BETSY: Synthesizing Circuits for a Specified BIST Environment

Zhe Zhao[1], Bahram Pouya[1,2], and Nur A. Touba[1]

[1]Computer Engineering Research Center
Dept. of Electrical and Computer Engineering
University of Texas, Austin, TX 78712

[2]Test Technology Group
Advanced Products Research & Development Lab
Motorola, Austin, TX 78721

Abstract

This paper presents a logic synthesis tool called BETSY (BIST Environment Testable SYnthesis) for synthesizing circuits that achieve complete (100%) fault coverage in a user specified BIST environment. Instead of optimizing the circuit for a generic pseudo-random test pattern generator (by maximizing its random pattern testability), the circuit is optimized for a specific test pattern generator, e.g., an LFSR with a specific characteristic polynomial and initial seed. This solves the problem of having to estimate fault detection probabilities during synthesis and guarantees that the resulting circuit achieves 100% fault coverage. BETSY considers the exact set of patterns that will be applied to the circuit during BIST and applies various transformations to generate an implementation that is fully tested by those patterns. When needed, BETSY inserts test points early in the synthesis process in an optimal way and accounts for them in satisfying timing constraints and other synthesis criteria. Experimental results are shown which demonstrate the benefits of optimizing a circuit for a particular test pattern generator.

1. Introduction

In a built-in self-test (BIST) environment, on-chip hardware is used to apply test patterns to the circuit-under-test (CUT). The most common approach is to use a pseudo-random test pattern generator (e.g., an LFSR) because of its low hardware overhead. The set of patterns that are applied to the CUT during BIST will be referred to as the "BIST pattern set." The BIST pattern set depends on the test pattern generator and on the test length. One problem that arises is that the BIST pattern set may not detect all of the faults in the CUT. If the fault coverage is insufficient, then steps must be taken to either add test points to the CUT to enable more faults to be detected by the BIST pattern set, or to modify the test pattern generator to change the BIST pattern set so that more faults are detected. In many cases, a substantial amount of hardware overhead must be added to obtain sufficient fault coverage with logic BIST.

The conventional approach for designing a circuit with BIST is to first synthesize the circuit, and then add a test pattern generator (e.g., an LFSR feeding a scan chain) and perform fault simulation for the BIST pattern set. If the fault coverage is insufficient, then test points must be inserted until the fault coverage requirement is achieved.

Chiang and Gupta [Chiang 94ab] observed that synthesized circuits typically require much longer random pattern test lengths to achieve high fault coverage compared with manually-optimized circuits even though the circuits implement the same logic function. These results indicate that the structure of a circuit greatly influences its random pattern testability.

One approach for improving the fault coverage for pseudo-random BIST is to consider random pattern testability during the synthesis process. The idea is to guide the synthesis process so that it minimizes the number of random pattern resistant faults in the resulting implementation. This results in higher fault coverage for the BIST pattern set and thus reduces the overhead and complexity of the test point circuitry required for satisfying fault coverage requirements in a BIST environment. Synthesis procedures that attempt to maximize random pattern testability have been proposed in [Touba 94], [Chiang 94b], and [Chatterjee 95]. One of the major difficulties in trying to maximize random pattern testability is that evaluating the random pattern testability of a circuit is very difficult. It requires determining the detection probabilities of the faults in the circuit. To reduce the complexity of this problem, the procedures described in [Touba 94] and [Chiang 94b] require that the starting point for the synthesis be a two-level circuit so that cube calculus operations can be used to determine the detection probabilities. However, some circuits have an exponential two-level representation and thus these techniques cannot be used in that case. The procedure described in [Chatterjee 95] does not require a two-level starting point, but rather uses techniques to estimate the detection probabilities. The drawback of this approach is that the estimations can be very inaccurate at times which

