Efficient Techniques for Transition Testing

XIAO LIU and MICHAEL S. HSIAO
Virginia Tech
and
SREEJIT CHAKRAVARTY and PAUL J. THADIKARAN
Intel Corporation

Scan-based transition tests are added to improve the detection of speed failures in sequential circuits. Empirical data suggests that both data volume and application time will increase dramatically for such transition testing. Techniques to address the above problem for a class of transition tests, called enhanced transition tests, are proposed in this article.

The first technique, which combines the proposed transition test chains with the ATE repeat capability, reduces test data volume by 46.5% when compared with transition tests computed by a commercial transition test ATPG tool. However, the test application time may sometimes increase. To address the test time issue, a new DFT technique, Exchange Scan, is proposed. Exchange scan reduces both data volume and application time by 46.5%. These techniques rely on the use of hold-scan cells and highlight the effectiveness of hold-scan design to address test time and test data volume issues. In addition, we address the problem of yield loss due to incidental *overtesting* of functionally-untestable transition faults, and we formulate an efficient adjustment to the algorithm to keep the overtest ratio low. Our experimental results show that up to 14.5% reduction in overtest ratio can be achieved, with an average overtest reduction of 4.68%.

Categories and Subject Descriptors: B.8.1 [**Performance and Reliability**]: Reliability, Testing, and Fault-Tolerance; I.1.2 [**Symbolic and Algebraic Manipulation**]: Algorithms; J.6 [**Computer-Aided Engineering**]

General Terms: Algorithms, Reliability

Additional Key Words and Phrases: Test application time reduction, test chain, test data volume reduction, transition faults, yield loss

1. INTRODUCTION

Higher clock rate, shrinking geometries, increasing metal density, and so forth are resulting in delay defects which cause speed failures. Delay defects in a circuit do not change the logic behavior of the circuit, but affect the speed at

This research was supported in part by a grant from Intel Corp., and in part by NSF under contract CCR-0196470.
A preliminary version of this work appeared in ITC 2002.
Authors' addresses: X. Liu, 8215 Southwestern Blvd., #1015, Dallas, TX 75206; email: x-liu@ti.com.

which the circuit can operate. Although the conventional stuck-at (s@) testing can uncover some delay defects, the stuck-at fault model [Eldred 1959] does not model speed related failures very well. This has prompted researchers to propose a variety of fault models for speed failures, that is, *transition fault* [Waicukauski et al. 1987], *path delay fault* [Smith 1985], and *segment delay fault* [Heragu et al. 1996].

A transition fault at node X assumes a large delay at X such that the transition at X will not reach the latch or primary output within the clock period. The path delay fault model considers the cumulative effect of gate delays along a specific path, from a primary input to a primary output. If the cumulative delay exceeds the slack for the path, then the chip fails. Segment-delay fault targets path segments instead of complete paths. Among these models, the transition fault model is the most practical, and commercial tools are available for computing such tests. The number of transition faults is linear to the number of circuit lines, while the number of path delay faults are exponential to the number of circuit lines, which makes the need for critical path analysis and identification procedure a necessity. Transition tests have been generated to improve the detection of speed failures in microprocessors [Tendulkar et al. 2002] as well as ASICs [Hsu et al. 2001].

At each line in the circuit, two transition faults are possible: *slow-to-rise* and *slow-to-fall*. Test pattern for a transition fault consists of a pair of vectors {V1,V2}, where V1 (*initial vector*) is required to set the target node to an initial value, and V2 (*test vector*) is required to launch the appropriate transition at the target node and also propagate the fault effect to a primary output [Waicukauski et al. 1987; Savir and Patil 1993].

Transition tests can be applied in three different ways: *Broadside* [Savir and Patil 1994], *Skewed-Load* [Savir and Patil 1994], and *Enhanced-Scan* [Dervisoglu and Stong 1991]. For broadside testing (also called functional justification), a vector is scanned in and the functional clock is pulsed to create the transition and subsequently capture the response. For each pattern in broadside testing, only one vector is stored in tester scan memory. The second vector is derived from the first by pulsing the functional clock. For skewed-load transition testing, an N-bit vector is loaded by shifting in the first N-1 bits, where N is the scan chain length. The last shift clock is used to launch the transition. This is followed by a quick capture. For skewed-load testing also, only one vector is stored for each transition pattern in tester scan memory; the first vector is a shifted version of the stored vector. Finally, for enhanced scan transition testing, two vectors (V1, V2) are stored in the tester scan memory. The first scan shift loads V1 into the scan chain. It is then applied to the circuit under test to initialize it. Next, V2 is scanned in, followed by an apply, and subsequently a capture of the response. During the shifting in of V2, it is assumed that the initialization of V1 is not destroyed. Therefore, enhanced scan transition testing assumes a hold-scan design [Dervisoglu and Stong 1991].

Among the three kinds of transition tests, broadside suffers from poor fault coverage [Savir and Patil 1994]. Since there is no dependency between the two vectors in enhanced scan, it can give better coverage than the skewed-load transition test. Skewed-load transition tests also lead to larger test data volume.

Table I. Test Data Volume Comparison

Name	S@ Test	Trans Test	Expansion
c3540	179	366	4.09
c5315	124	248	4
c6288	36	80	5
c7552	237	378	3.19
s13207	485	1002	4.13
s15850	464	924	3.98
s35932	75	137	3.65
s38417	1017	1927	3.79
s38584	727	1361	3.74

Compared to stuck-at tests, the increase in the number of vectors required for enhanced scan to get complete coverage is about 4X (Table I). This data, collected using a state-of-the-art commercial ATPG tool, shows the number of stuck-at vectors and the number of enhanced scan transition patterns for each circuit. Note that each transition pattern consists of two vectors. For the skewed-load transition test, it has been observed that the data volume for an ASIC has an increase of 5.9X [Hsu et al. 2001].

A drawback of enhanced scan testing is that it requires hold-scan scan cells. However, in microprocessors and other high performance circuits that require custom design such cells are used for other reasons. In custom designs, the circuit often is not fully decoded, and hold-scan cells are used to prevent contention in the data being shifted, as well as preventing excessive power dissipation in the circuit during the scan shift phase. Furthermore, if hold-scan cells are used, the failing parts in which only the scan logic failed can often be retrieved, thus enhancing, to some extent, the diagnostic capability associated with scan DFT. In this article, we will consider only enhanced scan transition tests.

A number of techniques on reducing test data volume and test application time for single cycle scan tests have been presented in the literature [Keller et al. 2001; Hamzaoglu and Patel 1999; Koeneman 1991; Chandra and Chakrabarty 2001; Lee et al. 1998; Das and Touba 2000]. These methods assume that only 5–10% of the bits are fully specified. Unspecified bits are filled to identify the easy-to-detect faults. Different codes to compress the information in the tester and decompressing them on chips [Keller et al. 2001; Chandra and Chakrabarty 2001; Das and Touba 2000], or using partitioning and applying similar patterns to the different partitions [Hamzaoglu and Patel 1999; Lee et al. 1998] have been proposed. The techniques proposed here compliment the work on compressing individual vectors.

