Submitted to Transactions on Computer-Aided Design

Test Point Insertion Based on Path Tracing

Nur A. Touba and Edward J. McCluskey

Dept. of Electrical & Computer Engineering
Engineering Science Building
The University of Texas at Austin
Austin, TX 78712-1084
Phone: (512) 232-1456
Fax: (512) 471-5532
Email: touba@ece.utexas.edu

Center for Reliable Computing
Gates Building 2A, Room 236
Stanford University
Stanford, CA 94305
Phone: (415) 723-1258
Fax: (415) 725-7398
Email: ejm@shasta.stanford.edu

ABSTRACT

The set of test patterns applied to a circuit during built-in self-test (BIST) may not provide sufficiently high fault coverage due to the presence of hard-to-detect faults. This paper presents an innovative method for inserting test points in a way that complete (100%) single stuck-at fault coverage is obtained for a specified set of test patterns. A very different approach is taken compared with previous test point insertion methods. Instead of using probabilistic techniques for test point placement, a path tracing procedure is used to place both control and observation points. Instead of adding extra scan elements to drive the control points, a few of the existing primary inputs to the circuit are ANDed together to form signals that drive the control points. By selecting which patterns the control point is activated for, the effectiveness of each control point is maximized. A comparison is made with the best previously published results for other test point insertion methods, and it is shown that the path tracing method requires fewer test points and less overhead to achieve the same or better fault coverage. It is also shown that the path tracing method can be used for timing-driven test point insertion that achieves zero performance degradation. Because the path tracing method is not based on randomness properties of the test patterns, it can be used for any set of test patterns. The set of test patterns can be pseudo-random (e.g., generated by a linear feedback register), quasi-random (e.g., generated by a multiple input signature analyzer), or not be random at all.

1. INTRODUCTION

In built-in self-test (BIST), on-chip circuitry is added for test pattern generation and output response analysis. Pseudo-random pattern testing is an attractive approach for BIST because very little hardware is required for test pattern generation. A linear feedback shift register (LFSR) or cellular automaton (CA) can be used to generate the pseudo-random patterns. These circuits can also be used as output response analyzers thereby serving a dual purpose during BIST. This advantage is exploited by BIST techniques such as BILBO registers [Koenemann 79] and circular BIST [Stroud 88], [Krasniewski 89], to reduce overhead.

Unfortunately, the pseudo-random patterns that are generated during BIST do not always provide high enough fault coverage for a reasonable test length. There are two ways to solve this problem: modify the pattern generator, or modify the circuit-under-test. A pseudo-random pattern generator can be modified by adding logic to weight the patterns [Schnurmann 75], [Wunderlich 87], [Pomeranz 92]; map the patterns [Touba 95a, 95b, 96], [Chatterjee 95a]; or reseed the generator [Venkataraman 93], [Hellebrand 95], [Zacharia 95]. The circuit-under-test can be modified by inserting test points [Eichelberger 83]; or by redesigning it [Touba 94], [Chiang 94], [Chatterjee 95b]. Each of these techniques has its advantages, and the one that is most suitable depends on the particular application. This paper presents a new method for inserting test points in combinational circuits that significantly reduces the number of test points required for a particular fault coverage compared with previous techniques.

1.1 Control and Observation Points

Test point insertion involves adding control and observation points to the circuit-under-test in a way that the system function remains the same, but the testability is improved. An *observation point* is an additional primary output that is inserted in the circuit to increase the observability of faults in the circuit. In the example in Fig. 1, an observation point is inserted at the output of gate *G1* such that faults are observable regardless of the logic value at node *y*. A *control point* is inserted in the circuit such that when it is activated, it fixes the logic value at a particular node to increase the controllability of some faults in the circuit. A control point can also affect the observability of some faults in the circuit because it can change the propagation paths in the circuit. In the example in Fig. 2, a control point is inserted to fix the logic value at the output of gate *G1* to a '1' when the control point is activated (this is called a *control-1 point*). This is accomplished by placing an OR gate at the output of gate *G1*. In the example in Fig. 3, a control point is inserted to fix the logic value at the output of gate *G1* to a '0' when the control point is activated (this is called a *control-0 point*). This is accomplished by placing an AND gate at the output of gate *G1*. During system operation, the control points are not activated and thus don't affect the system function.

However, control points do add an extra level of logic to some paths in the circuit. If a control point is placed on a critical timing path, it can increase the delay through the circuit.

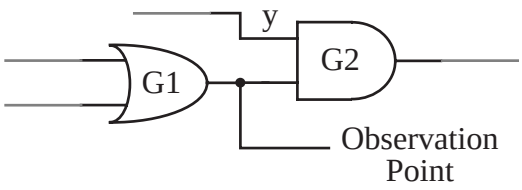

Figure 1. Example of Observation Point

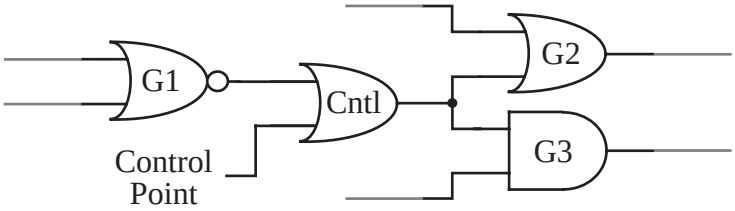

Figure 2. Example of Control-1 Point

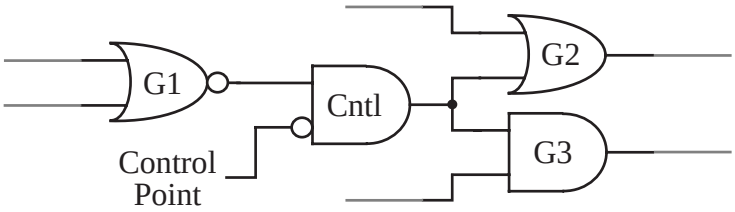

Figure 3. Example of Control-0 Point

1.2 Previous Work in Test Point Insertion

Since test points add both area and performance overhead, it is important to try to minimize the number of test points that are inserted to achieve the desired fault coverage. Optimal test point placement for circuits with reconvergent fan-out has been shown to be NP-complete [Krishnamurthy 87]. Briers and Totton [Briers 86] were the first to propose a systematic method for test point placement to increase pseudo-random pattern testability. They use simulation statistics to identify correlations between signals, and then insert test points to break the correlation. The number of test points inserted by this method is large. Iyengar and Brand [Iyengar 89] proposed an improved method that uses fault simulation to identify gates that block fault propagation, and then insert test points to allow propagation. Savaria *et al.*, in [Savaria 91] and [Youssef 93], use the COP testability measures [Brglez 84] to guide the placement of test