0-7803-5092-8/98 $10.00

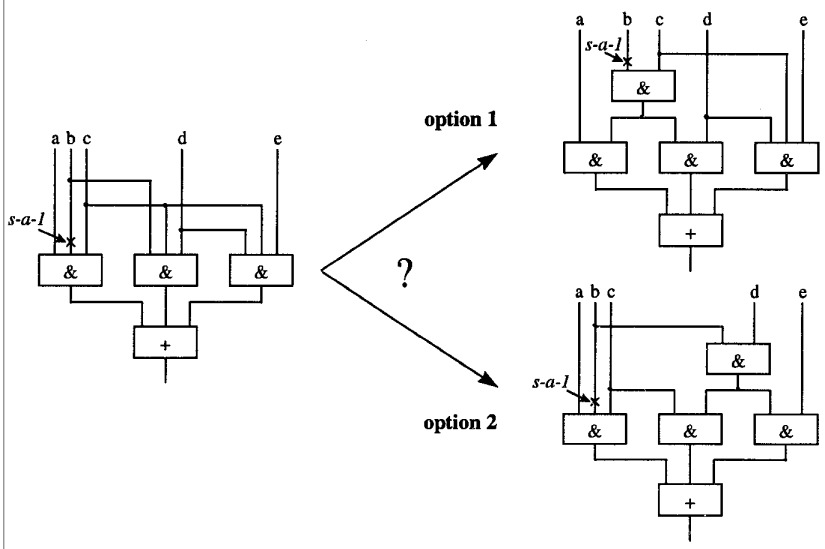

Figure 1. Example of Decision Between Two Options for Factoring Logic

reduces the effectiveness of the synthesis procedure. Moveover, the cost function used in [Chatterjee 95] to guide the synthesis does not guarantee any particular level of random pattern testability in the final implementation.

This paper presents a new logic synthesis tool called BETSY (BIST Environment Testable SYnthesis) which takes a completely new approach for synthesizing random circuits. The key idea is to optimize the circuit for a *specific* BIST test pattern generator. Instead of optimizing the circuit for a generic pseudo-random test pattern generator (by maximizing its random pattern testability), the circuit is optimized for a specific test pattern generator, e.g., an LFSR with a specific characteristic polynomial and initial seed. The BIST test pattern generator is chosen before synthesizing the circuit. So rather than trying to maximize the probability that faults in the circuit are detected by random patterns, BETSY considers the exact set of patterns that will be applied to the circuit during BIST and optimizes the testability of the circuit for those patterns. This approach provides a number of advantages which are summarized below:

1. Better decisions can be made during the synthesis process. Previous techniques which maximize the random pattern testability make synthesis decisions based on improving the detection probability for the random-pattern-resistant (r.p.r.) faults in the circuit structure. However, some r.p.r. faults may be detected by a specific test pattern generator already. For example, the initial seed of an LFSR can be chosen so that it detects some of the r.p.r. faults. Thus, there is no need to direct the synthesis towards improving the detection probability of those faults. Another example of how better synthesis decisions can be made by BETSY is illustrated by the simple example in Fig. 1. There are two options for factoring the circuit on the left-hand side. Both options have the same literal count and same random pattern testability (i.e., same fault detection probabilities), however, if the BIST pattern set contains the pattern 00110, but does not contain any pattern of the form 101XX, then the stuck-at 1 fault that is shown would be detected in option 1, but would be undetected in option 2. If option 2 were chosen, a test point would need to be inserted to achieve 100% coverage, whereas if option 1 is chosen, no test point would be required. This is a very simple example of how considering the BIST pattern set during the synthesis process enables better optimization decisions to be made.
2. No need to compute fault detection probabilities. Since the exact BIST pattern set is known by BETSY, fault simulation can be done to determine exactly which faults in the circuit structure are not detected. The synthesis process can then be directed towards logic optimizations that change the circuit structure in a way that enables the undetected faults to be detected. There is no need to determine fault detection probabilities, and hence there are no misdirected synthesis decisions due to inaccurate estimates of fault detection probabilities. Moreover, the synthesis procedure can guarantee the final fault coverage. Previous techniques only optimize fault detection probabilities and thus cannot guarantee the final fault coverage: the fault coverage must be verified after synthesis and additional test points must be inserted if it is insufficient.

3. Test point insertion can be done early in the synthesis process. BETSY can analyze the faults that are not detected by the BIST pattern set and determine whether it is possible to factor the logic in a way that will eliminate the faults. For undetected faults that cannot be eliminated through factoring, and thus will require test points in the final implementation, the test points can be inserted very early in the synthesis process. This provides two benefits: one is that it allows the logic to be restructured in a way that maximizes the effectiveness of each test point, and the other is that it allows the test points to be taken into consideration when the synthesis procedure optimizes delay. Conventional "post-synthesis" test point insertion procedures, which modify the circuit after synthesis, can cause timing problems.
4. Applicable for any type of test pattern generator, not just pseudo-random. Because BETSY is not based on detection probability, but rather based on the exact BIST pattern set, there is no requirement that the BIST patterns be pseudo-random. Thus, BETSY can be used in the case where a multiple-input signature register (MISR) is used as a test pattern generator, or even where the output patterns from some module *A* (which may have many correlations) are used to test module *B*. BETSY can optimize the testability of the circuit for any BIST pattern set.

There are two ways that BETSY can be used. One way is to start with an unoptimized circuit description and use BETSY globally to generate an optimized implementation. BETSY directly generates an implementation that will satisfy fault coverage requirements in the specified BIST environment. If test points are required, BETSY inserts them early in the synthesis process in an optimal way and accounts for them in satisfying timing constraints and other synthesis criteria. Considering testability requirements during synthesis (as opposed to the traditional approach of making back-end modifications after an implementation has already been generated), reduces design time, design mistakes, and test overhead.