In our previous work [Liu et al. 2003], we presented two algorithms for computing transition test patterns from generated s@ test vectors which can reduce the test set size by 20%. Nevertheless, there has not been much work on addressing the explosion in data and application time for transition tests. To tackle this problem, we first propose novel techniques to generate effective transition test chains based on stuck-at vectors, thus the need for a separate transition ATPG is eliminated. Next, we propose methods to reduce the transition test

data volume and test application time. Our optimization techniques consider factors across test patterns for transition tests. The first technique uses the ATE repeat option to reduce the test data volume and requires the use of transition test chains, rather than the random application of the individual transition test patterns. We describe the transition test chains and present a novel algorithm for computing such chains. Experimental results show an average data volume reduction of 46.5%, compared to the conventional enhanced scan transition test data volume computed by COM, a commercial ATPG tool. This technique does not necessarily decrease test application time. To reduce test application time, a new DFT technique called exchange scan is proposed . Combining exchange scan with transition test chains reduces both the test application time and test data volume by 46.5%, compared to a conventional transition test set computed by COM.

Nevertheless, one of the drawbacks of scan-based delay tests is possible yield loss due to overtesting. In Rearick [2001], the author showed that scan-based testing may fail a chip due to the delay faults that do not affect the normal operation, and thus it is unnecessary to target those functionally unsensitizable faults. In Lai et al. [2000a, 2000b], the authors addressed the impact of delay defects on the functionally untestable paths on the circuit performance. Moreover, a scan test pattern, though derived from targeting functionally testable transition faults, can incidentally detect functionally untestable faults if the starting state is an arbitrary state (could be an unreachable state). In this article, we use a low-cost implication engine to compute the functionally untestable faults and address the problem of overtesting in our graph-formulated transition ATPG algorithm.

The rest of the article is organized as follows. In Section 2, the ATE model we use is described. Section 3 proposes a novel transition test chain formulation which is mapped into a weighted transition-pattern-graph traversal problem. A new DFT technique to reduce test application time by reducing the number of scan loads is presented in Section 4. Section 5 describes a novel method to address the problem of incidentally overtesting of functionally untestable faults in our ATPG algorithm. Section 6 presents a summary of all the experimental results. Finally, Section 7 concludes the article.

2. ATE MODEL

Figure 1 is an abstraction of the tester model we use. ATE storage consists of two parts: scan and control memory. Scan memory is divided into several channels. Each channel consists of three bits. For each clock cycle of the scan shift operation, the scan memory contains the bit to be scanned in, the expected response bit from the circuit under test (CUT), and an indication as to whether or not this bit of the response is to be masked, indicated by M or U in Figure 2. Figure 2(a) shows the data stored for a single scan channel for the test set {V1, V2, V3, V4, V5, V6, V7, V8, V9, V10}, with Rj the expected response for Vj. The control memory controls the shift and the comparison operation. The scan memory depth required is (N+1)*S bits for a test set of size N and scan length S.

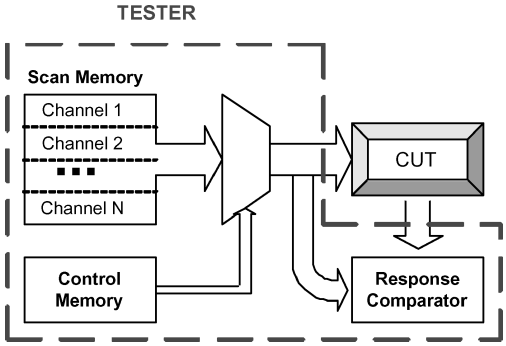

Fig. 1. Tester memory model.

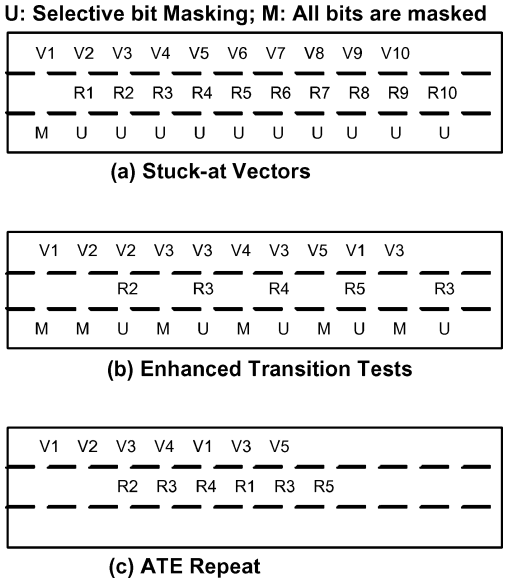

Fig. 2. ATE storage model.

The enhanced scan transition test set in Table II consists of 6 transition test patterns. Each pattern consists of a pair of vectors and the expected response to the test vector. The first pattern in our example consists of the pair (V1, V2) and the response R2 to V2. Storage of the test data in the scan memory is shown in Figure 2(b). The storage depth required is N*2*S+N*S bits. The control sequence for this test set, shown in Table III, is very repetitive and stored in the tester scan memory. Row 1 of the Table states that V1 is scanned in and applied to the CUT. Row 2 states that V2 is scanned in, applied to the CUT, and the response R2 captured. Row 3 states that the first vector of the next vector pair, that is V2, is scanned in, while the response R2 of the previous test pattern is scanned out and compared against the expected response. Once the scan operation is complete, the new vector is applied to the CUT. The rest of the entries can be similarly interpreted.

Table II. Enhanced Scan Transition Test Set Example

Vector1	Vector2	Response
V1	V2	R2
V2	V3	R3
V3	V4	R4
V3	V5	R5
V1	V3	R3*
V4	V3	R3**

Table III. Control Sequence of Transition Test

(i) Shift in V1; (ii) Apply;
(i) Shift in V2; (ii) Apply; (iii) Capture;
(i) Shift in V2; Shift Out and Compare R2; (ii) Apply;
(i) Shift in V3; (ii) Apply; (iii) Capture;
(i) Shift in V3; Shift Out and Compare R3; (ii) Apply;
(i) Shift in V4; (ii) Apply; (iii) Capture;
(i) Shift in V3; Shift Out and Compare R4; (ii) Apply;
(i) Shift in V5; (ii) Apply; (iii) Capture;
(i) Shift in V1; Shift Out and Compare R5; (ii) Apply;
(i) Shift in V3; (ii) Apply; (iii) Capture;
(i) Shift in V4; Shift Out and Compare R3*; (ii)Apply;
(i) Shift in V3; (ii)Apply; (iii) Capture;
(i) Shift out and compare R3**;

3. ATE REPEAT AND TRANSITION TEST CHAINS

3.1 ATE Repeat

There is considerable redundancy in the information stored in the tester. In Figure 2(b), V2, V3 are used several times in the test sequence. Ideally, one copy of the information should suffice. However, storing one copy of a vector and reusing it in any random order requires the ATE to index into random locations in the scan memory, an operation which is currently not available. Limited reuse of the information stored, however, is possible. In Figure 2(b), two copies of V2 are stored in consecutive locations in the scan memory. It is possible to store just one copy of V2 and scan in V2 as often as possible during consecutive scan cycles. Similarly, we can replace two copies of V3 in locations 4, and 5, from the left side of the scan memory in Figure 2(b), with just one copy of V3. Further reduction of the number of copies of V3 is not possible. Thus, we store the sequence {V1, V2*, V3*, V4, V3, V5, V1, V3} and repeatedly scan in the vectors marked with the flag *. Information about vectors that need to be scanned in multiple times is stored in control memory. Thus, 8 instead of 10 vectors need to be stored.

