7-1989

A 15-Valued Algorithm for Test Pattern Generation

Akhtar Uz Zaman

M. Ali

Carlos R.P. Hartmann
Syracuse University, chartman@syr.edu

SCHOOL OF COMPUTER
AND INFORMATION SCIENCE

Syracuse
University

SU-CIS-89-02

A 15-Valued Algorithm for Test Pattern Generation

Akhtar-uz-zaman M. Ali, Carlos R. P. Hartmann

July 1989

Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

SU-CIS-89-02

A 15-Valued Algorithm for Test Pattern Generation

Akhtar-uz-zaman M. Ali, Carlos R. P. Hartmann

July 1989

School of Computer and Information Science
Suite 4-116
Center for Science and Technology
Syracuse, New York 13244-4100

(315) 443-2368

A 15-VALUED ALGORITHM FOR TEST PATTERN GENERATION

Akhtar-uz-zaman M. Ali[1]
Carlos R. P. Hartmann[2]

Abstract In this report we present a new algorithm for detecting single stuck-at faults in combinational circuits. This algorithm is based on a 15-valued system and introduces several new concepts to make test generation more efficient. This 15-valued system allows us to impose all the constraints that must be satisfied in order to sensitize a single path. Consequently all deterministic implications are fully considered prior to the enumeration process. The resulting ability to identify inconsistencies prior to enumeration improves the possibility of quicker identification of redundant faults. Instead of sensitizing a single gate at a time, we sensitize subpaths by sensitizing all gates lying between successive fanout stems and then consider the deterministic implications of such a sensitization. We have introduced several speed-up techniques that effectively combine the information provided by the deterministic path sensitization and that obtained from the circuit topology. These techniques improve the efficiency of the enumeration phase by substantially pruning the search space.

[1]A. M. Ali is with the Department of Electrical & Computer Engineering, 111 Link Hall, Syracuse University, Syracuse, N.Y. 13244-1240. (e-mail: ali@top.cis.syr.edu)

[2]C. R. P. Hartmann is with the School of Computer and Information Science, Suite 4-116, Center for Science and Technology, Syracuse University, Syracuse, N.Y. 13244-4100. (e-mail: hartmann@top.cis.syr.edu)

Contents

List of Tables

List of Figures

1 Introduction

The generation of test patterns for combinational circuits has been long recognized by researchers as a well defined mathematical problem which belongs to the class of NP-complete problems [9,11]. Several Automatic Test Pattern Generation (ATPG) algorithms for detecting stuck-at-faults in combinational circuits exist in the literature [5,6,8,10,12,14,15]. Most researchers characterize test pattern generation as a search problem and address strategies to make this search process efficient. For realistic circuit sizes the search space is prohibitively large and, to make matters worse, a solution is not always guaranteed to exist. However, as PODEM [10] first demonstrated, it is not necessary to explicitly search the entire space — sometimes a partial search can determine a test pattern or the fact that none exists. In fact the huge amount of backtracking computation that is sometimes required before recognizing that a test cannot be generated for a particular fault (such faults are termed *redundant* faults) proves to be a major bottleneck in any ATPG algorithm. In order to overcome this difficulty different strategies have been developed by researchers. These strategies vary from making use of unique implications to using circuit topology information. In spite of the improvements achieved by these strategies test pattern generation still remains a complex problem and the possibility of further improvements a viable one.

In this report we present an ATPG algorithm, for detecting single stuck-at-faults in combinational circuits that contain NOT, AND, NAND, OR, NOR, XOR and XNOR gates. This algorithm is based on a 15-valued logic system and introduces some novel approaches to make test pattern generation more efficient.

Test generation involves considering the value of a net in the good and the faulty circuit. This can be done by representing the value of a net as an ordered pair (b_g, b_f) where $b_g(b_f)$ is the value of the net in the good (faulty) circuit [13]. Thus the value of a net can be one of the elements of the set $U = \{(0,0),(0,1),(1,0),(1,1)\}$. In the process of generating tests it might not be possible to uniquely specify the value of a net as one of the elements of U. However, we may already know that a net cannot

assume one or more of these values. We incorporate this information by defining the value of a net as one of the 15 non-empty subsets of U. We denote these 15 sets as 0, 1, D, $\overline{D}$, 0/1, 0/D, 1/D, 0/$\overline{D}$, 1/$\overline{D}$, $D/\overline{D}$, 0/1/D, 0/1/$\overline{D}$, 0/D/$\overline{D}$, 1/D/$\overline{D}$, and 0/1/D/$\overline{D}$ where 0 = $\{(0,0)\}$, 1 = $\{(1,1)\}$, D = $\{(1,0)\}$, $\overline{D}$= $\{(0,1)\}$ and "/" denotes set union. Note that U =0/1/D/$\overline{D}$. These 15 values are equivalent to the elements of the logic system developed by Akers [3] to provide a tool for test generation. Tables 1, 2 and 3 represent the AND, NOT, and XOR functions in our 15-value system for the values 0, 1, D, and $\overline{D}$. The complete table for all 15 values can be easily constructed from the given tables by using the set union operation. The tables for all other logic functions can be obtained from these three tables.

Using this notation we will define a sensitized net as one whose value is either D, $\overline{D}$, or $D/\overline{D}$. Furthermore, if all the nets along a path in the circuit are sensitized, then the path is said to be sensitized. As will be seen later on, this 15-valued system exploits the linearity of XOR/XNOR gates during test generation. It also allows us to characterize all restrictions that are imposed by a fault and the particular circuit path chosen in order to propagate its effect.

There are three distinct phases in the algorithm presented here:

(i) Pre-processing phase (§2). In this phase we construct a set of trees based on the interdependence of circuit nets. Among other things this forest will be used to easily identify which circuit nets *must* be sensitized to derive a test. We also compute the token vectors which keep track of the parity of inversions between nets. This information is useful because it can identify which inputs of a gate may or may not be simultaneously sensitized.

(ii) Propagation phase (§3). In this phase we deliberately sensitize a single path from the fault site to a primary output (PO) and find all the resulting deterministic forward and backward implications. In the process other paths may get sensitized. Path selection is the only choice made in this phase—implications are based on all the constraints that *must* be satisfied in order to sensitize the chosen path. This is possible because of the completeness of the 15-valued system and the use of deterministic

implication rules.

(*iii*) Enumeration phase (§4). In general, the test cube constructed by the propagation phase will not yield a test—particularly because no arbitrary choices were made. Thus there may be gates whose input net values contain combinations capable of desensitizing the chosen path. In this phase we use an enumeration procedure to choose values for the primary inputs (PIs) so that such combinations can never occur.

To illustrate the above phases of our algorithm we will consider the fault net 3 $s-a-0$ in the circuit of Fig. 1.

In order to make the last two phases more efficient we have developed some speed-up techniques (§5). One is the extension of the contrapositive procedure presented in SOCRATES [15] for backtracking 0 and 1 values. However, our procedure not only generates the contrapositive assertions for all 15 values of our system, but also requires less computation and storage than SOCRATES. TOPS [12] extended the concept of headlines introduced in FAN [8] by using circuit topology to identify more nets that have the same independence property and thus their value justification can be postponed till the last stage of test generation. We will present a procedure, that not only takes into account the circuit structure but also how it gets modified by the constraints imposed by the values of a test cube, in order to potentially identify a larger set of nets whose value justification can be postponed till the end because they are guaranteed not to cause contradictions. Furthermore, we will show how backtracking of the values that desensitize the chosen path can help in the selection of PI values during the enumeration phase.

2 Pre-processing Phase

2.1 Construction of Dominator Forest

The importance of identifying nets that *must* be sensitized for a fault to be detected was first highlighted by Akers [3] and later by Fujiwara and Shimono [8]. As pointed out in TOPS [12], the concept of graph dominators [16] can be used to identify the

nets which *must* be sensitized to detect a fault. In the context of test generation we term the set of dominators of a net m as the set of all nets in the circuit which lie on every path from net m to any PO. By definition, net m is a dominator of itself; however, for ease of notation we define $D(m)$ as the set of all dominators of m except m itself. To account for multiple output circuits the concept of dominator tree can be extended to that of a forest. We present here a procedure to construct this forest for a given circuit. This forest will not only be used to compute the dominators for a particular fault site; but also for the sensitization of subpaths, selection of PIs in the enumeration phase and generation of the initial list of target faults.

We construct a set of trees such that every net of the circuit corresponds to a node in one of the trees in the forest. We start by creating as many trees as there are POs such that each PO corresponds to a root of a tree. However, new trees may be created during the procedure. Thereafter, each node which has not been marked as a leaf is inspected and the tree construction is continued as follows:

(i) If the node m_i being considered corresponds to the output net of a logic gate G in the circuit, then every input net of G becomes a child of this node m_i. Furthermore, if the input net is a PI it is marked as a PI leaf. If the input net is a fanout branch (FOB), then it is marked as a FOB leaf.

(ii) If the node m_i being inspected is a fanout stem (FOS), then wait until all the FOBs corresponding to this FOS have been marked as FOB leaves. Then find the immediate ancestor of all these FOB leaves. If such an ancestor exists, then make m_i a child of this ancestor node. If it does not, then start a new tree with m_i as a root. In either case, mark m_i as an FOS node—if it is also a PI, then it must be marked as a PI leaf also.

The above procedure is continued until every net of the circuit becomes a node in some tree of the forest.

The forest construction is based on the following properties:

1. The dominance relation is transitive

2. If a FOS net m_i has n_i FOB nets denoted by $m_{i1}, m_{i2}, \ldots, m_{in_i}$, then

(a) $D(m_i) \subseteq D(m_{ij}) \quad \forall j = 1, 2, \ldots, n_i$

(b) $D(m_i) = \bigcap_{j=1}^{n_i} D(m_{ij})$

3. The output net of any gate G is a dominator for every input net of G

The root of any tree in the constructed forest is either a PO or a FOS. If any tree has a single node, then this node must either correspond to a PI which is also a FOS net or a PO which is also a FOB net. The leaves of the trees in the forest correspond to the *checkpoints*, i.e., the PIs and the FOBs. Thus our initial list of target faults consists of all leaves of the trees of the dominator forest and the output of all XOR/XNOR gates [4]. However, in case any of these target faults are undetectable additional target faults must be considered [1,7].

The set $D(m)$ contains all the nodes encountered when traversing the tree (in which m is a node) from m to the root. Recall that $m \notin D(m)$.

The "basis nodes," as defined in TOPS [12], can also be identified easily from the dominator forest. However, keeping in mind that a node cannot be a basis node unless all FOS nets that influence it have completely reconverged prior to it, we adopt a simpler approach of identifying which nodes are NOT basis nodes. Thus, instead of inspecting each node to verify whether it is a basis node or not, we pick one FOS net at a time to generate the set of nodes which are NOT basis nodes. Let there be k FOS nets denoted by m_i, $i = 1, 2, \ldots, k$. Furthermore, let the FOS net m_i have n_i FOB nets denoted by $m_{i1}, m_{i2}, \ldots, m_{in_i}$. The set of nodes which are NOT basis nodes is given by

$$\bigcup_{i=1}^{k} \left[\bigcup_{j=1}^{n_i} [D(m_{ij}) \cup \{m_{ij}\}] - D(m_i) \right].$$

To prove the above assertion consider a net $m_\ell \in \bigcup_{i=1}^{k} \left[\bigcup_{j=1}^{n_i} [D(m_{ij}) \cup \{m_{ij}\}] - D(m_i) \right]$. Thus there must exist i and j, $1 \leq i \leq k$ and $1 \leq j \leq n_i$, such that $m_\ell \in D(m_{ij}) \cup \{m_{ij}\}$ and $m_\ell \notin D(m_i)$. In other words m_ℓ is influenced by a FOS net m_i all of whose fanout branches do not reconverge prior to net m_ℓ. Thus m_ℓ is not a basis node. Conversely if m_ℓ is not a basis node then it must be influenced by some FOS net(s) all of whose fanout branches do not reconverge prior to net m_ℓ.

Tracing back paths from net m_ℓ to the PIs let m_i $(1 \leq i \leq k)$ be the first such FOS net (i.e. those whose branches do not reconverge prior to net m_ℓ) encountered. If there are any other FOS nets between m_i and m_ℓ then they must totally reconverge prior to m_ℓ. Thus there must be a FOB net m_{ij} $(1 \leq j \leq n_i)$ corresponding to the FOS net m_i such that $m_\ell \in D(m_{ij}) \cup \{m_{ij}\}$ and $m_\ell \notin D(m_i)$. Thus $m_\ell \in \bigcup_{i=1}^{k} \left[\bigcup_{j=1}^{n_i} [D(m_{ij}) \cup \{m_{ij}\}] - D(m_i)\right]$.

Explicit evaluation of the above expression is, however, not necessary. We can keep track of the basis nodes while constructing the dominator forest. Recall that we have to identify the immediate ancestor of all the FOB nets corresponding to a FOS net in order to determine the position of the latter in the forest. If such an ancestor exists then all nets, excluding the immediate ancestor, that are encountered when traversing the trees from every FOB net to the immediate ancestor belong to the set of NOT basis nodes. If such an ancestor does not exist, then all nets encountered when traversing the trees from every FOB net to the root of its tree belong to the set of NOT basis nodes. Note that in either case all the FOB nets are also included in this set. Consequently, all nodes not belonging to this set are basis nodes under the assumption that there is no net in the good circuit which has a constant value independent of the PIs.

The dominator forest for the circuit in Fig. 1 is shown in Fig. 2. Note that the only basis nodes for this circuit are the PIs.

2.2 Selection of Primitive D-cube of the Failure

The D-algorithm [14] is initialized by selecting a primitive D-cube of the failure (***pdcf***). However, depending on the nature of the fault (s-a-0 or s-a-1) and the type of faulty gate, there could be more than one ***pdcf***. In such a situation an arbitrary choice has to be made to initialize the algorithm. At this stage we do not want to make any arbitrary choices because they may result in mistaken decisions causing costly backtracking and re-computation. We avoid this problem by introducing a fictitious gate G_f at the site of the fault. If the fault is at net n we introduce G_f

between net n and a newly created net n_f as shown in Fig. 3. We now connect n_f to all nets which were previously connected to n. Accordingly, the unique ***pdcf*** depends only on the kind of stuck-at fault.

		n	n_f
n	s-a-0	1	D
n	s-a-1	0	$\overline{D}$

Thus in our example we will modify Fig. 1 to include the gate shown in Fig. 4.

2.3 Token Assignment

The goal of this stage is to identify which circuit nets can or cannot be affected by a fault. In order to convey this information we associate with every net a boolean token. This token will be TRUE if and only if there exists a path from n_f to any PO which passes through this net. These tokens can be computed by a single forward pass through the circuit. Table 4(a) shows the boolean token assignment for our example.

