# Satisfiability-Based Functional Delay Fault Testing

*Joonyoung Kim*[†], *João Marques Silva*[††] *and Karem A. Sakallah*[†]
[†]*Advanced Computer Architecture Laboratory, Department of Electrical Engineering and Computer Science, University of Michigan 1301 Beal Ave. Ann Arbor, MI 48109, U.S.A*
[††]*IST/INESC, Cadence European Labs Lisbon, Portugal*

Key words: testing, delay fault testing, boolean satisfiability

Functional delay fault testing is concerned with propagating a transition from a primary input to a primary output of a combinational circuit. Since it does not consider individual paths in the circuit, it can overcome the biggest limitation of path delay fault testing: the explosion in the size of fault lists. Functional delay fault testing can also be used to derive test sets for IP (Intellectual Property) circuits whose implementation details are not provided. Boolean Satisfiability (SAT) and BDDs have been widely used for a variety of EDA (Electronic Design Automation) applications. Even though there have been few experimental studies to conclude the superiority of one to the other, they have been compared for a number of specific tasks in the EDA field. In this paper we show that SAT-based functional delay fault testing can yield very competitive results with careful construction of the CNF formulas for the target faults. In particular, using simple structural analysis of the circuit formulas of minimum size can be easily generated. CNF formula construction based on the circuit consistency function is presented and experimental results for ISCAS 85 and 89 circuits are reported.

## 1. INTRODUCTION

Delay fault testing, which addresses manufacturing defects that affect temporal behavior, is usually performed after a fabricated circuit has been tested for stuck-at faults. There are two fault models that have been widely used in delay fault testing: the *gate delay fault model* [3] which ascribes faulty behavior to individual gates having excessive delay, and the *path delay fault model* [16] which attempts to capture the distributed effects of defects over entire circuit paths leading to excessive path delays. In either case, the existence of a delay fault causes the circuit to fail to operate at the expected clock frequency.

The original version of this chapter was revised: The copyright line was incorrect. This has been corrected. The Erratum to this chapter is available at DOI: 10.1007/978-0-387-35498-9_57

L. M. Silveira et al. (eds.), *VLSI: Systems on a Chip*
© IFIP International Federation for Information Processing 2000

There have been numerous research efforts on delay fault testing based on the path delay fault model [5, 10]. The major drawback of this model is that the size of the fault list (number of paths) for circuits with a large amount of reconvergence can become exponentially large. Attempts to overcome this limitation included the use of incremental path sensitization [6, 8] and targeting a group of paths using the primitive path delay fault model [7]. Such attempts, however, still fail to handle circuits with billions of paths (e.g., C6288 of the ISCAS 85 benchmark suites [2]).

An alternative to these two fault models, called the *functional delay fault model*, was first proposed in [12] and also investigated in [17]. In [13] two approaches for functional delay fault test generation, one based on a Binary Decision Diagram (BDDs) formulation and the other on a Boolean Satisfiability (SAT) formulation, were described and compared.[1] The experimental results indicated that several hours of run time for the ISCAS 85 benchmarks circuits were required in both approaches, with the BDD method resulting in a slightly better performance.

The Boolean satisfiability problem (SAT) has received a lot of attention recently, resulting in a number of robust and efficient heuristics and implementations [1, 14, 18]. It has also been used in many CAD applications such as test pattern generation for both stuck-at and delay fault models, logic verification and timing analysis [4, 8, 9, 11, 15]. In this paper, we present a SAT-based test pattern generation method for the functional delay fault model. We show that with careful construction of the target CNF formulas, this method can yield very competitive results. Promising results for both ISCAS 85 and 89 circuits are presented.

The remainder of the paper is organized as follows. In the next section, definitions that are used throughout the paper are presented. In Section 3, a method to generate the target CNF formula for functional delay fault testing based on the circuit consistency functions is presented with an example. In Section 4, experimental results for ISCAS 85 and 89 circuits are presented and Section 5 concludes the paper.

## 2. DEFINITIONS

Most of the definitions in Section 2.1 and Section 2.2 are taken directly from [15]. They are repeated here for completeness.

