Functional Test Generation for FSMs by Fault Extraction

Bapiraju Vinnakota*and Jason Andrews†
Department of Electrical Engineering, University of Minnesota, Minneapolis, MN 55455

Abstract

Recent results indicate that functional test pattern generation (TPG) techniques may provide better defect coverages than do traditional logic-level techniques. Functional TPG algorithms utilize a functional description of a circuit. Multi-level TPG algorithms attempt to realize the advantages of both approaches through fault translation. In such systems, gate-level faults are translated to functional faults and TPG is performed at the functional level. We develop and present new techniques for fast efficient fault translation from the logic to the functional level. These techniques are implemented in a multi-level sequential circuit test generation system. Performance results for benchmark circuits are presented.

I Introduction

Test sets are usually graded by the stuck-at fault coverage they provide. In product manufacture, it is the defect coverage that is of importance. Recent research indicates that for sequential circuits a high stuck-at fault coverage may not guarantee a high defect coverage [1, 2]. A test set which exercises the functionality of the circuit under test (CUT), or a mixed approach may provide a higher defect coverage than a purely stuck-at fault oriented test set [1]. This may be because functional test sequences expose defects not modeled by conventional fault models [1]. Hence, functional TPG is of practical interest. Functional TPG is based on a high level description of the CUT [2]-[6]. With any test set, gate-level fault simulation is still required to verify the coverage provided by, and to reduce the size of, functional test sets. One technique to both guarantee a high stuck-at coverage and retain the advantages of functional TPG, is test generation based on fault translation [3, 4]. That is, gate level faults are translated to functional faults and a functional TPG algorithm used to generate test sets. We concentrate on fault translation techniques for sequential circuits. However, the techniques developed may be extended to other applications as well.

The main contribution of the paper is the development of new efficient techniques to translate stuck-at faults to functional faults. We also discuss results in the area of sequential circuit fault simulation.

II Fault translation

The functionality of a sequential circuit is represented by a finite state machine (FSM). FSMs are described using state transition tables (STTs). A transition T is a 4-tuple, $< I, S_s, S_d, O >$, where, I and O are the input and output vectors, and S_s and S_d are the source and destination states. T is said to be incompletely specified if not all the bits in I are specified. T is said to be partially specified if not all the bits in O and/or S_d are specified. To translate a stuck-at fault to the functional level, all the transitions in the STT corrupted by the fault have to be identified. A stuck-at k fault, $k \epsilon \{0, 1\}$, f in an STT with incompletely specified transitions, is translated as follows [3, 4]. Each transition T_i is logic simulated in two phases with the faulty circuit. First, the circuit is simulated to the site of the fault. The logic value at the site of f may be k, unknown, or $\overline{k}$. The second phase of the simulation is executed only in the third case. After the second phase, if T_i is not corrupted by f *and* all of output and next state values in the faulty circuit are fully specified, T_i can be discarded. If, due to the fault f, some of the outputs and/or next state values in the faulty circuit are unknown, then an unspecified input in T_i has to be specified to set those values. As each input can be set in two ways, this results in two new transitions to be simulated. We extend this method to partially specified transitions.

A Partially Specified Transitions

Consider the two-phase simulation of a partially specified transition T_p with a stuck at k fault f, $k \epsilon \{0, 1\}$. As before, after phase 1, f is inserted only if the logic value at the site of the fault contradicts k. However, the results of the simulation are interpreted differently. After the second phase, if the logic value at the site of f is: (1) k, T_p can be discarded, (2) $\overline{k}$ and f corrupts T_p, then T_p is marked, (3) $\overline{k}$ and some outputs of the faulty or fault-free circuits are unknown or (4) x, and some outputs in the good circuit are unspecified, then an additional input will have to be set and the two new transitions derived from T_p simulated. Note that, often they will have to be simulated on both the good *and* the faulty circuits.

The process of simulating all the transitions in the STT will be referred to as a "pass." Two-phase translation contains redundant computation in each pass. If the logic value at the site of the fault does not contradict the stuck-at value, then all the simulation effort in phase 1 is redundant. In phase 2, signal values in the faulty circuit may become identical to those in the fault-free circuit only a few gate levels beyond the site of the fault. All work beyond this point is redundant. To improve efficiency redundant computation will have to be eliminated. Further, only two bits are required to adequately represent the signal values on every line in three-valued logic simulation. However, integer words are used in the logic simulator to represent the signal values on every line. Parallel pattern evaluation, using each bit position in the data word to simulate a different transition, can signifi-

*This research was supported in part by the University of Minnesota Graduate School

†Now with Tricord Systems Inc, Plymouth, MN

cantly accelerate fault extraction. Up to W transitions can be simulated simultaneously, where W is wordlength. However, after simulation to examine the values on the next state and output lines, they have to be unpacked. In large circuits this can be very time consuming.

