DYTEST: A Self-learning Algorithm Using Dynamic Testability Measures to Accelerate Test Generation

Weiwei Mao **Michael D. Ciletti**

Department of Electrical Engineering
University of Colorado - Colorado Springs
Colorado Springs, CO 80933-7150

ABSTRACT

This paper presents a self-learning algorithm using a dynamic testability measure to accelerate test generation. It also introduces the concepts of full logic value label backward implication, the dependent backtrack and K-limited backtracks. Results indicating a high fault coverage are also presented for ten benchmark combinational circuits.

INTRODUCTION

Test generation algorithms [1-5] include some [3-5] which use static testability measures (STM) to guide the test generation (TG) process. These measures, like those in SCOAP [6], cannot account for backtrack-causing conflicts that can occur dynamically between the primary input assertions required for line justification and for path sensitization. Use of STM's can lead to repeated mistakes in the attempt to generate a test. Ivanov's DTM [7] is really the recalculation of the STM's with some lines having specific logic value during test generation. This recalculation can be time-consuming.

DYTEST is a self-learning algorithm using a DTM to accelerate test generation. The DTM guides the choice of a "potential main sensitizing path" during TG. A "full logic value label backward implication" creates estimates of logic values for the primary inputs. Dependent and k-limited backtracks are used to further reduce the number of backtracks.

DYTEST - AN OVERVIEW

The combinational circuit to be tested is composed of connected AND, NAND, OR, NOR and XOR gates having stuck-at-0/1 faults. All signal lines in the circuit are assigned a distinct index number according to the following rules: The index number of a gate's output line is larger than that of any input of the gate; the index number of a fanout stem is smaller than that of any fanout branch of the stem; the index number of any fanout branch of a stem is assigned according to SCOAP's static observability. At a stem, the index number of the fanout branch having a lower observability value is larger than that of the other fanout branches whose observability values are higher.

DYTEST begins by identifying and sensitizing a given fault. Then, it determines the unique logic implications - forward and backward - that result from the sensitization condition. This process reveals "MUST-BE" value labels that indicate, but do not effect, the logic values that must be assigned to some lines. Then, a potential main sensitizing path (PMSP) is chosen according to a testability measure which is calculated dynamically on the basis of the TG experience. In order to sensitize the PMSP, a full logic value label backward implication (FLVLBI) is done. This provides a basis for estimating the logic values of the primary inputs. Then a PODEM-like implicit enumeration process is done. At each time when a failure occurs, the dependent backtrack and K-limited deep backtracks are used to reduce the number of backtracks.

DYNAMIC TESTABILITY MEASURE

When a TG algorithm using STM's chooses a sensitizing path it sometimes correctly indicates a path that is relatively easy to observe. But if one choice is misled by the STM, the wrong choice may be made repeatedly at subsequent stages of the sensitization process. In contrast, DYTEST avoids repetitive incorrect choices by adjusting its DTM according to the experience acquired during the TG process.

Consider the situation in Fig. 1. The line LS is a fanout stem in a circuit; its fanout branches are LB1, LB2,..., LBK. Let F(LS) be a fault set containing faults in the shadow area, including faults at LS. If a TG algorithm selects a sensitizing path through LS, it faces the decision of choosing one of the branches emanating from the stem. If the algorithm remembers the past experience of the TG for the faults belonging to F(LS) it can choose the sensitizing branch having the highest success over failure rate, because that choice has the highest likelihood of leading to TG success. Based on this idea,

CH2540-3/88/0000/0591$01.00 © 1988 IEEE

a DTM can be defined as follows.

DEFINITION 1: Let each fanout branch in a circuit have success and failure records n_s(i) and n_f(i). During TG, if a chosen sensitizing path successfully leads to a valid test, the success record for each of the fanout branches in the path is incremented, otherwise the failure record is incremented. The DTM for a fanout branch j is formed as DTM(j) = n_s(j) * W_s - n_f(j) * W_f, with weights W_s, W_f subject to choice.

Three points should be noted here: i. The testability measure is updated according to the TG experience; ii. the DTM implicitly accounts for both the observability and the controllability of the circuit, even though it is only considered at the fanout -- because the DTM is updated by the success and failure experience of the TG process; iii. the DTM is easy to calculate.