[1] The details of CNF formula generation for the SAT approach were not described.

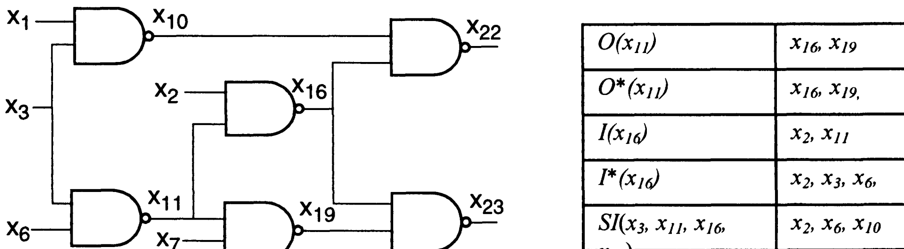
## 2.1 Combinational Circuits

A combinational circuit $C$ is represented as a directed acyclic graph $C = (V, E)$ where $V$ denotes the circuit nodes and $E \subseteq V \times V$ denotes the connections between nodes. The following definitions also apply:

- $O(x)$ denotes the **fanout** nodes of node $x$, i.e., $\{y \in V \mid (x, y) \in E\}$.
- $O^*(x)$ denotes the **transitive fanout** nodes of node $x$, i.e., the set of all nodes $y$ such that there is a path from $x$ to $y$.
- $I(x)$ denotes the **fanin** nodes of node $x$, i.e., $\{y \in V \mid (y, x) \in E\}$.
- $I^*(x)$ denotes the **transitive fanin** nodes of node $x$, i.e., the set of all nodes $y$ such that there is a path from $y$ to $x$.
- $SI(\Psi)$ denotes the **side inputs** of a set of nodes $\Psi \subseteq V$ and is defined as follows:

  $SI(\Psi) = \{x \mid x \in I(w) \wedge w \in \Psi \wedge x \notin \Psi\}$

The set of primary input nodes are referred to as PI, and the set of primary output nodes as PO. Figure 1 illustrates the definitions given in this section for a small benchmark circuit from [2].




| $O(x_{11})$ | $x_{16}, x_{19}$ |
|---|---|
| $O^*(x_{11})$ | $x_{16}, x_{19},$ |
| $I(x_{16})$ | $x_2, x_{11}$ |
| $I^*(x_{16})$ | $x_2, x_3, x_6,$ |
| $SI(x_3, x_{11}, x_{16},$ | $x_2, x_6, x_{10}$ |

*Figure 1.* Example ISCAS 85 circuit C17

## 2.2 Boolean Satisfiability

We consider Boolean functions represented in Conjunctive Normal Form (CNF). A *literal* is an occurrence of a variable $x$ or its complement $x'$. A *clause* is the disjunction (OR) of literals. A CNF formula $\varphi$ on $n$ binary variables $x_1, x_2, \ldots, x_n$ is the conjunction (AND) of $m$ clauses $w_1, w_2, \ldots, w_m$. A CNF formula is said to be *satisfiable* when there is at least one truth

assignment to its variables that makes all clauses equal to 1. A CNF formula is said to be *unsatisfiable* when no such assignment exists.

For each gate $g$ in a circuit, the *gate consistency function* $\varphi_g$ is a Boolean function that denotes the valid input-output assignments admissible by the gate's logic function. A detailed description of the gate consistency function and its CNF representation for primitive gates is given in [15]. The *circuit consistency function* of is defined as the conjunction of the gate consistency functions for each node in the circuit. If we view a CNF formula as a set of clauses, the circuit consistency function can be defined using a set union operator as follows:

$$\varphi = \bigcup_{x \in V} \varphi_x$$

## 2.3 Functional Delay Fault Testing

The objective of functional delay testing is to propagate a transition $tI \in \{rising, falling\}$ from a primary input node to a primary output node as a transition $tO \in \{rising, falling\}$. By analogy to the path delay fault model, robust propagation for functional delay fault testing can be defined as follows:

***Definition*** *1: For a given fault* $(I, O, tI, tO)$, *a two-pattern input combination* $(v_1, v_2)$ *is said to function-robustly propagate a transition tI from input I to a transition tO on output O if the value on O changes if and only if the value on I changes.*

In generating a test vector pair, two modes can be considered. In the single-input-transition mode, only one primary input is allowed to change, all other inputs being set to fixed values. In the multi-input- transition mode, any number of primary inputs are allowed to change In this paper, we consider only the single-input-transition mode; it must be noted, though, that the proposed test pattern generation procedure can be readily extended to the multi-input-transition mode.