points. They identify sectors of hard-to-detect faults and insert test points at the origins of the sectors. Seiss *et al.*, in [Seiss 91], form a cost function based on the COP testability measures and then compute, in linear time, the gradient of the function with respect to each possible test point. The gradients are used to approximate the global testability impact for inserting a particular test point. Based on these approximations, a test point is inserted and the COP testability measures are recomputed. This process iterates until the testability is satisfactory. Cheng and Lin, in [Cheng 95], enhance the procedure in [Seiss 91] to consider the performance impact of inserting a particular test point. They showed that by avoiding control point insertion on critical timing paths, high fault coverage can be achieved with zero performance degradation. In [Touba 94], a method is proposed for inserting test points during logic synthesis. The logic is factored in a way that minimizes the number of test points that are required.

1.3 New Approach for Test Point Insertion

This paper presents a new approach for test point insertion. Fault simulation is used to identify faults that are not detected by a specified set of test patterns. For each undetected fault, a path tracing procedure is used to identify the set of test points that will enable the fault to be detected, i.e., the set of *test point solutions* for the fault. Given the set of test points solutions for each undetected fault, a minimal set of test points to achieve the desired fault coverage is selected using a set covering procedure. A new technique is used for driving the control points. Rather than adding extra scan elements to drive the control points, a few of the existing primary inputs to the circuit are ANDed together to form signals that drive the control points. This logic selects which patterns the control points are activated for. A method is described for synthesizing this logic to maximize the effectiveness of each control point.

Unlike other methods for test point placement that are based on signal probabilities or detection probabilities for pseudo-random patterns, the method presented in this paper is not based on randomness properties of the test patterns and therefore can be used for any set of test patterns. The set of test patterns can be pseudo-random, quasi-random (e.g., generated by a multiple input signature register), or not be random at all. Other test point placement methods that assume pseudo-random patterns may not be effective for BIST techniques that use multiple input signature registers (e.g., circular BIST [Stroud 88], [Krasniewski 89]) to apply patterns to the circuit-under-test.

A procedure is also described in this paper for performing timing-driven test point insertion using path tracing. By considering the performance impact of each test point, the procedure avoiding inserting control points on critical timing paths. By so doing, the timing-driven test point insertion procedure achieves zero performance degradation.

The paper is organized as follows: In Sec. 2, an overview of the test point insertion procedure is given. In Sec. 3, a path tracing procedure for identifying the set of test point solutions for an undetected fault is described. In Sec. 4, results are shown which indicate that multiple test points are required to detect some faults, and methods for computing test point solutions for those faults are described. In Sec. 5, the method for selecting a set of test points to insert is explained. In Sec. 6, the new technique for control point activation is presented, and a method for synthesizing the activation logic is described. In Sec. 7, results are shown for the new test point insertion method, and they are compared with previously published results. In Sec. 8, a timing-driven test point insertion procedure is described for achieving zero performance degradation. Section 9 is a conclusion.

2. OVERVIEW OF TEST POINT INSERTION PROCEDURE

The problem of interest is given a set of test patterns that will be applied to the circuit-under-test (e.g., the patterns generated by an LFSR during BIST), insert as few test points as necessary to enable all of the faults in the circuit to be detected. An overview of the test point insertion procedure is as follows:

1. Perform fault simulation to identify the undetected faults.

   Fault simulation is performed for the set of test patterns applied to the circuit-under-test to determine which faults are already detected and which require test points in order to be detected.

2. Compute the set of test points that enable each of the undetected faults to be detected.

   For each of the faults that require test points, a set of test point solutions is computed such that if any test point in the set is inserted into the circuit, the fault will be detected. This is described in Sections 3 and 4.

3. Select a minimal set of test points that provides complete fault coverage.

   Given the set of test point solutions for each fault, a set covering procedure is used to find a minimal set of test points that enables all of the faults to be detected. This is described in Sec. 5.

4. Synthesize logic to activate the control points.

   Pattern decoding logic is synthesized to activate control points for certain patterns. This is described in Sec. 6.

3. COMPUTING TEST POINT SOLUTIONS

The faults that go undetected by the set of test patterns applied to the circuit-under-test are the faults that require test points in order to be detected. This section describes a method for computing the set of test points solutions for a given undetected fault for a specified set of test patterns.

DEFINITION 1: Test point *p* is said to be a *solution* for fault *f* if inserting test point *p* into the circuit enables fault *f* to be detected for the specified set of test patterns.

In order for a fault to be detected, it must be both provoked and propagated to a primary output. A stuck-at 1 (stuck-at 0) fault is *provoked* if the logic value at the fault site is *'0' ('1')*. A fault is *propagated to a primary output* if a sensitized path exists from the fault site to a primary output. An observation point can only help with propagating a fault, while a control point can help with both provoking and propagating a fault.

DEFINITION 2: A *sensitized path* exists from node *x* to node *y* in a circuit if complementing the logic value at node *x* complements the logic value at node *y*.

Note that a sensitized path exists from an input of gate *g* to the output of the gate *g* if all of the other inputs to gate *g* are at the *non-controlling logic value* ('0' for OR and NOR gates, and '1' for AND and NAND gate).

The method presented here for computing test point solutions involves identifying sensitized paths to and from fault sites in the circuit. Fault-free simulation is performed for a pattern, and then path tracing from the fault sites is used to identify the sensitized paths. A fast approximate method for path tracing is given in [Abramovici 84]. Techniques for faster operation are suggested in [Ramakrishnan 90]. An exact method for path tracing is given in [Menon 91]. These three papers describe path tracing from the primary outputs (called critical path tracing), however the techniques can be easily generalized for path tracing from a fault site.