FORCED FORWARD AND BACKWARD IMPLICATION

An initial step in TG sensitizes the fault for which a test is to be generated. For example, if L4 in Fig. 2 (a) has a S_A_0 fault, a logic value of 1 must be assigned to inputs L1, L2 and L3 in order to sensitize the fault. After a faulty line is sensitized, other lines must be assigned a logic value (0 or 1) to propagate the fault to a primary output. If fanout branch LB1 has a S_A_1 fault in Fig. 2(b), then to sensitize it and propagate it to the output a logic value of 0 must be assigned to LS, and a logic value of 1 must be assigned to L1. Also, a logic value of 0 must be assigned to LB2 and LB3 because 0 is assigned to LS. We say that these lines have "MUST-BE", or constrained, values. Other lines are assigned MUST-BE value if their logic value is uniquely implied.

In DYTEST, forced implications are done if there are lines to which MUST-BE values are uniquely implied. Its implication procedure is similar to the complete implication used in the D-algorithm and FAN, but it differs in the following important ways:(1) DYTEST conducts the forced forward and backward implications only once - after a faulty line is sensitized. (2) After conducting the forced forward and backward implications, DYTEST creates MUST-BE value labels for lines whose logic values are uniquely implied. These labels are used in the failure check and report phases of the algorithm. The failure report is used in the dependent backtrack.

THE POTENTIAL MAIN SENSITIZING PATH

In general, many paths exist for propagating a sensitized fault to a primary output. DYTEST chooses a PMSP according to its DTM.

DEFINITION 2: A path P is called a PMSP for a given fault if it satisfies the following conditions: i. the path P begins at a faulty line and ends at a primary output; ii. each line in the path P has logic value D, D* or X; iii. if several paths pass through a fanout stem and its branches seperately, and all of them satisfy conditions i. and ii. above, then the branch which the path P passes through has the highest dynamic testability value.

If a gate input along the PMSP has a value of X, then its logic value label is assigned in order to propagate the fault through the gate.

The logic value label for a line L is denoted as LVL(L). The label can be ZERO or ONE. The actual logic value for a line L is denoted as V(L). The logic value can be 0, 1, D, D* or X. For example, in Fig. 3 the PMSP is composed of L2, L4 , L5 and L8; lines L1 and L3 have logic value X. DYTEST assigns the logic value label ONE to L1 and ZERO to L3.

It should be noted here that the concept of logic value label is different from that of logic value. Logic value label ZERO (ONE) for a line l means that a logic value of 0 (1) may be required for the line in order to actually sensitize the PMSP.

FULL LOGIC VALUE LABEL BACKWARD IMPLICATION

After the PMSP is chosen and the logic value labels have been assigned to the other inputs of gates along the path, the FLVLBI is done for each gate type. Let Xi denote gate inputs, $1 \leq i \leq n$, and let Y denote gate output.

AND, OR Gate: If LVL(Y)=ZERO (ONE) and V(Xi)=X, then LVL(Xi)=ZERO (ONE).

NAND, NOR Gate: If LVL(Y)=ZERO (ONE) and V(Xi)=X, then LVL(Xi)=ONE (ZERO).

NOT Gate: If LVL(Y)=ZERO (ONE) and V(Xi)=X, then LVL(Xi)=ONE (ZERO).

XOR Gate:

a. If LVL(Y)=ZERO and V(X1)=V(X2)=X then LVL(X1)=LVL(X2)=ZERO or LVL(X1) = LVL(X2)=ONE

b. If LVL(Y)=ONE and V(X1)=V(X2)=X then LVL(X1)=ZERO, LVL(X2)=ONE or LVL(X1)=ONE, LVL(X2)=ZERO

c. If LVL(Y)=ZERO (ONE) and V(X2) (or V(X1))=0 then LVL(X1) (or LVL(X2))=ZERO (ONE)

d. If LVL(Y)=ONE (ZERO) and V(X2) (or V(X1))=1 then LVL(X1) (or LVL(X2))=ZERO (ONE)

e. If LVL(Y)=ONE or ZERO and V(X2) (or V(X1))=D or D* then LVL(X1) (or LVL(X2))=ONE or ZERO

