TranGen: A SAT-Based ATPG for Path-Oriented Transition Faults

Kai Yang, Kwang-Ting Cheng, and Li-C. Wang
Department of Electrical and Computer Engineering
University of California, Santa Barbara

Abstract— This paper presents a SAT-based ATPG tool targeting on a path-oriented transition fault model. Under this fault model, a transition fault is detected through the longest sensitizable path. In the ATPG process, we utilize an efficient false-path pruning technique to identify the longest sensitizable path through each fault site. We demonstrate that our new SAT-based ATPG can be orders-of-magnitude faster than a commercial ATPG tool. To demonstrate the quality of the tests generated by our approach, we compare its resulting test set to three other test sets: a single-detection transition fault test set, a multiple-detection transition fault test set, and a traditional critical path test set added to the single-detection set. The superiority of our approach is demonstrated through various experiments based on statistical delay simulation and defect injection using benchmark circuits.

I. Introduction

In traditional delay testing, two types of fault models are usually considered: the gate delay model [1] [2][3][4] and the path delay model [5][6][7]. In a gate delay model such as the single transition fault model [1][8][9], the size of a delay fault is not considered crucial. Whenever a fault occurs, it is assumed that the fault effect is large enough to cause a timing failure. Because of this assumption, it is generally believed that tests results based on a gate delay model are only useful for capturing large-size delay defects. To capture small-size delay defects, a path delay fault model is often used. However, since there can be an exponential number of sensitizable paths in a circuit, explicitly testing all paths is not practical. Hence, in practice, only a set of timing *critical paths* is selected for explicit testing. In practice, the transition fault model and critical path delay model are often combined in order to capture both small-size and large-size delay defects.

The gate delay model and the path delay model represent two orthogonal approaches to detect timing defects. It is natural to develop a hybrid fault model that can combine the merits of these two orthogonal approaches [10]. Intuitively, the advantage of using a gate delay model would be its complete topological coverage. The advantage of using a path delay model is that tests can focus specifically on the worst-case timing scenarios. A hybrid model should have both features. Moreover, it should have the following two additional features in order to justify its value: (1) The ATPG effort based on the hybrid model should be comparable to the total ATPG effort based on both a gate delay model and a path delay model. (2) The test quality based on the hybrid model should be better than the overall quality based on the gate delay model and the path delay model combined.

In this work, we utilize a fault model that combines the merits of both the transition fault model and critical path delay model. We this our model a *path-oriented transition fault* model. In this fault model, we assume that a transition fault should always be detected through the longest sensitizable (propagation) path (a propagation path starts from the fault site to one of the primary outputs). The idea of incorporating the constraints of longest paths into the detection of transition faults is not new [9][15]. However, we differentiate our work from previous work in the following two ways. First, our approach for test generation is based on an efficient circuit Boolean Satisfiability (SAT) solver that has been shown effective for verification applications [17, 18]. With a new *false-path pruning* technique, we demonstrate that the SAT solver can be used as an efficient test generator for the new fault model. Second, we demonstrate that the new fault model can lead to a test set whose quality is better than that achieved by combining both the transition fault model and the critical path delay model. In our work, we measure the test quality using statistical delay simulation and defect injection [23, 24, 25].

Because our fault model is path-oriented, fortuitous detection among faults can be low. In other words, a test pattern produced for a given fault may not fortuitously detect any other transition fault through its longest sensitizable path. As a result, more ATPG effort is required to produce tests for all faults in this model than that given by the single transition fault model. Usually, the fortuitous detection among transition faults is high. With the transition fault model, many faults can be removed by fault simulation. We note that this is a limitation in any path-oriented fault model.

The rest of the paper is organized as follows. In Section 2, we review some previous research results and introduce the circuit SAT solver used in our ATPG. In Section 3, we propose the test generation algorithm for the proposed path-oriented transition fault model. In Section 4, a statistical timing defect injection and simulation framework is described. We also describe other metrics for quality evaluation of delay test set, such as using the worst-case delays (slacks). Section 5 presents the experimental results for ISCAS 85/89 benchmarks. Conclusions are given in Section 6.

