property that such an algorithm must take advantage of is the local nature of most circuit faults. In particular, the algorithm should be able to design a test for a fault by beginning at the fault site or sites and working towards the extremities (inputs and outputs) for the circuit.

Such algorithms naturally proceed in two stages: the <u>propagation</u> (or <u>observational</u>) stage, which insures that the fault can be seen as a discrepency in the outputs, and the <u>justification</u> (or <u>generation</u>) stage, which insures that there exist inputs which are consistent with the fault propagation found during the first stage.

The first attempt at such an algorithm, based on results by R. D. Eldrid [10] and unpublished work by Steiglitz and Armstrong, is the <u>path sensitization</u> method of fault detection. Path sensitization is used primarily for stuck-at faults on circuits made up of simple logic gates (such as and, or, not, nand, and nor). The object of the algorithm is to construct a "sensitized path" from the fault to a circuit output and subsequently derive inputs which support such a path. It is based on the fact that for each gate of the above type and each input line to be "sensitized," there is a unique signal which can be placed on the other input line (if any) which allows the input signals on the sensitized line to be distinguished by the signal on the output line. For an "or" gate, as an example, a given input can be distinguished at output by placing an "0" signal on the other input. The output value then exactly matches the signal of the sensitized line. Note that if "1" signal is placed on the other input, the gate output registers "1" regardless of the signal of the sensitized input, and so this configuration cannot pass through a sensitized input. For a "nand" gate (one whose output registers the opposite of the "and" of its inputs) and given input to be distinguished, placing a "1" on the other input allows it to be distinguished at the output. The output here registers the opposite signal to the input, but can nonetheless distinguish the signal on the given input.

The path sensitization algorithm starts at the stuck-at fault by fixing a value to the line which will distinguish the fault mode operation on that line from the normal mode operations--specifically, a 1 signal for a stuck-at-0 fault and a 0 signal for a stuck-at-1 fault. It then proceeds to construct a sensitized path from the fault to a circuit output by setting the unused input at each successive gate to the proper sensitizing value. The consequence of sensitizing such a path is that now the change in the line signal resulting from a fault is able to propagate through succession lines in the path until it is registered at the output gate as the opposite signal from the normal signal expected for that output. In fact, it follows that a sensitized path will register the appropriate stuck-at fault on <u>any</u> line of the path and thus the construction of long sensitized paths is an efficient way of detecting a large number of faults by a small number of tests. If, after the sensitized path has been constructed, the inputs can be assigned values which are consistent with (that is, produce the correct values for) the lines assigned in producing the sensitized path, than these input values constitute a test vector for the fault.

Two problems need to be resolved concerning the path sensitization algorithm as described above. The first is a procedural one, namely, how to structure the search for a sensitized path and subsequently justifying inputs. A search must, in particular, be efficient in searching for or discarding conditions for sensitized paths and input values, and at the same time insure that an exhaustive search has been made for such a path. A second and more fundamental problem is that certain faults may not be able to be tested by a single sensitized path (See Subsection 2.4). This is especially true of simultaneous faults or non-classical faults such as the cross-wire faults, but is true of single line faults as well. A complete propagation/justification type algorithm must therefore be able to "sensitize" an arbitrary number of lines of a circuit in order to be able to do a complete search for test vectors for a fault.

Both problems were effectively answered by Roth [19] with a procedure known as the d-algorithm. The d-algorithm can be thought of as a multi-path sensitization algorithm, although the description must be more elaborate and the record keeping mechanisms more sophisticated. It has the further property that it either finds a test vector for the given fault or demonstrates that the fault is undetectable, that is, that no input values are capable of registering a discrepency in any of the outputs as a result of the presence of that particular fault. Roth's work provides both a way of describing the action of a fault on a circuit and a effective method of manipulating the operation of the circuit in order to find the appropriate input values. It has therefore been an invaluable seminal paper for research in the field of circuit reliability. Section 3 describes some of the methods used to speed up the search and processing so as to make the test generation as efficient as possible.

A great deal of research has been done in the area of circuit testing following the Roth paper. A good sample of the directions this research took is found in [36]. As an indication of the amount of interest in the field, we refer the reader to twelve annual IEEE Conferences on Fault-Tolerance Computing [45] -[56], nine issues of IEEE Transactions on Computers devoted to fault-tolerent computing [36] -[44], several books on the subject [32] -[36], and lists of compiled literature [16], [26]. Some specific extensions of Roth's work include: structural factors in fault diagnosis [1], [2], [23], multiple fault diagnosis [3], [5], [7], [9], sequential circuit testing [21], fault location and coverage [6], [14], [31], and more general circuit and fault models [4], [10], [17]. Recent work has tended to be in the areas of design for reliability and testability [1], [8], [12], [13], [30], and tests for specific types of circuits [11], [12], [18], [27], [28], whereas and relatively little research has been devoted to further development of fault-specific test generation algorithms [20], [21], [22]. This is partly due to the fact that the problem of fault-specific test generation has a large degree of inherent intractibility. Another reason, however, is that an understanding of the d-algorithm has remained relatively inaccessible to the general scientific community. This is due partly to the fact that the d-algorithm has never been presented in a general mathematical format and papers have retained machinery and terminology specific to the task of fault-specific test generation in combinatorial circuits. This is unfortunate, for the d-algorithm concept has great potential for use in more general types of electronic fault-testing as well as for applications to system maintenance and reliability outside the electronics industry. It is with these thoughts in mind that we attempt to present the d-algorithm in a more general setting, one which, we hope, will encourage it to be put to use in a wider scope than it has been in the past.

2. The Conventional d-Algorithm

In this section we give a more specific description of the d-algorithm as it is commonly implemented at the present time. Our description will stress a formal structure of the algorithm and will be in a form appropriate for generalization in Section 3.

Subsection 2.1 introduces a general search procedure known as the "backtrack algorithm," which is found as an underlying principle in a large number of combinatorial algorithms which require exhaustive searches.

Subsection 2.2 deals with a description of gate and fault functions which allows implementation of the propagation and justification formats described in Section 1.

Subsection 2.3 combines the concepts of Subsections 2.1 and 2.2 in presenting a specific description of the d-algorithm, with an example in Subsection 2.4.

2.1 Backtrack Algorithms

A "backtrack algorithm" describes a general search technique for exhaustively considering a sequence of alternatives in order to achieve a desired solution. The goal of a backtrack algorithm in any application is to discard undesirable alternatives as quickly as possible while still maintaining a complete search among all of the available alternatives. The general format for a backtrack algorithm involves a set of decision points which are usually encountered in a fixed order and at which the algorithm must choose among one or more alternatives. The alternatives are provided by a decision list available at each decision point. Any backtrack algorithm proceeds from decision point to decision point, choosing the first available alternative on each decision list. If at any decision point the alternative chosen is found to be inconsistent with the alternatives chosen at previous points —that is, the set of alternatives chosen thus far cannot possibly yield the desired solution --then the next alternative on the decision list is considered. If a decision list is exhausted so that no alternative at that point is consistent with the alternatives chosen at previous points, then the algorithm backtracks to a previous decision point and chooses the next available alternative in the decision list for that point. If a set of alternatives is found, one for each decision point, which gives the desired solution, then the algorithm stops and exhibits such a solution. If the algorithm completes the entire search, that is, finally exhausts the decision list at the very first decision point, then the algorithm has covered every possible choice of alternatives for all decision points and therefore no solution exists to the problem. Figure 2.1 gives a flow chart for the general backtracking algorithm.

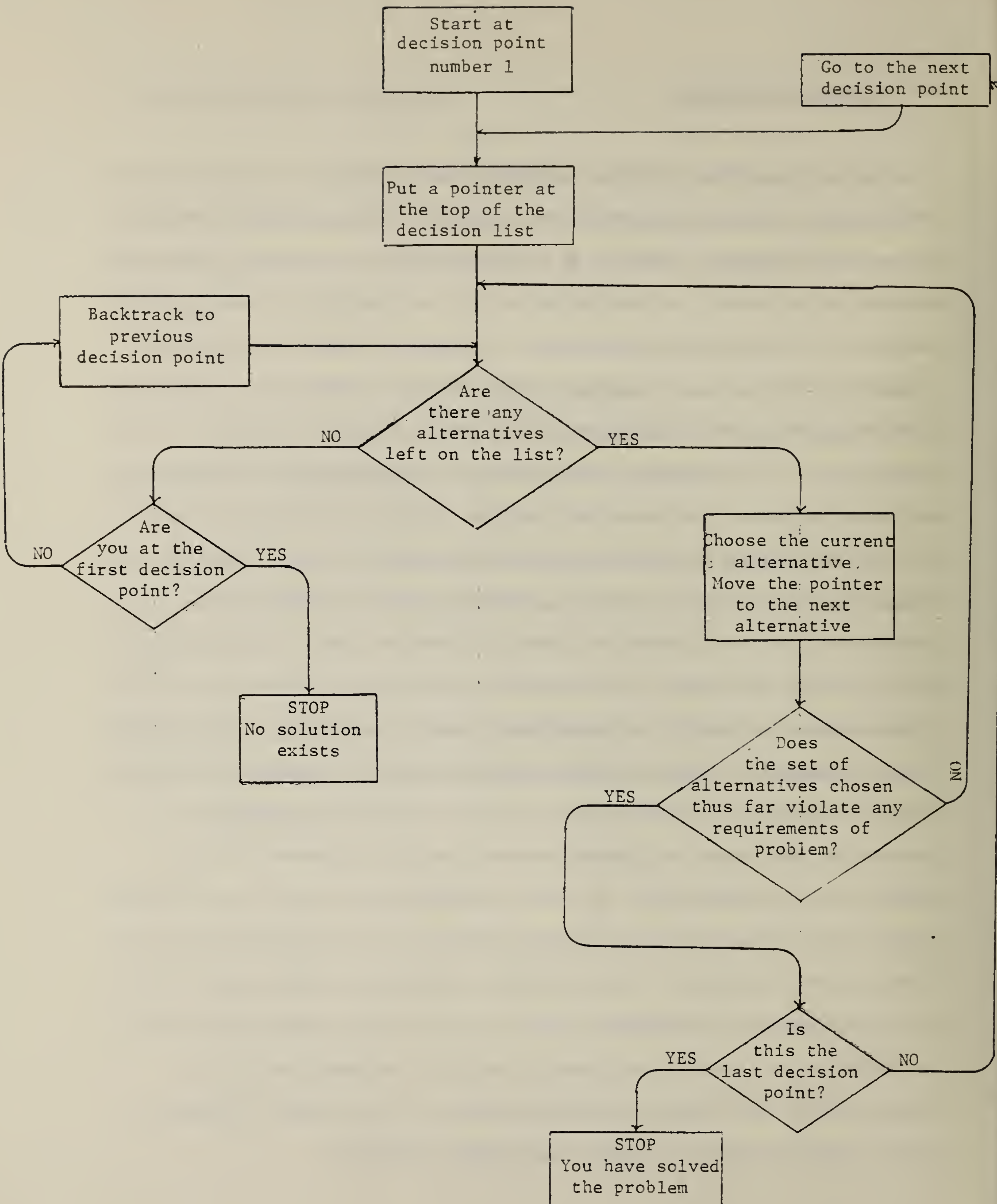


Figure 2.1: A general backtrack algorithm

We now give a broad description of the backtracking mechanism inherent in the d-algorithm, the details of which will be related in succeeding subsections. The decision points are the gates of the circuit (including pseudogates associated with faults), and the decision list associated with each gate is an assignment of "values" to the lines adjacent to that gate. The format of the decision list is called a <u>cube</u> for the gate, and it is explained in Subsection 2.2. The algorithm "processes" gates in some order, that order to be explained in Subsection 2.3 and in more detail in Section 3 and 4. At each gate, a set of line values is chosen for that gate which satisfies:

(1) it is consistent with the logical function of that gate;

(2) it is consistent with decisions made at previous gates, in that the line values chosen agree with the line values of every other gate on that line.

If the algorithm can find a set of values which satisfies these criteria, then it moves on to process the next gate. If it cannot find an acceptable set of values, then it backtracks to the previous gate considered and continues processing this gate. If a set of line values is chosen for all gates which is consistent from gate to gate and detects the given fault, then the algorithm stops and produces the test vector. If the decision list is exhausted for the first gate processed, then the algorithm stops and no set of inputs can detect the fault.

A backtrack algorithm may have to consider an enormous number of alternatives before it can ascertain whether or not a test vector exists for a circuit fault. In particular, if every possible sequence of decisions is considered, the algorithm would have to test as many alternative sequences as the product of the lengths of the decision lists for every decision point. For example, if a backtrack algorithm had twenty decisions to make consisting of two alternatives each, as many as 2^{20}, or approximately one million possibilities would have to be considered. The most important property of a good backtrack algorithm is the ability to discern at the earliest possible decision point when a sequence of alternatives is inconsistent with the desired solution. This may depend on features such as the order in which decision points are considered, the order in which the decision lists are scanned, and even the types of decisions which are made at each point. Subection 2.3 and Section 4 will address some of the methods which were considered in this study to improve the efficiency of the backtrack algorithm when applied to test generation and the d-algorithm.

2.2 Cube and Fault Propagation

We now describe more precisely the structure of the decision lists, or "cubes," associated with each gate. A cube is simply an assignment of line values to lines adjacent to the gate which is consistent with the logical function of the gate. Cubics are of two types; logic cubes and d-cubes. A logic cube defines the basic logical structure of the gate and does not have any bearing on fault propagation in the system. It is associated with a logical type rather than a particular gate and comprises simply a list of the input values and associated output values which result when the particular logical operator is applied to that particular set of inputs. Table 2.1 shows the logic cube for an "or" type gate and for a "1-bit adder" type gate.

	Inputs 1	Inputs 2	Outputs 3
1.	0	0	0
2.	0	1	1
3.	1	0	1
4.	1	1	1

An "or" cube

	Inputs 1	Inputs 2	Outputs 3	
1.	0	0	0	0
2.	0	1	0	1
3.	1	0	0	1
4.	1	1	1	0

A "1-bit adder" cube

Table 2.1: Two Logic Cubes

A logic cube completely describes the logic of a gate and has the additional property that if the signals for any subset of the inputs and outputs are known, then a list can be supplied of all remaining inputs and outputs which is consistent with the set of known signals. For example, if it is desired to have a signal of 0 on line 3 of the "or" gate given above, then the only set of signals for the input lines which is consistent with this signal is the single pair (0,0). If, however, a signal of 1 is desired on output line 3, then any of the three pairs (0, 1), (1, 0), and (1, 1) are consistent with this signal. What this means is that the circuit need not necessarily be processed in input to output order, and that when a gate is processed it may produce a list of possible signals rather than a unique vector. It will be this type of list which constitutes the decision list in the justification phase of the d-algorithm.

Faults and fault propagation add a new dimension to cubes and necessitate an extension of the logic cube to accommodate the presence of faults in the circuit. We begin by investigating the description of a fault pseudogate. Such a gate, as indicated in Subsection 1.2, requires two simultaneous descriptions, namely, the functioning of the pseudogate under normal operation and its functioning when a fault occurs. To facilitate such a simultaneous description, we introduce two new "signals," d and $\bar{d}$. A signal d assigned to a line means that when the fault is present the line will have the value 1, and when no fault is present the line will have the value 0. The signal $\bar{d}$ assigned to the line means precisely the opposite. The d and $\bar{d}$ signals represent the discrepancy which is necessary to distinguish the fault mode from the normal operation, and therefore it will be these types of signals which will need to propagate to the outputs.

Once a careful description is made for a fault pseudogate in terms of normal and fault mode operation, it is easy to translate this description into a cube description by using the symbols d and $\bar{d}$. This is called the d-cube for the fault. Table 2.2 shows the d-cubes for a stuck-at-1 fault and a stuck-at-0 fault.

Input	Output	Input	Output
0	d	0	0
1	1	1	$\bar{d}$
stuck-at 1		stuck-at-0	

Table 2.2: The stuck-at fault d-cubes

In the stuck-at-1 fault, an input signal of 0 will produce an output signal of 1 (stuck) when the fault occurs and 0 otherwise, and hence its output value will be represented by the symbol d. An input signal of 1 produces an output signal of 1 in either normal or fault mode. In the stuck-at-0 fault, it is the 1 input which causes the discrepancy in the output, and this is indicated by using the symbol $\bar{d}$ as the corresponding output. The cross-wire faults and other nonclassical faults are discussed in Section 3.

To propagate a fault through the circuit, we need lastly to modify the cubes for the normal gate types to allow them to transmit the discrepancy signals d and $\bar{d}$. This will be the d-cube for the gate type. The modification of the logic cube of the gate to a d-cube involves a simple argument for each case of the sort "if the fault occurs, then...," and "if the fault does not occur, then..." We give as an example the extension of the logic cube for the "or" gate shown in Table 2.3:

	Inputs 1	Inputs 2	Output 3
1.	0	0	0
2.	0	1	1
3.	1	0	1
4.	1	1	1
5.	0	d	d
6.	0	$\bar{d}$	$\bar{d}$
7.	1	d	1
8.	1	$\bar{d}$	1
9.	d	0	d
10.	d	1	1
11.	d	d	d
12.	d	$\bar{d}$	1
13.	$\bar{d}$	1	1
14.	$\bar{d}$	0	$\bar{d}$
15.	$\bar{d}$	d	1
16.	$\bar{d}$	$\bar{d}$	$\bar{d}$

Table 2.3: The "or" gate d-cube

Take, as a case, the row whose inputs are d and 0. We may argue as follows: "If the fault is present, then the first input will be 1 and the second input will be 0, so that the output is 1. If the fault is not present, then the first input will be 0 and the second input will be 0 so the output will be 0. Thus, when the fault exists, the output will be 1 and when the fault does not exist, the output will be 0. The output, therefore, is assigned the value d in this row." Of course, the complete description of the cube is fairly long, and we have made some effort to abbreviate this description in the coding of the d-algorithm (see Section 4). We note that for d-cubes, as for logic cubes, any set of values for a subset of input and output lines for a gate induces a sublist of the d-cube which is consistent with the assigned value. Thus, for example, an assignment of d to the first input of an "or" gate produces the list of possible assignments of the remaining inputs and outputs given in Table 2.4

	Inputs 1	Inputs 2	Output 3
9.	d	0	d
10.	d	1	1
11.	d	d	d
12.	d	$\bar{d}$	1

Table 2.4: A Restricted Cube for the "or" Gate

It will be this type of list which constitutes the decision list in the propagation mode.

2.3 Description of the d-Algorithm

We can now describe in a more precise fashion the general d-algorithm. We are given the circuit as described in Subsection 1.1, including as gates the fault pseudogates described in Subsection 1.2. We are also given d-cubes for the fault pseudogates and both logic and d-cubes for the standard gates in the circuit. The gates can technically be processed in any order, and some discussion on the merits of one ordering over another is undertaken in Section 4. Virtually every implementation of the d-algorithm, however, has the following restrictions on the ordering of gates:

1. All fault pseudogates and gates which are on a path originating from a fault pseudo-gate comprise the first set of gates to be processed. This is the propagation stage of the algorithm.

2. The remaining gates are then processed. This is the justification stage of the algorithm.

3. In the propagation stage, no gate is processed unless it is a fault pseudogate or until at least one gate (or pseudogate) immediately preceding that gate has been processed.

4. In the justification stage, no gate is processed until at least one gate (or pseudogate) immediately succeeding that gate has been processed.