Consider a gate G (not an XOR/XNOR gate) which lies on the path $\boldsymbol{p_i}$ that we deliberately sensitize. Evidently one input of G, say net m_ℓ, lies on path $\boldsymbol{p_i}$ and must have a sensitized value. If this value is D (or $\overline{D}$) then our deterministic implication procedure would eliminate the value $\overline{D}$ (or D) from the set of values of the other inputs of G. Consider the situation where net m_ℓ has the value $D/\overline{D}$ and the value of another input, say net m_k, of G contains both D and $\overline{D}$. Also let the D and $\overline{D}$ at nets m_ℓ and m_k be due to the value $D/\overline{D}$ at some FOS net m_j that influences both m_ℓ and m_k. Furthermore assume that a D ($\overline{D}$) at net m_ℓ requires a D ($\overline{D}$) at net m_j and that a D ($\overline{D}$) at net m_k requires a $\overline{D}$ (D) at net m_j. Thus net m_k cannot be sensitized. However since the value of net m_j contains both D and $\overline{D}$ we would not be able to arrive at this conclusion using the implication rules alone. This motivates the introduction of the concept of "sensitization parity" which will help us in identifying such relationships amongst the sensitized values of different nets. For ease of explanation we introduce the following definitions:

Definition 1 Net m_j is said to be the **sensitization source** for net m_ℓ with respect to the fault site n if and only if all paths from net n to net m_ℓ pass through net m_j.□

Note that the above definition does not necessarily imply that $m_\ell \in D(n)$ or that $m_\ell \in D(m_j)$.

Definition 2 The **path parity** of a path p_α, not containing any XOR/XNOR gates, is the parity of the number of inverting gates along p_α. □

As far as XOR (or XNOR) gates are concerned, the count of inversions is dependent on the exact inputs and not just circuit structure. Thus the path parity cannot be uniquely determined by circuit structure alone. The concept of path parity was effectively used in [2] for fault simulation purposes.

Definition 3 The **inversion parity** of net m_ℓ with respect to net m_j is b if and only if the path parity of all paths from net m_j to net m_ℓ is b. □

Definition 4 The **sensitization parity** of net m_ℓ with respect to net m_j is b if and only if net m_j is a sensitization source for net m_ℓ and the inversion parity of net m_ℓ with respect to net m_j is b. □

Let us consider again the gate G (not an XOR/XNOR gate) which lies on the path p_i that we deliberately sensitize. As before let the value of the input net m_ℓ of gate G that lies on path p_i be $D/\overline{D}$ and the value of another input, say net m_k, contain both D and $\overline{D}$. If the sensitization parity of net m_ℓ with respect to net m_j is b and that of m_k with respect to net m_j is $\bar{b}$ then net m_k cannot be sensitized when we are trying to sensitize path p_i. Hence we can eliminate both D and $\overline{D}$ from the value of net m_k. Note that we have excluded G to be an XOR/XNOR gate because the output of such a gate is sensitized only if it has an odd number of sensitized inputs which may include both D and $\overline{D}$.

In order to take advantage of the information provided by the sensitization parity we introduce the concept of a token vector of the form $[m, b]$. If the token vector of net m_h is $[m_j, b]$, then b is the sensitization parity of net m_h with respect to net m_j. To explain the assignment of token vectors we divide gates (which are not XOR/XNOR gates) which have at least one input with a token vector and whose output token

vector has not been assigned into two categories:

(i) Type I: gates for which all inputs with the TRUE token have token vectors.

(ii) Type II: gates for which there is at least one input with a TRUE token but no token vector.

In our procedure the token vector of the output net of a gate will be defined in terms of the input token vectors only when all the inputs with the TRUE token have identical token vectors. Otherwise we will restart the sensitization parity count at the output net.

Thus the following rules will be used for assigning token vectors for outputs of Type I and Type II gates:

($R1$) If all the input token vectors of a Type I gate G are identical (say, $[m, b]$), then its output is assigned $[m, b]$ if G is noninverting or $[m, b \oplus 1]$ if G is inverting.

($R2$) If a gate G is of Type II or it is a Type I gate such that all its input token vectors are not identical, then the output net m_g of G is assigned $[m_g, 0]$.

Accordingly, the algorithm for assigning token vectors consists of the following steps:

Step 1: For every XOR/XNOR gate that has a TRUE token at the output find (using the dominator forest) the first FOS net (say net m_s), if any, that this XOR/XNOR gate influences.

Step 2: The token vector for every net m_s generated by Step 1 is set to $[m_s, 0]$.

Step 3: If a net which was assigned a token vector in the previously executed step is an FOS net, then all its FOB nets are assigned the same vector.

Step 4: If there exists a gate of Type I, then assign its output token vector according to $R1$ or $R2$ (as appropriate) and go to Step 3. Otherwise, continue.

Step 5: If there exists a gate of Type II, then assign its output token vector according to $R2$ and go to Step 3. Otherwise, the assignment is complete.

The token vectors generated by the above algorithm, for our example, are given in Table 4(b).

Note that when the above procedure terminates (termination is due to the finite-

ness of the number of gates), all the gates (not XOR/XNOR gates) whose output nets do not have an assigned token vector are those for which no input has an assigned token vector.

There are two ways in which token vectors can provide information which a deterministic implication alone may not. If several inputs of a gate G (not an XOR/XNOR gate) have identical token vectors, then we may simultaneously sensitize any number of these inputs. Furthermore, we can never simultaneously sensitize two inputs of G whose token vectors differ only in their second component.

Note that we initialize the algorithm by assigning token vectors to only the first FOS net that is influenced by an XOR/XNOR gate which has a TRUE token. This is because only an XOR/XNOR gate can introduce a $D/\overline{D}$ on the sensitized path and until we reach a FOS net the question of comparing the sensitization parity of two nets with respect to a common net does not arise.

We will show in Appendix A that our algorithm for assigning token vectors satisfies the following properties:

Property 1 If the proposed algorithm assigns the token vector $[m_j, b]$ to net m_ℓ then b is the sensitization parity of net m_ℓ with respect to net m_j. □

Property 2 If the sensitization parity of net m_ℓ with respect to net m_j is b_1 and the algorithm assigns the token vector $[m, b]$ to net m_j then it would assign the token vector $[m, b \oplus b_1]$ to net m_ℓ. □

Not all the token vectors generated by the above procedure will be useful—however their computation was necessary in order to compute the useful token vectors. The token vector of a net m_1 may be useful only if it is the input to a gate G (where G is not an XOR/XNOR gate) which has at least one more input, say net m_2, such that the first component of the token vectors of m_1 and m_2 are identical. Accordingly the token vector of any net that does not satisfy the above condition can be deleted.

The remaining token vectors after this deletion, for our example, is shown in Table 4(c).

3 Propagation Phase

In this phase we sensitize a single path from net n_f to a PO, however, other paths may also get sensitized. In a manner analogous to DALG [14] we use test cubes whose entries reflect the current values of all nets during any stage of test generation. The entries of any test cube, $\boldsymbol{tc}_k$, are elements of our 15-valued system.

We initialize this phase by constructing $\boldsymbol{tc}_1$ in the following manner:

1. Set nets n and n_f to the values specified by the $\boldsymbol{pdcf}$.
2. Assign $D/\overline{D}$ to all nets belonging to the set $D(n)$.
3. Set all nets with FALSE tokens, except net n, to $0/1$.
4. Assign $0/1/D/\overline{D}$ to all unassigned nets of the test cube.

In our example $D(3) = \{31, 36, 45\}$, and the resulting $\boldsymbol{tc}_1$ is given in Table 5 where only nets whose entries are different from $0/1$ and $0/1/D/\overline{D}$ are shown.

For each test cube $\boldsymbol{tc}_k$ generated at any stage of our algorithm we find its corresponding "deterministic" test cube, $\boldsymbol{d}(\boldsymbol{tc}_k)$. We define a $\boldsymbol{d}(\boldsymbol{tc}_k)$ as one in which no entry can be changed without making some arbitrary choice(s) in one or more net values. That is, all unique implications of the net values must be considered. Rules for forward and backward implication procedures to be used in constructing $\boldsymbol{d}(\boldsymbol{tc}_k)$ from $\boldsymbol{tc}_k$ are given in Appendix B. If in any $\boldsymbol{d}(\boldsymbol{tc}_j)$ we have a sensitized path $\boldsymbol{p}_i$ from the fault site to any PO, then the enumeration phase is invoked. This test cube, $\boldsymbol{d}(\boldsymbol{tc}_j)$, is denoted as $\boldsymbol{T}_f(\boldsymbol{p}_i)$. The $\boldsymbol{d}(\boldsymbol{tc}_1)$ for our example is shown in Table 5. Only the entries for nets whose values are different from those in $\boldsymbol{tc}_1$ are listed. In fact, for each cube listed in Table 5 only the entries whose values are different from those in the preceding one are shown.

If $\boldsymbol{d}(\boldsymbol{tc}_1)$ cannot be constructed because contradictions were encountered, then there exists no test for the fault. Otherwise we have a sensitized path from n_f to all the FOB nets corresponding to the first FOS node (could be n itself!) encountered in traversing the appropriate tree of the dominator forest from n to the root. If there is

no FOS encountered, then we have a sensitized path from n_f to the PO corresponding to the root of the tree. In our example, since net 3 is an FOS we have sensitized paths only until its FOB nets, i.e., 14, 15, and 16.

At this point we have to select one of the FOB nets, say m_1, to extend the sensitized path. To obtain $\boldsymbol{tc}_2$ we should sensitize all nets belonging to the set $D(m_1)-D(n)$ by intersecting their values in $\boldsymbol{d}(\boldsymbol{tc}_1)$ with $D/\overline{D}$. If any empty intersection results, then the sensitized path cannot be extended through m_1 and alternate paths should be investigated. Note that this step is implicitly performing the equivalent of the X-path check [10] while setting up which gate outputs should be sensitized. As stated earlier, we would then construct $\boldsymbol{d}(\boldsymbol{tc}_2)$. If contradictions occur while constructing $\boldsymbol{d}(\boldsymbol{tc}_2)$, then an alternate path must be selected. Otherwise we have a sensitized path from n_f to at least the FOB nets corresponding to the next FOS net or some PO. Assume that we extend the sensitized path in our example through net 16. We use $D(16) - D(3) = \{21\}$ to construct the $\boldsymbol{tc}_2$ and $\boldsymbol{d}(\boldsymbol{tc}_2)$ shown in Table 5. We now have sensitized paths till the FOB nets 37, 38 and 39.

The process of extending the sensitized path by selecting a FOB net, constructing a $\boldsymbol{tc}_k$ and its corresponding $\boldsymbol{d}(\boldsymbol{tc}_k)$ is continued until we reach some PO and have constructed $\boldsymbol{T}_f(\boldsymbol{p}_i)$. If contradictions occur, then alternate paths should be investigated. If all possible paths give contradiction, then no test exists. Note that all possible single paths need not be explicitly investigated to arrive at this conclusion—for example, if all paths from net n to any net $m \in D(n)$ gives contradictions, then we can conclude that no test exists. Proceeding with our example, let us extend the sensitized path through net 39. Since $D(39) - D(36) = \{42, 43\}$, the $\boldsymbol{tc}_3$ shown in Table 5 results. Since the token vectors of nets 40 and 43 are [36,0] and [36,1], respectively, these nets can never be simultaneously sensitized. Thus net 40 must be set to 1. However, the attempt to construct $\boldsymbol{d}(\boldsymbol{tc}_3)$ fails as shown in Table 5. Thus we go back to $\boldsymbol{d}(\boldsymbol{tc}_2)$ and choose another path—say through net 37. The resulting $\boldsymbol{tc}_4$ and $\boldsymbol{d}(\boldsymbol{tc}_4)$ are shown in Table 5. Note that we could set the required value 1 at net 43 in $\boldsymbol{d}(\boldsymbol{tc}_4)$ only because of the use of token vectors. We now have a sensitized path (say $\boldsymbol{p}_1$) from 3_f to a PO,

and thus $d(tc_4)$ is $T_f(p_1)$.

Note that $T_f(p_i)$ represents all the constraints that *must* be imposed to sensitize path p_i. Since the backward implication rule does not make any arbitrary choices, there may be gates where the output value is a proper subset of the value implied by the input values, i.e., the input values include combination(s) that will desensitize path p_i. In view of this fact we introduce the following definition. If, in a deterministic test cube $d(tc_k)$, the value of the output net m of a gate G is a proper subset of the value implied at the output by the input values, in $d(tc_k)$, of G then net m is said to be a **variant** net in $d(tc_k)$. If a net is not variant it is defined to be **invariant** in $d(tc_k)$. In our example the only variant net in $T_f(p_1)$ is net 30.

If all the nets in the circuit are invariant nets in $T_f(p_i)$ then the specified primary inputs in $T_f(p_i)$ represent all the requirements that must be satisfied by any input pattern that detects the fault f by sensitizing path p_i. In general, however, not all nets in $T_f(p_i)$ will be invariant. In such a situation there exists an assignment of the unspecified primary inputs (i.e., inputs with the 0/1 value) in $T_f(p_i)$ which will desensitize path p_i. In order to obtain a test from $T_f(p_i)$ we must convert all variant nets to invariant ones by specifying one or more of these primary inputs. Moreover, the new deterministic test cube obtained by specifying these primary inputs in $T_f(p_i)$ should result in net values that are subsets of their corresponding values in $T_f(p_i)$ for all the nets of the circuit. This condition is required to prevent the setting of primary inputs in such a way as to result in a disallowed value at a net that was variant in $T_f(p_i)$.

Example 1. Consider the circuit of Fig. 5 with a s-a-0 fault at net 3. Denote by p_1 the unique path from the site of the fault to the primary output. The resulting $T_f(p_1)$ is shown below.

$T_f(p_1)$:

1	2	3	3_f	4	5
0/1	0/1	1	D	0	D

Note that net 4 is a variant net in $T_f(p_1)$. If the unspecified primary inputs in $T_f(p_1)$ (i.e., nets 1 and 2) are both set to 1, then the path from the fault site to the

output is desensitized. □

We re-emphasize that conversion of variant nets to invariant ones will always involve some arbitrary choice(s). This is because all deterministic choices have been taken care of by the forward and backward implication rules. Different approaches can be adopted to make choices that will convert all variant nets in $T_f(p_i)$ to invariant ones with values that are subsets of the corresponding net values in $T_f(p_i)$ for all the nets of the circuit, provided there exists an input pattern that sensitizes path p_i. In this report we will follow an enumeration procedure, to be discussed in the next section, to make these choices in order to derive a test. In Appendix G we will present an alternate procedure which is similar to line justification of DALG [14].

4 Enumeration Phase

The goal of this phase is to obtain a test by specifying the unassigned PIs in $T_f(p_i)$ such that all nets are invariant in the resulting deterministic test cube and have values that are subsets of their corresponding values in $T_f(p_i)$.

We choose an unspecified primary input I_{l_1} in $T_f(p_i)$ and assign a logic value (0 or 1) to it, thereby creating a new test cube which we denote by $tc_f(p_i, 1)$. Now we find its corresponding deterministic test cube $d(tc_f(p_i, 1))$ and update its list of variant nets (note that new variant nets may be created). However if $d(tc_f(p_i, 1))$ cannot be obtained due to some contradiction, then we complement the entry for I_{l_1} in $tc_f(p_i, 1)$ and construct its corresponding $d(tc_f(p_i, 1))$. If this also leads to a contradiction, then there exists no test corresponding to $T_f(p_i)$. If we are successful in constructing $d(tc_f(p_i, 1))$, we assign a logic value to some other unspecified primary input I_{l_2}, thereby creating $tc_f(p_i, 2)$. As before we must construct $d(tc_f(p_i, 2))$ and update its list of variant nets. This procedure is continued and we traverse the decision tree, in a manner analogous to PODEM [10], until one of the following two conditions occur:

- The list of variant nets corresponding to some $d(tc_f(p_i, j))$ becomes empty.

