Scan-Based Transition Test

Jacob Savir, *Fellow, IEEE,* and Srinivas Patil, *Member, IEEE*

***Abstract*—Skewed-load transition test is a form of scan-based transition test where the second vector of the delay test pair is a one bit shift over the first vector in the pair. This situation occurs when testing the combinational logic residing between scan chains. In the skewed-load test protocol, in order not to disturb the logic initialized by the first vector of the delay test pair, the second vector of the pair (the one that launches the transition) is required to be the next (i.e., one bit-shift) pattern in the scan chain.**

Although a skewed-load transition test is attractive from a timing point of view, there are various problems that may arise if this strategy is used. This paper investigates several issues of skewed-load transition test. Issues such as transition test calculus, detection probability of transition faults, transition fault coverage, and enhancement of transition test quality are thoroughly studied.

I. Introduction

A. Delay Faults

THE PURPOSE of a delay test is to verify that a given logic circuit operates correctly at the desired clock speed. Static tests (normally assuming the stuck-at fault model) are useful in verifying the input-output behavior of the logic circuit, assuming that the signals are allowed sufficient time to stabilize. However, when operating at the system clock speed, some of the signals in the logic circuit may not propagate in time to be latched at either the circuit outputs or the internal latches, eventually causing system failure. The purpose of a delay test (or, as sometimes called, AC test) is to verify that all the propagation paths in a logic circuit are within the specified tolerances. The variation of delays beyond the specified values may be due to process variations, stray capacitances, or physical defects such as open metal lines, open resistor, etc. The primary outputs, and outputs feeding into internal shift register latches (SRL's), will be collectively referred to as *circuit outputs*. Similarly, primary inputs, and inputs to the logic circuit originating at SRL's, will be referred to simply as *circuit inputs*.

In an LSSD [1] environment, the SRL's are reconfigured in test mode so that the logic circuit behaves like a combinational circuit. Delay test for combinational logic requires the application of pairs of vectors: an *initializing vector*, which establishes the initial conditions before a transition is propagated through the circuit, and a *final vector*, which launches the desired transition and establishes the conditions necessary to propagate it. The output response is captured after a predetermined time interval (usually after one system clock cycle) to verify whether there is a delay fault.

Manuscript received January 23, 1992; revised December 14, 1992. This paper was recommended by Associate Editor Sharad Seth.

J. Savir is with IBM Corporation, East Fishkill Facility, Hopewell Junction, NY 12533.

S. Patil is with IBM Corporation, Endicott, NY 13760.

IEEE Log Number 9207799.

B. Delay Fault Models

Two types of fault models have been popular in the literature: *gate delay* [2] and *path delay* [3]. The gate delay fault model is a localized fault model in a sense similar to the stuck-at fault model. The gate delay fault is assumed to be either associated with a gate input or a gate output. If a slow-to-rise delay fault is associated with a gate input, any logic 0-to-1 transition at that input will be delayed by a magnitude equal to the size of the delay fault. Thus, each delay fault has two attributes associated with it: the type of transition it affects (rising transition or a falling transition) and the magnitude (the amount by which an affected transition is delayed). Each line or net can potentially have two types of delay faults associated with it: a slow-to-rise (STR) fault (delays the rising transition) and a slow-to-fall (STF) fault (delays the falling transition).

We define the *slack* of a path as the difference between the clock period when the circuit outputs are latched (or observation time) and the propagation delay of the path under consideration. A negative value of the slack means that the propagation delay of the path exceeds the clock period, and a positive value of the slack means that the propagation delay is within the clock period. For a gate delay fault to cause an incorrect value to be latched at a circuit output, the size of the delay fault must be such that it exceeds the slack of at least one path from the site of the fault to an output. A gate delay fault whose magnitude is such that it causes the propagation delays of all paths passing through the fault site to exceed the clock period (or observation time) is called a *gross delay fault* [4]. To test for a gross delay fault, it is only necessary to excite and propagate the fault effect to any output of the circuit.

Conversely, the path delay fault model tries to account for the cumulative, or additive, effect of delays along a path. Even if the gate delays of individual gates along a path are within the specified tolerances, the cumulative effect of the delay variations may be such that the overall delay of a path exceeds the clock period, causing an incorrect value to be latched at a circuit output. The gate delay fault model is deficient in modeling such delay de-

fects. The path delay fault model is able to model a delay defect that may be distributed over a region of a circuit.

Both the gate and path delay fault models have their own advantages and disadvantages. If most of the delay defects seen in practice are point (or localized) defects, then the gate delay fault model is adequate. However, if the defects are more global in nature, then it would be necessary to adopt the path delay fault model. In addition, from the test generation, fault simulation, or diagnosis viewpoints, the gate delay fault model presents less complexity than the path delay fault model. For a logic circuit with n lines, there can be at most $2n$ gate delay faults (one STR and one STF fault on each line). However, there can be n^2 paths in a typical logic circuit (exponential in the worst case) of the same size that may have to be considered under the path delay fault model.

Unlike stuck-at-fault tests, where the detection of a modeled fault is guaranteed by a test as long as the signals in the circuit are allowed enough time to stabilize, delay tests are valid only under certain conditions—the conditions assumed during test generation. When interpreting coverage figures for delay tests, the assumptions used to generate the delay tests have to be taken into account. One of the primary concerns in delay testing is the presence of hazards that may potentially invalidate a test; that is, the circuit may pass a test although a fault may be present. The fault effect may also be masked due to delays in other parts of a circuit. Tests that ensure the delay test is valid irrespective of delays in other parts of the logic circuit are called *robust tests* [3]. However, robust tests do not exclude the existence of hazards in the portion of the circuit being tested. Hazards are allowed as long as they do not invalidate the test. *Hazard-free tests* are a subset of robust tests, and they exclude the existence of hazards in the signal values used to propagate the fault effect [4]. Both robust tests and hazard-free tests impose severe constraints on the signal values associated with the test. Generation of robust tests is more complex than generation of nonrobust tests. In addition, the coverage attainable by robust tests is lower than that attainable by nonrobust tests, even on small benchmark circuits [5].

