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

Chih-Ang Chen and Sandeep K. Gupta

CENG 96-07

Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, California 90089-2562
(213) 740-2251

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 Boolean satisfiability based formulation to generate robust tests for path delay faults in combinational circuits. Conditions to detect a target path delay fault are represented by a Boolean formula. Unlike the technique described in [30], 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.

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

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 its primary inputs such that (1) the fault is excited, and (2) the fault effect is propagated to its primary output(s). This problem can be viewed as a search problem of finding a point in the space of all possible input vectors that is a valid test for the fault. The ATPG problem is known to be *NP-complete* [16], which means that no polynomial-time algorithm is known to solve the problem. However, in the past two decades, tremendous progress has been made on the design of efficient ATPG algorithms. These techniques can be broadly categorized as *structural*, *algebraic*, and *satisfiability-based*.

Structural algorithms directly analyze gate-level description of a circuit and implicitly enumerate all possible input combinations to find test patterns. The search can be accelerated by efficient implication engines and intelligent selection criteria. Some well-known ATPG algorithms for single stuck-at faults are PODEM [17], FAN [14], SOCRATES [33], TOPS [18], etc. Extensions of these techniques to generate two-pattern tests for delay faults are also well studied [13, 22]. Efficiency of these approaches normally depends on "bags of tricks" to avoid entering the *non-solution area* [7].

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. Based on *Boolean difference* [34], early algebraic techniques were not practical due to their high computational complexity. Recent developments in efficient circuit representation via *binary decision diagrams (BDDs)* [5] have inspired many algebraic ATPG algorithms for stuck-at faults [15, 35, 36] and delay faults [1]. The memory requirements and computation time of these techniques are comparable (in some cases, superior) to those of the structural techniques. 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 approach proposed in [20] translates the test generation problem into a formula in *conjunctive normal form (CNF)* [11]. Any input combination which makes the CNF formula evaluate to 1, i.e. any assignment which *satisfies* the formula, is a test for the fault. This approach is not strictly algebraic because no algebraic manipulations are employed but a branch-and-bound strategy, similar to the one used in structural algorithms, is used to find input combinations that satisfy the formula. However, it is not a structural algorithm because all operations are performed on the formula in CNF instead of the gate-level

circuit description. The performance of an efficient implementation of this technique [37] compares favorably to the best known structural algorithms. Further improvements have been reported in [7, 8] by calculating global signal dependencies using *transitive closure.*

The method described in [20, 37] only deals with single stuck-at faults in combinational circuits. Extension of ATPG based on satisfiability to find two-pattern tests for path delay faults has been proposed in [30]. It is proven that any given path delay fault in a circuit is robustly testable 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 given target 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 [37]. In this approach, the circuit modification and CNF formula extraction must be repeated for each path delay fault. Typically, 70% to 80% of the overall time to generate tests for stuck-at faults is spent on CNF formula extraction [37]. In test generation for single stuck-at faults, the time spent for formula extraction is compensated by a faster satisfiability solver because a simpler branch-and-bound algorithm can be designed for the satisfiability problem. A test generator for path delay faults, however, must consider a large number of paths. Due to this fact, a large amount of time is spent on CNF formula extraction in the approach presented in [30], thereby adversely affecting the overall test generation time for circuits with a large number of paths.

This paper describes a new satisfiability based algorithm to generate tests for path delay faults in combinational circuits. The proposed technique directly formulates the ATPG problem for path delay faults as a satisfiability problem based on the value system proposed in [21]. Unlike other techniques [20, 30], the proposed technique extracts the CNF formula only once for each cone. All path delay faults in the cone use the same CNF formula merely by making appropriate on-path and off-path value assignments before solving the CNF formula to find a two-pattern test for the target fault. 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 (cones).

Another advantage of the proposed technique is the flexibility in handling different types of tests for path delay faults, such as *hazard-free robust*, *robust*, and *non-robust* tests [27, 28]. In structural algorithms, different value systems and implication procedures are often required to generate different types of tests for a path delay fault. In the technique described in [30] (though originally proposed to generate robust tests for path delay faults,

the technique can be extended to generate other types of tests as well), the circuit must be modified and the CNF formula extracted for each type of test for a given target path delay fault. In the proposed technique, however, the same CNF formula extracted for a cone can be used to generate different types of tests for all path delay faults in the cone. Hence, if a hazard-free robust or a robust test does not exist for the target path delay fault, the proposed technique can easily search for a non-robust test without any overhead of extracting a new CNF formula.

The paper is organized as follows. Section 2 describes the basic terminology used in this paper. Formulations of the ATPG problem as a Boolean satisfiability problem for stuck-at and path delay faults are presented in Sections 3 and 4, respectively. Experimental results are given in Section 5. Some extensions and applications are discussed in Section 6. Finally, conclusions are presented in Section 7.

