A TOPOLOGICAL SEARCH ALGORITHM FOR ATPG

Tom Kirkland

MCC
3500 W. Balcones Cen. Dr.
Austin, Tx 78759

M. Ray Mercer

Univ. of Texas at Austin
ENS 143
Austin, TX 78712-1084

ABSTRACT

The automatic generation of tests for combinational digital circuits is examined from the standpoint of a guided search through a search space. The limitations of this process, namely the size of the search space and the overall strategy, are identified and methods are presented to reduce the size of the search space as well as produce a more optimal ordering of node assignments. A new algorithm is proposed that uses the smaller search space and the improved ordering for node assignments based on a topological analysis of the circuit. Results are presented indicating that this new algorithm, termed TOPological Search (TOPS), is faster than existing algorithms and also rapidly identifies many redundant faults without search.

Introduction

The generation of tests for combinational digital circuits can be characterized as a search problem. A test for a fault can be found by trying different patterns until one is found that differentiates the good circuit from the faulty circuit. Finding an efficient way to try these different patterns makes the problem amenable to search techniques [Kirkland 87]. The search space can be quite large, however, and a solution may not exist. Some method must be used to organize the search process to assure that every possible pattern can be tried and that no pattern is tried twice. It is not necessary, however, to explicitly search the entire space; it may be possible in some cases to determine that no solution exists with only a partial search [Goel 81]. Similarly, if a way can be found to search a smaller space for the solution, the size of the search problem is reduced and the test generation problem becomes more tractable.

During a search for a test, values are assigned to nodes in the search space. The nodes fall into two categories: mandatory assignment nodes, and optional assignment nodes. Mandatory assignment nodes are those which must take on a unique value for a test to exist, while optional assignment nodes do not have a unique required value. For example, all tests for a stuck fault mandate that a difference be produced at the fault site between the good value and the faulty value. This assignment is not a matter of choice and is therefore, a mandatory assignment. On the other hand, if more than one path exists for observing the difference at the circuit outputs, some choice must be made as to which path or set of paths to use; such choices lead to optional assignments.

A second problem in optimizing the search for a test is identifying the smallest possible set of nodes that consititute a complete search space [Kirkland 86]. If the size of the search space can be reduced, then the time to search the space will be reduced as well. Since the set of nodes that constitute the search space should also be an independent set of nodes (to eliminate the possibility of internal conflict during resolution of the assignments at these nodes), the problem becomes one of finding the smallest set of independent nodes that completely differentiates the good circuit from the faulty one.

Mandatory Node Assignments

The problem of finding the mandatory nodes and their values can be broken into two sub-problems: (1) mandatory control assignments - the mandatory node assignments for exciting the fault site, and (2) mandatory observation assignments - the mandatory node assignments for propagating the fault effect to at least one primary output. Knowledge of these mandatory node assignments can be used to restrict the search space. Identifying the mandatory observation nodes involves the location of all single gates through which the fault effect must pass to reach a primary output. The set of mandatory observation nodes does not include any nodes that depend on a path choice. Consider the circuit of Figure 1 with a SA0 fault on input M to gate S. Clearly, the fault effect must eventually be propagated through S and W to be observable at the primary output Z. Therefore, gates S and W are mandatory observation nodes, while T and U are not mandatory observation nodes since a path choice exists. A method for finding mandatory observation nodes can be based on the theory of dominators in flow graphs [Tarjan 74]. Before discussing the method, we discuss the properties of dominators and circuit reconvergence.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the ACM copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Association for Computing Machinery. To copy otherwise, or to republish, requires a fee and/or specific permission.