II. BACKGROUND

In order to have a fault model that can provide a complete topological coverage and, simultaneously, exercise the worst-case timing scenarios, researchers have proposed methods to select critical paths that cover all transition fault sites [10][11][12] [13][14][15]. A common issue among these methods is that many of the selected critical paths may not be sensitizable. To resolve this, one possible solution is to

CKT	CSAT-Based ATPG			A Commercial ATPG			
	CPU Time (sec)	Coverage	Average ATPG time per path (ms)	CPU Time (sec)	Coverage	Average ATPG time per path (ms)	ATPG time speed up (X)
c5315	33.3	98.7%	1.7	1463.9	74.1%	227	136.6
c7552	61.1	96.8%	3.1	462.5	80.8%	213	69.6
s9234	28.3	99.7%	1.4	407.6	99.7%	89	63.3
s13207	47.7	100.0%	2.4	418.7	100.0%	134	36.1
s15850	47.8	100.0%	2.4	363.7	99.3%	255	106.7
s35932	184.1	100.0%	9.2	140.1	100.0%	101	10.9
s38417	115.3	97.1%	5.8	1415.1	97.1%	166	28.9
s38584	226.6	100.0%	11.4	312.9	100.0%	132	11.6

TABLE I: COMPARISON RESULTS ON PATH DELAY ATPG (20000 CRITICAL/TRUE PATHS ARE SELECTED FOR ATPG[19])

pre-select a number of long paths passing through each site and then, following the order of their timing lengths, to check one by one until the first sensitizable path can be identified [11][12][13]. This *path enumeration* approach can be inefficient, depending on the number of paths that have been checked before a sensitizable path is identified. Recently, new techniques [14] [15] were developed to identify the sensitizable critical paths without using the path enumeration approach.

Selecting critical paths to cover all sites is different from detecting each transition fault through the longest sensitizable (propagation) path. In the former case, all paths are ranked together based on their timing lengths and hence, there is only one ordering among all paths. In the latter case, all paths from a transition fault site are ranked together. Each site has its own ranking of the paths. An ATPG called *POTENT* [9] was first proposed for the path-oriented transition fault model. This ATPG was based on a *greedy path expansion* heuristic. A rather simplified metric was also proposed to evaluate the quality of the resulting tests. With their fixed delay model, the *worst-case slacks* were characterized by simulation as the measurement for quality.

In our ATPG, the timing length of a path is assumed to be a fixed value calculated by summing up the delays of all circuit elements along the path. To efficiently perform ATPG justification, a SAT-based ATPG [17, 18] solver is employed. Our SAT-based ATPG solver, named CSAT, operates on the circuit net-list rather than on the Conjunctive Normal Form (CNF) of the circuit. Our experiments show that the ATPG solver is very efficient for generating tests for path-oriented transition faults.

Unlike the detection of a stuck-at or transition fault, the detection of a path-oriented fault requires no *fault propagation*. It requires only justification of logic values. With fault propagation, a traditional ATPG would need to use a 5-value logic system (including D and $\overline{D}$) to propagate the faulty effects. Without fault propagation requirement, the SAT-based ATPG solver which uses two-valued logic, can be applied directly.

Once a path is selected for a given fault site, determining if the path is sensitizable is a highly-constrained search problem. This is because to sensitize the path, initially many signals would have been enforced with constant values. This restricts the search space dramatically as compared to that of ATPG for stuck-at or transition fault. Because the search space is highly-constrained, a SAT-based approach would be a more appropriate choice than a traditional ATPG process. The former utilizes *conflict clauses* to eliminate the impossible search sub-spaces that contain no solution. The latter often uses heuristics to guide the search process into a sub-space that is more likely to have a solution.

To illustrate the points discussed above, Table I presents comparison results between our SAT-based ATPG and a commercial ATPG for path delay faults. In this experiment, we use a path classification tool [19] to identify 20000 longest true paths for testing. The primary fault models used in the commercial ATPG are stuck-at and transition fault models. In the Table, we compare the two ATPGs based on their efficiency for path sensitization. As explained above, our ATPG would be more efficient in these cases. As the results show, our ATPG is 10X to 150X faster than the commercial ATPG (in comparing the average run-time per path). We note that for the stuck-at or transition faults, our SAT-based ATPG would not perform as well as the commercial ATPG.