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 *formula* is composed of parentheses, literals, and Boolean operations (e.g. NOT, AND, OR, NAND, NOR, etc.). 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. Different fault models require different CNF formula extractors, but can use identical satisfiability solvers (though some heuristics may be more efficient for a specific fault model). This section describes CNF

Table 1: CNF formulas for basic gates

Gate	CNF formula
NOT	$(a+y)(\overline{a}+\overline{y})$
AND	$(a+\overline{y})(b+\overline{y})(\overline{a}+\overline{b}+y)$
NAND	$(a+y)(b+y)(\overline{a}+\overline{b}+\overline{y})$
OR	$(\overline{a}+y)(\overline{b}+y)(a+b+\overline{y})$
NOR	$(\overline{a}+\overline{y})(\overline{b}+\overline{y})(a+b+y)$

formula extraction for single stuck-at faults. Much of the description is due to [20, 37] and is included here to delineate later discussion on test generation for path delay faults.

3.1 Normal Circuit

First, consider a 2-input AND gate represented by the equation

$$y = ab. \tag{1}$$

For clarity, the above representation is said to be in *equation form*. Alternatively, the equation can be written as

$$ab \bigoplus y = 0 \tag{2}$$

$$\overline{a}y + \overline{b}y + ab\overline{y} = 0 \tag{3}$$

$$(a+\overline{y})(b+\overline{y})(\overline{a}+\overline{b}+y) = 1. \tag{4}$$

Now the left-hand side of the equation is in CNF. With a slight abuse of terminology, we say the equation is in CNF. Note that only the values in the truth table of an AND gate can satisfy the CNF formula. The CNF formulas for basic gates are summarized in Table 1. The formulation can be easily extended to a basic gate with multiple inputs.

A Boolean network consists of gates interconnected by wires with possible fanout stems. The network can be represented in CNF by simply concatenating the CNF formula for each individual gate together. Consider the circuit shown in Figure 1. The output function y can be written in equation form as

$$y = \overline{abc + \overline{c}}. \tag{5}$$

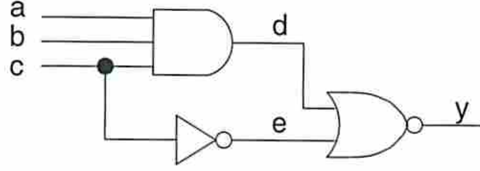

Figure 1: Example circuit

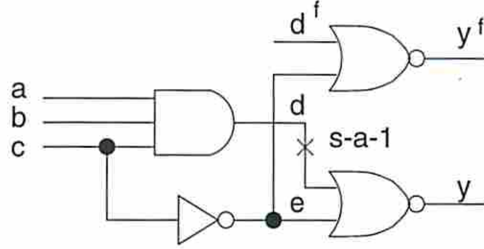

Figure 2: Example circuit with fault d *s-a-1*

The function y can also be written in CNF as

$$\mathcal{C}_g = \underbrace{(a+\overline{d})(b+\overline{d})(c+\overline{d})(\overline{a}+\overline{b}+\overline{c}+d)}_{\text{AND}}\underbrace{(c+e)(\overline{c}+\overline{e})}_{\text{NOT}}\underbrace{(\overline{d}+\overline{y})(\overline{e}+\overline{y})(d+e+y)}_{\text{NOR}} = 1. \quad (6)$$

3.2 Faulty Circuit

The above description only provides the formulation for a circuit's normal operation. When the circuit is faulty, circuit lines in the transitive fanout of the fault site may take values that are different from the corresponding values in the good circuit. 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. As an example, the circuit shown in Figure 1 is shown in Figure 2 with a fault d stuck-at-1. The CNF formula for the duplicate circuit is given by

$$\mathcal{C}_f = (\overline{d}^f + \overline{y}^f)(\overline{e} + \overline{y}^f)(d^f + e + y^f) = 1. \quad (7)$$

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

3.3 Active Clauses

To derive a test for a target fault, it is insufficient to concatenate the CNF formulas for the normal and faulty circuits and to find a satisfying assignment for the resulting CNF formula. In fact, if a satisfying assignment can be found for the good circuit, it will always satisfy the faulty circuit — by equating the normal and faulty variables. If a fault is detectable, there must exist at least one path from the fault site to a primary output of the circuit such that every line l along the path has different values in the normal and faulty circuits (i.e. the good value l and the faulty value l^f are different). 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 path of fault propagation (in our terminology, l is active), then the good value l and the faulty value l^f are different. In other words,

$$l^a \Rightarrow (l \neq l^f) \tag{8}$$

$$(\overline{l^a} + l + l^f)(\overline{l}^a + \overline{l} + \overline{l}^f) = 1. \tag{9}$$

