A Method of Delay Fault Test Generation

C. Thomas Glover and M. Ray Mercer

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, Texas 78712-1084
(512) 471-1804

ABSTRACT

It has been observed that random testing for delay faults can result in test sets of excessive length and high simulation costs. Consequently, we propose an efficient deterministic method of delay fault test generation. For most common circuits, our proposed technique has a time complexity which is polynomial in the size of the circuit, as opposed to existing deterministic methods which, for nearly all circuits, are exponential. We define a type of transition path, the fully transitional path, FTP, and demonstrate that it has several useful properties. An FTP can be created by applying a vector pair derived from a stuck-at test for a primary input, a technique introduced in [1]. We extend this method by using an alternate representation for switching functions, the binary decision diagram, to generate graphs representing stuck-at tests. The concept of free variables is defined as a tool for deriving several FTPs from one stuck-at test. Preliminary results are presented which indicate that our method provides a higher robust delay fault coverage than psuedorandom patterns at less than one-fifth the cost. Also, since vector pairs cannot be applied to combinational circuits using standard scan design, a simple scannable latch is introduced to facilitate this task.

INTRODUCTION

This work concerns automatic test pattern generation for delay faults in combinational circuits. A delay fault can be viewed as a condition which causes the propagation delay of a signal through the combinational part of a machine to exceed the clocking interval between the primary inputs and latches issuing the signal and the primary outputs and latches receiving the signal. Test generation systems generally operate in two phases. The first phase typically consists of random vectors to detect the "easy" faults. The few "difficult" faults which remain undetected after the first phase are tested by the deterministic second phase. It has been observed that random testing for delay faults can result in test sets of excessive length and high simulation costs [2], [3], [4]. Consequently, we propose an efficient deterministic method of delay fault test generation.

A transition path is a path which is sensitive to a transition fault (where a transition fault is a fault which changes the delay of a transition) [5]. The Fully Transitional Path, FTP, can be informally defined as a transition path where a transition occurs at each link along the path. One way to create such a path is to apply a vector pair (v_1, v_2) such that v_2 is equal to a stuck-at test for a primary input, x_i, and v_1 is equal to v_2 with x_i complemented. This technique was introduced in [1]. This was the first documented effort to attack the problem of delay fault testing using stuck-at tests. However, this work assumed the existence of a stuck-at test set, while we propose to generate an original set of tests in a more efficient manner. Furthermore, the initial analysis was limited to singly-sensitized paths, while we prove that the salient features of an FTP hold for multi-dimensional paths as well. Moreover, we show that the FTP is: 1) always valid in the presence of multiple delay faults and 2) except in one case which is easy to discern, valid in the presence of hazards.

Since delay and CMOS stuck-open faults are closely related, it is worth noting that stuck-at tests have also been used to detect CMOS stuck-open faults [6], [7]. However, these methods derive tests for one target fault and do not create fully transitional paths. A method of deriving vector pairs which create FTPs was presented in [8]. However, since this method is based on the analysis of random vectors, a low fault coverage is obtained for circuits resistant to random patterns. The recent research on delay testing has focused on creating transition paths using methods based on the D-algorithm or PODEM [9], [10], [11], [5]. However, some of this work does not consider the effects of hazards and multiple delay faults. Moreover, for nearly all circuits, such methods have time complexities which are exponential with respect to the size of the circuit. That is, despite clever ways of bounding the search space, such methods are variations on exhaustive search. On the other hand, for most common circuits, the time complexity of our approach is polynomial in the size of the circuit.

In addition, the work presented assumes a scan design methodology. However, it has been observed that any abitrary vector pair cannot be applied to a combinational circuit using standard scan path design. Therefore, in the first section, we propose a scannable latch which facilitates this task. In the next section we formally define the FTP and prove that every vector pair generated by the proposed method creates an FTP. This is followed by a discussion of three topics related to fault coverage: 1) proof that the FTP is a delay test, 2) the robustness of the FTP, and 3) a drawback of the test generation technique. Next we explain the test generation method and derive a cost function for the technique. Finally, some preliminary results are presented.

PROPOSED LATCH

It has been observed that an input vector pair cannot be applied to a combinational circuit using standard scan path design [11], [4]. Denote the vector pair being applied as (v_1, v_2). One solution would be to add an extra latch to each existing latch to store v_1 while v_2 is shifted in [4]. This proposal is obviously expensive in terms of chip area. Another solution would be to obtain v_2 by shifting v_1 [3]. Since this limits the number of vector pairs which can be applied, the fault coverage of the test set is limited. We propose to alter the latch itself.