III. DETECT TRANSITION FAULT THROUGH THE LONGEST PATH

In this section, we present an algorithm to detect each transition fault through the longest sensitizable propagation path. We use an efficient false-path pruning technique to reduce the search space for identifying the longest sensitizable path.

III-A. Test Generation

Given a transition fault site, the longest propagation path is first selected. To sensitize the path, a stepwise path-tracing technique is employed. If a conflict occurs during the sensitization process, an un-sensitizable segment of the path will be identified. This un-sensitizable segment is recorded so that any paths covering the segment will not be selected in the future steps. The recording of un-sensitizable segments is done by manipulating the *circuit graph*, which will be explained below in Section III-B. If the longest path is not sensitizable, the next longest path will be selected for test generation. The process repeats until a sensitizable path can be found. If all paths from the fault site are not sensitizable, then the ATPG will simply try to produce a pattern to detect the transition fault without enforcing the propagation through any particular path.

Figure 1 shows the detail of the ATPG flow. After one fault is selected from the transition fault list, the first step is to produce the 1st test vector in the 2-vector test pattern. If no conflict occurs during this process, a linear time algorithm [16] will be applied to identify the longest propagation path from the fault site to any primary output. Once the longest propagation path is selected, the ATPG will try to static-sensitize the path.

If the selected path is static-sensitizable, the next fault will be selected for pattern generation. If a conflict occurs during

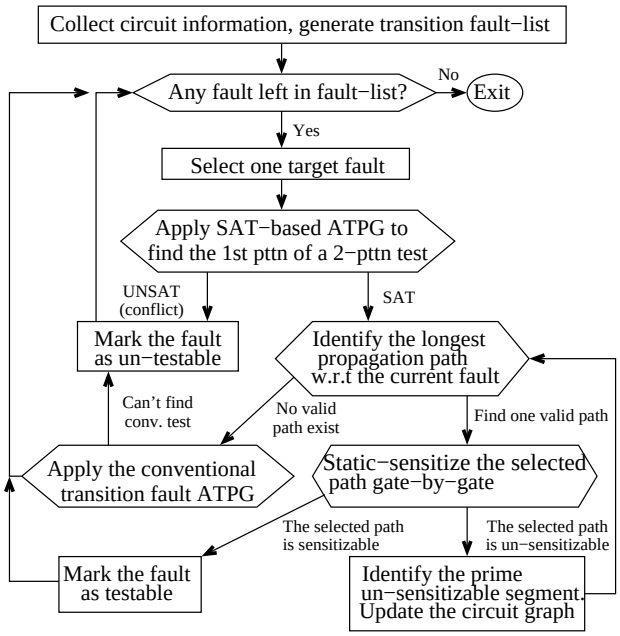

Fig. 1. ATPG Flow

the process of sensitization, a routine (explained below in Section III-B) will be used to identify and recording the *prime* un-sensitizable segments. The process repeats until a sensitizable path is found or proven that no sensitizable path exists.

If no sensitizable path exists, this fault might be detected with multiple propagation paths [9]. A conventional transition fault ATPG will be applied in this case.

III-B. False-Path Pruning Technique

Past research [19][20][21] has shown that there can be many un-sensitizable segments in a circuit. Any path covering an un-sensitizable segment will be un-sensitizable. To avoid selecting those paths, we maintain a *circuit graph* during the ATPG process. This circuit graph is updated each time an un-sensitizable segment is found. We utilize the stepwise path-tracing technique [19][22] to identify the prime un-sensitizable segments. We define the *unique operation* to update the circuit graph based on each given un-sensitizable segment.

The process of identifying an un-sensitizable segment consists of two phases [19]. In the first phase, we find a partial path $Q = (I_n, G_1, \ldots, G_t)$ where I_n is the transition fault site and each G_i $(1 \leq i \leq t)$ is a gate on the path. Q is an un-sensitizable path identified by adding the sensitization constraints associated with the gate G_t. Then, from gate G_t, we move backward on the path and identify the first gate G_j such that the path $S = (G_j, G_{j+1}, \ldots, G_t)$ is un-sensitizable. We designate S as the *prime un-sensitizable segment*.