For a fanout stem s in the fault propagation path, the fault effect must be propagated along one of the branches. This can be formulated by adding the active clause

$$\overline{s}^a + \sum_{i=1}^{|s|} s_i^a, \tag{10}$$

where s_i is the i-th fanout branches and $|s|$ is the number of fanout branches. The active clauses for the faulty circuit in Figure 2 are given by

$$C_a = (\overline{d}^a + d + d^f)(\overline{d}^a + \overline{d} + \overline{d}^f)(\overline{y}^a + y + y^f)(\overline{y}^a + \overline{y} + \overline{y}^f) = 1. \tag{11}$$

3.4 Fault Detection

To detect a fault, the fault site must have different values in the good and faulty circuit. Additionally, the fault effect must be propagated to at least one of the primary outputs. These *necessary assignments* can be accounted for by adding additional clauses to the CNF formula. Consider the fault d s-a-1 in the circuit shown in Figure 2. The fault site 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. $y^a = 1$). The clauses

$d^a\overline{d}d^f y^a$ can be added to the CNF formula to represent the necessary conditions in order to detect the fault.

Overall, test generation for the fault d s-a-1 in the example circuit has been translated to the problem of finding a satisfying assignment for the CNF formula given by

$$C_g C_f C_a d^a \overline{d} d^f y^a = 1. \quad (12)$$

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 $\{a = 0, b = x, c = 1\}$.

In the above formulation, a faulty variable and an active variable must be introduced for each line in the transitive fanout of the fault site. If the fault site is close to the primary outputs, then only few additional variables need to be introduced and the number of clauses is small. However, if the fault site is close to the primary inputs, then a large number of additional variables and clauses need to be introduced. In other words, a stuck-at fault near the primary inputs will require 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.

The technique proposed in [30] for generation of a robust test for a path delay fault uses the above formulation to find a test for a stuck-at fault in the modified circuit that is *equivalent* to the target path delay fault. The equivalent stuck-at fault is located on the *I-edge* of the target path P, which is defined as the on-path input to the first non-inverter gate on P [30]. (The I-edge is either a primary input or the output of an inverter fed by a primary input.) Since the equivalent stuck-at fault is close to a primary input, the CNF formula for the equivalent stuck-at fault normally has a large number of extra variables and clauses leading to high memory requirement and, perhaps, high run time. In the following section, we will show how faulty and active variables can be eliminated in the proposed formulation of test generation 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 to represent signal values is different.

2. The fault effect propagation path of the target fault is known.

3. A two pattern test for a path delay fault can be found by finding a primary input assignment that satisfies all the desired on-path and off-path constraints.

Due to the last two differences stated above, 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

Structural test generation algorithms for stuck-at faults normally use a 5-valued logic system consisting of $\{0, 1, x, D, \overline{D}\}$. The logic systems for delay fault testing are more complicated due to the need to span two time frames, corresponding to the two pattern tests. Many logic systems, which range from 5-valued to 23-valued, have been proposed in the literature [2, 6, 13, 21, 22, 24, 25]. A complicated logic system has powerful implication capability and can often speed up computation by reducing the number of backtracks. This is at the cost of higher memory requirement and more complex implementation.

Theoretically, it is possible to use any logic system in the CNF formulation described in the following section. However, the number of clauses in the CNF formula grows exponentially with the number of variables used to represent the values. To minimize the number of clauses, the *7-valued logic system* $\mathcal{L}_7$ proposed in [21] is used in the proposed formulation. According to [13], $\mathcal{L}_7$ 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 2.

A simple encoding scheme is to use two **ternary** variables s and v to represent the first and second element of each value in $\mathcal{L}_7$, respectively. A possible encoding scheme is shown in Table 3. 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. It is counter-intuitive to encode a 7-value logic system with only 2 Boolean variables. But note that any variable that does not have a determined value (i.e. 0

Table 2: 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 3: 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

or 1) in an assignment has the value x (don't-care). Hence, the variables are indeed ternary and the two tuples (l^s, l^v) can be used to represent the seven values.