3.1 Computing Observation Point Solutions

Some patterns may provoke a fault but not propagate it to a primary output. If a fault is provoked by a pattern, then an observation point that is inserted at a node that the fault can propagate to will enable the fault to be detected and therefore is a solution for the fault. To find the set of observation point solutions for a fault that is provoked by a particular pattern, path tracing can be used to identify the nodes that the fault can propagate to. An example is shown in Fig. 4. Fault-free simulation is performed for a pattern that provokes the stuck-at 1 fault, and forward path tracing from the fault site is used to identify the propagation path for the fault. The fault propagates through gates *G3* and *G5*, but is blocked at gates *G6* and *G8* and therefore doesn't propagate to a primary output. Inserting an observation point at node *a* or node *b* would enable the fault to be detected, so those two nodes form the set of observation point solutions for the fault for that pattern. The union of the set of observation point solutions for each pattern that provokes a particular fault gives the full set of observation point solutions for the fault.

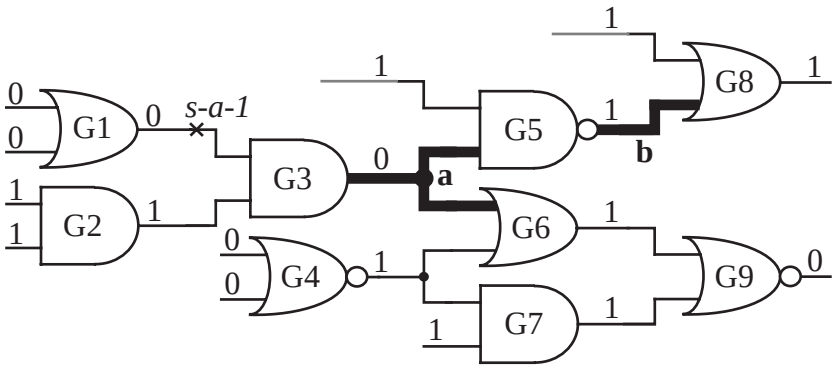

Figure 4. Example: Observation Point at Node *a* or *b* is a Solution.

3.2 Computing Control Point Solutions

Some patterns may propagate a fault to a primary output but not provoke the fault. In that case, a control point is a solution for the fault if it complements the logic value at the fault site thereby provoking the fault. For a fault that is propagated to a primary output by a particular pattern, path tracing can be used to find the nodes that have a sensitized path to the fault site for that pattern. Control points that complement the logic value at a node that has a sensitized path to the fault site are solutions for the fault provided that they don't block fault propagation to a primary output. An example is shown in Fig. 5. Fault-free simulation is performed for a pattern that propagates the fault to a primary output, and backward path tracing from the fault site is used to identify sensitized paths. Both inputs of gate *G6* have a sensitized path to the output of gate *G6*. Neither of the inputs of gate *G4* have a sensitized path to the output of gate *G4*. One of the inputs of gate *G3* has a sensitized path to the output of gate *G3*. Inserting a control-1 point at node *a*, *c*, *d*, or *e* would complement the value at the fault site thereby provoking the fault. However, forward path tracing from node *e* identifies that it has a sensitized path to gate *G9*, so inserting a control-1 point at node *e* would block the fault from propagating to a primary output. Therefore, only control-1 points at nodes *a*, *c*, and *d* are solutions.

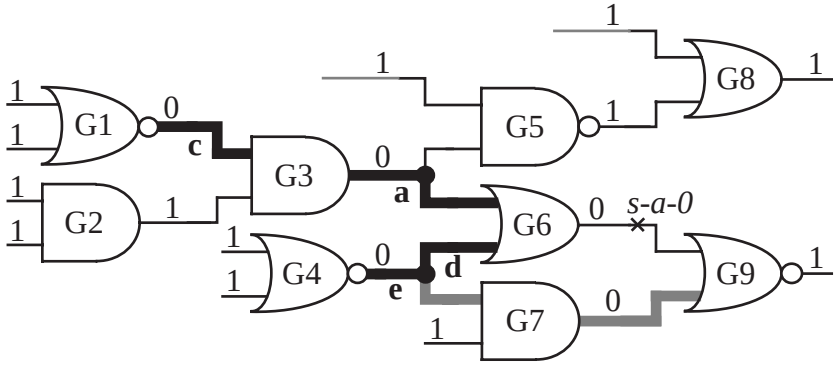

Figure 5. Example: Control-1 Point at Node *a*, *c*, or *d* is a Solution, but Node *e* is Not a Solution Because it Blocks Propagation to a Primary Output.

Some patterns may provoke a fault but a single gate may block propagation to a primary output. In that case, a control point can enable propagation to a primary output if it complements the logic value at the controlling input of the blocking gate. For a provoked fault for which fault propagation is blocked by a single gate, path tracing can be used to find the nodes that have a sensitized path to the controlling input of the blocking gate. Control points that complement the logic value at a node that has a sensitized path to the controlling input of a blocking gate are solutions for the fault provided that the fault remains provoked. An example is shown in Fig. 6. Fault-free simulation is performed for a pattern that provokes the fault, and backward path tracing from the controlling input of the blocking gate is used identify sensitized paths. Both of the inputs of gate *G7* are sensitized to the output of *G7*. Both of the inputs to *G4* are sensitized to the output of *G4*. The output of *G4* fans out to gate *G6*, and forward path tracing identifies that it has a sensitized path to the fault site. Inserting control points at the nodes before the fanout would cause the fault to not be provoked, and therefore they are not solutions. Control-0 points at nodes *f*, *g*, and *h* form the solution set for the fault for that pattern. The union of the set of control point solutions for a particular fault for each pattern gives the full set of control point solutions for the fault.

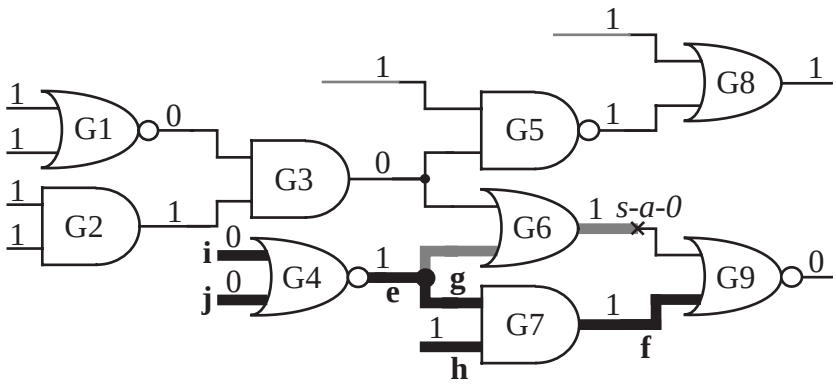

Figure 6. Example: Control-0 Point at Node *f*, *g*, or *h* are Solutions, but Node *e*, *i*, and *j* are Not Solutions Because They Don't Provoke Fault.