***Definition*** *2: A two-pattern input combination* $(v_1, v_2)$ *is a function robust test for a given fault* $(I, O, tI, tO)$ *under the single-input-transition mode if and only if I is the only input that changes its values between* $v_1$ *and* $v_2$ *as a transition tI and the transition tO is observed at the output O.*

Consider a primary output $O$ of a circuit that implements the function $f$ and a primary input $I$ of the circuit. The detection of the faults $(I, O, rising, rising)$ and $(I, O, falling, falling)$ can be accomplished by checking the function $f_I \,?\, (f_{I'})'$ for satisfiability. To detect the fauits $(I, O, rising, falling)$ and $(I, O, falling, rising)$ the satisfiability of the function $f_{I'} \,?\, (f_I)'$ must be checked. Note that these two formulas represent the two terms of $df / dI$, the Boolean derivative of $f$ with respect to $I$.

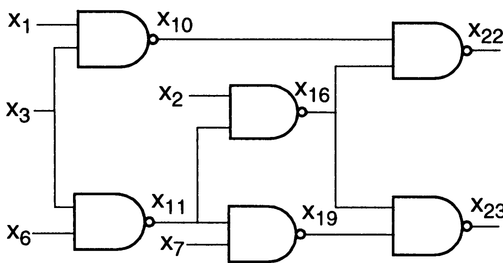
# 3. FUNCTIONAL DELAY FAULT GENERATION

## 3.1 Fault List Construction

Before the test pattern generation procedure can begin, a fault list needs to be created. The method used in [12] is used in constructing fault lists. The upper bound on the size of the fault list for a circuit with $|PI|$ inputs and $|PO|$ outputs is $4|PI||PO|$ since for each input/output pair, there are four $(I,O,tI,tO)$ tuples.

There are two techniques to reduce the size of the fault list. First, for a given output, the fan-in cone for the output node can be identified and only the primary input nodes that appear in the fan-in cone are considered when creating the fault list. Second, a check can be performed for each input/output pair to determine whether there is at least one path between them with an even or an odd number of inversions. If there are no paths with an even number of inversions $(I,O,tI,tO)$ tuples with identical $(tI,tO)$ values can be dropped from the fault list. Similarly, $(I,O,tI,tO)$ tuples with opposite $(tI,tO)$ values can be dropped if there are no path between $I$ and $O$ with an odd number of inversions.

Figure 2 shows the fault list created for the C17 circuit. E denotes that there is at least one path between the given primary input and primary output nodes with an even number of inversions and O denotes that there is at least one path with an odd number of inversion. "-" denotes that there are no paths between the two given nodes.




| PI | PO | Parity |
|---|---|---|
| $x_1$ | $x_{22}$ | E |
| $x_2$ | $x_{22}$ | E |
| $x_3$ | $x_{22}$ | E, O |
| $x_6$ | $x_{22}$ | O |
| $x_7$ | $x_{22}$ | - |
| $x_1$ | $x_{23}$ | - |
| $x_2$ | $x_{23}$ | E |
| $x_3$ | $x_{23}$ | O |
| $x_6$ | $x_{23}$ | O |
| $x_7$ | $x_{23}$ | E |

*Figure 2.* Fault list for C17

## 3.2 CNF Formula Construction

Once a fault list is set up, test generation is performed for each fault in the fault list. The overall flow of test pattern generation for delay fault testing is shown in Figure 3.