In the preceding example, the scan storage requirement was reduced at a price. Since vectors that are scanned in repeatedly do not form a regular pattern, the control memory requirement can increase drastically. To avoid such an increase in control memory, we impose a restriction that, except for the first vector and last vector stored in the scan memory, every vector is scanned in

exactly twice. In Figure 2(c), the sequence {V1, V2, V3, V4, V1, V3, V5} is stored. Assuming that all but the first and last vectors are scanned in twice, the set of transition test patterns applied is (V1, V2,), (V2, V3), (V3, V4), (V4, V1), (V1, V3,), (V3, V5). This set of patterns includes all the test patterns of Figure 2(a) as well as (V4, V1). Thus, by storing only 7 vectors (instead of 10 vectors), all the transition tests can be applied. For this example, not only is the control memory requirement lower, but the number of vectors stored is also reduced. Sequences in which all but the first and last vectors are scanned in twice are defined as *transition test chains*.

3.2 Transition Test Chains via Weighted Transition Graph

Computing transition test chains is different from computing a set of transition test patterns as is conventionally done. A novel algorithm, called *weighted transition graph algorithm*, to compute such chains is discussed next.

The algorithm constructs transition test chains from a given stuck-at test set. Instead of computing a set of vector pairs and chaining them together as alluded to in the preceding examples, the problem is mapped into a graph traversal problem. The algorithm builds a weighted directed graph called the *weighted transition-pattern graph*. In this graph, each node represents a vector in the stuck-at test set; a directed edge from node Vi to Vj denotes the transition test pattern (Vi, Vj), and its weight indicates the number of transition faults detected by (Vi, Vj). Directed edges may potentially exist between every node pair, resulting in a large (possibly complete) graph. In order to reduce the time required to construct the graph, only the subset of the faults missed by the original stuck-at test set are considered. The graph construction procedure is discussed next.

We start with the stuck-at test set T = $\{T_1 \ldots T_N\}$.

(1) Perform transition fault simulation using the stuck-at test set as a transition test set $\{(T_1, T_2), (T_2, T_3) \ldots (T_{N-1}, T_N)\}$ to compute *undet*, the set of undetected transition faults.

(2) Deduce the subset U of the stuck-at faults implied by *undet* as follows. If X slow-to-rise or slow-to-fall fault $\in$ *undet*, then both X stuck-at-0 and stuck-at-1 are included in U.

(3) Perform stuck-at fault simulation, without fault dropping, using the stuck-at test set T on the stuck-at faults in U. For each stuck-at fault f in U, record the vectors in T that excite f and the vectors that detect f. Also, for each vector, the faults excited and detected by it are recorded.

(4) The weighted directed graph contains a node corresponding to each of the stuck-at tests in T. The directed edge, from V_i to V_j, is inserted if the corresponding test pattern (T_i, T_j) detects at least one transition fault in *undet*. The weight of (T_i, T_j) is the number of transition faults in *undet* detected by (T_i, T_j).

For example, consider a circuit with 5 gates (10 stuck-at faults) and a stuck-at test set consisting of 4 vectors V1, V2, V3, and V4. The excitation and detection dictionary obtained by simulation without fault dropping are shown in Table IV.

Table IV. Fault Dictionary Without Fault-Dropping

Vector 1	*Excited Faults*	*Detected Faults*
V1	a-s-0, b-s-1, c-s-1, d-s-0, e-s-0	a-s-0, b-s-1
V2	b-s-1, c-s-0, d-s-0, e-s-1	c-s-0, d-s-0, e-s-1
V3	a-s-1, c-s-1,	a-s-1, c-s-1
V4	a-s-1, b-s-0, d-s-1, e-s-0	b-s-0, d-s-1, e-s-0

Assuming the test set order to be {V1, V2, V3, V4}, only 3 transition faults (slow-to-fall at c, e and slow-to-rise at c) are detected. However, using Table IV, we can make the following observations by combining nonconsecutive vectors: (V1, V3) detects a slow-to-fall; (V3, V1) detects a slow-to-rise; (V1, V4) detects d slow-to-fall; (V4, V2) detects d slow-to-rise; (V4, V1) detects a slow-to-rise, b slow-to-fall; and (V2, V4) detects b slow-to-rise, e slow-to-rise and d slow-to-fall. This results in the transition-pattern graph of Figure 4 .

Unlike general graphs, this weighted transition graph has a specific property which formulates the following theorem.

THEOREM 1. *Faults detected by pattern (V_i, V_j), and faults detected by pattern (Vj, Vk) are mutually exclusive.*

PROOF. We prove this by contradiction. Without loss of generality, consider fault f, *slow-to-fall*, detected by (V_i, V_j). Thus, V_i excites f *s-a-0* (sets line f to 1), and V_j detects f *s-a-1*. Assume (V_j,V_k) also detects f, *slow-to-fall*. Then, the initial vector V_j must set line f to 1, which is a contradiction. □

An Euler trail in the transition-pattern graph traverses all the edges in the graph exactly once. By inserting a minimal number of edges to the graph, an Euler trail that traverses all edges would tempt us to think that this is the best test chain. However, this actually only leads to a suboptimal solution. Traversing edge (Vi, Vj) is equivalent to selecting test (Vi, Vj). Once edge (Vi, Vj) is traversed, that is test (Vi, Vj) is selected, it detects a number of transition faults. This alters the weights on other edges and even removes some of the edges. Per Theorem 1, edges whose weights do not change are those originating from Vj. To improve the solution, the edge weights should be updated after traversing each edge. A preliminary version of the algorithm is outlined in Figure 3, where P is the transition test chain computed by the algorithm from the given stuck-at test set T.

The idea behind this algorithm is as follows. We are looking for the smallest test sequence that can cover all the detectable faults by traversing the weighted transition-pattern graph. In a traditional weighted graph, traversing any edge in the graph will *not* affect the weight on other edges; therefore, an Euler trail, which traverse each edge exactly once, will be the optimum solution. But in our case, traversing any edge in the weighted transition-pattern graph may result in the dection and removal of number of transition faults from the fault dictionary. Thus, the weight on other edges, which also detect these faults, will be altered and must be updated by simulation.

For example, in the original weighted transition-pattern graph on the left-hand side of Figure 4, (V2, V4, V1) is the heaviest-weight test chain of length 3.

```
construct the weighted transition pattern graph;
initialize P to T={T1...Tn}
while(TFC<100%) &&(iteration num<MAX))
    BEGIN
      identify an edge(Vi,Vj) with the heaviest weight;
      append vectors Vi,Vj to set set P;
      for all the edges starting from Vj
          BEGIN
            look for the edge (Vj, Vk) having the heaviest weight;
            append vector Vk to the test set P;
          END
     END
```

Fig. 3. Generic weighted transition graph algorithm.

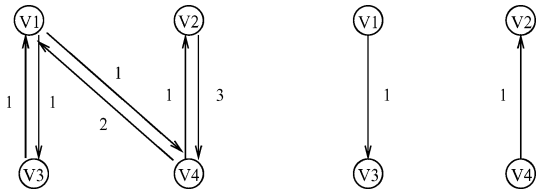

Fig. 4. Weighted transition-pattern graph example.