4.2 Normal Circuit

Consider a 2-input AND gate in equation form given by

$$y = ab. \tag{13}$$

The two-tuples (a^s, a^v), (b^s, b^v), and (y^s, y^v) are used to represent the codes on the lines a, b and y, respectively. Based on the implication table in Table 2 and the encoding in Table 3, the variables y^s and y^v can be represented by

$$y^s = a^s b^s + a^s \overline{a}^v + b^s \overline{b}^v \tag{14}$$

$$y^v = a^v b^v. \tag{15}$$

Equivalently, the two equations can be written as

$$(a^s b^s + a^s \overline{a}^v + b^s \overline{b}^v) \bigoplus y^s = 0 \tag{16}$$

$$a^v b^v \bigoplus y^v = 0. \tag{17}$$

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

$$\mathcal{E}_{AND}(a,b,c) = (\overline{a}^s + \overline{b}^s + y^s)(\overline{a}^s + a^v + y^s)(\overline{b}^s + b^v + y^s)(a^s + b^s + \overline{y}^s)$$
$$(a^s + \overline{b}^v + \overline{y}^s)(\overline{a}^v + b^s + \overline{y}^s)(a^v + \overline{y}^v)(b^v + \overline{y}^v)(\overline{a}^v + \overline{b}^v + y^v) = 1. \qquad (18)$$

Note that only the values in the truth table of an AND gate, where the output y has a determined binary value (0 or 1), 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.

The formulation for an n-input gate is more complicated. An n-input AND gate is represented by the Boolean equation

$$x_1 x_2 \cdots x_{n-1} x_n = y. \qquad (19)$$

The output variables y^s and y^v can be represented by

$$y^s = \prod_{i=1}^{n} x_i^s + \sum_{i=1}^{n} x_i^s \overline{x}_i^v \qquad (20)$$

$$y^v = \prod_{i=1}^{n} x_i^v. \qquad (21)$$

The variable y^s belongs to the class of functions that have polynomial number of minterms in their true form but have exponential number of minterms when complemented [3].

In the proposed technique, an alternative formulation is used to reduce the number of clauses. An n-input basic gate can be decomposed into $(n-1)$ 2-input gates as shown in Figure 3. As proven in [19], such a decomposition neither changes the number of paths in the circuit nor does it affect the detectability of any path delay fault. The CNF formula for the n-input gate can now be obtained by concatenating the CNF formula for its constituent 2-input gates. For example, the CNF formula for a 3-input NAND gate can be obtained by concatenating the CNF formulas for a 2-input AND gate and a 2-input NAND gate. The CNF representation of each n-input basic gates contains $9(n-1)$ clauses, of which $7(n-1)$ clauses have 3 literals and $2(n-1)$ clauses have 2 literals.

For a single output circuit, the CNF formula of 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 proven in the following theorem.

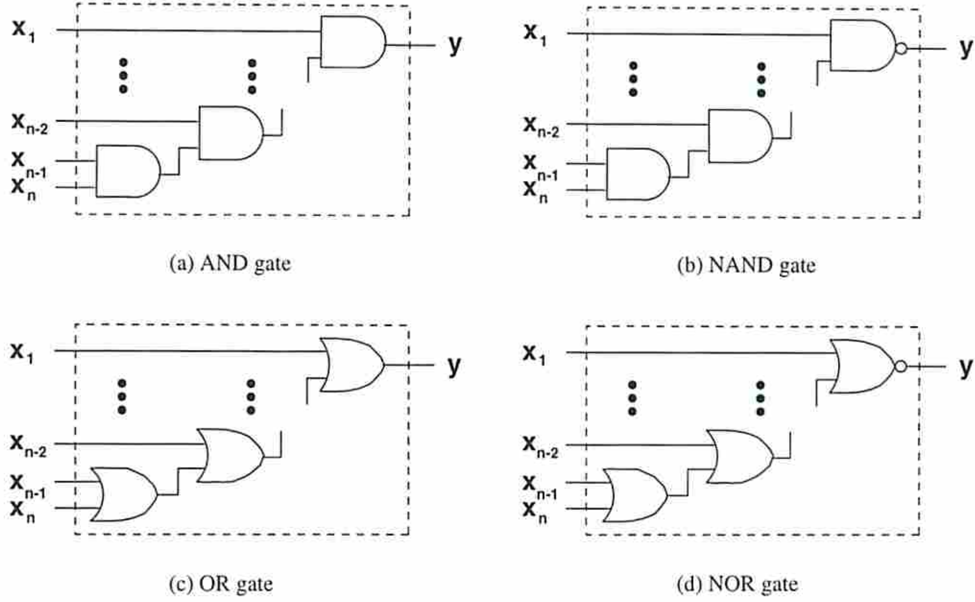

Figure 3: Decomposition of n-input gates

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.*

Proof: The number of clauses for the circuit is given by $4N_1 + \sum_{i=2}^{n} 9(N_i - 1) = 9(N - n + 1) - 5N_1$. As described earlier, the number of clauses with 3 literals is given by $\sum_{i=2}^{n} 7(N_i - 1) = 7(N - n + 1) - 7N_1)$. The number of clauses with 2 literals is given by $4N_1 + \sum_{i=2}^{n} 2(N_i - 1) = 2(N - n + 1) + 2N_1)$. ■

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 remain polynomial in terms of the number of gates even for a circuit with multiple outputs.