In most cases, the signal conditions required for delay tests are a superset of those required for the corresponding stuck-at tests. Thus, it can be expected that the fault coverage attainable by delay tests is in some way limited by the stuck-at fault coverage. To achieve fault coverage figures closer to the stuck-at-fault coverage figures, it becomes necessary to consider nonrobust tests. However, it is very difficult to characterize the behavior of some nonrobust tests without making some simplifying assumptions. The transition fault model [2] allows one to generate delay tests without a prohibitive increase in the costs of test generation and fault simulation. The gate delay fault model, or the point defect assumption, is implicit in the transition fault model. Another important assumption is the assumption of gross gate delay faults, that is, gate delay faults that manifest themselves on all paths passing through the site of the fault. We refer to tests generated using the transition fault model as *transition tests*. A transition test for a gate delay fault is such that:

1) It launches the desired transition at the site of the fault.
2) The final pattern is a test for the corresponding stuck-at-0 fault if the delay test is for a slow-to-rise fault, and a stuck-at-1 test if the delay test is for a slow-to-fall fault.

It is easy to see that the only extra condition (beyond the requirement for stuck fault tests) to be satisfied under the transition fault model is the presence of an initializing pattern immediately before the application of the stuck-at test pattern. It can be expected that the fault coverage figures obtained using transition test patterns will be very close to those obtained for stuck-at faults for the same logic circuit. One should keep in mind, however, that the fault coverage figures obtained for transition patterns are for gross delay faults, and more detailed analysis will be necessary to evaluate the coverage for smaller delay faults [6]. Another important distinction between transition tests and robust tests is that transition tests do not require complementary initial and final output values when the delay fault is detected, whereas robust tests do. A robust test would result in a 0-to-1 or a 1-to-0 transition on at least one output (probably with some hazards between transitions), whereas a transition test may result in a waveform with the same initial and final values, but a complementary value in between; the duration of the complementary value being determined by the size of the fault. It is precisely this type of behavior that makes analysis of transition tests for small delay faults difficult. However, in timing optimized designs, it can be shown that the transition (*gross delay*) fault coverage can be used to predict accurately the actual delay fault coverage, even when arbitrary delay fault sizes are assumed [7].

C. Focus of the Paper

The paper focuses on a scan-based transition test. There are basically two ways of performing such a test.

To describe the first way, assume an LSSD structure with multiple scan chains. For simplicity, let the structure have two scan chains. Let the first vector of the delay test pair be the contents of scan-chain 1. The second vector of the delay test pair is the circuit response to the vector residing in scan chain 2. We call this a *broad-side transition test*. This form of scan-based transition test does not suffer from shift dependency. It has, however, a problem that it requires two analysis time frames (one for the first vector, and one for the second vector in the pair). We will not, however, consider this type of delay test in this paper.

The second method, and the one discussed in this paper, uses a protocol where the second vector of the delay test pair is a one-bit shift over the first vector in the pair. In this test protocol, in order not to disturb the logic initialized by the first vector of the delay test pair, the second

vector of the pair (the one that launches the transition) is required to be the next (i.e., one-bit shift) pattern in the scan chain. We call this protocol a *skewed-load transition test*, or simply a SLOTT. Although SLOTT requires only a single time frame for analysis, it has a problem with shift dependencies.

This paper discusses several issues related to the transition test. Since the paper concentrates on the transition test, the gate delay fault model is adopted. Moreover, the patterns used for the transition test are generated by the scan hardware. Under constraints placed by scan hardware such as LSSD, it may not be possible to apply all transition test pattern pairs [8]. This paper examines the properties of transition tests under the constraints placed by scan hardware. Section II describes the LSSD hardware constraints and defines a calculus to provide a theoretical framework for analysis of transition faults under such constraints. Section III deals with detection probability for transition faults. Section IV describes a subcone-based input separation technique to enhance the transition fault coverage. Section V draws some conclusions from this study.

The interested reader is also directed to [9] and [10] for a more detailed treatment of the subject matter of this paper.

II. Skewed-Load Transition Test Calculus

Let V_1 and V_2 be the two test vectors that comprise a delay test pair, where V_1 is the first vector and V_2 is the second vector in the pair. For (V_1, V_2) to constitute a *transition test* for a given fault, V_1 needs to set the net in question to its initial value, and V_2 needs to be a test for the associated stuck-fault. The initial value for an STR (STF) fault is 0 (1), and the final vector is a stuck-at 0 (1) test [2].

The issue of the actual implementation of a scan-based delay test is beyond the scope of this paper. Issues such as proper clock timing that will allow for a meaningful delay test are real practical problems. The interested reader should refer to section 10.2 in [11] for information regarding the implementation of delay test in an LSSD environment.

To analyze the properties of a transition test it is sufficient to concentrate on the following simplified protocol. Let the logic to be tested be embedded between shift register chains [1]. Let the delay test protocol be such that the vector V_1 is shifted in through the chain and, then, applied against the logic. The vector V_2 is obtained from V_1 by shifting it one bit down the chain and adding to it an extra bit through the scan-in port. Thus, V_2, except for the appended bit, is a one-bit shift over its V_1 predecessor. The circuit responses to this delay test pair are captured at the output latches at a time dictated by the machine speed (or even slightly faster to guarantee that all signals will arrive in time when used in the system) and then scanned out. The scan-out port may be connected to a signature analyzer [11] if the test strategy uses built-in self-test (BIST). Any discrepancy between the expected and actual response indicates that some delay fault has been detected. This delay test protocol, where V_2 is a one-bit shift over V_1 is called a skewed-load transition test, or simply a SLOTT. The authors in [12] describe a practical implementation of this test. They refer to this test as a "twist" (i.e., the stuck-patterns are modified to propagate transitions).

In this section we develop a Boolean-difference–based calculus for SLOTT. With this calculus it is possible to compute all the SLOTT pairs (called complete SLOTT set) that detect a given transition fault. The calculus allows the computation of both V_1 and V_2 simultaneously. We, therefore, refer to this as a *single time-frame* calculus. Since the method is Boolean-difference–based, it has all the advantages and disadvantages that go with this method. In the discussion that follows it is assumed that the reader is familiar with the Boolean difference technique. The interested reader is referred to [1] and [2].