© 1987 ACM 0738-100X/87/0600-0502$00.75

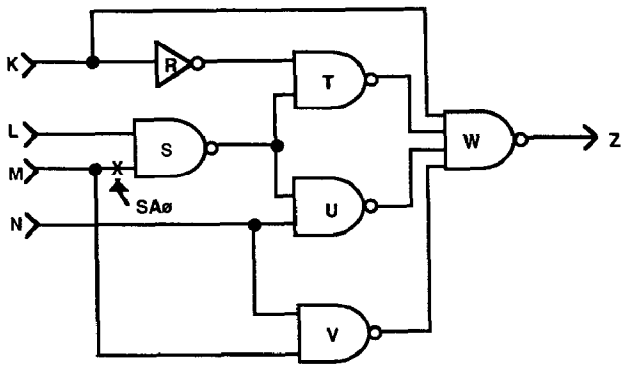

Figure 1. Mandatory Assignments

Dominators and Reconvergent Fanout

Consider a directed acyclic graph G with vertices V and edges E. A vertex r in V is the root of the graph if there is a path from r to every vertex in V. We shall assume that such a graph represents a single-output combinational circuit with output R. An example of such a graph and the circuit it represents is given in Figures 2a and 2b. Any vertex in G is a dominator of another vertex in G if and only if every path from the root of the graph to the dominated vertex passes through the dominator vertex [Tarjan 74], [Lengauer 79]. In Figure 2a, for example, M dominates every vertex except R. Obviously, the root of such a graph always dominates every vertex. In our example other dominators exist: K dominates A and L dominates C. Dominators form a sequence, and the dominator of a vertex closest to that vertex is called the *immediate dominator*. In our example, R, M and K dominate A, but only K is the immediate dominator of A. Dominator computation is often used in compiler design [Aho 79], and, with appropriate modifications, the same algorithms can be used for our purpose.

Consider the circuit of Figure 2b. The effect of reconvergence from fanout B is totally accounted for at M, the point of total reconvergence for B. We will call the set of elements between the point of fanout and the point of total reconvergence the *reconvergence diamond*. In our example, the reconvergence diamond for fanout B consists of B, K, L, and M. Notice that the reconvergence diamond always terminates at the immediate dominator of the fanout point.

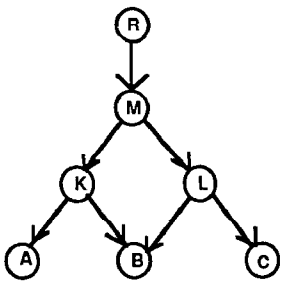

Figure 2a. Circuit Graph

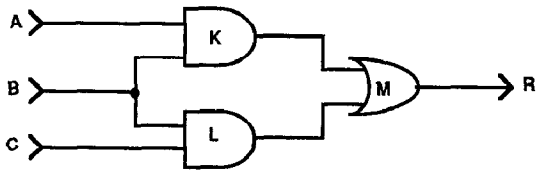

Figure 2b. Example Circuit

Finding Dominators

The dominators for a single-output graph such as Figure 2a can be found with the following simple algorithm:

1. Dom (R) = R;
2. For each vertex v in G except R until no change,
   Dom (v) = v + (& Dom (Pred (v)));
   Endfor;

Where R is the root of the graph, Dom (v) is the dominator of vertex v, Pred (v) is the set of immediate predecessors of v having more than one input, + is the union operator, and & is the intersection operator.

We illustrate the operation of this algorithm on the graph of Figure 2a. The process begins at R and proceeds breadth first through the graph. Assume that the nodes are processed in the order M, K, L, A, B, and C. Since the set of immediate predecessors of M contains only R, we perform the union of M + R and assign the result {M,R} as the dominators of M. K also has only one predecessor, M and the union of K + {M,R} forms {K,M,R} which is the set of dominators of K. Similarly, the set of dominators of L is {L,M,R} while the set of dominators of A is {A,K,M,R}. The predecessors of B are K and L. Performing the intersection of the dominators of K and L yields {M,R} and the union between B and {M,R} yields {B,M,R}. The dominators of C becomes {C,L,M,R}. This algorithm can be extended to multiple output circuits by executing it once for each output of interest.

A node which must be visited in observing a fault effect has previously been called a mandatory observation node. Now we shall call such a node an *absolute dominator*, defined as a dominator with respect to all reachable outputs. Observation of a node requires the simultaneous observation of every absolute dominator of that node. This implies that all inputs (not on the propagation path) to an absolute dominator gate must be set to a propagating value. Absolute dominators can be found from the dominator data by forming the intersection of the dominators for all outputs reachable from a node. For example, if we wish to know which elements are absolute dominators of node C in Figure 3, we simply intersect the dominator lists at C for all outputs reachable from C. The intersection is {C,L}, revealing that all paths from C to any output must pass through L.