An NMOS implementation of our proposed latch is shown in Fig. 1. The master or input latch was created for scannable single latch designs [12]. The output latch is a standard flip-flop. The signal clocking the feedback loop in the master latch, labeled C, is the logical NOR of the A clock and the

* This work was sponsored by the National Science Foundation under Grant MIP 8552537, also a grant from the Microelectronics and Computer Technology Corp. (MCC), and a grant from the International Test Foundation.

TEST signal. We assume that the TEST signal is low during normal operation and high when scanning is in progress.

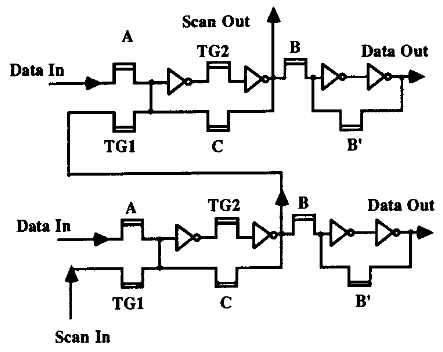

Fig. 1. Scan Path with Proposed Latch

During normal system operation, transistor TG2 is "on" (conducting), TG1 is "off" (nonconducting), TEST is "low", and clocks A and B are active. In test mode, all the transistors are active except the one driven by C. The first input vector, v_1, is scanned in by alternately clocking TG1 and TG2 while the values of A and B are low and TEST is high. After v_1 has been scanned in, B is pulsed so that v_1 is captured in the slave latch and applied to the combinational circuit. Now v_2 is shifted in with TG2 and TG1. Once v_2 has been scanned in, the circuit is tested by pulsing B and then A with the normal clock interval of the system. The result of the test is then scanned out of the master latch by clocking TG2 and TG1.

Of course, any design for testability, no matter how simple, has some cost associated with it. However, the only space-time penalty imposed by the latch is the transistor driven by TG2. The latch does not require any clocks or input signals which would not be present in a standard scan design. Note that the scan path is dynamic rather than static. That is, it contains no feedback loops. Fortunately, if scan-in and scan-out are performed at system clock rates, this is not a problem. This design is only applicable to double latch systems.

THE STUCK-AT FAULT MODEL AND THE FULLY TRANSITIONAL PATH

Under the single fault assumption and in the absence of hazards, the observability requirements of stuck-at and delay faults are equivalent. Consequently, the FTP is defined in terms of the classic D-algebra associated with the stuck-at model [13].

Definition: An FTP begins at some link in the circuit which is tested for both stuck-at-zero and stuck-at-one faults by two input vectors, v_1 and v_2. The path terminates at a primary output. Let $t_1(t_2)$ be some time after $v_1(v_2)$ has been applied. Let $\alpha_1(\alpha_2)$ represent the value of any link along the FTP during $t_1(t_2)$. An FTP exists if and only if the following conditions are satisfied for each link, α, along the path.

1. $\alpha_1, \alpha_2 \in \{D, D'\}$.
2. $\alpha_1 = \alpha_2'$. □

The following theorem states that every vector pair generated by our delay test method creates an FTP.

Theorem 1: If v_2 is a stuck-at test for a primary input, x_i and v_1 is obtained by inverting x_i while holding all other inputs constant, then the vector pair, (v_1, v_2), creates an FTP originating at x_i.

Proof: Let A denote the set of all links along the assumed FTP. Let the set of primary inputs be represented by $X = \{x_1, x_2, ..., x_n\}$, and let B be the set of links along a D-chain of a stuck-at test for x_i which is propagated to a primary output, f. By the definition of an FTP, it suffices to show that the following conditions are met for all $b \in B$:

1. $b_1, b_2 \in \{D, D'\}$
2. $b_1 = b_2'$.

Let s be any link in the combinational circuit. Note that the Boolean difference of s with respect to x_i, ds/dx_i, is independent of x_i. That is, $ds/dx_i = g(x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_n)$. (Recall that $ds/dx_i = s_i(0) \oplus s_i(1)$ (where $s_i(0)$ denotes the function s evaluated with x_i equal to zero)). It follows that ds/dx_i is equal for both t_1 and t_2. That is, $ds_1/dx_{i1} = ds_2/dx_{i2}$. Clearly, $ds/dx_i \in \{\emptyset, 1\}$. Each of these two cases must be considered.

