A Satisfiability-Based Test Generator for Path Delay Faults in Combinational Circuits *

Chih-Ang Chen and **Sandeep K. Gupta**
Electrical Engineering – Systems
University of Southern California
Los Angeles CA 90089-2562

Abstract

This paper describes a new formulation to generate robust tests for path delay faults in combinational circuits based on Boolean satisfiability. Conditions to detect a target path delay fault are represented by a Boolean formula. Unlike the technique described in [16], which extracts the formula for each path delay fault, the proposed formulation needs to extract the formula only once for each circuit cone. Experimental results show tremendous time saving on formula extraction compared to other satisfiability-based ATPG algorithms. This also leads to low test generation time, especially for circuits that have many paths but few outputs. The proposed formulation has also been modified to generate other types of tests for path delay faults.

1 Introduction

The problem of *automatic test pattern generation (ATPG)* for a target fault in a circuit is to find a set of assignments at the primary inputs such that the fault is excited, and the fault effect is propagated to the primary output(s). In the past two decades, tremendous progress has been made on the design of efficient ATPG algorithms, which can be broadly categorized as *structural*, *algebraic*, or *satisfiability-based*.

Structural algorithms [9, 10] directly analyze the gate-level description of a circuit and implicitly enumerate possible input combinations to find test patterns. Extensions of these techniques to generate two-pattern tests for delay faults are also well studied [8, 14]. Efficiency of these approaches often relies on the "bags of tricks" to avoid entering the *non-solution area* [5].

*This research was funded by NSF Research Initiation Award no. MIP-9210871 and NSF CAREER Award no. MIP-9502300.

An algebraic algorithm converts the test generation problem into an algebraic formula and applies algebraic techniques to simplify and then solve the formula to obtain a test. Early algebraic techniques based on *Boolean difference* [18] were not practical due to their high computational complexity. Recent developments in efficient circuit representations by *binary decision diagrams (BDDs)* [4] have inspired many algebraic ATPG algorithms for single stuck-at faults [19, 20] and delay faults [1]. These techniques work well for circuits that can be efficiently represented by BDDs. For some circuits, however, the BDD representations are impractically large, making the BDD-based techniques inapplicable to these circuits.

Satisfiability-based algorithms [5, 12, 21] translate the test generation problem into a formula in *conjunctive normal form (CNF)* [7]. A branch-and-bound strategy is then used to find a *satisfying assignment*, which corresponds to a test for the target fault. The performance of the implementation described in [21] compares favorably to the best known structural algorithms. An extension of satisfiability-based ATPG algorithms to generate robust tests for path delay faults has been reported in [16]. It was proven that a robust path delay fault in a circuit is detectable, if and only if an equivalent stuck-at fault is detectable in a modified circuit, which has at most four times the number of gates in the original circuit. A robust test for the path delay fault can then be determined by generating a test for the equivalent stuck-at fault in the modified circuit using the technique described in [21]. Typically, 70% to 80% of the overall test generation time for stuck-at faults is spent on CNF formula extraction. In test generation for single stuck-at faults, the time required for formula extraction is compensated by a faster satisfiability solver. A test generator for path delay faults, however, may need to consider an exponential number of paths. Due to this fact, a large amount of time is spent on CNF formula extraction using the approach described in [16], thereby adversely affecting the overall test generation time for circuits with a large number of paths.

This paper describes a new CNF formulation for path delay faults that can be extracted much more efficiently, making it practical for circuits with a large number of

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

paths. Unlike other satisfiability-based techniques, the proposed formulation needs to extract the formula only once for each cone. All path delay faults in a cone use the same CNF formula merely by making appropriate on-path and off-path value assignments before solving the CNF formula. Experimental results show tremendous time saving on formula extraction compared to other similar techniques. This also leads to significant savings on overall test generation time, especially for circuits with many paths but few outputs. The proposed formulation has also been modified to generate different types of tests for path delay faults, e.g. *hazard-free robust*, *robust*, and *non-robust tests* [15].

The paper is organized as follows. Section 2 describes the basic terminology used in this paper. Formulation of the ATPG problems for stuck-at faults and path delay faults as the Boolean satisfiability problem is presented in Sections 3 and 4, respectively. Experimental results are given in Section 5. Finally, conclusions are presented in Section 6.