Since the formulas that must be checked involve both the positive and negative cofactors of each output with respect to each input in its transitive fan-in, the circuit is duplicated and consistency CNF formulas for both "co-factor" circuits are generated. A set of clauses from the CNF formulas thus created are then selectively chosen to create a target SAT instance for a specific functional delay fault. The clauses from the duplicate circuit are added to the target formula only when necessary. After generating the circuit consistency function, outputs are chosen one at a time. For each $PO$, the fan-in cone is identified. Then for each $PI$ which has at least one path to the chosen output node, the fanout cone of the node is identified. For each $(PI, PO)$ pair $(i, o)$, the set of nodes that can potentially propagate a transition can be identified by $(I^*(o)|\ o)\quad (O^*(i)|\ i)$. The function duplicate_nodes() will extract the gate consistency functions from the duplicated circuit for those nodes.

Figure 4 presents an example of the test pattern generation procedure with a target fault $(x_6, x_{22}, \text{rising}, \text{rising})$. The fan-in cone for the target output node $x_{22}$ is shown in the upper half. Since $(I^*(x_{22})|\ x_{22})\quad (O^*(x_6)|\ x_6) = \{x_6, x_{11}, x_{16}, x_{22}\}$, the duplicated nodes for these four nodes are identified and their gate consistency functions are selected by the function duplicate_nodes(). The function connect_side_inputs() identifies the side inputs of the given nodes in the duplicated circuit and connects them to the corresponding nodes in the original circuit. In this example, $SI(x_6, x_{11}, x_{16}, x_{22}) = \{x_2, x_3, x_{10}\}$, and the clauses that enforce the condition that each node in this set has the same value as its duplicate equivalent are added, which has the same effect as connecting a node in the original circuit to its duplicate node. Finally, for each fault between the target primary input and output pair, constraints are added based on the direction of the input and output transitions. For the example in Figure 4, the fault $(x_6, x_{22}, \text{rising}, \text{rising})$ is detected if the function

$$f_{x_6}\ ?\overline{f_{\overline{x_6}}}$$

is satisfiable, where $f$ is the logic function at output node $x_{22}$. This is accomplished by setting $x_6$ and $x_{22}$ to logic 1 and $x'_6$ and $x'_{22}$ to logic value 0.

**begin** functional delay fault generation
**foreach** $o$ of $PO$
  $\varphi_{in}$ = Identify $I*(o)$;
  **foreach** $i$ of $PI$ // consider only $i$L $I*(o)$
    Identify $O*(i)$;
    $\varphi_{dup}$ = duplicate_nodes( $(I*(o) | \quad o) \quad (O*(i) | \quad i)$ );
    $\varphi_{conn}$ = connect_side_inputs( $SI((I*(o) | \quad o) \quad (O*(i) | \quad i))$ );
    **foreach** $(tI,tO)$L $\{(r,r),(r,f)\}$
      $\varphi_{cons}$ = generate_constraints( $(tI,tO)$ );
      $\varphi = \varphi_{in} | \quad \varphi_{dup} | \quad \varphi_{conn} | \quad \varphi_{cons}$;
      solve( $\varphi$ );
**end**

*Figure 3.* Overall flow of functional delay fault test generation

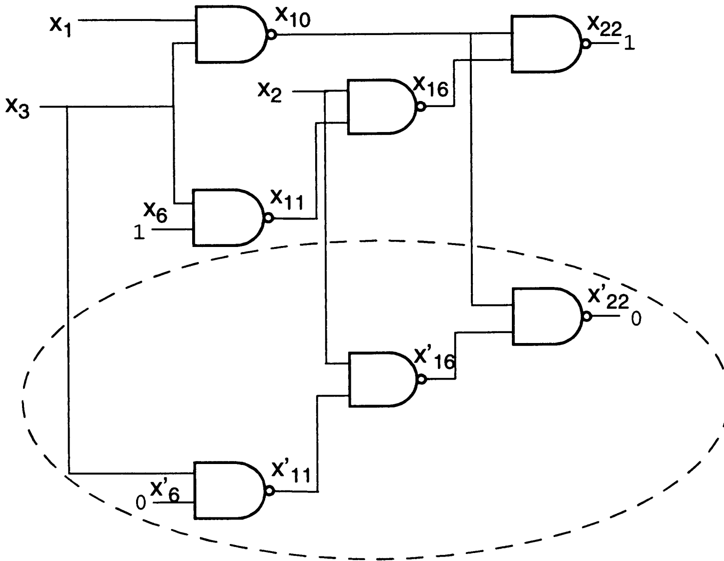



*Figure 4.* Target circuit for a fault $(x_6, x_{22}, \text{rising}, \text{rising})$