Case 1: $ds/dx_i = \emptyset$. Clearly, if $ds/dx_i = \emptyset$, then s is not logically dependent on x_i, i.e., $s_i(0) = s_i(1)$. Therefore $s \notin B$. Since $x_{j1} = x_{j2}$ $\forall$ $j \mid j \neq i$, it follows that $s_1 = s_2$. (Thus any sensitizing input along the path cannot have a transition applied to it).

Case 2: $ds/dx_i = 1$. Since s is logically dependent on x_i, $x_{i1} \neq x_{i2} \Rightarrow s_1 \neq s_2$. Note that using a stuck-at test for v_2 ($v_2 \in (df/dx_i)$) implies that $s_1, s_2 \in \{D, D'\}$ and $\{s \mid s \in B\} = A$ must be nonempty. □

FAULT COVERAGE

There are two main delay fault models: the gate delay model [5], [3], [8] and the path-fault model [14], [2], [9]. Under the gate delay model the delay fault inhibits a transition at a single node in a circuit. Under the path-fault model a delay fault occurs if the propagation delay along a path in the circuit is greater than the system clock interval. Fault coverage for the gate delay model is computed in the usual way: as the percentage of faults under consideration which are detected. However, under the path-fault model, fault coverage is determined as a percentage of preselected paths which are tested. The gate delay model suffers from the fact that delay faults resulting from the sum of several small incremental delay defects may not be detected. A disadvantage of the path-fault model is the possibility that gate delay faults which do not lie along the set of tested paths will not be detected.

Since our method generates at least one FTP between each primary input-output pair, our method tests many of the longest paths in a circuit. Moreover, by Case 1 in Theorem 1, any sensitizing input along an FTP cannot have a transtion applied to it. Hence, these paths are guaranteed to be tested in the presence of multiple delay faults. Furthermore, we will demonstrate that the paths are also tested in the presence of hazards, except in one case which is straightforward to detect. However, since timing data is not used to select the paths which are tested, the technique does not strictly conform to the path-fault model. In order to obtain data based on the entirety of each circuit and in the absence of precise timing data which would facilitate the selection of "critical paths", the reported delay fault coverages are computed in the usual way and are not path oriented. That is, the results are based on the percentage of gate delay faults under consideration which are detected as opposed to a percentage of preselected paths which are tested.

Penalty

The problem of testing combinational circuits has been shown to be NP-complete [15]. It has not been proven or disproven that NP-completeness implies exponential complexity. Hence, a polynomial time solution to an NP-complete problem has not been found. Consequently, if such a problem is attacked with a method of less-than-exponential complexity, then it is extremely likely that some penalty will be paid. The penalty associated with the proposed approach is that some irredundant fault sites may not be tested. Consider Fig. 2; although the node labeled X can be tested for both a stuck-at one and a stuck-at zero fault, the node does not lie along any FTP passing through the circuit and is not FT-testable. Fortunately, such faults appear to be few in number. This observation is based on the fact that every irredundant fault site in three actual circuits: an ALU, a PLA, and a two-bit full adder is testable by this method. Furthermore, at least 93 percent of the fault sites in C1908 of the ISCAS circuits are FT-testable. Moreover, an example where an irredundant circuit contains a node which is not FT-testable has not been found.

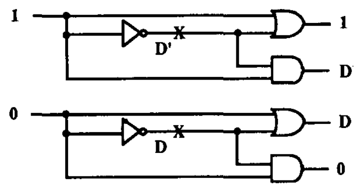

Fig. 2. Non FT-testable Fault Site

The FTP and the gate delay fault

It is important to prove that every vector pair which generates an FTP tests one or more gate delay faults. Let f(X) be the switching function of a primary output of a combinational circuit which contains a link ξ where the primary inputs are represented by the set $X = \{x_1, x_2, ..., x_n\}$. Denote the switching function realized at ξ as $\xi(X)$. The set of vector pairs which test for a delay fault at ξ, $\{(v_1, v_2)\}$, is equal to the intersection of the following:

1. $\{(X_1, X_2) \mid \text{the fault at } \xi \text{ is excited}\}$
2. $\{X_2 \mid df/d\xi = 1\}$ (where $df/d\xi = f_\xi(1) \oplus f_\xi(0)$).