The other way that BETSY can be used is as a replacement for conventional test point insertion procedures. Given an already optimized circuit (which may have been synthesized by another tool or may have been manually-designed) for which a BIST test pattern generator cannot provide sufficient fault coverage, BETSY can be used to modify the circuit to improve the fault coverage. Fault simulation can be performed to identify regions in the circuit where many faults are not detected by the BIST pattern set. The logic in those regions can be locally "resynthesized" using the proposed procedure in order to minimize the number of test points needed to satisfy fault coverage requirements. Rather than using conventional test point insertion procedures to insert test points in a fixed circuit structure, the idea here is to restructure local regions in the circuit to minimize the number of test points that need to be inserted. BETSY uses flexibility in restructuring logic to minimize the number of test points required, and thus minimize BIST overhead.

This paper is organized as follows: Section 2 defines some of the terms and concepts used in the paper. Section 3 describes the transformations used by BETSY to improve testability. Section 4 presents BETSY's global synthesis procedure for systematically applying transformations to satisfy fault coverage requirements. Section 5 describes how BETSY can be used as a replacement for conventional test point insertion procedures to locally resynthesize an optimized circuit to improve its testability. Section 6 shows experimental results for using BETSY on random-pattern-resistant benchmark circuits. Section 7 is a conclusion.

2. Preliminaries

A multilevel logic circuit can be represented by a Boolean network [Brayton 90] in which each node is a sum-of-products. Multilevel logic synthesis involves factorizing the Boolean network to restructure and optimize it. There are many different ways to factorize a given Boolean network which lead to different implementations. Various criteria (e.g., area, delay, power, etc.) are used to guide the factorization process to satisfy design objectives. If the circuit is to be tested in a BIST environment, then one of the design objectives is to obtain an implementation that can be adequately tested by the BIST hardware (as well as satisfying area, timing, power, and other constraints).

Logic transformations can be classified as Boolean transformations and algebraic transformations. *Boolean transformations* make use of the fact that $(a + a' = 1)$ and $(aa' = 0)$, while *algebraic transformations* do not. Boolean transformations are much more difficult to identify than algebraic transformations.

Fanout in a Boolean network occurs only at the output of nodes (primary inputs are considered nodes). Each node itself is a fanout-free subcircuit that implements a sum-of-products. Thus, each fault in a Boolean network is *equivalent* (i.e., detected by the same set of test vectors) to one of the following four types of faults: a literal in the sum-of-products for a node being stuck-at 1 (S-A-1), a cube in the sum-of-products for a node being stuck-at 0 (S-A-0), or the output of a node being S-A-1 or S-A-0. Assuming a tree-covering technology mapping procedure [Keutzer 87], [Detjens 87], is used, then each fault in the final implementation will be equivalent to a fault in the

Boolean network. Thus, only the four types of faults described above need to be considered.

A fault is said to be "BIST detected" if it is detected by the BIST pattern set, and "BIST undetected" if it is not. Given the BIST pattern set, fault simulation can be performed to identify all BIST undetected faults in a Boolean network. The goal of BETSY is to optimize the Boolean network for area, delay, power, etc., under the constraint that the final implementation must satisfy BIST fault coverage requirements. The conventional approach is to do normal synthesis and then perform "post-synthesis" test point insertion to improve the BIST fault coverage. In the conventional test point insertion approach, only two types of transformations are used to improve BIST fault coverage, namely control point insertion and observation point insertion. Both control points and observation points require additional scan elements to be added to the design (to drive the control points and to capture the response of the observation points). BETSY uses several transformations that are much more efficient for improving BIST fault coverage. The transformations are integrated with the factorization process to better optimize the resulting implementation.

3. Logic Transformations For Improving BIST Fault Coverage

This section describes several logic transformations that are used by BETSY to improve BIST fault coverage. Systematic procedures for applying these transformations to satisfy fault coverage requirements while optimizing the overall design will be described in subsequent sections. The transformations described here restructure the Boolean network in a way that eliminates BIST undetected faults.

3.1 Factoring Single Cube Algebraic Divisors

Factoring out a common cube among two or more cubes can be used to eliminate BIST undetected S-A-1 faults in the literals of a sum-of-products for a node. Consider the example shown in Fig. 2. The S-A-1 faults in l_1 and l_2 are combined into a single fault, l_a S-A-1 in the common cube after it is factored out. If either the l_1 or l_2 S-A-1 faults in the original Boolean network is BIST detected, then the l_a S-A-1 fault in the common cube will also be BIST detected. So given a BIST undetected S-A-1 fault in literal l of cube c_1 in a sum-of-products for a node, it can be eliminated by factoring out a single cube divisor that it has in common with some cube c_2 in which literal l S-A-1 is BIST detected.