Thus, gates are in general processed in forward order from the faults and then in backward order from the faults. This insures a sense of connectedness and direction to the problem. The general backtrack scheme, however, does not require such an ordering, and with modifications, the d-algorithm could accept gates in <u>any</u> conceivable order. This flexibility will be important to keep in mind when we deal with simultaneous faults.

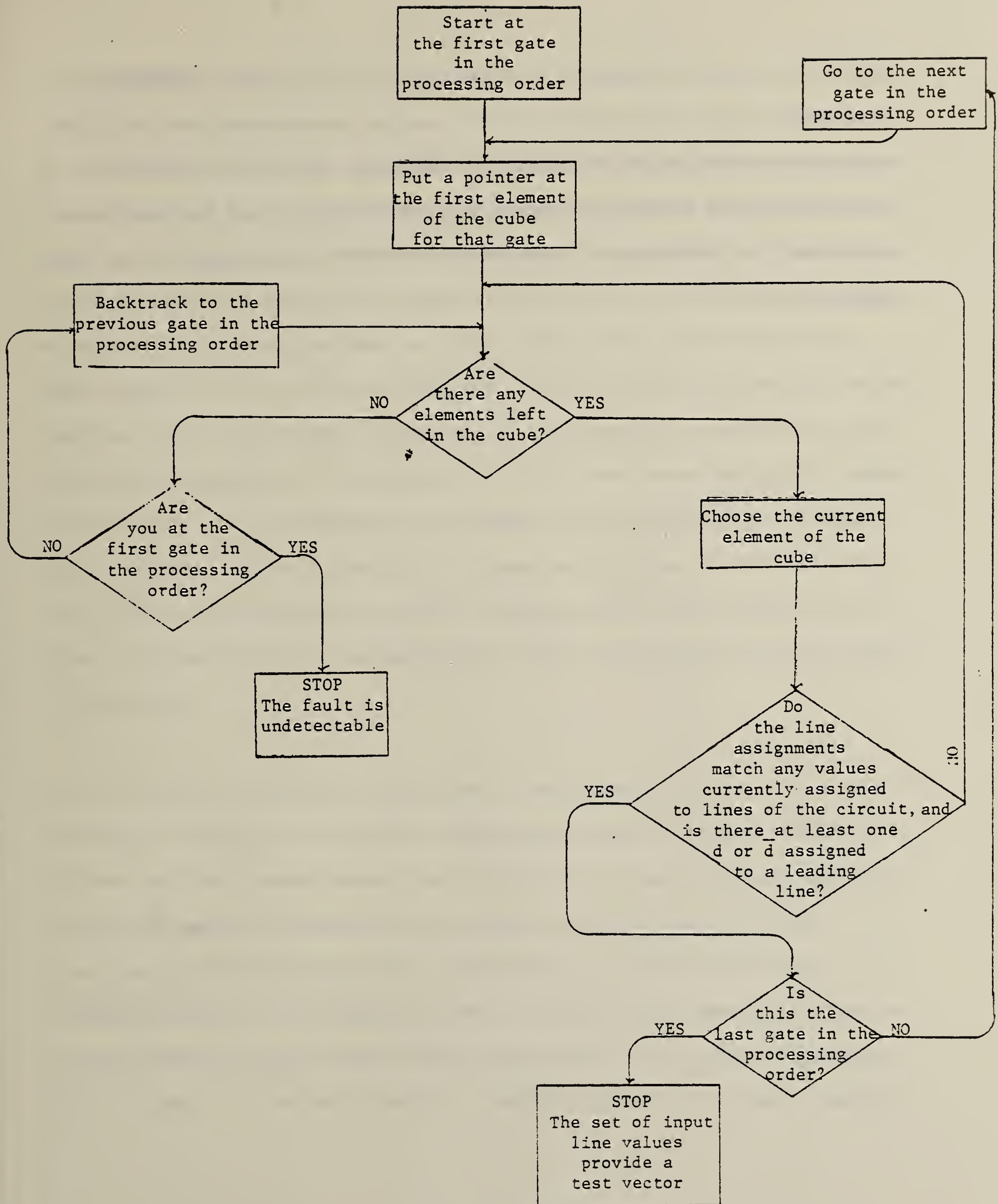
The "decision list" for a gate is simply the cube for that gate, or more precisely, the restriction of the cube to assignments consistent with previous values assigned to input or output lines of that gate. The gates processed in the propagation stage are assigned decision lists derived from d-cubes. Gates in the justification stage, however, are assigned lists from logic cubes, reflecting the fact that no fault signals occur on that part of the circuit. Consequently, output lines from a justification stage gate <u>must</u> be assigned a logical value, <u>even</u> if they are input lines to a propagation stage gate. The decision lists vary accordingly, both by the location of the gate and by the values which have been assigned to the adjacent lines up to that point in the algorithm.

The gates are processed in the order given and with the decision lists as described above, thus insuring that line values are consistent both with gate logic and with respect to adjacent gates. One further check must be made to insure that these values can actually detect the fault. This check is done in the propagation stage on <u>leading</u> lines —that is, an output lines from processed gates for which at least one successor gate to that line is either an output gate or an unprocessed gate. The rule which must be applied here is:

Propagation Rule: At least one leading line must have a value d or $\bar{d}$.

The d-algorithm proceeds in the format of the backtracking algorithm described in Subsection 2.1, checking, in the propagation stage, the propagation rule, until it either makes a consistent assignment to the final gate or backtracks through all of the gates without finding a consistent assignment. In the former case, the input gate assignments indicate the test vector to be applied, and the assignment to the output gates--which are the only remaining leading gates--insure that at least one d or $\bar{d}$ assignment has occurred, so that this vector actually detects the fault. In the latter case, it follows that every possible assignment has been tried subject to circuit consistency and fault detection, so that no possible set of input vectors could detect the fault. Figure 2.2 gives a flow chart for the d-algorithm.

The d-algorithm, then, is guaranteed either to find a vector which detects the fault or verify that no such test vector exists. In this sense, it is superior to the path sensitization algorithm, which may fail to find a test vector when one actually exists. The added power of the d-algorithm is due to the freedom allowed to gate assignments which is not available to the path sensitization algorithm, so that many lines with d or $\bar{d}$ values may be propagated simultaneously. It is also important to note the improvement in efficiency of the d-algorithm over a straightforward enumeration of input vectors. The backtrack algorithm, starting from the point of fault, insures that the <u>only</u> assignments made are those which could lead to fault detection. It may, therefore, backtrack before it even assigns input values and generally



<u>Figure 2.2</u>: Format for the d-algorithm

assigns only a small proportion of these values to an invalid test before establishing that it is invalid. It is therefore considerably more efficient than straight enumeration of inputs. By improving the order in which the gates are searched (within the restrictions given above), and the ease with which gates can be processed, we can make substantial improvement in the basic d-algorithm.

2.4 An Example

As an example to illustrate the functioning of the d-algorithm as presented in the subsection 2.3, we consider the circuit shown in Figure 2.3. The large gates are all "or" gates, whose logic cube was given in Table 2.1, and whose d-cube was given in Table 2.3. (Input lines will always be numbered from top to bottom. The triangular gates are "not" gates. They simply invert the input signal ($0 \to 1$, $d \to \bar{d}$) and hence will not be considered separately in the backtrack. The circles will contain the values assigned to the lines as the d-algorithm progresses. It is desired to find a test vector which will detect the stuck-at-0 fault represented by the square box in Figure 2.3, and whose fault cube was given in Table 2.2. The gates are numbered in the order they will appear in the backtrack, and this ordering satisfies the restrictions 1-5 given in Subsection 2.3. The cube elements will be scanned in the order given by the tables.

We now proceed to apply the d-algorithm. Since leading lines must have d or $\bar{d}$ assigned to them, gate 1 (the fault pseudo-gate) must be assigned element 2 of its cube. We then process gates 2 and 3 choosing for each the first element in its cube for which the associated line values are consistent with lines already assigned (elements 6 and 5, respectively). Gates 4 and 5 are processed similarly with elements 13 and 1 respectively. (Note that there is no requirement that gates 4 and 5 have outputs of d or $\bar{d}$, since the leading line from gate 3 alredy has value d.) Assigning element 9 to gate 6 completes the propagation stage. Figure 2.4 shows the assignments made thus far, with the number above each gate representing the cube element assigned to that gate.

For the justification stage, we assign gate 7 with its only consistent logical cube element 1, and then gate 8 has a unique consistent cube element 2. This gives the situation shown in Figure 2.5. But now gate 9 has no cube element which is consistent with its adjacent lines, and so the algorithm begins backtracking. Gates 8, 7, and 6 have no further consistent cube values, and gate 5 is now assigned element 2. Now gate 6 has inputs d and 1, and so cannot be assigned a cube element for which its output line -the sole leading line - has value d or $\bar{d}$. The algorithm backtracks to gate 5, for which there is no further element whose unassigned input line (to a justification stage gate) takes on a logical value. Thus the algorithm backtracks to gate 4 and gives it the next available consistent cube element 14. Gate 5 now goes through its cube once more, and the first available consistent element is 9. Gate 6 is now assigned element 11, and we arrive at Figure 2.6. We enter the justification mode once more, and again gates 7 and 8 have unique element assignments 1 and 2, respectively. Now, however, gate 9 has cube element 1 consistent with the assigned lines. Symmetrically, gates 10 and 11 are assigned cube elements 2 and 1, respectively. The algorithm now reaches the end of the gate proceeding order, and stops with all gates having consistent cube assignments as shown in Figure 2.7. The input vector (0,0,0,0) is therefore a test vector for the fault, as required. This completes the example.

The circuit given in this section was developed by P. R. Schneider [25], and has the property that no single path of d's and $\bar{d}$'s is sufficient to detect the fault given. The algorithm, as just used, in fact, tried all possible ways of "sensitizing" the single path through gates 1, 2, 3, and 6, and finally applied the cube elements which produced two such paths simultaneously to produce a test vector.