4.3 Fault Excitation

In the CNF formulation for stuck-at faults, faulty variables and active variables are introduced for lines in the transitive fanout of the fault site to represent possible fault propagation paths. Since the fault site and the fault propagation paths are different for each stuck-at fault, the clause extraction must be repeated for each fault. For a path delay fault, however, the propagation path is known. A two-pattern test can be generated for a delay fault by satisfying the CNF formula for the circuit subject to constraints on the values on the on-path and off-path inputs. The clauses extracted for the normal circuit are sufficient to model the faults and no faulty clauses or active clauses are required.

The formula extraction in the proposed technique includes two portions: (1) the extraction of the clauses for gates in a circuit cone; and (2) the assignment of values to the literals corresponding to the necessary conditions to detect a target path delay fault. As proven in Theorem 1, the complexity of the first portion is proportional to the number of gates in the circuit. However, the complexity of the second portion is proportional to the number of paths, which may grow exponentially in terms of the number of gates in the circuit. By carefully implementing the path enumeration algorithm, however, 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.

The fact that the clause extraction does not have to be repeated for each path delay fault is a very important feature of the proposed formulation. For all path delay faults in the same cone, the CNF formula is the same. Only the off-path input constraints are different. The CNF formula can be extracted once for each cone to generate the path delay faults in the cone. As noted in [37], the clause extraction accounts for a significant portion of the computation time in ATPG based on satisfiability. The savings in clause extraction can tremendously improve the efficiency of the proposed ATPG for path delay faults.

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) [31] (called *single-path propagating hazard-free robust tests* in [28]), robust tests, and non-robust tests. (Extensions to other types of tests for path delay faults [9, 29] are possible and are currently under investigation.) These tests differ on the constraints on the off-path values as shown in Table 4. In the proposed technique, these off-path input con-

Table 4: Constraints on off-path values

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

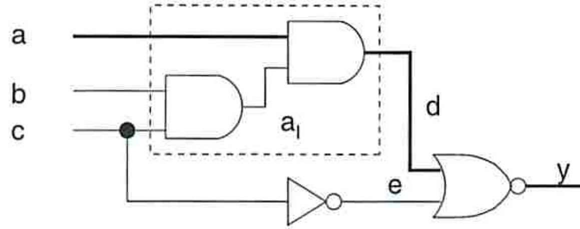

Figure 4: Decomposition of example circuit

straints can be uniformly represented by fixing the variables for the off-path inputs before solving the CNF formula to find a two-pattern test.

As an example, consider the path in the circuit shown in Figure 4. The CNF formula for the circuit is given by

$$\mathcal{E}_{AND}(b, c, a_I)\mathcal{E}_{AND}(a, a_I, d)\mathcal{E}_{NOT}(c, e)\mathcal{E}_{NOR}(d, e, y) = 1. \tag{22}$$

A robust test for the slow-to-rise delay fault on the 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 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 clause $\overline{a}^s a^v \overline{d}^s d^v \overline{y}^s \overline{y}^v a_I^v b^v c^v e^s \overline{e}^v$ to Eq. 22. 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. 22, together with additional clause 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. 22. Since no satisfying assignment can be found, no robust test exists for the path delay fault. However, by adding the clause $\overline{c}^s c^v \overline{a}_I^s a_I^v \overline{d}^s d^v \overline{y}^s y^v a^v b^v \overline{e}^v$ to Eq. 22, 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 type of test 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 to compare different search strategies that determine variable order for branching [37]. In [8], the *transitive closure* of the *implication graph* derived from the 2-clauses in a CNF formula is used to determine global signal dependencies. All 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 [4] circuits. The experimental results presented in the following are obtained using a HP-710 with 32 Mbytes of memory. The program consists of a front-end clause extractor that extracts the CNF formula from the gate-level description of the given circuit. 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 by our current implementation. They only differ in the off-path constraints specified for fault activation. The satisfiability solver is based on the implementation in [37], originally integrated in an ATPG for single stuck-at faults. As described earlier, this satisfiability solver can be used to find a satisfying assignment for the CNF formula extracted for a path delay fault.