# 4. EXPERIMENTAL RESULTS

The functional delay fault test pattern generator described in this paper was implemented in C++ and integrated with the GRASP SAT solver [14]. The experiment was run on a PC running Linux with a Pentium II 300 MHz CPU and 256 Mbyte of memory. The results for ISCAS 85 circuits are presented in Table 1.

The first two columns list the name and number of faults of each circuit. From column 3 to column 5, D denotes detectable faults, R redundant faults and A aborted faults. %D in column 6 denotes the percentage of detectable faults with respect to the total number of faults. Columns 7 and 8 give the average size of the generated CNF formulas for each circuit. Although the complexity of SAT problems is not necessarily proportional to the size of the target SAT formula, it is observed that large CNF formulas usually take longer to solve. Column 7 gives average number of variables and column 8 gives the average number of clauses in each formula. Finally column 9 shows the run time in seconds. The GRASP SAT solver is called with the "+dDLCS" switch, one of the decision making heuristics that turned out to be most effective for this application. The time limit for each fault was set to 100 seconds.

Functional delay test pattern can be generated for all the circuits in Table 1 with no aborted faults except for C6288. For this circuit, we applied an approximate method introduced in [13] (see Table 2). With this approximation method, a certain percentage of primary inputs are assigned random fixed values (0 or 1) before the target SAT instance is solved. The first column in Table 2 denotes the percentage of primary inputs that are assigned fixed values. This method can cause a detectable fault to be declared as redundant, but will not make a redundant fault detectable; it provides a lower bound on the number of detectable faults.

The same experiment was performed with the combinational portions of the ISCAS 89 benchmark circuits. The results are reported in Table 3. None of the faults were aborted in this case, with very reasonable run times.

*Table 1.* Results for ISCAS 85 circuits

| Circuit | # faults | #D | #R | #A | %D | Avg # vars | Avg # cls | Time(s) |
|---|---|---|---|---|---|---|---|---|
| C432 | 794 | 538 | 256 | 0 | 67.8 | 249.0 | 667.6 | 84.1 |
| C499 | 5248 | 5184 | 64 | 0 | 98.8 | 190.5 | 518.7 | 525.8 |
| C880 | 1004 | 1004 | 0 | 0 | 100.0 | 162.2 | 385.4 | 79.8 |
| C1355 | 5248 | 5184 | 64 | 0 | 98.8 | 442.6 | 1210.3 | 2628.4 |
| C1908 | 1870 | 1758 | 112 | 0 | 94.0 | 619.4 | 1522.4 | 322.7 |
| C2670 | 2440 | 2160 | 280 | 0 | 88.5 | 456.2 | 1125.5 | 487.8 |
| C3540 | 2284 | 2242 | 42 | 0 | 98.2 | 1033.4 | 2817.4 | 5308.2 |
| C5315 | 7440 | 7328 | 112 | 0 | 98.5 | 368.5 | 906.3 | 901.3 |
| C6288 | 3036 | 2862 | 44 | 130 | 94.3 | 2299.3 | 6826.7 | 30490.0 |

| Circuit | # faults | #D | #R | #A | %D | Avg # vars | Avg # cls | Time(s) |
|---|---|---|---|---|---|---|---|---|
| C7552 | 7144 | 6228 | 916 | 0 | 87.2 | 581.0 | 1374.5 | 1215.3 |

