ON THE ACCELERATION OF TEST GENERATION ALGORITHMS

Hideo Fujiwara
Takeshi Shimono

Reprinted from IEEE 1983 FTCS 13TH ANNUAL INTERNATIONAL SYMPOSIUM FAULT—TOLERANT COMPUTING

ON THE ACCELERATION OF TEST GENERATION ALGORITHMS

Hideo Fujiwara and Takeshi Shimono

Department of Electronic Engineering
Osaka University
2-1, Yamada-Oka, Suita
Osaka, 565 JAPAN

ABSTRACT

In order to accelerate an algorithm for test generation, it is necessary to reduce the number of backtracks in the algorithm and to shorten the process time between backtracks. In this paper we consider several techniques to accelerate test generation and present a new test generation algorithm called FAN (FANout-oriented test generation algorithm). It is shown that FAN algorithm is faster and more efficient than the PODEM algorithm reported by Goel. We also present an automatic test generation system composed of the FAN algorithm and the concurrent fault simulation. Experimental results on large combinational circuits of up to 3000 gates demonstrate that the system performs test generation very fast and effectively.

I. INTRODUCTION

With the progress of LSI/VLSI technology, the problem of fault detection for logic circuit is becoming more and more difficult. As the logic circuits under test get larger, generating tests is becoming harder. Recent work has established that the problem of test generation even for monotone circuits is NP-complete [1]. Hence, it appears that the computation is, for the worst case, exponential with the size of the circuit. One approach to overcome this is to take several techniques known as design for testability. The techniques using shift registers such as LSSD [2], Scan Path [3], etc. allow the test generation problem to be completely reduced to one of generating tests for combinational circuits. Hence, for these LSSD type circuits it is sufficient to develop a fast and efficient test generation algorithm only for combinational circuits.

Many test generation algorithms have been proposed over the years [4]-[9]. The most widely-used is the D-algorithm reported by Roth [5]. However, it has been pointed out that the D-algorithm is extremely inefficient in generating tests for combinational circuits that implement error correction and translation functions. To improve this point, a new test generation algorithm called PODEM was recently developed by Goel [8]. Goel showed that the PODEM algorithm is significantly faster than the D-algorithm by presenting the experimental results. Indeed, the PODEM algorithms has succeeded in reducing the number of occurrence of backtracks in comparison with the D-algorithm. However, there still remain many possibilities to reduce the number of backtracks in the algorithm.

In order to accelerate an algorithm for test generation, it is necessary to reduce the number of occurrence of backtracks in the algorithm and to shorten the processing time between backtracks. In this paper we consider several techniques to accelerate test generation and present a new algorithm for generating tests called FAN (FANout-oriented test generation algorithm). FAN is a complete algorithm in that it will generate a test if one exists. Experimental results on large combinational circuits of up to 3000 gates demonstrate that the FAN algorithm is faster and more efficient than the PODEM algorithm over these circuits. We also present an automatic test generation system composed of the FAN algorithm and the concurrent fault simulation [10], which performs test generation very fast and effectively over the above large combinational circuits.

II. ACCELERATION OF ALGORITHM

We assume that the readers are familiar with the D-algorithm and the PODEM algorithm, and so we shall use some terminologies such as D-frontier, D-drive, implication, line justification, backtrace, etc., without definitions (see [5,8] for definitions). In this paper, we shall consider multi-input and multi-output combinational circuits composed of AND, OR, NAND, NOR and NOT gates. The type of fault model assumed here is the standard stuck fault, i.e., all faults can be modeled by lines which are stuck at logical 0 (s-a-0) or stuck at logical 1 (s-a-1). A fault consisting of a single stuck line is called a single fault. We shall focus our attention only on detecting single stuck faults.

In this section, aiming at the acceleration of test generation, we shall point out some defects of the PODEM algorithm and consider several effective techniques to eliminate these disadvantages.

In generating a test, the algorithm creates a decision tree in which there is more than one choice available at each decision node. The initial choice is arbitrary but it may be necessary during the execution of the algorithm to return and try another possible choice. This is called a backtrack. In order to accelerate the algorithm, it is necessary

a) to reduce the number of backtracks, and

b) to shorten the processing time between backtracks. Particularly, the reduction of the number of backtracks is important.

Heuristics should be used to achieve an efficient search. The PODEM algorithm adopted heuristics in the backtrace operation as follows:

- If the current objective level is such that it can be obtained by setting any one input of the current gate to a controlling state, e.g., 0 for an AND/NAND gate or 1 for an OR/NOR gate, then choose that input which can be most easily set.

- If the current objective level can only be obtained by setting all inputs of the current gate to a non-controlling state, e.g., 1 for an AND/NAND gate or 0 for an OR/NOR gate, then choose that input which is hardest to set, since an early determination of the inability to set the chosen input will prevent fruitless time spent in attempting to set the remaining inputs of the gate.

In this heuristics we can use controllability measures. In the PODEM algorithm as well as the D-algorithm, the D-frontier usually consists of many gates, and a choice of which gate to D-drive through next must be made. To decide this choice, PODEM adopted heuristics using a very simple observability measure as follows:

- As a D-frontier to D-drive, choose a gate closest to a primary output.