2 Basic Terminology

A *literal* is a Boolean variable or its negation (e.g. a or $\overline{a}$). An *OR-clause* is an OR of one or more literals. An n-clause is an OR-clause with exactly n distinct literals. A Boolean formula is in *conjunctive normal form (CNF)*, if it is expressed as an AND of OR-clauses. A Boolean formula is in n-CNF, if each OR-clause is an n-clause. An *assignment* for a Boolean formula is a set of Boolean input values (0 or 1). A *satisfying assignment* for a single-output Boolean formula y is an assignment such that y evaluates to 1. A Boolean formula is *satisfiable* if it has a satisfying assignment. The problem of *Boolean satisfiability* is to determine whether a Boolean formula is satisfiable. As an example, the Boolean formula $y = (a+b+\overline{d})(\overline{a}+\overline{c}+d)$ is in 3-CNF. The formula is satisfiable with the satisfying assignment $\{a = 1, c = 0\}$.

3 ATPG Formulation: Stuck-at Faults

Test generation based on satisfiability can be divided into two independent steps: extraction of the CNF formula and identification of a satisfying assignment. This section describes CNF formula extraction for single stuck-at faults. Much of the description is due to [12] and is included here to delineate later discussion on test generation for path delay faults.

First, consider a 2-input AND gate represented in *equation form* by $y = ab$. Alternatively, the equation can be written in CNF as

$$(a + \overline{y})(b + \overline{y})(\overline{a} + \overline{b} + y) = 1. \qquad (1)$$

Note that only the values in the truth table of an AND gate, where the output y has a determined binary value 0/1, can satisfy the CNF formula. The CNF formulas for other basic gates with multiple inputs can be derived similarly.

A Boolean network consists of gates interconnected by wires with possible fanout stems. The network can be represented in CNF by concatenating the CNF formula for each individual gate. For example, the circuit shown in Figure 1 can be written in CNF as

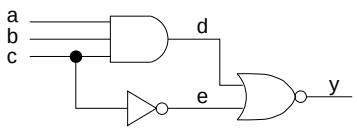

Figure 1: Example circuit

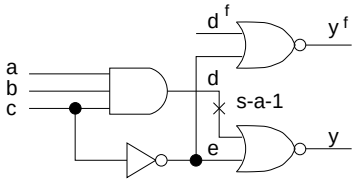

Figure 2: Faulty circuit for d *s-a-1*

$$\mathcal{C}_g = \underbrace{(a + \overline{d})(b + \overline{d})(c + \overline{d})(\overline{a} + \overline{b} + \overline{c} + d)}_{\text{AND}}$$
$$\underbrace{(c + e)(\overline{c} + \overline{e})}_{\text{NOT}} \underbrace{(\overline{d} + \overline{y})(\overline{e} + \overline{y})(d + e + y)}_{\text{NOR}} = 1. \qquad (2)$$

To model a faulty circuit, the gates and lines in the transitive fanout of the fault site are duplicated and a new literal is introduced for each duplicate line. For example, the faulty circuit for the fault d stuck-at-1 in the example circuit is shown in Figure 2. The CNF formula for the duplicate circuit is given by

$$\mathcal{C}_f = (\overline{d}^f + \overline{y}^f)(\overline{e} + \overline{y}^f)(d^f + e + y^f) = 1. \qquad (3)$$

The new variables d^f and y^f (faulty variables) are introduced to indicate that lines d and y may have different values in the normal and faulty circuits.

The effect of fault propagation can be formulated by introducing *active variables* and *active clauses*. For each line l in the transitive fanout of the fault site, an active variable l^a is defined. If l is on the fault propagating path, then the good value l and the faulty value l^f are different. The active clauses for the faulty circuit in Figure 2 are given by

$$\mathcal{C}_a = (\overline{d}^a + d + d^f)(\overline{d}^a + \overline{d} + \overline{d}^f)(\overline{y}^a + y + y^f)(\overline{y}^a + \overline{y} + \overline{y}^f) = 1. \qquad (4)$$

To detect a fault, the fault site must have different values in the good and faulty circuits, and the fault effect must be propagated to the primary output(s). For the fault d s-a-1, line d must have the value 0 in the good circuit and 1 in the faulty circuit (i.e. $d^a = 1$, $d = 0$, and $d^f = 1$). Since y is the only output, the fault effect must be observed at y (i.e. $y^a = 1$). Overall, test generation for the fault d s-a-1 in the example circuit has been translated to the problem of finding a satisfying