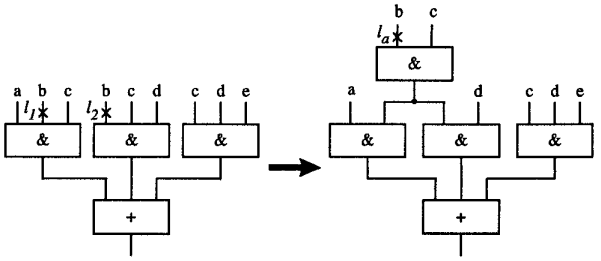

Figure 2. Factoring Out Single Cube: If literal l_1 S-A-1 or literal l_2 S-A-1 is BIST detected, then literal l_a S-A-1 will be BIST detected.

3.2 Factoring Double Cube Algebraic Divisors

Factoring out a common double cube divisor among two nodes can be used to eliminate both BIST undetected S-A-1 faults in literals as well as BIST undetected S-A-0 faults in cubes. Consider the example shown in Fig. 3. If either the S-A-0 fault in cube c_1 (c_2) or cube c_3 (c_4) is BIST detected in the original Boolean network, then the S-A-0 fault c_a (c_b) will be BIST detected in the common double cube divisor after factoring it out. Moreover, if either the S-A-0 fault in cube c_1 (c_3) or cube c_2 (c_4) is BIST detected in the original Boolean network, then the S-A-0 fault c_c (c_d) will be BIST detected. So given a BIST undetected S-A-0 fault in some cube c_1 in a node n_1, it can be eliminated by factoring out a double cube divisor that it has in common with another node n_2 provided the S-A-0 fault in the second cube of the double cube divisor is BIST detected and the S-A-0 fault in the corresponding cube in the node n_2 is BIST detected.

3.3 Boolean Resubstitution

Boolean resubstitution involves creating a fanout from node n_1 to another node n_2. This is possible if node n_1 is a Boolean divisor of node n_2 [Brayton 90]. Efficient techniques for checking for special types of Boolean resubstitution using ATPG [Entrena 95], [Chang 96], or recursive learning [Chatterjee 95] have been developed. These techniques involve making a connection from one node to the input of another node in a way that the logic function of the Boolean network is unchanged. This results in some lines becoming redundant which are then subsequently removed with a redundancy removal procedure. In some cases, Boolean resubstitution reduces the size of the Boolean network and in other cases it increases it. Creating fanout from node n_1 to another node n_2 increases the observability of node n_1 and thus may eliminate BIST undetected faults in the cone of logic feeding node n_1. However, unlike algebraic factoring,

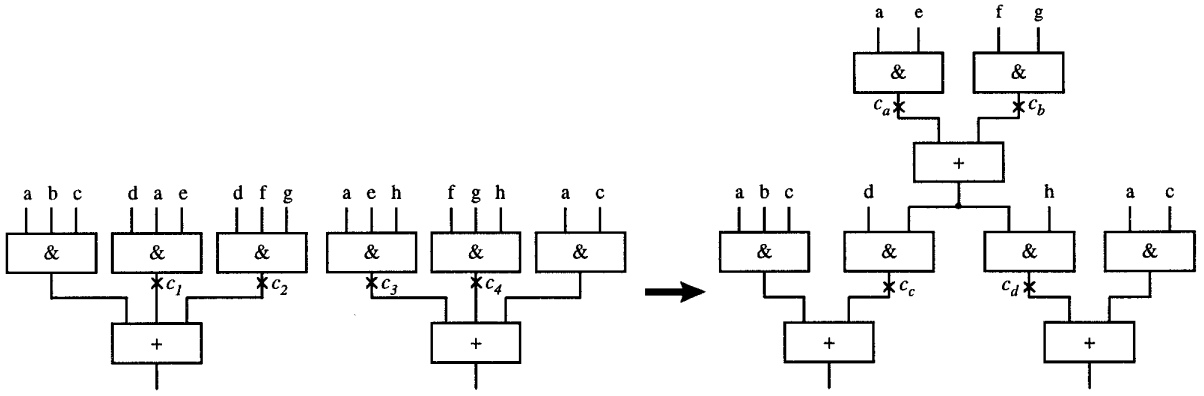

Figure 3. Factoring Out Double Cube: If cube c_1 S-A-0 or cube c_3 S-A-0 is BIST detected, then cube c_a S-A-0 will be BIST detected. If cube c_1 S-A-0 or cube c_2 S-A-0 is BIST detected, then cube c_d S-A-0 will be BIST detected.

Boolean resubstitution does not monotonically increase the detectability of faults in the circuit. It may cause some faults to have better detectability and others to have worse detectability. Boolean resubstitution must be used carefully to eliminate existing BIST undetected faults without causing several other faults to become BIST undetected. Techniques for beneficially exploiting the power of Boolean resubstitution will be described in subsequent sections.