After traversing this chain, 5 transition faults (a slow-to-rise, b slow-to-rise, b slow-to-fall, d slow-to-fall, and e slow-to rise) have been detected. The updated graph is shown on the right-hand side of Figure 4. It should be noted that, in addition to removing the edges (V2, V4) and (V4, V1), the other two edges ((V1, V4) and (V3, V1)) are also removed from the graph. This is because the fault (d slow-to-fall), which can be detected by (V1, V4), has been detected by selecting the test chain (V2, V4, V1). Therefore, the edge (V1, V4) can be removed from the weighted pattern graph because it has no contribution to the future selection of edges. A similar idea can be applied on the edge (V3, V1) as well. Finally, all the 7 undetected faults in Table IV are detected with the test chain {V2, V4, V1, V3, V4, V2}.

Several optimizations were applied to the generic version of the algorithm to improve the results. First, we investigated the impact of different lengths of test chain on the number of faults detected. Instead of considering one edge at a time, we use Theorem 1 to inspect path segments of length 3. This reduces the amount of simulation required, and a transition test chain is generated by incrementally concatenating the vector chains of length 3. While this solution is better than simply concatenating vector pairs for each of the remaining undetected transition faults, it still may not be optimal. When the test chain length increases beyond 3, the difference between the actual number of transition faults detected by the chain and the sum of edge weights in the chain can increase. This difference determines whether it is worthwhile to continue extending the chains. Thus, using longer chains reduces the number of graph updates (hence reducing the run-time of the algorithm), but it will increase the

size of the final solution. Our experiments suggest that the chain lengths of 3 or 4 give the best results.

Secondly, expanding on the essential test definition for stuck-at fault from Hamzaoglu and Patel [1999], we define an essential vector for transition faults to be any test vector that excites (or detects) at least one transition fault that is not excited (or detected) by any other test vector in the test set. All essential vectors must occur in the transition test chain at least once. We include essential vectors early in the chaining process. The transition test chain generation process is divided into two phases by constructing two weighted transition-pattern graphs.

(1) Identify all essential vectors, generate the transition-pattern graph using only essential vectors and construct the test chains with only the essential vectors in the graph. Append that to the initial transition test chain P in the generic version of the weighted transition graph algorithm.
(2) Reduce the number of faults by dropping faults detected in the first step. Generate the transition-pattern graph for the remaining faults and extend the partial transition test chain from the previous Step as described in Step 3 of the generic weighted transition graph algorithm.

The number of edges in the second step of the modified algorithm is significantly reduced because most of the edges incident on the essential vectors have been traversed in the previous Step and thus removed.

During the test pattern generation procedure, some of the faults detected by the earlier test vectors may also be detected by the test patterns generated later. Therefore, vectors added early in our transition test set might become redundant. To identify such redundant patterns, we perform a *reverse-order pair-wise compaction*. After $(U_1, U_2, U_3), (U_4, U_5, U_6), \ldots, (U_{n-2}, Un-1, U_n)$ are generated, they are appended to the original stuck-at test set in that order. The test patterns $(U_{n-1}, U_n), (U_{n-2}, U_{n-1}), \ldots, (U_1, U_2)$ are simulated in the reverse order from which they were generated. If neither $(U_{i-1}, U_i), (U_i, U_{i+1})$ detects any additional fault, then U_i is redundant and can be eliminated. Note that eliminating a vector U_i will give rise to a new transition test pattern (U_{i-1}, U_{i+1}); therefore, we follow this by performing a forward-order pair-wise compaction step to further reduce the size of the test sequence.

4. EXCHANGE SCAN

We saw that by using transition test chains and ATE repeat, we can reduce the test data storage. Data presented in Section 6 will show the average reduction to be about 46.5%. However, test application time can sometimes increase due to repeating on every vector in the chain. To reduce test application time while still retaining the improvement in data storage, we propose a new DFT technique.

Instead of using the ATE repeat option to re-use the vector, a low overhead alternative is possible in which each vector is only scanned in once. Reducing scan-in operations reduces test application time. The net result is a reduction in both the data storage requirement and the test application time.

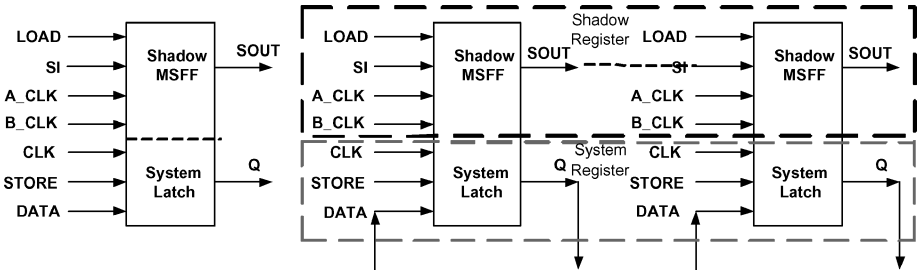

Fig. 5. Block diagram of a hold-scan system.

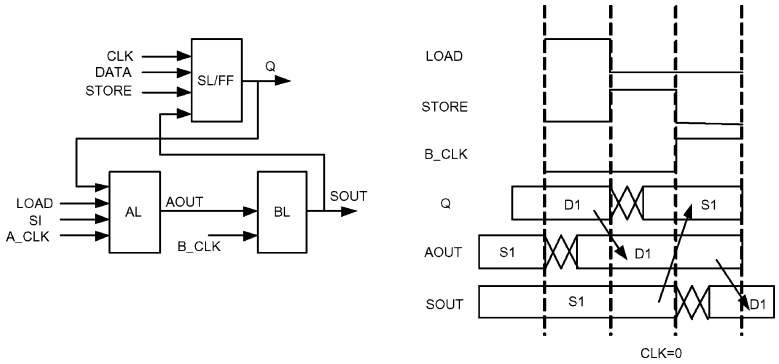

Fig. 6. Hold scan cell and exchange scan timing.

The block diagram of a conventional hold-scan system is shown in Figure 5. The scan cells, each of which consists of two parts, System Latch and Shadow MSFF, are chained together to form two related registers: SYSTEM REGISTER and SHADOW REGISTER. During normal operation, the SYSTEM REGISTER is in use and the SHADOW REGISTER does not play any role. For scan testing options, three scan operations—SCAN_SHIFT, SCAN_LOAD, and SCAN_STORE—are supported.

Assume the scan cell implementation of Figure 6. For SCAN_SHIFT, the A_CLK and B_CLK are pulsed so that data passes from the SI to the SOUT. For SCAN_STORE, the STORE signal is pulsed to transfer the data from SOUT to Q. The content of SHADOW REGISTER is transferred to the SYSTEM REGISTER. In SCAN_LOAD, the contents of the SYSTEM REGISTER is transferred to the SHADOW REGISTER. Pulsing LOAD transfers data from Q to AOUT. Pulsing the B_CLK transfers the data from AL to BL.

