An Efficient Automatic Test Generation System for Path Delay Faults in Combinational Circuits

Ananta K. Majhi and James Jacob
Dept. of Electrical Comm. Eng.
Indian Institute of Science
Bangalore 560 012, India
majhi@ece.iisc.ernet.in, james@ece.iisc.ernet.in

Lalit M. Patnaik
Microprocessor Applications Lab.
Indian Institute of Science
Bangalore 560 012, India
lalit@micro.iisc.ernet.in

Vishwani D. Agrawal
AT&T Bell Laboratories
600 Mountain Avenue
Murray Hill, NJ 07974
va@research.att.com

Abstract

The new test pattern generation system for path delay faults in combinational logic circuits considers robust and nonrobust tests, simultaneously. Once a robust test is obtained for a path with a given transition, another test for the same path with the opposite transition is immediately derived with a small extra effort. To facilitate the simultaneous consideration of robust and nonrobust tests, we derive a new nine-value logic system. An efficient multiple backtrace procedure satisfies test generation objectives. We also use a path selection method which covers all lines in the logic circuit by the longest and the shortest possible paths through them. A fault simulator in the system gives information on robust and nonrobust detection of faults either from a given target set or all path faults. Experimental results on ISCAS'85 and ISCAS'89 benchmark circuits substantiate the efficiency of our algorithm in comparison to other published results.

1 Introduction

The path delay fault model [16] is extensively used since it represents the influence of distributed delays on the operation of clocked systems. The complexity of test generation for delay faults is rather high due to factors such as the large number of possible paths, large number of tests, and excessive computation time for test generation. In this paper, we present a novel test generation algorithm capable of efficiently generating robust and nonrobust tests for path delay faults.

Most of the delay fault testing methods reported hitherto [3, 10] are based on the relatively simple concept of the PODEM algorithm [7]. Fuchs *et al* use an extended multiple backtrace procedure and the concept of static and dynamic learning for path delay fault testing [5]. Non-enumerative methods for test generation and fault simulation of delay faults have recently appeared in the literature [9, 14].

Our method, uses a multiple backtrace procedure similar to that of the FAN algorithm [6] for satisfying the test generation objectives. The new ideas incorporated in our algorithm are as follows:

1. A new 9-value logic system to derive robust and nonrobust tests for path delay faults (Section 2).
2. A multiple backtrace procedure finds and resolves conflicts at internal nodes rather than at PIs. This is illustrated by Example 1.
3. Direct derivation of a test for the opposite transition once a test for a path for a given transition is found (Examples 2 and 3).
4. Path delay fault simulation is done immediately following the generation of a test to determine robust and nonrobust detection status of other faults.
5. Tests are generated for all paths or, especially in case of too many paths, for one longest and one shortest path per line (Section 3).

2 A Nine-Value Logic System

Multi-valued logic has been used by practically all researchers for test generation and fault simulation of path delay faults. In a recent paper, Bose *et al* [2] derive a 23-value logic system for the test generation for multiple paths. Fuchs *et al* [5] introduce a 10-valued logic for robust and a three-valued logic for nonrobust test generation.

Our goal is to derive an optimal logic system that will allow simultaneous consideration of robust and nonrobust tests. In earlier literature [2, 5], multi-valued logic system has been derived by manipulating three (0, 1 and X) logic states which are simultaneously considered in two consecutive time frames (initial, final) of the clock period. In this work, we manipulate the 0, 1 and X logic states for three (initial, intermediate and final) intervals within two clock periods. Thus, we get $3^3 = 27$ possible transition states which are given in Table 1. These 27 states eventually collapse to a 9-value logic system.