4. FAULTS REQUIRING MULTIPLE TEST POINTS

Some faults may not have single test point solutions. If none of the patterns provoke or propagate the fault, then multiple test points are required. The existence of a single test point solution for a fault can easily be checked when fault simulation is performed to identify the undetected faults. Table 1 shows data for some benchmark circuits using 32,000 pseudo-random patterns. The following information is given for each circuit:

Num Undetected - The number of uncollapsed faults that are not detected and hence require test points (this includes redundant faults).

Num Provoked - The number of undetected faults that are provoked by at least one pattern.

Num Propagated to PO - The number of undetected faults that are propagated to a primary output by at least one pattern.

Num Single Test Point - The number of undetected faults that have single test point solutions.

Num Two Test Points - The number of undetected faults that require two test points to enable them to be detected.

As can be seen from Table 1, most undetected faults have single test points solutions. However, some circuits do have a significant number of faults that require two test point solutions. None of the circuits have faults that require more than two test points.

Table 1. Existence of Single Test Point Solutions for Faults in Benchmark Circuits

Circuit Name	Num Undetected	Num Provoked	Num Propagated to PO	Num Single Test Point	Num Two Test Points
s420	2	2	2	2	0
s641	22	21	5	22	0
s713	50	45	19	46	4
s838	90	70	64	64	26
s1196	5	5	3	5	0
s1238	82	82	76	79	3
C2670.s	97	92	57	96	1
C7552.s	296	289	65	282	14

In Sec. 3, a method was described for computing a set of single test point solutions for a fault such that if any test point in the set is inserted in the circuit, the fault will be detected. This method can be extended to handle faults that require multiple test points. If a fault requires *n* test points, then *n* sets of single test points can be computed such that if one test point from each set is inserted in the circuit, the fault will be detected. For each pattern, path tracing can be used to find a set of test points that provoke the fault and the set of test points that propagate the fault to a primary output as was described in Sec. 3. In the example in Fig. 7, the sensitized paths that provoke the

fault and the paths where the fault can propagate are identified using path tracing. The control points that provoke the fault form one set (control-0 at node *m* and control-0 at node *n*), and the observation points that enable the fault to propagate to a primary output form another set (observation point at node *a* and node *b*). If a test point from each set is inserted in the circuit, then the fault will be detected. For faults with single test point solutions, the full set of solutions was formed by taking the union of the sets of single test points solutions for each pattern, however this cannot be done for faults with multiple test point solutions. The reason is that if the union of the sets is taken, then a test point from one set will provoke the fault for some pattern, but the test point from the other set may propagate the fault to a primary output for a different pattern, thus there is no guarantee that the fault will be detected. So instead of computing the full set of multiple test point solutions, the largest set of multiple test point solutions for a single pattern is used.

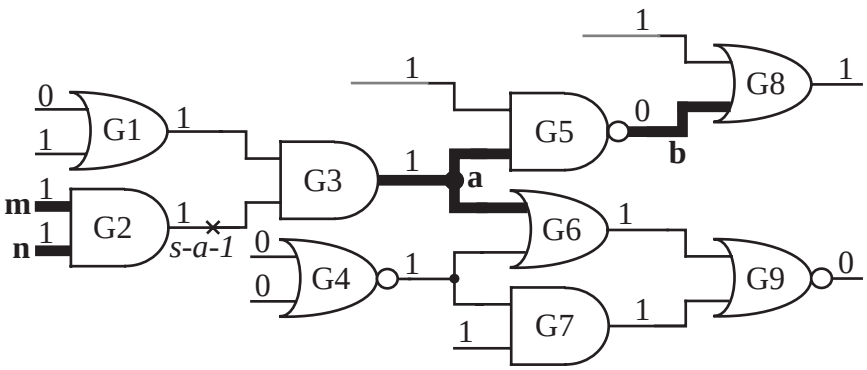

Figure 7. Example: A Control-0 Point at Node *m* or *n* Provokes the Fault, and an Observation Point at Node *a* or *b* Propagates the Fault to a Primary Output.

5. SELECTING A SET OF TEST POINTS TO INSERT

Once the set of test point solutions for each undetected fault has been computed, a set covering procedure can be used to select a minimal set of test points that will enable all of the faults to be detected. A matrix is constructed in which each column corresponds to a test point solution. For each undetected fault, a row is added to the matrix in which an 'X' is placed in each column that corresponds to a test point solution for the fault. If the fault requires a multiple test point solution, then multiple rows are added. An example is shown in Fig. 8. The first row corresponds to *fault 1* for which the set of single test point solutions is an observation point at node *w*, a control-1 point at node *u*, and a control-0 point at node *v*. *Fault 2* requires a multiple test point solution, so both the second and the third row correspond to it. The set of control points that provokes *fault 2* is a control-1 point at node *u*, a control-0 point at node *v* and a control-1 point at node *y*. The set of observation points that propagate *fault 2* to a primary output is an observation point at node *v* and an observation point at node *x*.

	O-v	O-w	O-x	C1-u	C0-v	C0-w	C1-y	C1-z
Fault 1		X		X	X			
Fault 2				X	X		X	
	X		X					
Fault 3			X		X			X
Fault 4	X		X		X			
Fault 5				X		X		

Figure 8. Example: Matrix of Test Point Solutions for Each Fault

A set covering procedure is used to select a minimal set of columns that has at least one 'X' in each row (set covering is NP-complete, but good heuristics exist [Christofedes 75]). One 'X' in each row ensures that all of the faults will be detected. In the example in Fig. 8, one such solution is the third column (observation point at node *x*) and the fourth column (control-1 point at node *u*). The test points corresponding to the selected columns are inserted into the circuit. Note that for the example in Fig. 8, if the test points were inserted one at a time based on maximizing the fault coverage that results after each test point was inserted (as is the case in most other test point insertion methods), then the first test point to be inserted would be a control-0 point at node *v* because that would detect *fault 1*, *fault 3*, and *fault 4*. However, in order to detect *fault 2* and *fault 5*, at least two more test points would have to be inserted. Thus, for this example, the greedy method results in 3 test points compared with only 2 if the set covering procedure is used.