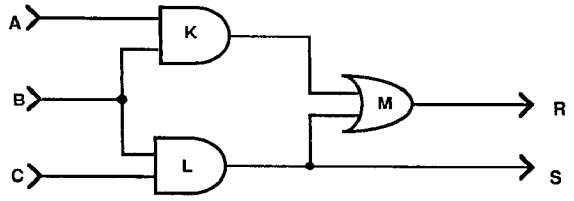

Figure 3. Dominators in Multiple Output Circuits

Therefore, by begining at the fault site and working toward the outputs, it is possible to identify the mandatory observation nodes for a given fault. We shall call this process *fault cone tracing*. Fault cone tracing often reveals redundant faults without searching, as we will now illustrate.

Detection of Redundant Faults

Consider Figure 4, a portion of the C432 circuit proposed as an ATPG benchmark [Brglez 85]. Assume that it is desired to find a test for a SA1 fault on the input to gate 429 from gate 393. The steps are: (1) Since the fault is SA1, the output of 393 must be "0" which implies that 417 is "1." (2) Since 429 and 432 are absolute dominators of the fault, their non-fault-cone inputs must be set to propagating values. Considering 429 first: 420, 407, and 386 must be set to "1." (3) The implications of these assignments cause 422 to become "0" and 432 to become "1." (4) Considering 432 next, its output must eventually be either an "X", "D", or "DB" since it is an absolute dominator of the fault, but its value has already been found by implication from mandatory values to be a "1." This conflict indicates that a test is not possible since all the assigned values are mandatory.

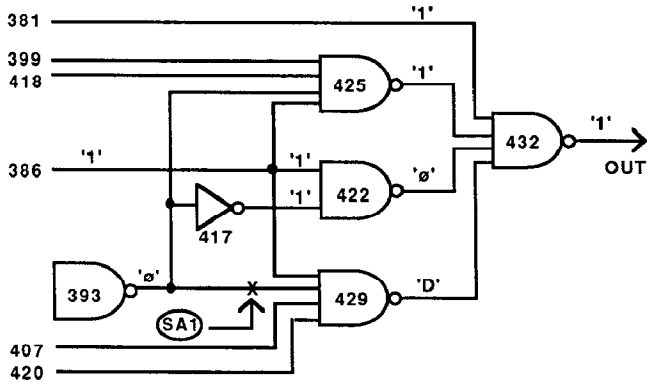

Figure 4. Redundant Fault Identification

Basis-nodes

In previous work, the size of the search space has been reduced through the use of headlines [Fujiwara 83]. We have identified a potentially larger and more general class of nodes with similar relevant properties. Points of complete reconvergence have the same independence property as headlines, since values can be assigned to these points independently. If a set of such independent nodes can be found that completely differentiates the good circuit from the faulty one, then that set of nodes forms a basis for ATPG search. Note that we do not require that a set of such nodes for one fault be identical with the set of such nodes for another fault. These nodes, which we shall call *basis-nodes*, can be computed from the dominator data since they are the immediate dominators of the fanout points.

Consider the circuit in Figure 5. It is obvious that all the primary inputs are independent. However, node K also has an independence property with respect to output R, since it is a point of total reconvergence for all fanouts that influence it (in this case fanout C only). Therefore, node K is a basis node for faults outside its reconvergence diamond since any value desired at K can be justified without affecting any line outside K's reconvergence diamond. The justification of node K can be deferred until the last step of the ATPG algorithm, just as headlines are.

The basis-nodes can be identified from the dominator data with a single forward pass, beginning at the primary inputs and proceding in forward level order. At each node an intersection of the predecessors' intersection lists is performed to find that node's intersection list. If the node of interest is in the resulting intersection, then it is a point of complete reconvergence and is a basis-node. The procedure must be repeated for each output of interest. Single-input gates such as buffers, inverters, etc., cannot be points of reconvergence and are handled separately with the intersection operation being replaced by a substitution of the immediate predecessor's intersection list with the immediate predecessor removed. If a node is a basis-node with respect to every output of interest, then it is independent and may be used as a member of the search basis.

In the circuit of Figure 5, the basis-nodes are found as shown in Table 1. Since the dominators for each primary input node are already known, we begin at the primary inputs and set their intersection lists to their dominator lists. Proceeding to the primary output by intersecting the intersection lists of each element's predecessors, we test each gate to see if it is a member of the resulting intersection list. If the element being computed appears on the intersection list, then it is a basis- node with respect to the output of interest. Since this example has only one output, references to the output are not included. As the table shows, the primary inputs A, B, C, D, and E all qualify as basis-nodes as do K and N.