The transition states 1-4 and 10-13 of Table 1 in which the logic values are fully specified are represented by unique values, i.e., (S0, G0, FT) and (S1, G1, RT) respectively. The composite states (transition states 5-9 and 14-18) are absorbed together and are represented by X0 and X1 respectively. The last 9 states (19-27) are collapsed to a single logic value XX since the final logic value is X (*don't care*). Signal values G0 and G1 represent static hazards (i.e., glitches

Table 1: Possible logic states

State id	Possible states	Fully specified	Composite states
1	0 0 0	{S0}	-
2	0 1 0	{G0}	-
3	1 0 0	{FT}	-
4	1 1 0	{FT}	-
5	0 X 0	-	{S0,G0}
6	X 0 0	-	{S0,FT}
7	X X 0	-	{S0,G0,FT}
8	1 X 0	-	{FT,FT}
9	X 1 0	-	{G0,FT}
10	1 1 1	{S1}	-
11	1 0 1	{G1}	-
12	0 1 1	{RT}	-
13	0 0 1	{RT}	-
14	0 X 1	-	{RT,RT}
15	1 X 1	-	{S1,G1}
16	X 0 1	-	{RT,G1}
17	X X 1	-	{S1,G1,RT}
18	X 1 1	-	{RT,S1}
19	0 0 X	-	-
20	1 1 X	-	-
21	X 0 X	-	-
22	X 1 X	-	-
23	0 X X	-	-
24	1 X X	-	-
25	0 1 X	-	-
26	1 0 X	-	-
27	X X X	-	-

0-1-0 and 1-0-1) and these are explicitly used for generation of nonrobust tests. More details on generating nonrobust tests will be given in a following section. We believe that our 9-value logic will be the minimum required for the simultaneous consideration of robust and nonrobust tests. In Table 2, we have given the signal value representation and their corresponding logic values in both vectors.

3 Test Generation

In most practical cases, the large number of paths makes it impossible to consider all path faults for the purpose of test generation and fault simulation. There are several methods available in the literature for the selection of a subset of paths [13]. Recently, Pomeranz *et al* [14] and Heragu [9] have described methods without enumerating paths. In recent papers [8, 11], it has been shown that the determination of an optimal clocking period highly depends on the accuracies of the estimated longest path length and the shortest path length in the circuit. In this work, we have introduced a method for selecting a subset of paths which covers all lines in the logic circuit at least once and includes the shortest (S) as well as the longest (L) possible path through each line.

Table 2: Signal value representation

Name of signal value	Initial value	Final value	Hazard conditions
S0	0	0	no static hazard
X0	X	0	hazard possible
FT	1	0	no dynamic hazard
S1	1	1	no static hazard
X1	X	1	hazard possible
RT	0	1	no dynamic hazard
XX	X	X	hazard possible
G0	0	0	static hazard (0-1-0)
G1	1	1	static hazard (1-0-1)

First, we trace the circuit from POs to PIs in a breadth-first manner for assigning a label *depth* to each line L which represents the length (in terms of logic levels) of the longest path from L to any PO. Next, for selecting and enumerating the longest paths through each line, we do a depth-first trace from each PI toward POs along the lines with maximum *depth*. For each intermediate line which is not covered by the longest paths, we continue backward trace toward PIs along the line with maximum *level* and forward trace along the line with maximum *depth*. Similarly, for selecting the shortest paths, we trace the circuit in both directions (i.e., from PIs to POs and vice versa) for assigning two labels $[depth_{pi}, depth_{po}]$ to each line L which represent the minimum length (in terms of logic levels) from L to any PI and PO. We continue the backward and forward trace along the minimum depth lines for selecting the shortest paths.

Our test generation method uses a multiple-backtrace procedure similar to that of the FAN algorithm [6].

First we do an implication with respect to the transition (rising or falling) at the input of the path and other inputs unspecified (XX). The off-path inputs of the gates along the target path are then set to suitable logic values in order to propagate the transition. Next, we backtrace from the primary outputs towards the primary inputs in a breadth-first manner in order to satisfy the objectives. During the backtrace operation, if we create a new objective on an input of a gate G and this input happens to be a fanout branch, then we immediately create another objective with the same logic value on the fanout stem. After doing an implication we mark all fanout branches as *tried.* Hence, before setting an objective on any fanout branch during backtrace, we check whether or not it has already been *tried* by previous objectives. The procedure for test generation is illustrated by the following example.