6. CONTROL POINT ACTIVATION

Once the test points have been inserted, the remaining task is to design the logic that drives the control points. A control point must be activated for certain patterns in order to detect the faults for which it was inserted. However, a control point cannot be activated for all patterns because that would reduce the fault coverage. Previous test point insertion techniques add extra scan elements to drive the control points. This is illustrated in Fig. 9 where two extra scan elements are added to drive the two control points. The pseudo-random pattern generator is used to apply values to the extra scan elements. Thus, a control point is randomly activated for roughly half of the patterns. This approach limits the potential of each control point. There may be some patterns for which a control point is not activated, but if the control point had been activated, some faults would have been detected. Conversely, there may be some patterns for which the control point is activated, but if it hadn't been activated, some faults would have been detected.

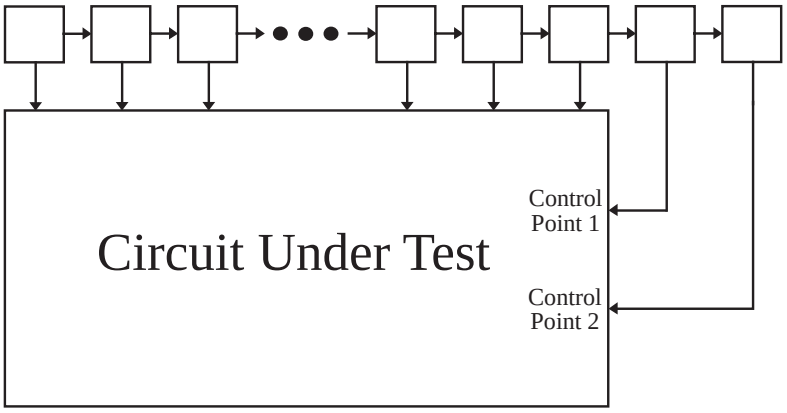

Figure 9. Control Points Driven by Extra Scan Elements

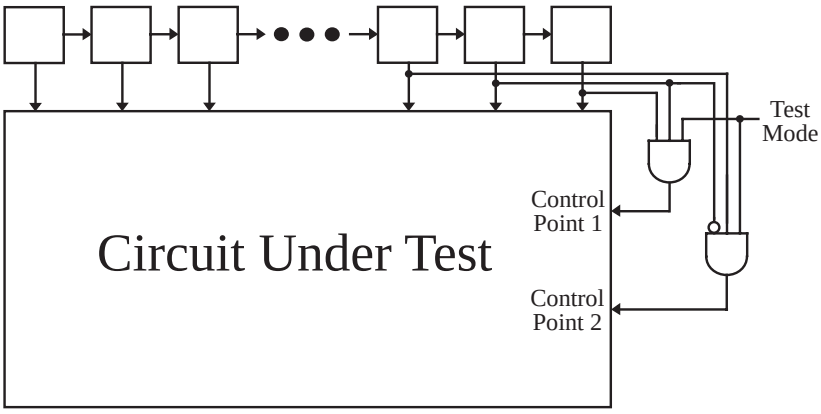

Figure 10. Control Points Driven by Pattern Decoding Logic

A new approach for activating control points is presented here. Pattern decoding logic is used to drive the control points. An example is shown in Fig. 10 where AND gates are used to drive each of the control points. Control point 1 is activated for any pattern that has a '1' in the last two bit positions. Control point 2 is activated for any pattern that has a '1' in the third to last bit position and a '0' in the second to last bit position. The decoding logic function for activating each control point is formed by placing all of the patterns for which the control point should be activated in the on-set, all of the patterns for which the control point should not be activated for in the off-set, and the remaining patterns in the don't care set (how to determine which patterns to place in the on-set and off-set will be explained later). This function is then passed to a logic synthesis tool to generate the pattern decoding logic. Using this pattern decoding logic to activate the control point maximizes the control point's effectiveness while ensuring that it won't cause faults that were previously detected to become undetected. Moreover, because of the large number of don't cares, the pattern decoding logic usually amounts to only one or two gates (as indicated by the experimental results in Sec. 7) and therefore results in less area overhead than adding an extra scan element. A test mode line is used to disable the control points during system operation. The delay introduced by a control point during system operation is the same regardless of which method is used to drive the control point. In either case, the signal driving the control point is a static '0' during system operation, so the delay through the control point is equal to the delay through the control gate (see Figs. 2 and 3).

Now the process of determining which patterns should be placed in the on-set and off-set of the pattern decoding logic function for each control point will be explained. First consider the off-set. A fault that was detected by pattern v before inserting a control point may no longer be detected if the control point is activated for pattern v. So one way to ensure that inserting control points doesn't cause faults to no longer be detected is to place one pattern that detects each fault into the off-set so that the control points won't be activated for those patterns. During the initial fault simulation, when a new fault is detected, the pattern that detected it is recorded. These patterns are placed in the off-set of the decoding logic function for each control point. This is a conservative approach since some of the patterns may detect the same faults regardless of whether the control point is activated or not. An optional step to reduce the off-set for a control point is use fault simulation to check which patterns are really affected by the control point. Fault simulation can be done with the control point activated for each pattern in the off-set, and the patterns which drop the same set of faults as before (i.e., with no control point) can be removed from the off-set since it doesn't matter for those patterns whether the control point is activated or not.

The on-set of the decoding logic function contains the patterns for which the control point is activated. The purpose of a control point is to enable detection of the faults for which it is a solution. It must be activated for a pattern that detects each of the faults for which it is a solution.