Hence, for a slow-to-rise fault at ξ, $\{v_2\} = \xi(X)(df(X)/d\xi)$. Similarly, for a slow-to-fall fault at ξ, $\{v_2\} = \xi'(X)(df(X)/d\xi)$. The set $\{v_1\}$ consists of those vectors which excite the delay fault by provoking a transition at ξ, i.e., $\xi_1 = \xi_2'$. Thus the set of vector pairs which detects a slow-to-rise fault at ξ is $\{(v_1, v_2) \mid v_1 \in \xi'(X)$ and $v_2 \in \xi(X)(df(X)/d\xi)\}$. Similarly, the set of vector pairs which detects a slow-to-fall fault is $\{(v_1, v_2) \mid v_1 \in \xi(X)$ and $v_2 \in \xi'(X)(df(X)/d\xi)\}$. Moreover, the set $\{v_2\}$ for a slow-to-rise (slow-to-fall) fault at ξ is equal to $T_{\xi 0}$ ($T_{\xi 1}$), the test set for a stuck-at zero (stuck-at one) fault at ξ. At this point, we are prepared to justify the assertion that if a fault site along an FTP is tested for a stuck-at fault, then the fault site is also tested for a delay fault:

Lemma 1: Every link along a robust FTP which is tested for a stuck-at fault by v_2 is tested for the analogous delay fault.

Proof: Let the link under test be z. Since z is tested for a stuck-at fault by v_2, z is observable during t_2. Since z lies along the FTP, $z_1 = z_2'$. Thus the delay fault at z is both excited and observed. □

Robustness of the FTP

Unlike stuck-at faults, satisfying the intersection of the excitation and observability requirements of a fault site is not sufficient for testing delay faults. It is possible for delay tests to be invalidated by hazards or multiple delay faults which set the output of a gate under test to its fault-free value just prior to t_2. Such tests are said to be nonrobust. On the other hand, a test which is valid under arbitrary delays in the circuit (i.e., in the presence of hazards and/or multiple delay faults) is referred to as robust [9],[16].

Definition: An FTP is nonrobust if and only if there is a gate along the path such that the output of the gate can be driven to its fault-free value prior to t_2. □

Hazards in combinational circuits can be modelled using the logic values defined in Fig. 3 [17]. The Cayley table for the logical OR operation is shown in Fig. 4. The symbol # in Fig. 4 denotes a condition which invalidates an FTP. There is only one case where an FTP is nonrobust.

Logic Symbols	Interpretation
0	**steady state 0**
1	**steady state 1**
R	**hazard free transition from 0 to 1**
F	**hazard free transition from 1 to 0**
0*	**static zero logic hazard**
1*	**static one logic hazard**
R*	**dynamic logic hazard**
F*	**dynamic logic hazard**

Fig. 3. Eight Valued Logic for Hazard Analysis

OR	0	1	F	R	0*	1*	F*	R*
0	0	1	F	R	0*	1*	F*	R*
1	1	1	1	1	1	1	1	1
F	F	1	F	1*	F*	1*	F*	1*
R	R	1	1*	R	R*#	1*	1*	R*
0*	0*	1	F*	R*#	0*	1*	F*	R*#
1*	1*	1	1*	1*	1*	1*	1*	1*
F*	F*	1	F*	1*	F*	1*	F*	1*
R*	R*	1	1*	R*	R*#	1*	1*	R*

Fig. 4. Cayley Table for Logical OR

Theorem 2: An FTP is nonrobust (invalidated under arbitrary delays through the circuit) if and only if the following two conditions are satisfied:

1. Two or more transition paths must terminate at a gate which is not fed by a controlling input. (This is the only way a static logic hazard can be created.)

2. A line on which there is a logic hazard must converge with the FTP such that at some point along the path the output of a gate is driven to its fault-free value prior to t_2.

Proof: Since, by Case 1 in Theorem 1, any sensitizing input along an FTP cannot have a transition applied to it, multiple delay faults cannot invalidate an FTP. Thus an FTP can only be invalidated by hazards. We assume that all latches feeding the circuit under test are hazard-free.

$\Rightarrow$: The fact that invalidation implies condition 2 follows from the definition of a nonrobust FTP. It remains to show that hazards can only be created if condition 1 is satisfied. By the table in Fig. 4:

1. The existence of a dynamic logic hazard implies the

existence of a static hazard.

2. The existence of a static logic hazard implies condition 1.

Hence, condition 1 is necessary for FTP invalidation.

$\Leftarrow$: Inspection of Fig. 4 shows that condition 1 is sufficient for the creation of a static hazard. If the hazard is propagated according to Fig. 4 such that it forces a gate along the FTP to its fault-free value prior to t_2, then by definition the path is invalid. □

An example of FTP invalidation is shown in Fig. 5. The two complementary transitions feeding the AND gate satisfy condition 1 of Theorem 2. Consequently, the upper line feeding the OR gate may temporarily take the value of 1. This could cause the fault-free value of the test to be observed for some short interval regardless of whether or not a transition was propagated along the path being tested. Note that the element in Fig. 4 which corresponds to this case, the logical OR of R and 0*, is marked #. A comparison of two types of hazard convergence is shown in Fig. 6. The upper figure depicts a nonrobust FTP. On the other hand, the lower figure shows a case where hazard convergence merely causes the test to be pessimistic, an acceptable condition. Note that the element in Fig. 4 which corresponds to this case, the logical OR of F and 0*, is not marked #.