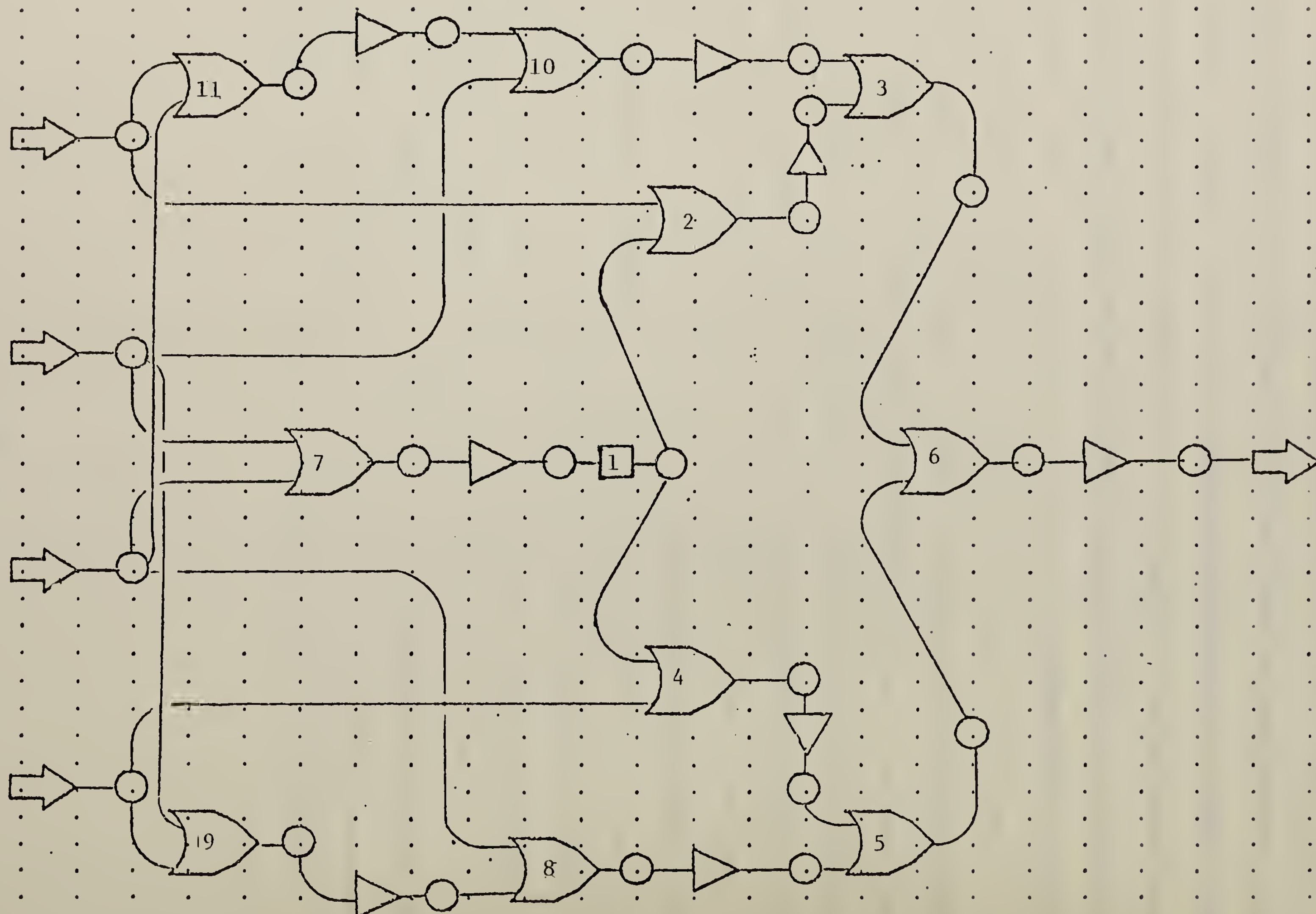


Figure 2.3: The Schneider example

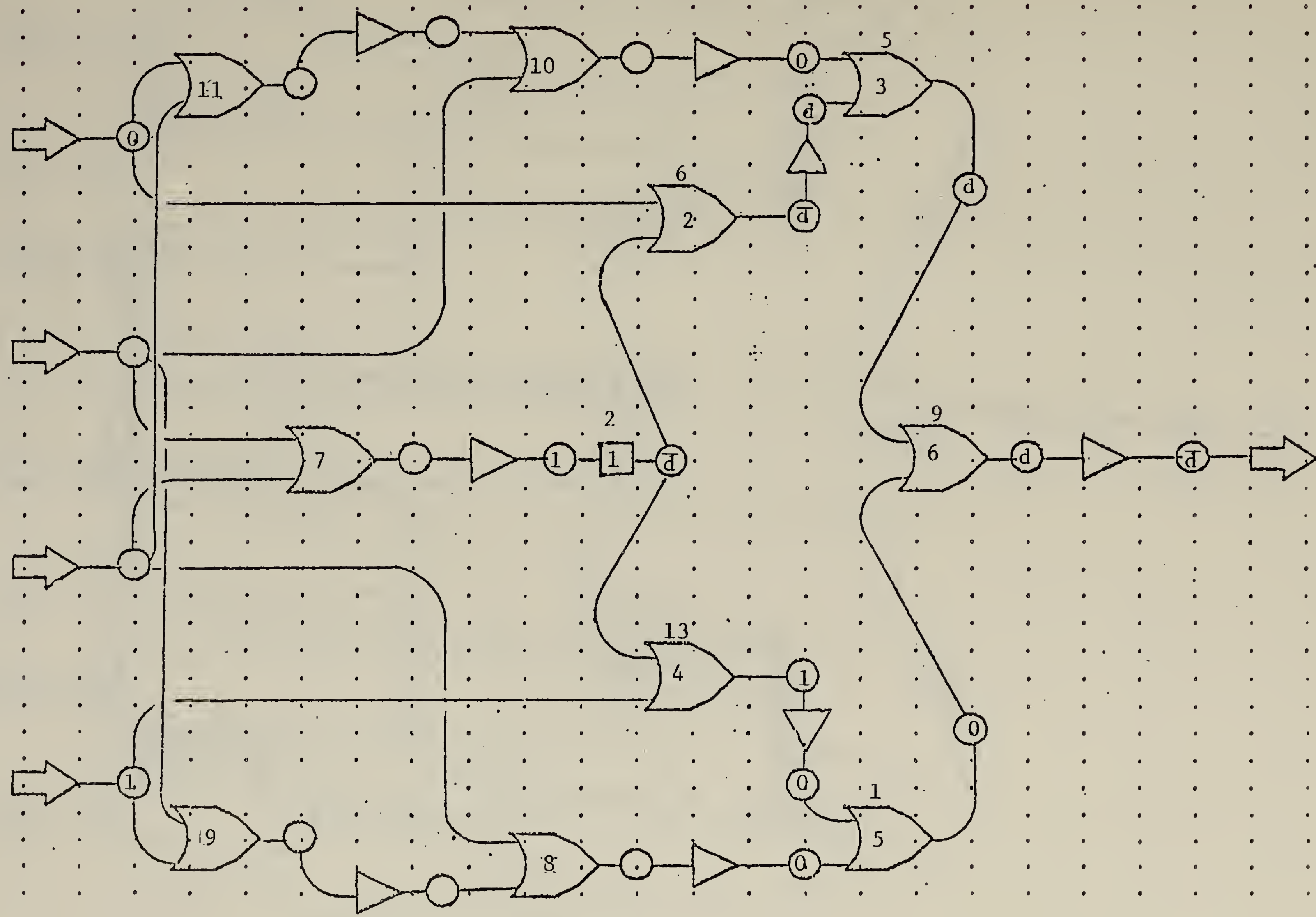


Figure 2.4

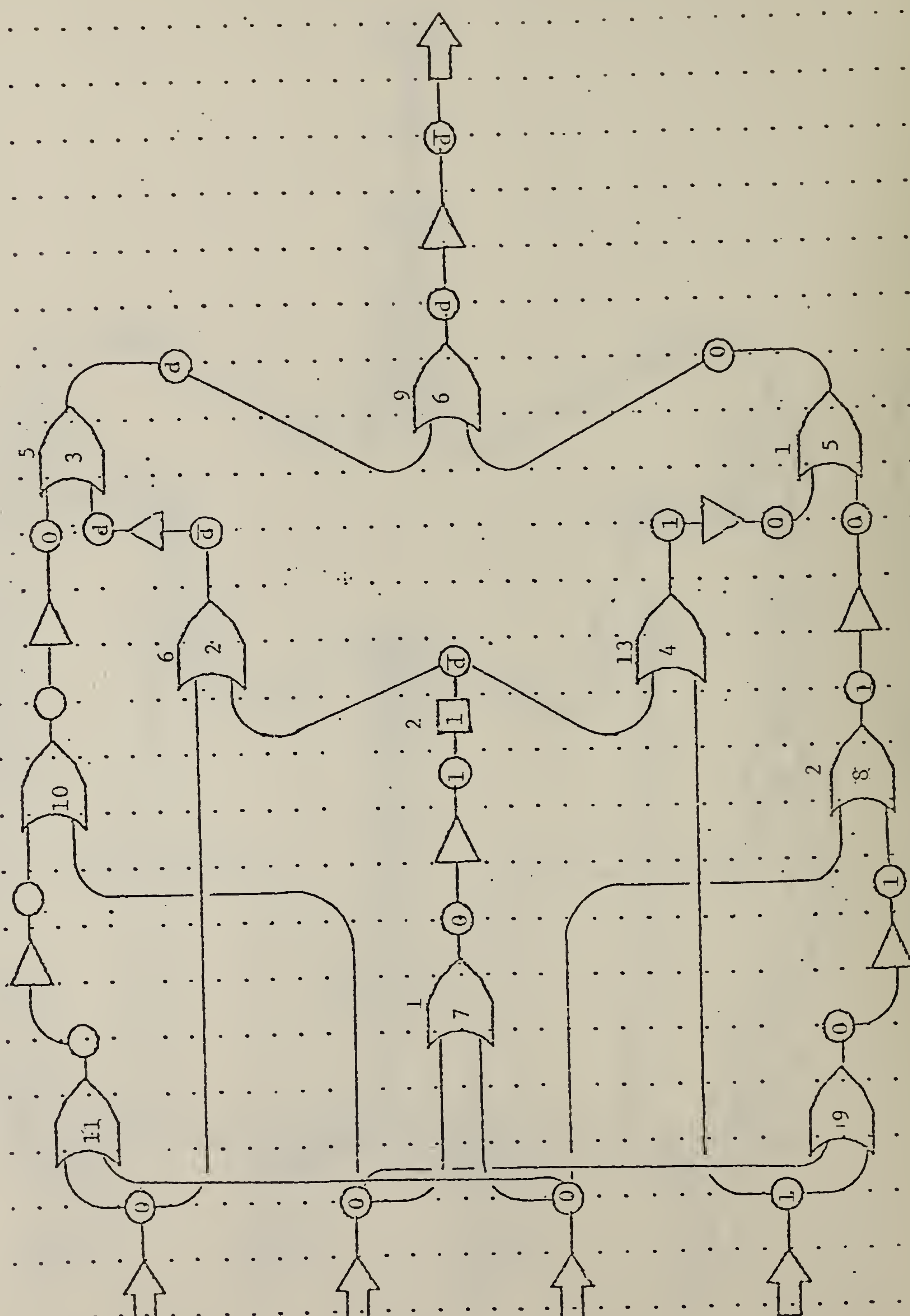


Figure 2:5

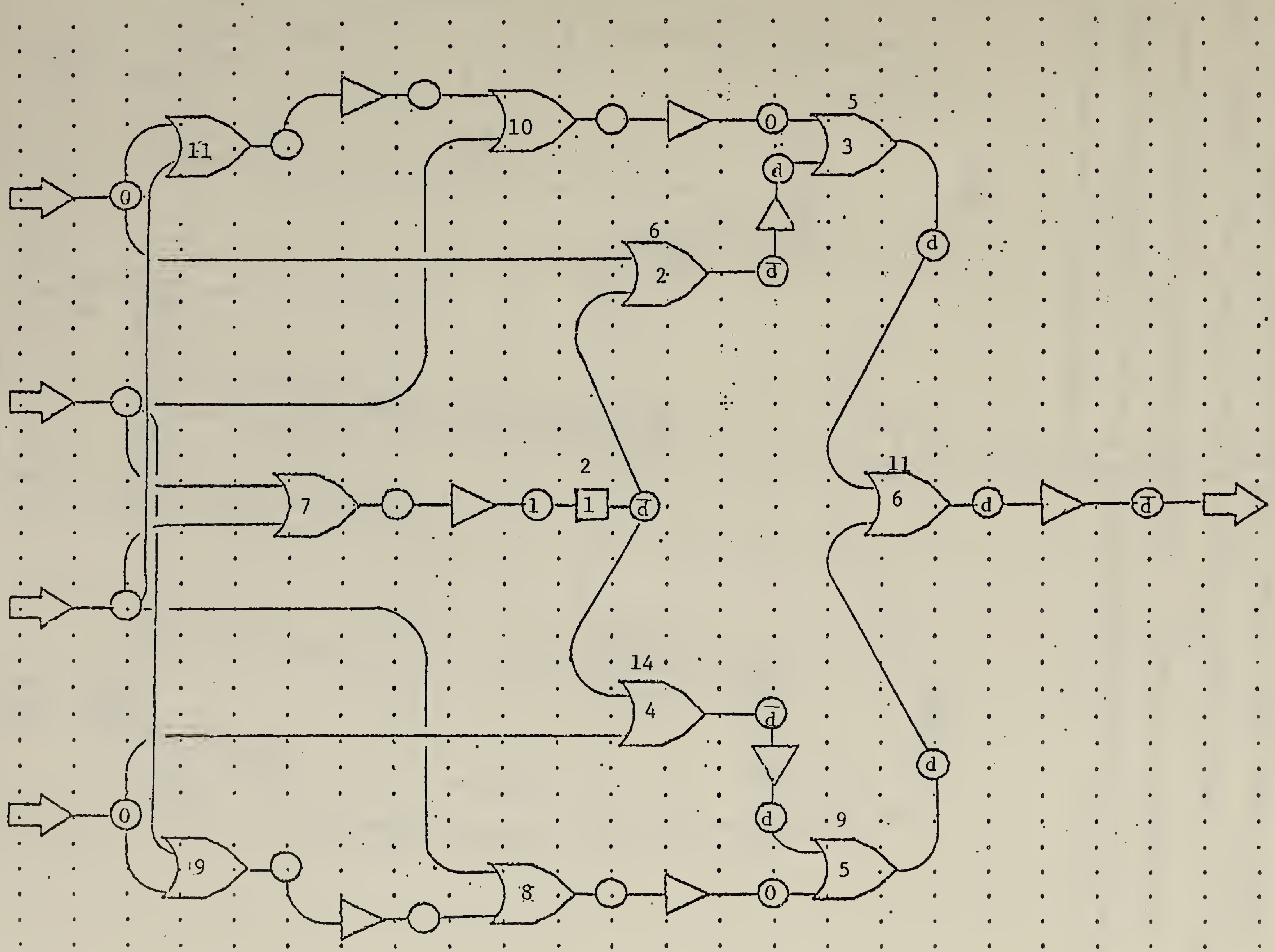


Figure 2.6

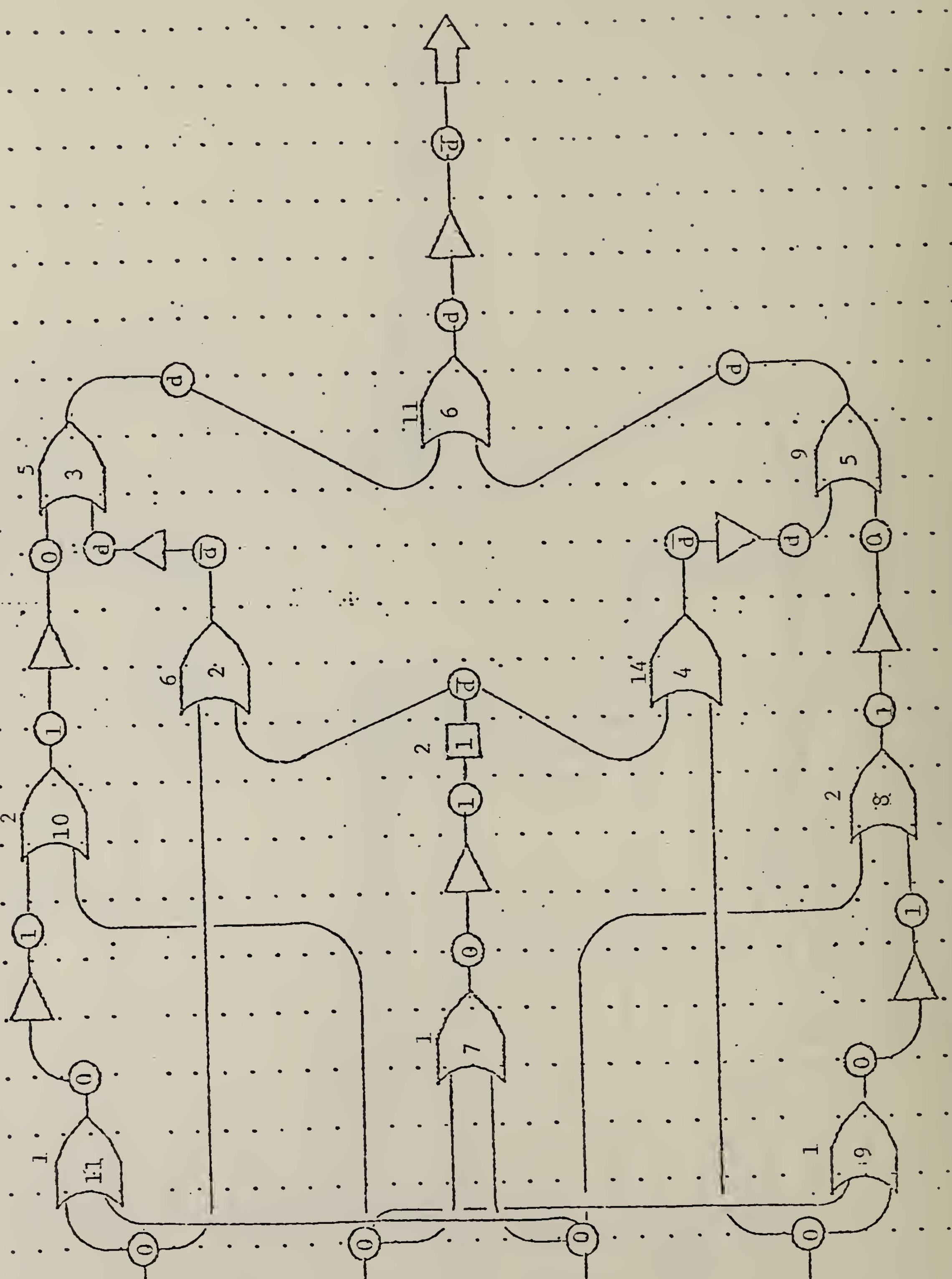


Figure 2.7

3. Generalizations of the d-Algorithm

We describe in this section some important ways in which the d-algorithm described in Section 2 can be generalized to handle a wider class of faults and circuits. Subsection 3.1 covers more general classes of faults, including cross-wire, inductive, and simultaneous faults. Subsection 3.2 covers more general cube descriptions, including abbreviated descriptions and empirical or operational descriptions of cubes.

3.1 Non-classical and Simultaneous Faults

We describe in this subsection three classes of faults — cross-wire faults, gate faults, and simultaneous faults — which are not treated by the standard d-algorithm, but which are of practical concern in circuit testing.

The standard cross-wire faults were described in Subsection 1.2. We can use that description to form cubes for cross-wire "and" and "or" gates as shown in Table 3.1:

Inputs		Outputs	
1	2	3	4
0	0	0	0
1	0	1	d
0	1	d	1
1	1	1	1

cross-wire "or"

Inputs		Outputs	
1	2	3	4
0	0	0	0
1	0	$\bar{d}$	0
0	1	0	$\bar{d}$
1	1	1	1

cross-wire "and"

Table 3.1: The cross-wire faults

A more general type of cross-wire fault frequently tested in circuits is the "inductive" type of fault, where one wire of a set of parallel wires changes signal if both of its neighbors are the opposite signal. A d-cube for a four wire fault is given in Table 3.2, listing only the non-trivial elements.

1--------------5

2--------------6

3--------------7

4--------------8

Inputs				Outputs			
1	2	3	4	5	6	7	8

Outputs equal corresponding Inputs

except for the following elements:

0	0	1	0	0	0	$\bar{d}$	0
0	1	0	0	0	$\bar{d}$	0	0
0	1	0	1	0	$\bar{d}$	d	1
1	0	1	0	1	d	$\bar{d}$	0
1	0	1	1	1	$\bar{d}$	1	1
1	1	0	1	1	1	$\bar{d}$	1

Figure 3.2: A four-wire "inductive" fault and its d-cube

Any number of other types of signal switching faults can be described using this same format.

A second type of fault which can be handled in the format of the generalized d-algorithm is the gate fault. This type of fault is useful when one is faced with circuits made up of complex gates whose faulty behavior may not be attributable to simpler faults. Here one must construct a fault cube for the entire gate which reflects the faulty behavior, and then substitute for the gate cube this fault cube. For example, suppose that the 1-bit adder described in Table 2.1 was known to have as one of its more frequent faults the inclination to add 1 to 1 and produce outputs (0,0). It may not be clear what in the internal circuitry of the adder has produced such a fault. It is nevertheless easy to describe the fault succinctly, as is done in Table 3.3.

Inputs		Outputs	
1	2	3	4
0	0	0	0
1	0	1	0
0	1	0	1
1	1	$\bar{d}$	0

Table 3.3: The cube for a faulty "1-bit adder"

By simply replacing this cube for that of the standard logic cube shown in Table 2.1 we can incorporate the fault into any "1-bit adder" type gate which might be suspected of having this type of fault.

The final type of fault is the simultaneous fault. This is mentioned frequently in the literature as a type of fault which is difficult to incorporate into test generation algorithms. In the context of a backtracking algorithm, however, simultaneous faults can be tested with only slight modification. A simultaneous fault is actually a set of faults which occur simultaneously at various locations of the circuit. It is important to distinguish this composite fault from the individual faults themselves since a test vector for a simultaneous fault distinguishes the case when all of these faults occur from the case where none of the faults occur. Simultaneous faults are likely to occur in equipment which is being used for the first time or which has been assembled by hand (properties frequently true of the prototype instruments developed at the Bureau of Standards), and should be available as a consideration when the circuit is malfunctioning and single fault tests fail to detect the fault.

Each single fault of the simultaneous set is furnished with the appropriate fault cube just as shown in Section 2 and this subsection. The d and $\bar{d}$ signals, moreover, represent this same situation for all of the fault gates, that is, under normal operation the lines assigned d have value 0 and those assigned $\bar{d}$ have value 1 for every fault gate in the simultaneous set, whereas in fault mode operation, the lines asssigned d have value 1 and those assigned $\bar{d}$ have value 0 for every fault gate in the set. With the possibility of faults now occurring in tandem, it is necessary to modify the fault cubes for any fault gate whose input might be affected by another fault in the simultaneous set. This is done by extending the fault cube to allow d and $\bar{d}$ inputs, and is again accomplished by a simple argument of the sort "if the faults occur,..." and "if the faults do not occur,...." We can modify, for example, the cross-wire "or" fault to allow d and d inputs as indicated in Table 3.4.

	Inputs		Outputs	
	1	2	3	4
1.	0	0	0	0
2.	0	1	d	1
3.	1	0	1	d
4.	1	1	1	1
5.	0	d	d	d
6.	0	$\bar{d}$	0	$\bar{d}$
7.	1	d	1	d
8.	1	$\bar{d}$	1	1
9.	d	0	d	d
10.	d	1	d	1
11.	d	d	d	d
12.	d	$\bar{d}$	d	1
13.	$\bar{d}$	0	$\bar{d}$	0
14.	$\bar{d}$	1	1	1
15.	$\bar{d}$	d	1	d
16.	$\bar{d}$	$\bar{d}$	$\bar{d}$	$\bar{d}$

Table 3.4: A full d-cube for the cross-wire "or" fault

Consider the element whose inputs are d and $\bar{d}$. Under normal operation the inputs would be (0, 1). These would be passed through normally to the outputs, which would therefore also be (0, 1). In the fault mode, however, the inputs would be (1,0), which would then be altered by the simultaneous cross-wire fault to the common outputs (1, 1). The output values for this element are therefore (d, 1), obtained by comparing the outputs obtained under each operating mode. Notice now that a partial cancellation has occurred, the cross-wire fault countermanding the previous fault. This effect is even more apparent in the cube element whose inputs are (1, $\bar{d}$), where a complete cancellation occurs in the outputs (1, 1). It is this kind of cancellation which causes single fault tests to fail in some instances to detect those faults in the presence of other faults, and indicates the need for simultaneous fault capabilities.

To handle simultaneous faults we must also be more careful in specifying gate ordering for the backtrack algorithm in the presence of simultaneous faults. In particular, we must insure that the first fault gates processed are those gates which have no fault gates preceding them in the circuit. Call these <u>minimal</u> fault gates. The ordering restriction (3) of Subsection 2.3 must now be modified to read

3'. In the propagation stage, no gate is processed unless it is a <u>minimal</u> fault pseudogate or until at least one gate (or pseudogate) immediately preceding that gate has been processed.

Now with the modification of the fault-cubes described above, all fault gates encountered later are processed as if they were normal gates. We must also modify the propagation rule of Subsection 2.3 to allow propagation from any fault gate. The modified rule is:

Propagation Rule': After all fault pseudogates are processed, at least one leading line must have the value d or $\bar{d}$.

With these modifications the d-algorithm can do a complete analysis of a simultaneous fault condition, and will always produce a test vector if one exists.

3.2 Finding Faults Detected by a Given Test

Up to now, we have been dealing primarily with finding a test vector to detect a given fault. The procedure given in the Subsection 3.1 provides an interesting and efficient method for doing the converse, namely, determining which faults are detected by a given test. In testing circuits, one is more often interested in how broad a class of faults can be detected by a set of test vectors rather than what test will detect a single fault. We give now a quick method which determines, given a set of input vector and a fault, whether the vector tests this particular fault. Since the procedure is so efficient, it can be used effectively to test a large set of faults in turn against a given input vector, and consequently to obtain a sense of the "fault coverage" of that input vector.

The procedure works as follows. For a given input vector and fault (single or simultaneous), associate a simultaneous fault which consists of the given fault, along with a set of input-set-to faults. These "faults" occur at the input gates, but their cube is comprised of a single element whose output value is the desired value of the input. Thus they do not contribute a d or $\bar{d}$ to the system, but instead merely set the input line to its appropriate value. Since the input gates are considered to be fault gates, however, the d-algorithm will always process these gates first. From this point on, the elements of the rest of the gates—including the fault pseudo-gates—are determined uniquely. Therefore in one pass through the gates (entirely in the propagation stage) the algorithm determines whether a d or $\overline{d}$ appears at an output, i.e., whether the given input values detect the given fault.

The fault testing procedure given above is easily implemented within the context of the d-algorithm as presented. It is part of the FORTRAN Code developed, and is illustrated in Section 5.

3.3 Generalized Cubes

There are three special techniques for constructing and modifying cubes which can improve both the applicability and the efficiency of the d-algorithm. The first technique is the use of extra symbols to denote "unassigned" line values. These allow, when possible, the delaying of assignments of values to certain non-critical lines until later in the backtrack, so that greater freedom can be exercised in choosing values when the lines become critical. The use of this technique in the d-algorithm can improve efficiency substantially. The second technique is the use of empirical rather then logical descriptions of gates to construct d-cubes for those gates. This allows not only the capability of describing circuits on a variety of levels, but also permits the limiting of gate operation to reflect the environment in which the gate or circuit is used which again yield an improvement in efficiency of the algorithm. The final technique is a fast cube search and element retrieval, based on a compressed data structure for storing the d-cube. This final improvement we leave until Section 4, since it is essentially an implementational rather then algorithmic feature.