Table 1: Implication table for AND gate

	$s0$	$s1$	$\overline{s}0$	$\overline{s}1$	$x0$	$x1$	xx
$s0$	$s0$	$s0$	$s0$	$s0$	$s0$	$s0$	$s0$
$s1$	$s0$	$s1$	$\overline{s}0$	$\overline{s}1$	$x0$	$x1$	xx
$\overline{s}0$	$s0$	$\overline{s}0$	$\overline{s}0$	$\overline{s}0$	$x0$	$\overline{s}0$	$x0$
$\overline{s}1$	$s0$	$\overline{s}1$	$\overline{s}0$	$\overline{s}1$	$x0$	$\overline{s}1$	xx
$x0$	$s0$	$x0$	$x0$	$x0$	$x0$	$x0$	$x0$
$x1$	$s0$	$x1$	$\overline{s}0$	$\overline{s}1$	$x0$	$x1$	xx
xx	$s0$	xx	$x0$	xx	$x0$	xx	xx

Table 2: Encoding for $\mathcal{L}_7$

	$s0$	$s1$	$\overline{s}0$	$\overline{s}1$	$x0$	$x1$	xx
s	1	1	0	0	x	x	x
v	0	1	0	1	0	1	x

assignment for the CNF formula given by

$$\mathcal{C}_g\mathcal{C}_f\,\mathcal{C}_a d^a\overline{d}d^f y^a = 1. \tag{5}$$

Simple branch-and-bound heuristics can then be used to search for satisfying assignments. One possible solution is given by $\{a = 0, c = 1, d^a = 1, d = 0, d^f = 1, e = 0, y^a = 1, y = 1, y^f = 0\}$, which corresponds to the test $\{a = 0, b = x, c = 1\}$.

The technique described in [16] to generate a robust test for a path delay fault uses the above formulation to find a test for an *equivalent* stuck-at fault in the modified circuit. The equivalent stuck-at fault is located on the *I-edge* of the target path P, which is either a primary input or the output of an inverter fed by a primary input. In general, a stuck-at fault away from the primary outputs requires more memory to store the CNF formula and it may be harder to find a satisfying assignment, when compared to another fault closer to the primary outputs. In the following section, we will show how faulty and active variables can be eliminated in the proposed ATPG formulation for path delay faults.

4 Proposed ATPG Formulation: Path Delay Faults

CNF formula extraction for a path delay fault has the following major differences from that for a stuck-at fault: (1) the logic system used to represent signal values is different; (2) the fault propagation path of a path delay fault is known; and (3) a path delay fault can be detected by satisfying the on-path and off-path values. Due to the last two differences, extracting the CNF formula for a path delay fault is in fact simpler than that for a stuck-at fault as will be shown below.

4.1 Logic Systems

Many logic systems, which range from 5-valued to 23-valued, have been proposed in the literature [2, 8, 13, 14]. The proposed formulation uses the *7-valued logic system* $\mathcal{L}_7$ proposed in [13]. According to [8], $\mathcal{L}_7$ can be partitioned into 4 basic values and 3 composite values as $\mathcal{L}_7 = \{\{s0, \overline{s}0, s1, \overline{s}1\}, \{x0, x1, xx\}\}$. The second element of each value, which indicates the final value of a two-pattern test, is either 0, 1, or x (don't care). The first element of each value is either s (static), $\overline{s}$ (not static), or x (unknown). A signal line has the basic value $s0$ ($s1$), if both its initial and final values are 0 (1) and no transient hazard exists during the entire time interval. A signal line has the basic value $\overline{s}0$ ($\overline{s}1$), if the line has the final value 0 (1) but not $s0$ ($s1$). The implication table for a 2-input AND gate using $\mathcal{L}_7$ is shown in Table 1. Table 2 shows a simple encoding scheme using two **ternary** variables s and v to represent the first and second elements of each value in $\mathcal{L}_7$, respectively. This encoding is carefully chosen to minimize the number of clauses in the resulting CNF formulas. For each line l in a circuit, a two-tuple (l^s, l^v) is used to represent the code of the value on l.