In this heuristics we can use other observability measures. Methods for determining controllability/observability measures are available in published literature such as [11].

In order to reduce the number of backtracks, it is important to find the nonexistence of the solution as soon as possible. In the "branch and bound" algorithm, when we find that there exists no solution below the current node in the decision tree, we should backtrack immediately to avoid the subsequent unnecessary search. The PODEM algorithm seems to lack the careful consideration in this point.

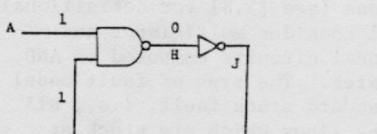
- In each step of the algorithm, determine as many signal values as possible which can be <u>uniquely</u> implied.

To do this we take the implication operation which completely traces such signal determination both forwards and backwards through the circuit. Moreover, we can take other techniques as follows:

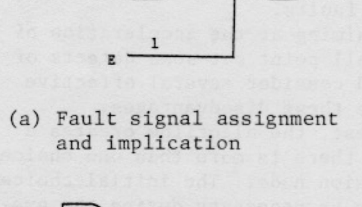
- Assign a faulty signal value D or $\overline{D}$ which is uniquely determined or implied by the fault under consideration.

As an example, consider the circuit of Fig. 1. For the fault L s-a-1, we assign the value $\overline{D}$ to the line L and 1's to the inputs J, K, and E. Then after the implication operation we have a test pattern for the fault without backtracks as shown in Fig. 1(a). On the other hand, in PODEM, the initial objective $(L,0)$ is determined to set up the faulty signal $\overline{D}$ to the line L, and then the backtrace procedure starts. As shown in Fig. 1(b), the backtrace procedure causes a path to be traced from the initial objective line L backwards to a primary input B. The assignment B=0 implies L=1. This contradicts the initial objective, and setting L to $\overline{D}$ fails and a backtrack occurs. As seen in this example, the assignment of Fig. 1(a) is a condition necessary for a test of the fault L s-a-1. By assigning the values which are uniquely determined we can avoid the unnecessary choice.

Consider the circuit of Fig. 2(a). Suppose that the D-frontier is {G2}. When the D-frontier consists of a single gate, we often have specific paths such that every path from the site of D-frontier to a primary output always goes through those paths. In this example, every path from the gate G2 to a primary output passes through the paths F-H and K-M. In order to propagate the value D or $\overline{D}$ to a primary output we have to propagate the fault signal along both F-H and K-M. Therefore, if there exists a test in this point, paths F-H and K-M should be sensitized. Then we have the assignment C=1, G=1, J=1, and L=1 to sensitize them. This partial sensitization which is uniquely determined is called a <u>unique sensitization</u>. In Fig. 2(a) after the implication of this assignment we have A=1, B=0, F=$\overline{D}$ and

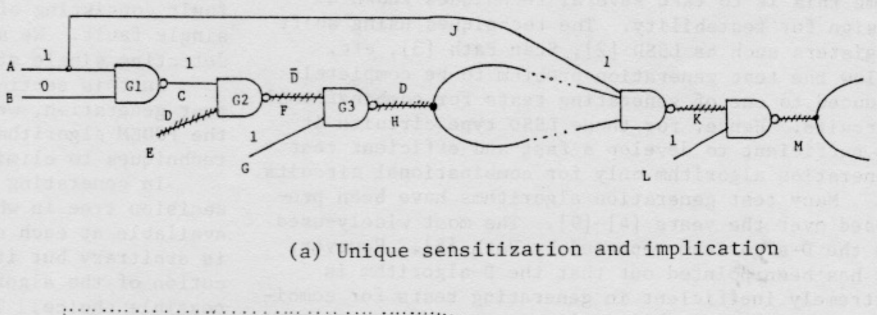


(a) Fault signal assignment and implication

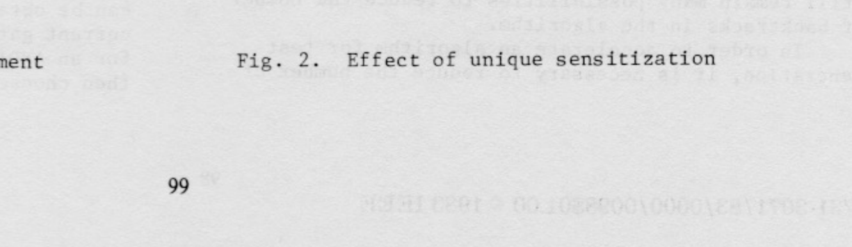


(b) PODEM

Fig. 1. Effect of fault signal assignment



(a) Unique sensitization and implication



(b) PODEM

Fig. 2. Effect of unique sensitization

H=D without backtracks. On the other hand, PODEM sets the initial objective (F,0) to propagate the fault signal to the line F and performs the backtrace procedure. If the backtrace performs along the path as shown in Fig. 2(b), we have to assign 0 to A, and we have J=0 and K=1 by the implication. Though no inconsistency appears in this point of time, an inconsistency or the disappearance of the D-frontier will occur in the future when the fault signal propagates from H to K.