At a fanout stem, the FLVLBI is done as follows:

Let the fanout stem be LS; its branches are LB1, LB2,..., LBK and N_of_0 (N_of_1) are the number of branches having logic value label ONE (ZERO). The label choice is decided by a "majority rule": CHOICE 1: LVL(LS)=ZERO if N_of_0 > N_of_1 and V(LS)=X; LVL(LS)=ONE if N_of_0 < N_of_1 and V(LS)=X.
CHOICE 2: LVL(LS)=ZERO if N_of_0 > N_of_1 and V(LS)=X; LVL(LS)=ONE if N_of_0 ≤ N_of_1 and V(LS)=X.

If a faulty line L is an output of an AND (or NAND) gate that is S_A_1 (or S_A_0), then the FLVLBI is done for the gate as if LVL(L)=ZERO (or ONE) is assigned to it. If a faulty line L is an output of an OR (or NOR) gate that is S-A-0 (or S-A-1), then the FLVLBI is done for the gate as if LVL(L)=ONE (or ZERO) is assigned to it. If a faulty line L is an output of XOR gate that is S_A_1 (S_A_0), the FLVLBI is done as if LVL(L)=ZERO (ONE) is assigned to it. If a MUST-BE 0 (or MUST-BE 1) label is assigned to an output L of a gate and some of the gate's inputs have logic value X, then the FLVLBI is done as if LVL(L)=ZERO (ONE) is assigned to it.

When a primary input is met in the FLVLBI, it is pushed into a first-in-last-out stack for later use in constructing a binary decision tree. The FLVLBI is used to estimate the logic value requirement for the primary inputs.

DEPENDENT BACKTRACK

When a failure occurs in the TG process, it is important that the algorithm backtracks as little as possible. Most algorithms, including PODEM and FAN, only backtrack to the last choice and take an alternative choice. For example, the binary tree for the backtracking of the circuit of Fig. 4(a) is illustrated in Fig. 4(b), where X1=1, X2=1, X3=1 and X4=0 are hypothetical initial choices for the primary input logic values. These result from FLVLBI, where a failure is assumed to occur at X4=0. The failure at X4=1 is due to violation of the MUST-BE 0 shown. A backtracking algorithm that examines the choices X3=0 and X2=0 is meaningless because these primary inputs have *no* influence on the line having the MUST-BE 0 requirement.

If a line having a MUST-BE 0 (1) label changes its logic value to 1 (0), or if a line at the PMSP has logic value 1 or 0, then a failure line report is given. If there is more than one failure line, the failure line with lowest index number is reported.

DEFINITION 3: Let L_fail be a reported failure line. A primary input set PI(L_fail) is called the dependent primary input set of line L_fail if for any $PI_i \in$ PI(L_fail), there is at least one path from PI_i to L_fail *and no* line in the path is a faulty line or has a MUST-BE value label. The backtrack in which the alternative choices are all members of PI(L_fail) is called the dependent backtrack.

In the example of Fig. 4(a), the dependent primary input set for the MUST-BE 0 line condition failure consists of X1 and X4. Backtracking on the basis of this set leads to the efficient backtrack shown in Fig. 4(c).

K-LIMITED DEEP BACKTRACKS

DEFINITION 4: Let A and B be primary inputs. Let node A (V(A) ∈ {0,1}) have descendants B (V(B)=0) and B (V(B)=1) in the TG binary decision tree shown in Fig. 5. If a failure appears at B (V(B)=0) and if the algorithm backtracks to A and chooses B (V(B)=1), then primary input A is said to have a one-level backtrack. If a failure appears at a level lower than B (V(B)=0) and if the algorithm backs to A and chooses B (V(B)=1), then the primary input A is said to have a deep backtrack.