- The decision tree is exhausted, i.e. no test exists.

If the procedure is terminated because the former condition is satisfied, then the values of the PIs in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{j}))$ represent test(s) for the fault. To derive test patterns for the fault we would then assign either 0 or 1 to those PIs in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{j}))$ which have the value 0/1.

We now continue with our example for the fault net 3 $s-a-0$ in the circuit of Fig. 1. As stated earlier, net 30 is the only variant net in $\boldsymbol{T}_f(\boldsymbol{p}_1)$. By inspecting the dominator forest we notice that nets 7 and 8 are the PIs which are "closest" to net 30. We thus start by setting net 7 to 0—however, this does not change the value of any other net. We continue by setting net 8 to 0—once again no new changes result. We now use the dominator forest to reach the FOS net 24 and thus determine that nets 2 and 5 are the next "closest" PIs. We could, for example, set net 2 to 0—the only resulting change is a $0/D$ at net 22. Net 30 is still the only variant net, so we now set net 5 to 0. This changes the value of net 23 to 0 and that of nets 24, 25, 26, 27, and 28 to $0/D$. Also, all nets are verified to be invariant and have values that are subsets of their values in $\boldsymbol{T}_f(\boldsymbol{p}_1)$, thus a test has been generated.

Since the conversion of variant nets to invariant ones is the key to generating a test from $\boldsymbol{T}_f(\boldsymbol{p}_i)$ it is useful to keep track of nets which are variant in the process of constructing $\boldsymbol{T}_f(\boldsymbol{p}_i)$. This would avoid the unnecessary checking of every net as variant or invariant after $\boldsymbol{T}_f(\boldsymbol{p}_i)$ has been constructed. Note that if a net is invariant at some stage of generating a test for a fault it will not become variant unless a new backward implication (with a value which is a proper subset of the existing value) is made for the net.

We will show in Appendix C that the test generation procedure presented in this report is an algorithm.

The algorithm described so far can be substantially improved by the introduction of several speed-up techniques which we discuss in the next section.

5 Speed-up Techniques

5.1 Use of the Contrapositive

In this subsection we will discuss how checking for contrapositive assertions during the pre-processing phase can be used as an effective speed-up technique. The use of the contrapositive to reduce the search space was first suggested by Schulz, et. al., in SOCRATES [15]. However, the procedure presented in SOCRATES can only be used to backtrack the value 0 or 1. We present here a procedure that can not only generate the contrapositive assertions for all 15 values of our system, but also requires less computation and storage than SOCRATES. The contrapositive of the logic expression $P \Longrightarrow Q$ is the equivalent expression $\sim Q \Longrightarrow \sim P$. Referring to the circuit of Fig. 6 we notice that $X_3 = 0 \Longrightarrow Z = 0$. Hence the contrapositive would yield $Z = 1 \Longrightarrow X_3 = 1$. However, if we require the value 1 at Z given that all other nets have the value $0/1$, no deterministic change would be implied by the backward implication procedure alone. Note, however, that in some cases a backward implication will yield the information provided by the contrapositive property. For example, $X_3 = 0 \Longrightarrow Y_4 = 0$ yields $Y_4 = 1 \Longrightarrow X_3 = 1$. However, a backward implication of $Y_4 = 1$ yields a 1 at X_3, X_4, and X_5. Hence it is useful to identify the conditions under which a backward implication cannot yield the information provided by a contrapositive assertion. In such cases we may store this information for possible use later in the test generation process.

In our 15-valued system, assume that the forward implication of a value L_1 at net m_1 with $0/1/D/\overline{D}$ at all other nets yields the value L_2 at net m_2. Thus when we require a value $L_2' \subseteq((0/1/D/\overline{D})-L_2)$ at net m_2, then the value of net m_1 cannot contain any element of L_1. To obtain the implications for all possible values of L_1 we only need to perform implications for each individual element of $0/1/D/\overline{D}$. Thus the procedure to obtain the implications for the 15-valued system, henceforth referred to as **15-VP**, would be to set the value of net m_1 to each of the values 0, 1, D and $\overline{D}$, one at a time and with $0/1/D/\overline{D}$ at all other nets, and observe the implied value at

net m_2. Each such implication is referred to as a **15-VP** "experiment." We will show that the information yielded by **15-VP** can be obtained from a simpler procedure that utilizes a 3-valued $(0, 1, 0/1)$ logic system. In this procedure, which we denote as **3-VP**, we set the value of net m_1 to each of the values 0 and 1, one at a time and with 0/1 at all other nets, and observe the implied value at net m_2. Each such implication is referred to as a **3-VP** experiment. For ease of explanation we define the values 0 and 1 as "singleton" values. Table 6 shows the nine possible combinations of values obtained by **3-VP** at net m_2 when the values 0 and 1 are applied at net m_1. Note that cases (ii) and (iii) show that net m_2 has a constant value independent of the circuit inputs. As a consequence, at least one of the stuck-at faults at net m_2 is undetectable. Cases (iv) and (v) simulate a wire and an inverter between nets m_1 and m_2, respectively.

The following theorems, whose proofs appear in Appendix F, establish the relationship between the results of a **3-VP** experiment and the corresponding **15-VP** experiment.

Theorem 1 If a **3-VP** experiment yields a singleton value at net m_2, then the corresponding **15-VP** experiment yields the same singleton value at this net.

□

Theorem 2 If a **3-VP** experiment yields the value 0/1 at net m_2, then the corresponding **15-VP** experiment yields the value $0/1/D/\overline{D}$ at this net.

□

Consequently Table 7 is obtained from Table 6 when a 0 and 1 implication is performed in **15-VP**. We now show that the information yielded by **15-VP** can be obtained from that yielded by **3-VP**. We do this by illustrating how Table 7 can be used to obtain the implications due to a D or a $\overline{D}$ at net m_1. Note that a D at net m_1 corresponds to a change in value of net m_1 from a 1 to a 0. Thus, to obtain the implied value at net m_2 due to a D at net m_1 we only need to know the value at net m_2 due to a 1 and a 0 at net m_1 in **15-VP**.

Consider the situation where a 1 and a 0 at net m_1 yields a 0 and $0/1/D/\overline{D}$, respectively, at net m_2. Thus, with a 0 at net m_1 we can obtain both 0 and 1 at net m_2. Therefore, with a D at net m_1 we can only obtain a 0 or a $\overline{D}$ at net m_2. Under the same conditions, by similar reasoning, a $\overline{D}$ at net m_1 can only yield a 0 or a D at net m_2. Analogously, we can inspect the other cases to generate the implications of a D or $\overline{D}$ at net m_1. Table 8 summarizes our results and shows how the **15-VP** table can be obtained from the **3-VP** table.

We now discuss the conditions under which the information yielded by a contrapositive assertion cannot be obtained by a deterministic backward implication alone and hence should be stored for future use. We will first derive these conditions for the **3-VP** implications and then discuss what additional information from the **3-VP** experiments need to be stored in order to obtain the contrapositive assertions for our 15-valued system.

Consider the situation where a singleton value L_1 at net m_1 yields, using **3-VP**, a singleton value L_2 at the output net m_2 of a gate G. The corresponding contrapositive assertion should be stored if and only if the value L_2 can be obtained at the output of G by setting all its inputs to non-controlling values. Consequently, Table 9 shows the L_2 and G combinations for which this implication should be stored for future use. In general, for the cases that satisfy the (L_2, G) combinations given in Table 9 we will not be able to drop L_1 from the set of all possible values at net m_1 when we require the value $\overline{L}_2$ at net m_2 by using only the backward implication procedure. We now show that the stored contrapositive assertions in **3-VP** and the deterministic backward implications rules would yield all the other contrapositive assertions in **3-VP**.

From the given condition if a **3-VP** implication is not stored, then the value L_2 can be obtained at the output of G by setting at least one of its inputs (say net m_j) to a controlling value. From Lemma 3 of Appendix F we know that this **3-VP** experiment creates a path (say p_1) of singleton values from net m_1 to net m_j. Furthermore, if the singleton value of any net on this path is a non-controlling value for the gate G_i it drives, then this experiment sets all inputs of G_i to non-controlling values. Let us

consider the two possible ways this could happen:

(i) in path $\boldsymbol{p}_1$ all nets have values that are controlling values for the gates they drive. Thus when we require $\overline{L}_2$ at net m_2 we can, by backward implication, change the value of all these nets to non-controlling values for the gates they drive. This would set the value of net m_1 to $\overline{L}_1$.

(ii) In path $\boldsymbol{p}_1$ there exists gates G_i such that the value L_1 at net m_1 sets all inputs of these gates to non-controlling values. Thus using Table 9 the implication of the value obtained at the output of G_i due to the value L_1 at net m_1 would be stored for future use. When we require $\overline{L}_2$ at net m_2 we can, by backward implication, change the value of all the nets on path $\boldsymbol{p}_1$ to non-controlling values for the gates they drive until we reach one of the gates G_i. However, we can use the contrapositive assertion from the stored implication for gate G_i to conclude that net m_1 must be set to $\overline{L}_1$.

We now present a procedure which, when incorporated into the pre-processing phase, can derive all the contrapositive assertions for our 15-valued system.

1. Construct two test cubes $\boldsymbol{tc}_{00}$ and $\boldsymbol{tc}_{01}$ in which the values of all nets of the circuit are set to 0/1.

2. In $\boldsymbol{tc}_{00}$ ($\boldsymbol{tc}_{01}$) change the value of net m_1, where m_1 is a FOS net , to the singleton value $L_1(\overline{L}_1)$ and perform a forward implication of this value.

   Let L_2 (L_3) be the implied value at the output net m_2 of gate G.

3. If both L_2 and L_3 are singleton values, then both these implications (L_1 at $m_1 \Longrightarrow L_2$ at m_2 and $\overline{L}_1$ at $m_1 \Longrightarrow L_3$ at m_2) need to be stored.

4. If only one of the values (say L_2) is singleton and this value L_2 and the gate G happen to be one of the combinations listed in Table 9, then this implication (L_1 at $m_1 \Longrightarrow L_2$ at m_2) should be stored.

5. Repeat steps 1–4 for all FOS nets.

The "learning procedure" presented in SOCRATES [15] performs the 0 and 1 implications for all nets of the circuit. However, we have reduced the amount of

computation and storage requirements by performing the implications for only FOS nets. It is easy to show that the information for all other nets can be derived from this because of the deterministic nature of our backward implication procedure.

The contrapositive assertions in the 15-valued system corresponding to the implications stored by the above procedure can be generated using Table 8. So, it has to be shown that if any implication was not stored by the above procedure, then either its corresponding contrapositive assertion yields no information or the information yielded can be derived by using the stored contrapositive assertions and the backward implication rules. The former situation refers to the trivial case when both L_2 and L_3 are $0/1$. The latter situation refers to the case where only one of L_2 or L_3 (say L_2) is singleton and (L_2, G) is not one of the combinations listed in Table 9. We will prove this by discussing one possible (L_2, G) combination—all other cases can be proved analogously.

Consider the particular case where a 0 at net m_1 implies the value 0 at the output net m_2 of an AND gate G and a 1 at net m_1 implies the value $0/1$ at net m_2. Thus if we had stored this implication we could have generated the implications (D at net $m_1 \Longrightarrow 0/D$ at net m_2) and ($\overline{D}$ at net $m_1 \Longrightarrow 0/\overline{D}$ at net m_2) using Table 8 (see Case (vi)). Consequently, if we had, for example, required the value $1/\overline{D}$ at net m_2, then by the contrapositive of the assertions (0 at net $m_1 \Longrightarrow 0$ at net m_2) and (D at net $m_1 \Longrightarrow 0/D$ at net m_2) we would update the set of values L of net m_1 to $L' = (L - (0/D))$. We will show that L' can be obtained by the stored contrapositive assertions and the backward implication rules. Since the required value at net m_2 does not include 0, then from Theorem 1 and the argument presented to justify what values should be stored by **3-VP** we would eliminate the value 0 from the set L. Note that a $\overline{D}$ at net m_2 implies a change in the value of net m_2 from 0 to 1. This value of 1, by the stored contrapositive assertions and the backward implication rules, will imply that the value of net m_1 cannot include a 0 or a change to 0. Thus the values 0 and D (which represent a change from 1 to 0) will be eliminated from the set of values of net m_1 by the backward implication procedure and the stored contrapositive

assertions. Hence we will obtain $L' = (L-(0/D))$.

5.2 Conditional Headlines

TOPS [12] extended the concept of headlines introduced in FAN [8] by using circuit topology to identify more nodes whose value justification can be postponed until the last stage of test generation because they are guaranteed not to cause any contradictions. However, none of these schemes take advantage of the additional restrictions imposed by a particular fault. These restrictions might identify a potentially larger set of circuit nets whose value justification may also be postponed.

Let the output net m_1 of a gate G have the value 0/1 or be a variant net with a singleton value in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$ (a deterministic test cube obtained at some stage of the enumeration phase). Consider the tree T which is a subgraph of the dominator forest and whose root is net m_1. From T construct another tree T_1 by deleting all the subtrees of T whose roots m_k, $m_k \neq m_1$, correspond to FOS nets. Note that T_1 corresponds to the largest fanout-free subcircuit whose output is net m_1 and whose inputs are FOBs and/or PIs. Net m_1 is defined to be a **conditional headline** if and only if all the nets corresponding to the FOB nodes in T_1 have singleton values in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$.

We now show that if net m_1 is a conditional headline, then it can be set to either of the singleton values 0 or 1, subject to the condition that the values, as required by $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$, of all the FOB nodes in T_1 can be satisfied. Note that if m_1 is a conditional headline, then T_1 satisfies the following properties:

(i) The values in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$ of the nodes in T_1 can only be 0, 1, or 0/1. This is because if the value of any node includes either D or $\overline{D}$, then this must be due to a fault at node m_ℓ which belongs to T_1 since all FOB nodes have singleton values. But $m_1 \in D(m_\ell)$, and hence m_1 would have a sensitized value.

(ii) At least one leaf of T_1 is a PI net whose value in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$ is 0/1. This is because either m_1 is a variant net or has the value 0/1.

Consider a node m_j in T_1 which has a singleton value and whose parent node has

the value 0/1 in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$. Note that the value of m_j is a non-controlling input value for the gate G_j it drives since $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$ is a deterministic test cube. If we delete from T_1 all the subtrees which have any such m_j as a root, then the remaining tree corresponds to a fanout free circuit whose output is net m_1 and whose inputs have the value 0/1 in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$. (Note that if G_j is an XOR/XNOR gate, then we must complement the function performed by G_j if an odd number of inputs of G_j with the value 1 were deleted from T_1.) Thus any required singleton value of net m_1 can be satisfied by specifying the unassigned PIs in T_1 subject to the condition that the values of the FOB nets in T_1 can be satisfied. Note that this assignment does not interfere with the requirement of other variant nets since m_1 is a dominator for all these PIs.