When the set of control point solutions are computed for each undetected fault, the patterns for which each control point enables the fault to be detected are recorded. For a control point that is selected for insertion, one of the recorded patterns is added to the on-set for each fault for which the control point is a solution. The patterns that are added to the on-sets for each of the inserted control points are chosen so that the on-sets are disjoint. This ensures that there are no conflicts with more than one control point being activated for the same pattern. An example of selecting the on-sets for 3 control points is shown in Fig. 11. For each fault, the set of patterns for which each control point will enable the fault to be detected are listed. One pattern for each fault is selected and added to the appropriate control point's on-set. The patterns are selected so that the on-set for each control point is disjoint. If *control point 1* is activated for the pattern *101110*, it enables *fault 1* and *fault 3* to be detected. If *control point 2* is activated for the same pattern, *101110*, it enables *fault 2* to be detected. However, if both *control point 1* and *control point 2* are activated for the same pattern, then it is possible that there would be a conflict such that one of the faults would not be detected. So in order to avoid that, *control point 2* can be activated for the pattern *011100* instead.

	Control Point 1	Control Point 2	Control Point 3
Fault 1	**101110**, 110101, 001110, ...		
Fault 2		101110, **011100**, 110110, ...	000110, 000111, 001111, ...
Fault 3	**101110**, 010110, 010011, ...	011010, 011011, 011111, ...	
Fault 4	**001010**, 110111, 111001, ...		
Fault 5			**110011**, 011010, 011001, ...
On-Set	{101110, 101110}	{0111000}	{110011}

Figure 11. Example: Selecting On-Sets for 3 Control Points

7. EXPERIMENTAL RESULTS

The method described in this paper was used to insert test points in some of the ISCAS 85 [Brglez 85] and ISCAS 89 [Brglez 89] benchmark circuits that contain random-pattern-resistant faults. LFSR's were used to apply 32,000 pseudo-random test patterns to each circuit. The test length of 32,000 was chosen in order to compare with results from previously published papers. The flip-flops in the ISCAS 89 circuits were configured as part of the LFSR during testing so that the circuits are tested like combinational circuits. The number of stages in the LFSR for each circuit was equal to the number of primary inputs plus the number of flip-flops.

The procedure described in this paper was used to insert test points into each circuit so that all single stuck-at faults were detected for the set of 32,000 pseudo-random test patterns. The results are shown in Table 2. The total number of faults is shown followed by the number of redundant faults. The redundant faults are made testable by the test point insertion procedure. Simplified versions of the circuits *C2670* and *C7552* were made by removing the redundant logic; these circuits are labeled *C2670.s* and *C7552.s*. The fault coverage before test point insertion and after test point insertion is shown. The fault coverage is for all faults including redundant faults. The number of control points (*Num Con*) and the number of observation points (*Num Obs*) that were inserted are shown. The amount of pattern decoding logic that was needed to drive the control points is shown. It is measured in gate equivalents (GE's) that reflect a static CMOS technology: *(0.5)(n)* GE's for an *n*-input NAND or NOR, and *(2.5)(n-1)* GE's for an *n*-input XOR. As can be seen, very few gates are required for the pattern decoding logic. The average number of gate equivalents for the pattern decoding logic for each control point is less than 2 GE's.

The total hardware overhead added to each circuit is shown for two cases. The first case is where no condensation network is used to combine the observation points; each observation point is fed into an extra scan element. The number of extra gate equivalents and extra scan elements added to the circuit are shown for this case. The extra gates are due to the pattern decoding logic plus the control gate for each control point. There is one extra scan element for each observation point. The second case that is shown is where the observation points are combined through a condensation network. The condensation network was constructed using the techniques in [Fox 77] to ensure that no aliasing occurs and that the condensation network itself is fully tested. In [Fox 77], a set of observation points for which no more than one observation point can have a faulty value at a time is said to be *fault independent.* A set of fault independent observation points can be combined through an XOR tree without aliasing. Moreover, a set of observation points is said to be *value independent* if the faults to be observed at each observation point can be observed for all combinations of values at the other observation points. It was shown in [Fox 77], that a set of observation points that is both fault independent and value independent can be combined through any irredundant

combinational circuit without aliasing. By analyzing the observation points, a suitable condensation network can be constructed. The condensation network adds more gates, but reduces the number of extra scan elements. *Note that the fault coverage after test point insertion is 100% of all faults, including those in the pattern decoding logic and the condensation network.*

For each circuit, Table 3 shows the dimensions of the test point solution matrix described in Sec. 5, the total number of test points that are inserted, and the total CPU time for the test point insertion procedure. The procedure is implemented in 'C' and was run on an UltraSPARC 2. Note that the CPU times could be significantly improved by using an industrial quality fault simulator.

Table 2. Results for Test Point Insertion in Benchmark Circuits

Circuit Name	Num Faults	Num Red	Coverage Before TPI	Coverage After TPI	Num Con	Num Obs	Decode Gates	No Condensation		With Condensation	
								Gates	Scan Elem	Gates	Scan Elem
s420	215	0	90.2%	100%	0	2	0	0	2	3	1
s641	346	0	97.6%	100%	1	1	2	3	1	3	1
s713	405	38	88.3%	100%	1	1	2	3	1	3	1
s838	667	0	93.8%	100%	2	0	6	7	0	7	0
s1196	968	0	99.6%	100%	1	0	1	2	0	2	0
s1238	1063	67	93.5%	100%	6	5	18	24	5	31	2
C2670	1881	92	91.3%	99% 100%	4 7	3 18	6 8	10 15	3 18	15 55	1 2
C2670.s	1717	0	96.7%	100%	2	2	4	6	2	9	1
C7552	5101	133	94.4%	99% 100%	4 19	6 61	9 24	13 43	6 61	23 183	2 5
C7552.s	4830	0	97.3%	100%	6	8	12	18	8	33	2

Table 3. Execution Time for Test Point Insertion Procedure

Circuit Name	Test Point Solution Matrix		Num Test Points Inserted	CPU Time (seconds)
	Num Rows	Num Cols		
s420	2	22	2	30
s641	22	60	2	50
s713	54	155	2	80
s838	116	150	2	350
s1196	5	38	1	150
s1238	85	302	11	690
C2670	214	877	25	1600
C2670.s	98	609	4	610
C7552	494	775	80	2300
C7552.s	310	418	14	1400

In Table 4, the results for the path tracing method described in this paper are compared with the published results for the test point insertion methods in [Briers 86], [Seiss 91] and [Youssef 93]. The number of control points (*Con)* and observation points (*Obs*) that were inserted by each

method is shown along with the resulting fault coverage (*Cov*). As can be seen, the path tracing method uses fewer test points to achieve the same or better fault coverage than the other methods. For most circuits, there is a significant reduction in the number of test points.