B Multiple-Valued Logic Simulation

Multiple-valued logic simulation can reduce unpacking costs. In five-valued simulation, the possible logic values on a wire are $\{0, 1, D, \overline{D}, X\}$. Readers are referred to [7] for more details on five-valued logic. For each signal line j, three data words D_j, U_j, and S_j are used to store the signal values. When using parallel pattern evaluation, $S_j(i)$ represents the signal value on line j for transition $T(i)$. $D_j(i) = 1$ if the signal values in the fault-free and faulty circuits differ. That is, if the signal value is D or $\overline{D}$. With parallel pattern evaluation, the word D_j on line j is non-zero only if the faulty circuit has signal values different from those in the good circuit for at least one of the transitions. The value in the faulty circuit is $D_j(i) \oplus S_j(i)$. $U_j(i) = 1$ if the value of line j in either the fault-free or faulty circuit is x. Thus, for a given transition both the good and faulty circuits can be simultaneously simulated. This is beneficial when the STT contains partially specified transitions.

If a fault f corrupts a transition T then: (1) A D or a $\overline{D}$ is introduced at the site of the fault and (2) The D or $\overline{D}$ is propagated to the outputs of the circuit. In a sequential circuit, the outputs consist of both the primary outputs of the CUT as well as the next state lines. To minimize the extra computation associated with using three data words for each signal value, two separate gate evaluation routines are used. When none of the inputs to a gate have a D or a $\overline{D}$, the simpler evaluation technique corresponding to three-valued logic is used. Extra computation due to five-valued logic is minimized by limiting the use of the complex evaluation routine to the signal path from the fault site to the outputs. To examine if one of the transitions simulated is corrupted, the D words for all the output and next state lines are ORed (a bit-parallel operation). The resulting word need be unpacked to identify the corrupted transitions only if it is a non-zero integer. Since only a few transitions are corrupted by a particular fault, this reduces the need to unpack data words significantly. A similar technique can be used to examine U data words.

C Early Abort Translation

In the second phase of the simulation process, gates continue to be evaluated though all activity in the good and faulty circuits is identical. With five-valued simulation, such redundant computation can be eliminated. Consider the circuit in Figure 1. Let lines d and e be present state lines. Let the simulated transition be $T = < abc, de > = < 101, 10 >$ and the target fault be $f = b$ sa1. Clearly, f is excited by T. The $\overline{D}$ introduced is also propagated across gate G1 by the value on line a. However, the $\overline{D}$ does not propagate beyond the second level of gates. Any further gate evaluations are redundant as the transition is guaranteed to be uncorrupted.

In general, this redundant computation can be avoided by identifying a circuit level D_{MAX} beyond which D's and $\overline{D}$'s disappear. Given a gate G, let $L_F(G)$ be maximum value from among the levels of gates on the fanout of G. For example, the value of $L_F(G1)$ in Figure 1 is 2. Let the gate associated with the fault be G_f. Initially, at least phase 1 has to be executed. Thus, at the start of simulation D_{MAX} is set to the level of gate G_f. At gate G_f, if the fault is not excited, the simulation can be stopped. Else, its effects(D's or $\overline{D}$'s) will propagate to all the gates on the fanout of G_f. D_{MAX} is now set to $L_F(G_f)$. In general, if a gate G produces either a D or a $\overline{D}$ on its output, and $L_F(G)$ is higher than the current value of D_{MAX}, then D_{MAX} is reset to $L_F(G)$. During simulation, gates are visited in ascending order of their levels. Once the evaluation proceeds beyond level D_{MAX}, the simulation can be halted as all differences between the faulty and fault-free circuits have disappeared. For the example circuit, the initial value of D_{MAX} is 1. Once the faulty gate is evaluated D_{MAX} is set to 2. Since the fault is not propagated, Since D_{MAX} remains at 2 even after all level 2 gates are evaluated, the level 3 gate need not be evaluated. Thus, simulation can be halted almost as soon as differences disappear. With parallel pattern evaluation, the simulation aborts only when the activities for *all* transitions become identical to that in the good circuit.