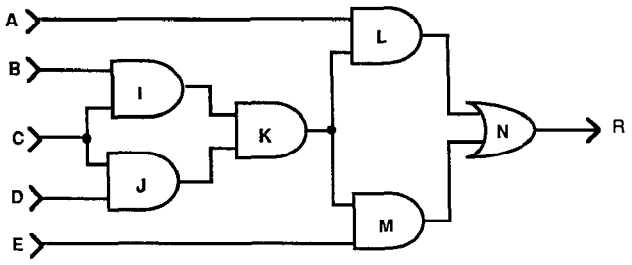

Figure 5. Identifying Basis Nodes

Element Name	Dominators	Immed. Pred.	Inter-section	Basis Node?
A	A,L,N,R	-	A,L,N,R	Y
B	B,I,K,N,R	-	B,I,K,N,R	Y
C	C,K,N,R	-	C,K,N,R	Y
D	D,J,K,N,R	-	D,J,K,N,R	Y
E	E,M,N,R	-	E,M,N,R	Y
I	I,K,N,R	B,C	K,N,R	N
J	J,K,N,R	C,D	K,N,R	N
K	K,N,R	I,J	K,N,R	Y
L	L,N,R	A,K	N,R	N
M	M,N,R	E,K	N,R	N
N	N,R	L,M	N,R	Y
R	R	N	R	Y

TABLE 1. Computation of Basis-Nodes

The Algorithm

The absolute dominator and basis-node data provide topological information that is used during ATPG. The new algorithm for ATPG using this information is shown in the flow chart of Figure 6. A preprocessing step is used to calculate the basis-nodes and absolute dominators. Then, an undetected fault is selected for test generation and the good and faulty circuits are initialized by setting the value of each node to X. Fault sensitization is then begun, which (1) sets the sensitizing value at the fault site, and (2) calculates the implications of this assignment to determine if a conflict exists. If such a conflict exists, the fault is redundant and the algorithm terminates. If no conflict occurs the algorithm enters fault cone tracing, illustrated in Figure 7. Fault cone tracing follows, in level order, every X path from the fault site to the primary outputs. At each gate a check is made to determine if the gate is an absolute dominator of the fault site. If the gate is an absolute dominator of the fault site, a propagating assignment is made to all inputs not on the fault path and the implications of these assignments are computed. If the implications produce a conflict, the algorithm terminates since no test is possible. If not, the algorithm proceeds to the search loop.

The search loop is an implicit enumeration procedure, similar to the PODEM [Goel 81] and FAN Algorithms [Fujiwara 83]. The major differences are in (1) the use of basis-nodes instead of headlines, (2) the use of nine value notation instead of five value [Muth 76], and (3) the ordering of fault propagation assignments through fault cone tracing. These differences are further discussed below. If, at any point in the search, a conflict is encountered, the stack is backed up and choices are remade until a non-conflicting assignment is found, at which point the search continues. If at any point a test is found, the algorithm terminates the search loop and enters the cleanup phase to resolve the basis node assignments. If the stack ever becomes empty, the algorithm terminates without a test since one cannot exist.

During this search phase, additional use is made of the information derived from fault cone tracing, specifically which nodes are in the potential fault observation path and which are not. Obviously if a node is not in the fault observation path it can never have an error indicating value. Therefore, such nodes are always assigned a value of either "0" or "1." On the other hand, if a node is on the fault observation path, it can be assigned a "G1", "G0", "F1", "F0", "D" or "DB" value, but should not initially be assigned a value of "1" or "0." Only during forward implication can a node in the fault observation path become a "1" or "0." If such a node is determined to have a "1" or "0" value during forward implication, then that observation path has become blocked and is removed from consideration. If all such observation paths become blocked, the stack must be backed up by remaking one or more node assignments.