Once a prime un-sensitizable segment is identified, we alter the circuit graph based on the *unique operation* explained below. In Figure 2 (a), the current fault is located at the output of node *0*, and the segment {*1-2-3-4*} is an un-sensitizable segment. Therefore, any path passing through segment {*1-2-3-4*} will be un-sensitizable. To avoid selecting those paths in future ATPG steps, the segment need to be removed from the graph. However, we cannot simply remove all the nodes on the segment because this removal of the segment will result in the deletion of other potentially sensitizable paths covering

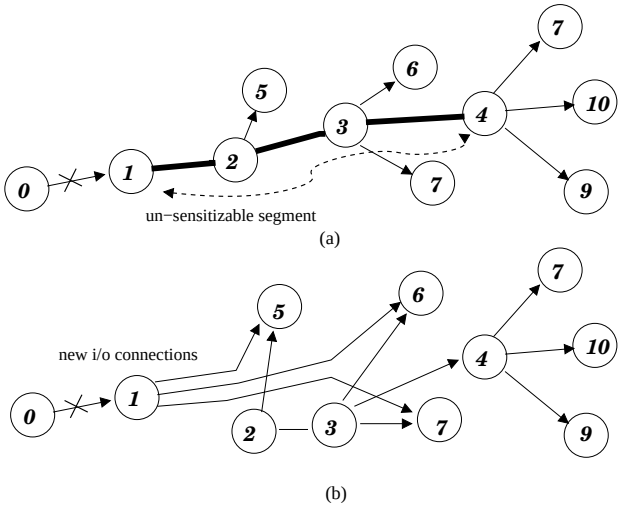

Fig. 2. Unique Operation Example: (a) Original Graph (b) Updated Graph

part but not all of the segment. For example, segment {*0-1-2-5*} can be a sensitizable path. In order to avoid the selection of paths covering the entire segment, we need duplicate the connections as illustrated in Figure 2 (b) so that all other paths not passing through the entire segment {*1-2-3-4*} can still be selected. Figure 2 (b) shows the resulting graph after the update. The duplication process not only updates the i/o connections but also re-calculates the delay value between any two nodes involving in the change. The *unique operation* on the circuit graph can be performed in linear time (*O(n)*) complexity.

Un-sensitizable segments can be classified into two categories according to their relationship with the transition-faulty value:

Transition-Independent Un-sensitizable Segment: The un-sensitizable segment can be identified without considering the corresponding faulty value. The side-inputs along this segment cannot be assigned to non-controlling values simultaneously. Therefore, any path that covers this segment is un-sensitizable.

Transition-Dependent Un-sensitizable Segment: The segment becomes un-sensitizable because of the corresponding faulty value. No pattern can set all side-inputs along this segment to non-controlling values and simultaneously initialize the faulty value. If this is determined to be true, the ATPG will restore the circuit graph.

Theoretically, the size of the circuit graph may grow exponentially. However, as it will be seen in the experimental section, for most testcases in our experiments, circuit graphs do not grow very large (usually from 1x to 26x). In general, the larger growth rate is always due to the huge path count. For example, c3540 and s15850 have more than $5 * 10^7$ paths. One exception that we can not handle is c6288 which has more than 10^{20} paths. For this kind of circuit, the proposed technique may not be suitable. More powerful methodologies need to be developed which will be one of our future works.

IV. EVALUATION OF THE TEST QUALITY

To demonstrate that the new fault model can indeed lead to test sets that are with better quality than that by traditional

models, we use several quality metrics. One of these metrics is based on the circuit delay/slack achieved by each pattern. We also calculate the worst-case delay among all patterns as well as the average delay over all patterns.

IV-A. Delay/Slack-Based Quality Evaluation

Intuitively, if a pattern can achieve a large delay (or equivalently, a small slack), this pattern has a better chance to capture a delay defect (if the defect falls on the sensitized path resulting in the large delay). Given a pattern, we utilize a statistical dynamic simulation tool to calculate the resulting circuit delay as a distribution [23, 24, 25]. Because the result is a distribution, there can be many ways to define the concept of a delay (and a slack) if we are allowed to compare only a single value. In our experiments, we use the mean of the distribution as the delay value. Given a clock period, the difference between the clock and the mean value will be denoted as the slack. This is illustrated in Figure 3 below.