The experimental results for test generation of robust tests are shown in Table 5. Column 1 gives the circuit name (only the combinational parts of these circuits are used). Columns 2–5 show the number of path delay faults that are detected, untestable, and aborted, followed by the total number of path delay faults in the circuit. In all these

Table 5: Results of test generation: robust tests

Ckt	Path delay faults				CPU time (sec)			CNF/total
	det.	untst.	ab.	total	CNF	SAT	total	(%)
s27	50	6	0	56	0.02	0.07	0.09	22.22
s208	290	0	0	290	0.20	1.00	1.20	16.66
s298	343	119	0	462	0.25	1.44	1.69	14.79
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
s526n	695	121	0	816	0.51	3.31	3.82	13.36
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

experiments, each path delay fault is individually targeted and no fault simulation or *fault dropping* is performed. Note that the proposed technique is able to find a robust test or to prove that no robust test exists for every path delay fault in the circuits we experimented on. The remaining columns show the time spent on formula extraction (CNF) and solving (SAT), followed by the total time required for test generation. Finally, the percentage of the total test generation time that was spent on formula extraction is presented. As shown in the table, formula extraction (CNF) accounts for 10% of the total computation time on the average, compared to 63% reported in [30]. The formula extraction time for the proposed technique is significantly lower 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 5. The number of paths in `s1423` is about 12.6 times the number of paths 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 are shown in Tables 6 and 7, respectively. As shown in the tables, the number of untestable faults increases (decreases) as more (less) restricted tests are generated for the path delay faults. The total run time as well as the percentage of the total run time that is spent on formula extraction are both about the same for different types of tests for path delay faults.

6 Discussion

CNF formulation has the advantages of simplicity and uniformity. The CNF formula can be obtained by merely concatenating CNF formulas for individual circuit components. As shown in the formulation for stuck-at faults, faulty and active variables need to be introduced to represent fault propagation path in the faulty circuit. On the other hand, for path delay faults, the CNF formulation does not require any such additional variables. Hence, this formulation is very well suited for path delay faults.

For the technique described in [30], the time spent on formula extraction is propor-

Table 6: Results of test generation: RDTPs

Ckt	Path delay faults				CPU time (sec)			CNF/total
	det.	untst.	ab.	total	CNF	SAT	total	(%)
s27	48	8	0	56	0.01	0.07	0.08	12.50
s208	290	0	0	290	0.16	1.08	1.24	12.91
s298	332	130	0	462	0.23	1.49	1.72	13.37
s344	578	132	0	710	0.40	3.33	3.73	10.73
s349	576	154	0	730	0.39	3.44	3.83	10.18
s382	632	168	0	800	0.31	2.64	2.95	10.51
s386	412	2	0	414	0.22	1.68	1.90	11.58
s400	624	272	0	896	0.44	2.71	3.51	13.96
s420	738	0	0	738	0.30	5.23	5.53	5.42
s444	504	566	0	1070	0.47	3.12	3.59	13.09
s510	720	18	0	738	0.46	6.31	6.77	6.79
s526	680	140	0	820	0.49	3.37	3.86	12.70
s526n	682	134	0	816	0.55	3.32	3.87	14.21
s641	1576	1912	0	3488	2.77	34.47	37.24	7.44
s713	400	43224	0	43624	24.53	105.25	129.78	18.90
s820	970	14	0	984	0.72	8.94	9.66	7.46
s832	962	50	0	1012	0.78	9.54	10.32	7.56
s838	2018	0	0	2018	1.75	31.95	33.70	5.19
s953	2292	20	0	2312	1.94	25.91	27.85	6.97
s1196	3088	3108	0	6196	8.49	154.11	162.60	5.22
s1238	2852	4266	0	7118	10.71	201.10	211.81	5.06
s1423	24458	64994	0	89452	79.74	1310.44	1390.18	5.73
s1488	1832	92	0	1924	1.50	20.34	21.84	6.87
s1494	1826	126	0	1952	1.38	20.82	22.20	6.22
s5378	17254	9830	0	27084	42.88	511.93	554.81	7.73
s9234	14696	475012	0	489708	1689.99	7952.41	9642.40	17.52