The initial goal of the algorithm is to justify the implications of fault sensitization and fault cone tracing. Each justification is a separate subgoal that is satisfied by backtracing from the subgoal to a basis-node. An assignment is made to the basis-node and the implications of this assignment are found. This process continues until all the subgoals are achieved or the search stack becomes empty. If the search stack becomes empty then no test is possible. Once the sensitization and fault cone subgoals are justified, fault propagation begins. Since fault cone tracing has already sensitized all dominators along the fault observation path, the fault propagation process must only propagate the fault effect between dominators. This process proceeds by attempting to advance the current D-frontier toward the primary outputs along an X path. At each stage, a new goal is generated and backtraced to a basis-node. The value must be assigned to the basis-node and its implications determined to see if the goal has been achieved or a conflict has occurred.

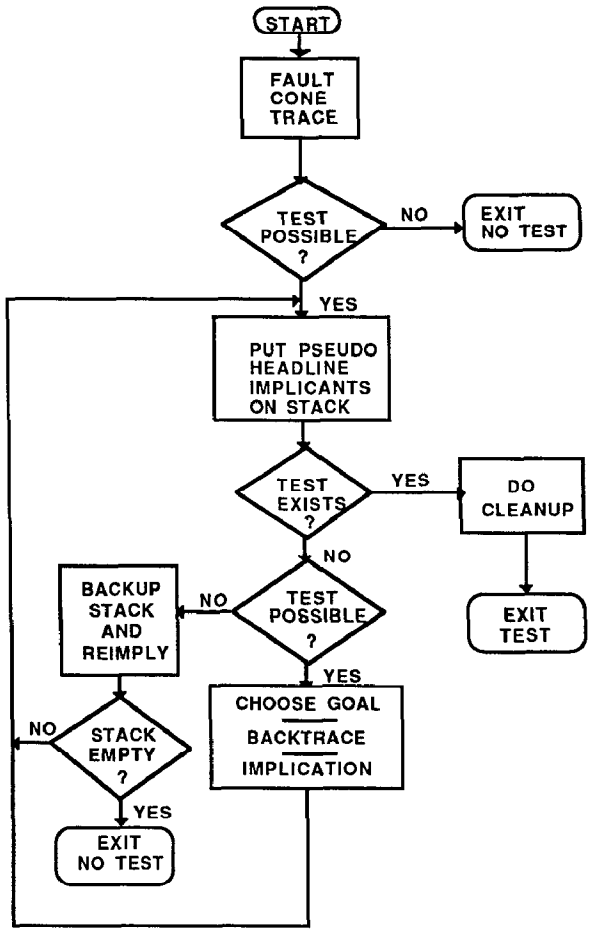

Figure 6. TOPS Algorithm Flow

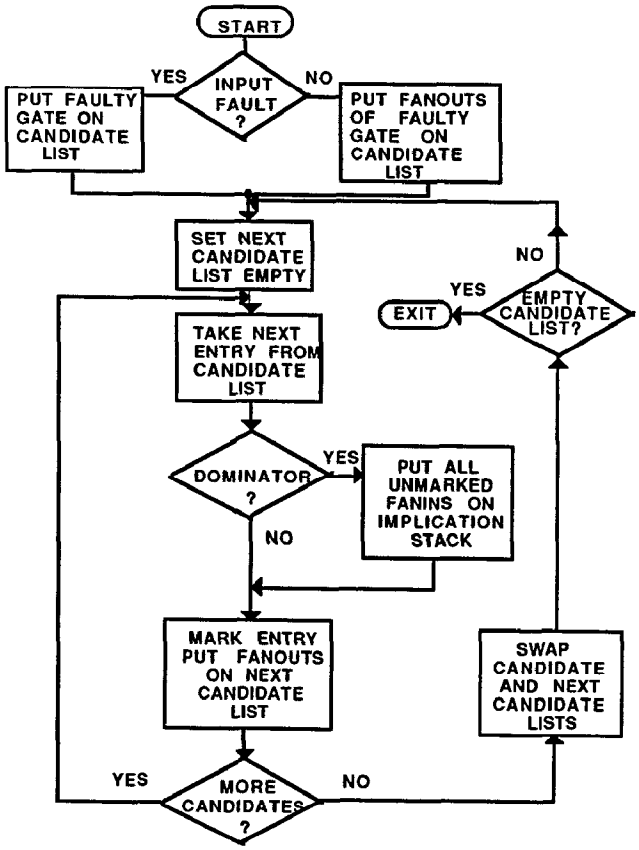

Figure 7. Fault Cone Tracing

Results

The TOPS algorithm (along with PODEM and FAN) was implemented in Lisp on a Symbolics 3600 (roughly equivalent in speed to a VAX 11/750) for purposes of comparison. The characteristics of the test circuits used are summarized in Table 2. Some of the faults in the original set are undetectable as shown in the last column of Table 2. Fault coverage was computed for a simple reduced fault set, that is not necessarily minimal. Except for the 74181, a 4-bit ALU, all the circuits were taken from the set proposed at ISCAS 85 as benchmarks for ATPG. Three different experiments were used to compare the algorithms and the results of these experiments are shown in below. The first experiment (Figure 8) required each algorithm to generate a test for every fault, or prove that the fault was redundant. The second experiment (Figure 9) required each