● When the D-frontier consists of a single gate, apply a unique sensitization.

As seen in the above examples, in order to reduce the number of backtracks it is very effective to find as many values as possible which are uniquely determined in each step of the algorithm. This is because the assignment of the uniquely determined values could decrease the number of possible selection.

The execution of the techniques mentioned above may result in specifying the output of a gate G but leaving the inputs of G unspecified. This type of output line is called an unjustified line. It is necessary to specify input values so as to produce the specified output values. In PODEM, since all the values are first assigned only to the primary inputs and only the forward implication is performed, unjustified lines never appear. However, if we take the techniques mentioned above, the unjustified lines may appear and thus in this case some initial objectives will be produced simultaneously so as to justify them. This will be managed by introducing a multiple backtrace procedure which is an extention of the backtrace procedure of Goel [8].

When a signal line L is reachable from some fanout point, that is, there exists a path from some fanout point to L, we say that L is bound. A signal line which is not bound is said to be free. When a free line L is adjacent to some bound line, we say that L is a head line. As an example, consider the circuit of Fig. 3(a). In the circuit, A,B,C,E,F,G, H,and J are all free lines, and K,L, and M are bound lines. Among the free lines, J and H are head lines of the circuit since J and H are adjacent to the bound lines L and M, respectively.

The backtrace procedure in PODEM traces a single path backwards to a primary input. However, it suffices to stop the backtrace at a head line by the following reasons. The sub-circuit composed of only free lines and the corresponding gates is a fanout-free circuit since it contains no fanout-point. For fanout-free circuits, line justification can be performed without backtracks. Hence we can find the values on the primary inputs which justify all the values on the head lines without backtracks. It is sufficient, or even efficient, to let the line justification for head lines wait to the last stage of test generation.

● Stop the backtrace at a head line, and postpone the line justification for the head line to the later.

To illustrate this, consider the circuit shown in Fig. 3(a). Suppose that we want to set J=0 and don't know at the current stage that there exists no test under the condition J=0. In PODEM, the initial objective is set to (J,0) and the backtrace may result in the assignment A=1. Since the value of J is still not determined, PODEM starts again the backtrace procedure, and get the assignment B=0. A=1 and B=0 imply J=0. Here, by hypothesis, there exists no test under J=0, and thus an inconsistency occurs for the current assignment and PODEM must backtrack to change the assignment on B as shown in Fig. 3(b). In this case, if we stop the backtrace to the head line J, we can decrease the number of backtracks as shown in Fig. 3(c).

Performing a unique sensitization, we need to identify paths which would be uniquely sensitized. Also, we need to identify all the head lines in the circuit. These must be identified and those topological information should be stored in some manner before the test generation starts. The computation time of these preprocess can be, however, as small as negligible compared with the total computation time for test generation.

● Multiple backtrace, that is, concurrent tracing more than one path is more efficient than the backtrace along a single path.

Consider the circuit of Fig. 4. For the objective to set C=0, PODEM repeats backtrace three times along the same path C-B-A and also along the same path C-F-E before the value 0 is specified on C by the implication. Thus, we can see that the backtrace along a single path is inefficient and wastes the time which may be avoided. From this point of view, we could guess that the multiple backtrace along plural paths is more efficient than the single backtrace.

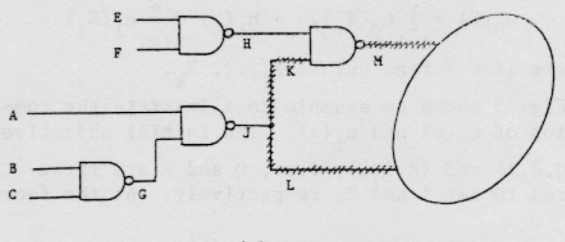
In the backtrace of PODEM, an objective is defined by an objective logic level 0 or 1 and an objective line which is the line at which the objective logic level is desired. An objective which will be used in the multiple backtrace is defined by a triple:

$$(s,\ n_0(s),\ n_1(s))$$

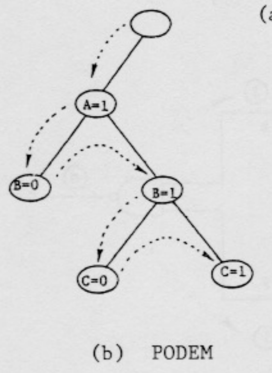
where

s is an objective line,

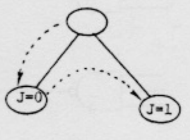
$n_0(s)$ is the number of times the objective logic level 0 is required at s, and



(a) Illustrative circuit



(b) PODEM



(c) Backtracking at head lines

Fig. 3. Effect of head lines

$n_1(s)$ is the number of times the objective logic level 1 is required at s.