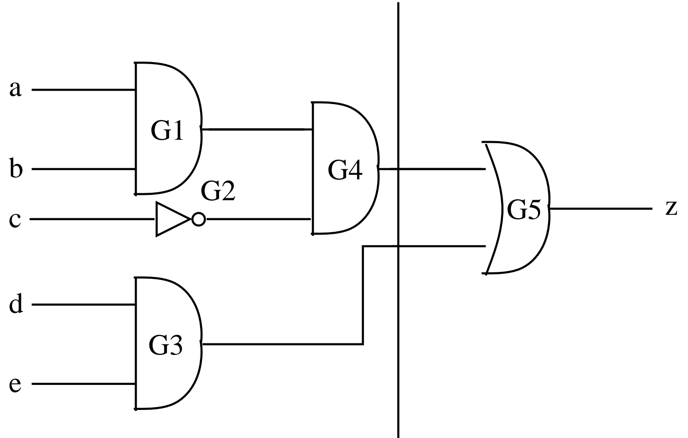

Figure 1: Monitoring D_{MAX}

D Gate-based Translation

Usually, on each pass only one fault is targeted for translation. If the logic value at the site of the fault does not contradict the stuck-at value the the first phase is wasted. The property of fault exclusion can be used to minimize this waste. Consider a three-input AND gate. Let the inputs be a, b and c and the output be z. The distinct non-equivalent faults associated with the gate are a sa1, b sa1, c sa1, z sa0 and z sa1. Consider the faults a sa1 and b sa1. They can never both corrupt the same transition. They are exclusive faults. For any completely specified transition T_c, only one out of the set of faults $S_e = \{a$ sa1, b sa1, c sa1, z sa0$\}$ can potentially corrupt T_c. If no fault in S_e is excited by T_c, only the fault z sa1 will corrupt T_c. If the transition is incompletely or partially specified, the output of the gate may not be specified and no fault will corrupt it. Sets of exclusive faults for common gate types are shown in Table I.

This feature can be used as follows. Instead of a single fault, on each pass *all the (remaining undetected) faults*

Table I: Exclusive fault sets for logic gates

Type	Exclusive fault set
AND	All input single s-a-1, Output s-a-0
NAND	All input single s-a-1, Output s-a-1
OR	All input single s-a-0, Output s-a-1
NOR	All input single s-a-0, Output s-a-0

Table II: Inserting exclusive faults

Line	T(1)	T(2)	T(3)	T(4)
U_a	1	0	0	0
S_a	0	1	0	0
U_b	0	0	0	0
S_b	1	0	1	1
U_c	0	0	0	0
S_c	1	1	1	0
D_z	0	1	1	0
U_z	1	0	0	0
S_z	0	0	0	0

associated with a single gate G_t are targeted for translation. For each transition, the values on the inputs to gate G_t are examined and the appropriate fault from the set S_e of exclusive faults is inserted. If no fault from S_e can be inserted and the gate output is specified, an output fault of appropriate polarity is inserted. Gate-at-a-pass translation can also be combined with parallel pattern evaluation. As shown below, faults can be inserted without unpacking the data words at the fault site. The combination of parallel-pattern evaluation and gate-based fault translation leads to a unique feature. *Now, the transitions in the set of W patterns may actually be translating different faults.*

E Fault Translation Example

For a word size of 4, let the values on the lines a, b, c and z, of the AND gate be as shown in Table II. Let the remaining undetected faults be a sa1, b sa1 and z sa0. The gate-input vectors which will detect them are abc = 011, 101 and 111 respectively. The following sequence of logical operations identifies all those transitions for which the gate-input vector is abc = 011 and also inserts the appropriate fault, line a sa1. (Let $\overline{S}$ be the complement of S. "+" (".") denotes a logical OR (AND).) Set $S_a = S_a + ((\overline{S_a}).(S_b).(S_c).\overline{(U_a + U_b + U_c)})$. Hence, S_a = 0100+(1011.1011.1110). $\overline{(1000 + 0000 + 0000)}$ = 0110. $T(3)$ is the only transition for which a sa1 should be inserted. Correspondingly, only that bit position in S_a has been flipped to indicate the insertion. Thus, the data words at the input to the target gate need not be unpacked for fault insertion. Using a similar technique, the fault b sa1 can be inserted for T(2). No fault is inserted for T(1) since the gate output is unspecified. No fault is inserted for T(4) since z sa1 has already been detected. Though only two faults have been inserted for two vectors, *all* the undetected faults have been simulated explicitly or implicitly with transitions T(1) through T(4).