4.2 Normal Circuit

For a 2-input AND gate represented by $y = ab$, the two-tuples (a^s, a^v), (b^s, b^v), and (y^s, y^v) are used to represent the codes on lines a, b, and y, respectively. Based on the implication table in Table 1 and the encoding in Table 2, the variables y^s and y^v can be represented by

$$y^s = a^s b^s + a^s\overline{a}^v + b^s\overline{b}^v \tag{6}$$
$$y^v = a^v b^v. \tag{7}$$

Equivalently, the two equations can be written as

$$(a^s b^s + a^s\overline{a}^v + b^s\overline{b}^v) \bigoplus y^s = 0 \tag{8}$$
$$a^v b^v \bigoplus y^v = 0. \tag{9}$$

After some Boolean manipulation, the equations can be represented in CNF by

$$\begin{aligned}\mathcal{E}_{AND}(a,b,c) = {} & \\ & (\overline{a}^s + \overline{b}^s + y^s)(\overline{a}^s + a^v + y^s)(\overline{b}^s + b^v + y^s) \\ & (a^s + b^s + \overline{y}^s)(a^s + \overline{b}^v + \overline{y}^s)(\overline{a}^v + b^s + \overline{y}^s) \\ & (a^v + \overline{y}^v)(b^v + \overline{y}^v)(\overline{a}^v + \overline{b}^v + y^v) = 1.\end{aligned} \tag{10}$$

Note that only those values in Table 1, where both y^s and y^v have determined binary values, can satisfy the CNF formula. The CNF formulas for other basic gates can be derived similarly. For a 2-input basic gate, there are 9 clauses in its CNF formula extracted for delay testing, of which 7 clauses have 3 literals and 2 clauses have 2 literals. A 1-input gate (an inverter or a buffer) is a special case which has 4 clauses, each with 2 literals.

A direct extension of the above approach to derive the CNF formula for an n-input basic gate leads to an exponential number of terms. Alternative, an n-input basic gate can be decomposed into $(n-1)$ 2-input gates as shown in Figure 3. Such a decomposition neither changes the number of paths in the circuit nor does it

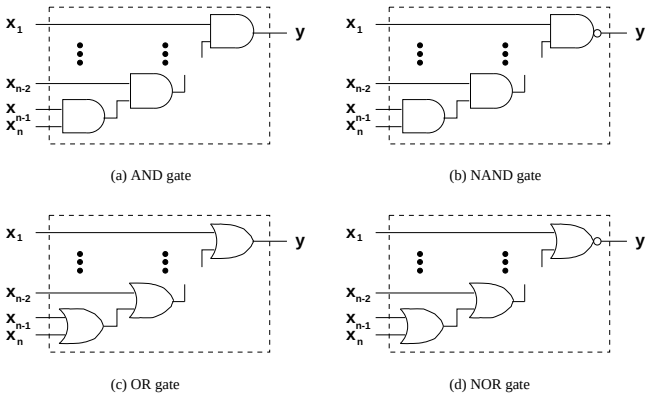

Figure 3: Decomposition of n-input gates

affect the detectability of the path delay faults [11]. The CNF formula for the n-input gate can then be obtained by concatenating the CNF formula for its constituent 2-input gates.

For a single-output circuit, the CNF formula for each individual gate can be concatenated to form a CNF formula for the circuit. The number of clauses and literals remains polynomial in terms of the number of gates in the circuit, as stated in the following theorem.

Theorem 1 *For a single-output circuit with* $N = \sum_{i=1}^{n} N_i$ *gates, where* N_i *is the number of* i*-inputs gates, there are* $R_N = 9(N-n+1) - 5N_1$ *clauses in its CNF representation for delay testing, of which* $7(N-n+1) - 7N_1$ *clauses have 3 literals and* $2(N-n+1)+2N_1$ *clauses have 2 literals.*

For a circuit with multiple outputs, the CNF formulas must be extracted once for each cone. If a gate G belongs to k cones, then the CNF formula for G must be extracted k times. In the worst case where all the m cones in an m-output circuit completely overlap, the number of clauses that need to be extracted is given by mR_N. Since $m << N$, the number of clauses and literals remains polynomial in terms of the number of gates for a general circuit.

4.3 Fault Excitation

During test generation, the literals are assigned values corresponding to the necessary conditions to detect the target path delay fault. By carefully implementing the path enumeration algorithm, the computation required to determine the necessary conditions for each path delay fault can be kept small. During traversal of the circuit to enumerate a path P_1, the necessary assignments to detect a delay fault on P_1 can be stored in a stack. Since only a few nodes need to be modified in *depth-first search* to identify the next path P_2, the stack can be updated incrementally and the necessary conditions for P_2 can be quickly determined.