The new operation SCAN_EXCHANGE exchanges the data between the SHADOW and the SYSTEM registers, without destroying either of them. Pulsing LOAD transfers the contents of SL to AL. Pulsing STORE transfers the contents of BL to SL. Pulsing B_CLK to transfer the content of AL to BL follows this. The corresponding timing diagram is shown in Figure 6(b). No additional hardware or signal is needed to support the exchange operation. It may require the global scan controller to be modified slightly to realize the exchange operation. The exchange operation takes about three clock cycles.

WITHOUT EXCHANGE	WITH EXCHANGE
Scan in V1	Scan in V1
Store V1	Store V1
Scan in V2	Scan in V2
Store V2	Store V2
Capture R2	Capture R2
Load R2	Exchange (R2, V2)
Scan in V2, Scan out R2	Scan in V3, Scan out R2
Store V2	Store V3
Scan in V3	Capture R3
Store V3	Exchange (R3, V3)
Capture R3	Scan in V4, Scan out R3
Load R3	Store V4
Scan in V3, Scan out R3	Capture R4
Store V3	Exchange (R4, V4)
Scan in V4	Scan in V5, Scan out R4
Store V4	
Capture R4	
Scan in V4, Scan out R4	

Fig. 7. Scan operation with/without exchange.

SCAN_EXCHANGE, for the transition test chain {V1, V2, V3, V4} is used as follows. Test pairs applied are (V1, V2), (V2, V3), (V3, V4). The expected response on application of V2 is R2, V3, is R3, and V4 is R4. The sequence of operations, without exchange scan, is shown in the first column of Figure 7. *Capture R2* implies that the response of the CUT on application of V2 is latched onto the SYSTEM REGISTER. Scan in V2, Scan Out R2 implies SCAN_SHIFT, where V2 is scanned out and the response of the CUT, from the previous pattern, is compared to R2.

The second column of Figure 7 shows the sequence of operations using SCAN_EXCHANGE. Once V2 is loaded into the SHADOW REGISTER, the subsequent store and capture operations do not destroy the contents of the SHADOW register. So, the SCAN_LOAD operation that destroys the contents of the SHADOW REGISTER is replaced by the SCAN_EXCHANGE operation. It exchanges the contents of the SHADOW REGISTER and the SYSTEM REGISTER. Thus, the captured response is transferred to the SHADOW REGISTER, and V2 is applied to the circuit under test as the *initial* vector for the next test pair. We can, therefore, skip the sequence of operations that rescans V2 and stores it. In addition, the SCAN_SHIFT of the response from V2, that is R2, can now be merged with the SCAN_SHIFT of the final vector of the

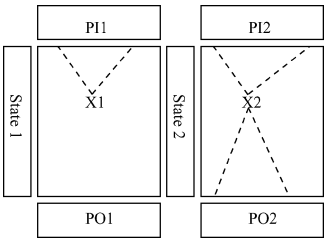

Fig. 8. Arbitrary starting states in enhanced scan.

next pattern V3. The net effect of this is that we can replace an entire scan operation with a 3-cycle SCAN_EXCHANGE operation. Considering that the SCAN_SHIFT operation may take 1000 or more clock cycles, this overhead of the SCAN_EXCHANGE operation is negligible and will be neglected from our calculations. Note that transition test chains, but not ATE repeat capability in the testers, is required to realize the gains of the exchange scan operation. If the ATE repeat capability is available, each of the vectors that are stored can be compressed as discussed in Keller et al. [2001], and the benefits of transition test chains can be realized using exchange scan. Our experimental result show that both test data volume and test application time decrease by 46.5%, compared to a commercial tool.

5. CONSTRAINED ATPG TO MINIMIZE OVERTESTING

In this section, we present a novel algorithm to address the possible yield loss due to overtesting of functionally untestable faults in enhanced-scan testing. A transition fault is functionally untestable if either launching of the transition, or the propagation of its effect, is impossible in the functional mode due to constraints imposed by the circuit. Such constraints include the requirement for an illegal/unreachable state in the test pattern, or the two state combination in the enhanced-scan pattern is functionally impossible. Because the enhanced-scan model assumes total independence between the two vectors in the test pattern, some functionally untestable transition faults may become detected. Figure 8 illustrates the scenario. Since an enhanced-scan pattern consists of (State 1, V1, State 2, V2), if either (1) State 1/State 2 is illegal, or (2) the State 1/State 2 pair is not a valid state transition combination, the transition pattern may detect some of the functionally untestable faults. To avoid detection of such functionally untestable faults, we must make sure that these two scenarios do not arise.

We first describe a low cost implication-based, functionally-untestable fault identification method without involving any sequential ATPG. Then we try to minimize the overtesting of these functionally untestable faults in our graph-formulated ATPG algorithm described in Section 3.

In general, identifying functionally untestable fault in sequential circuits is of the same complexity as sequential ATPG. In Hsiao [2002], a method for identifying untestable stuck-at faults in sequential circuits by maximizing local conflicting value assignments has been proposed. The technique first computes a large number of logic implications across multiple timeframes and stores them

in an implication graph. Then the algorithm identifies impossible combinations of value assignments locally around each gate in the circuit and those redundant stuck-at faults requiring such impossibilities.

For identifying functionally assignments untestable transition faults, in addition to searching for the impossibilities locally around each gate, we also check the excitability of the initial value in the previous timeframe. In other words, if opposing values on a signal in two consecutive timeframes are not possible, we would know that no transition can be launched on that signal in the functional mode. Furthermore, if the corresponding s@ fault becomes unobservable due to the imposed constraints, the transition fault also would be unobservable. In employing this technique, we can identify a large set of untestable transition faults in the circuit.

The identified functionally-untestable transition faults are mapped onto the graph traversal problem. We build the weighted transition-pattern graph for the given stuck-at test set as before. Again, in the graph, each node represents a vector in the stuck-at test set, and a directed edge from Vi to Vj denotes the transition test pattern (Vi, Vj). To address the problem of overtesting functionally-untestable faults, each edge in the weighted transition-pattern graph has *two* weights: W1 indicates the number of functionally-untestable faults detected by test pattern (Vi, Vj), and W2 represents the number of functionally-testable faults detected by the pattern. Therefore, our target is to achieve the highest transition fault coverage, while minimizing the overtesting of functionally-untestable faults.

The heuristic we used to minimize the overtesting is presented next. Assume the stuck-at test set T = $T_1 \ldots T_n$ is given.

(1) Identify the functionally-untestable fault set R using the transition implication tool.
(2) Perform transition fault simulation using the stuck-at test set as a transition test set $\{(T_1, T_2), (T_2, T_3) \ldots (T_{N-1}, T_N)\}$ to compute *undet*, the set of undetected transition faults.
(3) Deduce the subset U of the stuck-at faults implied by *undet* as follows. If X slow-to-rise or slow-to-fall fault $\in$ *undet*, then both X stuck-at-0 and stuck-at-1 are included in U.
(4) Categorize the transition faults in U into two parts: $U_1 = U \cap R$, standing for the functionally-untestable fault set in U, and $U_2 = U - U_1$ standing for the functionally-testable fault set in U.
(5) Build the fault dictionary for U_1 and U_2 respectively, using the fault simulation without fault-dropping. And generate the weighted pattern graph with W_1 and W_2 on each edge.
(6) Compute the weight W on each edge, using the formula W = F(W_1, W_2, threshold). For small circuits, we set the threshold to zero and for bigger circuits, we predetermined the threshold as a small fraction of the number of functionally-untestable faults in the circuit.
(7) Greedily construct transition test chains with the maximum W and append the transition test chains to the original stuck-at test set T.