3.4 Factoring Out Algebraic Divisors with Observation Points

When a BIST redundant fault cannot be eliminated by factoring, then test points must be inserted. Conventional test point insertion procedures select an existing line in the circuit to insert either a control or observation point. BETSY identifies BIST redundant faults that require test points early in the synthesis process before the circuit has been fully factorized. One advantage of this is that if several BIST redundant faults require test points in order to be detected, common factors that contain those faults can be factored out such that the effectiveness of an observation point in eliminating BIST redundant faults can be maximized. This results in fewer test points in the final implementation.

3.5 Factoring Out Algebraic Divisor with Control Points

As with observation point insertion, combining control point insertion with factoring can be used to increase the effectiveness of each control point. Consider the example in Fig. 4. By factoring out a common cube, a single control point can be used to increase the controllability of all three nodes x, y, and z.

One issue with inserting control points is that depending on how the pseudo-random pattern generator is configured, adding an additional scan element to drive the control point may alter the pseudo-random patterns that are applied to the circuit which would then change the BIST pattern set. This problem can be avoided by using either the multi-phase control point scheme described in [Tamarapalli 96], or using pattern decoding logic to drive the control points as described in [Touba 96]. Neither of those techniques require the insertion of additional scan elements. The scheme in [Tamarapalli 96] activates the control points as a function of the pattern counter, and the scheme in [Touba 96] activates the control points as a function of other primary inputs.

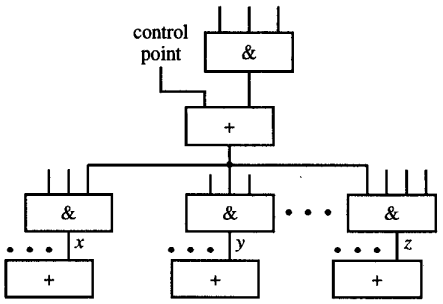

Figure 4. Factoring Out Common Cube Enables Control Point to Improve Controllability at x, y, and z

4. Global Synthesis Procedure

Given a Boolean network and a BIST pattern set, BETSY can be used to synthesize an optimized implementation that satisfies BIST fault coverage requirements. The initial Boolean network may be flat, or it may have an initial structure. Conventional synthesis procedures factorize the Boolean network based on optimizing various criteria such as area, delay, power, etc. The concurrent decomposition and factorization procedure

described in [Rajski 92] can be used to efficiently identify all single and double cube divisors. The divisors can then be factored out one at a time based on the synthesis criteria. For example, if optimizing for area, then the divisors that reduce the literal count the most could be factored out first. It was shown in [Rajski 92] that the factorization procedure is test-set preserving which means that the factorized Boolean network can be tested by the same set of test vectors as the original Boolean network. Thus, the factorization process will not introduce new BIST undetected faults. The strategy used in BETSY is to first transform the Boolean network to eliminate the BIST undetected faults, then once all faults in the Boolean network can be tested by the BIST pattern set, conventional factorization can proceed to optimize the other synthesis criteria without introducing new BIST undetected faults. All faults in the final optimized Boolean network will be detected by the BIST pattern set.

The first step in BETSY is to do fault simulation for the BIST pattern set to identify which faults in the Boolean network are BIST undetected. As was explained earlier, there are only 4 types of faults that need to be considered. S-A-1 in a literal, S-A-0 in a cube, S-A-1 at the output of a node, and S-A-0 at the output of a node. Note that the only *stem faults* (faults on a line that fans out) in the circuit are at the outputs of a node.

4.1 Eliminating BIST Undetected Faults at Output of Nodes

Once the BIST undetected faults have been identified, then BETSY transforms the Boolean network to eliminate them. BIST undetected faults in either a literal or cube of a node can be eliminated through algebraic factoring, but BIST undetected faults at the output of a node cannot.

Consider a BIST undetected fault at the output of a node n. There are 4 cases:

Case 1: No fanout - The output of the node n connects directly to the input of another node p with no fanout. In this case, the fault is equivalent to either a literal S-A-1 or a cube S-A-0 in node p. Thus, it can be removed from consideration.

Case 2: Dominating stem fault - The output of node n fans out to nodes $p_1 \ldots p_n$, but there exists at least one path from node n to a primary output that does not reconverge with another path from n with opposite inversion parity. In this case, the fault at the output of node n is a dominating stem fault (stem fault whose set of test vectors is a superset of one of the branch faults). Thus, it can be removed from consideration since detecting the branch fault will guarantee detection of the stem.

Case 3: Non-dominating stem fault - The output of node n fans out to nodes $p_1 \ldots p_n$, and each path from n through p_i reconverges with another path from n through p_j with different inversion parity. In this case, the fault at the output of node n is a non-dominating stem fault. If node n is not a primary input, then BETSY collapses node n into nodes $p_1 \ldots p_n$ (by substituting the sum-of-products for node n into the sum-of-products for nodes $p_1 \ldots p_n$), thereby eliminating node n. This does not eliminate the BIST undetected fault, but rather collapses it into nodes $p_1 \ldots p_n$ so that it is possible to subsequently factor those nodes in a different way which would not lead to a BIST undetected fault or would lead to fewer test points in the final implementation. For example, consider the circuit shown back in Fig. 1, if the original Boolean network was like option 2, then by collapsing it and factoring it again, it could be transformed to option 1 where there are no BIST undetectable faults.