5.3 Backtracking Desensitizing Values

In this subsection we discuss how backtracking the desensitizing value from variant nets may help speed-up the enumeration process. Consider the output net m_1 of a gate G_1 which has the value L_1 and is a variant net in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$, a deterministic test cube obtained at some stage of the enumeration phase. Let L_1' be the value implied at net m_1 by the values in $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$ of the inputs of G_1. We construct a new test cube $\boldsymbol{d}'(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$ which is identical to $\boldsymbol{d}(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$ except that net m_1 has the value $L_1' - L_1$. Note that the value $L_1' - L_1$ at net m_1 desensitizes path $\boldsymbol{p}_i$. Using $\boldsymbol{d}'(\boldsymbol{tc}_f(\boldsymbol{p}_i, \boldsymbol{k}))$ we backtrack the value $L_1' - L_1$ at net m_1 by applying only the backward implication rules and the stored contrapositive assertions and observe the nets whose values change in the process. Let m_j, $2 \leq j \leq J$, be the nets where this backtracking terminates. Note that m_j is either a PI or the output of a gate whose input values do not change during this process. Also, let $L_j', 2 \leq j \leq J$, be the new value obtained at net m_j by the above procedure.

Since the value $L_1' - L_1$ at net m_1 implies that the value of net m_j is L_j', we know from the contrapositive principle that, for *any* j, $2 \leq j \leq J$, if the value of net m_j does not contain any of the values in the set L_j', then the value at net m_1 will not

contain any of the values in the set $L_1' - L_1$ and hence m_1 will become an invariant net. A sufficient condition to make m_1 an invariant net without interfering with the requirements of other variant nets is that there exists some m_j such that $m_1 \in D(m_j)$ and m_j is a basis node. If m_j is not a basis node but is a conditional headline in $d(tc_f(p_i, k))$, then net m_1 can still be made invariant by removing the value L_j' from the set of values of net m_j, provided the conditions that make net m_j a conditional headline are satisfied.

5.4 Selection of Alternate Sensitizing Paths

Suppose that it is not possible to generate a test from $T_f(p_i)$. This means that there exists no test for the given fault f that sensitizes path p_i. We now have to select some $T_f(p_j)$, if any exists, to generate a test for the fault. The following theorem may be helpful in deciding whether there exists a test for the fault that sensitizes path p_j.

Theorem 3 . *If the value of every net in $T_f(p_j)$ is a subset of the corresponding value in $T_f(p_i)$ (denoted by $T_f(p_j) \subseteq T_f(p_i)$) and there is no input pattern that sensitizes path p_i, then there is no input pattern that sensitizes path p_j.*

Proof. (By contradiction.) Assume that there is an input pattern I that sensitizes path p_j. Since $T_f(p_j)$ imposes only those restrictions that must be satisfied by any input pattern that sensitizes path p_j, then values of all the nets of the circuit in the presence of input I must be subsets of their corresponding values in $T_f(p_j)$. Since $T_f(p_j) \subseteq T_f(p_i)$, then the values of all the circuit nets in the presence of input I must be subsets of their corresponding values in $T_f(p_i)$. Thus I satisfies all the restrictions imposed by $T_f(p_i)$ and hence it sensitizes path p_i, which is a contradiction. □

6 Examples

Example 2. Let us reconsider the circuit of Fig. 1 with the fault net 3 $s-a-0$ to highlight the improvements obtained by the speed-up techniques. The resulting test cubes are shown in Table 10. Notice that $\boldsymbol{tc}_1$ is identical to that in Table 5. However, the new $\boldsymbol{d}(\boldsymbol{tc}_1)$ is different because the use of the contrapositive assertion and the value of net 30 drops the value 1 from net 24 which has further deterministic implications. When the sensitized path is extended through net 16 further use of the contrapositive drops $\overline{D}$ from the value of net 24 and more deterministic changes occur. Since net 42 has the value 1 in $\boldsymbol{d}(\boldsymbol{tc}_2)$ the attempt to sensitize the path through net 39 leads to a contradiction in the construction of $\boldsymbol{tc}_3$ and the unnecessary computation for the construction of $\boldsymbol{d}(\boldsymbol{tc}_3)$ is avoided. As before, we now extend the sensitized path through net 37 and obtain $\boldsymbol{d}(\boldsymbol{tc}_4)$ as shown. The only variant net is net 30 and its desensitizing value is 1. Using the procedure explained in §**5.3** we backtrack the value 1 at net 30 to get the value 1 at nets 7 and 8. Since both nets 7 and 8 are basis nodes and $30 \in (D(7) \cap D(8))$, then removing the value 1 from either net 7 or 8 would give a test for the fault. □

Example 3. Consider the class of circuits shown in Fig. 7 with the fault net 3 $s-a-0$. Note that the ECAT circuit considered by Goel to illustrate the efficiency of PODEM [10] is an element of this class. Using $D(3) = \{5, 7\}$ we construct $\boldsymbol{tc}_1$ as shown below where all other nets have the value 0/1.

1	2	3	3_f	4	5	6	7
0/1	0/1	1	D	0/1	$D/\overline{D}$	0/1	$D/\overline{D}$

The deterministic test cube $\boldsymbol{d}(\boldsymbol{tc}_1)$, corresponding to $\boldsymbol{tc}_1$, is then constructed and the values for nets 1 through 7 are shown below. The values of all other nets remain unchanged at 0/1.

1	2	3	3_f	4	5	6	7
1	1	1	D	0/1	$D/\overline{D}$	0/1	$D/\overline{D}$

The only nets whose values change in the construction of $\boldsymbol{d}(\boldsymbol{tc}_1)$ are 1 and 2. Since we have a sensitized path from 3_f to the PO and there are no variant nets, a test has been generated. Note that the algorithm specifies only the value of PI nets 1 and 2 because it takes full advantage of the linearity of XOR gates. □

Example 4. In this example we illustrate the use of conditional headlines. Consider the circuit in Fig. 8 whose only basis nodes, other than the PO, are the PIs. The only possible $\boldsymbol{T}_f(\boldsymbol{p}_1)$ for the fault net 2 $s-a-0$ is shown below where all other nets have the value 0/1.

1	2	2_f	3	11	12	23	32	34	35
1	1	D	1	1	1	$\overline{D}$	D	0	D

Net 34 is the only variant net and its desensitizing value is 1. Thus we backtrack the value 1 from net 34 which sets nets 31 and 33 to the value 1. The use of the contrapositive sets the value of nets 20 and 24 to 1. It can now be verified, using the procedure of §**5.2**, that net 24 is a conditional headline and net 20 is not. Furthermore, the only condition required for net 24 to have the same independence property as a headline is that the value 1 at net 12 be satisfied. This condition is already met because net 3 is a PI. Thus the required value 0 at 24, which makes net 34 an invariant net with the value 0, can be met by specifying the PI nets 4, 5 and 6. □

7 Conclusions

We have presented a 15-valued ATPG algorithm that introduces several new concepts to make test generation more efficient. It is the only algorithm that takes into account all the deterministic implications of sensitizing a path prior to the enumeration process. The resulting ability to identify inconsistencies prior to enumeration improves the possibility of quicker identification of redundant faults. Instead of sensitizing a single gate at a time, we sensitize subpaths by sensitizing all gates lying between successive FOS nets, thereby reducing the number of times deterministic test cubes have to be constructed. Our algorithm exploits the linearity of the XOR/XNOR gate and

also shows how use of the sensitization parity can speed-up test generation for circuits containing such gates. The speed-up techniques introduced are based on both circuit topology and the net values that must be satisfied to sensitize the chosen path. We have also shown how use of the desensitizing values can guide the selection of PIs in the enumeration phase. The contrapositive assertions for our 15-valued system was obtained from a simple procedure that uses a 3-valued system and performs implications for FOS nets only. We emphasize that the different aspects of the algorithm discussed above owe their efficiency to the strength of the 15-valued system used. We have also shown how the dominator forest of a circuit can be effectively used in several phases of test generation.

Since we allow $D/\overline{D}$ as a possible sensitized value we can perform sensitization of subpaths starting from different FOS nets in parallel. Parallelism can also be exploited in the enumeration phase because the conditional headlines will identify subcircuits that can be processed independently.

Acknowledgment

The authors would like to thank Elaine Weinman for her help in the preparation of this manuscript. Thanks are also due to Jonathan Greenfield for his valuable suggestions.

Appendix

A Proof of Properties in §2.3

In this appendix we will prove the properties of the algorithm, for assigning token vectors, that were presented in §2.3.

Proof of Property 1 This property is a consequence of the following features of the algorithm for assigning token vectors:

(i) The token vector of all nets on any path between nets m_j and m_ℓ must have been assigned using $R1$. As a consequence any path from net n to net m_ℓ must pass through m_j.

(ii) Consider a path between nets m_q and m_r such that for all gates lying on this path the only input with a TRUE token is the input that lies on this path. If the token vectors of two nets lying on this path are $[m_t, b_1]$ and $[m_t, b_2]$ respectively then the path parity of the subpath between them is $b_1 \oplus b_2$.

□

Proof of Property 2 Since the sensitization parity of net m_ℓ with respect to net m_j is b and net m_j has an assigned token vector, then all nets on any path from net m_j to net m_ℓ must have TRUE tokens and there exists no XOR/XNOR gate on any of these paths. Thus the algorithm will assign a token vector to net m_ℓ. We now show by contradiction that the algorithm assigns the token vector $[m, b \oplus b_1]$ to net m_ℓ. Assume that the first component of the token vector assigned by the algorithm to net m_ℓ is different from m. Thus there exists at least one gate G on a path from net m_j to net m_ℓ whose token vector at the output is $[m', b']$ where $m' \neq m$ and which has at least one input with a token vector whose first component is m. Thus the algorithm must have assigned the token vector at the output of G using $R2$. This can only happen in one of three ways listed below:

(i) The first component of the token vector of some input of G is different from m.

(ii) There is an input of G which has a TRUE token but no token vector (i.e. the

algorithm identifies G as a Type II gate).
(iii) The algorithm identifies G as a Type I gate in which all inputs with TRUE tokens have token vectors whose first component is m but not all second components are identical.

Case (iii) above implies the existence of two distinct paths, $\boldsymbol{p}_\alpha$ and $\boldsymbol{p}_\beta$, from net m to net m_ℓ with different path parity. Since net m_j is a sensitization source for net m_ℓ, then these paths must pass through net m_j. Since the token vector of net m_j is $[m, b]$, all paths from net m to net m_j must have the path parity. Hence there exists two distinct paths from net m_j to net m_ℓ which have different path parity. This is a contradiction because the sensitization parity of net m_ℓ with respect to net m_j is b.

We now show that Case (ii) cannot occur. Select the input of G which has a TRUE token but no token vector. Thus there exists a path from net m_j to this input such that all nets on this path have TRUE tokens but no token vector. (This is because the assignment of token vectors to output of Type I gates have priority over that for Type II gates.) This is a contradiction because net m_j has a token vector.

Thus it remains to be shown that even Case (i) cannot occur. Select the input of G whose first component of the token vector is different from m. Since we have shown that Case (ii) and (iii) cannot occur, then there must exist a path from net m_j to this input such that the first component of the token vectors of all the nets on this path are different from m. This is a contradiction because the token vector of net m_j is $[m, b]$.

Thus the algorithm assigns m as the first component of the token vector of net m_ℓ. The fact that the algorithm assigns $b \oplus b_1$ as the second component is a consequence of Property 1. □

B Construction of Deterministic Test Cubes

In a $\boldsymbol{d}(\boldsymbol{tc}_k)$ all deterministic implications (no arbitrary choice) of all entries of the test cube $\boldsymbol{tc}_k$ are fully considered. For example, if the output of an AND gate is

$0/1/D/\overline{D}$ and one of its inputs changes to $0/D$, then, irrespective of the other inputs, the output is changed to $0/D$.

To construct $\boldsymbol{d}(\boldsymbol{tc}_1)$ from $\boldsymbol{tc}_1$ we perform backward and forward implications of all nets whose values in $\boldsymbol{tc}_1$ are different from $0/1$ and $0/1/D/\overline{D}$ and all other nets whose values change during this implication process. In the general case, when we are constructing $\boldsymbol{d}(\boldsymbol{tc}_k)$ from $\boldsymbol{tc}_k$, we start by considering the forward and backward implications of the nets whose values in $\boldsymbol{tc}_k$ are different from those in the last successfully constructed deterministic test cube and that of all other nets whose values change during this implication process. During the construction of $\boldsymbol{d}(\boldsymbol{tc}_k)$ from $\boldsymbol{tc}_k$, if a backward or forward implication request results in a new value L_j' for any net m_j of the circuit, then we should update the corresponding net entry L_j by setting it to $L_j \cap L_j'$. If this intersection yields the empty set then $\boldsymbol{d}(\boldsymbol{tc}_k)$ cannot be constructed.

In order to obtain $\boldsymbol{d}(\boldsymbol{tc}_k)$ the process of forward and backward implications should be continued until no more changes occur in the values associated with any net. Note that this process will terminate in a finite number of steps because we are performing set intersection on finite sets.

The rules for constructing deterministic test cubes must include the provision for appropriately handling the values of nets associated with fanout points and should also take into account the information provided by the token vectors. We now present the rules for forward and backward implication.

B.1 Forward Implication

The process of forward implication of the values associated with every net is done with the help of Tables 1, 2 and 3. These tables are a generalization of the truth tables of the respective gates. For gates with more than two inputs the method adopted is similar to that used by Akers [3]. We view every gate as being constructed out of 2 input gates and use the existing values at the inputs of a gate to generate a new value for the output. Depending on the gate in question, appropriate tables are used. Note that the three tables are sufficient because OR, NOR, and NAND functions can be

derived by appropriately using Tables 1 and 2, whereas Tables 2 and 3 can be used to generate the XNOR function.

Suppose we are performing forward implications due to change(s) in input(s) of a gate G whose output is net m. Let L_O be the set of values associated with net m in the test cube prior to forward implication being performed. Also let L_N be the value obtained at net m by using the new values of the inputs of G. Net m will then be set to $L_O \cap L_N$ unless $L_O \cap L_N = \emptyset$ which implies a contradiction. Four other situations are possible:

1. $L_O = L_N$. No further action is needed for this forward implication.

2. $L_N \subset L_O$ (proper subset). We now have to consider the forward implication of the value of L_N at net m on all gates driven by G.

3. $L_O \subset L_N$. We now have to perform a backward implication of the value L_O at net m. This may result in further changes in the inputs of gate G.

   Example 5. An example of the situation where $L_O \subset L_N$ is shown in Fig. 9. If input A of gate G is changed from 0/1 to 1, then forward implication using Table 1 would yield $L_N = 0/1$. Since $L_O \subset L_N$, we now perform a backward implication of the value 0 at the output of gate G. It will be clear from Appendix B.2 that this backward implication yields a 0 at input B. □

4. $L_O \not\subseteq L_N$ and $L_N \not\subseteq L_O$. Both forward and backward implications of the value $L_O \cap L_N$ at net m should be performed.