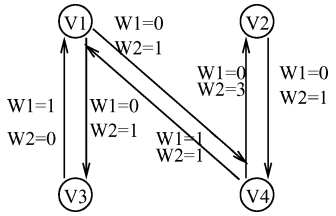

Fig. 9. Weighted pattern graph with two weights.

Function F is defined as follows:

$$F = \begin{cases} W_2 - W_1 & \text{if } W_1 \leq threshold \\ 0 & \text{otherwise} \end{cases}$$

Let us look at the example in Section 3 again. Suppose the transition fault, a slow-to-rise, is a functionally-untestable fault, then the weighted pattern graph in Figure 4 will be modified to Figure 9. Again, recall that every vertex in the figure is a test vector, and there exists an edge between two vertex if they detect at least one fault. Different from the previous example, every edge now has two weights, namely W_1 and W_2, which are the number of functionally-testable fault and functionally-untestable faults detected by the pair of test vectors, respectively. So, from the fault dictionary, we can see that (V4, V1) detect one functional testable fault (b slow-to-fall) and one functionally untestable fault (a slow-to-rise). Therefore, the weights on edge (V4, V1) are $W_1 = 1$ and $W_2 = 1$. Similarly, (V3, V1) detects only a slow-to-rise which is a functionally-untestable fault; correspondingly, the weights on edge (V3, V1) are $W_1 = 1$ and $W_2 = 0$.

Therefore, selecting a transition test chain no longer depends on merely the total number of transition faults it can potentially detect. We must distinguish the number of functionally-testable faults and the number of functionally-testable faults each chain can detect. For instance, consider the transition test chain {V2, V4, V1, V3, V4, V2} generated in Section 3. Although all the functionally-testable faults will be detected, the functionally-untestable fault (a slow-to rise) will also be detected. To avoid the overtesting of this functionally-untestable fault, we generate the transition test chain by avoiding traversal of the edges containing nonzero-W_1. In doing so, the new transition test chain {V2, V4, V2, V1, V3} (that avoids traversing nonzero W_1 edges) will now only detect 5 out of the 6 remaining functionally-testable faults because all the vectors that can detect b slow-to-fall also detect the functionally-untestable fault a slow-to-rise. Consequently, a slight drop in fault coverage of the functionally-testable faults may result.

We can also relax this condition of avoiding all nonzero W_1 edges to increase coverage of functionally-testable faults. In essence, we set a threshold on W_1 such that we will still consider some nonzero W_1 edges, but we make sure that detection of such untestable faults is bounded by the threshold value.

Table V. Results with Different Chain Lengths

	2		3		4		5		6		7	
Circuit	*ID*	*AC*	*ID*	*AC*	*ID*	*AC*	*ID*	*AC*	*ID*	*AC*	*ID*	AC
s344	38	38	64	64	90	72	111	89	143	89	164	103
s382	24	24	43	43	63	44	75	56	92	61	111	61
s832	60	60	87	87	114	89	136	95	158	95	180	103
s1196	47	47	78	78	116	93	146	105	177	105	208	119
s1423	36	36	60	60	79	79	106	95	129	118	153	124
s5378	59	59	103	103	149	142	190	144	235	172	284	176
s35932	1072	1072	1175	1175	1270	1247	1382	1261	1475	1312	1564	1334
s38417	132	132	190	190	274	249	380	285	439	296	500	319

Table VI. Results with/without Essential Vectors

		Without essential vectors				With essential vectors			
	S@	*Tran.*	*Comp.*	*Time*	*TFC*	*Tran.*	*Comp.*	*Time*	TFC
Circuit	Set	*Tests*	*Tests*	(*s*)	(%)	*Tests*	*Tests*	(*s*)	(%)
c1355	198	928	285	3.51	99.77	915	270	2.82	99.77
c1908	143	918	318	4.57	99.67	966	298	3.32	99.67
c3540	202	1222	515	25.43	96.27	1181	514	22.06	96.27
c5315	157	816	342	11.80	99.54	762	313	9.79	99.54
c6288	141	310	122	5.60	99.19	334	120	5.70	99.19
s344	31	135	63	0.37	100	207	64	0.37	100
s832	179	988	310	4.36	99.20	937	292	2.78	99.20
s1196	197	1004	362	5.24	99.97	1022	358	4.38	99.97
s1423	97	566	186	2.25	99.11	528	177	2.09	99.11
s5378	332	1672	722	35.73	98.40	1685	722	29.76	98.40
s35932	78	542	196	133.13	90.50	633	197	133.01	90.50
s38417	1207	5142	2682	1073.03	99.66	5208	2686	858.85	99.66

6. EXPERIMENTAL RESULTS

The weighted transition graph algorithm, with all the optimizations described above, was implemented in C. Experimental data are presented for ISCAS85 and full-scan versions of ISCAS89 benchmarks, on a 1.7GHz Pentium 4 with 512MB of memory, running on the Linux Operating System.

In Table V, results for the full-scan version of ISCAS89 circuits with different chain lengths are presented. For each benchmark, the ideal (ID), given by the sum of the edge weights, and actual (AC) faults detected by the chains are shown. The difference between the ideal (ID) and actual (AC) increases with the chain length. For example, for circuit s5378, when the chain length is 2, the ideal and actual numbers of detected transition faults are the same. Likewise, when the chain length is increased to 3, they are still equal as explained by the proposed Theorem. When we increase the chain length beyond 3, the actual number of detected transition faults start to differ as some of the faults detected by this last chain segment may be detected by the first 2 pairs.

Table VI presents the results of the weighted transition graph algorithm with and without using essential vectors. In Table VI, column 2 gives the number of stuck-at vectors in the original STRATEGATE test set, followed by the results for our algorithm without using essential vectors. The final four columns show the results when essential vectors are used. For each approach, the number of

Table VII. ATE Repeat Vs. COM

	Storage				TF COV		CPU Time			IMPROVEMENTS	
	COM		WT_GR		*COM*	*WT_GR*	*COM*	WT_GR		*Str.*	Time
CKT	*s@*	*Tran*	3	4				3	4		
c1908	177	526	353	346	99.7	99.72	4.2	3.33	3.47	32.89	−34.22
c2670	167	396	185	184	78.6	79.26	4.3	9.25	9.99	53.28	6.57
c3540	247	732	410	419	82.9	87.62	6.8	19.45	20.11	43.99	−12.02
c5315	213	496	310	323	96.6	97.05	4.8	10.20	10.77	37.50	−25.00
c6288	47	180	130	118	99	98.54	3.6	3.72	3.62	27.77	−44.44
c7552	348	756	428	431	91	91.61	11	28.34	28.21	43.38	−13.23
s5378	391	980	455	464	86.6	87.51	5.3	37.66	37.76	53.57	7.14
s9234	630	1674	644	646	68.6	70.58	14.7	319.18	324.20	61.53	23.06
s13207	662	2004	681	679	80.5	82.29	27.4	292.89	295.30	66.02	32.06
s15850	641	1848	729	749	85	85.76	28	350.00	358.58	60.55	21.10
s35932	81	274	224	209	90	90.33	94.3	87.56	86.30	18.29	−63.50
s38417	1449	3854	1555	1547	89.9	91.19	116	1416.82	1406.83	59.65	19.30

transition vectors produced is shown first. Next, the number of compacted test vectors and its transition fault coverage are shown. Note that the compaction step achieves considerable reduction without losing fault coverage. The transition fault coverage achieved with or without essential vectors are the same, as indicated in column 6 and column 10 of the Table—only the test set sizes are different in the two approaches. In most cases, the use of essential vectors yields smaller test sets. However, because this is a greedy heuristic, optimality is not guaranteed. The execution time with essential vectors is also generally shorter due to the quick elimination of a large number of faults detected by the essential vectors. The extra computation needed in s38417 was due to the lack of trimming in the weighted transition-pattern graph. Because our target was to show the proof of the concept that stuck-at vectors can be chained through the proposed WTG, we did not explicitly target reduction in the execution time.