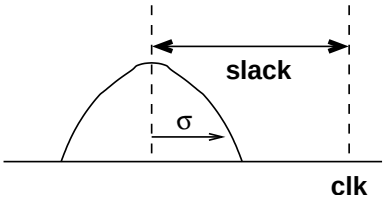

Fig. 3. Delay/Slack Value

IV-B. Statistical Timing Defect Injection and Simulation

To perform more realistic test quality evaluation, we utilize a statistical timing defect injection and simulation tool to estimate the test coverage of injected delay defects [23, 24, 25]. In the statistical timing analysis (STA) framework , the delays of cells/interconnects are modeled as correlated random variables with supposedly known probability density functions (pdf's). In this work, the analysis is cell-based [23]. Different types of delay defects with specified locations and sizes can be injected into a circuit for simulation.

IV-B.1 Defect Injection and Simulation

To simulate the manufacturing process of N chip instances, we use the STA framework to produce N sampled instances, each with a different delay configuration. On each instance, we inject a random-location, random-size delay defect. The size of the delay defect is drawn from an exponential distribution [26]. Applying statistical timing simulation with a set of patterns on these N instances will tell how many of them exceed a given clock period *clk*. We use this number divided by N as the *defect test coverage*.

V. EXPERIMENTAL RESULTS

We implemented the proposed ATPG algorithm in C++ on a Linux workstation. The Circuit-SAT solver [17, 18] was utilized as our fundamental engine to perform value justification. We experimented with the proposed ATPG on ISCAS85 and some of the large ISCAS89 circuits (combinational part only). For the fixed delay value of each type of gate, we used a 0.25μ delay library generated by SPICE. Table II shows the statistics for the ATPG.

In Table II, Columns 1 and 2 show the circuit name and the number of collapsed transition faults. Column 3 shows the total CPU time in terms of seconds while column 4 shows the average CPU time for each fault. As can be seen, the CPU time for individual faults is small. Column 5 represents the test coverage and the fault coverage. Within a reasonable CPU time, high coverage can be achieved. The average number of tried propagation paths for each ATPG procedure is listed in column 6. With the false-path pruning methodology, these numbers are small. Column 7 shows the number of identified prime un-sensitizable segments in each case. Column 8 lists the maximum graph size resulting from the unique operation for segment removal. As it can be seen, the circuit graphs do not grow too large.

In order to demonstrate the test quality among different test models, five different pattern sets are applied: **(1)** random patterns, **(2)** single-detection transition fault, **(3)** 4-detection transition fault, **(4)** critical path delay fault where the number of critical paths selected is equal to the number of faults listed in Table II. and **(5)** transition fault detected through the longest path, named as *TF-Long*.

Table III shows the percentage of topological coverage by the critical path set selected for each circuit. For example, in the case of c880, only 16.73% of all transition fault sites are covered by the critical path set.

CKT	c880	c1355	c1908	c2670	c7552
%	16.73%	20.26%	11.24%	9.62%	30.05%
CKT	s9234	s13207	s35932	s38417	38584
%	18.70%	9.68%	69.23%	10.84%	2.13%

TABLE III: TOPOLOGICAL COVERAGE BY CRITICAL PATH SET

To demonstrate the distribution of derived delay for each pattern, Figure 4 shows the spectrum of the delay value for circuit s13207. The x-axis shows the delay value range and the y-axis shows the percentage of patterns whose resulting delays falling into each corresponding delay value range. As can be seen, critical path patterns always result in the highest percentage of long-delay patterns. However, the TF-Long model achieve higher percentage than other models. Figure 5 shows a similar plot. Results from other circuits show similar behavior in this experiment.