Let g be a line in a circuit having n inputs and a single output, F. We will sometimes use g to refer to the name of the line, and sometimes to refer to the Boolean function realized by this line. The actual meaning should be clear from the context. We define the *displaced function*, $d(g)$, associated with the function g, as the Boolean function derived from g by replacing every input x_i of g by the input x_{i+1}. For example, if $g = x_1\bar{x}_2$, then $d(g) = x_2\bar{x}_3$. The displaced function gives an indication as to how the Boolean function of a line changes when its input domain undergoes a one-bit shift.

Consider the computation of a SLOTT for an STR fault on line g. The vector V_2 needs to be a stuck-at-0 test for line g. As a result, it must be a member of the solution set of

$$g \frac{\partial F}{\partial g} = 1.$$

Furthermore, since V_2 is a one-bit shift over V_1, and since V_1 needs to set line g to a value 0, V_2 needs to also be a member of the solution set of $d(\bar{g}) = 1$. Combining these two requirements, we get the equation whose solution yields the complete SLOTT set for an STR fault on line g:

$$d(\bar{g}) g \frac{\partial F}{\partial g} = 1. \tag{1}$$

By a similar argument, the equation whose solution yields the complete SLOTT set for an STF fault on line g is

$$d(g) \bar{g} \frac{\partial F}{\partial g} = 1. \tag{2}$$

Notice that although (1) and (2) have been created so that their solution constitutes a complete enumeration of all the V_2's, it also reveals all the V_1's attached to these V_2's. To clarify this point, recall that V_1 is a reverse one-bit shift of its successor, V_2. Thus, the first bit of V_1 is

the same as the second bit of V_2; the second bit of V_1 is the same as the third bit of V_2, etc. The only bit that cannot be determined by this shift correspondence is the last bit of V_1. This information, however, is maintained by the calculus under an extraneous variable, x_{n+1}. This point will become clearer once we review a couple of examples.

Example 1: Consider the circuit of Fig. 1. We have

$$F = g + x_3, \quad g = x_1x_2, \quad \bar{g} = \overline{x_1} + \overline{x_2},$$

$$d(g) = x_2x_3, \quad d(\bar{g}) = \overline{x_2} + \overline{x_3}$$

$$\frac{\partial F}{\partial g} = \overline{x_3}$$

$$d(g)\bar{g}\frac{\partial F}{\partial g} = x_2x_3(\overline{x_1} + \overline{x_2})\overline{x_3} = 0 \mapsto V_2(g_{STF}) = \phi$$

$$d(\bar{g})g\frac{\partial F}{\partial g} = (\overline{x_2} + \overline{x_3})x_1x_2\overline{x_3} = x_1x_2\overline{x_3} = 1$$

$$\mapsto V_2(g_{STR}) = \{110\}, \quad V_1(g_{STR}) = \{10X\}.$$

Notice that in this case no extraneous variable entered the calculation and, therefore, the last bit of V_1 is a don't care. Also notice that there is no SLOTT pair that detects an STF fault on line g. □

Example 2: The circuit of Fig. 2 is identical to that of Fig. 1, except for the input line ordering along the scan chain. We would like to see if this new input ordering will help detect the previously undetected fault, g_{STF}. We have

$$F = g + x_1, \quad g = x_2x_3, \quad \bar{g} = \overline{x_2} + \overline{x_3},$$

$$d(g) = x_3x_4, \quad d(\bar{g}) = \overline{x_3} + \overline{x_4}$$

$$\frac{\partial F}{\partial g} = \overline{x_1}$$

$$d(g)\bar{g}\frac{\partial F}{\partial g} = x_3x_4(\overline{x_2} + \overline{x_3})\overline{x_1} = \overline{x_1}\overline{x_2}x_3x_4 = 1$$

$$\mapsto V_2(g_{STF}) = \{001\}, \quad V_1(g_{STF}) = \{011\}.$$

Notice that here the extraneous variable, x_4, does participate in the solution. Its value ($x_4 = 1$) determines the value of the last bit of V_1.

$$d(\bar{g})g\frac{\partial F}{\partial g} = (\overline{x_3} + \overline{x_4})x_2x_3\overline{x_1} = \overline{x_1}x_2x_3\overline{x_4} = 1$$

$$\mapsto V_2(g_{STR}) = \{011\}, \quad V_1(g_{STR}) = \{110\}. \ \square$$

Next we generalize (1) and (2) to networks with multiple outputs. To that end, let $F_1, F_2, \cdots, F_m$ be the set of all outputs latched in the scan chain. To allow a transition fault to be detected at an arbitrary output, a sum (simple OR) of the Boolean differences of each output with respect to the line in question will replace the simple Boolean difference in (1) and (2).

Thus, to compute the complete SLOTT set for an STR fault on line g solve

$$d(\bar{g})g\sum_{i=1}^{m}\frac{\partial F_i}{\partial g} = 1. \tag{3}$$

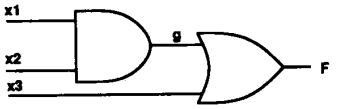


Fig. 1. Circuit for Example 1.

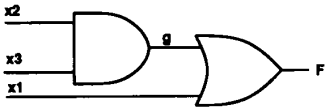


Fig. 2. Circuit for Example 2.

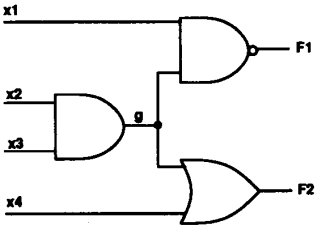


Fig. 3. Circuit for Example 3.