6.1 Experimental Results for ATE Repeat

In Table VII, we compare our results with results using COM, a commercial ATPG tool. We first tabulate the data for the storage required (STORAGE). Both the size of the stuck-at test set and the number of transition test vectors for COM are presented. These were generated using the dynamic compaction option of COM. Thus, for C1908 we need to store a total of 526 vectors. WT_GR used the stuck-at test set generated using COM without compaction. The next two columns show the transition test chain lengths obtained using the proposed algorithm, with chain length 3 and 4, respectively. Thus, for C1908, we need to store 353 vectors or 346 vectors depending on the chain length used in the algorithm. The storage improvement obtained using transition test chains and the ATE repeat option is shown in column 11, where a chain length of 3 was assumed. Thus, for C1908, the storage improvement is calculated as 100* (526–353)/(526). Note the substantial reduction in all cases. The average reduction in scan memory requirement is 46.5%.

Columns 6 and 7 compare the transition fault coverage obtained by weighted transition graph (WT_GR) and COM. Note that there is no loss in fault coverage

Table VIII. ATE Weighted Transition-Pattern Graph Algorithm vs. COM

	STORAGE		IMPROVEMENT
CIRCUIT	*COM*	*WT_GR*	DATA, APPTIME (%)
c1908	526	353	32.89
c2670	396	185	53.28
c3540	732	410	43.99
c5315	496	310	37.50
c6288	180	130	27.77
c7552	756	428	43.38
s5378	980	455	53.57
s9234	1674	644	61.53
s13207	2004	681	66.02
s15850	1848	729	60.55
s35932	274	224	18.29
s38417	3854	1555	59.65

using WT_GR. Columns 8, 9, and 10 compare the CPU time required by COM and the two versions of our algorithm. For most of the circuits, COM is much faster. The last column of the table shows changes in the test application time. Recall that for a given transition test chain, all but the first and last vectors are scanned in twice. Therefore, the test application time gain for C1908 is computed as 100*(526-2*353)/(526) = −34.22. This implies a 34.22% increase in test application time if transition test chains with ATE repeat are used. However, in a number of cases, the test application time actually decreases by a significant amount. The average increase in test application time is 6.9%.

Next, we will present the results for reducing the extra test application time with exchange scan.

6.2 Experimental Results for Exchange Scan

Benefits of using the exchange scan versus COM are reported in Table VIII. Transition test chains were computed using our heuristic with the chain length set to 3. The improvement in both test application time and test data volume are reported in column 4. Thus, for C1908, the test application time reduction is calculated as 100*(526–353)/(526) = 32.89%. We note that now there is a substantial reduction in both the scan memory requirement and the test application time, compared to COM. The average reduction in both test application time and data storage requirement is 46.5%.

Results from Tables VII and VIII are illustrated graphically in Figure 10. For each circuit, the data storage requirement and test application time are plotted for the conventional ATE, ATE repeat, and exchange scan.

6.3 Experimental Results for Constrained ATPG

In Table IX, we compare the results of the weighted transition graph algorithm with and without considering the constraints on functionally-untestable faults. The number of functionally-untestable faults identified by our implication engine is presented in column 2, followed by the percentage of

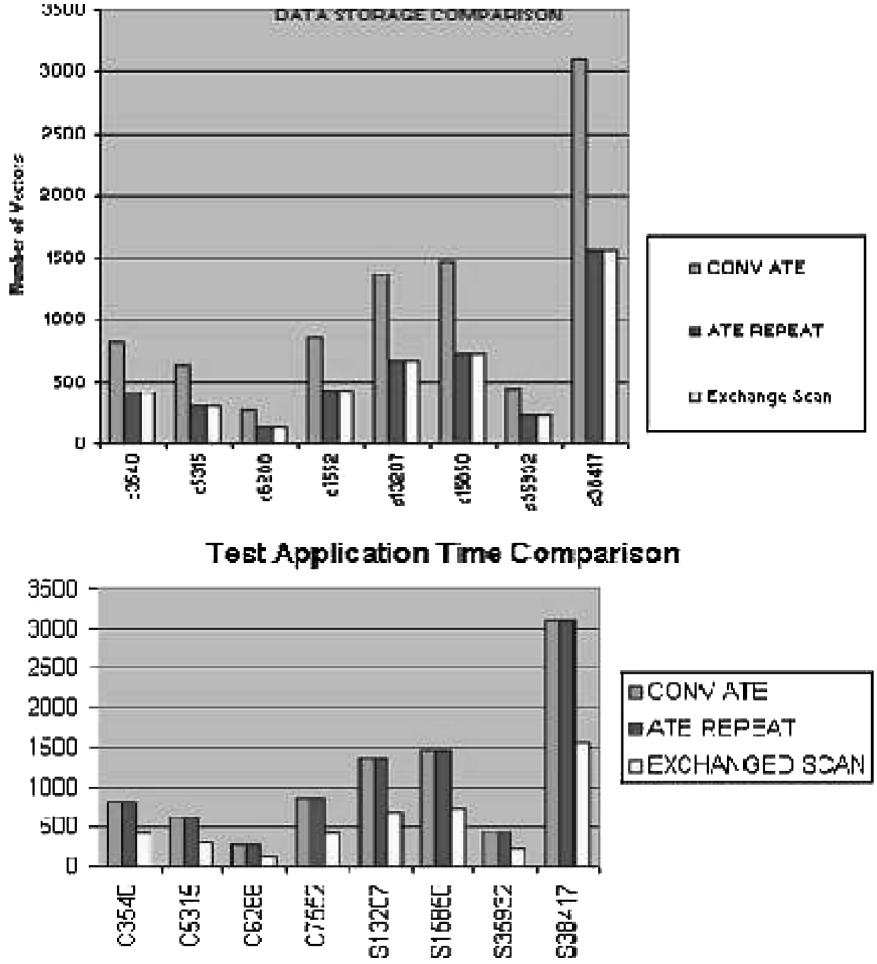

Fig. 10. Graphical experimental results.

functionally-untestable faults for each circuit. The next three columns show the transition fault coverage by the STRATEGATE stuck-at vectors: the total transition fault coverage, the functional testable fault coverage, and the overtesting fault ratio are given, respectively. Next, the results without considering the functionally-untestable faults are shown. And the final three columns tabulate the results on the transition fault coverage while considering the constraint on functionally-untestable faults.