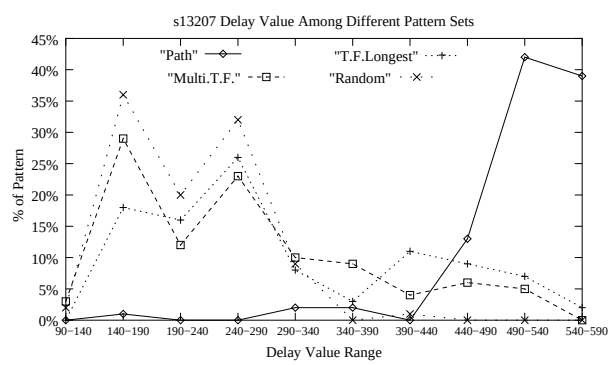

Fig. 4. Delay Spectrum for s13207

CKT	# of Fault	CPU Time	Avg Time/ATPG	TC (FC)	Avg-Tried	# of Un-Sen-Seg	Growth Rate of Graph
c880	1582	2.39	0.001511	100% (100%)	1.00	0	1x
c1355	2566	21.36	0.02004	100% (91.69%)	3.377	1165	8.56x
c1908	2938	100.2	0.03272	100% (92.17%)	16.12	942	6.73x
c2670	4154	25.72	0.006192	100% (93.05%)	1.006	211	1.36x
c3540	5654	660.1	0.1167	100% (87.32%)	5.42	3927	15.91x
c5315	8842	43.44	0.004864	100% (91.02%)	1.056	690	1.59x
c7552	12284	386.5	0.03175	100% (95.88%)	1.073	1125	1.89x
s9238	11328	90.48	0.00805	100% (92.16%)	1.035	708	1.29x
s13207	15602	225.8	0.01446	100% (98.72%)	1.039	1097	1.33x
s15850	19046	4635	0.2434	100% (90.89%)	5.699	83811	26.22x
s35932	63502	1383	0.02178	100% (88.89%)	1.092	4067	1.78x
s38584	61254	3127	0.05104	100% (91.55%)	1.021	915	1.22x
s38417	49738	2267	0.04558	100% (89.63%)	1.03	1649	1.21x

TABLE II: ATPG STATISTICS

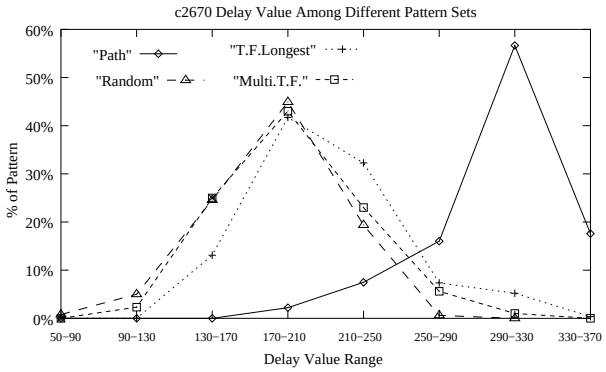

Fig. 5. Delay Spectrum for c2670

Table IV shows comparison results based on the average delay values. In each case, the average delay value (over all patterns) is shown on the top, and below it standard deviation associated with the average is shown. In general, higher average delay can be derived by the path-oriented transition fault model than others.

CKT	Random	T.F	4-Detection T.F.	Critical Path	Longest Path T.F
c880	107.25	112.94	115.06	207.15	126.86
	(26.7)	(27.7)	(27.3)	(14.1)	(43.2)
c1355	96.33	147.32	158.65	181.16	176.22
	(87.5)	(70.3)	(66.4)	(74.4)	(54.1)
c2670	188.88	191.30	201.92	298.47	199.46
	(37.2)	(37.4)	(40.2)	(34.8)	(49.9)
s9234	193.68	227.33	229.92	294.81	231.71
	(45.3)	(56.9)	(52.1)	(30.9)	(60.1)
s35932	190.82	219.82	228.82	253.03	229.19
	(73.5)	(81.7)	(72.2)	(55.6)	(81.1)
s38417	304.92	310.63	309.31	365.89	310.86
	(37.9)	(37.6)	(38.1)	(48.2)	(79.0)
s38584	335.82	352.43	357.21	587.55	356.23
	(89.6)	(104.0)	(99.3)	(67.3)	(109.4)

TABLE IV: AVERAGE DELAY PER PATTERN

Table V shows the worst-case delay values. The path-oriented transition fault model (last column) usually achieves larger worst-case delays than results from other models, and achieves comparable worst-case delays to the results from the critical path model.

To achieve more realistic comparison for test quality, we apply statistical defect injection and simulation. In these experiments, we compare the test quality of the conventional delay test patterns to the path-oriented transition fault patterns. In order to make a fair comparison, we normalized the total number of patterns to be the same. In other words, for each conventional test set, we use a 2-detection transition test set plus a critical path delay test set where the total number of test patterns is the same as the number of the path-oriented transition fault patterns.