Different types of tests for path delay faults have been proposed in the literature. In this paper, we consider three kinds of two-pattern tests: *restricted delay test pairs (RDTPs)* [17], robust tests, and non-robust tests, which differ on the constraints on the off-path input values as shown in Table 3. These off-path input constraints can be uniformly represented by fixing the variables for the off-path inputs before solving the CNF formula to find the two-pattern tests.

Table 3: Constraints on off-path input values

	RDTP	Robust		Non-robust
		rising	falling	
AND/NAND	s1	x1	s1	x1
OR/NOR	s0	s0	x0	x0

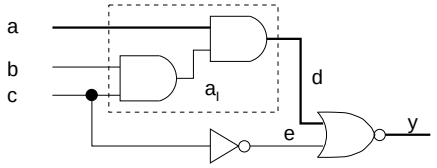

Figure 4: Decomposition of example circuit

Consider the circuit shown in Figure 4, which can be written in CNF by

$$\mathcal{E}_{AND}(b,c,a_I)\mathcal{E}_{AND}(a,a_I,d) \\ \mathcal{E}_{NOT}(c,e)\mathcal{E}_{NOR}(d,e,y) = 1. \qquad (11)$$

A robust test for the slow-to-rise delay fault on path a–d–y can be generated by fixing the on-path variables: $(a^s,a^v) = (0,1),(d^s,d^v) = (0,1),(y^s,y^v) = (1,0)$; and the off-path variables: $a_I^v = b^v = c^v = 1,(e^s,e^v) = (1,0)$. These constraints can be represented by adding the clauses $\overline{a}^s a^v \overline{d}^s d^v \overline{y}^s \overline{y}^v a_I^v b^v c^v e^s \overline{e}^v$ to Eq. 11. A possible satisfying assignment is given by $(a^s,a^v) = (0,1),(b^s,b^v) = (x,1),(c^s,c^v) = (1,1)$, which corresponds the robust test $\{a = \overline{s}1, b = x1, c = s1\}$. Using the same CNF formula, a non-robust test for the same path delay fault can be generated by changing the off-path variables: $a_I^v = b^v = c^v = 1, e^v = 0$. A possible satisfying assignment is given by $(a^s,a^v) = (0,1),(b^s,b^v) = (c^s,c^v) = (x,1)$, which corresponds to the non-robust test $\{a = \overline{s}1, b = x1, c = x1\}$.

Since the circuit in Figure 4 has a single output, the CNF formula extracted in Eq. 11, together with additional clauses corresponding to the necessary conditions for detection of the target fault, can be used to generate tests for other path delay faults in the circuit. For example, to generate a robust test for the slow-to-rise delay fault on the path c–a_I–d–y, the clause $\overline{c}^s c^v \overline{a}_I^s a_I^v \overline{d}^s d^v \overline{y}^s \overline{y}^v a^v b^v e^s \overline{e}^v$ can be added to Eq. 11. Since no satisfying assignment can be found, no robust test exists for the path delay fault. However, by adding the clauses $\overline{c}^s c^v \overline{a}_I^s a_I^v \overline{d}^s d^v \overline{y}^s y^v a^v b^v \overline{e}^v$ to Eq. 11, a non-robust test $\{a = x1, b = x1, c = \overline{s}1\}$ can be derived for the path delay fault.

4.4 Boolean Satisfiability

After an ATPG problem has been converted into a satisfiability problem, any satisfiability solver can be used to find a satisfying assignment that corresponds to a valid test, irrespective of the original circuit structure and the types of tests desired. Efficient branch-and-bound algorithms have been developed to avoid the exponential worst-case run time for solving the satisfiability problem. Extensive experiments have been performed in [21] to compare different search strategies that determine variable order for branching. In [5], the *transitive closure* of the *implication graph* derived from the 2-clauses in a CNF formula is used to determine global signal dependencies. These techniques can be applied directly to solve the CNF formulas extracted for path delay faults.