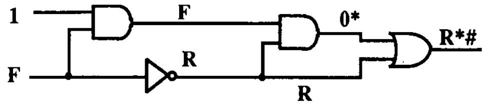

Fig. 5. Invalid FTP

TEST GENERATION

This section concerns the proposed method of test generation. First we discuss a way to derive more than one FTP from a stuck-at test. Next, we summarize the test generation strategy and derive a cost function for the technique. Three concepts must be introduced before we begin: 1) the main data structure, the binary decision diagram, 2) structural dependence, and 3) the test graph. The binary decision diagram, BDD, is an alternative representation for switching functions with many desirable characteristics [18], [19]. Unlike other canonical representations, only one common circuit has been found, an integer multiplier, for which the size of the BDD is an exponential function of the number of inputs. An example BDD, which represents the function **a + bc**, is shown in Fig. 7. A BDD is evaluated by beginning at the root and descending in the graph until a vertex with either of the values **1** or **0** is encountered. Next, assume that the circuit has been partitioned into subcircuits such that each segment feeds one primary output. If x_i feeds the segment rooted by f, then f is structurally dependent on x_i. Note that structural dependence does not imply functional dependence.

Let f be a primary output of the circuit under test such that f is functionally dependent upon primary input x_i. The test graph for x_i with respect to f is simply a BDD which represents the Boolean difference of f with respect to x_i. Recall that $df/dx_i = f_i(0) \oplus f_i(1)$. The Boolean function df/dx_i represents a set of input vectors $V = \{(x_1, x_2, ..., x_n) \mid df/dx_i = 1\}$. Since x_i is totally controllable, each element of V is a stuck-at test for x_i. V contains every possible stuck-at test for x_i such that x_i is observed at f. Let $h(x_1, x_2, ..., x_n)$ be a product of primary inputs. If h = 1 implies $df/dx_i = 1$, then h is said to be an implicant of df/dx_i and h represents one or more elements of V. Each path from the root of the test graph to the one terminal vertex represents an implicant of df/dx_i.

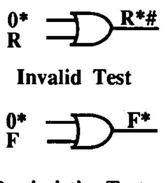

Fig. 6. Hazard Convergence

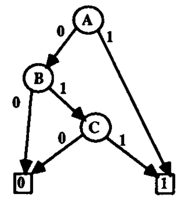

Fig. 7. Example BDD

Free Variables

One implicant from the test graph may represent more than one FTP. Furthermore, the number of FTPs between a primary input-output pair is often much greater than the number of implicants in the test graph. For example, consider the primary input-output pair (x_i, f) in Fig. 8 where $f = x_1x_2 + x_1'x_3$. The Boolean difference of f with respect to x_3 is x_1'. Thus the test graph contains one implicant. However, there are three possible FTPs between x_3 and f: a two-dimensional path through gates B and C, a two-dimensional path through gates C and D, and a one-dimensional path through gate E. Hence, it is desirable to derive more than one test from a single implicant. This is facilitated by the concept of free variables.

Definition: Consider a primary input-output pair (x_i, f). Denote the set $\{x_j \mid df/dx_i(x_j = 0) \neq df/dx_i(x_j = 1)\}$ as S_{td} and the set of primary inputs which primary output f is structurally dependent upon as S_{sd}. The set of primary inputs given by $S_{sd} - S_{td} - x_i$ is the set of free variables, S_{fv}, with respect to the primary input-output pair (x_i, f). □

For the primary input-output pair (x_3, f) in Fig. 8: $S_{sd} = \{x_0, x_1, x_2, x_3\}$, $S_{fd} = \{x_1, x_2, x_3\}$, $S_{td} = \{x_1\}$, and $S_{fv} = \{x_0, x_2\}$. All three possible FTPs can be generated by deriving tests with all possible assignments of the free variables: x_0 and x_2. This technique has the drawback that different free variable assignments may result in identical FTPs. However, we feel that the positive benefit of additional FTPs (each of which can test several fault sites) greatly outweighs the negative aspect of a test set which may be slightly longer than necessary.