Table 4. Comparison of Number of Test Points and Fault Coverage

Circuit Name	[Briers 86]			[Youssef 93]			[Seiss 91]			Path Tracing		
	Con	Obs	Cov	Con	Obs	Cov	Con	Obs	Cov	Con	Obs	Cov
s420	NA	NA	NA	2	0	100%	NA	NA	NA	2	0	100%
s641	NA	NA	NA	2	1	100%	NA	NA	NA	1	1	100%
s713	NA	NA	NA	2	1	97.8%	NA	NA	NA	1	1	100%
s838	NA	NA	NA	3	12	98.5%	NA	NA	NA	2	0	100%
s1238	NA	NA	NA	9	13	98.8%	NA	NA	NA	6	5	100%
C2670	7	46	99.9%	1	10	96.1%	NA	NA	NA	4 7	3 18	99% 100%
C2670.s	NA	NA	NA	NA	NA	NA	3	7	100%	2	2	100%
C7552	38	55	99.9%	11	5	98.9%	NA	NA	NA	4 19	6 61	99% 100%
C7552.s	NA	NA	NA	NA	NA	NA	18	2	100%	6	8	100%

8. TIMING-DRIVEN TEST POINT INSERTION

The path tracing method described in this paper can also be used to minimize the performance impact of inserting test points. The key is to avoid placing control points on critical timing paths. If a control point is placed on a critical timing path, then it will increase the delay through the circuit. So instead of selecting the set of control points to insert based only on minimizing the total number of test points, timing-driven test point insertion can be performed by considering the performance impact of each control point as well (as was proposed in [Cheng 95]).

The procedure for timing-driven test point insertion is as follows. First the circuit is mapped to a cell library and a static timing analyzer is used to compute the timing slack for each node. Then the path tracing procedure is used to form the matrix of test point solutions (as described in Sec. 5). Since the timing analysis is done at the cell level, it is desirable to insert test points only at the cell boundaries. If test points are inserted inside the cells, then the logic would need to be re-mapped which would invalidate the timing analysis. Thus, the columns in the matrix of test points solutions that don't correspond to test points at cell boundaries are removed from the matrix. Then for each of the remaining columns that corresponds to a control point, a check is made to see if there is sufficient timing slack for the control point to be inserted without impacting the performance of the circuit. If there is not enough timing slack for the control point to be inserted, then the column is removed from the matrix. So the resulting matrix contains only test point solutions that do not impact performance and are at cell boundaries. The remaining task is to select a set of test points to insert.

There are two ways to proceed. The first is to use a greedy approach in which the test points are selected one at a time. If a control point is selected, then it is inserted in the circuit and the timing slacks are recomputed. A check is made again to see if there is sufficient timing slack for each of the remaining control point solutions. Columns corresponding to control points for which there is not enough timing slack are removed before selecting the next test point to insert. The second approach is to use a set covering procedure to select all of the test points at once. If more than one control point is inserted on the same path, then a check must be made to ensure that the timing slack for the path is sufficient for the combined effect of the control points. If the timing slack is not sufficient, then one of the control points is removed from the matrix and the set covering is redone. Either of these approaches will ensure that the set of test points that are inserted in the circuit will not degrade performance.

Results for timing-driven test point insertion based on path tracing are shown in Table 5. The circuits were mapped using the MOSIS 2 micron standard cell library. The original fault coverage and delay is shown followed by the number of test points, the fault coverage, and the delay using the normal test point insertion procedure and using the timing-driven test point insertion procedure

described in this section. For *s1238* and *C7552.s*, the normal test point insertion procedure placed control points along critical timing paths resulting in some performance degradation. The timing-driven test point insertion procedure achieved 100% fault coverage for all of the circuits without placing control points on critical timing paths. The timing-driven test point insertion procedure requires more test points than the normal procedure, but it is very useful for timing critical applications.

Table 5. Results for Timing-Driven Test Point Insertion

Circuit	Original		Normal Test Point Insertion			Timing-Driven Test Point Insertion		
Name	Fault Cov	Delay (ns)	Test Points	Fault Cov	Delay (ns)	Test Points	Fault Cov	Delay (ns)
s1238	93.5%	16.7	11	100%	17.1	15	100%	16.7
C2670.s	96.7%	22.0	4	100%	22.0	4	100%	22.0
C7552.s	97.3%	29.0	14	100%	30.3	18	100%	29.0

9. CONCLUSIONS

This paper presented two innovations for test point insertion: (1) a path tracing method for test point placement of *both* control and observation points, and (2) the use of pattern decoding logic to activate control points. These two innovations greatly improve the effectiveness of control points thereby reducing the number of test points that are required to provide a desired fault coverage. Experimental results indicate a significant reduction in the number of test points compared with previous methods. Fewer test points means less area and performance overhead for BIST.

Unlike other test point insertion methods, the method described in this paper is not based on signal probabilities or fault detection probabilities. This provides two important advantages. The first is that this method can be used to increase fault coverage for any set of test patterns, not just pseudo-random test patterns. The second is that this method can be used to target fault models for which computing fault detection probabilities is difficult. One such fault model is bridging faults. Using path tracing to insert test points to improve bridging fault coverage is a promising approach that is currently being investigated.

ACKNOWLEDGMENTS

This work was supported in part by the Ballistic Missile Defense Organization, Innovative Science and Technology (BMDO/IST) Directorate and administered through the Department of the Navy, Office of Naval Research under Grant No. N00014-92-J-1782, by the National Science Foundation under Grant No. MIP-9107760, and by the Advanced Research Projects Agency under prime contract No. DABT63-94-C-0045.

REFERENCES

[Abramovici 84] Abramovici, M., P.R. Menon, and D.T. Miller, "Critical Path Tracing: An Alternative to Fault Simulation," *IEEE Design & Test of Computers*, Vol. 1, pp. 89-93, Feb. 1984.

[Briers 86] Briers, A.J., and K.A.E. Totton, "Random Pattern Testability by Fast Fault Simulation," *Proc. of International Test Conference*, pp. 274-281, 1986.