Table 7: Results of test generation: non-robust tests

Ckt	Path delay faults				CPU time (sec)			CNF/total
	det.	untst.	ab.	total	CNF	SAT	total	(%)
s27	50	6	0	56	0.04	0.05	0.09	44.44
s208	290	0	0	290	0.09	1.08	1.17	7.69
s298	364	98	0	462	0.22	1.46	1.68	13.10
s344	654	56	0	710	0.33	3.07	3.40	9.71
s349	656	74	0	730	0.33	3.09	3.42	9.65
s382	734	66	0	800	0.33	2.57	2.90	11.37
s386	414	0	0	414	0.26	1.58	1.84	14.13
s400	753	143	0	896	0.44	2.79	3.24	13.58
s420	738	0	0	738	0.39	5.00	5.39	7.23
s444	813	257	0	1070	0.49	3.26	3.75	13.06
s510	738	0	0	738	0.48	6.23	6.71	7.16
s526	720	100	0	820	0.46	3.32	3.78	12.17
s526n	718	98	0	816	0.40	3.37	3.77	10.61
s641	2270	1218	0	3488	2.99	35.00	37.99	7.87
s713	4922	38702	0	43624	26.76	202.13	228.89	11.69
s820	984	0	0	984	0.73	8.85	9.58	7.62
s832	996	16	0	1012	0.73	9.57	10.30	7.09
s838	2018	0	0	2018	2.02	31.23	33.24	6.05
s953	2312	0	0	2312	1.85	24.84	26.69	6.93
s1196	3759	2437	0	6196	8.72	149.88	158.60	5.50
s1238	3684	3434	0	7118	11.08	182.70	193.78	5.72
s1423	45198	44254	0	89452	78.79	1364.71	1437.50	5.45
s1488	1916	8	0	1924	1.53	20.17	21.71	7.05
s1494	1927	25	0	1952	1.49	20.57	22.06	6.76
s5378	21928	5156	0	27084	43.40	583.52	626.92	6.73
s9234	59854	429854	0	489708	1836.20	8300.58	10136.78	18.11

tional to the number of paths in the circuit. Therefore, the time spent on formula extraction remains a large portion of the total computation time. In the proposed technique, the time spent on formula extraction is proportional to the number of outputs of the circuit. As demonstrated above, the time spent on the formula extraction is very small even for circuits with a large number of paths.

Structural ATPG algorithms often explore the circuit topology to discover additional unique assignments on some of the lines. These unique assignments can be represented by fixing the value of the corresponding variable. Another method often used in structural algorithms is *learning* [33], which explores the contrapositive of a logic consequence that can not be inferred by direct implications. These logic inferences (sometimes called *global* or *non-local implications*) [37] can be easily represented by adding clauses to the CNF formula. Hence, the CNF formulation can take advantage of many techniques developed to accelerate structural ATPG algorithms.

In the above discussion, no constraint is imposed on the variables. Any input assignment that satisfies the CNF formula is a test for the target fault. In some ATPG applications, the variables may have correlations which can be represented by imposing extra constraints on these variables. Such ATPG variations are called *constrained ATPG*. In the following, we will show how constrained ATPG can be uniformly formulated after the ATPG problem is translated into a Boolean satisfiability problem.

In the following discussion, each two-pattern test $V = (V_1, V_2)$ for a sequential circuit consists of the primary input part I and the secondary input part D, i.e. $V = (V_1, V_2) = (I_1|D_1, I_2|D_2)$. Extra clauses can be introduced into the CNF formulas to represent the constraints imposed by the *design-for-testability (DFT)* technique employed.

Scan Shifting A standard full scan design does not allow the application of arbitrary two-pattern tests due to shift correlations. Determining the order of the flip-flops in a standard scan chain to enhance the delay fault coverage (called the *scan-chain ordering problem*) has been studied [10, 23, 26]. In scan shifting [10] (called *skewed-load transition test* in [26]), assuming that the scan chain order is known, the final vector D_2 of a two-pattern test is a one bit shift of its initial vector D_1. The scan chain is configured in the test mode and the test clock is used to scan-in the initial vector and trigger the final vector.

Given a scan chain order, two-pattern tests that can be applied via scan shifting can be generated by introducing additional clauses into the CNF formulae. Consider two consecutive flip-flops d_1 and d_2 in a scan chain. The final value of d_2 must equal to the

initial value of d_1. In the proposed CNF formulation, this constraint can be represented by adding the following clauses

$$(\overline{d}_1^s + d_1^v + \overline{d}_2^v)(\overline{d}_1^s + \overline{d}_1^v + d_2^v)(d_1^s + d_1^v + d_2^v)(d_1^s + \overline{d}_1^v + \overline{d}_2^v) = 1. \tag{23}$$

Functional Justification In *functional justification* [10], the initial vector of a two-pattern test is scanned-in with the flip-flops configured in test mode as a scan chain. The second vector is the obtained by the implication of the initial vector via the next state (functional) logic which is captured by configuring the flip-flops in their normal modes and activating the circuit clock. As noted in [10, 32], combining both methods (functional justification and scan shifting) can often achieve higher delay fault coverage than each individual method alone.

A two-pattern test $V = (V_1, V_2)$ that can be applied in the functional justification mode must satisfy the property $D_2 = f(V_1)$, where f is the next state function for the circuit. Let d_i and δ_i be the input (secondary output) and output (secondary input) of the i-th flip-flop, respectively. In functional justification, the values in the flip-flops can be correlated by the following equation

$$(\overline{d}_i^s + d_i^v + \overline{\delta}_i^v)(\overline{d}_i^s + \overline{d}_i^v + \delta_i^v)(d_i^s + d_i^v + \delta_i^v)(d_i^s + \overline{d}_i^v + \overline{\delta}_i^v) = 1. \tag{24}$$

Clock Grouping *Clock grouping* was introduced in [12] to further enhance delay fault coverage by separating the flip-flops into *clocked groups* and allowing extra freedom to individually clock each group. The technique can be combined with functional justification and/or scan shifting to obtain very high delay fault coverage not possible with each individual method. Experimental results show that typically two groups are sufficient to achieve acceptable delay fault coverage.