CKT	Random	T.F	4-Detection T.F.	Critical Path	Longest Path T.F
c880	195.01	186.58	207.83	239.24	239.16
c1355	225.56	222.16	231.92	234.25	233.63
c2670	299.03	320.47	325.13	362.81	359.23
s9234	386.56	404.42	412.12	416.70	416.27
s13207	473.05	590.20	592.39	593.74	594.96
s35932	363.95	363.84	364.21	364.81	364.48
s38417	401.43	420.89	421.32	425.21	424.31
s38584	588.57	641.53	642.10	653.37	645.37

TABLE V: WORST-CASE DELAY

Figures 6 and 7 show the defect test coverage with respect to the corresponding defect size. The x-axis shows the defect test coverage and the y-axis shows the size of defect that we inject into the circuit. In these experiments, both pattern sets can achieve high defect test coverage when the defect sizes are large. As the defect sizes become smaller, the test coverages decrease for both sets. The experimental data show that the test coverages from our pattern sets exceed those from the conventional delay test sets (when the defect sizes are small).

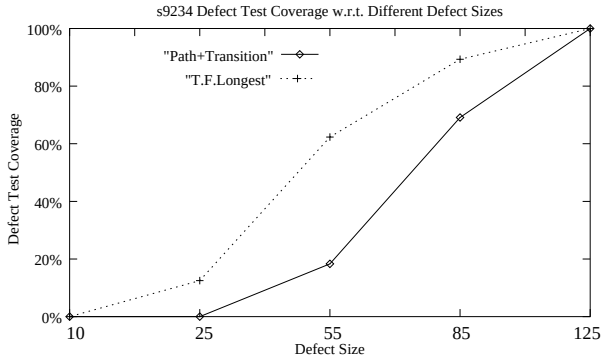

Fig. 6. Delay Test Coverage for s9234

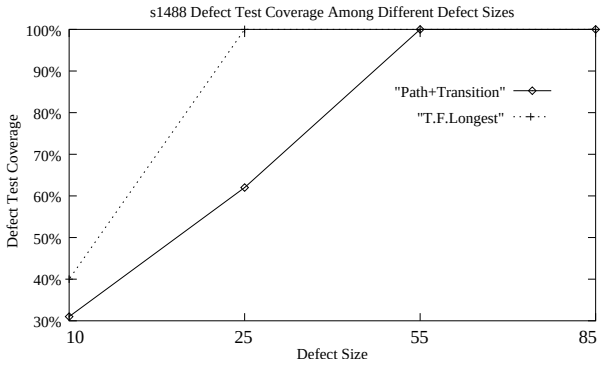

Fig. 7. Delay Test Coverage for s1488

VI. CONCLUSIONS

In this paper, we have presented an efficient path-oriented ATPG that is SAT-based. Experimental results show that the SAT-based ATPG can dramatically speed up the path sensitization process. With a new false-path pruning technique, we demonstrate that our ATPG can be used to produce tests that detect transition faults through the longest timing paths. To justify the value of our path-oriented transition fault model, we measure quality via statistical simulation and defect injection. Experimental results show that our ATPG can produce higher-quality test sets than conventional delay test methodologies.

REFERENCES

[1] J. A. Waicukauski; E. Lindbloom; B. K. Rosen; and V. S. Iyengar. "Transition Fault Simulation", *IEEE Design and Testa, pp. 32-38,* April 1987

[2] A. K. Pramanick, S. M. Reddy. "On the Detection of Delay Faults", *Proc. ITC, pp. 845-856,* September 1988.

[3] A. K. Pramanick, S. M. Reddy. "On the fault coverage of gate delay fault detecting tests", *IEEE TCAD, Volume: 16 Issue: 1, pp. 78-94,* January 1997

[4] M. Nakao, et al. "Test Generation for Multiple-Threshold Gate-Delay Fault Model" *Proc. ATS'01, pp. 244-249,* 2001.

[5] G. L. Smith. "Model for Delay Faults Based Upon Paths", *Proc. ITC, pp. 342-349,* September 1985.