Case 4: Primary input or primary output fault - If node n is a primary input or feeds a primary output, then the fault will require a test point in order to be detected. The set of test vectors that detects a fault at the primary inputs or primary outputs is a property of the Boolean function and is not implementation-dependent; the fault would be BIST undetected in any implementation of the Boolean function. The only way to enable such a fault to be detected by the BIST pattern set is to augment the Boolean function by inserting a test point. Thus, BETSY marks the fault as requiring a test point.

4.2 Eliminating BIST Undetected Faults within Nodes

Once all of the BIST undetected faults at the output of a node have been either collapsed or marked as requiring a test point, then all the remaining BIST undetected faults are either literal or cube faults in a node. BETSY determines which of those faults can be eliminated through factoring and which will require test points. This is done by identifying single and double cube divisors that can be factored out to eliminate each BIST undetected fault. Concurrent decomposition [Rajski 92] is used to quickly identify all such factors. BIST undetected faults that cannot be eliminated through factoring are marked as requiring test points.

4.3 Boolean Resubstitution

For the faults marked as requiring test points, BETSY attempts to eliminate them through Boolean resubstitution. BETSY does fault simulation of the BIST pattern set only for the faults requiring test points and records the nodes that the effects of each fault propagate to. Increasing the observability of those nodes may allow the faults to be detected without test points, so Boolean resubstitution is considered for those nodes. Boolean resubstitution creates additional fanout from those nodes which increases

their observability. The transformations described in [Chatterjee 95] are performed for the candidate nodes and redundancy removal is used to eliminate the resulting redundant lines. Unlike single and double cube factorization [Rajski 92], Boolean resubstitution may cause other faults to become BIST undetected. Thus, BETSY performs fault simulation to evaluate the resulting Boolean network. If the new Boolean network has fewer BIST undetected faults requiring test points, then the transformations are kept. If not, then BETSY reverts back to the original Boolean network. By evaluating where the new BIST undetected faults appeared after performing the previous set of Boolean resubstitutions, BETSY can iteratively choose a different set of Boolean resubstitutions to try the next time. The number of times Boolean resubstitution is attempted can be limited by the user to control runtime. BETSY only keeps the transformations if the resulting Boolean network is an improvement over the original.

4.4 Inserting Test Points

For the faults marked as requiring test points, BETSY identifies factors that minimize the number of test points that need to be inserted. Observation points are preferred because they do not have as much effect on timing. However, some BIST undetected faults may not be provoked by the BIST pattern set and thus require control points in order to be detected. BETSY does fault simulation of the BIST pattern set for the faults requiring test points and determines if each fault is provoked to the output of a gate and if so it records all the nodes that the effects of the fault propagate to. For BIST undetected faults that require control points, BETSY uses concurrent decomposition [Rajski 92] to find single and double cube divisors that maximize the number of faults that are provoked by each control point and thereby minimize the number of control points that are required.

After all the necessary control points have been inserted, BETSY does fault simulation of the BIST pattern set to verify that all faults are provoked and to record the nodes that the effect of each BIST undetected fault propagates to. If an observation point is inserted on a path that the effect of a fault propagates to (for the BIST pattern set), then the fault will be detected. Each fault has some set of nodes that it propagates to. Conventional observation point insertion procedures use a set covering procedure to select the minimum number of observation points that cover all the faults [Iyengar 89]. However, BETSY factors the Boolean network in a way that further reduces the number of observation points that are required. Consider the example in Fig. 5. There are three BIST undetected faults shown. The path that the effects of each fault propagate down are highlighted in bold (fault propagation is blocked at the OR gates in all three cases).