As mentioned in Subsection 2.1, one of the most critical feature of a backtrack algorithm is the number of elements in each of the decision lists, since a complete backtrack sequence considers a number of decision configurations equal to the product of the length of the individual decision lists. One way to decrease the number of elements in a d-cube is to combine several elements into one. This is accomplished by using line values which represent a delayed non-assignment for that line. The simplest example of this, and one which we have incorporated into the code, is based on the fact that an "or" gate which has one of its inputs assigned the value 1 will always have its output assigned value 1 regardless of the other input. Thus if we assign a 1 to either of the inputs, we can process many of the succeeding gates without knowing immediately the value of the other input. We therefore give the other input an "unassigned" value, say -1, which is changed to a value of 0, 1, d, or $\bar{d}$ only when necessary later in the backtrack. This decreases the size of the d-cube for that "or" gate from sixteen elements to eleven as shown in Table 3.5:

Inputs		Outputs
-1	1	1
1	-1	1
0	0	0
0	d	d
0	$\bar{d}$	$\bar{d}$
d	0	d
d	d	d
d	$\bar{d}$	1
$\bar{d}$	0	$\bar{d}$
$\bar{d}$	d	1
$\bar{d}$	$\bar{d}$	$\bar{d}$

-1 = unassigned

Table 3.5: A reduced d-cube for the "or" gate

The −1 value assigned to the input line is reassigned when the output of the gate preceding that line is finally assigned. A further, and more elaborate, reduction can be made by assigning "partially unassigned" values to gates. These are assignments to a certain subset of inputs and outputs which are left unassigned, but for which an assignment of a value for one line will dictate the values assigned to every other line in that subset. Again, taking the "or" gate as an example, suppose a value of 0 is assigned to one of the inputs to that gate. Now the output will always have a value equal to that of the other input, regardless of what value is given to that input. We can therefore leave the values of these two lines unassigned with the restriction that they must be given equal values when they are finally assigned. The d−cube now needs only seven elements as shown in Table 3.6:

Inputs		Outputs
0	−1	−1
1	−1	1
−1	0	−1
−1	1	1
−1	−1	−1
d	$\bar{d}$	1
$\bar{d}$	d	1

All lines assigned −1 must be given the same value

Table 3.6: A further reduced d−cube for the "or" gate

We can also handle the case where a pair of lines must have opposite values. The "not" gate for example always has as its output the complement of the input, 1 and 0 being complements of each other, and d and $\overline{d}$ being complements of each other. We can therefore leave the input and output of this gate unassigned with the restriction that they must be given complementary values when they are finally assigned. We can use the symbols two symbols -1 and -2 to represent this condition in the d-cube, by requiring that each line with a -1 in the cube must be given the same value when they are assigned, and each line with -2 in the cube must be given the complementary value to those with -1. Using this convention, we reduce the "not" d-cube to exactly one element, as shown in Table 3.7:

Input	Output
-1	-2

Table 3.7: A reduced cube for the "not" gate

By careful manipulation of these complementary unassigned symbols we can make substantial reductions in the size of more complex gates. Table 3.8 gives complete d-cubes for the "or" and "1-bit adder" gates, both of which have only six elements.

Inputs 1	2	Outputs 3
0	-1	-1
1	-1	1
-1	0	-1
-1	1	1
-1	-1	-1
-1	-2	1

"or"

Inputs 1	2	Outputs 3	4
0	-1	0	-1
1	-1	-1	-2
-1	0	-1	0
-1	1	-1	-2
-1	-1	-1	0
-1	-2	0	1

"1-bit adder"

Table 3.8: Reduced cubes for the "or" and "1-bit adder" gates

The backtrack algorithm needs only to keep track of each set of unassigned lines whose elements must take the same value along with the complementary set whose elements must take the complementary value, merging sets if necessary when a common element occurs in both sets. It assigns all of the lines in a pair of complementary sets simultaneously whenever any one line is given a value, and backtracks when it detects inconsistencies occurring when two lines from the same set are assigned complementary values, or when lines in complementary sets are assigned the same value. Although the necessary structures to maintain this are fairly simple, we dispense with the details in this report, as it is not operational in the current version of the code.

The final part of this subsection will be spent describing the ways in which cubes can be constructed using empirical data on gate operation. Not only can this allow more general gate types to be developed, but it can also provide the user with a method of directing the search for test vectors by restricting gate operation according to his perception of the operation of the circuit.

Suppose, for example, it is desired to test a circuit which contains a 4-bit adder, whose eight inputs comprise two 4-bit numbers, and whose output is their 5-bit sum. A full non-reduced d-cube for this adder would require 4^8 = 65,536 elements. Although it may be feasible to store this many elements, it is unlikely that the d-algorithm would be practical on a circuit containing many 4-bit adder type gates. What a user of the d-algorithm needs to do in this situation is to review the environment in which the adder is used and choose a subset of elements which are most likely to occur in the operation of the circuit. Thus, he might conclude that numbers greater than 4 are unlikely to be processed by certain of these adders, or that the numbers processed by some adders are always in multiples of 4, thereby allowing him in either case to decrease cube size to 4^4 = 256 elements. He may have gates of which he does not have or does not desire to obtain complete information, and therefore may be reduced to describing only a limited operation of the cube. Again, simply by listing those elements for which he has some knowledge, or which are critical to circuit operation, he thereby obtains a limited but accurate operating description for that gate.

Restricted gate descriptions have the further property that they can be used to determine certain faults to be "non-critical" in the operational environment of a circuit. Consider, for example, the 1-bit adder whose gate configuration is shown in Figure 1.1 and for which it is desired to find a test vector to detect a stuck-at-1 fault in the line between the "or" gate and its succeeding "and" gate. If it is known that the first input gate of this circuit is always set to 1 when the circuit is in normal operation, then the stuck-at-fault given above is undetectable, since it can never be given a 0 input. Such a fault is therefore not necessary to test in order to declare the circuit fault free, since the situation under which the fault is critical never arises.

The use of restricted cubes, it can be seen, allows a potentially rewarding dialogue between the designer of a circuit and the tester of that circuit. With the designer's knowledge of the operating environment and characteristics of the circuit, and the tester's knowledge of the operation of the d-algorithm, a battery of tests can be built for a circuit which efficiently and effectively demonstrates both the design integrity and the operating capabilities of that circuit.

4. Implementation of the d-Algorithm

This section presents the techniques which can be employed in implementing the d-algorithm in order to improve its efficiency. Subsection 4.1 outlines the ordering schemes which were used to process gates in the backtrack algorithm, along with the advantages and disadvantages of each. Subsection 4.2 deals with a method of coalescing gates into components, allowing for efficient use of repeated gate groupings. Subsection 4.3 gives a specific data structure for cubes which improves cube searches and at the same time reduces storage requirements.

A FORTRAN code which incorporates many of the implementation ideas given in this Section, and has, as well, the capabilities of generalization given in the previous section. A listing of the code is available from the authors. The code is capable of handling general fault and gate cubes, as well as simultaneous faults. It can test an input to find detectable faults (Subsection 3.2), form components (Subsection 4.2) and run several orders of gate ordering for the backtrack. One has to note, however, that many of these capabilities involve changes or additions in the code itself, since they are too cumbersome and confusing to be effective as a user option. The specific user capabilities are outlined in the appendix. It is virtually impossible to use many of the features of the generalized d-algorithms, however, without a firm knowledge of the concepts underlying these generalizations, and many of these could not be included as standard specifications of the code. What was <u>not</u> incorporated in the present form of the code was any sequential capabilities, of the more elaborate reduced cube schemes (Subsection 3.3), and the compressed data structure (Subsection 4.3). These are left for future development.

4.1 Ordering Schemes for Gate Processing

The restrictions given in Section 2 for the order in which the d-algorithm processes gates still allows a great deal of flexibility in gate ordering. We give four schemes for gate processing which have different effects on the flow of the d-algorithm. They are called the depth-first ordering, the breadth-first ordering, the restricted depth-first ordering, and the restricted breadth-first ordering. They all follow the basic ordering rules given in Subsection 2.3, namely, processing first fault gates, then gates forward of fault gates, and finally gates behind fault gates, and never processing a gate until the appropriate adjacent gate is processed. To do this, a general search procedure requires the maintenance of two lists, the propagation elgibility list (PEL) and the justification elegibility list (JEL), which contain gates elegible in the respective phase according to the restrictions given in Subsection 2.3. Both lists are initially empty, and the general search procedure is as follows.

(1) Place the fault gate (or minimal fault gates) on PEL.

Propagation stage--while PEL is nonempty, perform the following:

(2) Choose a gate G from PEL and process that gate.

(3) Place into PEL all unprocessed gates that are on an output line of G and place into JEL all unprocessed gates that are on an input line of G.

(4) Mark G processed, and delete it from PEL.

When PEL is empty, the propagation stage is completed, and the justification stage begins.

Justification stage--while JEL is nonempty, perform (2'), (3'), and (4') on JEL:

2' Choose a gate G from JEL and process that gate.

3' Place into JEL all unprocessed gates that are on an input line of G.

4' Mark G processed and delete it from JEL.

When JEL is empty the justification stage is complete.

Any ordering which follows the above general format satisifies the basic requirements of gate ordering in the d-algorithm. The details of how to place and remove elements from PEL and JEL are what differentiate the four ordering schemes given above. The ordering schemes are:

Depth-first search - gates are inserted and removed from the top of the lists (LIFO list maintenance).

Breadth-first search - gates are inserted at the bottom of the lists and removed from the top of the lists (FIFO list maintenance).

Restricted breadth-first and depth-first search:

Propagation stage - The gate processed is the topmost gate in PEL for which all input lines coming from propagation stage gates have been assigned values.

Justification stage - The gate processed is the topmost gate in JEL for which all output lines going to justification phase gates have been assigned values.

Roughly speaking, depth-first search attempts to assign values to the outputs (or inputs) as fast as possible, whereas breadth first search keeps as broad a base as possible in which to choose new adjacent gates. Restricted search is a modification which holds processing of a gate until all lines on "one side" of the the gate are either assigned values or will never be assigned in that phase. Thus it could be considered a "local breadth-first search" rule.

Restricted searches also allow for a quicker search of the d-cube, since the data can often be keyed to input or to output values (see Subsection 3.3). It is a non-trivial fact that, in any circuit without feedback, there is always a way of ordering the gates so that a restricted search is possible.

The algorithm as coded is capable of using any of these four search procedures given. We have found thus far that the most efficient option is to use nonrestricted depth first search in the propagation stage and restricted breadth first search in the justification stage. The reasoning for this option is that in the propagation stage the most important goal is to drive a d or $\overline{d}$ to the outputs, hence to employ an unrestricted depth-first search, whereas in the justification stage the most important goal is to maintain logical consistency, hence to employ a restricted breadth first search. Some limited testing seems to bear out this option.

4.2 Component Formation

VLSI's today tend to be made up of components, that is, devices which are used as single units but which comprise internally a system of simpler logical gates, that is, "circuits-within-circuits." There may be many identical components in a circuit, and therefore some initial pre-processing of these components can save time when the algorithm must repeatedly process identical components. The preprocessing performed consists of producing a d-cube for the set of gates in the component so as to treat that component as a single gate. The algorithm then removes the gate system and replaces it by the newly created component/gate.

Creating the d-cube associated with a component involves a complete enumeration of all the possible input vectors and determining the output vector of the component for each of these input vectors. The method of determining the output values for a specific input vector is essentially that of applying the d-algorithm to a smaller circuit with input gates set to the appropriate values. Since cube construction requires only the forward propagation drive mechanism, the algorithm for determining logic cube values can be made faster than the general d-algorithm.

Restrictions on the use of components in place of gate systems are the following:

1. All gates in the component must be contiguous, that is must form a connected set of gates within the circuit.

2. The ordering of gates in a component must be consistent with all other components of the same type.

3. Faults may not be contained in any one of the listed components.

Because of restriction 3, separate consideration of a fault within an individual component necessitates removal of that component from the list, and treating it separately. The main advantage of component formation is the reduction in number of backtracks the algorithm must performs before locating a fault finding test vector. Having determined the internal logic of the component once, the algorithm is not required to re-discover the logic upon encountering each component. Component formation also illustrates the important "macro" capabilities available for the d-algorithm as generalized in this paper. Using component formation, together with the restricted cubes outlined in Subsection 3.2, cubes for large components can be constructed which enable the d-algorithm to find test-vectors more efficiently when large components are being used in quantity.

4.3 Data Structures for d-Cubes

The most expensive structure to maintain and process in the d-algorithm is the d-cube itself. For a reasonably complex gate the cube may be enormous, and the cost in both storing and retrieving data for the cube is a big consideration in constructing the code for the d-algorithm. One method which can save both storage and retrieval time is to "pack" the input and output data each into a single integer element. In particular, if we represent 0, 1, d, and $\bar{d}$, respectively, as 00, 01, 10, 11 (in binary), then the input and output values can each be represented by a single integer comprised of the concatenation of the individual values. The d-cube is then stored as a single-valued array, whose index number is the integer corresponding to the input values and whose value is the integer corresponding to the output values. As an example, the full d-cube for the cross-wire "or" fault can be derived directly from Table 3.4 and is shown in Table 4.1.

Input	Output	Input	Output	Input	Output	Input	Output
1. 0	0	3. 4	7	9. 8	10	13. 12	6
2. 1	13	4. 5	5	10. 9	9	14. 13	5
5. 2	10	7. 6	6	11. 10	10	15. 14	6
6. 3	3	8. 7	5	12. 11	9	16. 15	15

Table 4.1: Storing the d-cube for the cross-wire "or" fault

The inputs ($\bar{d}$, 1) for instance, correspond to the integer 1101 = 13, and the corresponding output integer 5 = 0101 corresponds to the outputs (1, 1).

If stored as the four line values indexed by the cube element number, the d-cube for this gate would take 16 x 4 = 64 words of storage. By using the "packed" scheme, we have reduced the storage to an array of only 16 elements. Furthermore, the time taken for a cube search can decrease dramatically when some of the inputs are already known. For example, if the gate given above, when processed had its second input already assigned d = 10, then the search indices reduce to $4i + 2$ for $i = 1, 2, 3, 4$, that is, 2, 6, 10, and 14. Rather than scanning each element of the cube until a match is found, the algorithm can now reference the above four indices directly, thus reducing the number of search calls four-fold.

For retrieval purposes, this packed format works best in the propagation stage, when one or more inputs have been assigned values. It is especially fast when restricted search is employed, since now most of the gates processed have <u>all</u> of their inputs assigned values, and as a result the packed array is only referenced once. One can think of reordering the cube so that cube elements can be retrieved by their output values instead of their input values. This would work particularly well in the justification stage, where one or more output values have been assigned values when the gate is processed. Unfortunately, an output value may correspond to more than one input value.

To employ the search technique described above, it would be necessary to group the inputs which correspond to a given output and have the output integer reference this list. The resulting data structures would be somewhat more cumbersome, but could still save time over a sequential cube search. The compressed data structure was not incorporated into the current code.

REFERENCES:

Journal articles:

[1] Abraham, J. A. and D. D. Gajski (1981), "Design of Testable Structures Defined by Simple Loops", IEEE Transactions on Computers C-39 pp. 875-883.

[2] Abramovici, M. (1982),"A Hierarchical, Path-Oriented Approach to Fault Diagnosis in Modular Combinatorial Circuits", IEEE Transactions on Computers, C-31, pp. 672-676.

[3] Abramovici, M. and M. Breuer (1980), "Multiple Fault Diagnosis in Combinatorial Circuits Based on Cause-Effect Analysis", IEEE Transactions on Computers C-26, pp. 451-460.

[4] Acken, C. (1981), "A Mathematical Model of Digital Systems", Proceedings of the 24th Midwest Symposium on Circuits and Systems.

[5] Agarwal, V. K. and A. S. F. Fung (1981), "Multiple Fault Testing of Large Circuits by Single Fault Test Sets", IEEE Transactions on Computers C-30, pp. 855-865.

[6] Armstrong, D. B., (1966), "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinatorial Logic Nets", IEEE Transactions on Electronic Computers EC-15, pp. 66-73.

[7] Bossen, D. C . and S. J. Hong, (1971), "Cause-Effect Analysis for Multiple Fault Detection in Combinatorial Networks, IEEE Transactions on Computers, Vol C-20, pp. 1252-57.

[8] Dandapani, R. and S. M. Reddy (1974), "On the Design of Logic Networks with Redundacy and Testability Considerations", IEEE Transactions on Computing C-23, pp. 1139-1149.

[9] Du, M-W. and C. D. Weiss (1973), "Multiple Fault Detection in Combinatorial Circuits: Algorithms and Computational Results, " IEEE Transactions on Computers C-22, pp. 235-240.

[10] Eldred, R. (1959), "Test Routines Based on Symbolic Logic Statements", Journal of the Association for Computing Machinery, Vol. 6, PP. 33-36.

[11] El-zig, Y. M. and S. Y. H. Su (1982), "Fault Diagnosis of MOS Combinatorial Networks, IEEE Transactions on Computers C-31, pp. 129-139.

[12] Fujiwara, H. and K. Kinoshita (1981), "A Design of Programmable Logic Arrays with Universal Tests, IEEE Transactions on Computers C-30, pp. 823-826.

[13] Goldstein, L. H., (1979), "Controllability/Observability Analysis of Digital Circuits", IEEE Transactions on Circuits and Systems, Vol. CAS-26, pp. 685-693.

[14] Gray, F. G. and J. F. Meyer, (1970), "Locatability of Faults in Combinatorial Networks, IEEE Transactions on Computers C-20, pp 1407-1412.

[15] Kime, C. R., (1979), "Sequential Test Generation", IBM Technical Disclosure Bulletin, N20, pp. 3332-3336.

[16] Leedy, T. F. (1979), Large Scale Integration Digital Testing -Annotated Bibliography 1969-1978, NBS Technical Note 1102, U.S. Department of Commerce/Bureau of Standards, Washington, D.C.

[17] Mallela, S. and G. M. Masson (1978), "Diagnosable Systems for Intermittent Faults, " IEEE Transactions on Computers C-27, pp. 560-566.

[18] Ostapko, D. L. and S. J. Hong, (1979), "Fault Analysis and Test Generation for Programmable Logic Arrays (PLA's)", IEEE Transactions on Computers C-28, pp. 617-626.

[19] Roth, J. P. (1966), "Diagnosis of Automata Failures", IBM Journal of Research and Development 10, pp. 278-291.

[20] Roth, J. P., (1978), "Improved Test-Generating D-algorithm", IBM Disclosure Bulletin, Vol. 20, pp. 392-294.

[21] Roth, J. P., (1978), "Sequential Test Generation", IBM Technical Disclosure Bulletin, N 20, Pp 3332-3336.

[22] Roth, J. P., W. G. Bourilions, and R. Schneider, (1972), "Programming Algorithms to Compute Tests to Detect and Distinguish Failures in Logic Circuits", IEEE Transactions on Computers C-21, pp. 404-471.

[23] Russell, J. D. and C. R. Kime, (1971), "Structural Factors in the Fault Diagnosis of Combinatorial Networks," IEEE Transactions on Computers C-20, pp. 1276-1285.

[24] Schertz, D. R. and G. Metze, (1971), "Locatability of Faults in Combinatorial Networks, IEEE Transactions on Computers C-20, pp. 1407-1412.

[25] Schneider, P. R. "On the Necessity to Examine D-chains in Diagnostic Test Generation - An Example", IBM Journal of Research and Development, Vol. 11, p. 114.

[26] Short, R. A. and J. Goldberg, (1971), "Soviet Progress in the Design of Fault-Tolerant Digital Machines", IEEE Transactions on Computers C-20, pp. 1337-1352.

[27] Suk, D. S. and S. M. Reddy (1980), Test Procedures for a Class of Pattern-Sensitive Faults in Semiconductor Random-Access Memories, IEEE Transactions on Computers C-29, pp. 410-429.

[28] Thatte, S. M. and J. A. Abraham, (1980), "Test Generation for Micro processors, IEEE Transactions on Computers C-29, pp. 429-441.

[29] Visvanathan, V., A. Sangiovanni-Vincentelli, and R. Sacks (1981), Diagnosability of Nonlinear Circuits and Systems", IEEE Transactions on Computers C-30, pp. 889-904.

[30] Williams, T. W. and K. P. Parker, (1982), "Design for Testability - A Survey", IEEE Transactions on Computers, Vol. C-31, p 2-15.