algorithm to generate a test for each fault, but on a known detectable fault set, that is the original set with all redundant faults removed. The third experiment (Figure 10 and Table 3) used fault simulation after each test was generated to eliminate all fortuitously detected faults from further consideration. All times shown are normalized to the TOPS value.

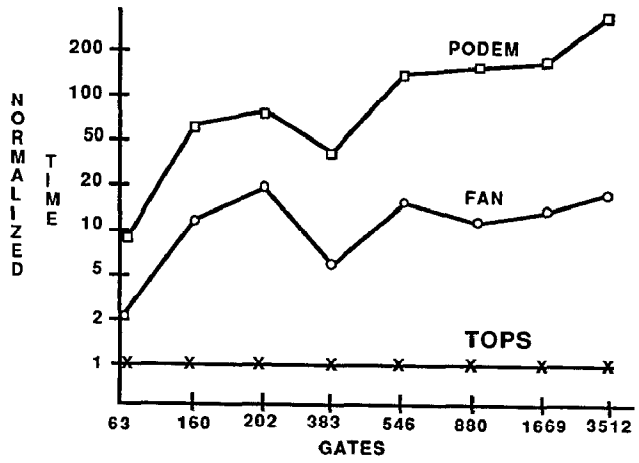

Figure 8. Algorithm Performance with Redundant Faults

#	CKT	GATES	PI's	FLTS	UNDET
1	74181	63	14	212	0
2	C432	160	36	524	7
3	C499	202	41	758	8
4	C880	383	60	942	0
5	C1355	546	41	1574	8
6	C1908	880	33	1879	7
7	C3540	1669	50	3428	132
8	C7552	3512	207	7550	141

TABLE 2. Circuit Characteristics

The third experiment also used a strategy of giving up after 100 remade decisions without a test [Fujiwara 85]. The known detectable fault set was used. The results in Figure 10 and Table 3 are, therefore, presented as a percentage of the detectable faults. Since each algorithm was implemented and executed in the same language on the same machine, the comparisons seem to be as fair as possible, but some cautionary notes are in order. First, no attempt was made to further optimize either PODEM or FAN over the original author's published algorithm. The use of nine value notation in TOPS, rather than the five values used by the original PODEM and FAN algorithms, could have caused some of the performance increase.

Finally, as Table 3 shows, the test set sizes produced by each algorithm varied. This appears to be an artifact of the order in which faults are considered, rather than a characteristic of any algorithm. Since the number of fortuitously detected faults is a function of the order of prior tests, it is difficult to derive meaningful information from the 1-3% differences in test set size.

CKT	PODEM		FAN		TOPS	
#	SIZE	%	SIZE	%	SIZE	%
1	19	100	21	100	20	100
2	66	96.1	77	97.5	64	100
3	73	100	67	99.9	67	100
4	72	100	68	100	63	100
5	124	99.6	110	99.8	114	100
6	181	99.4	157	99.6	151	99.9
7	203	96.8	197	98.3	205	99.1
8	276	98.2	281	98.5	278	99.1