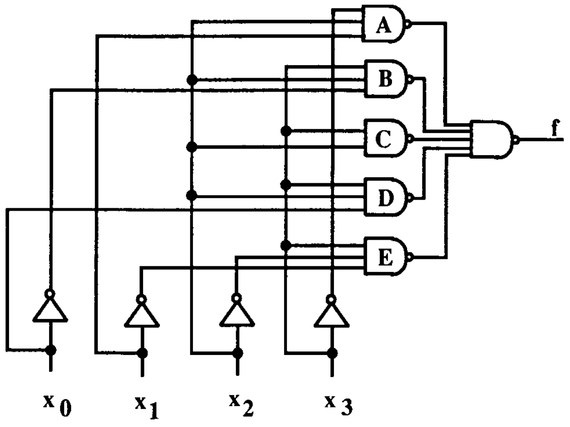

Fig. 8. Example Circuit

Implementation

The current implementation is shown in Fig. 9. The global strategy of the approach is to generate paths from each primary input to each primary output. The first step of the procedure is to construct control graphs representing the switching function of each node in the circuit and determine topological dependencies. Let f be a primary output which is functionally dependent on the set of primary inputs, S_{fd}, such that $x_i \in S_{fd}$. The program attempts to generate a number of FTPs (test_quota, defined by the user) from f, to each primary input in S_{fd}. First the test graph, df/dx_i, is constructed and an implicant (stuck-at test vector) is selected from the test graph. Next we determine the set of free variables, S_{fv}, and derive 2 ** $|S_{fv}|$ vector pairs. If the number of vector pairs becomes greater than the test_quota, or there are no more implicants in the test graph, then the next primary input in S_{fd} is selected. Otherwise, another implicant is selected from the test graph. A test (v_1, v_2) is derived from a vector (implicant) by copying the vector and complementing x_i so that v_1 is a stuck-at one (stuck-at zero) test and v_2 is stuck-at zero (stuck-at one) test for x_i.

```
delay test generation
{
        read in test_quota;
/* test_quota = number of tests per primary input-output pair */
        generate_control_graphs;
        for all primary outputs
        {
                construct_$S_{sd}$;
                /* $S_{fd} = \{x_j \mid f_j(0) \neq f_j(1)\}$ */
                for all primary inputs in $S_{fd}$
                {
                /* let the primary input-output pair be $(x_i, f)$ */
                        generate_df/dx$_i$_graph;
                        test_count = 0;
                        while(test_count < test_quota
                                and more implicants in df/dx$_i$)
                        {
                                select_next_implicant;
                                construct_$S_{fv}$;
                                k = 2 ** $|S_{fv}|$;
                                derive_k_tests_from_implicant;
                                test_count = test_count + k;
                        }       /* end while */
                }       /* end for all primary inputs in $S_{fd}$ */
        }       /* end for all primary outputs */
}       /* end delay test generation */
```

Fig. 9. Psuedo Code for Delay Test Generation

It should be noted that the vector pairs are actually applied as 3-tuples of (v_1, v_2, v_1). This is done in order to create FTPs which test the path through the circuit with respect to both inversion parities. This leads to the following:

Corollary 1: The existence of an FTP, p, created by (v_1, v_2) implies the existence of a complementary FTP, p', created by (v_2, v_1).

Proof: Since any primary input, x_i, is completely controllable, it follows that:

$x_i(df/dx_i) \neq \emptyset \Leftrightarrow df/dx_i \neq \emptyset \Leftrightarrow x_i'(df/dx_i) \neq \emptyset$. Hence the proof. □

Complexity

The time complexity is dominated by the tasks of creating the control graphs and determining the Boolean differences. Both of these tasks are performed using the procedure Apply by Bryant. Apply accepts two graphs representing operands f_1 and f_2, and a binary operator, <op>, and constructs a graph representing the result, f_1 <op> f_2. In order to convert the result to its canonical form, an additional procedure, Reduce, must be invoked following each execution of Apply. Detailed descriptions of Apply and Reduce can be found elsewhere [19]. The time complexity of Apply is given by Bryant as $O(|G_1||G_2|)$, where $|G_1|$ and $|G_2|$ denote the number of vertices in the graphs which represent the operands. Since Reduce must be called after each execution of Apply, its cost must be included. The time complexity of Reduce is $O(|G|\log(|G|))$, where G is the number of vertices in the graph representing the result of Apply (f_1 <op> f_2). Since whether the cost of Apply or Reduce is greater depends on the operands and the logic function performed, we represent the cost of one call of Apply as $\mathbf{MAX}(|G_1||G_2|, |G|\log(|G|))$. As stated previously, only one type of circuit has been found, an integer multiplier, for which the size of the BDD is exponential in the number of primary inputs. Furthermore, for many common circuits the dependence of the BDD size upon the number of primary inputs is a low order polynomial, e.g., quadratic for symmetric functions and linear for an ALU bit slice [19].