   Example 6. An example of the situation where $L_O \not\subseteq L_N$ and $L_N \not\subseteq L_O$ can be seen from the incompletely specified circuit of Fig. 10. Assume that at some stage of test generation we have the following $d(tc_k)$.

1	2	3	4	5	6	7	8
0/1	D	D	D	0/1	0/1	0/1	$1/D$

9	10	11	12	13	14	15	16
$0/D$	0/1	$0/D$	$0/D$	$0/1/D$	$0/D$	$0/D/\overline{D}$	$D/\overline{D}$

The value at net 14 is $L_O = 0/D$. If we now extend the sensitized path through net 4 by setting nets 5, 6 and 7 to 1 then forward implication would yield the value $L_N = 1/D$ at net 14. Hence $L_O \cap L_N = D$, $L_O \not\subseteq L_N$ and $L_N \not\subseteq L_O$. □

B.2 Backward Implication

The process of backward implication involves determining the changes required at the inputs of a gate in order to satisfy a requested change at the output. A change in the value of a net will mean that one or more possible values associated with the net has been deleted. In that sense an input change can be made only if the deleted value can never be used with the existing values at the other inputs to generate any of the requested output value(s).

Example 7. Consider a two-input AND gate whose values at inputs and output is $0/D$. If we require that the output be changed to 0, we cannot change any of the inputs because all the input values can be used in some input pattern to generate a 0 at the output. □

A general set of backward implication rules can be derived in terms of the input values and the requested output value. However, in a manner similar to that presented in [3] we consider each multiple input gate as a cascade of two input gates. The backward implication rules for a two-input AND gate is shown in Table 11. Note that the element $\emptyset$ has been included in this table to detect an unsatisfiable backward implication request. The complete table for all 15 values is obtained by the set union operation. The resulting table is equivalent to that proposed by Akers [3]. To perform backward implication for a two-input AND gate we reference the table using the requested value at the output and the existing value at one input to generate the value of the other input. Since the XOR gate is linear, Table 3 can be used for backward implication also. Thus Tables 2, 3 and 11 can be used to perform backward implication for any two-input gate. Irrespective of the gate in question, the value generated by the appropriate table must be intersected with the existing value of the input to generate the new value of the input. Analogously, the new value of the input

and the requested value of the output must now be used to generate the new value of the other input. For example, consider a 2-input gate whose input values are L_1 and L_2. If the requested value of the output of the gate is L_G, then we use L_G and L_1 to determine the new value L_2' of the second input and then L_2' and L_G to determine the new value L_1' of the first input.

Example 8. Consider the two-input AND shown in Fig. 11(a). Initially, input A has the value $0/D/\overline{D}$, input B has the value $0/1/D$, and the output has the value $0/D/\overline{D}$. If we require a D at the output, then the backward implication process using Table 11 and values of C and A would yield a $1/D$ at B. That, intersected with its existing value of $0/1/D$, yields $1/D$. (See Fig. 11(b).) Now a backward implication of a D at C with a $1/D$ at B yields $1/D$ at A. This value of A intersected with the existing value of $0/D/\overline{D}$ results in a D at input A. (See Fig. 11(c).) □

As stated before, any gate with more than two inputs will be represented as a cascade of two-input gates. Consider an n-input gate G represented as a cascade of $(n-1)$ two-input gates G_1, $G_2, \ldots, G_{n-2}$ and G_{n-1}, with net numbers as shown in Fig. 12. Assume that the values at nets $1, 2, \ldots, n$ are $X_1, X_2, \ldots, X_n$ respectively. We first use forward implication of these values to compute $Y_1, Y_2, \ldots, Y_{n-2}$, the values of nets $n+1, n+2, \ldots, n+(n-2)$ respectively. Then using the value Z, which is the required value at the output of gate G, we apply the backward implication rules for gate G_{n-1} to obtain Z_{n-2} and X_n', the new values of nets $n+(n-2)$ and n respectively. Having done that, we proceed backwards and apply the backward implication rules for all the gates, one at a time, ending with gate G_1. Since the binary operation represented by any logic gate is associative, the order in which the inputs X_i are cascaded is irrelevant.

It will be shown in Appendix D that the above procedure will stabilize in a single pass, unlike the approach followed in [3] which may require several passes.

Example 9. Consider the 3-input XOR gate G shown in Fig. 13(a) with associated net values. Assume that we request the value D at net 5. We now view gate G as constructed out of 2-input gates G_1 and G_2 as shown in Fig. 13(b). The

values of nets 1 and 2 are first used to compute the value $0/1/D/\overline{D}$ at net 4. By using Table 3 and the requested value D at the output of G_2 we obtain the value $0/1$ at net 4, but the value of net 3 remains unchanged. By requesting a $0/1$ at net 4 we obtain a $D/\overline{D}$ at input net 2, and the value of net 1 remains unchanged (See Fig. 13(c)). □

Note that in Example 9 the value of the intermediate net 4 does not have to be sensitized in order that the overall gate output be sensitized. This can only happen for XOR/XNOR gates.

From the discussion on backward implication it should be clear that it is not always possible to make changes at the inputs of a gate G such that the new value of the inputs yield exactly the requested value at the output of the gate (in Example 9 the new values of the inputs produce a $D/\overline{D}$ at the output of G). In Appendix D it is shown that the requested value L_G at the output is always a subset of L'_G, the value at the output implied by the new values of the inputs obtained by the backward implication procedure.

C Proof of Algorithm

We will now show, in two stages, that the proposed procedure is an algorithm.

(i) Assume that the input pattern $\boldsymbol{I}$ is a test for the fault $\boldsymbol{f}$. Thus we can always find a path $\boldsymbol{p_i}$, from the fault site to a primary output, such that in the presence of input $\boldsymbol{I}$ all nets along path $\boldsymbol{p_i}$ have different values in the normal and faulty circuit. We now apply our procedure with $\boldsymbol{p_i}$ being the path we deliberately try to sensitize. The following are consequences of the fact that our procedure imposes only those restrictions that must always be satisfied in order that $\boldsymbol{p_i}$ is sensitized.

- No contradictions can occur in the construction of $\boldsymbol{T_f(p_i)}$.
- If there are primary inputs which have been assigned values in $\boldsymbol{T_f(p_i)}$ (i.e. their values are different from $0/1$) then these inputs have the same value in $\boldsymbol{I}$.

- Changing the unassigned primary inputs in $T_f(p_i)$ to their values specified by I will convert all variant nets in $T_f(p_i)$ to invariant ones whose values are subsets of their corresponding values in $T_f(p_i)$.

Hence any procedure, that enumerates all the possible choices that convert all variant nets in $T_f(p_i)$ into invariant ones such that their values are subsets of their corresponding values in $T_f(p_i)$, will generate a set of input patterns of which I is a member.

(*ii*) Consider any input pattern I generated by our enumeration procedure. Since I is obtained from $d(tc_f(p_i, j))$, a deterministic test cube in which there are no variant nets, then applying I to the good circuit and that with the fault f will result in different values for all nets along path p_i. Thus I is a test for the fault f.

D Properties of the Backward Implication Procedure

In this appendix we discuss some useful properties of the backward implication procedure. For ease of explanation we will use the following notation in this appendix. Let G be a two-input gate. If A and B are the set of values of the inputs of G, then the set of values of the output, implied by these inputs, is denoted by $G(A, B)$. Also, let (L_0/L_i) denote the set of values at one input that can produce L_0 at the output of the gate, given that the other input is L_i.

Property 1. Let G be a two-input gate with inputs A and B. Consider a backward implication of the value Z, where $Z \subseteq G(A, B)$, at the output of G. If this backward implication causes the inputs to be changed to A' and B', respectively, then

$$Z \subseteq Z'$$

where $Z' = G(A', B')$.

Proof. Let $z \in Z$. Since $Z \subseteq G(A, B)$ there exists $a \in A$ and $b \in B$ such that $\{z\} = G(\{a\}, \{b\})$. Since $z \in Z$, after a backward implication of Z at the output of G is performed using the given tables, the new values of the inputs A' and B' are such that $a \in A'$ and $b \in B'$. Thus $z \in G(A', B')$, i.e., $z \in Z'$. Therefore $Z \subseteq Z'$. □

Note that the above property can be extended to gates which have more than two inputs.

Consider a two-input gate whose input values are L_1 and L_2 and the requested value at the output is L_G. The new values of the inputs after one pass of the backward implication procedure will be

$$L_2' = L_2 \cap (L_G / L_1)$$

and

$$L_1' = L_1 \cap (L_G / L_2').$$

If $L_2' \subseteq (L_G / L_1')$ then another pass of the backward implication procedure is unnecessary because $L_2' \cap (L_G / L_1')$ is going to yield L_2'—implying no further changes at the input.

Property 2. $L_2' \subseteq (L_G / L_1')$

Proof. Select any $l_2 \in L_2'$. Thus $l_2 \in (L_G / L_1)$. Hence there exists $l_1 \in L_1$ such that $\{l_g\} = G(\{l_1\}, \{l_2\})$ where $l_g \in L_G$. Since $l_2 \in L_2'$ and $G(\{l_1\}, \{l_2\}) \subseteq L_G$ then $l_1 \in (L_G / L_2')$ and consequently $l_1 \in L_1'$. Therefore $l_2 \in (L_G / L_1')$ since $l_1 \in L_1'$ and $G(\{l_1\}, \{l_2\}) \subseteq L_G$. This proves that $L_2' \subseteq (L_G / L_1')$. □

As a consequence of Property 2, every new application of the backward implication procedure requires only a single pass to determine the new values of each input of the gate in question.

E Properties of Variant and Invariant Nets

We now discuss some properties of circuit nets that characterize them as being variant or invariant. These properties would be useful in their identification which in turn is

necessary in the test generation process. The properties will be discussed in terms of the values of the input and output nets of a gate as given by their corresponding entries in any deterministic test cube, $\boldsymbol{d}(\boldsymbol{tc}_k)$. Some of these properties are straightforward and will be stated without proof.

Property 1. Consider an n input gate G. If in any $\boldsymbol{d}(\boldsymbol{tc}_k)$ the set of values associated with $n-1$ of its inputs have cardinality equal to 1, then the output of G is an invariant net in $\boldsymbol{d}(\boldsymbol{tc}_k)$. □

Property 2. Let G be a two-input XOR/XNOR gate. If in any $\boldsymbol{d}(\boldsymbol{tc}_k)$ one input and the output of G have the value $D/\overline{D}$, then the output net is invariant in $\boldsymbol{d}(\boldsymbol{tc}_k)$. □

In Property 2, note that the set of values at the other input of G cannot contain D or $\overline{D}$ because they desensitize the output.

Property 3. Let G be a gate which is not an XOR/XNOR gate. If, in any $\boldsymbol{d}(\boldsymbol{tc}_k)$, one input of the gate has the value D or $\overline{D}$, but not both, and the output is either D or $\overline{D}$ (depending on whether G is non-inverting or inverting), then the output of G is an invariant net with respect to $\boldsymbol{d}(\boldsymbol{tc}_k)$. □

Proof of Property 3. Assume that G is an AND gate whose output and one input has the value D. As a result of the backward implication of a D at the output, the value of the other inputs of G in $\boldsymbol{d}(\boldsymbol{tc}_k)$ can never contain 0 or $\overline{D}$ because these values can only produce at 0 at the output, given that one input is D. Since all other input combinations produce a D at the output of the gate, the output is an invariant net. The proof for the other cases are analogous. □

The next property is a consequence of Property 3.

Property 4. Consider a single path $\boldsymbol{p}_i$ from n_f to a primary output. Recall that n_f has the value D or $\overline{D}$ but not both. If there are no XOR/XNOR gates on path $\boldsymbol{p}_i$, then all the nets on this path are invariant with respect to their values in $\boldsymbol{T}_f(\boldsymbol{p}_i)$. □

Analogous to Appendix D let $G(I_1, I_2, \ldots, I_m)$ be the resultant set of values at the output of an m-input gate G when the sets $I_1, I_2, \ldots, I_m$ are applied at its inputs.

Recall the procedure for performing a backward implication for a multi-input gate

G by using a cascade of 2-input gates G_1, G_2, ..., G_{n-1} as described in Appendix B.2. As stated earlier the values X_1, X_2, ..., X_{n-1} of nets $1, 2, \ldots, n-1$, are first used to compute Y_1, Y_2, ..., Y_{n-2}, the values of nets $n+1, n+2, \ldots, n+(n-2)$ respectively. Then using the requested value of Z at the output of gate G, we apply the backward implication rules to obtain the values X_1', X_2', ..., X_n' at nets $1, 2, \ldots, n$ and the values Z_1, Z_2, ..., Z_{n-2} at nets $n+1, n+2, \ldots, n+(n-2)$ respectively. For the ease of notation let us denote the value Z at net $n+(n-1)$ as Z_{n-1} and the value X_1' at net 1 as Z_0. Note that by using Property 1 of Appendix D we obtain $Z_i \subseteq G_i(Z_{i-1}, X_{i+1}')$ for $i = 1, 2, \ldots, n-1$.

Property 5. Consider the backward implication procedure for a n-input gate. In terms of the notation outlined above $G(X_1', X_2', \ldots, X_n') = Z$ if and only if $G_i(Z_{i-1}, X_{i+1}') = Z_i$ for $i = 1, 2, \ldots, n-1$. Stated in words, the output of the composite gate G is an invariant net if and only if the output nets of all the 2-input gates G_i are also invariant. □

Proof of Property 5.

(i) We prove that $G(X_1', X_2', \ldots, X_n') = Z$ implies that $G_i(Z_{i-1}, X_{i+1}') = Z_i$ for $i = 1, 2, \ldots, n-1$ by contradiction. Assume that there exists a j, $1 \leq j \leq n-1$, such that $G_j(Z_{j-1}, X_{j+1}') \neq Z_j$. By Property 1 of Appendix D, $Z_j \subset G_j(Z_{j-1}, X_{j+1}')$. Thus there exists $z_j \in (G_j(Z_{j-1}, X_{j+1}') - Z_j)$. Moreover by the deterministic nature of the backward implication procedure we can state that there exists $z_{j+1} \in (G_{j+1}(\{z_j\}, X_{j+2}') - Z_{j+1})$. This is because z_j cannot produce any of the required values at the output of gate G_{j+1}, given that the other input has the value X_{j+2}', and was thus not included in Z_j. Using an inductive procedure we can state that with $z_j, X_{j+2}', X_{j+3}', \ldots, X_n'$ at the inputs of gates $G_{j+1}, G_{j+2}, \ldots, G_{n-1}$ we can obtain a value z_{n-1}, at the output of gate G_{n-1}, that does not belong to the set Z_{n-1}. Since $z_j \in G_j(Z_{j-1}, X_{j+1}')$ there exists $z_{j-1} \in Z_{j-1}$ such that $z_j \in G_j(\{z_{j-1}\}, X_{j+1}')$. Therefore with $X_1', X_2', \ldots, X_j'$ at the inputs of gates $G_1, G_2, \ldots, G_{j-1}$ we can obtain a value z_{j-1} at the output of gate G_{j-1} because of the deterministic nature of the

backward implication procedure that was used to obtain Z_{j-1}. This implies that with $X'_1, X'_2, \ldots, X'_{j+1}$ at the inputs of gates $G_1, G_2, \ldots, G_j$ we can obtain z_j at the output of gate G_j. Consequently with $X'_1, X'_2, \ldots, X'_n$ at the inputs of gates $G_1, G_2, \ldots, G_{n-1}$ we can obtain the value z_{n-1} at the output of gate G_{n-1}. Thus $z_{n-1} \in G(X'_1, X'_2, \ldots, X'_n)$. But $z_{n-1} \notin Z_{n-1} = Z$; therefore $G(X'_1, X'_2, \ldots, X'_n) \neq Z$, which is a contradiction.