The computation of $n_0(s)$ and $n_1(s)$ will be described later. The multiple backtrace starts with more than one initial objective, that is, a set of initial objectives. Beginning with the set of initial objectives, a set of objectives which appear in the midst of the procedure is called a set of current objectives. A set of objectives which will be obtained at head lines is called a set of head objectives. A set of objectives on fanout-points is called a set of fanout-point objectives.

An initial objective required to set 0 to a line s is

$$(s, n_0(s), n_1(s)) = (s,1,0)$$

and, an initial objective required to set 1 to s is

$$(s, n_0(s), n_1(s)) = (s,0,1)$$

Working breadth-first from these initial objectives backwards to head lines, we determine next objectives from the current objectives successively as follows:

1) AND gate: Let X be an input which is the easiest to control setting 0. Then
$$n_0(X) = n_0(Y), \quad n_1(X) = n_1(Y)$$
and for other inputs X_i
$$n_0(X_i) = 0, \qquad n_1(X_i) = n_1(Y)$$
where Y is the output of the AND gate.

2) OR gate: Let X be an input which is the easiest to control setting 1. Then
$$n_0(X) = n_0(Y), \quad n_1(X) = n_1(Y)$$
and for other inputs X_i
$$n_0(X_i) = n_0(Y), \quad n_1(X_i) = 0.$$

3) NOT gate:
$$n_0(X) = n_1(Y), \qquad n_1(X) = n_0(Y).$$

4) Fanout-point:
$$n_0(X) = \sum_{i=1}^{k} n_0(X_i), \qquad n_1(X) = \sum_{i=1}^{k} n_1(X_i)$$
where line X fans out to $X_1, \ldots, X_k$.

Fig. 5 shows an example to illustrate the computation of $n_0(s)$ and $n_1(s)$. The initial objectives are (Q,0,1) and (R,1,0), i.e., Q and R are first required to set 1 and 0, respectively. At the fanout point H, $n_1(H)$ is obtained by summing $n_1(K)$ and $n_1(L)$. The flowchart of Fig. 6 describes the multiple backtrace procedure. Each objective arrived at a fanout-point stops its backtracing while there exist other current objectives. After the set of current objectives becomes empty, a fanout-point objective closest to a primary output is taken out, if one exists. If the fanout-point objective satisfies the following condition, the objective becomes the final objective in the backtrace process and the procedure ends at the exit (D) in Fig. 6. The condition is that the fanout-point p is not reachable from the fault line and both $n_0(p)$ and $n_1(p)$ are nonzero. In this case, we assign a value (0 if $n_0(p) \geq n_1(p)$ or 1 if $n_0(p) < n_1(p)$) to the fanout-point and perform the implications. The first part of the condition is necessary to guarantee that the value assigned is binary, that is, neither D nor $\overline{D}$.

In PODEM, the assignment of a binary value is allowed only to the primary inputs. In our algorithm, FAN, we allow to assign a value to fanout-points as well as head lines, and hence the backtracking could occur only at fanout-points and head lines not at primary inputs. The reason why we assign a value to a fanout-point p is that there might exist a great possibility of an inconsistency when the objective in backtracing has an inconsistent requirement such that both $n_0(p)$ and $n_1(p)$ are nonzero. So as to avoid the fruitless computation, we assign a binary value to the fanout-point as soon as the objective involves a contradictory requirement. This leads to the early detection of inconsistency which would decrease the number of backtracks.

- In the multiple backtrace, if an objective at a fanout-point p has a contradictory requirement, that is, both $n_0(p)$ and $n_1(p)$ are nonzero, stop the backtrace so as to assign a binary value to the fanout-point.

When an objective at a fanout-point p has no contradiction, that is, either $n_0(p)$ or $n_1(p)$ is zero, the backtrace would be continued from the fanout-point. If all the objectives arrive at head lines, that is, both sets of current objectives and fanout-point objectives are empty, then the multiple backtrace procedure terminates at the exit (C) in Fig. 6. After this, we sort the set of head objectives in the descending order of n_0+n_1. Then taking out a head line one by one from the set of head objectives, we assign the correspoinding value to the head line and perform the implication. For details

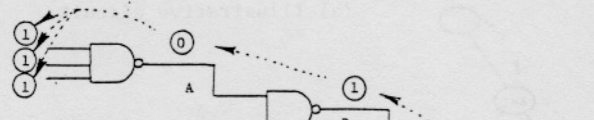


Fig. 4. Backtrace

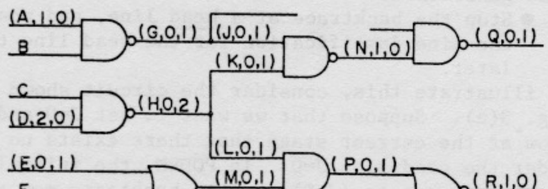


Fig. 5. Computation of n_0 and n_1

see the flowchart of FAN algorithm shown in Fig. 7.

III. DESCRIPTION OF THE FAN ALGORITHM

The FAN test generation algorithm is similar to PODEM based on the implicit enumeration process. However, the FAN algorithm characterizes to put empahsis on the following points:

1) FAN pays special attention to fanout-points in circuits.
2) FAN is a branch and bound algorithm which adopts many techniques presented in the preceding section so as to detect an inconsistency as early as possible.