Example 1: Consider the circuit in Figure 1. A test is to be generated for the rising transition on path 4-5-7-8-10-12. During implication the fanout branches 5, 6, 8 and 9 will be marked as *assigned.* The primary objectives obtained are (2=S1, 3=X1 and 11=X1). During backtrace the last level objective (11=X1) will not create any new objective since both fanins (6 and 9) are already marked as *assigned.* The other two objectives are fanout branches of stem 1. Hence, after intersection of the two logic values ($S1 \cap X1 = S1$), S1 will be assigned to the stem 1, which is a PI. Thus, after simulation we find a robust test (1=S1, 4=RT), since all primary objectives are satisfied. ■

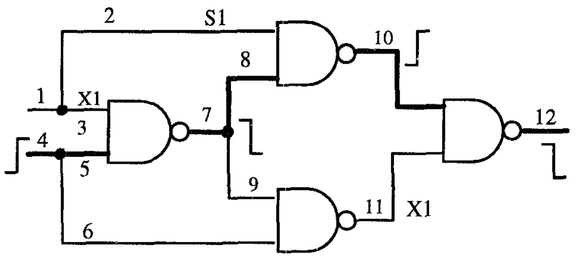

Figure 1: Example of multiple backtrace

Once we find a robust test for a transition on a target path, instead of generating a completely new test for the opposite transition we derive the second test by replacing X1(X0) with steady values S1(S0) in the already generated test and reversing the transition at the input of the target path. The underlying principle is that since the second vector is a test for a stuck fault on the PI at the source of the path, the opposite transition will always reach the destination of the path if the sensitizing condition remains unchanged. In most cases this modified vector pair will be a valid robust/nonrobust test for the opposite transition. If the implied logic values are a subset ($S0$ *and* $G0 \subset X0$; $S1$ *and* $G1 \subset X1$) of the primary objective logic values or are the same, then the derived test will be a robust test. The following example illustrates such derivation.

Example 2: Consider the circuit given in Figure 2, where a test is generated for the rising transition on path 1-2-10-12. The primary objectives to be justified are 8=X1 and 11=S1. Following Example 1, we obtain a robust test, i.e., 1=RT and 4=S0. The new set of primary objectives for the opposite transition on the same path (1-2-10-12) will be 8=S1 and 11=X1. The new test is obtained only by reversing the transition on line 1 (1=FT and 4=S0) since input 4 has a steady value S0. After implication, we find that the implied logic value on line 11 is S1 which is a subset of X1 (i.e., primary objective logic value on line 11). Hence, we find a robust test for the opposite transition with minimal effort. ■

In some cases, the test derived for the opposite transition as described above may not be a robust test. If at least one of the primary objective logic values is S1(S0) and the implied logic value becomes G1(G0), then the derived test will be a nonrobust test for the opposite transition. The following example illustrates the derivation of a nonrobust test.

Example 3: In Example 1, suppose we have to find a test for the opposite transition (i.e., falling) on the same path (4-5-7-8-10-12). The new set of primary objectives will be 2=X1, 3=S1 and 11=S1. The test

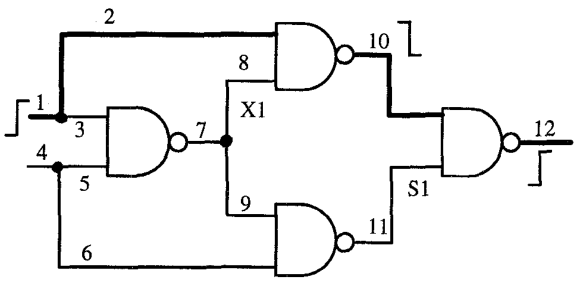

Figure 2: Robust test for opposite transition

will be obtained by reversing the transition on line 4 (1=S1, 4=FT) since input 1 has a steady value S1. After implication, we find that the implied logic value of line 11 is G1, whereas its primary objective logic value is S1. Hence, the derived test will be a nonrobust test for the falling transition. In such situations, a fresh attempt will be made to generate a robust test for the given path and transition. ■

4 Results

We have implemented the proposed path delay test generation algorithm in the C language (about 3000 lines of code) on an IBM RS-6000/580 workstation. In order to benchmark and demonstrate the efficiency of our test generator, we have performed ATG of robust and nonrobust tests on the ISCAS'85 and ISCAS'89 benchmark circuits.