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

[7] A. K. Majhi. et al. "On Test Coverage of Path Delay Faults", *Proc. VLSI Design '96, pp. 418-421,* 1996.

[8] Kwang-Ting Cheng. "Transition Fault Simulation For Sequential Circuits", *Proc. ITC, pp. 20-24,* September 1992.

[9] Yun Shao, I. Pormeranz, and S. M. Reddy. "On Generating High Quality Tests for Transiton Faults", *Proc. 11th ATS, pp. 1-8,* 2002.

[10] W.-N. Li, S. M. Reddy, and S. Sahni. "On Path Selection in Combinational Logic Circuits", *IEEE TCAD, vol.8, pp. 56-63,* January 1989

[11] A. K. Majhi, V. D. Agrawal, J. Jacob, and L. M. Patnaik, "Line Coverage of Path Delay Faults", *IEEE Trans. on VLSI Systems, vol.8, no.5, pp.610-613,* October 2000.

[12] A. Murakmi, S. Kajihara, T. Sasao, R. Pomeranz, and S. M. Reddy, "Selection of potentially testable path delay faults for test generation", *Proc. ITC pp.376-384,* 2000

[13] Yun Shao, S. M. Reddy, I. Pomeranz, and S. Kajihara, "On Selecting Paths in Scan Designs" *IEEE European Test Workshop,* 2002.

[14] Manish Sharma, and Janak H. Patel, "Finding a Small Set of Longest Testable Paths that Cover Every Gate", *Proc. ITC pp.974-982,* 2002.

[15] Wangqi Qiu, and D. M. H. Walker, "Finding a Small Set of Longest Testable Paths that Cover Every Gate", *Proc. ITC,* 2003.

[16] Thomas H. Cormen, Charles E. Lesiserson, Ronald L. Rivest, Clifford Stein. *Introduction to Algorithms*. MIT Press; 2ed edition (September 1, 2001).

[17] F. Lu, Li-C. Wang, and K-T. and Ric C-Y. Huang. "A Circuit SAT Solver with Signal Correlation Guided Learning", *Proc. DATE* 2003.

[18] F. Lu, Li-C. Wang, and K-T. J. Moondanos, and H. Ziyad. "A Signal Correlation Guided ATPG Solver and Its Application for Solving Difficult Industrial Cases", *Proc. ACM/IEEE DAC* June 2003.

[19] K-T. Cheng, and H-C. Chen. "Classification and Identification of Non-robust Untestable Path Delay Faults", *IEEE TCAD, Vol: 15, pp. 845-853* August 1996.

[20] S. Kajihara, K. Kinoshita, I. Pomeranz, and S. M. Reddy, "A Method for Identifying Robust Dependent and Functionally Unsensitizable Paths" *Proc. VLSI Design 10th International Conf.,* 1997.

[21] Yun Shao, S. M. Reddy, S. Kajihara, and I. Pomeranz, "An Efficient Method to Identify Untestable Path Delay Faults" *Proc. Test Symposium, 10th Asian,* 2001.

[22] K. Fuchs, F. Fink, and M. H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults", *IEEE TCAD, Vol: 10, pp. 1323-1355* 1991.

[23] J.-J. Liou, A. Krstić, K.-T. Cheng, D. Mukherjee, and S. Kundu, "Performance Sensitivity Analysis Using Statistical Methods and Its Applications to Delay Testing", *Proc. ASP DAC, pp. 587-592,* Jan 2000.

[24] J.-J. Liou, L.-C. Wang, K.-T. Cheng, "On Theoretical and Practical Considerations of Path Selection For Delay Fault Testing", *Proc. ICCAD,* Nov. 2002.

[25] J.-J. Liou, A. Krstić, L.-C. Wang, K.-T. Cheng, "False-Path-Aware Statistical Timing Analysis and Efficient Path Selection for Delay Testing and Timing Validation", *Proc. DAC,* June 2002.

[26] Li-C. Wang, Angela Krstic, Kwang-Ting Cheng, Ray Mercer, Thomas Williams, Magdy Abadir. "Predicting Defect Level and Defect Detection Behavior: Logic Models Vs. Statistical Timing Defects", *Proc. ITC* 2003.