[Brglez 84] Brglez, F., "On Testability of Combinational Networks," *Proc. of International Symposium on Circuits and Systems*, pp. 221-225, 1984.

[Brglez 85] Brglez, F., and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan," *Proc. of International Symposium on Circuits and Systems*, pp. 663-698, 1985.

[Brglez 89] Brglez, F., D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. of International Symposium on Circuits and Systems*, pp. 1929-1934, 1989.

[Chatterjee 95a] Chatterjee, M., and D.K. Pradhan, "A Novel Pattern Generator for Near-Perfect Fault-Coverage," *Proc. of VLSI Test Symposium*, pp. 417-425, 1995.

[Chatterjee 95b] Chatterjee, M., D.K. Pradhan, and W. Kunz, "LOT: Logic Optimization with Testability - New Transformations using Recursive Learning," *Proc. of International Conference on Computer-Aided Design (ICCAD)*, 1995.

[Chiang 94] Chiang, C.-H., and S.K. Gupta, "Random Pattern Testable Logic Synthesis," *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 125-128, 1994.

[Cheng 95] Cheng, K.-T., and C.J. Lin, "Timing-Driven Test Point Insertion for Full-Scan and Partial-Scan BIST," *Proc. of International Test Conference*, pp. 506-514, 1995.

[Christofedes 75] Christofedes, N., and K. Korman, "A Computational Survey of Methods for the Set Covering Problem," *Management Science*, Vol. 21, No. 5, pp. 591-599, Jan. 1975.

[Eichelberger 83] Eichelberger, E.B., and E. Lindbloom, "Random-Pattern Coverage Enhancement and Diagnosis for LSSD Logic Self-Test," *IBM Journal of Research and Development*, Vol. 27, No. 3, pp. 265-272, May 1983.

[Fox 77] Fox, J.R., "Test-point Condensation in the Diagnosis of Digital Circuits," *Proceedings of the IEE*, Vol. 124, No. 2, Feb. 1977, pp. 89-94.

[Hartmann 93] Hartmann, J., and G. Kemnitz, "How to Do Weighted Random Testing for BIST," *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 568-571, 1993.

[Hellebrand 95] Hellebrand, S., J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Transactions on Computers* , Vol. 44, No. 2, pp. 223-233, Feb. 1995.

[Iyengar 89] Iyengar, V.S., and D. Brand, "Synthesis of Pseudo-Random Pattern Testable Designs," *Proc. International Test Conference*, pp. 501-508, 1989.

[Koenemann 79] Koenemann, B., J. Mucha, and G. Zwiehoff, "Built-in Logic Block Observation Technique," *Proc. of International Test Conference*, pp. 140-150, 1979.

[Krasniewski 89] Krasniewski, A., and S. Pilarski, "Circular Self-Test Path: A Low-Cost BIST Technique for VLSI Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 8, No. 1, pp. 46-55, Jan. 1989.

[Krishnamurthy 87] Krishnamurthy, B., "A Dynamic Programming Approach to the Test Point Insertion Problem," *Proc. of the 24th Design Automation Conference*, pp. 695-704, 1987.

[Menon 91] Menon, P., Y. Levendel, and M. Abramovici, "SCRIPT: A Critical Path Tracing Algorithm for Synchronous Sequential Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 10, No. 6, pp. 738-747, Jun. 1991.

[Pomeranz 92] Pomeranz, I., and S.M. Reddy, "3-Weight Pseudo-Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuits," *IEEE Transactions on Computer-Aided Design*, Vol. 12, No. 7, pp. 1050-1058, Jul. 1993.

[Ramakrishnan 90] Ramakrishnan, T., and L. Kinney, "Extension of the Critical Path Tracing Algorithm," *Proc. of the 27th Design Automation Conference*, pp. 720-723, 1990.

[Savaria 91] Savaria, Y., M. Youssef, B. Kaminska, and M. Koudil, "Automatic Test Point Insertion for Pseudo-Random Testing," *Proc. of International Symposium on Circuits and Systems*, pp. 1960-1963, 1991.

[Schnurmann 75] Schnurmann, H.D., E. Lindbloom, and R.G. Carpenter, "The Weighted Random Test-Pattern Generator," *IEEE Transactions on Computers* , Vol. C-24, No. 7, pp. 695-700, Jul. 1975.

[Seiss 91] Seiss, B.H., P.M. Trouborst, and M.H. Schulz, "Test Point Insertion for Scan-Based BIST," *Proc. of European Test Conference*, pp. 253-262, 1991.

[Stroud 88] Stroud, C.E., "Automated BIST for Sequential Logic Synthesis," *IEEE Design & Test of Computers*, Vol. 5, No. 6, pp. 22-32, Dec. 1988.

[Touba 94] Touba, N.A., and E.J. McCluskey, "Automated Logic Synthesis of Random Pattern Testable Circuits," *Proc. of International Test Conference*, pp. 174-183, 1994.

[Touba 95a] Touba, N.A., and E.J. McCluskey, "Transformed Pseudo-Random Patterns for BIST," *Proc. of VLSI Test Symposium*, pp. 410-416, 1995.

[Touba 95b] Touba, N.A., and E.J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," *Proc. of International Test Conference*, pp. 674-682, 1995.

[Touba 96] Touba, N.A., and E.J. McCluskey, "Altering a Pseudo-Random Sequence of Bits for Scan-Based BIST," *Proc. of International Test Conference*, 1996.

[Venkataraman 93] Venkataraman, S., J. Rajski, S. Hellebrand, and S. Tarnick, "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers," *Proc. of International Conference on Computer-Aided Design (ICCAD),* pp. 572-577, 1993.

[Youssef 93] Youssef, M., Y. Savaria, and B. Kaminska, "Methodology for Efficiently Inserting and Condensing Test Points," *IEE Proceedings-E*, Vol. 140, No. 3, pp. 154-160, May 1993.

[Wunderlich 87] Wunderlich, H.-J., "Self-Test Using Unequiprobable Random Patterns," *Proc. of FTCS-17*, pp. 258-263, 1987.

[Zacharia 95] Zacharia, N., J. Rajski, and J. Tyszer, "Decompression of Test Data Using Variable-Length Seed LFSRs," *Proc. of VLSI Test Symposium*, pp. 426-433, 1995.