In clock grouping, the final vector can be obtained by configuring the flip-flops in functional justification or scan shifting mode. In addition, flip-flops in any group may be either clocked or allowed to hold their V_1 values. In CNF formulation, a flip-flop d operated in the hold mode can be represented by setting d^s to 1 during ATPG.

7 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 [30]. Unlike their technique which extracts the CNF formula for each path delay fault, the proposed formulation needs to extract 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 algorithm has the advantage of simplicity and uniformity, and is very well suited for path delay faults because faulty and active variables are not required. The formula extraction time is proportional to the number of circuit outputs and is typically small even for circuits with a large number of paths. As described earlier, the proposed formulation can also be used to solve many *design-for-testability* problems.

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] R. K. Brayton, G. D. Hachtel, C. McMullen, and A. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis.* Kluwer Academic Publishers, Boston, MA, 1984.

[4] 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.

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

[6] T. J. Chakraborty, V. D. Agrawal, and M. L. Bushnell. Delay Fault Models and Test Generation for Random Logic Sequential Circuits. In *Proc. IEEE-ACM Design Automation Conference*, pages 165–172, 1992.

[7] S. T. Chakradhar and V. D. Agrawal. A Transitive Closure Based Algorithm for Test Generation. In *Proc. IEEE-ACM Design Automation Conference*, pages 353–358, 1991.

[8] 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.

[9] K.-T. Cheng and H.-C. Chen. Delay Testing for Non-Robust Untestable Circuits. In *Proc. IEEE Int. Test Conf.*, pages 954–961, 1993.

[10] K.-T. Cheng, S. Devadas, and K. Keutzer. A Partial Enhanced-Scan Approach to Robust Delay-Fault Test Generation for Sequential Circuits. In *Proc. IEEE Int. Test Conf.*, pages 403–410, 1991.

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

[12] W.-C. Fang and S. K. Gupta. Clock Grouping: A Low Cost DFT Methodology for Delay Fault Testing. In *Proc. IEEE-ACM Design Automation Conference*, pages 94–99, 1994.

[13] 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.

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

[15] R. K. Gaede, M. R. Mercer, K. M. Butler, and D. E. Ross. CATAPULT: Concurrent Automatic Testing Allowing Parallelization and Using Limited Topology. In *Proc. IEEE-ACM Design Automation Conference*, pages 597–600, 1988.

[16] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman, San Francisco, CA, 1979.

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

[18] T. Kirkland and M. R. Mercer. A Topological Search Algorithm for ATPG. In *Proc. IEEE-ACM Design Automation Conference*, pages 502–508, 1987.

[19] 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.

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

[21] 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.

[22] 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.

[23] W. Mao and M. D. Ciletti. Reducing Correlation to Improve Coverage of Delay Faults in Scan-Path Design. *IEEE Trans. on CAD*, 13(5):638–646, May 1994.

[24] E. S. Park and M. R. Mercer. Robust and Non-Robust Tests for Path Delay Faults in a Combinational Circuit. In *Proc. IEEE Int. Test Conf.*, pages 1027–1034, 1987.

[25] E. S. Park and M. R. Mercer. An Efficient Delay Test Generation System for Combinational Logic Circuits. In *Proc. IEEE-ACM Design Automation Conference*, pages 522–528, 1990.

[26] S. Patil and J. Savir. Skewed-Load Transition Test: Part II, Coverage. In *Proc. IEEE Int. Test Conf.*, pages 714–722, 1992.

[27] A. K. Pramanick and S. M. Reddy. On the Detection of Delay Faults. In *Proc. IEEE Int. Test Conf.*, pages 845–856, 1988.

[28] 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.

[29] A. K. Pramanick and S. M. Reddy. On Multiple Path Propagating Tests for Path Delay Faults. In *Proc. IEEE Int. Test Conf.*, pages 393–402, 1991.

[30] 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.

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

[32] J. Savir and S. Patil. Broad-Side Delay Test. *IEEE Trans. on CAD*, 13(8):1057–1064, Aug. 1994.

[33] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: A Highly Efficient Automatic Test Pattern Generation System. In *Proc. IEEE Int. Test Conf.*, pages 1016–1026, 1987.

[34] 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.

[35] 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.

[36] 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.

[37] 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.