In addition, the equation whose solution yields the complete SLOTT set for an STF fault on line g is

$$d(g)\bar{g}\sum_{i=1}^{m}\frac{\partial F_i}{\partial g} = 1. \tag{4}$$

Example 3: Consider the circuit of Fig. 3. We have

$$g = x_2x_3, \quad \bar{g} = \overline{x_2} + \overline{x_3}, \quad d(g) = x_3x_4,$$

$$d(\bar{g}) = \overline{x_3} + \overline{x_4}$$

$$\overline{F_1} = gx_1, \quad F_2 = g + x_4, \quad \frac{\partial F_1}{\partial g} = x_1, \quad \frac{\partial F_2}{\partial g} = \overline{x_4}$$

$$d(\bar{g})g\sum_{i=1}^{2}\frac{\partial F_i}{\partial g} = (\overline{x_3} + \overline{x_4})x_2x_3(x_1 + \overline{x_4})$$

$$= x_2x_3\overline{x_4} = 1.$$

Thus, $V_2(g_{STR}) = \{X110\}$, $V_1(g_{STR}) = \{110X\}$.

$$d(g)\bar{g}\sum_{i=1}^{2}\frac{\partial F_i}{\partial g} = x_3x_4(\overline{x_2} + \overline{x_3})(x_1 + \overline{x_4})$$

$$= x_1\overline{x_2}x_3x_4 = 1.$$

Thus,

$$V_2(g_{STF}) = \{1011\}, \quad V_1(g_{STF}) = \{011X\} \qquad \square$$

It is important to note that Boolean difference had some severe problems when implemented as a vehicle to compute test patterns for stuck-faults. The problems stemmed from the fact that the method is symbol-driven rather than number-driven. Other methods, such as path sensitization techniques [13], have been found to be more effective for calculation of stuck-faults test patterns. This may still be true for transition faults. Thus, it may be important to

implement a SLOTT generator by employing such techniques.

III. Detection Probability of Transition Faults

When random patterns are used to detect faults in a circuit, it is often necessary to compute their detection probabilities [11]. The detection probabilities are needed to quantify the goodness of the test. Low detection probabilities may indicate resistance to detection. By knowing the detection probability of every fault in the circuit, one can compute the test length needed to achieve a given level of detection.

To quantify the goodness of a random pattern-based SLOTT, it is necessary to compute the detection probability of all transition faults in question. In an unbiased random SLOTT (i.e., all bits are shifted into the scan with equal probabilities), the detection probability of any *detectable* transition fault is at least $2^{-(n+1)}$. This is due to the fact that at most $n + 1$ bits have to be specified to create the (V_1, V_2) pair. As we have seen in the previous section, some transition faults have no SLOTT pairs. These faults are undetectable, and therefore have zero detection probability.

There are two major algorithms in existence today that allow the computation of detection probabilities of stuck-faults. The Parker–McCluskey algorithm [14] uses symbolic manipulation to compute exact detection probabilities of faults. The cutting algorithm [15] computes lower bounds on detection probabilities of faults. Computation of lower bounds of detection probabilities is much easier than the computation of exact values. The improved cutting algorithm [16] is a combination of both the cutting algorithm and the Parker–McCluskey algorithm. The improved cutting algorithm has the benefit of custom-tailoring the closeness of the bound to the user needs, while keeping the cost of computation within a reasonable range.

In this section we explore the extension of these two algorithms to compute the SLOTT detection probabilities. The extensions are based on the findings of the previous section.

First, consider the extension to the Parker–McCluskey algorithm. As seen in the previous section, computation of SLOTT pairs requires that V_2 be a member of a displaced function, besides being also a stuck-fault test. Fig. 4 shows a block diagram of the circuit that has to be analyzed to compute the SLOTT detection probability a la Parker–McCluskey. We assume that the circuit has a single output F. The generalization to multiple-output circuits is straightforward.

The block N is the original circuit. The line, g, for which the SLOTT detection probability has to be computed, is shown explicitly in this block as a pseudo-output line. The block NF is the circuit N with the stuck-fault injected into line g (stuck-at 0 for an STR and stuck-at 1 for an STF fault). The input lines (1 through n) are connected so that they feed both N and NF in parallel. The circuit $d(N)$ is that portion of circuit N that includes all

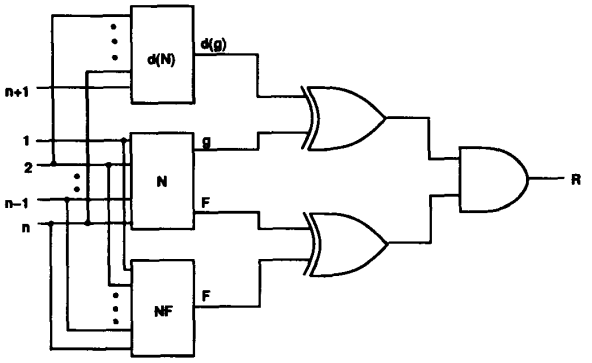


Fig. 4. Circuit for Parker–McCluskey-type analysis.

the logic in the cone of line g. The inputs to block $d(N)$ are connected so that its output implements the displaced function $d(g)$. This is done by connecting input 2 to the first input of $d(N)$; input 3 to the second input of $d(N)$; etc. In cases where input n participates in the cone of line g, the last input of $d(N)$ needs to be an extraneous input. For these cases the last input of $d(N)$ is the extraneous input $n + 1$. This extraneous input is shown explicitly as an input to block $d(N)$.

For a pattern to constitute the V_2 vector of a SLOTT pair, the outputs g and $d(g)$ need to be different (namely, 0 in one and 1 in the other). Furthermore, the outputs F emanating from circuit blocks N and NF need to be different. As a result, an input pattern will constitute a valid V_2 if the output $R = 1$. Consequently, the signal probability of output R is the SLOTT detection probability. This is the SLOTT detection probability and not just the probability of V_2, since the extraneous variable is part of the detection probability calculation.

The following example illustrates the computation.

Example 4: Fig. 5 shows the circuit that needs to be analyzed in order to compute the SLOTT detection probability of g_{STR} in Fig. 1.

In this case $d(N)$ is a single AND gate. It has no extraneous input, since line g (in Fig. 1) does not depend on input x_3. The block NF reduces to a single line after the stuck-at 0 injection into line g.