5 Experimental Results

The proposed ATPG program for path delay faults has been implemented and tested on the combinational parts of the ISCAS89 [3] circuits. The experimental results presented in the following are obtained using an HP-710 with 32 Mbytes of memory. The program consists of a front-end clause extractor that extracts the CNF formulas from the gate-level descriptions of the circuits. A path delay fault is activated by fixing values at the on-path and off-path inputs. Three kinds of two-pattern tests — RDTPs, robust tests, and non-robust tests — can be generated in current implementation. The satisfiability solver is based on the implementation in [21], originally integrated in an ATPG for single stuck-at faults.

The experimental results for test generation of robust tests are shown in Table 4. After the circuit name, the number of detected, untestable, and aborted path delay faults are shown in columns 2–5, respectively, followed by the total number of faults considered. In the experiments, each path delay fault is individually targeted. Note that the proposed technique is able to find a robust test or prove that no robust test exists for every path delay fault in these circuits. Columns 6–7 show the time spent on extraction of CNF formulas (CNF) and identification of satisfying assignments (SAT), followed by the total test generation time. The percentage of the formula extraction time over the total test generation time is also included. As shown in the table, formula extraction (CNF) accounts for 10% of the total computation time on the average, compared to 63% reported in [16]. The formula extraction time can be reduced significantly, because the formula extraction needs to be performed only once for each cone and the necessary conditions to detect the path delay faults can be quickly determined and updated.

One important feature of the proposed technique is that the time spent on formula extraction does not grow rapidly with the number of paths in the circuit. Consider the circuits `s1238` (with 428 gates and 32 outputs) and `s1423` (with 490 gates and 79 outputs) in Table 4. The number of paths in `s1423` is about 12.6 times that in `s1238`. Though `s1423` has more gates and outputs than `s1238`, the formula extraction time for `s1423` is only about 7.7 times that for `s1238`, while the percentage of the formula extraction over the total test generation time increases only slightly from 5.12% to 5.62%.

Similar experiments have been performed to generate RDTPs and non-robust tests for the benchmark circuits. The results can be found in [6]. In summary, no fault is aborted for all types of tests and the number of untestable faults increases (decreases) as more (less) restricted tests are generated for the path delay faults. The total run time and the percentage of the formula extraction over the total run time are about the same for different types of tests for path delay faults.

6 Conclusion

In this paper, the problem of ATPG for path delay faults has been converted to a Boolean satisfiability problem. Any set of input assignments that satisfies the CNF formula is a test for the target fault. The proposed formulation is simpler and faster than the technique proposed in [16]. Unlike their technique which extracts the CNF formula for each path delay fault, the proposed formulation needs to extracts the CNF formula only once for each cone. All path delay faults in the same cone can use the same CNF formula, but only differ on the on-path and off-path input constraints. Experimental results show tremendous time savings on formula extraction. The proposed formulation has also been modified to generate other types of tests for path delay faults.

Satisfiability-based ATPG algorithms have the advantages of simplicity and uniformity, and is very well suited for path delay faults because no faulty or active variables is required. The formula extraction time is proportional to the number of circuit's outputs and is typically small even for circuits with a large number of paths. The proposed formulation can also be used to solve many *design-for-testability* problems. Some preliminary results can be found in [6].

References

[1] D. Bhattacharya, P. Agrawal, and V. D. Agrawal. Delay Fault Test Generation for Scan/Hold Circuits using Boolean Expressions. In *Proc. IEEE-ACM Design Automation Conference*, pages 159–164, 1992.

[2] S. Bose, P. Agrawal, and V. D. Agrawal. Logic Systems for Path Delay Test Generation. In *Proc. European Design Automation Conf.*, pages 200–205, 1993.

[3] F. Brglez, D. Bryan, and K. Kozminski. Combinational Profiles of Sequential Benchmark Circuits. In *IEEE Int. Symp. on Circuits and Systems*, pages 1929–1934, 1989.

[4] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Trans. on Computers*, C-35(8):677–691, Aug. 1986.

[5] S. T. Chakradhar, V. D. Agrawal, and S. G. Rothweiler. A Transitive Closure Algorithm for Test Generation. *IEEE Trans. on CAD*, 12(7):1015–1028, July 1993.