(*ii*) The fact that $G_i(Z_{i-1}, X'_{i+1}) = Z_i$ for $i = 1, 2, \ldots, n-1$ implies that $G(X'_1, X'_2, \ldots, X'_n) = Z$ follows from the associative property of all the logic gates we are considering.

□

The above property may useful in determining whether the output net of a gate is variant or not.

F Proof of Theorems in §5.1

In this appendix we will always denote the net at which we apply a value (either in **3-VP** or **15-VP**), to observe the implications at other nets, as net m_1. In order to prove the theorems stated in §5.1 we will make use of the following lemmas. The proofs of the first three lemmas are straightforward and will not be presented.

Lemma 1 The value of every net in a **3-VP** experiment is a subset of the value of this net in the corresponding **15-VP** experiment. □

Lemma 2 In a **3-VP** experiment, if we traverse backwards along any path of singleton values from net m_j which has a singleton value, then we will always reach net m_1. □

Lemma 3 If a **3-VP** experiment yields a singleton value at net m_j, then this experiment yields a path of singleton values from net m_1 to net m_j. Furthermore, if this experiment yields a singleton value at the output net of any gate G, then it either

sets one or more of its inputs to controlling values or all its inputs to non-controlling values. □

Lemma 4 If the output of a gate G has a singleton value in a **3-VP** experiment and additional value(s) in the corresponding **15-VP** experiment, then there exists at least one input to this gate which has a singleton value in this **3-V** experiment and additional value(s) in this **15-VP** experiment.

Proof (*i*) Assume that in the **3-VP** experiment the singleton value at the output of G was obtained due to a controlling value at one or more inputs of G. In this situation all these inputs must contain additional value(s) in the corresponding **15-VP** experiment. Otherwise, we will obtain the singleton value at the output as given by the **3-VP** experiment.

(*ii*) From Lemma 3, the only remaining alternative is that the singleton value at the output of gate G in the **3-VP** experiment was obtained by setting all its inputs to non-controlling values. Thus it is obvious that at least one input must contain additional value(s) in the corresponding **15-VP** experiment in order to produce additional value(s) at the output. □

Proof of Theorem 1 Let G be the gate whose output is net m_2 or whose output is a FOS net with m_2 as a FOB net. Let L_2 be the singleton value yielded at net m_2 by the **3-VP** experiment. Let the corresponding **15-VP** experiment yield the value L_2' at net m_2. By Lemma 1, $L_2 \subseteq L_2'$. Assume that $L_2 \subset L_2'$. Then, by Lemma 4, at least one input of G must contain a singleton value in this **3-VP** experiment and additional value(s) in this **15-VP** experiment. This input must either be net m_1 or be connected to the output of some gate. If the latter is true, then this gate must also possess at least one input that has a singleton value in this **3-VP** experiment and additional value(s) in this **15-VP** experiment. Proceeding backwards in this manner we must, by Lemma 2, eventually conclude that net m_1 has a singleton value in this **3-VP** experiment and additional value(s) in this **15-VP** experiment. This is a contradiction. Thus, $L_2 = L_2'$. □

Proof of Theorem 2. Let G be the gate whose output is net m_2 or whose output is a FOS net with m_2 as a FOB net. Since net m_2 has the value 0/1 in the **3-VP** experiment, then at least one input of G must also have the value 0/1 in this experiment and all the values of all other inputs must include the non-controlling value for gate G. The input with the 0/1 value is either a primary input or is connected to the output of some gate which in turn must have at least one input with the value 0/1 and the value of all other inputs must include the non-controlling value for this gate. Proceeding in this manner we will reach some primary input, say I_1, in a finite number of steps. Thus there is a path $\boldsymbol{p}_1$ from I_1 to net m_2 such that all nets on this path have the value 0/1 in this **3-VP** experiment. Furthermore, the value of all inputs of every gate on this path $\boldsymbol{p}_1$ must contain the non-controlling value. By Lemma 1, in the corresponding **15-VP** experiment, the value of all inputs of every gate along path $\boldsymbol{p}_1$ must contain the non-controlling value. Thus a $0/1/D/\overline{D}$ from input I_1 will propagate through every gate along path $\boldsymbol{p}_1$ yielding a $0/1/D/\overline{D}$ at net m_2. □

G Line Justification Approach

In this appendix we present a procedure which can be used instead of the enumeration procedure in order to derive tests from $\boldsymbol{T}_f(\boldsymbol{p}_i)$. The approach followed here is similar to line justification in DALG [14]. Recall that a gate whose output is a variant net is characterized by having inputs, specified in $\boldsymbol{T}_f(\boldsymbol{p}_i)$, that can produce values at the output of this gate which do not appear in $\boldsymbol{T}_f(\boldsymbol{p}_i)$. So we must restrict the inputs of this gate in several ways—multiple choices exist because it is a variant net—such that these disallowed values cannot appear at its output. When making these input restrictions, care must be taken to see that every input pattern from $\boldsymbol{T}_f(\boldsymbol{p}_i)$ that yields permissible output values is accounted for in at least one of the choices. If the constraints imposed by a particular choice cannot be met, then another choice is selected. If there is no input combination that converts all variant nets into invariant

ones, then there exists no test pattern that sensitizes path p_i. Different heuristics can be used to decide the priority among different choices. For the sake of efficiency we would like to minimize both the number of choices and the overlap between different choices.

Example 10. Assume that the entries corresponding to a two-input AND gate (with input nets m_1 and m_2, output net m_g) in $T_f(p_i)$ is as follows:

m_1	m_2	m_g
$0/D/\overline{D}$	$0/1/D/\overline{D}$	0

Note that net m_g is variant in $T_f(p_i)$ since its value, as implied by nets m_1 and m_2, is $0/D/\overline{D}$. All the permissible input combinations that convert net m_g into an invariant one are covered by the following three input patterns:

	m_1	m_2	m_g
(i)	0	$0/1/D/\overline{D}$	0
(ii)	D	$0/\overline{D}$	0
(iii)	$\overline{D}$	$0/D$	0

□

We now give a procedure that can be used to obtain the different choices that can be made to convert a variant net to an invariant one. Only a 2-input AND gate and a 2-input XOR gate need to be considered because, as stated earlier, all other cases can be derived from these. Consider a 2-input gate G with input nets m_1 and m_2 and output net m_g. Let L_1, L_2, and L_G be the set of values in $T_f(p_i)$, associated with m_1, m_2, and m_g, respectively. Since m_g is a variant net, $|L_1| > 1$ and $|L_2| > 1$. Without loss of generality, assume that $|L_2| \geq |L_1|$ and let $L_1 = \{\ell_{11}, \ell_{12}, \ldots, \ell_{1m}\}$. With L_G as the requested output value and $\ell_{1i} \in L_1$ as the value of one input we use either Table 11 (if G is an AND gate) or Table 3 (if G is an XOR gate) to obtain the set L_{2i}. The allowable value at net m_2, with ℓ_{1i} at net m_1, is given by $L'_{2i} = L_{2i} \cap L_2$. This procedure is repeated for all the elements of L_1. This yields the following choices:

Net m_1	$\{\ell_{11}\}$	$\{\ell_{12}\}$ ...	$\{\ell_{1m}\}$
Net m_2	L'_{21}	L'_{22} ...	L'_{2m}

Net m_g is an invariant net for any of the above choices because net m_1 has a value whose cardinality is equal to 1. We may reduce the number of choices by combining values of input m_1. This can be done only when $L'_{2i} = L'_{2j}$. In this situation the i^{th} and j^{th} choices can be replaced by the input pattern that has $\{\ell_{1i}, \ell_{1j}\}$ at m_1 and L'_{2i} at m_2. The same procedure can be used to combine three choices if possible.

Example 11. Consider a 2-input AND gate (input nets m_1, m_2; output net m_g) whose net entries in $\boldsymbol{T}_f(\boldsymbol{p}_i)$ is shown below:

m_1	m_2	m_g
$0/D/\overline{D}$	$0/1/D/\overline{D}$	$0/D$

.

Thus net m_g is a variant net in $\boldsymbol{T}_f(\boldsymbol{p}_i)$. The procedure described above yields the following choices:

Net m_1	0	D	$\overline{D}$
Net m_2	$0/1/D/\overline{D}$	$0/1/D/\overline{D}$	$0/D$

Since the value of m_2 is identical for the first two choices they can be combined to yield:

Net m_1	$0/D$	$\overline{D}$
Net m_2	$0/1/D/\overline{D}$	$0/D$

□

Whenever we choose input values of a gate in accordance with the procedure described above, we must find the resultant deterministic test cube. The whole procedure has to be repeated as long as there are variant nets in the resulting deterministic test cube.

References

[1] M. Abramovici, P. R. Menon and D. T. Miller, "Checkpoint Faults are not sufficient Faults for Test Generation," *IEEE Transactions on Computers*, Vol. C-35, pp 770–771, August 1986.

[2] M. Abramovici, P. R. Menon and D. T. Miller, "Critical Path Tracing - An Alternative To Fault Simulation," in *Proceedings of the* 20^{th} *Design Automation Conference*, pp 214–220, 1983.

[3] Sheldon B. Akers, "A Logic System for Fault Test Generation," Presented at Symposium on Fault-Tolerant Computing, Paris, France, June 1975. Also *IEEE Transactions on Computers*, Vol. C-25, pp 620–630, June 1976.

[4] M. A. Breuer and A. D. Friedman, *Diagnosis & Reliable Design of Digital Systems.* Computer Science Press, 1976.

[5] Charles W. Cha, William E. Donath and Füsun Özgüner, "9-V Algorithm for Test Pattern Generation of Combinational Digital Circuits," *IEEE Transactions on Computers*, Vol. C-27, pp 193–209, March 1978.

[6] Wu-Teng Cheng, "Split Circuit Model for Test Generation," in *Proceedings of the* 25^{th} *ACM/IEEE Design Automation Conference,* pp 96–101, 1988.

[7] Warren Debany, "On Using the Fanout-Free Substructure of General Combinational Networks," PhD Dissertation, Syracuse University, December 1985.

[8] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Transactions on Computers*, Vol. C-32, pp 1137–1144, December 1983.

[9] H. Fujiwara and S. Toida, "The complexity of fault detection: An approach to design for testability," in *Proceedings of the* 12^{th} *International Symposium on Fault Tolerant Computing*, pp 101–108, June 1982.

[10] Prabhakar Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, Vol. C-30, pp 215—222, March 1981.

[11] O. H. Ibarra and S. K. Sahni, "Polynomially Complete fault detection problems," *IEEE Transactions on Computers*, Vol. C-24, pp 242–259, March 1975.

[12] Tom Kirkland and M. Ray Mercer, "A Topological Search Algorithm for ATPG," in *Proceedings of the* 24^{th} *ACM/IEEE Design Automation Conference,* pp 502–508, 1987.

[13] P. Muth, "A Nine-Valued Circuit Model for Test Generation," *IEEE Transactions on Computers*, Vol. C-25, pp 630–636, June 1976.

[14] J. P. Roth, W. G. Bouricius and P. R. Schneider, "Programmed algorithms to compute tests to detect and distinguish between failures in logic circuits," *IEEE Transactions on Computers*, Vol. C-16, pp 567–579, October 1967.

[15] Michael H. Schulz, Erwin Trischler and Thomas M. Sarfert, "Socrates—A Highly Efficient Automatic Test Pattern Generation System," in *Proceedings of the 1987 International Test Conference,* pp 1016–1026, 1987.

[16] R. Tarjan, "Finding Dominators in Directed Graphs," *SIAM Journal of Computing*, Vol. 3, pp 62–89, 1974.

AND	0	1	D	$\overline{D}$
0	0	0	0	0
1	0	1	D	$\overline{D}$
D	0	D	D	0
$\overline{D}$	0	$\overline{D}$	0	$\overline{D}$

Table 1. AND table

Variable	0	1	D	$\overline{D}$
Complement	1	0	$\overline{D}$	D

Table 2. NOT table

XOR	0	1	D	$\overline{D}$
0	0	1	D	$\overline{D}$
1	1	0	$\overline{D}$	D
D	D	$\overline{D}$	0	1
$\overline{D}$	$\overline{D}$	D	1	0

Table 3. XOR table

Nets with TRUE Token	3_f, 14, 15, 16, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 30, 31, 36, 37, 38, 39, 40, 41, 42, 43, 45
Nets with FALSE Token	1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 17, 18, 29, 32, 33, 34, 35, 44, 46

(a) Boolean token assignment

Net	36	37	38	39	40	41	42	43	45
Token Vector	[36,0]	[36,0]	[36,0]	[36,0]	[36,0]	[36,1]	[36,1]	[36,1]	[45,0]

(b) Token vector assignment

Net	40	41	42	43
Token Vector	[36,0]	[36,1]	[36,1]	[36,1]

(c) Reduced set of token vectors

Table 4. Token assignment for net 3 $s-a-0$ in Fig. 1

tc_1:	3	3_f	31	36	45
	1	D	$D/\overline{D}$	$D/\overline{D}$	$D/\overline{D}$

$d(tc_1)$:	14	15	16	19	20	21	22	23	30	37	38	39	40	41	42	43
	D	D	D	$0/D$	$0/\overline{D}$	$0/D$	$0/1/D$	$0/1/\overline{D}$	$0/D/\overline{D}$	$D/\overline{D}$	$D/\overline{D}$	$D/\overline{D}$	$1/D/\overline{D}$	$1/D/\overline{D}$	$1/D/\overline{D}$	$1/D/\overline{D}$

tc_2:	21
	D

$d(tc_2)$:	6	30	31
	1	$0/D$	D

tc_3:	42	43
	$D/\overline{D}$	$D/\overline{D}$

Steps in $d(tc_3)$ construction:

(*i*) $\frac{40}{1} \longrightarrow \frac{35\ 9\ 32}{1\ 1\ 1}$

(*ii*) $\frac{29}{1} \longrightarrow \frac{4\ 17\ 18}{0\ 0\ 0} \longrightarrow \frac{20}{\overline{D}} \longrightarrow \frac{23\ 24}{1/\overline{D}\ \ 1/\overline{D}} \longrightarrow \frac{25\ \ 26\ \ 27\ \ 28}{1/\overline{D}\ \ 1/\overline{D}\ \ 1/\overline{D}\ \ 1/\overline{D}} \longrightarrow \frac{30}{(1/\overline{D}) \cap (0/D) = \emptyset}$ (Contradiction)

tc_4:	40
	$D/\overline{D}$

$d(tc_4)$:	4	9	11	17	18	20	23	24	25	26	27	28	29	32	35	41	42	43
	1	0	0	1	1	0	0/1	$0/1/D$	$0/1/D$	$0/1/D$	$0/1/D$	$0/1/D$	0	0	0	1	1	1