We assign signal probability symbols p_1, p_2, p_3 to the input lines and propagate these signal probabilities according to Parker–McCluskey until the signal probability of line D is computed:

$$K = p_1p_2 + p_3 - p_1p_2p_3.$$

The signal probability expression for B (after exponent suppression) is

$$B = p_2p_3 + p_1p_2 - 2p_1p_2p_3.$$

Similarly,

$$C = 2p_3 + p_1p_2 - p_1p_2p_3 - 2p_3(p_1p_2 + p_3 - p_1p_2p_3).$$

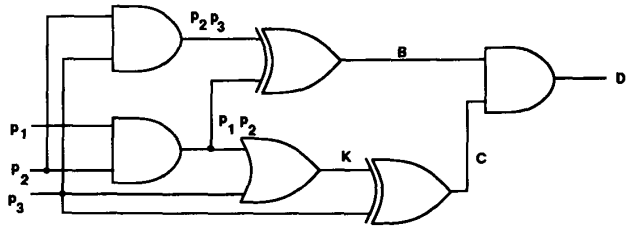


Fig. 5. SLOTT detection probability calculation according to Parker-McCluskey.

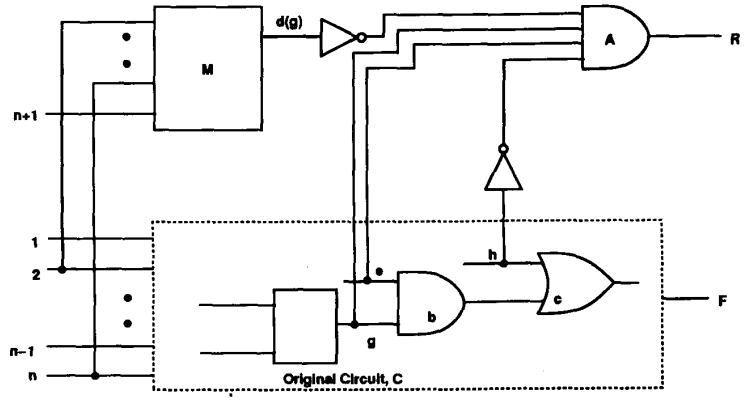


Fig. 6. Circuit for cutting-algorithm-type analysis.

And, after exponent suppression,

$$C = p_1p_2(1 - p_3).$$

Thus, after exponent suppression we get

$$D = BC = p_1p_2(1 - p_3).$$

The SLOTT detection probability of g_{STR} is, therefore, $1/8$ in an unbiased random test. □

Next we proceed to extend the cutting algorithm to handle SLOTT detection probabilities. The cutting algorithm chooses a propagation path from the site of the fault to a primary output. It creates an auxiliary gate that assembles all the necessary conditions to detect the fault along the selected path. The cutting algorithm can be extended to compute lower bounds on SLOTT detection probabilities by adding the conditions set by the displaced function into the auxiliary gate. This extension is described in Fig. 6.

Fig. 6 shows the circuit that needs to be analyzed by the cutting algorithm to compute a lower bound of the SLOTT detection probability of g_{STR}. A path $gbc \ldots$ has been selected to propagate the effect of the fault to a primary output, F. For illustration purposes, the selected path has been chosen to traverse an AND and an OR gate. In addition to the original circuit, C, the displaced function, M, has been added where its input domain is shifted one bit over that of C. As in the Parker–McCluskey-type analysis, the extraneous variable, $n + 1$, feeds the block M. The auxiliary gate is an AND gate that is connected in such a way that its output $R = 1$ whenever the fault g_{STR} is detected along the prescribed path. To detect this fault we need to have $d(g) = 0, g = 1, e = 1, h = 0, \cdots$. Every 0 value requirement is implemented by connecting the corresponding line to the auxiliary gate via an inverter [such as lines $d(g)$ and h]. Every 1 value requirement is implemented by connecting the corresponding line directly to the auxiliary gate (such as lines g and e). Since a single path has been selected to propagate the effect of the fault, the signal probability of line R constitutes a lower bound on the detection probability of g_{STR}. Notice that if we would have wanted to compute the SLOTT detection probability of g_{STF}, line $d(g)$ would have had to be connected to the auxiliary gate directly, and line g through an inverter. The following example illustrates this method.

Example 5: We compute the lower bound on the SLOTT detection probability of g_{STR}, in Fig. 7(a). Fig. 7(b) shows the circuit that needs to be analyzed by the cutting algorithm. In Fig. 7(b) gates 1, 2, and 3 constitute the original circuit. Gates 4 and 5 constitute the displaced function. Gate 6 is the auxiliary gate. Fig. 7(b) displays the propagation of the signal probability bounds according to the improved cutting algorithm [16]. Notice that in Fig. 7(b) one fan-out stem is processed symbolically (symbol = p) and two fan-out branches belonging to the other fan-out stems have been cut (branches a and b). Since $p = 1/2$, the lower bound of the SLOTT detection probability of g_{STR} is $1/8$. The exact detection probability happens to be $3/16$ in this case. □

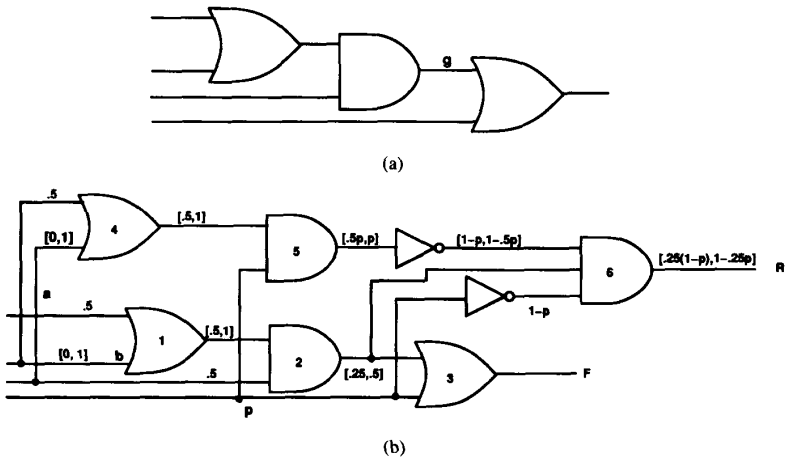


Fig. 7. (a) Circuit for Example 5; (b) circuit to be analyzed by the cutting algorithm.