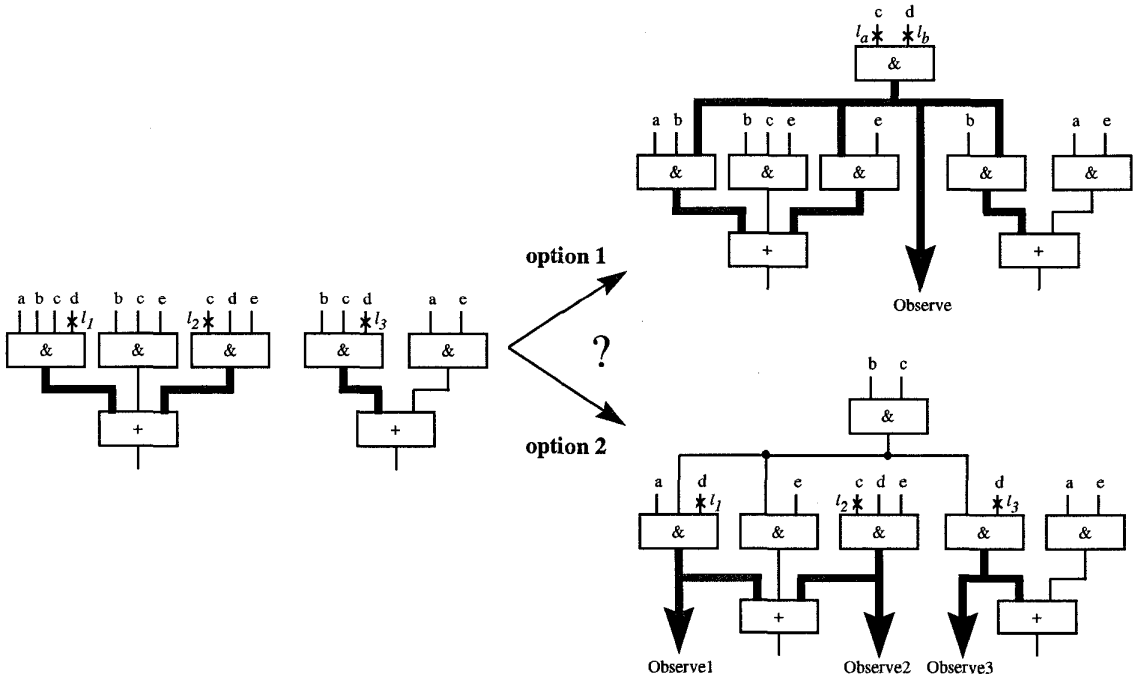

Figure 5. Example of Reducing Number of Observation Points Through Factoring (Bold Lines Indicate Fault Propagation Path for Some Test Pattern in the BIST Pattern Set)

If the cube *cd* is factored out, then only one observation point is sufficient to allow all three faults to be detected. However, if the cube *bc* was factored out, then three observation points would be needed. For each BIST undetected fault, BETSY uses concurrent decomposition [Rajski 92] to factor single and double cube divisors that reduce the number of observation points that need to be inserted.

After a sufficient set of test points has been inserted, then the circuit is factorized to eliminate all remaining BIST undetected faults. Once all BIST undetected faults have been eliminated, conventional factorization can proceed to optimize the other synthesis criteria without introducing new BIST undetected faults. All faults in the final optimized Boolean network will be detected by the BIST pattern set.

5. Local Transformation Procedure

In some cases, it may not be desirable to globally synthesize a circuit with BETSY. In that case, BETSY can be used as a replacement for conventional test point insertion procedures. The idea is to identify regions that contain BIST undetected faults and use BETSY to locally transform those regions to minimize the number of test points.

The simplest application is to use BETSY to attempt Boolean resubstitution. The procedure described in Sec. 4.3 can be used. Fault simulation identifies the nodes that the effects of BIST undetected fault propagate to, and Boolean resubstitution is attempted for those nodes. If the resulting Boolean network requires fewer test points, then it is kept. If not, then BETSY analyzes the effects of the set of Boolean resubstitutions and tries to identify a better set of Boolean resubstitutions. This process can be iterated as desired to find the best Boolean network.

A more powerful application of BETSY is to selectively collapse parts of the initial optimized Boolean network and then resynthesize it. The key is to choose which nodes to collapse so as not to alter the optimized Boolean network more than necessary. BETSY accomplishes this in the following way. It first does fault simulation of the BIST pattern set to identify the BIST undetected faults. BIST undetected faults at the output of nodes are eliminated by collapsing as described in Sec. 4.1. For literal and cube faults, BETSY traces back from the fault site to the first fanout branch point. The node whose output is the stem of the fanout branch is collapsed into all of the nodes that it fans out to. The number of times that this backtracing and collapsing process is repeated can be specified by the user depending on how much restructuring of the initial optimized Boolean network is desired. After collapsing the logic closely related to the BIST undetected faults, BETSY resynthesizes that portion of the Boolean network using the procedure described in the previous section. This allows factorization and test point insertion to be combined in a way that better optimizes the final implementation compared to using conventional test point insertion procedures.