Table 5. Test cubes for faulty net 3 $s-a-0$ in Fig. 1

Value applied at net m_1	Implied value at net m_2								
	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	$(viii)$	(ix)
0	0/1	0	1	0	1	0	1	0/1	0/1
1	0/1	0	1	1	0	0/1	0/1	0	1

Table 6. Implications in **3-VP**

Value applied at net m_1	Implied value at net m_2								
	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	$(viii)$	(ix)
0	$0/1/D/\overline{D}$	0	1	0	1	0	1	$0/1/D/\overline{D}$	$0/1/D/\overline{D}$
1	$0/1/D/\overline{D}$	0	1	1	0	$0/1/D/\overline{D}$	$0/1/D/\overline{D}$	0	1

Table 7. Implications of a 0 and 1 in **15-VP**

	Value applied at net m_1	Implied value at net m_2								
		(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)	$(viii)$	(ix)
3-VP	0	0/1	0	1	0	1	0	1	0/1	0/1
	1	0/1	0	1	1	0	0/1	0/1	0	1
15-VP	0	$0/1/D/\overline{D}$	0	1	0	1	0	1	$0/1/D/\overline{D}$	$0/1/D/\overline{D}$
	1	$0/1/D/\overline{D}$	0	1	1	0	$0/1/D/\overline{D}$	$0/1/D/\overline{D}$	0	1
	D	$0/1/D/\overline{D}$	0	1	D	$\overline{D}$	$0/D$	$1/\overline{D}$	$0/\overline{D}$	$1/D$
	$\overline{D}$	$0/1/D/\overline{D}$	0	1	$\overline{D}$	D	$0/\overline{D}$	$1/D$	$0/D$	$1/\overline{D}$

Table 8. Relationship between **3-VP** and **15-VP**

L_2	G			
0	OR	NAND	XOR	XNOR
1	NOR	AND	XOR	XNOR

Table 9. (L_2, G) combinations that yield useful contrapositve assertions

tc_1:	3	3_f	31	36	45
	1	D	$D/\overline{D}$	$D/\overline{D}$	$D/\overline{D}$

$d(tc_1)$:	2	5	14	15	16	19	20	21	22	23	24	25	26	30	37	38	39	40	41	42	43
	0	0	D	D	D	$0/D$	$0/\overline{D}$	$0/D$	$0/D$	$0/\overline{D}$	$0/D/\overline{D}$	$0/D/\overline{D}$	$0/D/\overline{D}$	$0/D/\overline{D}$	$D/\overline{D}$	$D/\overline{D}$	$D/\overline{D}$	$1/D/\overline{D}$	$1/D/\overline{D}$	$1/D/\overline{D}$	$1/D/\overline{D}$

tc_2:	21
	D

$d(tc_2)$:	4	6	17	18	20	23	24	25	26	27	28	29	30	31	42
	1	1	1	1	0	0	$0/D$	$0/D$	$0/D$	$0/1/D$	$0/1/D$	0	$0/D$	D	1

Steps in tc_3 construction:

$$\frac{42}{1 \cap (D/\overline{D}) = \emptyset}$$ (Contradiction)

tc_4:	40
	$D/\overline{D}$

$d(tc_4)$:	9	11	32	35	41	43
	0	0	0	0	1	1

Table 10. Test cubes for Example 2

* / **	0	1	D	$\overline{D}$
0	$0/1/D/\overline{D}$	$\emptyset$	$\emptyset$	$\emptyset$
1	0	1	D	$\overline{D}$
D	$0/\overline{D}$	$\emptyset$	$1/D$	$\emptyset$
$\overline{D}$	$0/D$	$\emptyset$	$\emptyset$	$1/\overline{D}$