Table 3 gives the results for ISCAS'85 benchmarks. *Path faults* is the number of logical paths considered for test generation. This is twice the number of the physical paths selected to cover each line via the longest (L) path through it. For the purpose of comparison, we have also given the test generation results when the path selection procedure is modified to choose the shortest (S) path through each line. As expected, in general, the coverage is higher for the shorter paths. We have integrated a fault simulator [12] in the system. Once a robust or nonrobust test is generated, immediately we do fault simulation with respect to the test vector. Both robust and nonrobust detection of path delay faults are reported. Robustly detected paths are then marked in the targeted path list and hence not considered for further test generation. Third and forth columns of Table 3 give the number of robust and nonrobust tests generated by the test generator within the backtrack limit of 100. Fifth and sixth columns give the number of path faults which are robustly and nonrobustly tested from the targeted fault list while columns seven and eight are for the non-targeted faults. The CPU time (in seconds) is given for the complete ATPG process. For circuit c6288, the number of nonrobustly detected paths was extremely large even for small number of tests and it requires large amount of CPU time and memory for successful completion. Hence, we have not considered the nonrobust detection of paths during simulation.

Our results in Table 3 may be compared to two earlier works [14, 15] which give test generation re-

Table 3: Delay test results for ISCAS'85 benchmarks

Circuit	Path	Tests		Paths detected		Paths detected		
		generated		(Targeted)		(Non-targeted)		
	faults	Rob.	Nrob.	Rob.	Nrob.	Rob.	Nrob.	CPU s*
c880	744(L)	469	0	520	5	244	711	15.45
	744(S)	620	3	712	8	41	439	9.08
c1355	1100(L)	0	0	0	0	0	0	39.31
	1100(S)	16	0	26	0	28	34	85.12
c1908	1286(L)	57	1	83	13	24	902	177.93
	1286(S)	319	0	366	61	37	1221	114.37
c2670	2046(L)	572	9	589	59	498	1897	249.31
	2046(S)	944	4	1031	61	218	4598	139.20
c3540	2584(L)	68	0	87	2	113	2436	579.11
	2584(S)	541	0	683	88	1017	7999	554.83
c5315	4404(L)	2242	26	2245	433	2220	26138	808.30
	4404(S)	2183	4	2889	198	787	17275	616.98
c6288	4832(L)	27	0	34	NA	8	NA	1198.57
	4832(S)	195	52	213	NA	102	NA	2401.01
c7552	5480(L)	846	64	953	578	1968	28370	2083.40
	5480(S)	2095	35	2502	330	5023	25345	2257.43