IV. Test Coverage Enhancement

In [8], a method to reorder the scan chain was proposed based on tracing the cones of logic behind each primary output or pseudo-output (input to a latch element). The reordering was based on one important observation: Under constraints placed by the actual placement of primary inputs or pseudo-inputs (inputs to the combinational logic originating from latches), it may not be possible to apply all possible pattern pairs to the logic for delay test. In SLOTT, the number of distinct pattern pairs that can be applied is only $2(2^n - 1)$, where n is the number of inputs to the logic circuit [compare this with $2^n(2^n - 1)$ when the patterns can be independently applied]. However, in real circuits, very few outputs depend on all of the n inputs. Thus, only a subset of all possible $2^n(2^n - 1)$ patterns need be applied to exhaustively delay test the logic belonging to a particular output. The technique presented in [8] attempts to assign inputs in the scan chain such that no two adjacent inputs in the scan chain feed the same output, thus eliminating the shift dependency. When such an assignment is not possible, dummy latches are introduced between the inputs feeding into nondisjoint sets of outputs.

The use of scan path reordering, combined with the introduction of dummy latches, will always guarantee that an exhaustive set of patterns can be applied [8]. However, in the case of single-output circuits, and circuits where there is a great deal of overlap between logic feeding different output cones, the cost of introducing dummy latches may become prohibitive.

We propose here an alternative to the method presented in [8] in cases where one is not allowed to use dummy latches. The proposed method is based on fault simulation of the logic circuits which allows one to concentrate only on those portions of the circuit where undetectable transition faults exist. This method was the outcome of a rather surprising observation: The transition fault coverage for the ISCAS combinational benchmark circuits [17] by using SLOTT was very close to the coverage achieved by using independent patterns (see Table I). By a closer examination of the simulation data, the faults undetectable under SLOTT assumption were found to be distributed nonuniformly among all gate levels (after the circuit was levelized). Most of the undetectable faults appeared to be at levels closer to the circuit inputs than to circuit outputs. One possible explanation for this behavior is that the intervening logic serves to cancel out the shift dependencies that exist as one traverses the circuit from the inputs toward the outputs. This observation is key to our scan path reordering technique. This indicates that it may only be necessary to reorder the flip-flops to account for undetectable faults in the first few levels of logic.

TABLE I
Input Separation Based on Subcones

Circuit	Lmax	Lslice	Coverages		
			Default	Re-Ord	Indep
c432	30	6	97.07	98.72	98.72
c499	17	4	99.02	99.13	99.13
c880	35	17	95.89	99.75	100.00
c1355	40	5	96.80	98.79	98.83
c1908	61	31	97.48	99.56	99.69
c2670	53	26	82.88	84.70	86.48
c3540	71	36	91.01	95.46	96.55
c5315	68	35	96.56	99.30	99.30
c6288	218	20	99.19	99.31	99.32
c7552	62	30	94.70	95.39	95.42

Fig. 8 shows a cumulative plot of the percentage of undetectable transition faults versus the logic level for the largest ISCAS circuit (c7552). It can be seen that most of the undetectable faults are accounted for by the time 50% of the logic levels are traversed. Before discussing the actual algorithm, we shall illustrate the method using a simple example. Consider the circuit shown in Fig. 9.

Assume that the scan path ordering is as shown by the downward arrow. This is actually the worst possible or-

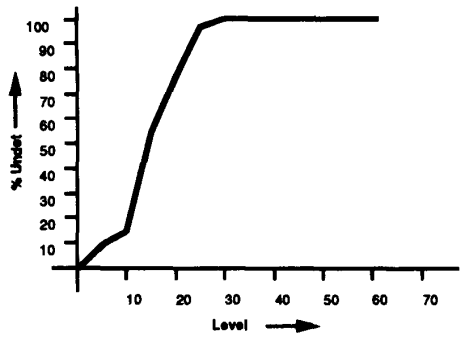


Fig. 8. Cumulative distribution of undetectable faults in c7552.

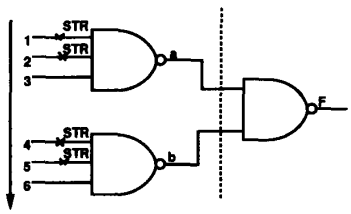


Fig. 9. Circuit with low SLOTT coverage.

dering for this circuit, and four of the 18 faults turn out to be undetectable (as marked in Fig. 9 where STR denotes a slow-to-rise fault), giving a fault coverage figure of 78%. If inputs are considered to be at level 1, all undetectable faults occur at level 1. A plot of undetectable faults versus logic level are shown in Fig. 10. We call this the *undetectability profile*.

Since all four undetectable faults of interest occur at level 2 or lower, we are going to "slice" the circuit at level 2 as shown by the vertical line in Fig. 9. We then create *secondary outputs* wherever the imaginary line cuts—in the case of our example, at lines a and b. For the purpose of reordering, we assume that there are outputs at lines a and b. Thus, output a depends on inputs 1, 2, and 3, and output b depends on inputs 4, 5, and 6. Using the input separation algorithm given in [8], and without using any dummy latches, we get the scan path ordering $1 \rightarrow 4 \rightarrow 2 \rightarrow 5 \rightarrow 3 \rightarrow 6$, which gives a coverage of 94%. This figure is also the maximum possible SLOTT coverage for the example circuit without the use of dummy latches. Since this is a heuristic, it is not always guaranteed to find the optimal ordering.

The heuristic is outlined below:

1) Levelize the circuit. Set the iteration count, $iter = 1$. Let the maximum allowable iterations be $iter_{max}$.
2) Perform SLOTT fault simulation on the circuit using an efficient fault simulation algorithm, such as PPSFP [18], and assuming some random ordering of inputs in the scan chain.
3) Plot the undetectability profile (similar to the plot shown in Fig. 10).

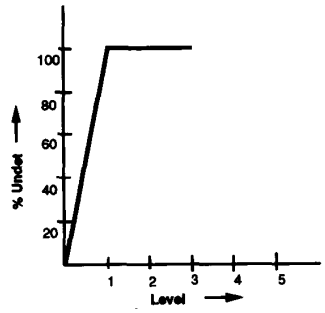


Fig. 10. Undetectability profile for circuit shown in Fig. 9.