[31] Yam, S. S. and Y-S. Tang, (1971), "An Efficient Algorithm for Generating Complete Test Sets for Combinatorial Logic Circuits, IEEE Transactions on Computers C-20, pp. 1245-1251.

Books:

[32] Breuer, I. and A. P. Friedman (1976) Diagnosis and Reliable Design of Digital Systems, Computer Science Press, Potomac, Maryland.

[33] Chang, H. Y., E. Manning, and G. A. Metze (1970), Fault Diagnosis of Digital Systems, Wiley-Interscience, New York.

[34] Friedman, A. D. and P. R. Menon, (1971), Fault Detection in Digital Circuits, Prentice Hall, New Jersey.

[35] Roth, J. P. (1980), Computer Logic, Testing, and Verfication, Computer Science Press, Potomac, Maryland.

Journal issues devoted to fault testing and prevention:

[36] IEEE Transactions on Computers C-20, November 1971

[37] IEEE Transactions on Computers C-22, March 1973

[38] IEEE Transactions on Computers C-23, July 1974

[39] IEEE Transactions on Computers C-24, May 1975

[40] IEEE Transactions on Computers C-25, June 1976

[41] IEEE Transactions on Computers C-27, June 1978

[42] IEEE Transactions on Computers C-29, June 1980

[43] IEEE Transactions on Computers C-30, Nov. 1981

[44] IEEE Transactions on Computers C-31, July 1982

Conference Proceedings:

[45] Digest of the 1971 Symposium on Fault Tolerant Computing, Pasaidena, CA, IEEE Computer Society, Mar., 1971.

[46] Digest of the 1972 Symposium on Fault Tolerant Computing, Newton, MA, IEEE Computer Society, June, 1972.

[47] Digest of the 1973 Symposium on Fault Tolerant Computing, Palo Alto, CA, IEEE Computer Society, June, 1973.

[48] Digest of the 1974 Symposium on Fault Tolerant Computing, Urbana, IL, IEEE Computer Society, June, 1974.

[49] Digest of the 1975 Symposium on Fault Tolerant Computing, Paris, France, IEEE Computer Society, June, 1975.

[50] Digest of the 1976 Symposium on Fault Tolerant Computing, Pittsburgh, PA, IEEE Computer Society, June, 1976.

[51] Digest of the 1977 Symposium on Fault Tolerant Computing, Los Angeles, CA, IEEE Computer Society, June, 1977.

[52] Digest of the 1978 Symposium on Fault Tolerant Computing, Toulouse, France, IEEE Computer Society, June, 1978.

[53] Digest of the 1979 Symposium on Fault Tolerant Computing, Madison, WI, IEEE Computer Society, June, 1979.

[54] Digest of the 1980 Symposium on Fault Tolerant Computing, Kyoto, Japan, IEEE Computer Society, June, 1980.

[55] Digest of the 1981 Symposium on Fault Tolerant Computing, Santa Monica, Ca, IEEE Computer Society, June, 1981.

[56] Digest of the 1982 Symposium on Fault Tolerant Computing, Portland, ME, IEEE Computer Society, June, 1982.

Appendix: Code and Sample Output

The d-algorithm was implemented on the UNIVAC 1108 in ANSI FORTRAN77.* The capabilities of the compiled code are documented in the program text. The code processes nine gate types (AND, OR, NOT, NAND, NOR, exclusive OR as well as input gates, output gates, and dummy gates), four fault types (stuck-at and cross-wire faults) and the input setting feature (Subsection 3.2). It can form components from a given set of gates, creating a new cube and replacing the component gates with the component in the circuit. The user can specify the number of test vectors to be found for a given fault. Also included are two parameters specifying the minimum number of leading lines which are allowed to have the values d or $\bar{d}$ (called sensitized paths in the code). These allow the user to test a circuit just constructed by setting the inputs and observing the resulting outputs for correct logic, or to find which faults are detected by a particular test vector (Subsection 3.2). In any case it may be desired to have only logical circuit values, and so the minimum number of sensitized paths could be zero. Setting an upper limit on the number of sensitized paths permits the algorithm to act as a path sensitization algorithm (Subsection 1.3) or otherwise limits the extent to which faults are propogated through the system.

There are also several parameters in the program which can easily be reset to broaden the capabilities of the algorithm. The algorithm has a DEBUG flag, which when set allows the user to see each step of the backtrack. The number

---

* The sole exception was the use of in-line comments in the parameter and declaration statement.

and size of gates can be changed, and new gates can be inserted by developing the cube and putting it into the program in the format specified. The algorithm currently uses restricted depth first gate orderings in the propogation stage and restricted breadth-first ordering in the justification stage, but these can also be changed by reordering the appropriate indices.

The output streams for six test runs appear in this Appendix. These runs illustrate the narrow features of the code. They were all performed on the arithmetic logic unit circuit shown in Figure A.1. It is made up of NOT, AND, NAND, and NOR gates as indicated. The multi-input gates were decomposed into the appropriate 2-input gates. Then faults were treated in the runs, and they are indicated in the figure. Faults #1 and #2 are stuck-at-0 faults, and Fault #3 is a cross-wire AND fault. The first three runs found several tests each for Fault #1, using three levels of component formation: Run #1 had no components formed, Run #2 had the sets of circled gates in the figure coalesced into a component, and Run #3 had the set of circled and squared gates in the figure coalesced into components. The amount of time needed to form components is indicated (in milliseconds) at the top of each run. Component formation was limited primarily by the size of the d-cube which had to be produced, and larger component formation will probably have to be accompanied by a substantial paring of the d-cube by the user (Subsection 3.3). A very surprising phenomena is that the cube formation implementd in this algorithm actually <u>slows down</u> the backtrack, as indicated by the TIME FOR COMPUTATION which accompanies the output (also in milliseconds). This is probably due to the increase in time for searching the very large cubes produced, and also to the loss of control of the internal gates of the cube.

If component formation is to speed up the backtrack, then more sophisticated search mechanisms will have to be employed. Note that after the first test vector was found, finding additional test vectors is very inexpensive. Run #4 tested the input found in Run #1 against Fault #1. This shows the checking capabilities of the code (Subsection 3.2). Notice the time to check a test is in hundredths of a second, as opposed to several seconds to find a test.

Runs #5 and #6 show the simultaneous fault capabilities of the code. A run was first made for Fault #2 alone, and the test vector and resulting line values are shown in Figure A-2. When Fault #3 is also present the input vector shown will detect neither fault, for the cross-wire fault nullifies the stuck-at fault. When the program was run on the two faults simultaneously, however, a new input is produced which does in fact test the fault, and the resulting line values are shown in Figure A-3.

One final note about the testing of the code. The d-algorithm, due to its inherent intractibility, can perform very badly even on a circuit such as the one tested. An example is the seven-fold jump in computation time between the single and simultaneous faults in Runs #5 and #6. There were, moreover, faults tested on this circuit which would not yield test vectors in the CPU time alloted (usually 5 to 10 minutes). Thus it is difficult to make statements of the efficiency of a program such as this, and also indicates that substantial improvements can be made in running times by adding good heuristics to the algorithm.

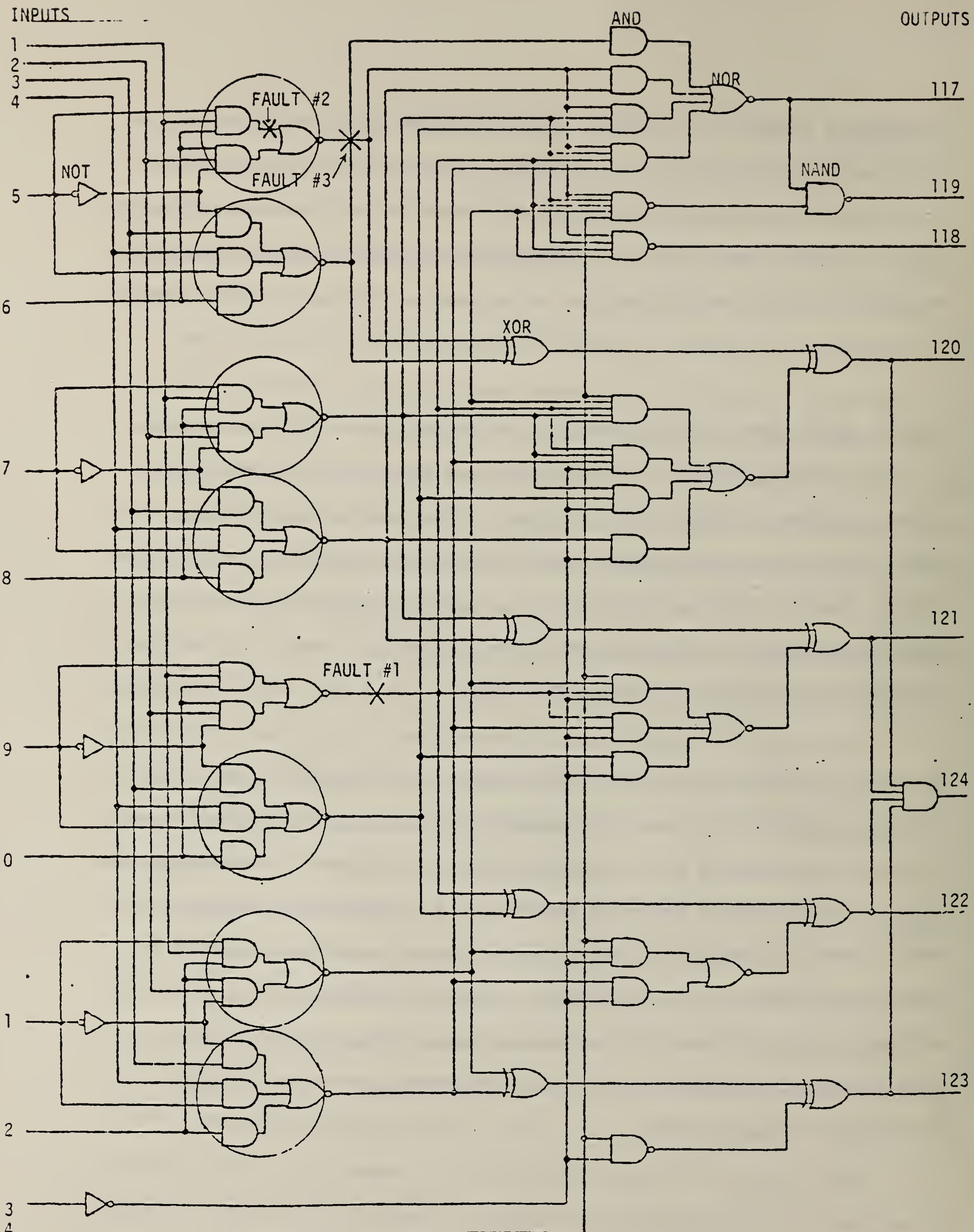


Figure A.1: Test circuit (from The TTL Data Book, Texas Instruments, 1973, p. 390)

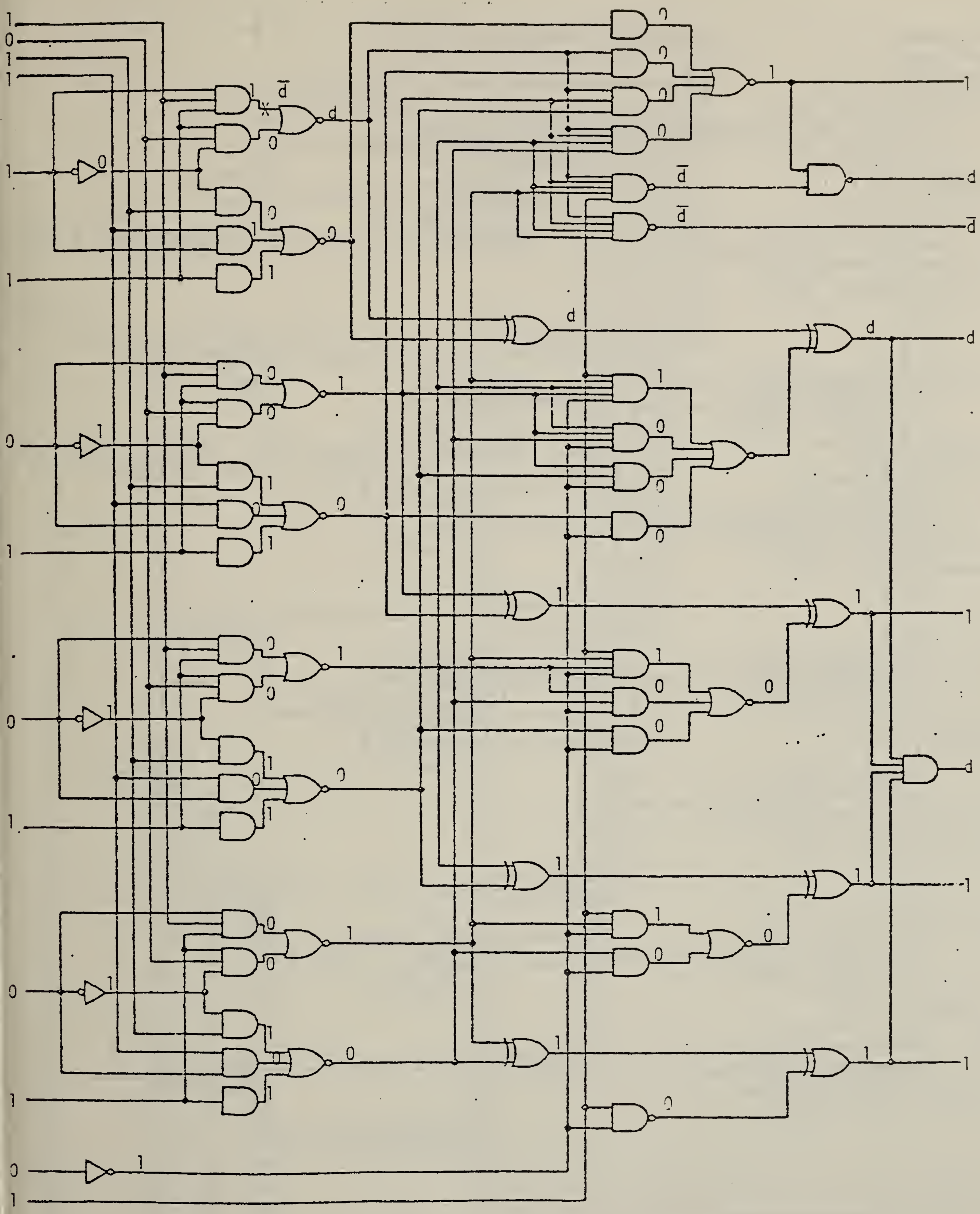


Figure A.2: Single fault test vector

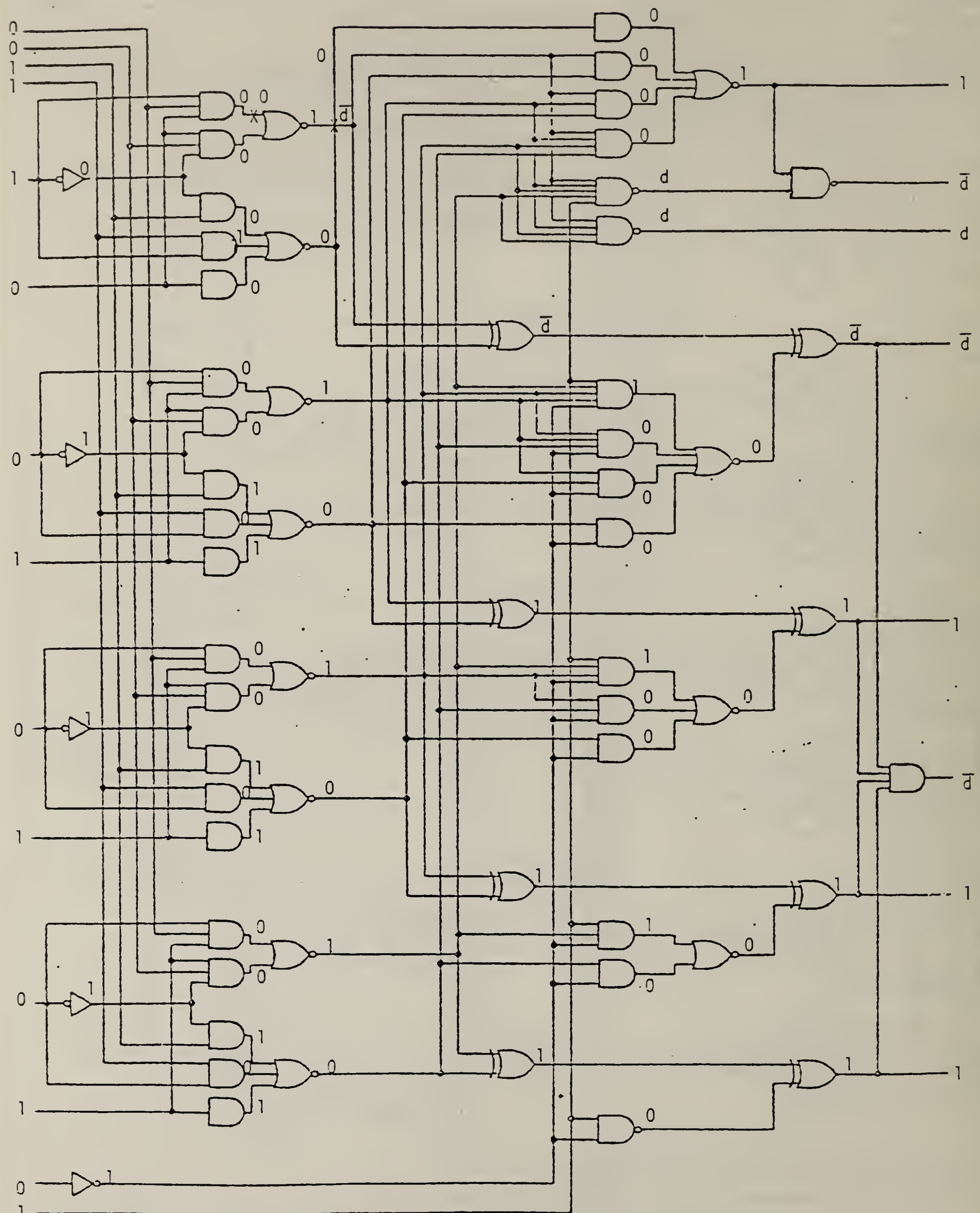


Figure A.3: Multiple fault test vector

Problem #1

JOUT

TIME USED TO FORM COMPONENTS IS 0

FAULT RUN NUMBER 1

FAULTS

GATE	TYPE OF FAULT
125	LINE 3 OF GATE 44 STUCK-AT-0

GATE ASSIGNMENTS

GT	TP	ADJACENT PINS		
1	1	1		
2	1	2		
3	1	3		
4	1	4		
5	1	5		
6	1	6		
7	1	7		
8	1	8		
9	1	9		
10	1	10		
11	1	11		
12	1	12		
13	1	13		
14	1	14		
15	4	5	15	
16	4	7	16	
17	4	9	17	
18	4	11	18	
19	4	13	19	
20	5	1	5	20
21	5	6	20	21
22	5	6	2	22
23	5	15	22	23
24	8	21	23	24
25	5	3	15	25
26	5	4	5	26
27	3	6	27	
28	6	25	26	28
29	8	27	28	29
30	5	7	1	30
31	5	8	30	31
32	5	8	2	32
33	5	16	32	33
34	8	31	33	34
35	5	16	3	35
36	5	4	7	36
37	3	8	37	
38	6	35	36	38
39	8	37	38	39
40	5	9	1	40