For instance, for circuit s5378, our low-cost transition implication engine identified 3695 out of the total 15322 faults as functionally untestable. So the percentage of functionally-untestable faults is 3695/15322 = 24.12%. In other words, only 75.88% (1–24.12%) of the faults can be detected in the functional mode.

When looking at the STRATEGATE stuck-at test set for s5378, we found that it can detect 92.96% of the transition faults. While most of the

Table IX. Results With/Without Constraint

		Red	Orig. S@ vectors			Without Constraint			Constrained ATPG		
	Red	Ratio	*FC*	*FFC*	*OVT*	*FC*	*FFC*	*OVT*	*TFC*	*FFC*	OVT
Circuit	fault	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)	(%)
s344	66	6.53	88.02	82.57	5.45	98.51	92.07	6.44	93.37	91.79	1.58
s832	96	4.26	72.45	69.92	2.53	94.29	90.79	3.50	91.04	90.51	0.53
s1196	5	0.12	86.17	86.05	0.12	94.74	94.62	0.12	91.28	91.28	0
s1423	387	9.35	94.40	86.50	7.90	98.14	89.88	8.26	88.57	83.29	4.69
s5378	3695	24.12	92.96	75.08	17.89	94.16	75.87	18.29	84.30	74.86	3.75
s35932	11255	11.21	86.63	85.25	1.38	89.61	87.90	1.71	89.12	87.42	1.70
s38417	32086	27.03	97.36	72.14	25.22	97.63	72.38	25.25	87.91	69.34	18.57

functionally-testable faults can be detected (75.87% out of total 75.88%), the overtesting ratio for functionally-untestable faults is 17.29%. If we ignore the overtesting factor and only target those functionally-testable faults, our weighted transition graph algorithm can improve the total transition fault coverage to 94.16%, but at the cost of an overtesting ratio of 18.29%. Finally, if we impose the constraint of minimizing the overtesting of functionally-untestable faults in our graph algorithm, we can reduce the overtesting ratio to only 3.75% and still capture most of the functionally-testable faults. Only 75.88%-74.86 = 1.02% of the functionally-testable faults are missing. Note that for this constrained ATPG, we do not include the original sequence of s@ vectors in our final test set since the original order of vectors can potentially detect many functionally-untestable faults. Results for other benchmark circuits can be explained in a similar manner.

7. CONCLUSION

We have presented efficient techniques to reduce test data volume and test application time for transition faults. First, we propose a novel transition test chain formulation via a weighted transition-pattern graph. Only s@ ATPG is needed to construct the necessary test chains for transition faults. By combining the proposed transition test chain and ATE repeat capability, we reduce the test data volume by 46.5%, compared to the conventional approach. The second technique that replaces the ATE repeat option with Exchange Scan improves both the test data volume and the test application time by 46.5%. In addition, we address the problem of yield loss due to incidental *overtesting* of functionally-untestable transition faults, By formulating it into a constraint in our weighted pattern graph, we can efficiently reduce the overtesting ratio. The average reduction on the overtesting ratio is 4.68%, with a maximum reduction of 14.5%.

REFERENCES

CHANDRA, A. AND CHAKRABARTY, K. 2001. Frequency directed run length(FDR) codes with application to system on a chip data compression. In *Proceedings of the VLSI Testing Symposium*. 42–47.

DERVISOGLU, B. AND STONG, G. 1991. Design for testability: Using scanpath techniques for path-delay test and measurement. In *Proceedings of the IEEE International Test Conference*. 365–374.

DAS, D. AND TOUBA, N. A. 2000. Reducing Test Data Volume Using External/BIST Hybrid Test Patterns. In *Proceedings of the IEEE International Test Conference*. 115–122.

ELDRED, R.D. 1959. Test routing based on symbolic logical statement. *J. ACM*, *6*, 1 (Jan.), 33–36.

HSU, F. F., BUTLER, K. M., AND PATEL, J. H. 2001. A Case Study of the Illinois scan architecture. In *Proceedings of the IEEE International Test Conference*. 538–547.

HAMZAOGLU, I. AND PATEL, J. H. 1999. Reducing Test application time for full scan embedded cores. In *29th International Symposium on Fault-Tolerant Computing*. 260–267.

HERAGU, K., PATEL, J. H., AND AGRAWAL V. D. 1996. Segment delay faults: a new fault model. In *Proceedings of the VLSI Testing Symposium*. 32–39.

HSIAO, M. S. 2002. Maximizing impossibilities for untestable fault identification. In *IEEE Design Automation and Test in Europe Conference*. 949–953.

KELLER, B., BARNHART, C., BRUNKHORST, V., DISTLER, F., FERKO, A., FARNSWORTH, O., AND KOENEMAN, B. 2001. OPMISR: The foundation of compressed ATPG vectors. In *Proceedings of the IEEE International Test Conference*. 748–757.

KOENEMAN, B. 1991. LFSR-coded test patterns for scan designs. In *IEEE European Test Conference*. 237–242.

LEE, K.-J., CHEN, J.,-J., AND HUANG, C.-H. 1998. Using a single input to support multiple scan chains. In *IEEE/ACM International Conference on Computer-Aided Design*. 74–78.

LIU, X., HSIAO, M. S., CHAKRAVARTY, S., AND THADIKARAN, P. J. 2003. Efficient transition fault ATPG algorithms based on stuck-at test vectors. *J. Electr. Test. Theo. Applicat.*, *19*, 4 (Aug.), 437–445.

LAI, W. C., KRISTIC, A., AND CHENG, K. T. 2000a. On testing the path delay faults of a microprocessor using its instrcution set. In *Proceedings of the VLSI Testing Symposium*. 15–20.

LAI, W. C., KRISTIC, A., AND CHENG, K. T. 2000b. Test program synthesis for path delay faults in microprocessor cores. In *Proceedings of the IEEE International Test Conference*. 1080–1089.

REARICK, J. 2001. Too much delay fault coverage is a bad thing. In *Proceedings of the IEEE International Test Conference*. 624–633.

SMITH, G. L. 1985. Model for delay faults based upon paths. In *Proceedings of the IEEE International Test Conference*. 342–349.

SAVIR, J. AND PATIL, S. 1993. Scan-based transition test. *IEEE Trans. on Comput.-Aid. Des. Integr. Circuit Syst. 12*, 8 (Aug.).

SAVIR, J. AND PATIL, S. 1994. On broad-side delay test. In *Proceedings of the VLSI Testing Symposium*. 284–290.

TENDULKAR, N., RAINA, R., WOLTENBURG, R., LIN, X., SWANSON, B., AND ALDRICH, G. 2002. Novel techniques for achieving high at-speed transition fault coverage for Motorola's microprocessors based on PowerPC instruction set architecture. In *Proceedings of the VLSI Testing Symposium*. 3–8.

WAICUKAUSKI, J. A., LINDBLOOM, E., ROSEN, B. K., AND IYENGAR, V. S. 1987. Transition fault simulation. *IEEE Des. Test Comput.*, (April), 32–38.

Received March 2003; revised August 2003, January 2003; accepted January 2004