To avoid too many local backtracks, DYTEST uses a heuristic method to limit the depth of the backtrack at each primary input to a specified value, K. For example, if K=1 for the binary decision tree shown in Fig. 6, the algorithm initially chooses nodes A (V(A)=1), B (V(B)=1), D (V(D)=1), E (V(E)=0), G (V(G)=0) and H (V(H)=0). A failure appears at H (V(H)=0) then again at H (V(H)=1). DYTEST then backs to E (V(E)=0) and chooses G (V(G)=1). If a failure appears at G (V(G)=1), then the algorithm backs to D (V(D)=1) and chooses E (V(E)=1). If a failure also appears at E (V(E)=1), then the algorithm backs to B (V(B)=1) and chooses D (V(D)=0), E (V(E)=0), G (V(G)=0) and H (V(H)=0). If a failure appears at H (V(H)=0) and H (V(H)=1) again, then the algorithm backs directly to A (V(A)=1) and chooses B (V(B)=0) because primary inputs D and E have been limited to 1 deep backtrack.

If the number of deep backtracks is limited to a fixed *number* K, the TG *algorithm* may be incomplete. However, if the selected number K is large enough, then the completeness of the algorithm is not affected. The number K can be determined according to the maximum backtrack number for practical TG.

TEST PROCEDURES AND TEST MODES

DYTEST is composed of two test procedures. One is a simple test procedure, another is a complete test procedure. The flowchart of DYTEST is shown in Fig. 7. Its steps

are explained as follows.
First, the fault is senstized. This step is similar to the D-algorithm and FAN. Then the forced forward and backward implication is executed. Then a PMSP is chosen using DYTEST's DTM. If no PMSP exists, the fault is untestable. Next the FLVLBI is made. Next, one of four test modes is invoked. These modes differ in their choice of label assignment at conflict fanout stems and the order of PIs in the stack. The test modes are denoted by TM(i,j) (i,j ∈ {0, 1}). Each mode makes choices in FLVLBI when a conflict fanout stem is met. These are given below:

<u>TM(0,0)</u>: the majority rule CHOICE 1 is made. No change is made in the order of PIs in the stack.
<u>TM(0,1)</u>: the majority rule CHOICE 2 is made. No change is made in the order of the PIs in the stack.
<u>TM(1,0)</u>: the majority rule CHOICE 1 is made. The order of the PIs in the stack is reversed.
<u>TM(1,1)</u>: the majority rule CHOICE 2 is made. The order of PIs in PI_stack is reversed.

After the test mode is chosen the forward logic implication is made. It should be noted that in Test Procedure 1, if the FI fails for an assignment of logic value, the complementary logic value is tried immediately. This step will cut the branches of the binary decision tree quickly.

The binary decision tree in DYTEST is identical to PODEM's. However, DYTEST uses the dependent backtrack and K-limited deep backtrack techniques to reduce the number of backtracks.

EXPERIMENTAL RESULTS

DYTEST is implemented on a VAX8600 in the C programming language. Ten benchmark combinational circuits were tested using only DYTEST -- no fault simulation was used.

Table 1 shows a comparison of the fault coverage provided by the STM's and DTM's after one pass, with the DTM using test procedure 1 with the test mode TM(0,0). The maximum backtrack number is 1. In every case the DTM achieves better fault coverage.

Next, all four passes of TG with test modes TM(0,0), TM(0,1), TM(1,0) and TM(1,1) were done using Test Procedure 1. The maximum backtrack number is 1. After each pass the detected faults are deleted from the fault list and the result is shown in Table 2. For the remaining undetected faults, another four passes with test modes TM(0,0), TM(0,1), TM(1,0) and TM(1,1) were done using Test Procedure 2 with 50 maximum backtracks and 1 limited deep backtrack for each primary input in a binary decision tree. This results in the final fault coverage shown in Table 3. The achieved fault coverages are better than those reported in [5]. Table 1 also shows that a single pass of the DTM requires less time than the STM.

CONCLUSION

DYTEST is a TG algorithm guided by a DTM. The experimental results show that DYTEST achieves higher fault coverage with a DTM than it does with a STM and it does so in less time. Its fault coverage compares favorably with other STM-driven TG [5].

DYTEST uses the dependent backtrack and K-limited deep backtrack techniques. The experimental results show that high fault coverage can be achieved in a few backtracks. The experimental results also show that the combinational use of four test modes and two test procedures is a good TG strategy. Additional speedup can be achieved by combining DYTEST with fault simulation.