4) Let L_{max} be the maximum number of logic levels in the circuit. Select a point on the graph where the percentage of undetectable faults reaches 100% for the first time. Call the corresponding value of the gate level L_{undet}. This means that all undetectable faults are at level L_{undet} or less. Instead of selecting 100% of the undetectable faults, one can also select a figure lower than 100%; say, 90% if L_{undet} is too close to L_{max}.
5) Cut the circuit at level L_{undet} and create secondary outputs.
6) Treat the secondary outputs in the same manner as the primary outputs in [8] and use the input separation algorithm to get a scan path ordering without dummy latches.
7) Fault simulate the circuit again with the new scan path ordering to get a new SLOTT fault coverage figure.
8) If the fault coverage is unacceptable and $iter < iter_{max}$, then choose a small integer δ, increment $iter$, go to 9. Else, go to 10.
9) Compute the transition fault coverages by splicing the circuit of $L_{undet} + \delta$ and $L_{undet} - \delta$. If the coverage obtained by splicing at $L_{undet} + \delta$ is greater than that obtained by splicing at $L_{undet} - \delta$, replace the value of L_{undet} by $L_{undet} + \delta$, else replace the value of L_{undet} by $L_{undet} - \delta$. Go back to 8.
10) If the fault coverage is acceptable we are done.
11) If $iter = iter_{max}$ then an acceptable solution using input separation by subcones could not be found. Use methods detailed in [8] instead.

If the time to compute the undetectability profile by transition fault simulation is considered prohibitive, then the value of L_{undet} can be based on empirical estimates from a few test cases. For example, if it is found from test cases that most of the undetectable faults are in the first half of the logic circuit (i.e., first 50% of the logic levels), then an empirical value for L_{undet} can be chosen to be $0.5L_{max}$.

As pointed out earlier, input separation based on subcones is not guaranteed to yield a SLOTT coverage equal to the stuck-fault coverage (unlike the method proposed in [8], which is guaranteed to give the maximum fault coverage). In the case of the example shown in Fig. 9,

that maximum SLOTT coverage achievable by reordering of the inputs is 94%, while the stuck-fault coverage is 100%. To achieve 100% transition fault coverage, an extra dummy latch has to be used after reordering, which is still less than the three dummy latches required by the techniques presented in [8]. It should be noted that we have not considered any routing overhead that might arise in the implementation of this method, but since the input ordering is based on subcones, we believe the routing penalty is going to be much less severe compared with a technique based on full output cones. Finally, if the fault coverage attained by subcone separation is still unsatisfactory, we can always fall back on the use of dummy latches to enhance the transition fault coverage even further.

To illustrate the effectivensss of the coverage enhancement techniques presented in this section, consider the results presented in Tables I and II. All the ISCAS combinational circuits were simulated assuming the default input ordering in the network description file and a skewed load of pattern pairs for 100,000 random patterns. The undetectable faults were then tabulated. **Lmax** indicates the maximum number of levels in the logic circuit (each fanout branch is treated as a gate and assigned its own level). **Lslice** indicates the level at which the circuit was cut to create secondary outputs for input reordering. This was chosen so that most of the undetectable faults occurred at a level that was less than **Lslice.** Each primary input (PI) was then associated with a list of all secondary outputs in its forward cone (call this the *dependency list*). Then a greedy algorithm was used to reorder the PI's in the scan chain so that two consecutive PI's in the scan chain had as few common secondary outputs in the dependency list as possible (similar to the technique presented in [8] but without the use of dummy latches). The skewed-load fault coverages for 100,000 random patterns for this new ordering was then calculated. The SLOTT fault coverage figures shown under the column **Default** are for the default input ordering in the ISCAS description file, the ones under **Re-Ord** are using input separation based on subcones (based on the greedy algorithm). The transition fault coverages under the column **Indep** are the fault coverage figures assuming the pattern pairs are independent (and hence indicate the maximum fault coverage achievable). All the fault coverage figures are based on 100,000 random patterns. It can be seen that the fault coverage figures under **Re-Ord** indicate a significant improvement over those listed under **Default** and are, in many cases, very close to or equal to those listed under **Indep.** Thus, input separation based on subcones provides a viable alternative to the use of dummy latches for enhancement of SLOTT fault coverage. The fault coverage obtained by using this technique can be enhanced further by using dummy latches in only those portions of the circuit where undetectable faults still exist. In addition, it may be possible to further improve the SLOTT fault coverage by iterating over a few more values of **Lslice.**

TABLE II
OVERHEAD USING DUMMY LATCHES

Circuit	PIs	POs	Dummies	Overhead (%)
c432	36	7	35	77.78
c499	41	32	40	19.51
c880	60	26	29	5.00
c1355	41	32	40	19.51
c1908	33	25	32	21.21
c2670	233	140	10	0.00
c3540	50	22	49	54.00
c5315	178	123	0	0.00
c6288	32	32	31	0.00
c7552	207	108	182	35.75

Table II shows the overhead in terms of dummy latches in case the techniques listed in [8] were to be used. The number of PI's and primary outputs (PO's) in the ISCAS circuit are shown, and each of the PI's is assumed to originate from a shift register latch [1] and each of the PO's is assumed to feed an SRL. The overhead was calculated by first subtracting the number of latches fed by PO's from the latches listed under the **Dummies** column (assuming these latches are free to be used as dummy input latches as suggested in [8]) and then dividing by the number of latches fed by circuit PI's. Of course, if the number of SRL's fed by PO's is greater than or equal to those required for dummy latches, then no actual dummy latches have to be used. The overheads listed in Table II are based on the optimistic assumption that all SRL's fed by PO's are available for use in the place of dummy latches. Table II shows the overhead incurred to achieve the maximum possible SLOTT fault coverage. Thus, the proposed method of input separation based on subcones provides a viable alternative in case the cost of introducing dummy latches is considered prohibitive.

V. CONCLUSIONS

A skewed-load transition test (SLOTT) is both fast and cost-effective in scan-based designs. Most transitions can be generated using such a test. It has a relatively low complexity compared with other delay test schemes.