* IBM RS-6000/580

Table 4: Test results for scan/hold versions of ISCAS'89 benchmarks

Circuit	Path	Our ATPG				[1]		[3]		[4]	
		Rob./nrob.	Rob.	Nrob.	CPU	Paths	CPU	Paths	CPU	Paths	CPU
	faults	tests	det.	det.	s†	det.	s**	det.	s‡	det.	s‡
s27	56	28/0	50	0	0.03	50	0.0	50	0.1	50	0.0
s208	290	232/0	288	0	0.49	290	0.2	290	2.7	290	4.9
s298	462	221/1	328	22	1.24	343	0.2	331	4.9	343	5.1
s344	710	363/4	559	45	2.07	611	0.6	602	8.6	611	23.4
s349	730	361/4	554	49	2.02	611	0.6	602	9.0	611	14.6
s382	800	349/13	660	65	2.02	667	0.5	664	14.1	667	21.6
s386	414	210/0	406	2	0.71	413	0.3	-	-	-	-
s400	896	348/5	646	59	2.59	665	0.6	-	-	-	-
s420	738	612/0	736	0	2.04	738	1.3	738	13.8	738	34.7
s444	1070	376/60	579	190	3.69	586	0.8	513	31.7	586	13.9
s510	738	391/2	723	14	2.20	729	0.9	703	11.9	729	50.2
s526	820	419/2	651	43	3.03	694	0.5	-	-	694	41.8
s526n	816	420/2	652	40	3.52	695	0.5	-	-	695	38.4
s641	3488	1302/49	1955	168	28.58	1979	8.5	-	-	1979	275.6
s820	984	554/0	964	7	5.62	980	1.1	961	18.3	980	122.0
s832	1012	546/0	966	11	6.70	984	1.3	971	19.8	984	125.5
s838	2018	1756/0	2016	0	13.19	2018	9.7	-	-	2018	496.4
s953	2266	985/0	2190	32	14.61	2256	4.1	2221	44.2	2302	256.6
s1196	6194	1748/7	3408	119	252	3579	13.9	2938	282.0	3581	1146
s1238	7118	1656/3	3450	42	275	3587	14.0	-	-	3589	1253
s1488	1924	805/1	1830	35	20.79	1875	3.1	1830	72.3	1875	315.5
s1494	1952	801/1	1837	37	21.84	1882	3.1	1838	75.7	1882	304.5
s5378	27062	8341/83	14421	1662	3236	18656	98.0	16222	26600	6554	8450
s713	480*	146/4	183	1	9.70	1184	22.0	-	-	232	50.0
s1423	1172*	677/0	893	19	38.1	28696	436.3	-	-	2730	1300
s9234	5284*	1913/161	2549	403	1241	21389	481.3	-	-	231	1050
s13207	8008*	3920/219	5665	760	1581	27503	2649	-	-	133	2740
s15850	9114*	3793/257	5146	763	2957	-	-	-	-	6	2190
s35932	28504*	11149/1787	15033	4383	22294	21783	308.8	-	-	335	1270
s38417	23182*	12479/300	16532	1196	19531	-	-	-	-	242	4120
s38584	30466*	18631/265	24268	1146	28642	92235	2614	-	-	12	950

* Partial set of faults † IBM RS-6000/580 ** DECstation 2000/500 ‡ SUN SPARC2

sults for ISCAS'85 benchmarks. However, since the number of modeled path faults and the machines used are different, a direct comparison may not be meaningful. It may be noticed that the fraction of modeled paths that are robustly tested in our method is significantly higher than that reported in [14, 15]. The results support the merit of selecting the longest and shortest paths and the efficiency of the novel ideas in our system.

Table 4 presents the results of our algorithm for scan/hold versions of ISCAS'89 benchmark circuits. *Path faults* is the number of both rising and falling transitions on all possible physical paths. Under *Our ATPG, Rob./nrob. tests* is the number of robust and nonrobust tests generated within the backtrack limit of 100. *Rob. det.* is the number of path faults which are robustly tested. *Nrob. det.* is the number of path faults that are nonrobustly tested by the fault simulator. The CPU time (in seconds), given for IBM RS-6000/580 workstation, includes the time for test generation and fault simulation. For the last eight benchmarks shown in Table 4, we have considered only a subset of all possible path faults since the total number of paths in these circuits was very large. These paths were obtained by the path selection algorithm given in Section 3 and include one shortest path per line.

For comparison, we include the results of [1, 3, 4] in Table 4 which reports the coverage of robustly tested paths and the CPU times for these three approaches. All possible path faults are modeled in [1, 3]. Chakradhar *et al* [4] consider 10,000 path faults for the last nine benchmarks of Table 4 and all possible path faults for other benchmarks. For a direct comparison, CPU times must be normalized for the corresponding machine speeds. Such normalization is not done in Table 4. However, our ATPG appears to be more efficient than the techniques of [3, 4] and the fault coverage is comparable. The CPU times reported in [1] are better than our results. Such performance is generally expected from the binary decision diagram (BDD) method. It must be remembered, however, that the time and memory complexities of the BDD approach [1] can be impractical for some circuits.