* Requested Output

** Existing value at one input

Table 11. Backward implication for a 2-input AND gate

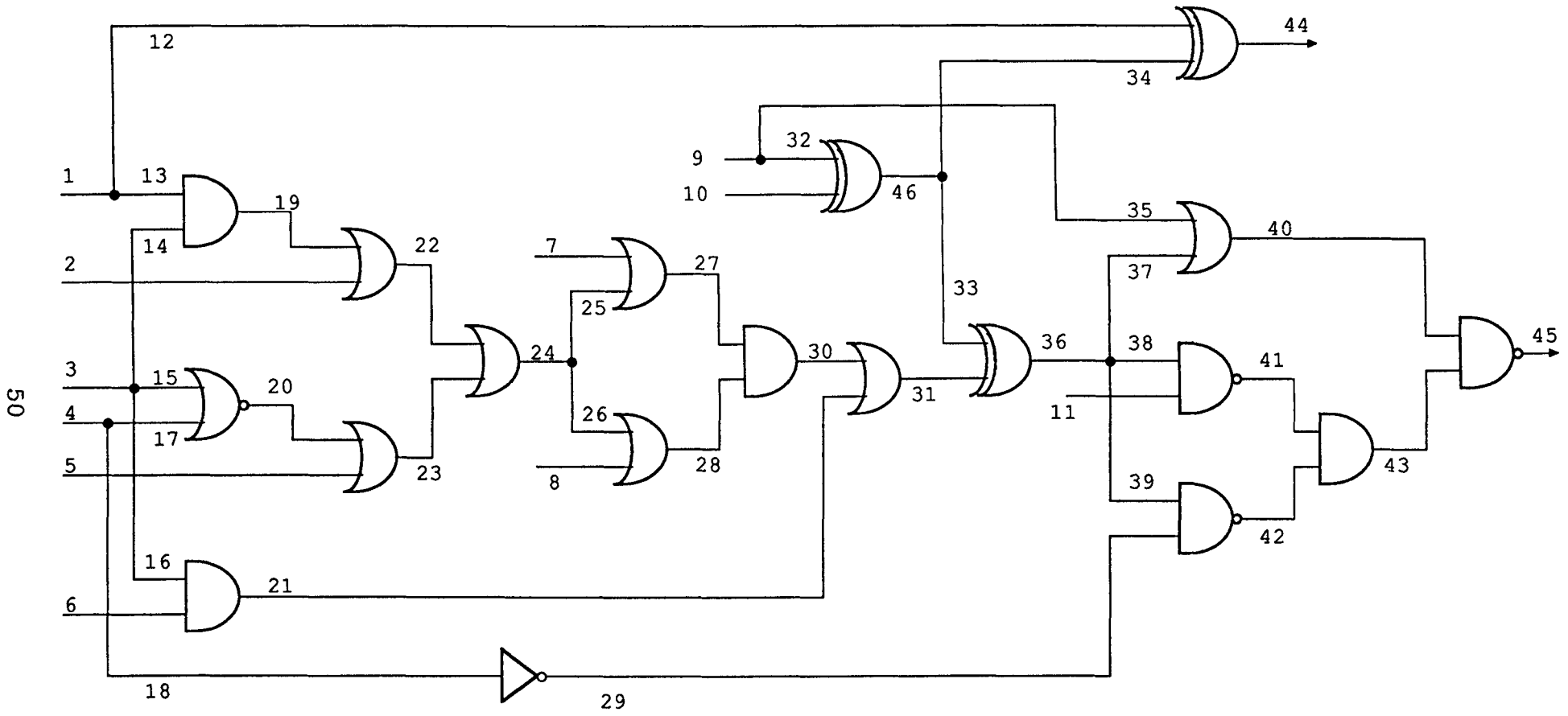


Fig.1 An example circuit

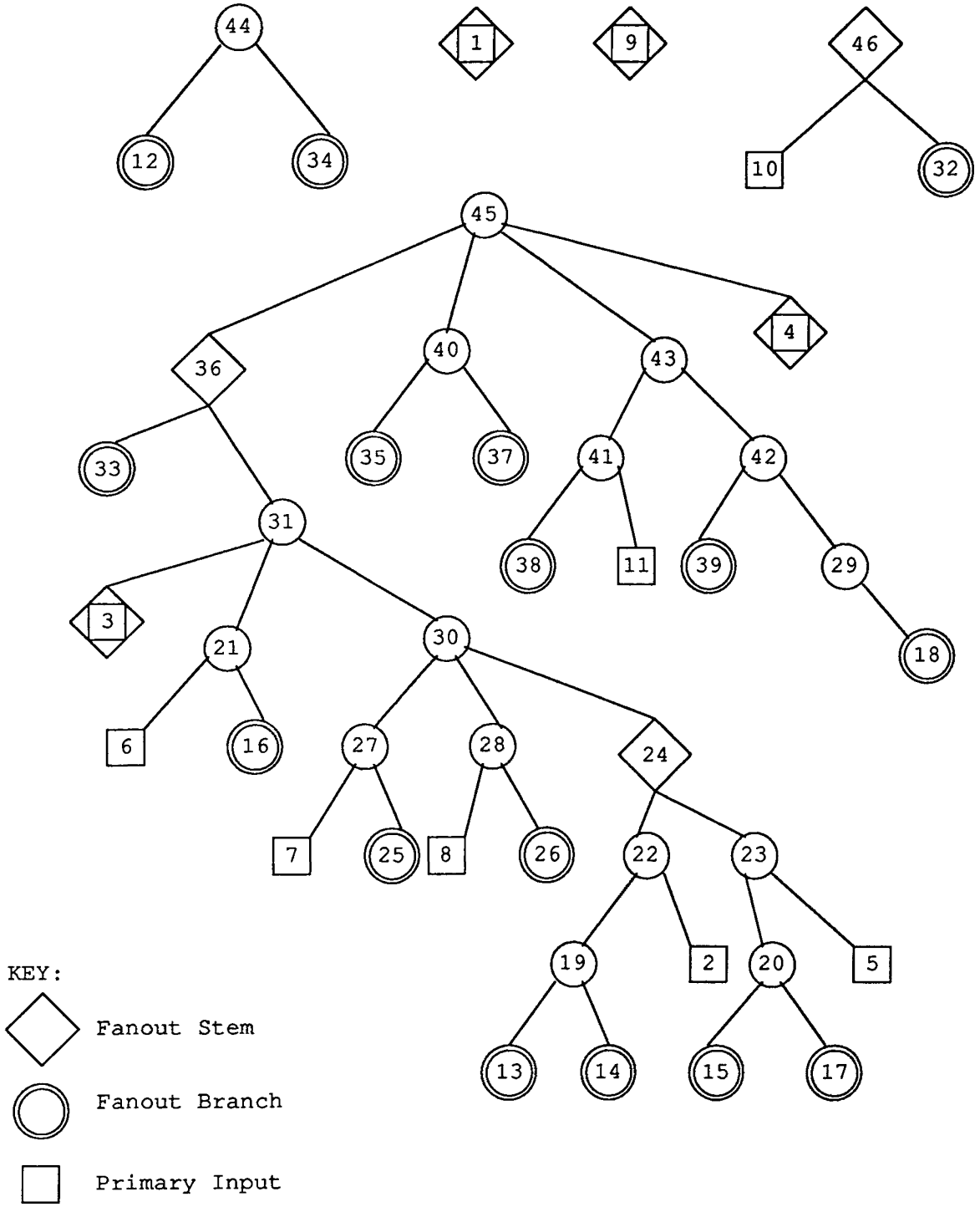


Fig.2 Dominator forest for circuit of Fig.1

Fig.3 Introduction of fictitious gate

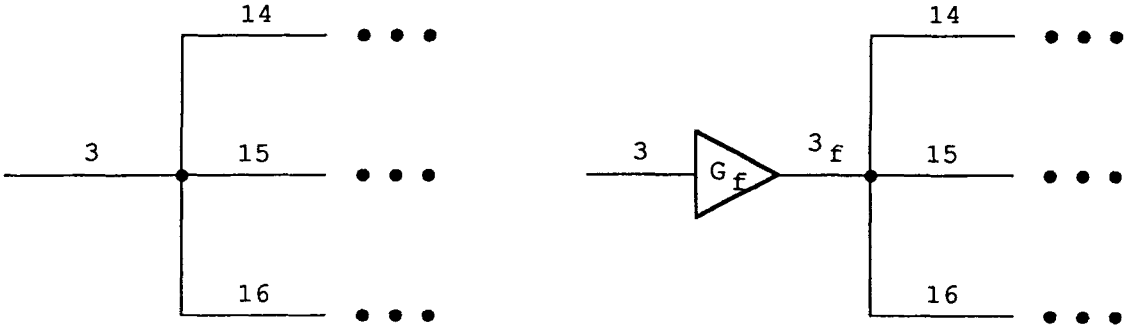


Fig.4 Fictitious gate for net 3 s-a-0 in circuit of Fig.1

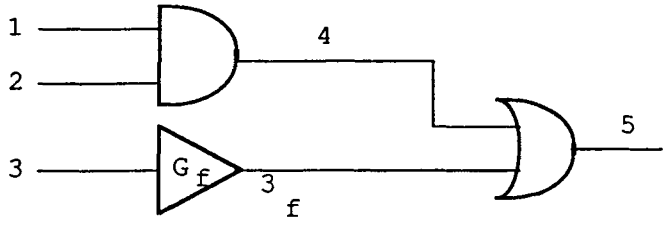


Fig.5 Circuit for Example 1

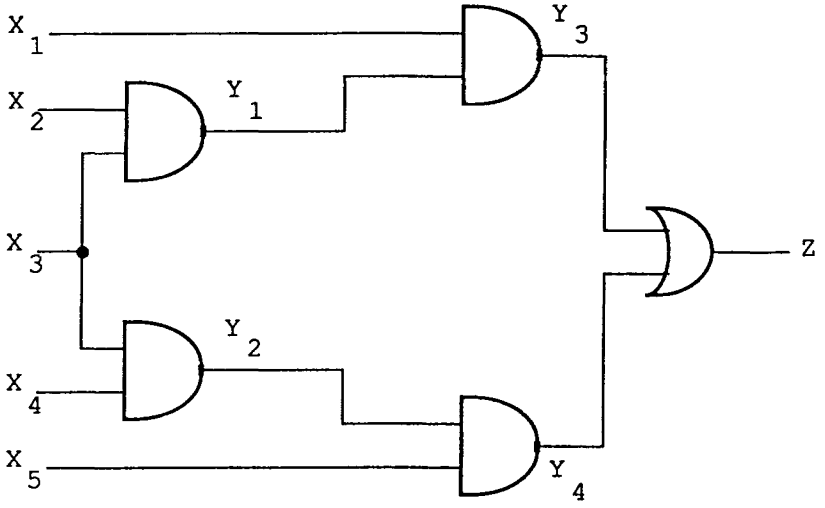


Fig.6 Use of the Contrapositive

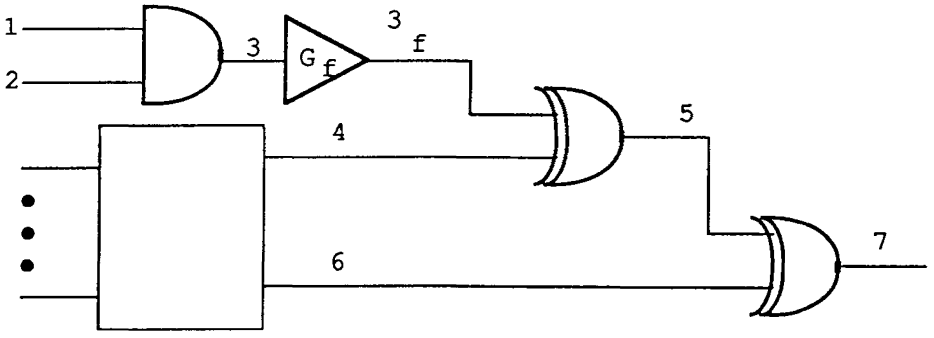


Fig.7 Circuit for Example 3

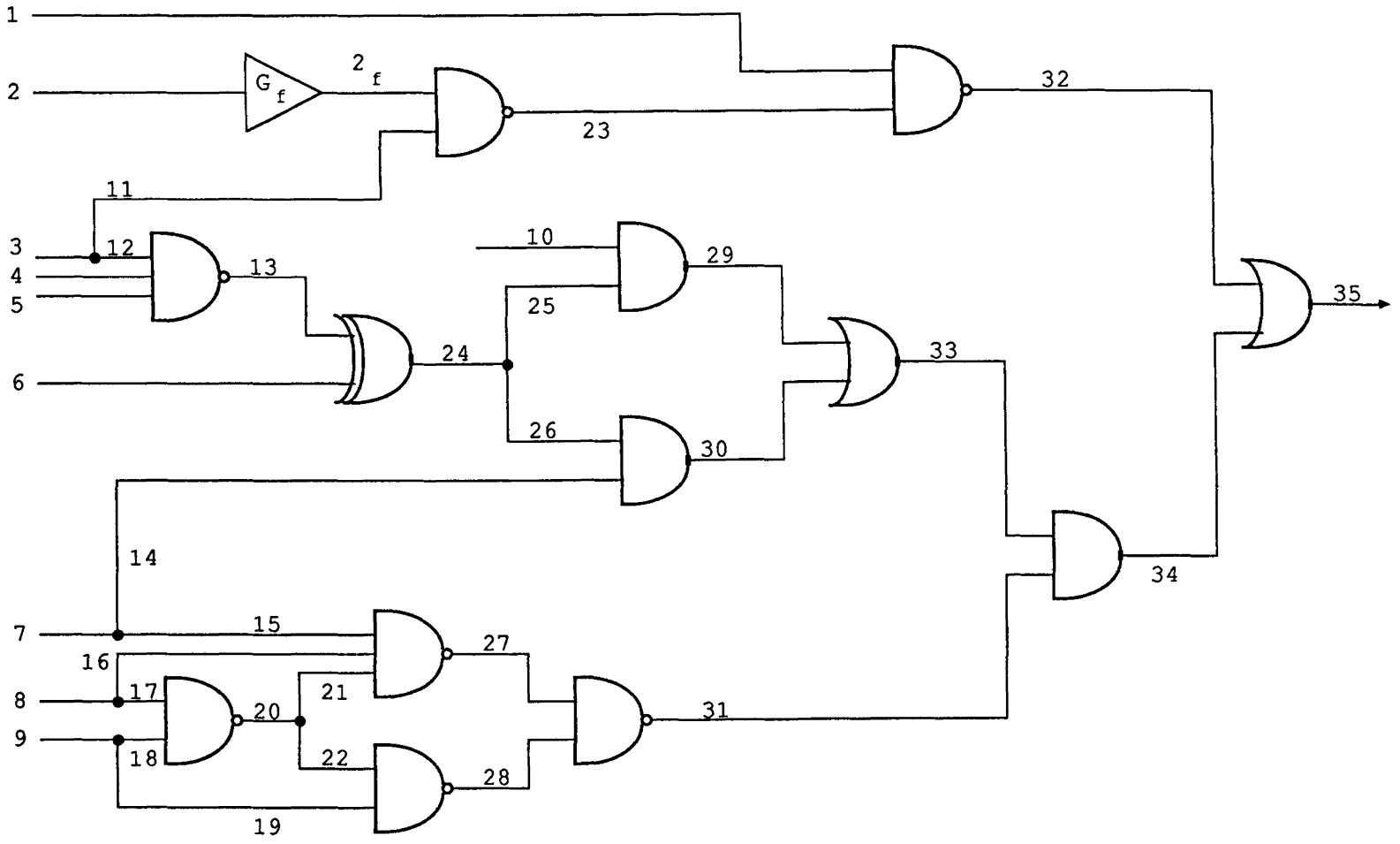


Fig.8 Circuit for Example 4

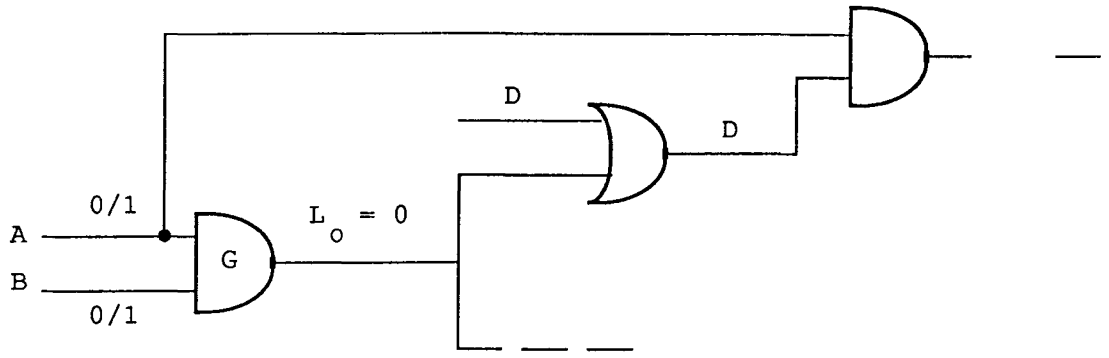


Fig.9 Circuit for Example 5

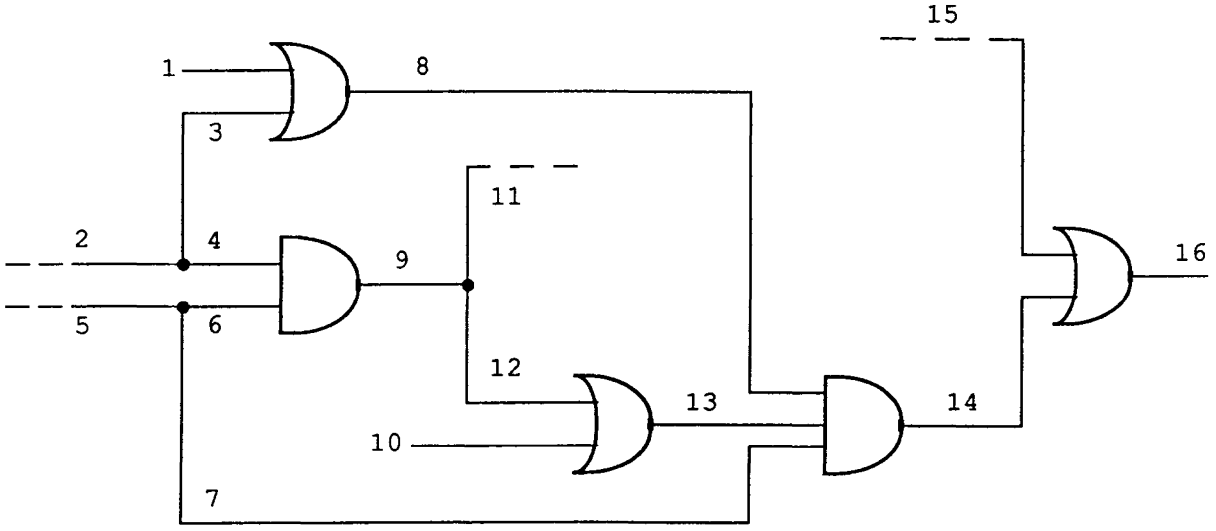
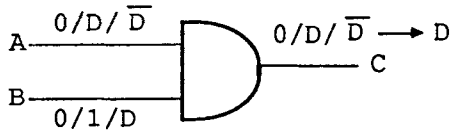
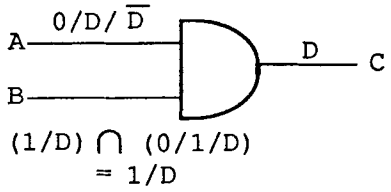


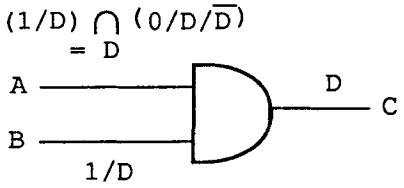
Fig.10 Circuit for Example 6



(a)



(b)



(c)

Fig.11 Circuit for Example 8

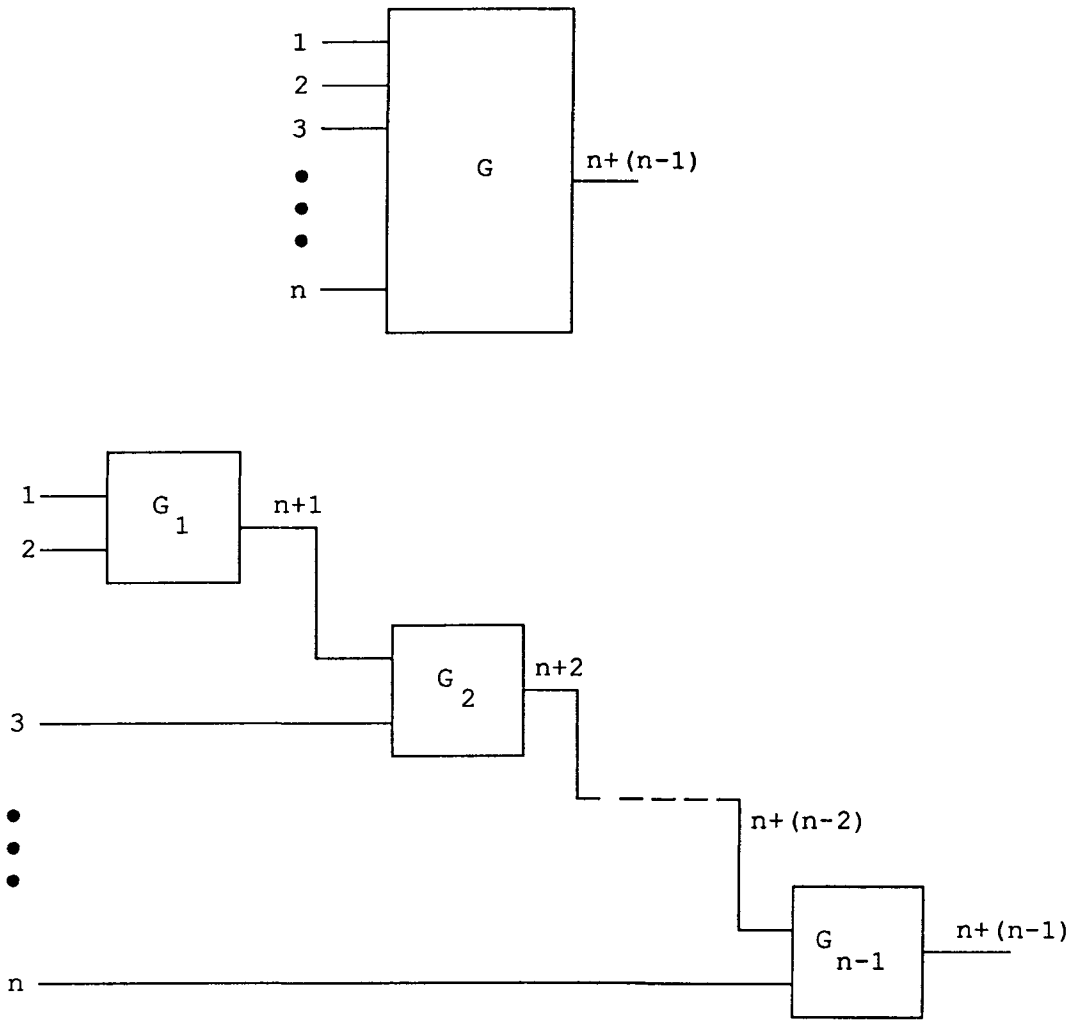


Fig.12 Gate decomposition

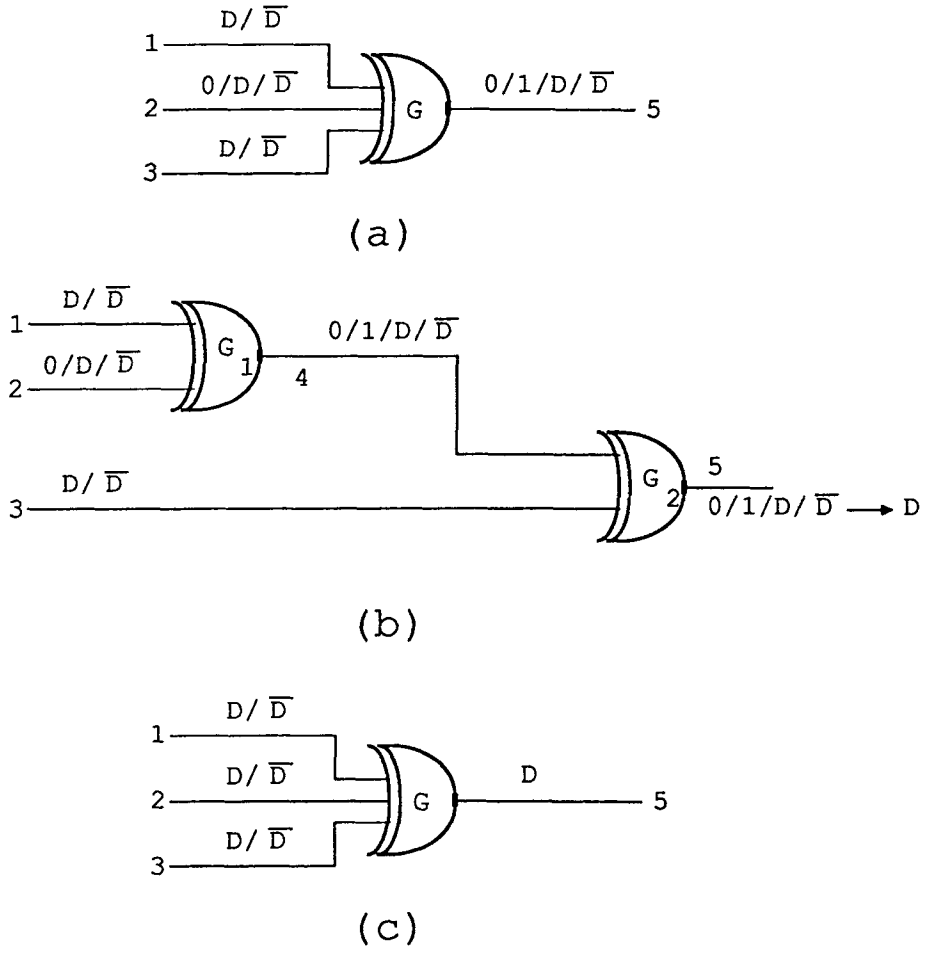


Fig.13 Circuit for Example 9