To analyze the complexity of creating the control graphs and determining the Boolean difference we simply count the number of times Apply is invoked. For the control graphs this is equal to the left-hand term in parentheses, below, where n is the number of gates in the circuit and k_i is the number of inputs to the ith gate. For all determinations of the Boolean difference the number of calls is represented by the right-hand term in the parentheses, below, where m is the number of primary outputs and p_j is the number of primary inputs in the control graph of the jth primary output. Therefore, the number of times Apply is called is equal to the sum of the two terms in parentheses and the cost of the test generation method can be no greater than:

$$\left(\sum_{i=1}^{n}(k_i - 1) + \sum_{j=1}^{m} p_j\right)\mathbf{MAX}(|G_1||G_2|, |G|\log(|G|))$$

where $|G_1|$,$|G_2|$, and $|G|$ represent the number of vertices in the largest occurring BDDs. Hence, we claim that if the sizes of the control graphs do not grow exponentially with the circuit size, then our technique has a time complexity which is polynomial with respect to the size of the circuit.

On the other hand, for nearly all circuits, delay fault test generation methods based on the D-algorithm or PODEM have a complexity which is exponential in the size of the circuit. Difficulty occurs when a redundant or hard-to-test fault is targeted by such a method and the entire space of possible input vectors must be searched. However, for our approach, the complexity of constructing a test graph is constant for all primary inputs in the control graph of a primary output (regardless of whether or not a primary input is redundant or difficult to test).

PRELIMINARY RESULTS

Preliminary implementations of the test generation method and the fault simulator have been completed and applied to the 74LS181 ALU. As stated previously, in order to obtain data based on the entirety of each circuit and in the absence of precise timing data which would facilitate the selection of "critical paths", the reported delay fault coverages are computed in the usual way and are not path oriented. The fault set considered by the simulator is the set of checkpoint faults, i.e., the set of all primary inputs, fanout branches, and outputs of XOR gates. A robust fault coverage was determined, i.e., delay fault effects were propagated according to the tables in Fig. 4. A plot of fault coverage versus pattern count is shown in Fig. 10. A cost comparison between our method and a psuedorandom sequence is shown in Table 1 (where the units are CPU minutes:seconds on a VAX 11/780).

Since no effort was made to optimize the code, the run times are rather large. However, the times still provide a basis for a comparison of relative costs. The psuedorandom sequence (no vector is allowed to repeat) was exhaustive. Row three in Table 1, the computational enhancement, was determined by dividing the time required to simulate the psuedorandom sequence by the sum of the deterministic test generation and simulation times. Column two contains results for our method where 4 tests (paths) were generated for each primary input-output pair, p = 4 (the third column concerns the case for p = 2). Our method achieved higher fault coverages while providing substantial cost advantages (computational enhancements of 6.32 and 5.55). While the ALU is not a large circuit, we feel these results are promising.

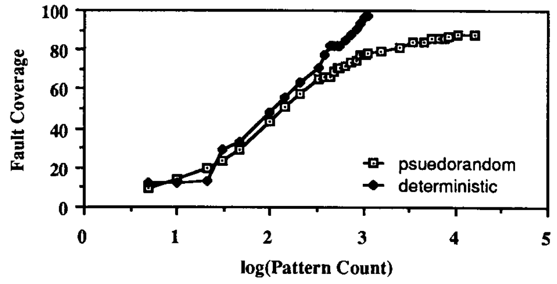

Fig. 10. Fault Coverage vs. log(Pattern Count) (four paths per primary input-output pair)

The simulation enhancement is the cost of simulating the psuedorandom sequence divided by the cost of simulating the deterministic test sets. Note that the reductions in the test set lengths are much less than the reductions in the simulation costs. Since the test sets are simulated as sequences of vector pairs, the number of bits which differ between successive vector pairs has an obvious impact on the amount of work required for event-driven simulation. It is easy to show that one-third of the adjacent vector pairs in the deterministic test set differ in only one bit position. Consequently, for psuedorandom and deterministic test sets of equal length, the psuedorandom sequence places a greater burden on an event-driven simulator.