REFERENCES

[1] J.P.Roth, "Diagnosis of automata failures: A calculus and a method", IBM J. Res. Develop., vol.10, pp.278-291, July 1966.
[2] C.W.Cha, W.E.Donath, and F.Ozguner, "9-V algorithm for test pattern generation of combinational digital circuits", IEEE Trans. Comput., vol.C-27, pp.193-200,Mar. 1978.
[3] P.Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits", IEEE Trans. Comput. vol.C-30, pp.215-222, Mar. 1981.
[4] H. Fujiwara, T.Shimono, "On the acceleration of test generation algorithm", IEEE Trans. Comput., vol.C-32, pp.1137-1144, Dec. 1983.
[5] S.Patel and J.Patel, "Effectiveness of heuristics for automatic test pattern generation", IEEE Proc. of 23rd Design Automation Conf., pp.547-552, June 1986.
[6] L.H.Goldstein, "Controllability /observability analysis of digital circuit," IEEE Trans. Circuit. Syst., CAS-26, No.9, pp.685-693, Sept., 1979.
[7] A.Ivanov and V.K.Agarwal, "Testability measures - what do they do for ATPG?", IEEE Proc. of International Test Conf., pp.129-138, Sept. 1986.

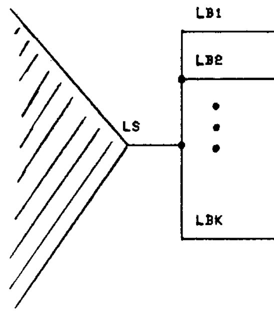

Fig. 1

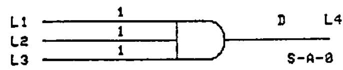

Fig. 2a

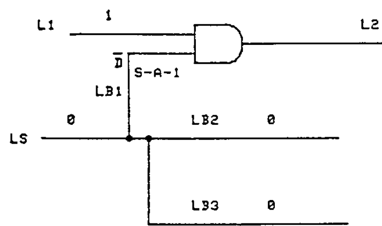

Fig. 2b

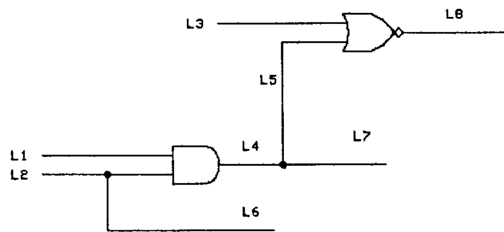

Fig. 3

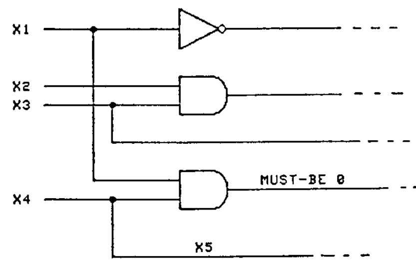

Fig. 4a

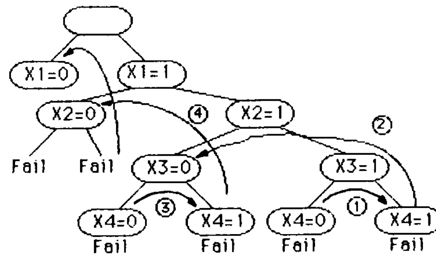

Fig. 4b

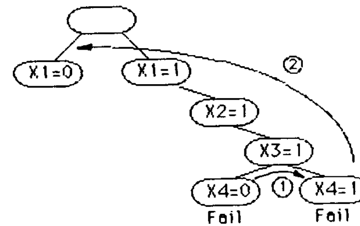

Fig. 4c