As mentioned in the preceding section, those techniques would be very useful to decrease the number of backtracks, and thus FAN could be faster and more efficient than PODEM. The flowchart of the FAN algorithm is given in Fig. 7. Each box in the flowchart will be explained in the following:

1) Assignment of fault signal:
   In this box 1, a fault signal D or $\overline{D}$ which is uniquely determined is assigned.
2) Backtrace flag:
   The multiple backtrace procedure of Fig. 6 has two entries: One entry is (A) where the multiple backtrace starts from a set of initial objectives, and the other entry is (B) where the multiple backtrace starts with a fanout-point objective to continue the last multiple backtrace which terminated at a fanout-point. The backtrace flag is used to distinguish the above two modes.
3) Implication:
   We determine as many values as possible which can be uniquely implied. To do this we take the implication operation which completely traces such signal determination both forwrads and backwards through the circuit.
   In PODEM, since all the values are assigned only to the primary inputs and only the forward implication is performed, unjustified lines never appear. However, in FAN, since both forward and backward implications are performed, the unjustified lines might appear. Therefore, so as to justify those lines, the multiple backtrace is necessary not only to propagate the fault signal (D or $\overline{D}$) but also to justify the unjustified lines.
4) Continuation check for multiple backtrace:
   In this box 4, we check whether it is meaningful to continue the backtrace or not. We consider that it is not meaningful to continue the backtrace if the last objective was to propagate D or $\overline{D}$ and the D-frontier has changed, or if the last objective was to justify unjustified lines and all the unjustified lines have been justified. When it is not meaningful to continue, the backtrace flag is set so as to start the multiple backtrace with new initial objectives.
5) Unique sensitization:
   The unique sensitization is performed in the manner mentioned in the previous section (see Fig. 2). Although the unique sensitization might leave some lines unjustified, those lines will be justified by the multiple backtrace.
6) Determination of a final objective:
   The detailed flowchart of this box 6 is described in Fig. 8. By using the multiple backtrace procedure, we determine a final objective, that is, choose a value and a line such that the chosen value assigned to the chosen line has a good likelihood of helping towards meeting the initial objectives.
7) Backtracking:
   The decision tree is identical to that of PODEM, that is, an ordered list of nodes with each node identifying a current assignment of either 0 or 1 to one head line or one fanout-point, and the ordering reflects the relative sequence in which the current assignments were made. A node is flagged if the initial assignment has been rejected and the alternative is being tried. When both assignment choices at a node are rejected, then the associated node is removed and the predecessor node's current assignment is also rejected.