This implies that on each pass of the STT, a gate rather than a single fault should be targeted for translation. Gate based translation would work as follows. The first time a gate G in the circuit is targeted for fault translation, the simulation of transitions starts at the first transition in the STT. Faults are inserted at the faulty gate using a bit-parallel comparison process. Suppose that a fault f on gate G corrupts an transition T' and that a test sequence to detect f is successfully generated. Then, when the translation process resumes, and if there are still undetected faults on gate G, the simulation can begin at the successor to T'. All the transitions up to T' have already been simulated, implicitly if not explicitly, with the remaining undetected faults on G. Hence, unlike the translation process in [3, 4] a significant amount of redundant computation is avoided.

III Implementation

The techniques in Section II have been implemented in multi-level TPG system. The system works as follows. Given a target gate, a stuck-at fault on the gate is translated to a transition fault. A functional TPG algorithm, which exercises only the specified functionality of the CUT is then used. A gate-level fault simulator is used to verify if other faults are detected by the same sequence. As do other TPG algorithms [8], we assume a reset state exists. Since STT descriptions were not available for benchmar circuits, they were extracted. In many circuits it was only feasible to partially enumerate transitions in the STT. A PODEM-based procedure [9] was used to extract the STT.

A Functional Test Generation

Given a stuck-at fault f, our technique is to identify a single component of the single or multiple transition fault created by f. The component fault is targeted for TPG using the algorithm, based on the faullt-free STT, in [2]. We have found that this strategy almost always succeeds. Given a stuck-at fault f, let the transition $< I_T, A_T, D_T, O_T >$ be one corrupted by f. The first step is to transfer the CUT from the reset state to the source state of the erroneous transition A_T. In a preprocessing phase the distances, from the reset state, of all the (reachable) states in the FSM are computed. The justification sequence is generated in reverse from state A_T to the reset state. Once the CUT is in state A_T, the test transition is exercised by applying the vector I_T. If the fault corrupts the output produced on the transition, TPG is complete. If, however, the fault only causes an erroneous next state, this error has to be propagated to the outputs of the circuit. Let the erroneous next state in the presence of the fault be F_T. A state pair differentiation sequence [2] for states D_T and F_T will propagate the errors in the internal state. As in [2], We use a breadth-first search algorithm to generate such a sequence.

The functional description of the CUT may only enumerate transitions for those states reachable from the reset state. The state F_T may not be one such state. Hence, it is possible that the transitions from the erroneous next state F_T may not be enumerated in the STT. Transitions from state F_T are enumerated using logic simulation. This approach minimizes computation (and the memory used) in two ways: 1) Transitions from states not reachable from the reset state are enumerated only as required, 2) The input vectors used for F_T are only those used in transitions defined for D_T. Parallel pattern evaluation was also used to enumerate the transitions from F_T. Note that as in [2], in all cases the fault-free CUT is used for error propagation. Ideally, one must use the faulty CUT [3, 4]. Hence, fault simulation is required to

verify if the target fault is detected by the sequence generated. In our experience we have found, as have others, this to be nearly always true.

B Fault Simulation

Sequential circuit fault simulation has received significant attention. Recent results include the development of the PROOFS fault simulator [10]. Parallel pattern single fault propagation (PPSFP) is a technique used, in general, to simulate combinational logic circuits [7]. The fault propagation techniques in PPSFP can also be used with parallel fault simulation. Consider the simulation of a vector V and W (single or multiple) faults. The vector V is logic simulated and the good circuit values are stored at every wire in the circuit. Let the set of faulty gates be N_f. Ideally, only those gates whose outputs could potentially be different in the good and one or more of the W faulty circuits should be evaluated. At each stage of the simulation, let G_{ev} be the set of gates remaining to be evaluated in the fault simulation. Initially, G_{ev} consists of the set N_f of faulty gates. Now, when a gate $g \epsilon G_{ev}$ is scheduled for evaluation, if the output of gate g *in any one* of W circuits is different from that in the fault-free circuit, the gates on the fanout of g are added to the G_{ev}. Once a gate in G_{ev} is evaluated, it is removed from G_{ev}. Simulation stops when G_{ev} is empty or when all faults have been detected. G_{ev} is empty only when the activity in *all* faulty circuits becomes identical to that of the good circuit. A simple event driven logic simulator can be used to implement propagation-based parallel fault simulation. In a logic simulator an event is defined to be a logic transition on a wire. In the context of enhanced parallel fault simulation, an event is said to occur when the signal values in the good and a faulty circuit differ.