	Psuedo-random	Deterministic p = 4	p = 2
Test Generation Cost	NA	3:38	3:32
Simulation Cost	27:01	1:14	0:48
Computational Enhancement	1	5.55	6.23
Test Set Length	16,384	1,116	558
Fault Coverage	88%	97%	91%
Simulation Enhancement	1	21.9	33.8
Test Set Length Reduction Ratio	1	14.7	29.4

Table 1. Cost Comparison of Deterministic and Psuedorandom Sequences

CONCLUSION

We have presented a novel approach to delay fault test generation which has the attributes of efficiency and robustness. For most common circuits, the method has a time complexity which is polynomial as opposed to those based on the D-algorithm or PODEM which have an exponential complexity for nearly all circuits. Preliminary results were presented which indicate that our method is a cost-effective alternative to random testing for delay faults. Our tests are valid in the presence of multiple delay faults. Our tests are also valid in the presence of hazards except in one case which is straightforward to detect. We also proposed a simple scannable latch which facilitates the application of vector pairs to the combinational section of the machine. The space-time penalties imposed by the latch are minimal.

REFERENCES

[1] C. C. Liaw, S.Y. Su, and Y.K. Malaiya, "Test Generation for Delay Faults Using Stuck-at-Fault Test Set," Proc. 1980 Int'l Test Conf., pp.167-175, Nov. 1980.
[2] J. Savir and W. H. McHanney, "Random Pattern Testability of Delay Faults," Proc. 1986 Int'l Test Conf., pp. 263-273, Sept. 1986.
[3] J. A. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar, "Transition Fault Simulation by Parallel Pattern Single Fault Propagation," Proc. 1986 Int'l Test Conf., pp. 560-571, 1986.
[4] Y.K. Malaiya and R. Narayanaswamy, "Testing for Timing Failures in Synchronous Sequential Integrated Circuits," Proc. 1983 Int'l Test Conf., pp.560-571, Oct. 1983.
[5] Y. Levendel and P.R. Menon, "Transition Faults in Combinational Circuits: Input Transition Test Generation and Fault Simulation," Proc. 15th Fault Tolerant Comput. Symp., pp. 278-283, 1986.
[6] S. K. Jain and V. D. Agrawal, "Test Generation for MOS Circuits using D-Algorithm," Proc. 20th Design Automation Conf., pp. 64-70, 1983.
[7] Y. El-ziq, "Automatic Test Generation for Stuck-Open Faults in CMOS VLSI," Proc. 18th Design Automation Conf., pp.347-354, June 1981.
[8] J. Rajski and H. Cox, "A Method of Test Generation and Fault Diagnosis in Very Large Circuits," Proc. 1987 Int'l Test Conf., pp. 932-943.
[9] C. J. Lin and S. M. Reddy, "On Delay Fault Testing in Logic Circuits," IEEE Trans. Comput. Aided Design, pp. 694-703, Sept. 1987.
[10] K. Kishida, F. Shirotori, Y. Ikemoto, S. Ishiyama, and T. Hayashi, "A Delay Test System for High Speed Logic LSI's," Proc. 23rd Design Automation Conf., pp. 786-790, July 1986.
[11] T. Hayashi, K. Hatayama, K. Sato, and T. Natabe, "A Delay Test Generator for LSI Logic," Proc. 14th Fault Tolerant Comput. Symp., pp. 146-149, 1984.
[12] D. K. Bhavsar, "A New Economical Implementation for Scannable Flip-Flops in MOS," IEEE Design and Test of Computers, pp. 52-56, June 1986.
[13] J. P. Roth, "Diagnosis of Automata Failures: a Calculus and a Method," IBM Journal of Research and Development, vol. 10, pp. 278-291, 1966.
[14] G. L. Smith, "Model for Delay Faults Based Upon Paths," Proc. 1985 Int'l Test Conf., pp. 342-349, 1985.
[15] H. Ibarra and S. K. Sahni, "Polynomially Complete Fault Detection Problems," IEEE Trans. Comput., vol. c-24, No. 3, March 1975.
[16] E. S. Park and M. R. Mercer, "Robust and Nonrobust Tests for Path Delay Faults in a Combinational Circuit," Proc. 1987 Int'l Test Conf., pp. 1027-1034.
[17] M. A. Bruer and L. Harrison, "Procedures for Eliminating Static and Dynamic Hazards in Test Generation," IEEE Trans. Comput., vol. c-23, pp. 1069-1078, Oct. 1974.
[18] S. B. Akers, "Binary Decision Diagrams," IEEE Trans. Comput., vol. c-27, pp. 509-516, June 1978.
[19] R. E. Bryant, "Graph-Based Algorithms for Boolean Function Manipulation," IEEE Trans. Comput., vol. c-35, No. 8, Aug. 1986.