*Table 2.* Results for C6288 using approximation

| Approximation factor | #D | #R | #A | %D | time (s) |
|---|---|---|---|---|---|
| 10% | 2950 | 46 | 40 | 97.2 | 35950.9 |
| 25% | 2972 | 54 | 10 | 97.9 | 32456.7 |
| 50% | 2972 | 62 | 2 | 97.9 | 31162.7 |
| 75% | 2964 | 72 | 0 | 97.6 | 30876.7 |

*Table 3.* Results for ISCAS 89 circuits

| Circuit | # faults | #D | #R | #A | %D | Avg # vars | Avg # cls | Time(s) |
|---|---|---|---|---|---|---|---|---|
| s208.1 | 152 | 152 | 0 | 0 | 100.0 | 48.8 | 108.2 | 6.8 |
| s298 | 230 | 218 | 12 | 0 | 94.8 | 39.3 | 96.7 | 9.9 |
| s344 | 282 | 278 | 4 | 0 | 98.6 | 47.4 | 108.2 | 12.8 |
| s349 | 284 | 280 | 4 | 0 | 98.6 | 47.6 | 108.8 | 12.3 |
| s382 | 408 | 400 | 8 | 0 | 98.0 | 41.0 | 91.4 | 17.5 |
| s386 | 308 | 308 | 0 | 0 | 100.0 | 49.7 | 111.7 | 13.5 |
| s400 | 388 | 380 | 8 | 0 | 97.9 | 41.3 | 93.1 | 16.8 |
| s420 | 390 | 390 | 0 | 0 | 100.0 | 72.5 | 157.0 | 19.0 |
| s420.1 | 424 | 424 | 0 | 0 | 100.0 | 72.3 | 156.8 | 20.9 |
| s444 | 544 | 528 | 16 | 0 | 97.1 | 46.5 | 105.9 | 24.0 |
| s510 | 310 | 310 | 0 | 0 | 100.0 | 77.1 | 189.8 | 16.0 |
| s526 | 492 | 480 | 12 | 0 | 97.6 | 47.4 | 113.6 | 21.8 |
| s526n | 492 | 480 | 12 | 0 | 97.6 | 47.4 | 113.6 | 22.0 |
| s641 | 1064 | 1010 | 54 | 0 | 94.9 | 160.1 | 349.6 | 71.4 |
| s713 | 1148 | 994 | 154 | 0 | 86.6 | 165.3 | 370.4 | 78.5 |
| s820 | 514 | 514 | 0 | 0 | 100.0 | 64.3 | 165.3 | 25.8 |
| s832 | 514 | 514 | 0 | 0 | 100.0 | 64.4 | 167.8 | 25.7 |
| s838 | 790 | 790 | 0 | 0 | 100.0 | 112.4 | 242.7 | 45.3 |
| s838.1 | 1352 | 1352 | 0 | 0 | 100.0 | 107.2 | 222.8 | 75.0 |
| s953 | 934 | 934 | 0 | 0 | 100.0 | 99.1 | 232.7 | 52.3 |
| s1196 | 1208 | 1158 | 50 | 0 | 95.9 | 183.3 | 469.8 | 94.5 |
| s1238 | 1224 | 1160 | 64 | 0 | 94.8 | 188.2 | 497.8 | 98.4 |
| s1423 | 5836 | 5458 | 378 | 0 | 93.5 | 209.7 | 477.0 | 469.4 |
| s1488 | 794 | 792 | 2 | 0 | 99.7 | 95.9 | 240.5 | 44.6 |
| s1494 | 794 | 792 | 2 | 0 | 99.7 | 95.7 | 241.3 | 45.1 |
| s5378 | 5760 | 5404 | 356 | 0 | 93.8 | 197.1 | 436.4 | 442.6 |
| s9234.1 | 9474 | 8070 | 1404 | 0 | 85.2 | 327.0 | 738.7 | 999.9 |
| s13207.1 | 11448 | 10286 | 1162 | 0 | 89.8 | 445.8 | 1135.4 | 1676.9 |
| s15850.1 | 57160 | 44326 | 12834 | 0 | 77.5 | 973.6 | 2369.7 | 15440.1 |
| s38417 | 87394 | 83236 | 4158 | 0 | 95.2 | 403.4 | 922.6 | 11117.4 |
| s38584.1 | 47272 | 41420 | 5852 | 0 | 87.6 | 194.1 | 449.5 | 3697.8 |
| s35932 | 22274 | 20674 | 1600 | 0 | 92.8 | 92.5 | 223.6 | 1228.4 |

# 5. CONCLUSIONS

In this paper, a satisfiability-based functional delay fault test generation method which uses the circuit consistency function in generating the target CNF formula was presented. Promising results for ISCAS benchmark circuits are reported.