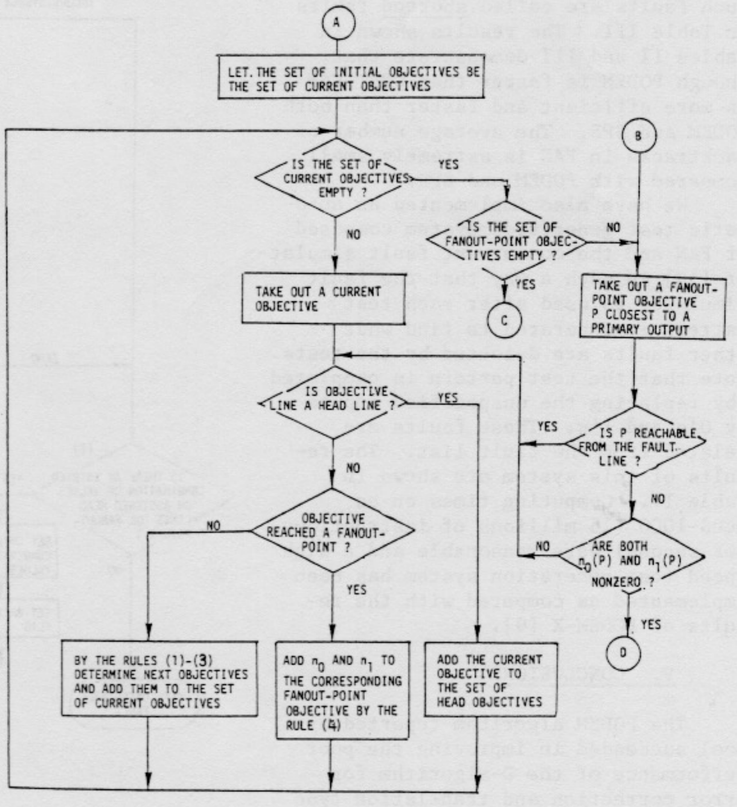

A
LET. THE SET OF INITIAL OBJECTIVES BE THE SET OF CURRENT OBJECTIVES
B
IS THE SET OF CURRENT OBJECTIVES EMPTY ?
YES
NO
IS THE SET OF FANOUT-POINT OBJECTIVES EMPTY ?
NO
YES
C
TAKE OUT A CURRENT OBJECTIVE
TAKE OUT A FANOUT-POINT OBJECTIVE P CLOSEST TO A PRIMARY OUTPUT
IS OBJECTIVE LINE A HEAD LINE ?
YES
YES
IS P REACHABLE FROM THE FAULT LINE ?
NO
NO
OBJECTIVE REACHED A FANOUT-POINT ?
NO
NO
ARE BOTH $n_0(P)$ AND $n_1(P)$ NONZERO ?
YES
YES
D
BY THE RULES (1)-(3) DETERMINE NEXT OBJECTIVES AND ADD THEM TO THE SET OF CURRENT OBJECTIVES
ADD n_0 AND n_1 TO THE CORRESPONDING FANOUT-POINT OBJECTIVE BY THE RULE (4)
ADD THE CURRENT OBJECTIVE TO THE SET OF HEAD OBJECTIVES

Fig. 6. Flowchart of multiple backtrace

8) Line justification of free lines:
   We can find the values on the primary inputs which justify all the values on the head lines without backtracks. This can be done by an operation identical to the consistency operation of the D-algorithm.

IV. EXPERIMENTAL RESULTS

We have implemented three programs, SPS (single path sensitization), PODEM, and FAN test generation algorithms, in FORTRAN on a NEAC System ACOS-1000. The SPS is a test generation algorithm which is restricted to sensitize only single paths and thus it is simpler than the D-algorithm. These programs were applied to a number of combinational circuits shown in Table I. The number of gates in these circuits ranged from 718 to 2982. The results are shown in Tables II and III. To obtain the data of Tables II and III, three programs were executed to generate a test for each stuck fault.

The number of times a backtrack occurs during the generation of each test pattern was calculated by the programs, and the average number of backtracks is shown in Table II. Since PODEM and FAN are complete algorithms, given enough time, both will generate tests for each testable fault. However, being limited in computing time, we gave up to continue test generation for the faults the number of backtracks exceeds 1000. Such faults are called aborted faults in Table III. The results shown in Tables II and III demonstrate that, though PODEM is faster than SPS, FAN is more efficient and faster than both PODEM and SPS. The average number of backtracks in FAN is extremely small compared with PODEM and SPS.

We have also implemented an automatic test generation system composed of FAN and the concurrent fault simulator [10] in such a way that the fault simulator is used after each test pattern is generated to find what other faults are detected by the tests. Note that the test pattern is completed by replacing the unspecified inputs by 0's and 1's. These faults are deleted from the fault list. The results of this system are shown in Table IV. Computing times on an ACOS-1000 (15 millions of instructions per second) were reasonable and a high speed test generation system has been implemented as compared with the results of PODEM-X [9].

V. CONCLUSIONS

The PODEM algorithm reported by Goel succeeded in improving the poor performance of the D-algorithm for error correction and translation type circuits. However, we have shown that there still remain many possibilities to reduce the number of backtracks in the algorithm. The FAN algorithm presented in this paper adopts many techniques to reduce the number of backtracks. Experimental results on large combinational circuits of up to 3000 gates show that the FAN algorithm is faster and more efficient than the PODEM algorithm in computing time, the number of backtracks and test coverage. An automatic test generation system composed of the FAN test generator and the concurrent fault simulator has also been implemented. The results on large circuits show that computing time on an ACOS-1000 were reasonable and the system achieved a high-speed test generation.

ACKNOWLEDGMENT

We would like to express our thanks to Profs. Hiroshi Ozaki and Kozo Kinoshita for their support and encouragement on this work.