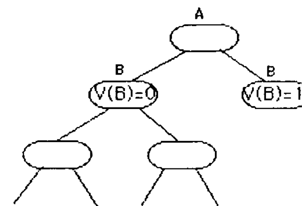

Fig. 5

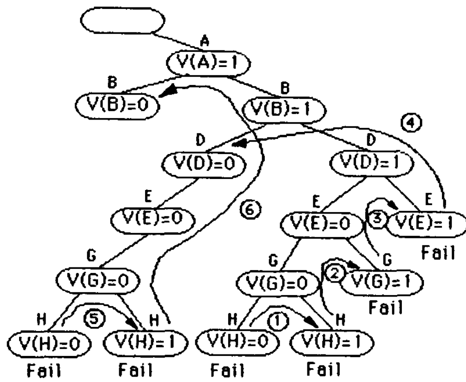

Fig. 6

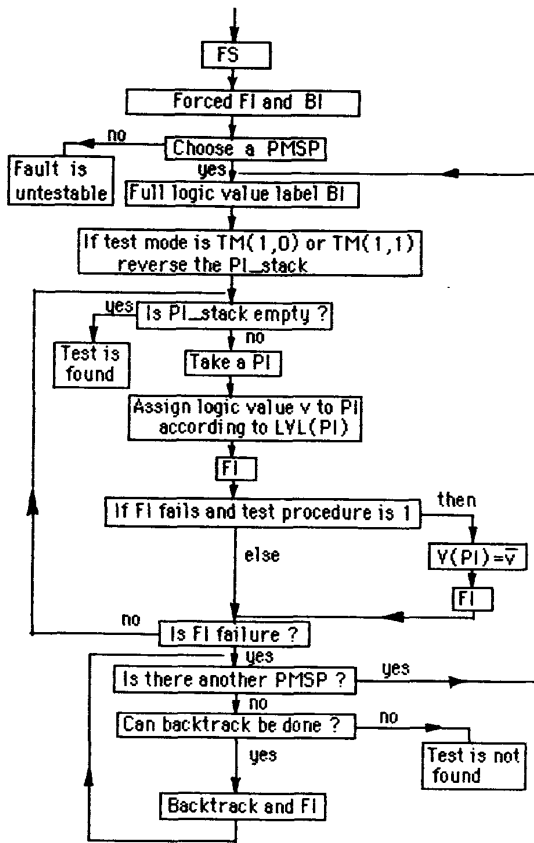

Fig. 7

TABLE 1

CIRCUIT	NUMBER OF FAULTS	FAULT COVERAGE IN %		NORMAL RUN TIME	
		STM	DTM	STM	DTM
C432	524	86.45	86.83	1.04	1.00
C499	758	67.94	78.63	1.16	1.00
C880	942	100.00	100.00	1.15	1.00
C1355	1574	76.87	79.54	1.09	1.00
C1908	1879	88.56	91.33	1.12	1.00
C2670	2747	71.20	83.91	1.02	1.00
C3540	3428	51.20	69.19	1.18	1.00
C5315	5350	92.58	94.52	0.98	1.00
C6288	7744	54.48	55.09	1.32	1.00
C7552	7550	73.17	77.97	1.15	1.00

TABLE 2

CIRCUIT	NUMBER OF FAULTS	FAULT COVERAGE IN %			
		T(0,0)	T(0,1)	T(1,0)	T(1,1)
C432	524	86.83	98.09	98.66	----
C499	758	78.63	86.94	98.15	98.42
C880	942	100.00	----	----	----
C1355	1574	79.54	87.04	97.27	98.79
C1908	1879	91.33	98.72	99.10	99.15
C2670	2747	83.91	90.57	91.92	93.52
C3540	3428	69.19	81.13	91.57	92.36
C5315	5350	94.52	96.90	97.85	98.19
C6288	7744	55.09	79.60	89.62	94.56
C7552	7550	77.97	83.60	94.97	96.19

TABLE 3

CIRCUIT	NUMBER OF FAULTS	FAULT COVERAGE IN %			
		T(0,0)	T(0,1)	T(1,0)	T(1,1)
C432	524	----	98.85	----	----
C499	758	98.94	----	----	----
C880	942	----	----	----	----
C1355	1574	99.24	----	99.49	----
C1908	1879	99.41	99.57	----	----
C2670	2747	95.23	95.63	95.74	95.78
C3540	3428	94.78	95.10	95.62	95.80
C5315	5350	98.58	98.64	98.73	98.77
C6288	7744	94.58	98.67	----	99.56
C7552	7550	97.01	97.31	98.11	98.19