Table 4: Test generation for robust tests

Ckt	path delay faults				CPU time (sec)			CNF /
	det.	untst.	ab.	total	CNF	SAT	total	total (%)
s344	611	99	0	710	0.35	3.24	3.59	9.75
s349	611	119	0	730	0.29	3.31	3.60	8.06
s382	667	133	0	800	0.38	2.49	2.87	13.24
s386	413	1	0	414	0.30	1.57	1.87	16.04
s400	663	233	0	896	0.41	2.69	3.10	13.23
s420	738	0	0	738	0.33	5.19	5.52	5.98
s444	586	484	0	1070	0.47	3.44	3.92	11.99
s510	729	9	0	738	0.47	6.21	6.68	7.04
s526	694	126	0	820	0.42	3.39	3.81	11.02
s641	1979	1509	0	3488	2.98	34.91	37.89	7.87
s713	1184	42440	0	43624	25.69	126.20	151.89	16.91
s820	980	4	0	984	0.62	9.02	9.65	6.42
s832	984	28	0	1012	0.82	9.60	10.42	7.87
s838	2018	0	0	2018	1.91	31.59	33.50	5.70
s953	2302	10	0	2312	1.99	24.63	26.62	7.47
s1196	3581	2615	0	6196	8.40	152.28	160.68	5.22
s1238	3589	3529	0	7118	10.08	186.78	196.86	5.12
s1423	28696	60756	0	89452	78.09	1310.32	1388.41	5.62
s1488	1875	49	0	1924	1.52	20.26	21.78	6.98
s1494	1882	70	0	1952	1.45	20.79	22.24	6.52
s5378	18656	8428	0	27084	42.33	524.61	566.94	7.46
s9234	21389	468319	0	489708	1629.09	8575.07	10204.16	15.96

[6] C.-A. Chen and S. K. Gupta. A Satisfiability-Based Test Generator for Path Delay Faults in Combinatio nal Circuits. Technical Report CENG 96-07, Univ. of Southern California, 1996.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. The MIT Press, New York, NY, 1992.

[8] K. Fuchs, F. Fink, and M. H. Schulz. DYNAMITE: An Efficient Automatic Test Pattern Generation System for Path Delay Faults. *IEEE Trans. on CAD*, 10(10):1323–1335, Oct. 1991.

[9] H. Fujiwara and T. Shimono. On the Acceleration of Test Generation Algorithms. *IEEE Trans. on Computers*, C-32(12), Dec. 1983.

[10] P. Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Trans. on Computers*, C-30(3), Mar. 1981.

[11] S. Kundu, S. M. Reddy, and N. K. Jha. Design of Robustly Testable Combinational Logic Circuits. *IEEE Trans. on CAD*, 10(8):1036–1048, Aug. 1991.

[12] T. Larrabee. Efficient Generation of Test Patterns Using Boolean Difference. In *Proc. IEEE Int. Test Conf.*, pages 795–801, 1989.

[13] C. J. Lin and S. M. Reddy. On Delay Fault Testing in Logic Circuits. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 148–151, 1986.

[14] C. J. Lin and S. M. Reddy. On Delay Fault Testing in Logic Circuits. *IEEE Trans. on CAD*, CAD-6(5):694–703, Sept. 1987.

[15] A. K. Pramanick and S. M. Reddy. On the Design of Path Delay Fault Testable Combinational Circuits. In *Proc. IEEE Int. Conf. on Computer-Aided Design*, pages 374–381, 1990.

[16] A. Saldhana, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Equivalence of Robust Delay Fault and Single Stuck Fault Test Generation. In *Proc. IEEE-ACM Design Automation Conference*, 1992.

[17] J. Savir and W. H. McAnney. Random Pattern Testability of Delay Faults. In *Proc. IEEE Int. Test Conf.*, pages 263–273, 1986.

[18] F. F. Sellers, Jr., M. Y. Hsiao, and L. W. Bearnson. Analyzing Errors with the Boolean Difference. *IEEE Trans. on Computers*, C-17(7):676–683, July 1968.

[19] S. Srinivasan, G. Swaminathan, J. H. Aylor, and M. R. Mercer. Combinational Circuit ATPG Using Binary Decision Diagram. In *IEEE VLSI Test Symposium*, pages 251–258, 1993.

[20] T. Stanion and D. Bhattacharya. TSUNAMI: A Path Oriented Scheme for Algebraic Test Generation. In *Proc. IEEE Int. Conf. on Fault-Tolerant Computing*, pages 36–43, 1991.

[21] P. R. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Combinational Test Generation using Satisfiability. Technical Report UCB/ERL M92/112, Univ. of California, Berkeley, 1992.