This technique can be extended to sequential circuits. Let the sequence of vectors to be simulated be $\{V_0, V_1, \ldots, V_k\}$. Denote the set of gates remaining to be evaluated at each stage in the simulation of vector V_i by G^i_{ev}. First, all the vectors in the sequence are logic simulated on the good circuit. The node values in the good circuit for each vector in the sequence are stored. The storage requirements are minimized by storing the data in a bit-packed format. Recall that the states of the various faulty circuits have also to be tracked. If a reset signal exists, the initial states of the good and all faulty circuits are set to the reset state. For V_0, the present states in the good and all faulty circuits are identical. Hence, V_0 can be simulated as described above

However, for V_1, the values on the present state lines in a faulty circuit may differ from those in the fault-free circuit. Without the loss of any generality, assume that for one of the W faults, fault f on gate G_f, the signal value on some present state line ps differs from that in the good circuit. Then, for vector V_1, the fault f is actually treated as the multifault $f \cup ps$. In addition to gate G_f, all the gates on the fanout of ps will also have to be placed in G^1_{ev} prior to the start of the simulation. In fact, G^1_{ev} initially consists of the set of gates $N_f \cup G_N$, where G_N is the set of the gates on the fanout of all present state lines which have a different value from that of the good circuit in any one of the W faulty circuits being simulated. The simulation for a vector V_i ends when G^i_{ev} becomes a null set or all the W faults have been detected. To further reduce the number of gate evaluations, once a fault being simulated is detected, the logic values in the faulty circuit are forced to be identical to those in the good circuit. Thus, no gate associated exclusively with a detected fault can again be added to the set of gates to be simulated. The simulation for the sequence ends when all the k vectors have been simulated or when all faults have been detected.

Table III: Test generation times

circ name	STT Ext	flt tran	func tpg	flt sim	total time
s27	0.1	0.05	0.05	0.3	0.5
s298	0.3	4.28	7.44	0.28	12.3
s386	0.06	.65	0.07	0.56	1.38
s510	0.11	0.36	0.15	1.27	1.94
s820	0.17	7.39	0.19	3.59	11.4
s832	0.17	8.47	0.22	3.87	12.8
s1488	0.21	7.63	0.2	10.8	19
s1494	0.21	8.57	0.17	11	20.1
s344	0.4	7.51	0.25	0.19	8.4
s349	0.4	8.08	0.27	0.21	9
s382	58	172	5,230	8.22	5,470
s444	13.8	179	261	3.02	458
s526	17.2	651	2,060	17.4	2,750
s526n	17	637	2,030	16.8	2,700
s953	479	879	2,740	3.08	4,100
s1196	32.4	453	28,400	15.3	28,900
s1238	32.5	460	28,200	119	28,800

C Results

The performance of the TPG system was evaluated on the ISCAS [11] set of benchmarks. Readers are referred to [11] for benchmark circuit statistics. The results, generated on a Pentium-based computer, are shown in Tables III and IV. Statistics have also been gathered for partially enumerated state transition tables with partially specified transitions. In such tables, a transition fault is not guaranteed to be found for every target stuck-at fault. The translation time includes successful as well as unsuccessful fault translation efforts. In all cases, time limits have been set on each phase of the TPG process. Table V compares our multi-level TPG technique with the method in [5] (the symbol "-" indicates the unavailability of data). Effective fault coverages are used when the complete state table is available. Unlike the techniques used in [5] fault extraction has the added advantage of redundancy identification when the complete state table is available. Experimental results presented in [3, 4, 6] are limited to the MCNC benchmark set [12]. While a direct comparison is not possible, it is worth noting that the circuits in that set are smaller than those listed in Table III

The substantial benefits offered by using propagation techniques in parallel fault simulation are shown in Table VI. The percentage decrease in the number of evaluations is higher for larger circuits. The percentage decrease in the number of evaluations is significantly higher than the percentage decrease in the simulation time. In our implementation, the fault simulator visits every gate in the circuit to check if it is to be evaluated. The number of gate visits is shown in column 2. When the number of actual gate evaluations is very low, the cost of visiting