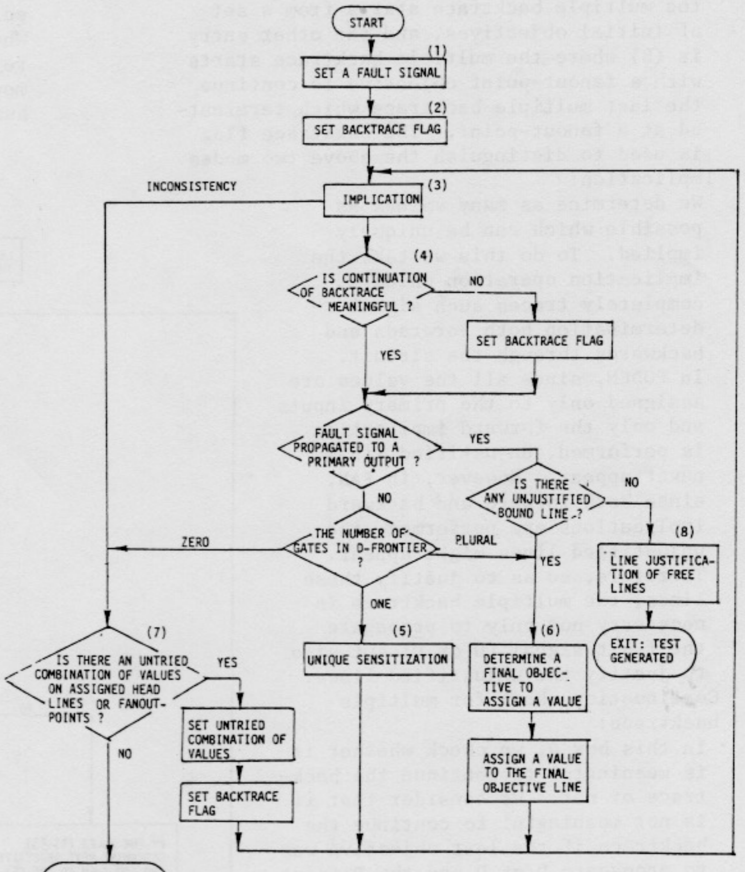

START
(1) SET A FAULT SIGNAL
(2) SET BACKTRACE FLAG
INCONSISTENCY
(3) IMPLICATION
(4) IS CONTINUATION OF BACKTRACE MEANINGFUL ?
NO
SET BACKTRACE FLAG
YES
FAULT SIGNAL PROPAGATED TO A PRIMARY OUTPUT ?
YES
NO
IS THERE ANY UNJUSTIFIED BOUND LINE ?
NO
YES
THE NUMBER OF GATES IN D-FRONTIER ?
ZERO
PLURAL
ONE
(8) LINE JUSTIFICATION OF FREE LINES
EXIT: TEST GENERATED
(7) IS THERE AN UNTRIED COMBINATION OF VALUES ON ASSIGNED HEAD LINES OR FANOUT-POINTS ?
YES
NO
SET UNTRIED COMBINATION OF VALUES
SET BACKTRACE FLAG
(5) UNIQUE SENSITIZATION
(6) DETERMINE A FINAL OBJECTIVE TO ASSIGN A VALUE
ASSIGN A VALUE TO THE FINAL OBJECTIVE LINE
EXIT: UNTESTABLE FAULT

Fig. 7. Flowchart of FAN algorithm

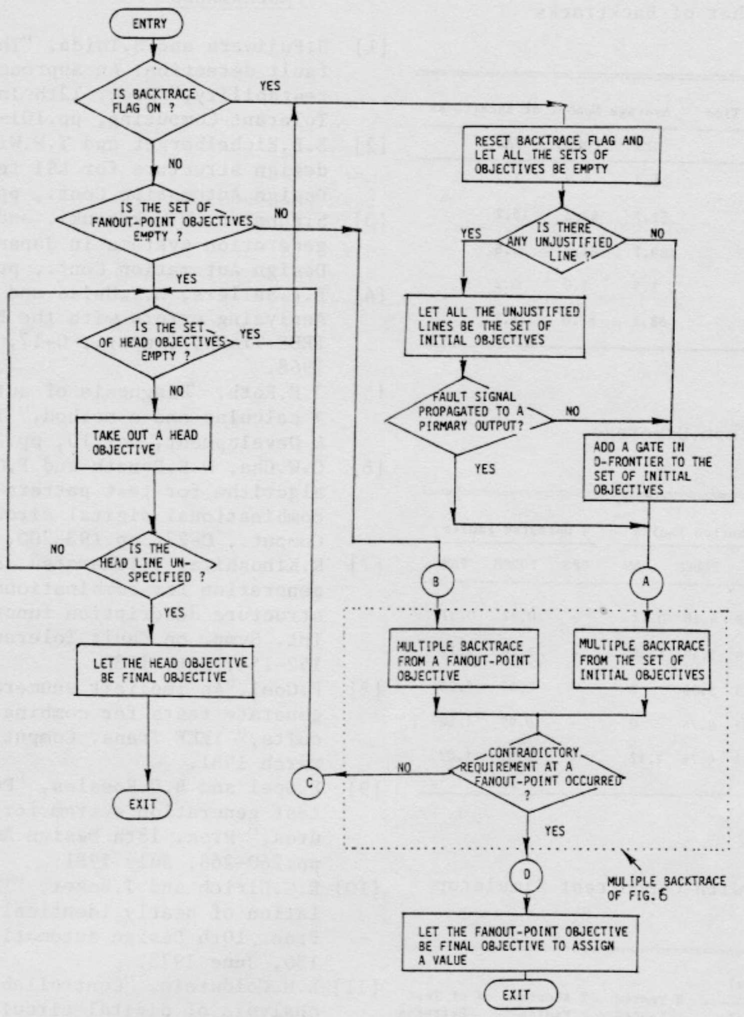

ENTRY
IS BACKTRACE FLAG ON ?
YES
NO
IS THE SET OF FANOUT-POINT OBJECTIVES EMPTY ?
NO
YES
IS THE SET OF HEAD OBJECTIVES EMPTY ?
YES
NO
TAKE OUT A HEAD OBJECTIVE
IS THE HEAD LINE UN-SPECIFIED ?
NO
YES
LET THE HEAD OBJECTIVE BE FINAL OBJECTIVE
C
EXIT
RESET BACKTRACE FLAG AND LET ALL THE SETS OF OBJECTIVES BE EMPTY
IS THERE ANY UNJUSTIFIED LINE ?
YES
NO
LET ALL THE UNJUSTIFIED LINES BE THE SET OF INITIAL OBJECTIVES
FAULT SIGNAL PROPAGATED TO A PIRMARY OUTPUT?
NO
YES
ADD A GATE IN D-FRONTIER TO THE SET OF INITIAL OBJECTIVES
B
A
MULTIPLE BACKTRACE FROM A FANOUT-POINT OBJECTIVE
MULTIPLE BACKTRACE FROM THE SET OF INITIAL OBJECTIVES
CONTRADICTORY REQUIREMENT AT A FANOUT-POINT OCCURRED ?
NO
YES
D
MULIPLE BACKTRACE OF FIG.6
LET THE FANOUT-POINT OBJECTIVE BE FINAL OBJECTIVE TO ASSIGN A VALUE
EXIT