JOUT

41	5	10	40	41
42	5	10	2	42
43	5	17	42	43
44	8	41	43	117
45	5	17	3	45
46	5	4	9	46
47	3	10	47	
48	6	45	46	48
49	8	47	48	49
50	5	11	1	50
51	5	12	50	51
52	5	12	2	52
53	5	18	52	53
54	8	51	53	54
55	5	18	3	55
56	5	4	11	56
57	3	12	57	
58	6	55	56	58
59	8	57	58	59
60	3	29	60	
61	5	24	39	61
62	5	24	34	62
63	5	49	62	63
64	5	24	34	64
65	5	44	64	65
66	5	59	65	66
67	6	60	61	67
68	6	63	67	68
69	8	66	68	69
70	5	24	34	70
71	5	44	70	71
72	5	54	71	72
73	7	14	72	73
74	5	24	34	74
75	5	44	74	75
76	7	54	75	76
77	11	24	29	77
78	5	8	54	78
79	5	44	78	79
80	5	34	79	80
81	5	19	80	81
82	5	44	34	82
83	5	59	82	83
84	5	19	83	84
85	5	34	49	85
86	5	19	85	86
87	5	19	39	87
88	6	81	84	88
89	6	86	88	89
90	8	87	89	90
91	11	34	39	91
92	5	14	54	92
93	5	44	92	93
94	5	19	93	94
95	5	44	59	95
96	5	19	95	96
97	5	49	19	97
98	6	94	96	98

JOUT

99	8	97	98	99
100	11	44	49	100
101	5	14	54	101
102	5	19	101	102
103	5	59	19	103
104	8	102	103	104
105	11	54	59	105
106	7	14	19	106
107	4	69	107	
108	4	73	108	
109	6	107	108	109
110	11	77	90	110
111	11	91	99	111
112	11	100	104	112
113	11	105	106	113
114	5	110	111	114
115	5	112	114	115
116	5	113	115	116
117	2	69		
118	2	76		
119	2	109		
120	2	110		
121	2	111		
122	2	112		
123	2	113		
124	2	116		
125	22	117	44	

PIN ASSIGNMENTS

PIN	ADJACENT	GATES									
1	1	20	30	40	50						
2	2	22	32	42	52						
3	3	25	35	45	55						
4	4	26	36	46	56						
5	5	15	20	26							
6	6	21	22	27							
7	7	16	30	36							
8	8	31	32	37	78						
9	9	17	40	46							
10	10	41	42	47							
11	11	18	50	56							
12	12	51	52	57							
13	13	19									
14	14	73	92	101	106						
15	15	23	25								
16	16	33	35								
17	17	43	45								
18	18	53	55								
19	19	81	84	86	87	94	96	97	102	103	106
20	20	21									
21	21	24									
22	22	23									
23	23	24									
24	24	61	62	64	70	74	77				
25	25	28									
26	26	28									
27	27	29									

JOUT

28	28	29							
29	29	60	77						
30	30	31							
31	31	34							
32	32	33							
33	33	34							
34	34	62	64	70	74	80	82	85	91
35	35	38							
36	36	38							
37	37	39							
38	38	39							
39	39	61	87	91					
40	40	41							
41	41	44							
42	42	43							
43	43	44							
44	125	65	71	75	79	82	93	95	100
45	45	48							
46	46	48							
47	47	49							
48	48	49							
49	49	63	85	97	100				
50	50	51							
51	51	54							
52	52	53							
53	53	54							
54	54	72	76	78	92	101	105		
55	55	58							
56	56	58							
57	57	59							
58	58	59							
59	59	66	83	95	103	105			
60	60	67							
61	61	67							
62	62	63							
63	63	68							
64	64	65							
65	65	66							
66	66	69							
67	67	68							
68	68	69							
69	69	107	117						
70	70	71							
71	71	72							
72	72	73							
73	73	108							
74	74	75							
75	75	76							
76	76	118							
77	77	110							
78	78	79							
79	79	80							
80	80	81							
81	81	88							
82	82	83							
83	83	84							
84	84	88							
85	85	86							

```
 JOUT
  86  86  89
  87  87  90
  88  88  89
  89  89  90
  90  90 110
  91  91 111
  92  92  93
  93  93  94
  94  94  98
  95  95  96
  96  96  98
  97  97  99
  98  98  99
  99  99 111
 100 100 112
 101 101 102
 102 102 104
 103 103 104
 104 104 112
 105 105 113
 106 106 113
 107 107 109
 108 108 109
 109 109 119
 110 110 114 120
 111 111 114 121
 112 112 115 122
 113 113 116 123
 114 114 115
 115 115 116
 116 116 124
 117  44 125

LEVELLING
FLEVEL =                1 PLEVEL =              41 TOPLEV =             124
SCANNING SEQUENCE
 125 100 112 122  95  96  93  94  98  99 111 121  82  83  84  79  80  81  88  89
  90 110 120 114 115 116 124  75  76 118  71  72  73 108  65  66  69 117 107 109
 119  44 104  92  97  91  78  86  87  77 113  74  70  64  68  41  43 102 103  85
 105  59  57  58 106  19  13  63  49  47  48  67  40  42  10  45  17  46 101  54
  14  51  53  55  56  50  18  52  12  62  34  31  33  30  32  60  29  27  28  61
  39  24  37   8  38  21  23   9  11  35  16  36   7  20   1  22   2   6  25   3
  15  26   4   5

LEADFL
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0  35   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  36  37   0   0  39   0  32  33  34   0  29   0   0   0  17  18
  19  14  15  19   0   0   0  20  21  22   0   0   8   9   6   9   0  10  11   3
```

```
 JOUT

   0    0    0    0    0    0   40   40    0   24   24   25    0   25   26    0    0

NUMBER OF TESTS =                  2

                       TIME FOR VECTOR COMPUTATION IS         13285
 NL =   21571

TEST NUMBER              1

PIN VALUES
 PIN        1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20
           22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38   39   40   41
           43   44   45   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60   61   62
           64   65   66   67   68   69   70   71   72   73   74   75   76   77   78   79   80   81   82   83
           85   86   87   88   89   90   91   92   93   94   95   96   97   98   99  100  101  102  103  104
          106  107  108  109  110  111  112  113  114  115  116  117
 VALUE      0    0    0    1    1    1    0    0    1    0    0    0    0    1    0    1    0    1    1    0
            0    0    1    0    1    1    1    0    0    0    0    0    1    0    0    0    0    1    0    0
            0    3    0    1    0    1    0    0    0    0    0    1    0    0    0    0    1    0    1    1
            1    3    3    1    1    0    1    3    3    2    1    3    2    1    0    0    0    0    3    3
            0    0    1    3    3    0    0    1    3    3    3    3    0    3    2    3    1    1    1    0
            0    1    3    1    1    2    3    0    2    0    0    1

PRINCIPAL INPUTS
GATE VALUE
   1    0
   2    0
   3    0
   4    1
   5    1
   6    1
   7    0
   8    0
   9    1
  10    0
  11    0
  12    0
  13    0
  14    1

PRINCIPAL OUTPUTS
GATE VALUE
 117    0
 118    2
 119    1
 120    1
 121    2
 122    3
 123    0
 124    0

                       TIME FOR VECTOR COMPUTATION IS              3
 NL =   21579
```

JOUT

TEST NUMBER 2

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	1	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	1	1	0
	1	0	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0
	0	3	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1
	1	3	3	1	1	0	1	3	3	2	1	3	2	1	0	0	0	0	3	3
	0	0	1	3	3	0	0	1	3	3	3	3	0	3	2	3	1	1	1	0
	0	1	3	1	1	2	3	0	2	0	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	1
3	0
4	1
5	1
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1
120	1
121	2
122	3
123	0
124	0

END OF RUN

END ONSITE PRINTOUT ON JULY 21, 1982 AT 08:47:22
VLSI*JOUT(1).

```
   ACCOUNT: 33232-PRZYBO     TRAIN: A     REEL: 37        TRACK: 7     PUNCH: N     LINES/INCH:
SOF  CONTROL WORD, OCTAL - 500130000000, I =  1, FT = S, P =  0, CT = FIELDATA
ABOVE CONTROL WORD IN BLOCK        1 LOOKS WRONG      ????????????????????????????????????????
*SCFF*                                                ------------------------------------------


                        TIME USED TO FORM COMPONENTS IS       7533



 FAULT RUN NUMBER                2


 FAULTS

         GATE    TYPE OF FAULT
          132    LINE               3 OF GATE              44  STUCK-AT-0


 GATE ASSIGNMENTS
  GT TP ADJACENT PINS
    1   1   1
    2   1   2
    3   1   3
    4   1   4
    5   1   5
    6   1   6
    7   1   7
    8   1   8
    9   1   9
   10   1  10
   11   1  11
   12   1  12
   13   1  13
   14   1  14
   15   4   5  15
   16   4   7  16
   17   4   9  17
   18   4  11  18
   19   4  13  19
   20   5   1   5  20
   21   5   6  20  21
   22   5   6   2  22
   23   5  15  22  23
   24   8  21  23  24
   25   5   3  15  25
   26   5   4   5  26
   27   3   6  27
   28   6  25  26  28
   29   8  27  28  29
   30   5   7   1  30
   31   5   8  30  31
   32   5   9   2  32
   33   5  15  32  33
   34   8  31  33  34
   35   5  16   3  35
```

36	5	4	7	36
37	3	8	37	
38	6	35	36	38
39	8	37	38	39
40	5	9	1	40
41	5	10	40	41
42	5	10	2	42
43	5	17	42	43
44	8	41	43	117
45	5	17	3	45
46	5	4	9	46
47	3	10	47	
48	6	45	46	48
49	8	47	48	49
50	5	11	1	50
51	5	12	50	51
52	5	12	2	52
53	5	18	52	53
54	8	51	53	54
55	5	18	3	55
56	5	4	11	56
57	3	12	57	
58	6	55	56	58
59	8	57	58	59
60	3	29	60	
61	5	24	39	61
62	5	24	34	62
63	5	49	62	63
64	5	24	34	64
65	5	44	64	65
66	5	59	65	66
67	6	60	61	67
68	6	63	67	68
69	8	66	68	69
70	5	24	34	70
71	5	44	70	71
72	5	54	71	72
73	7	14	72	73
74	5	24	34	74
75	5	44	74	75
76	7	54	75	76
77	11	24	29	77
78	5	8	54	78
79	5	44	78	79
80	5	34	79	80
81	5	19	80	81
82	5	44	34	82
83	5	59	82	83
84	5	19	83	84
85	5	34	49	85
86	5	19	85	86
87	5	19	39	87
88	6	81	64	88
89	6	86	88	89
90	8	87	89	90
91	11	34	39	91
92	5	14	54	92
93	5	44	92	93

94	5	19	93	94			
95	5	44	59	95			
96	5	19	95	96			
97	5	49	19	97			
98	6	94	96	98			
99	6	97	98	99			
100	11	44	49	100			
101	5	14	54	101			
102	5	19	101	102			
103	5	59	19	103			
104	8	102	103	104			
105	11	54	59	105			
106	7	14	19	106			
107	4	69	107				
108	4	73	108				
109	6	107	108	109			
110	11	77	90	110			
111	11	91	99	111			
112	11	100	104	112			
113	11	105	106	113			
114	5	110	111	114			
115	5	112	114	115			
116	5	113	115	116			
117	2	69					
118	2	76					
119	2	109					
120	2	110					
121	2	111					
122	2	112					
123	2	113					
124	2	116					
125	12	1	2	5	6	15	24
126	12	1	2	7	8	16	34
127	12	1	2	11	12	18	54
128	13	3	4	5	6	15	29
129	13	3	4	7	8	16	39
130	13	3	4	9	10	17	49
131	13	3	4	11	12	18	59
132	22	117	44				

PIN ASSIGNMENTS

PIN	ADJACENT	GATES			
1	1	125	126	40	127
2	2	125	126	42	127
3	3	128	129	130	131
4	4	128	129	130	131
5	5	15	125	128	
6	6	125	128		
7	7	16	126	129	
8	8	126	129	78	
9	9	17	40	130	
10	10	41	42	130	
11	11	18	127	131	
12	12	127	131		
13	13	19			
14	14	73	92	101	106
15	15	125	128		

16	16	126	129								
17	17	43	130								
18	18	127	131								
19	19	81	84	86	87	94	96	97	102	103	106
20	20	21									
21	21	24									
22	22	23									
23	23	24									
24	125	61	62	64	70	74	77				
25	25	26									
26	26	28									
27	27	29									
28	28	29									
29	128	60	77								
30	30	31									
31	31	34									
32	32	33									
33	33	34									
34	126	62	64	70	74	80	82	85	91		
35	35	36									
36	36	38									
37	37	39									
38	38	39									
39	129	61	87	91							
40	40	41									
41	41	44									
42	42	43									
43	43	44									
44	132	65	71	75	79	82	93	95	100		
45	45	46									
46	46	48									
47	47	49									
48	48	49									
49	130	63	65	97	100						
50	50	51									
51	51	54									
52	52	53									
53	53	54									
54	127	72	76	78	92	101	105				
55	55	56									
56	56	58									
57	57	59									
58	58	59									
59	131	66	83	95	103	105					
60	60	67									
61	61	67									
62	62	63									
63	63	68									
64	64	65									
65	65	66									
66	66	69									
67	67	68									
68	68	69									
69	69	107	117								
70	70	71									
71	71	72									
72	72	73									
73	73	108									

74	74	75	
75	75	76	
76	76	118	
77	77	110	
78	78	79	
79	79	80	
80	80	81	
81	81	88	
82	82	83	
83	83	84	
84	84	88	
85	85	86	
86	86	89	
87	87	90	
88	88	89	
89	89	90	
90	90	110	
91	91	111	
92	92	93	
93	93	94	
94	94	98	
95	95	96	
96	96	98	
97	97	99	
98	98	99	
99	99	111	
100	100	112	
101	101	102	
102	102	104	
103	103	104	
104	104	112	
105	105	113	
106	106	113	
107	107	109	
108	108	109	
109	109	119	
110	110	114	120
111	111	114	121
112	112	115	122
113	113	116	123
114	114	115	
115	115	116	
116	116	124	
117	44	132	

LEVELLING
FLEVEL = 1 PLEVEL = 41 TOPLEV = 96
SCANNING SEQUENCE

132	100	112	122	95	96	93	94	98	99	111	121	82	83	84	79	80	81	88	89
90	110	120	114	115	116	124	75	76	118	71	72	73	108	65	66	69	117	107	109
119	44	104	92	97	91	78	86	87	77	113	74	70	64	68	41	43	102	103	85
105	131	106	19	13	63	130	17	67	40	9	42	10	101	127	14	12	18	11	62
126	60	128	61	129	3	4	8	16	7	125	1	2	6	15	5				

LEADFL

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

```
  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0  35   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
   0   0   0   0  36  37   0   0  39   0  32  33  34   0  29   0   0   0  17  18
  19  14  15  19   0   0   0  20  21  22   0   0   8   9   6   9   0  10  11   3
   0   0   0   0   0   0  40  40   0  24  24  25   0  25  26   0   0


NUMBER OF TESTS =            3


                    TIME FOR VECTOR COMPUTATION IS       20899
 NL =    23410


TEST NUMBER            1

PIN VALUES
 PIN      1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
   ????  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41
   ???*  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62
   ????  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83
   ???T  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100 101 102 103 104
   ???I 106 107 108 109 110 111 112 113 114 115 116 117
 VALUE    0   1   0   1   1   1   0   0   1   0   0   0   0   1   0   1   0   1   1  -1
   ___^  -1  -1   1  -1  -1  -1  -1   0  -1  -1  -1  -1   1  -1  -1  -1  -1   1   0   0
   ????   0   3  -1  -1  -1  -1   0  -1  -1  -1  -1   1  -1  -1  -1  -1   1   0   1   1
   ????   1   3   3   1   1   0   1   3   3   2   1   3   2   1   0   0   0   0   3   3
   ????   0   0   1   3   3   0   0   1   3   3   3   3   0   3   2   3   1   1   1   0
   ????   0   1   3   1   1   2   3   0   2   0   0   1

PRINCIPAL INPUTS
GATE VALUE
   1   0
   2   1
   3   0
   4   1
   5   1
   6   1
   7   0
   8   0
   9   1
  10   0
  11   0
  12   0
  13   0
  14   1

PRINCIPAL OUTPUTS
GATE VALUE
 117   0
 118   2
 119   1
```

21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	35	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	36	37	0	0	39	0	32	33	34	0	29	0	0	0	17	18
19	14	15	19	0	0	0	20	21	22	0	0	8	9	6	9	0	10	11	3
0	0	0	0	0	0	40	40	0	24	24	25	0	25	26	0	0			

NUMBER OF TESTS = 3

TIME FOR VECTOR COMPUTATION IS 20859

NL = 23410

TEST NUMBER 1

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
.???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	1	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	1	1	-1
___^	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0
????	0	3	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	1	1
????	1	3	3	1	1	0	1	3	3	2	1	3	2	1	0	0	0	0	3	3
????	0	0	1	3	3	0	0	1	3	3	3	3	0	3	2	3	1	1	1	0
????	0	1	3	1	1	2	3	0	2	0	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	1
3	0
4	1
5	1
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1

120	1
121	2
122	3
123	0
124	0

TIME FOR VECTOR COMPUTATION IS 22

NL = 23435

TEST NUMBER 2

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	1	0	1	1	0	0	0	1	0	0	0	0	1	0	1	0	1	1	-1
___^	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0
????	0	3	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	1	1
????	1	3	3	1	1	0	1	3	3	2	1	3	2	1	0	0	0	0	3	3
????	0	0	1	3	3	0	0	1	3	3	3	3	0	3	2	3	1	1	1	0
????	0	1	3	1	1	2	3	0	2	0	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	1
3	0
4	1
5	1
6	0
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1
120	1
121	2
122	3
123	0
124	0

TIME FOR VECTOR COMPUTATION IS 155

NL = 23554

TEST NUMBER 3

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	0	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	1	1	-1
___^	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0
????	0	3	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	1	1
????	1	3	3	1	1	0	1	3	3	2	1	3	2	1	0	0	0	0	3	3
????	0	0	1	3	3	0	0	1	3	3	3	3	0	3	2	3	1	1	1	0
????	0	1	3	1	1	2	3	0	2	0	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	0
3	0
4	1
5	1
6	1
7	0
8	0
9	1
10	0
11	0
12	0
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1
120	1
121	2
122	3
123	0
124	0

END OF RUN

END OF SCF FILE, ACCOUNT: 33232-PRZYBO PCY= 411, SPC= 475, PGS= 7, CRD=

Problem #3

```
ACCOUNT: 33232-PRZYBO     TRAIN: A     REEL: 37      TRACK: 7     PUNCH: N     LINES/INCH:
SDF  CONTROL WORD, OCTAL - 50C130C00000, I =  1, FT = S, P =  0, CT = FIELDATA
ABOVE CONTROL WORD IN BLOCK       1 LOOKS WRONG     ?????????????????????????????????????
*SDFF*
```