TABLE 3. Algorithm Performance With Fault Simulation

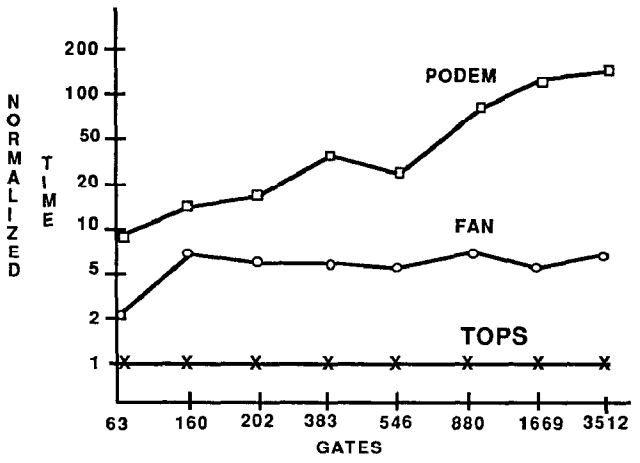

Figure 9. Algorithm Performance without Redundant Faults

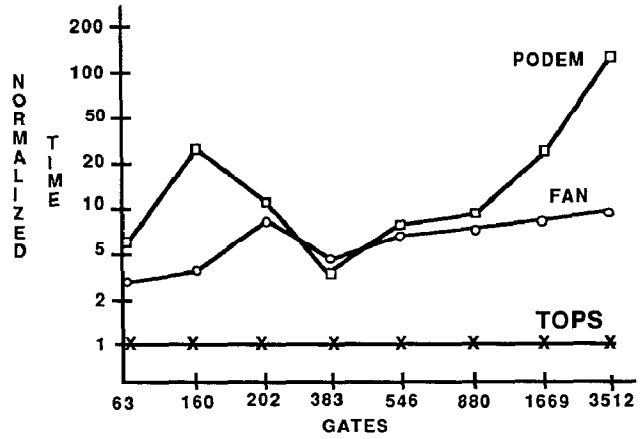

Figure 10. Algorithm Performance with Fault Simulation

Conclusions

The results indicate that topological ordering of the choices in a search method for ATPG is superior to an ordering based on circuit occurence alone. The TOPS algorithm is able to group nodes and their associated values according to a mandatory/optional dichotomy. This topological ordering not only produces a faster search algorithm, but also reveals many undetectable faults without searching. The combination of quicker resolution of most redundant faults along with superior ordering of node value assignments results in better ATPG times without loss of test set size or overall detection percentage. While ATPG algorithms can only truly be judged by their performance on a wide range of circuits, it is felt that the results obtained indicate that TOPS is faster and just as accurate as existing ATPG algorithms.

Acknowledgements

Special thanks are due to Dr. Jack Ferguson of MCC who provided many helpful suggestions and to Mr. Steve Vavasis, who first suggested the possible application of graph theory in ATPG. Thanks are also due to Mr. Arun Ramachandran who has subsequently implemented the TOPS algorithm in the C language which helped greatly in further understanding the algorithm's characteristics and performance. This work was sponsored in part by a grant from the Microelectronics and Computer Technology Corporation (MCC).

REFERENCES

[Aho 79] A. Aho and J. Ullman, Chapter 13 of PRINCIPLES OF COMPILER DESIGN, Addison-Wesley 1979.

[Brglez 85] F. Brglez, P. Pownall and R. Hum, "Accelerated ATPG and Fault Grading Via Testability Analysis," Proc. of ISCAS 85, June, 1985, pp. 695-698.

[Fujiwara 83] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," IEEE Trans. on Computers, Vol. C-32, December 1983, pp. 1137-1144.

[Fujiwara 85] H. Fujiwara, "FAN: A Fanout-Oriented Test Pattern Generation Algorithm," Proc. of ISCAS 85, June 1985, pp. 671-674.

[Goel 81] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", IEEE Trans. on Computers, Vol. C-30, Mar. 1981, pp. 215-222.

[Kirkland 86] T. Kirkland and M. Mercer, "A Two-Level Guidance Heuristic for ATPG," Proc. FJCC 86, November 1986, pp. 841-847.

[Kirkland 87] T. Kirkland and M. Mercer, "Automatic Test Pattern Generation Algorithms," IEEE Design and Test, to appear.

[Lengauer 79] T. Lengauer and R.E. Tarjan, "A Fast Algorithm for Finding Dominators in a Flowgraph," ACM Trans. on Prog. Lang. and Systems, Vol. 1, No. 1, July 1979, pp. 121-141.

[Muth 76] P. Muth, "A Nine Valued Circuit Model for Test Generation," IEEE Trans. on Computers, VOL C-25, No. 6, June 1976.

[Tarjan 74] R. Tarjan, "Finding Dominators in Directed Graphs," SIAM Journal of Computing, Vol. 3, pp. 62-89, 1974.