This paper studies a variety of issues associated with SLOTT. A single time-frame calculation of SLOTT pairs has been developed. This calculus is based on the Boolean difference technique and enables one to compute SLOTT pairs in small circuits. For the most part, this calculus enables one to understand the different issues associated with SLOTT, in particular the problem of undetectability due to shift dependency. The calculus features have been extended to allow for the computation of detection probabilities of transition faults. We have shown that the only difference between the method used for the computation of stuck-fault detection probabilities and transition fault detection probabilities using SLOTT is the inclusion of the displaced function in the calculation. The displaced function accounts for the shift dependency between the first and second vectors of the delay test pair. Extensions

to the Parker–McCluskey and the cutting algorithms were developed in this paper. These extensions included the effect of the displaced function.

Test coverage enhancement is another issue discussed in this paper. The test coverage enhancement is achieved by seeking a nearly optimal ordering of the inputs on the scan chain. We have shown that the input ordering plays a significant role in the attainable transition fault coverage. The input ordering tries to minimize the effect of shift dependency on the creation of valid SLOTT pairs. As shown in this paper, some orderings may suffer from low transition fault coverage, whereas others may achieve close to the possible maximum.

REFERENCES

[1] E. B. Eichelberger and T. W. Williams, "A logic design structure for LSI testability," *J. Design Automation and Fault-Tolerant Computing*, vol. 2, pp. 165–178, 1978.

[2] Z. Barzilai and B. Rosen, "Comparison of AC self-testing procedures," in *Proc. Int. Test Conf.*, Oct. 1983, pp. 89–94.

[3] G. L. Smith, "Model for delay faults based upon paths," in *Proc. Int. Test Conf.*, Nov. 1985, pp. 342–349.

[4] J. Savir and W. H. McAnney, "Random pattern testability of delay faults," in *Proc. Int. Test Conf.*, Sept. 1986, pp. 263–273.

[5] S. M. Reddy, C. J. Lin, and S. Patil, "An automatic test pattern generator for the detection of path delay faults," in *Proc. Int. Conf. Computer-Aided Design*, Nov. 1987, pp. 284–287.

[6] V. S. Iyengar, B. K. Rosen, and J. A. Waicukauski, "On computing the sizes of detected delay faults," *IEEE Trans. Computer-Aided Design*, vol. CAD-9, pp. 299–312, March 1990.

[7] E. S. Park, B. Underwood, T. W. Williams, and M. R. Mercer, "Delay testing quality in timing-optimized designs," in *Proc. Int. Test Conf.*, Oct. 1991, pp. 897–905.

[8] J. Savir and R. Berry, "At-speed test is not necessarily an AC test," in *Proc. Int. Test Conf.*, Oct. 1991, pp. 722–728.

[9] J. Savir, "Skewed-load transition test: Part I, calculus," in *Proc. Int. Test Conf.*, Baltimore, MD, Sept. 1991, pp. 705–713.

[10] S. Patil and J. Savir, "Skewed-load transition test: Part II, coverage," in *Proc. Int. Test Conf.*, Baltimore, MD, Sept. 1991, pp. 714–722.

[11] P. H. Bardell, W. R. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*. New York: Wiley Interscience, 1987.

[12] O. Bula *et al.*, "Gross delay defect evaluation for CMOS logic design system product," *IBM J. Res. Dev.*, vol. 34, pp. 325–338, 1990.

[13] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," *IEEE Trans. Computers*, vol. C-30, no. 3, pp. 215–222, Mar. 1981.

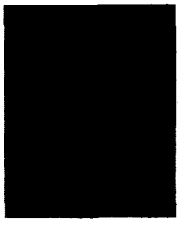
[14] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Computers*, vol. C-24, no. 6, pp. 668–670, 1975.

[15] J. Savir, G. S. Ditlow, and P. H. Bardell, "Random pattern testability," *IEEE Trans. Computers*, vol. C-33, no. 1, pp. 79–90, 1984.

[16] J. Savir, "Improved cutting algorithm," *IBM J. Res. Dev.*, vol. 34, pp. 381–388, 1990.

[17] F. Brglez and H. Fujiwara, "Neutral netlist of ten combinational benchmark circuits and a target translator in FORTRAN," in *Special session on ATPG and fault simulation, Proc. IEEE Int. Symp. Circuits and Systems*, June 1985.

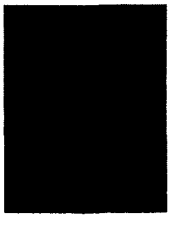
[18] J. A. Waicukauski, E. Lindbloom, B. K. Rosen, and V. S. Iyengar, "Transition fault simulation," *IEEE Design and Test of Computers*, vol. 4, pp. 32–38, Apr. 1987.



Jacob Savir (S'76–M'78–SM'90–F'93) received the B.Sc. and an M.Sc. degrees in electrical engineering from the Technion, Israel Institute of Technology, and the M.S. degree in statistics and the Ph.D. degree in electrical engineering from Stanford, University.

He is currently a Senior Engineer at IBM Microelectronics in East Fishkill, NY, and was previously a Senior Engineer at IBM Enterprise Systems in Poughkeepsie, NY, and a Research Staff Member at the IBM T. J. Watson Research Center, Yorktown Heights, N.Y. He has taught courses in electrical engineering, mathematics and computer science for many years. He taught at the Technion, Israel Institute of Technology; Singalovsky Technical Institute in Tel-Aviv, Israel; Pace University, NY; and SUNY Purchase, NY. His research interests lie primarily in the testing field, where he has published numerous papers and coauthored the text *Built-In Test for VLSI: Pseudorandom Techniques* (New York: Wiley, 1987). Other research interests include design automation, design verification, design for testability, statistical methods in design and test, fault simulation, fault diagnosis, and manufacturing quality.

Dr. Savir has received three IBM Invention Achievement Awards, four IBM Publication Achievement Awards, and three IBM Patent Application Awards. He is a member of Sigma Xi.



Srinivas Patil (S'86–M'90) received the B.Tech. degree in electrical engineering from the Indian Institute of Technology, New Delhi, India, in 1985, the M.S. degree in electrical engineering from the University of Iowa, Iowa City, IA, in 1987 and the Ph.D. degree in electrical engineering from the University of Illinois, Urbana, IL, in 1990.

He has been with IBM Corporation since 1991 and is currently with the IBM Test Design Automation group in Endicott, NY. His research interests include deterministic test generation, simulation, random pattern test and BIST, parallel algorithms and architectures.