Table IV: Test generation performance

circ name	num flts	num det	num red	flt cov	eff cov	num vec
s27	32	32	0	100	100	20
s298	326	284	38	87.12	98.77	238
s386	401	330	71	82.29	100	287
s510	570	570	0	100	100	835
s820	839	746	78	88.92	98.21	788
s832	851	741	93	87.07	98.00	812
s1488	1561	1488	72	95.32	99.94	1497
s1494	1567	1485	81	94.77	99.94	1459
s344	367	338	-	92.1	-	127
s349	373	340	-	91.15	-	127
s382	428	388	-	90.65	-	900
s444	506	506	-	100.00	-	1202
s526	567	455	-	80.25	-	2066
s526n	567	457	-	80.60	-	2066
s953	1079	982	-	91.01	-	449
s1196	1341	721	-	53.77	-	147
s1238	1391	701	-	50.40	-	267

Table V: Fault coverage MLTPG vs. GDSEQ

Circuit	MLTPG	GDSEQ
s298	98.77	86.0
s386	100	81.7
s510	100	100
s820	98.81	94.0
s832	98.00	92.10
s1488	99.94	94.2
s1494	99.94	94.5
s344	92.1	90.6
s349	91.15	91.7
s382	90.65	87.2
s444	100	75.5
s526	80.25	67.6
s526n	80.60	65.6
s953	91.01	-
s1196	53.77	-
s1238	50.40	-

Table VI: Performance of enhanced parallel simulation

Circuit	Gate Visits	Par. Eval	Enh. Eval	% Reduction	
				Eval	Time
s298	8.3e4	6.2e4	2.0e4	67	47
s386	2.1e5	1.5e5	3.8e4	75	48
s820	1.8e6	1.5e6	2.0e5	87	62
s832	2.0e6	1.7e6	2.2e5	87	64
s1488	8.2e6	6.3e6	7.7e5	88	68
s1494	8.3e7	6.4e6	7.8e5	88	69
s344	6.1e4	4.1e4	1.7e3	58	38
s349	6.6e4	4.5e4	1.8e3	60	39

every gate in the circuit on every vector dominates fault simulation time. Using a levelized linked list to hold the gates in G_{ev} can address this problem.

IV Conclusion

We have presented new techniques for the fast translation of gate level faults to functional faults. These methods use parallel pattern evaluation, multiple valued logic simulation, early abort detection and novel gate-based fault translation techniques. These techniques were incorporated into a multi-level TPG system for sequential circuits. A very significant decrease in the time needed to translate faults was obtained. The fast translation techniques enabled the system to generate functional test sets for circuits much larger than those addressed by previous functional algorithms. We also presented new fault simulation techniques for sequential circuits.

References

[1] P.C. Maxwell and R.C. Aitken, "All fault coverages are not created equal," *IEEE Design and Test of Computers*, pp. 42-51, March 1993.

[2] K-T. Cheng and J.Y. Jou, "A Functional fault model for sequential machines," *IEEE T-CAD*, pp. 1065-1073, Sep. 1992.

[3] I. Pomeranz and S.M. Reddy, "On achieving complete fault coverage for sequential machines using the transition fault model," in *Proc. DAC*, June 1991, pp. 341-346.

[4] I. Pomeranz and S.M. Reddy, "Test generation for synchronous sequential circuits based on fault extraction," in *Proc. ICCAD*, pp. 450-453, 1991.

[5] M.K. Srinivas, J. Jacob, and V.D. Agrawal, "Finite state machine testing based on growth and disappearance faults," in *Proc. FTCS*, 1992, pp. 238-245.

[6] G. Buonanno, F. Fummi, and D. Sciuto, "Functional fault models and gate level coverage for sequential architectures," in *Proc. ICCD.*, pp. 572-575, Oct. 1993.

[7] M. Abromovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, Rockville, MD, Computer Science Press, 1990.

[8] H-K.T. Ma, S. Devadas, A.R. Newton and A.S. Vincentelli, "Test generation for sequential circuits," *IEEE T-CAD*, pp. 1081-1093, Oct. 1988.

[9] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Comput.*, pp. 215-222, Mar. 1981.

[10] T.M. Nierman, W.T. Cheng and J.H. Patel, "PROOFS: A fast memory efficient sequential circuit fault simulator," *IEEE T-CAD*, pp. 198-207, Feb. 1992.

[11] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. ISCAS.*, May 1989.

[12] R. Lisanke, "Finite-state machine benchmark set," Preliminary benchmark collection, Sept. 1987.