Fig. 8. Determination of final objective

Table I. Characteristic of Circuits

Circuit	Number of Gates	Number of Lines	Number of Inputs	Number of Outputs	Number of Fanout-Points	Number of Faults
#1 Error Correcting Circuit	718	1925	33	25	381	1871
#2 Arithmetic Logic Unit	1003	2782	233	140	454	2748
#3 ALU	1456	3572	50	22	579	3428
#4 ALU and Selector	2002	5429	178	123	806	5350
#5 ALU	2982	7618	207	108	1300	7550

Table II. Normalized Computing Time and Average Number of Backtracks

Circuit	Normalized Computing Time			Average Number of Backtracks		
	SPS	PODEM	FAN	SPS	PODEM	FAN
# 1	5.2	1.3	1	31.2	4.9	1.2
# 2	4.5	3.6	1	51.7	42.3	15.2
# 3	14.5	5.6	1	189.7	61.9	0.6
# 4	3.1	1.9	1	1.5	5.0	0.2
# 5	3.4	4.8	1	38.1	53.0	23.2

Table III. Test Coverage

Circuit	% Tested Faults			% Aborted Faults			% Untested Faults		
	SPS	PODEM	FAN	SPS	PODEM	FAN	SPS	PODEM	FAN
# 1	99.04	99.20	99.52	0.48	0.48	0.11	-	0.32	0.37
# 2	91.15	94.25	95.49	4.70	3.49	1.38	-	2.26	3.13
# 3	66.25	92.53	96.00	16.25	5.05	0	-	2.42	4.00
# 4	98.77	98.75	98.90	0.07	0.26	0	-	0.99	1.10
# 5	94.73	94.38	96.81	3.54	4.79	2.17	-	0.82	1.02

Table IV. FAN Combined with Concurrent Simulator

Circuit	Computing Time (seconds)			% Tested Faults	% Aborted Faults	# of Test Patterns
	FAN	Concurrent Simulator	Total			
# 1	3.8	4.6	8.4	99.52	0.11	151
# 2	34.6	3.7	38.3	95.74	1.13	159
# 3	7.4	9.5	16.9	96.00	0	215
# 4	4.0	10.5	14.5	98.90	0	195
# 5	76.5	18.9	95.4	98.20	0.78	283

REFERENCES

[1] H.Fujiwara and S.Toida, "The complexity of fault detection: An approach to design for testability," Proc. 12th Int. Symp. on Fault Tolerant Computing, pp.101-108, June 1982.

[2] E.B.Eichelberger and T.W.Williams,"A logic design structure for LSI testing," Proc. 14th Design Automation Conf., pp.462-468, June 1977.

[3] S.Funatsu, N.Wakatsuki, and T.Arima,"Test generation systems in Japan," Proc. 12th Design Automation Conf., pp.114-122, June 1975.

[4] F.F.Sellers, M.Y.Hsiao and L.W.Bearnson, " Aanlyzing errors with the Boolean difference," IEEE Trans. Comput., C-17, pp.676-683, July 1968.

[5] J.P.Roth, "Diagnosis of automata failures: A calculus and a method," IBM J. of Research & Development, vol.10, pp.278-291, July 1966.

[6] C.W.Cha, W.E.Donath and F.Ozguner, "9-V algorithm for test pattern generation of combinational digital circuits," IEEE Trans. Comput., C-27, pp.193-200, March 1978.

[7] K.Kinoshita, Y.Takamatsu, and M.Shibata,"Test generation for combinational circuits by structure description functions," Proc. 10th Int. Symp. on Fault Tolerant Computing, pp. 152-154, Oct. 1980.

[8] P.Goel,"An implicit enumeration algorithm to generate tests for combinational logic circuits," IEEE Trans. Comput., C-30, pp.215-222, March 1981.

[9] P.Goel and B.C.Rosales, "PODEM-X: An automatic test generation system for VLSI logic structures," Proc. 18th Design Automation Conf., pp.260-268, July 1981.

[10] E.G.Ulrich and T.Baker, "The concurrent simulation of nearly identical digital networks," Proc. 10th Design Automation Conf., pp.145-150, June 1973.

[11] L.H.Goldstein, "Controllability/observability analysis of digital circuits," IEEE Trans. Circuits and Systems, CAS-26, pp.685-693, Sept. 1979.