5 Conclusion

We have presented a novel path delay test generation algorithm which incorporates an efficient multiple backtrace procedure for signal value justification. The nine-value logic system provides an efficient way of deriving both robust and nonrobust tests. Once we find a robust test for a path delay fault, we modify it to derive another test for the opposite transition. In most cases, the derived test becomes either a robust or a nonrobust test for the same path with opposite transition. Thus, we are able to considerably reduce the test generation time. A subset of paths is selected for test generation covering all lines in the logic circuit at least once. This subset includes the longest and shortest paths through each line. We use a fault simulator for robust and nonrobust detection of path faults.

We are currently extending this approach for multiple pass test generation through longest sensitizable paths to cover all lines.

References

[1] D. Bhattacharya, P. Agrawal, and V. D. Agrawal, "Delay Fault Test Generation for Scan/Hold Circuits using Boolean Expressions", *Proc. Design Autom. Conf.*, pp. 159-164, June 1992.

[2] S. Bose, P. Agrawal, and V. D. Agrawal, "Logic Systems for Path Delay Fault Test Generation", *Proc. EURO-DAC*, pp. 200-205, September 1993.

[3] S. Bose, P. Agrawal, and V. D. Agrawal, "Generation of Compact Delay Tests by Multiple Path Activation", *Proc. Int'l Test Conf.*, pp. 714-723, October 1993.

[4] S. T. Chakradhar, M. A. Iyer, and V. D. Agrawal, "Energy Minimization Based Delay Testing", *Proc. European Conf. Design Autom. (EDAC)*, pp. 280-284, March 1992.

[5] K. Fuchs, F. Fink, and M. H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults", *IEEE Trans. CAD*, Vol. 10, pp. 1323-1334, October 1991.

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

[7] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits", *IEEE Trans. Comput.*, Vol. C-30, pp. 215-222, March 1981.

[8] C. T. Gray, W. Liu, and R. K. Cavin, "Timing Constraints for Wave-Pipelined Systems", *IEEE Trans. CAD*, Vol. 13, pp. 987-1004, August 1994.

[9] K. Heragu, "Approximate and Statistical Methods to Compute Delay Fault Coverage", *Master's Thesis*, Dept. of ECE, Rutgers University, New Brunswick, New Jersey, 1994.

[10] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits", *IEEE Trans. CAD*, Vol. CAD-6, pp. 694-703, September 1987.

[11] L.-R. Liu, H.-C. Chen, and D. H. C. Du, "The Calculation of Signal Stable Ranges in Combinational Circuits", *IEEE Trans. CAD*, Vol. 13, pp. 1016-1023, August 1994.

[12] A. K. Majhi, J. Jacob, and L. M. Patnaik, "A Novel Path Delay Fault Simulator using Binary Logic", to appear in *VLSI Design: An Int'l Jour. Custom-Chip Design, Simulation and Testing*, 1994.

[13] S. Patil and S. M. Reddy, "A Test Generation System for Path Delay Faults", *Proc. Int'l Conf. Computer Des.*, pp. 40-43, 1989.

[14] I. Pomeranz, S. M. Reddy, and P. Uppaluri, "NEST: A Non-Enumerative Test Generation Method for Path Delay Faults in Combinational Circuits", *Proc. Design Autom. Conf.*, pp. 439-445, June 1993.

[15] M. H. Schulz, K. Fuchs, and F. Fink, "Advanced Automatic Test Pattern Techniques for Path Delay Faults", *Proc. Fault Tolerant Comp. Symp.*, pp. 44-51, 1989.

[16] G. L. Smith, "Model for Path Delay Faults Based on Paths", *Proc. Int'l Test Conf*, pp. 342-349, 1985.