TIME USED TO FORM COMPONENTS IS   15975

FAULT RUN NUMBER   3

FAULTS

GATE	TYPE OF FAULT			
146	LINE	3 OF GATE	44	STUCK-AT-0

GATE ASSIGNMENTS

GT	IP	ADJACENT PINS		
1	1	1		
2	1	2		
3	1	3		
4	1	4		
5	1	5		
6	1	6		
7	1	7		
8	1	8		
9	1	9		
10	1	10		
11	1	11		
12	1	12		
13	1	13		
14	1	14		
15	4	5	15	
16	4	7	16	
17	4	9	17	
18	4	11	18	
19	4	13	19	
20	5	1	5	20
21	5	6	20	21
22	5	6	2	22
23	5	15	22	23
24	8	21	23	24
25	5	3	15	25
26	5	4	5	26
27	3	6	27	
28	6	25	26	28
29	8	27	28	29
30	5	7	1	30
31	5	8	30	31
32	5	9	2	32
33	5	16	32	33
34	8	31	33	34
35	5	16	3	35

36	5	4	7	36
37	3	8	37	
38	6	35	36	38
39	8	37	38	39
40	5	9	1	40
41	5	10	40	41
42	5	10	2	42
43	5	17	42	43
44	8	41	43	117
45	5	17	3	45
46	5	4	9	46
47	3	10	47	
48	6	45	46	48
49	8	47	48	49
50	5	11	1	50
51	5	12	50	51
52	5	12	2	52
53	5	18	52	53
54	8	51	53	54
55	5	18	3	55
56	5	4	11	56
57	3	12	57	
58	6	55	56	58
59	8	57	58	59
60	3	29	60	
61	5	24	39	61
62	5	24	34	62
63	5	49	62	63
64	5	24	34	64
65	5	44	64	65
66	5	59	65	66
67	6	60	61	67
68	6	63	67	68
69	8	66	68	69
70	5	24	34	70
71	5	44	70	71
72	5	54	71	72
73	7	14	72	73
74	5	24	34	74
75	5	44	74	75
76	7	54	75	76
77	11	24	29	77
78	5	8	54	78
79	5	44	78	79
80	5	34	79	80
81	5	19	80	81
82	5	44	34	82
83	5	59	82	83
84	5	19	83	84
85	5	34	49	85
86	5	19	85	86
87	5	19	39	87
88	6	81	84	88
89	6	86	88	89
90	8	87	89	90
91	11	34	39	91
92	5	14	54	92
93	5	44	92	93

94	5	19	93	94			
95	5	44	59	95			
96	5	19	95	96			
97	5	49	19	97			
98	6	94	96	98			
99	8	97	98	99			
100	11	44	49	100			
101	5	14	54	101			
102	5	19	101	102			
103	5	59	19	103			
104	8	102	103	104			
105	11	54	59	105			
106	7	14	19	106			
107	4	69	107				
108	4	73	108				
109	6	107	108	109			
110	11	77	90	110			
111	11	91	99	111			
112	11	100	104	112			
113	11	105	106	113			
114	5	110	111	114			
115	5	112	114	115			
116	5	113	115	116			
117	2	69					
118	2	76					
119	2	109					
120	2	110					
121	2	111					
122	2	112					
123	2	113					
124	2	116					
125	12	1	2	5	6	15	24
126	12	1	2	7	8	16	34
127	12	1	2	11	12	18	54
128	13	3	4	5	6	15	29
129	13	3	4	7	8	16	39
130	13	3	4	9	10	17	49
131	13	3	4	11	12	13	59
132	14	24	34	44	59	65	
133	14	19	34	44	59	84	
134	14	14	19	44	54	94	
135	14	110	111	112	113	116	
136	15	14	24	34	44	54	73
137	16	60	61	63	66	69	
138	16	81	84	86	87	90	
139	17	24	34	49	63		
140	17	19	34	49	86		
141	17	19	44	59	96		
142	17	14	19	54	102		
143	18	24	34	44	54	76	
144	19	94	96	97	99		
145	26	8	19	34	44	54	81
146	22	117	44				

PIN ASSIGNMENTS

PIN ADJACENT GATES

1	1	125	126	40	127

2	2	125	126	42	127						
3	3	128	129	130	131						
4	4	128	129	130	131						
5	5	15	125	128							
6	6	125	128								
7	7	16	126	129							
8	8	126	129	145							
9	9	17	40	130							
10	10	41	42	130							
11	11	18	127	131							
12	12	127	131								
13	13	19									
14	14	136	134	142	106						
15	15	125	128								
16	16	126	129								
17	17	43	130								
18	18	127	131								
19	19	145	133	140	87	134	141	97	142	103	74
20	20	21									
21	21	24									
22	22	23									
23	23	24									
24	125	61	139	132	136	143	77				
25	25	28									
26	26	28									
27	27	29									
28	28	29									
29	128	60	77								
30	30	31									
31	31	34									
32	32	33									
33	33	34									
34	126	139	132	136	143	145	133	140	91		
35	35	38									
36	36	38									
37	37	39									
38	38	39									
39	129	61	87	91							
40	40	41									
41	41	44									
42	42	43									
43	43	44									
44	146	132	136	143	145	133	134	141	100		
45	45	48									
46	46	48									
47	47	49									
48	48	49									
49	130	139	140	97	100						
50	50	51									
51	51	54									
52	52	53									
53	53	54									
54	127	136	143	145	134	142	105				
55	55	58									
56	56	58									
57	57	59									
58	58	59									
59	131	132	133	141	103	105					

60	60	137	
61	61	137	
62	62	63	
63	139	137	
64	64	65	
65	65	66	
66	132	137	
67	67	68	
68	68	69	
69	137	107	117
70	70	71	
71	71	72	
72	72	73	
73	136	108	
74	74	75	
75	75	76	
76	143	118	
77	77	110	
78	78	79	
79	79	80	
80	80	81	
81	145	138	
82	82	83	
83	83	84	
84	133	138	
85	85	86	
86	140	138	
87	87	138	
88	88	89	
89	89	90	
90	138	110	
91	91	111	
92	92	93	
93	93	94	
94	134	144	
95	95	96	
96	141	144	
97	97	144	
98	98	99	
99	144	111	
100	100	112	
101	101	102	
102	142	104	
103	103	104	
104	104	112	
105	105	113	
106	106	113	
107	107	109	
108	108	109	
109	109	119	
110	110	135	120
111	111	135	121
112	112	135	122
113	113	135	123
114	114	115	
115	115	116	
116	135	124	
117	44	146	

LEVELLING

FLEVEL = 1 FLEVEL = 26 TOPLEV = 71

SCANNING SECUENCE

146	100	112	122	141	134	144	111	121	133	145	138	110	120	135	124	143	118	136	108
132	137	117	107	109	119	44	104	97	91	140	87	77	113	60	61	139	130	126	125
41	43	17	142	103	19	13	129	8	16	7	128	105	131	127	3	4	12	18	11
106	14	6	15	5	40	9	1	42	10	2									

LEADFL

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117			
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	22	0	0	24	0	0	0	20	0	0	0	0	0	0	0
12	0	0	12	0	0	0	0	0	13	0	0	0	7	0	7	0	0	8	3
0	0	0	0	0	0	25	25	0	15	15	15	0	0	0	0	0			

NUMBER OF TESTS = 5

TIME FOR VECTOR COMPUTATION IS 205657

NL = 186700

TEST NUMBER 1

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	1	1	0	1	1	1	0	1	1	1	0	0	1	0	0	0	0	1	-1
___	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0
????	0	3	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	1	-1
????	-1	-1	3	-1	-1	0	-1	-1	-1	2	-1	-1	2	1	-1	-1	-1	0	-1	-1
????	-1	0	1	-1	-1	0	0	-1	-1	3	-1	3	0	-1	2	3	-1	1	1	0
????	0	1	3	1	1	2	3	0	-1	-1	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	1
3	1
4	0
5	1
6	1
7	1
8	0
9	1
10	1
11	1
12	0

13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1
120	1
121	2
122	3
123	0
124	0

TIME FOR VECTOR COMPUTATION IS 103

NL = 186806

TEST NUMBER 2

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	1	1	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	1	-1
___	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0
????	0	3	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	1	-1
????	-1	-1	3	-1	-1	0	-1	-1	-1	2	-1	-1	2	1	-1	-1	-1	0	-1	-1
????	-1	0	1	-1	-1	0	0	-1	-1	3	-1	3	0	-1	2	3	-1	1	1	0
????	0	1	3	1	1	2	3	0	-1	-1	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	1
3	1
4	0
5	0
6	0
7	1
8	0
9	1
10	1
11	1
12	0
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1
120	1

121	2
122	3
123	0
124	0

TIME FOR VECTOR COMPUTATION IS 246

NL = 187059

TEST NUMBER 3

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
. ????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	0	1	0	1	1	1	0	1	1	1	0	0	1	0	0	0	0	1	-1
___	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0
????	0	3	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	1	-1
????	-1	-1	3	-1	-1	0	-1	-1	-1	2	-1	-1	2	1	-1	-1	-1	0	-1	-1
????	-1	0	1	-1	-1	0	0	-1	-1	3	-1	3	0	-1	2	3	-1	1	1	0
????	0	1	3	1	1	2	3	0	-1	-1	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	0
3	1
4	0
5	1
6	1
7	1
8	0
9	1
10	1
11	1
12	0
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1
120	1
121	2
122	3
123	0
124	0

TIME FOR VECTOR COMPUTATION IS 74

NL = 187136

TEST NUMBER 4

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	0	1	0	0	1	1	0	1	1	1	0	0	1	1	0	0	0	1	-1
___	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0
????	0	3	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	1	-1
????	-1	-1	3	-1	-1	0	-1	-1	-1	2	-1	-1	2	1	-1	-1	-1	0	-1	-1
????	-1	0	1	-1	-1	0	0	-1	-1	3	-1	3	0	-1	2	3	-1	1	1	0
????	0	1	3	1	1	2	3	0	-1	-1	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	0
3	1
4	0
5	0
6	1
7	1
8	0
9	1
10	1
11	1
12	0
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1
120	1
121	2
122	3
123	0
124	0

TIME FOR VECTOR COMPUTATION IS 61

NL = 187201

TEST NUMBER 5

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	0	1	0	0	0	1	0	1	1	1	0	0	1	1	0	0	0	1	-1
___	-1	-1	1	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	0
????	0	3	-1	-1	-1	-1	0	-1	-1	-1	-1	1	-1	-1	-1	-1	1	0	1	-1
????	-1	-1	3	-1	-1	0	-1	-1	-1	2	-1	-1	2	1	-1	-1	-1	0	-1	-1

```
 ????  -1   0   1  -1  -1   0   0  -1  -1   3  -1   3   0  -1   2   3  -1   1   1   0
 ????   0   1   3   1   1   2   3   0  -1  -1   0   1
```

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	0
3	1
4	0
5	0
6	0
7	1
8	0
9	1
10	1
11	1
12	C
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	0
118	2
119	1
120	1
121	2
122	3
123	0
124	0

END OF RUN

END OF SCF FILE.   ACCOUNT: 33232-PRZYEO   PCY=   508. SPC=   587. PGS=   9. CPD=

Problem #4

```
  ACCOUNT: 33232-PRZYBO    TRAIN: A    REEL: 00      TRACK: 7    PUNCH: N    LINES/INCH:
SDF  CONTROL WORD, OCTAL - 500130000000, I =  1, FF = S, P =  0, CT = FIELDATA
ABOVE CONTROL WORD IN BLOCK        1 LOOKS WRONG     ?????????????????????????????????????
*SDFF*                                               -------------------------------------


                 TIME USED TO FORM COMPONENTS IS          0



 FAULT RUN NUMBER                4


 FAULTS

          GATE    TYPE OF FAULT
             1    INPUT SET TO 0
             2    INPUT SET TO 0
             3    INPUT SET TO 0
             4    INPUT SET TO 1
             5    INPUT SET TO 1
             6    INPUT SET TO 1
             7    INPUT SET TO 0
             8    INPUT SET TO 0
             9    INPUT SET TO 1
            10    INPUT SET TO 0
            11    INPUT SET TO 0
            12    INPUT SET TO 0
            13    INPUT SET TO 0
            14    INPUT SET TO 1
           125    LINE              3 OF GATE           44  STUCK-AT-0


 GATE ASSIGNMENTS
  GT IP ADJACENT PINS
   1  20   1
   2  20   2
   3  20   3
   4  21   4
   5  21   5
   6  21   6
   7  20   7
   8  20   8
   9  21   9
  10  20  10
  11  20  11
  12  20  12
  13  20  13
  14  21  14
  15   4   5  15
  16   4   7  16
  17   4   9  17
  18   4  11  18
  19   4  13  19
  20   5   1   5  20
  21   5   6  20  21
```

22	5	6	2	22
23	5	15	22	23
24	8	21	23	24
25	5	3	15	25
26	5	4	5	26
27	3	6	27	
28	6	25	26	28
29	8	27	28	29
30	5	7	1	30
31	5	8	30	31
32	5	8	2	32
33	5	16	32	33
34	8	31	33	34
35	5	16	3	35
36	5	4	7	36
37	3	8	37	
38	6	35	36	38
39	8	37	39	39
40	5	9	1	40
41	5	10	40	41
42	5	10	2	42
43	5	17	42	43
44	8	41	43	117
45	5	17	3	45
46	5	4	9	46
47	3	10	47	
48	6	45	46	48
49	8	47	48	49
50	5	11	1	50
51	5	12	50	51
52	5	12	2	52
53	5	18	52	53
54	8	51	53	54
55	5	18	3	55
56	5	4	11	56
57	3	12	57	
58	6	55	56	58
59	8	57	58	59
60	3	29	60	
61	5	24	39	61
62	5	24	34	62
63	5	49	62	63
64	5	24	34	64
65	5	44	64	65
66	5	59	65	66
67	6	60	61	67
68	6	63	67	68
69	8	66	68	69
70	5	24	34	70
71	5	44	70	71
72	5	54	71	72
73	7	14	72	73
74	5	24	34	74
75	5	44	74	75
76	7	54	75	76
77	11	24	29	77
78	5	8	54	78
79	5	44	78	79

80	5	34	79	80
81	5	19	80	81
82	5	44	34	82
83	5	59	82	83
84	5	19	83	84
85	5	34	49	85
86	5	19	85	86
87	5	19	39	87
88	6	81	84	88
89	6	86	88	89
90	8	87	89	90
91	11	34	39	91
92	5	14	54	92
93	5	44	92	93
94	5	19	93	94
95	5	44	59	95
96	5	19	95	96
97	5	49	19	97
98	6	54	96	98
99	8	97	98	99
100	11	44	49	100
101	5	14	54	101
102	5	19	101	102
103	5	59	19	103
104	8	102	103	104
105	11	54	59	105
106	7	14	19	106
107	4	69	107	
108	4	73	108	
109	6	107	108	109
110	11	77	90	110
111	11	91	99	111
112	11	100	104	112
113	11	105	106	113
114	5	110	111	114
115	5	112	114	115
116	5	113	115	116
117	2	69		
118	2	76		
119	2	109		
120	2	110		
121	2	111		
122	2	112		
123	2	113		
124	2	116		
125	22	117	44	

PIN ASSIGNMENTS

PIN	ADJACENT GATES				
1	1	20	30	40	50
2	2	22	32	42	52
3	3	25	35	45	55
4	4	26	36	46	56
5	5	15	20	25	
6	6	21	22	27	
7	7	16	30	36	
8	8	31	32	37	78

9	9	17	40	46							
10	10	41	42	47							
11	11	18	50	56							
12	12	51	52	57							
13	13	19									
14	14	73	92	101	106						
15	15	23	25								
16	16	33	35								
17	17	43	45								
18	18	53	55								
19	19	81	84	86	87	94	96	97	102	103	106
20	20	21									
21	21	24									
22	22	23									
23	23	24									
24	24	61	62	64	70	74	77				
25	25	26									
26	26	28									
27	27	29									
28	28	29									
29	29	60	77								
30	30	31									
31	31	34									
32	32	33									
33	33	34									
34	34	62	64	70	74	80	82	85	91		
35	35	36									
36	36	38									
37	37	39									
38	38	39									
39	39	61	87	91							
40	40	41									
41	41	44									
42	42	43									
43	43	44									
44	125	65	71	75	79	82	93	95	100		
45	45	46									
46	46	48									
47	47	49									
48	48	49									
49	49	63	85	97	100						
50	50	51									
51	51	54									
52	52	53									
53	53	54									
54	54	72	76	78	92	101	105				
55	55	56									
56	56	58									
57	57	59									
58	58	59									
59	59	66	83	95	103	105					
60	60	67									
61	61	67									
62	62	63									
63	63	68									
64	64	69									
65	65	66									
66	66	69									

67	67	68	
68	68	69	
69	69	107	117
70	70	71	
71	71	72	
72	72	73	
73	73	108	
74	74	75	
75	75	76	
76	76	118	
77	77	110	
78	78	79	
79	79	80	
80	80	81	
81	81	88	
82	82	83	
83	83	84	
84	84	88	
85	85	86	
86	86	89	
87	87	90	
88	88	89	
89	89	90	
90	90	110	
91	91	111	
92	92	93	
93	93	94	
94	94	98	
95	95	96	
96	96	98	
97	97	99	
98	98	99	
99	99	111	
100	100	112	
101	101	102	
102	102	104	
103	103	104	
104	104	112	
105	105	113	
106	106	113	
107	107	109	
108	108	109	
109	109	119	
110	110	114	120
111	111	114	121
112	112	115	122
113	113	116	123
114	114	115	
115	115	116	
116	116	124	
117	44	125	

LEVELLING

FLEVEL = 85 FLEVEL = 125 TOPLEV = 125

SCANNING SEQUENCE

1	14	13	19	106	12	57	11	18	50	51	10	47	9	17	40	41	8	37	7
16	30	31	6	27	5	15	20	21	4	26	36	46	56	3	25	28	29	60	35

38	39	45	48	49	55	56	59	103	97	87	2	22	23	24	61	67	77	32	33
34	74	70	64	62	65	63	68	91	42	43	44	52	53	54	105	78	113	123	86
101	102	104	92	125	93	79	65	71	72	75	76	118	62	80	95	83	56	69	117
107	100	112	122	96	94	98	99	111	121	84	81	88	39	90	110	120	114	115	116
124	73	108	109	119															

LEADFL

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60
61	62	63	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80
81	82	83	84	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100
101	102	103	104	105	106	107	108	109	110	111	112	113	114	115	116	117			
28	73	46	34	31	53	32	77	33	70	34	73	4	122	54	60	71	74	112	29
55	54	55	65	37	37	38	38	58	23	61	60	61	95	41	41	42	42	69	17
72	71	72	102	44	44	45	45	102	11	75	74	75	92	47	47	48	48	98	57
57	67	69	83	98	99	68	99	101	89	90	122	123	91	92	0	116	87	95	112
113	97	111	113	80	114	115	114	115	116	109	36	106	107	105	107	108	108	109	103
82	83	83	103	78	78	124	124	0	118	118	119	120	119	120	0	85			

NUMBER OF TESTS = 1

TIME FOR VECTOR COMPUTATION IS 90

NL = 0

TEST NUMBER 1

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???1	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	0	0	0	1	1	1	0	0	1	0	0	0	0	1	0	1	0	1	1	0
????	0	0	1	0	1	1	1	0	0	0	0	0	1	0	0	0	0	1	0	0
????	0	3	0	1	0	1	0	0	0	0	0	1	0	0	0	0	1	0	1	1
????	1	3	3	1	1	0	1	3	3	2	1	3	2	1	0	0	0	0	3	3
????	0	0	1	3	3	0	0	1	3	3	3	3	0	3	2	3	1	1	1	0
????	0	1	3	1	1	2	3	0	2	0	0	1								