[12] suggests generating more than one test per each fault in the fault list in order to achieve high path delay fault coverage. Incremental Satisfiability (ISAT) has been proven successful in solving multiple instances of closely related SAT problems [8]. We are currently looking at using ISAT to generate multiple tests per each functional delay fault.

# 6. REFERENCES


[1] R. Bayardo Jr. and R. Schrag, "Using CSP Look-Back Techniques to Solve Real-World SAT Instances," *National Conference on Artificial Intelligence*, pp. 203-208, 1997.

[2] F. Brglez and H. Fujiwara, "A Neutral List of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," *International Symposium on Circuits and Systems*, 1985.

[3] J. L. Carter, V. S. Iyengar and B. K. Rosen, "Efficient Test Coverage Determination for Delay Faults," *International Test Conference*, pp. 418-427, September, 1987.

[4] C. -A. Chen and S. K. Gupta, "A Satisfiability-Based Test Generator for Path Delay Faults in Combinational Circuits," *Design Automation Conference*, pp. 209-214, 1996.

[5] K. -T. Cheng and H. -C. Chen, "Delay Testing For Non-Robust Untestable Circuits," *International Test Conference*, pp. 954-961, 1993.

[6] K. Fuchs, F. Fink and M. H. Schulz, "DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults," *IEEE Trans. on Computer-Aided Design*, vol. 10, pp. 1323-1335, October 1991.

[7] W. Ke and P. R. Menon, "Synthesis of Delay-Verifiable Combinational Circuits," *IEEE Trans. on Computers*, vol. 44, pp. 213-222, February 1995.

[8] J. Kim, J. Whittemore, J. P. M. Silva and K. A. Sakallah, "Incremental Satisfiability and its Application to Delay Fault Testing," *International Workshop on Logic Synthesis*, 1999.

[9] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. on Computer-Aided Design*, vol. 11, pp. 4-15, January 1992.

[10] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," *IEEE Trans. on Computer- Aided Design*, vol. 6, pp. 694-703, 1987.

[11] P. C. McGeer, A. Saldanha, P. R. Stephan, R. K. Brayton and A. L. Sangiovanni-Vincentelli, "Timing Analysis and Delay-Fault Test Generation Using Path-Recursive Functions," *International Conference on Computer Aided Design*, pp. 180-183, November, 1991.

[12] I. Pomeranz and S. M. Reddy, "Functional Test Generation for Delay Faults in Combinational Circuits," *International Conference on Computer Aided Design*, pp. 687-694, 1995.

[13] S. Tragoudas and M. Michael, "ATPG tools for Delay Faults at the Functional Level," *Design, Automation and Test in Europe*, 1999.

[14] J. P. M. Silva and K. A. Sakallah, "GRASP- A New Algorithm for Satisfiability," *International Conference on Computer Aided Design*, pp. 220-227, 1996.

[15] J. P. M. Silva and K. A. Sakallah, "Robust Search Algorithms for Test Pattern Generation," *Fault Tolerant Computing Symposium*, 1997.

[16] G. L. Smith, "Model for Delay Faults Based upon Paths," *International Test Conference*, pp. 342- 349, 1985.

[17] B. Underwood, W. -O. Law, S. Kang and H. Konuk, "Fastpath: A Path-Delay Test Generator for Standard Scan Designs," *International Test Conference*, pp. 154-163, 1994.

[18] H. Zhang, "SATO: An Efficient Propositional Prover," *International Conference on Automatic Deduction*, pp. 272-275, 1997.