PRINCIPAL INPUTS

GATE	VALUE
1	0
2	0
3	0
4	1
5	1
6	1
7	0
8	0
9	1
10	0
11	0

```
  12    0
  13    0
  14    1

PRINCIPAL OUTPUTS
GATE VALUE
 117    0
 118    2
 119    1
 120    1
 121    2
 122    3
 123    0
 124    0



END OF RUN

END OF SCF FILE,   ACCOUNT: 33232-PRZYBO     PCY=     336, SPC=     383, PGS=       6, CRD=
```

Problem #5

```
   ACCOUNT: 33232-PRZYBO     TRAIN: A     REEL: 24      TRACK: 7    PUNCH: N    LINES/INCH
SDF  CONTROL WORD, OCTAL - 500130000000, I =  1, FT = S, P =  0, CT = FIELDATA
ABOVE CONTROL WORD IN BLOCK       1 LOOKS WRONG    ?????????????????????????????????????
*SDFF*
```

TIME USED TO FORM COMPONENTS IS 0

FAULT RUN NUMBER 5

FAULTS

GATE	TYPE OF FAULT
125	LINE 1 OF GATE 24 STUCK-AT-0

GATE ASSIGNMENTS

GT	TP	ADJACENT PINS		
1	1	1		
2	1	2		
3	1	3		
4	1	4		
5	1	5		
6	1	6		
7	1	7		
8	1	8		
9	1	9		
10	1	10		
11	1	11		
12	1	12		
13	1	13		
14	1	14		
15	4	5	15	
16	4	7	16	
17	4	9	17	
18	4	11	18	
19	4	13	19	
20	5	1	5	20
21	5	6	20	21
22	5	6	2	22
23	5	15	22	23
24	8	117	23	24
25	5	3	15	25
26	5	4	5	26
27	3	6	27	
28	6	25	26	29
29	8	27	28	29
30	5	7	1	30
31	5	8	30	31
32	5	8	2	32
33	5	16	32	33
34	8	31	33	34
35	5	16	3	35

36	5	4	7	36
37	3	8	37	
38	6	35	36	38
39	8	37	38	39
40	5	9	1	40
41	5	10	40	41
42	5	10	2	42
43	5	17	42	43
44	8	41	43	44
45	5	17	3	45
46	5	4	9	46
47	3	10	47	
48	6	45	46	48
49	8	47	48	49
50	5	11	1	50
51	5	12	50	51
52	5	12	2	52
53	5	18	52	53
54	8	51	53	54
55	5	18	3	55
56	5	4	11	56
57	3	12	57	
58	6	55	56	58
59	8	57	58	59
60	3	29	60	
61	5	24	39	61
62	5	24	34	62
63	5	49	62	63
64	5	24	34	64
65	5	44	64	65
66	5	59	65	66
67	6	60	61	67
68	6	63	67	68
69	8	66	68	69
70	5	24	34	70
71	5	44	70	71
72	5	54	71	72
73	7	14	72	73
74	5	24	34	74
75	5	44	74	75
76	7	54	75	76
77	11	24	29	77
78	5	8	54	78
79	5	44	78	79
80	5	34	79	80
81	5	19	80	81
82	5	44	34	82
83	5	59	82	83
84	5	19	83	84
85	5	34	49	85
86	5	19	85	86
87	5	19	39	87
88	6	81	84	88
89	6	86	88	89
90	8	87	89	90
91	11	34	39	91
92	5	14	54	92
93	5	44	92	93

94	5	19	93	94
95	5	44	59	95
96	5	19	95	96
97	5	49	19	97
98	6	94	96	98
99	8	97	98	99
100	11	44	49	100
101	5	14	54	101
102	5	19	101	102
103	5	59	19	103
104	8	102	103	104
105	11	54	59	105
106	7	14	19	106
107	4	69	107	
108	4	73	108	
109	6	107	108	109
110	11	77	90	110
111	11	91	99	111
112	11	100	104	112
113	11	105	106	113
114	5	110	111	114
115	5	112	114	115
116	5	113	115	116
117	2	69		
118	2	76		
119	2	109		
120	2	110		
121	2	111		
122	2	112		
123	2	113		
124	2	116		
125	22	21	117	

PIN ASSIGNMENTS

PIN	ADJACENT GATES										
1	1	20	30	40	50						
2	2	22	32	42	52						
3	3	25	35	45	55						
4	4	26	36	46	56						
5	5	15	20	26							
6	6	21	22	27							
7	7	16	30	36							
8	8	31	32	37	78						
9	9	17	40	46							
10	10	41	42	47							
11	11	18	50	56							
12	12	51	52	57							
13	13	19									
14	14	73	92	101	106						
15	15	23	25								
16	16	33	35								
17	17	43	45								
18	18	53	55								
19	19	81	84	86	87	94	96	97	102	103	106
20	20	21									
21	21	125									
22	22	23									

23	23	24							
24	24	61	62	64	70	74	77		
25	25	28							
26	26	28							
27	27	29							
28	28	29							
29	29	60	77						
30	30	31							
31	31	34							
32	32	33							
33	33	34							
34	34	62	64	70	74	80	82	85	91
35	35	38							
36	36	38							
37	37	39							
38	38	39							
39	39	61	87	91					
40	40	41							
41	41	44							
42	42	43							
43	43	44							
44	44	65	71	75	79	82	93	95	100
45	45	48							
46	46	48							
47	47	49							
48	48	49							
49	49	63	85	97	100				
50	50	51							
51	51	54							
52	52	53							
53	53	54							
54	54	72	76	78	92	101	105		
55	55	58							
56	56	58							
57	57	59							
58	58	59							
59	59	66	83	95	103	105			
60	60	67							
61	61	67							
62	62	63							
63	63	68							
64	64	65							
65	65	66							
66	66	69							
67	67	68							
68	69	69							
69	69	107	117						
70	70	71							
71	71	72							
72	72	73							
73	73	108							
74	74	75							
75	75	76							
76	76	118							
77	77	110							
78	78	79							
79	79	80							
80	80	81							

```
  81  81  88
  82  82  83
  83  83  84
  84  84  88
  85  85  86
  86  86  89
  87  87  90
  88  88  89
  89  89  90
  90  90 110
  91  91 111
  92  92  93
  93  93  94
  94  94  98
  95  95  96
  96  96  98
  97  97  99
  98  98  99
  99  99 111
 100 100 112
 101 101 102
 102 102 104
 103 103 104
 104 104 112
 105 105 113
 106 106 113
 107 107 109
 108 108 109
 109 109 119
 110 110 114 120
 111 111 114 121
 112 112 115 122
 113 113 116 123
 114 114 115
 115 115 116
 116 116 124
 117 125  24
```

```
LEVELLING
FLEVEL =            1 PLEVEL =           31 TOPLEV =          122
SCANNING SEQUENCE
 125  24  77 110 120 114 115 116 124  74  75  76 118  70  71  72  73 108  64  65
  66  62  63  61  67  68  69 117 107 109 119  21  23  90 111 112 113  60  29  20
  22  27   6  28  87  89  91  39  99 100 104 105 106  37  38  25  15  26   5  86
  88  97  98 102 103  35  36  85  49  47  48  45  46  81  84  94  96  19  13 101
  80  83  93  95  59  57  58  55  56   3   4  79  82  34  44  31  33  41  43  30
  16  32  40  17  42  10   7   9  92  14  78  54  51  53   8  50   1  18  52   2
  12  11
```

```
LEADFL
   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20
  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40
  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60
  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80
  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 100
 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117
```

0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	23	26	20	21	27	26	27	29	15	16	17	18	11	12	0	4	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	30	30	0	6	0	0	0	7	8	0	2			

NUMBER OF TESTS = 1

TIME FOR VECTOR COMPUTATION IS 27666

NL = 49726

TEST NUMBER 1

PIN VALUES

PIN	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
????	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41
???*	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62
????	64	65	66	67	68	69	70	71	72	73	74	75	76	77	78	79	80	81	82	83
???T	85	86	87	88	89	90	91	92	93	94	95	96	97	98	99	100	101	102	103	104
???I	106	107	108	109	110	111	112	113	114	115	116	117								
VALUE	1	0	1	1	1	1	0	1	0	1	0	1	0	1	0	1	1	1	1	1
????	0	0	2	0	1	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0
????	0	1	1	0	1	1	0	0	0	0	0	1	1	0	1	1	0	0	0	2
????	2	2	0	0	0	1	2	2	2	3	2	2	3	2	1	1	1	1	1	0
????	0	0	0	1	1	0	1	1	1	1	0	0	0	1	0	1	1	1	0	0
????	0	0	2	2	2	1	1	1	2	2	2	3								

PRINCIPAL INPUTS

GATE	VALUE
1	1
2	0
3	1
4	1
5	1
6	1
7	0
8	1
9	0
10	1
11	0
12	1
13	0
14	1

PRINCIPAL OUTPUTS

GATE	VALUE
117	1
118	3
119	2
120	2
121	1
122	1
123	1
124	2

END OF RUN

END OF SDF FILE,   ACCOUNT: 33232-PRZYBO   PCY=   322, SPC=   369, PGS=   6, CRD=

Problem #6

```
   ACCOUNT: 33232-PRZYBO     TRAIN: A     REEL: 24       TRACK: 7    PUNCH: N    LINES/INCH
SOF  CONTROL WORD, OCTAL - 500130000000, I =  1, FT = S, P =  0, CT = FIELDATA
ABOVE CONTROL WORD IN BLOCK       1 LOOKS WRONG     ?????????????????????????????????????
*SOFF*


                    TIME USED TO FORM COMPONENTS IS          0



FAULT RUN NUMBER               6


FAULTS

         GATE   TYPE OF FAULT
          125   LINE             1 OF GATE              24  STUCK-AT-0
          126   CROSS WIRE AND BETWEEN LINE                3 OF GATE             24
                                   AND LINE                3 OF GATE             29


GATE ASSIGNMENTS
 GT TP ADJACENT PINS
   1   1   1
   2   1   2
   3   1   3
   4   1   4
   5   1   5
   6   1   6
   7   1   7
   8   1   8
   9   1   9
  10   1  10
  11   1  11
  12   1  12
  13   1  13
  14   1  14
  15   4   5  15
  16   4   7  16
  17   4   9  17
  18   4  11  18
  19   4  13  19
  20   5   1   5  20
  21   5   6  20  21
  22   5   6   2  22
  23   5  15  22  23
  24   8 117  23 118
  25   5   3  15  25
  26   5   4   5  26
  27   3   6  27
  28   6  25  26  28
  29   8  27  28 119
  30   5   7   1  30
  31   5   8  30  31
  32   5   8   2  32
  33   5  16  32  33
```

34	8	31	33	34
35	5	16	3	35
36	5	4	7	36
37	3	8	37	
38	6	35	36	38
39	8	37	38	39
40	5	9	1	40
41	5	10	40	41
42	5	10	2	42
43	5	17	42	43
44	8	41	43	44
45	5	17	3	45
46	5	4	9	46
47	3	10	47	
48	6	45	46	48
49	8	47	48	49
50	5	11	1	50
51	5	12	50	51
52	5	12	2	52
53	5	18	52	53
54	8	51	53	54
55	5	18	3	55
56	5	4	11	56
57	3	12	57	
58	6	55	56	58
59	8	57	58	59
60	3	29	60	
61	5	24	39	61
62	5	24	34	62
63	5	49	62	63
64	5	24	34	64
65	5	44	64	65
66	5	59	65	66
67	6	60	61	67
68	6	63	67	68
69	8	66	68	69
70	5	24	34	70
71	5	44	70	71
72	5	54	71	72
73	7	14	72	73
74	5	24	34	74
75	5	44	74	75
76	7	54	75	76
77	11	24	29	77
78	5	8	54	78
79	5	44	78	79
80	5	34	79	80
81	5	19	80	81
82	5	44	34	82
83	5	59	82	83
84	5	19	83	84
85	5	34	49	85
86	5	19	85	86
87	5	19	39	87
88	6	81	84	88
89	6	86	88	89
90	8	87	89	90
91	11	34	39	91

92	5	14	54	92	
93	5	44	92	93	
94	5	19	93	94	
95	5	44	59	95	
96	5	19	95	96	
97	5	49	19	97	
98	6	94	96	98	
99	8	97	98	99	
100	11	44	49	100	
101	5	14	54	101	
102	5	19	101	102	
103	5	59	19	103	
104	8	102	103	104	
105	11	54	59	105	
106	7	14	19	106	
107	4	69	107		
108	4	73	108		
109	6	107	108	109	
110	11	77	90	110	
111	11	91	99	111	
112	11	100	104	112	
113	11	105	106	113	
114	5	110	111	114	
115	5	112	114	115	
116	5	113	115	116	
117	2	69			
118	2	76			
119	2	109			
120	2	110			
121	2	111			
122	2	112			
123	2	113			
124	2	116			
125	22	21	117		
126	24	118	119	24	29

PIN ASSIGNMENTS

PIN	ADJACENT GATES										
1	1	20	30	40	50						
2	2	22	32	42	52						
3	3	25	35	45	55						
4	4	26	36	46	56						
5	5	15	20	26							
6	6	21	22	27							
7	7	16	30	36							
8	8	31	32	37	78						
9	9	17	40	46							
10	10	41	42	47							
11	11	18	50	56							
12	12	51	52	57							
13	13	19									
14	14	73	92	101	106						
15	15	23	25								
16	16	33	35								
17	17	43	45								
18	18	53	55								
19	19	81	84	86	87	94	96	97	102	103	106

20	20	21							
21	21	125							
22	22	23							
23	23	24							
24	126	61	62	64	70	74	77		
25	25	28							
26	26	28							
27	27	29							
28	28	29							
29	126	60	77						
30	30	31							
31	31	34							
32	32	33							
33	33	34							
34	34	62	64	70	74	80	82	85	91
35	35	38							
36	36	38							
37	37	39							
38	38	39							
39	39	61	87	91					
40	40	41							
41	41	44							
42	42	43							
43	43	44							
44	44	65	71	75	79	82	93	95	100
45	45	48							
46	46	48							
47	47	49							
48	48	49							
49	49	63	85	97	100				
50	50	51							
51	51	54							
52	52	53							
53	53	54							
54	54	72	76	78	92	101	105		
55	55	58							
56	56	58							
57	57	59							
58	58	59							
59	59	66	83	95	103	105			
60	60	67							
61	61	67							
62	62	63							
63	63	68							
64	64	65							
65	65	66							
66	66	69							
67	67	68							
68	68	69							
69	69	107	117						
70	70	71							
71	71	72							
72	72	73							
73	73	108							
74	74	75							
75	75	76							
76	76	118							
77	77	110							

78	78	79	
79	79	80	
80	80	81	
81	81	88	
82	82	83	
83	83	84	
84	84	88	
85	85	86	
86	86	89	
87	87	90	
88	88	89	
89	89	90	
90	90	110	
91	91	111	
92	92	93	
93	93	94	
94	94	98	
95	95	96	
96	96	98	
97	97	99	
98	98	99	
99	99	111	
100	100	112	
101	101	102	
102	102	104	
103	103	104	
104	104	112	
105	105	113	
106	106	113	
107	107	109	
108	108	109	
109	109	119	
110	110	114	120
111	111	114	121
112	112	115	122
113	113	116	123
114	114	115	
115	115	116	
116	116	124	
117	125	24	
118	24	126	
119	29	126	

LEVELLING

FLEVEL = 3 PLEVEL = 33 TOPLEV = 123

SCANNING SEQUENCE

125	24	126	60	77	110	120	114	115	116	124	74	75	76	118	70	71	72	73	108
64	65	66	62	63	61	67	68	69	117	107	109	119	21	23	29	90	111	112	113
20	22	27	6	28	87	89	91	39	99	100	104	105	106	37	38	25	15	26	5
86	88	97	98	102	103	35	36	85	49	47	48	45	46	81	84	94	96	19	13
101	80	83	93	95	59	57	58	55	56	3	4	79	82	34	44	31	33	41	43
30	16	32	40	17	42	10	7	9	92	14	78	54	51	53	8	50	1	18	52
2	12	11																	

LEADFL

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

```
 21   22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38   39   40
 41   42   43   44   45   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60
 61   62   63   64   65   66   67   68   69   70   71   72   73   74   75   76   77   78   79   80
 81   82   83   84   85   86   87   88   89   90   91   92   93   94   95   96   97   98   99  100
101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119
  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  0    0    0   26    0    0    0    0    5    0    0    0    0    0    0    0    0    0    0    0
  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0   27
 27   25   28   22   23   29   28   29   31   17   19   19   20   13   14    0    6    0    0    0
  0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0    0
  0    0    0    0    0    0   32   32    0    8    0    0    0    9   10    0    2    3    0

NUMBER OF TESTS =                1


                    TIME FOR VECTOR COMPUTATION IS       196438
 NL =  323031


TEST NUMBER              1

PIN VALUES
 PIN        1    2    3    4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20
   ????    22   23   24   25   26   27   28   29   30   31   32   33   34   35   36   37   38   39   40   41
   ???*    43   44   45   46   47   48   49   50   51   52   53   54   55   56   57   58   59   60   61   62
   ????    64   65   66   67   68   69   70   71   72   73   74   75   76   77   78   79   80   81   82   83
   ???T    85   86   87   88   89   90   91   92   93   94   95   96   97   98   99  100  101  102  103  104
   ???I   106  107  108  109  110  111  112  113  114  115  116  117  118  119
 VALUE      0    0    1    1    1    0    0    1    0    1    0    1    0    1    0    1    1    1    1    0
   ????     0    0    3    0    1    0    1    0    0    0    0    0    1    1    0    1    1    0    0    0
   ????     0    1    1    0    1    1    0    0    0    0    0    1    1    0    1    1    0    0    0    3
   ????     3    3    0    0    0    1    3    3    3    2    3    3    2    3    1    1    1    1    1    0
   ????     0    0    0    1    1    0    1    1    1    1    0    0    0    1    0    1    1    1    0    0
   ????     0    0    3    3    3    1    1    1    3    3    3    0    1    0

PRINCIPAL INPUTS
GATE VALUE
   1    0
   2    0
   3    1
   4    1
   5    1
   6    0
   7    0
   8    1
   9    0
  10    1
  11    0
  12    1
  13    0
  14    1

PRINCIPAL OUTPUTS
GATE VALUE
 117    1
 118    2
 119    3
```

```
 120    3
 121    1
 122    1
 123    1
 124    3


END OF RUN

END OF SDF FILE,   ACCOUNT: 33232-PRZYBO    PCY=     327, SPC=     379, PGS=       6, CRD=
```

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. NBSIR 83-2794	2. Performing Organ. Report No.	3. Publication Date September 1983

4. TITLE AND SUBTITLE

ON GENERALIZING THE D-ALGORITHM

5. AUTHOR(S)

J. Scott Provan and Paul Domich

6. PERFORMING ORGANIZATION (If joint or other than NBS, see instructions)

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (Street, City, State, ZIP)

10. SUPPLEMENTARY NOTES

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT (A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)

We consider in this paper the d-algorithm of J. P. Roth, which tests for specific faulty behavior in the integrated circuit. We develop a formal and general mathematical description of the algorithm, which allows a large degree of flexibility and extension in its implementation. We include a subsequent FORTRAN coding of such an extended d-algorithm, along with some sample testing.

12. KEY WORDS (Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)

backtrack; d-algorithm; fault-specific testing; VLSI

13. AVAILABILITY

☒ Unlimited
☐ For Official Distribution. Do Not Release to NTIS
☐ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.
☒ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

126

15. Price

$14.50

USCOMM-DC 6043-P80