6. Experimental Results

Experiments were performed to compare BETSY with conventional post-synthesis test point insertion. In Table 1, results are shown for synthesizing unoptimized benchmark circuits with SIS (which is an updated version of MIS [Brayton 87]) using **script.rugged** (which uses the concurrent decomposition and factorization procedure [Rajski 92]). The pseudo-random test length required to achieve 100% fault coverage for each implementation is shown along with the literal count (number of factored-form literals in the Boolean network). All of the benchmark circuits in Table 1 synthesized with SIS require a test length of over 100K patterns. A test point insertion tool was then used to insert a sufficient set of test points in each circuit to achieve 100% fault coverage for a pseudo-random BIST pattern set containing 10K patterns. The number of control and observation points inserted for each circuit is shown in Table 1. The same BIST pattern set and the initial circuit description was then given as an input to BETSY. An implementation was generated by BETSY which provides 100% fault coverage. The literal count and number of control and observation points is shown. As can be seen, BETSY generated implementations that require fewer test points with only a modest increase in the total literal count. The increase in the literal count results from choosing some factors on the basis of minimizing the number of test points instead of minimizing the literal count.

In Table 2, results are shown for using BETSY as a replacement for conventional test point insertion procedures. BETSY was used to locally transform some large benchmark circuits so that they achieved 100% fault coverage for a pseudo-random BIST pattern set with 32K patterns. The literal count and number of control and observation points that were inserted are shown for the implementations generated by BETSY. The results published in [Tamarapalli 96] are shown for comparison. These results demonstrate the power of optimizing a circuit for the particular test pattern generator that will be used.

Table 1. Results for Global Synthesis Procedure on Unoptimized Benchmark Circuits
(Fault Coverage = 100%)

Circuit		SIS		SIS + TPI			BETSY			
Name	Num PI	Test Length	Lits	Test Length	Control Points	Obsv. Points	Test Length	Lits	Control Points	Obsv. Points
apex6	135	710K	764	10K	2	3	10K	826	1	2
chkn	29	33M	332	10K	3	3	10K	394	2	2
in3	35	800K	325	10K	2	4	10K	363	2	1
vtx1	27	290K	90	10K	2	0	10K	103	2	0
x1	51	100K	302	10K	2	1	10K	329	2	0
x2dn	82	140K	194	10K	4	1	10K	202	1	1
x9dn	27	300K	107	10K	2	2	10K	120	2	0

Table 2. Results for Local Transformations on Optimized Benchmark Circuits
(Fault Coverage = 100%)

Circuit				[Tamarapalli 96]			BETSY			
Name	Num PI	Test Length	Lits	Test Length	Control Points	Obsv. Points	Test Length	Lits	Control Points	Obsv. Points
C2670	233	4.6M	810	32K	1	5	32K	842	2	2
C7552	207	>100M	2445	32K	18	2	32K	2624	6	6
s9324	247	>100M	2231	32K	8	10	32K	2413	5	9
s15850	611	>100M	4056	32K	15	17	32K	4495	12	10

7. Conclusions

Merging test point insertion into the synthesis process provides a number of advantages in terms of reducing the number of test points required and allowing better area and timing optimizations to be done. Knowledge of the BIST pattern set can be used to overcome the difficulties of predicting random pattern testability during synthesis. The techniques used by BETSY exploit the knowledge of the BIST pattern set to make better synthesis decisions.

Acknowledgements

This material is based on work supported in part by the Defense Advanced Research Projects Agency (DARPA) under Contract No. DABT63-94-C-0045, in part by the National Science Foundation under Grant No. MIP-9702236, and in part by the Texas Advanced Research Program under Grant No. 1997-003658-369.

References

[Brayton 87] Brayton, R.K., R. Rudell, A. Sangiovanni-Vincentelli, A.R. Wang, "MIS: A Multiple-Level Logic Optimization System," *IEEE Transactions on Computer-Aided Design*, Vol. 6, Nov. 1987, pp. 1062-1081.

[Brayton 90] Brayton, R.K., G.D. Hachtel, and A.L. Sangiovanni-Vincentelli, "Multilevel Logic Synthesis," *Proceedings of the IEEE*, Vol. 78, No. 2, pp. 264-300, Feb. 1990.

[Chatterjee 95] Chatterjee, M., D.K. Pradhan, and W. Kunz, "LOT: Logic Optimization with Testability - New Transformations using Recursive Learning," *Proc. of Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 318-325, 1995.

[Chang 96] Chang, S.C., M. Marek-Sadowska, and K.-T. Cheng, "Perturb and Simplify: Multilevel Boolean Network Optimizer," *IEEE Trans. on Computer-Aided Design*, Vol. 15, No. 12, pp. 1494-1504, Dec. 1996.

[Chiang 94a] Chiang C.-H., and S.K. Gupta, "Random Pattern Testable Logic Synthesis," Technical Report CENG 94-08, University of Southern California, 1994.

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

[Detjens 87] Detjens, E., G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology Mapping in MIS," *Proc. of Int. Conference on Computer-Aided Design (ICCAD)*, pp. 116-119, 1987.

[Entrena 95] Entrena, L.A., and K.T. Cheng, "Combinational and Sequential Logic Optimization by Redundancy Addition and Removal," *IEEE Transactions on Computer-Aided Design*, Vol. 14, No. 7, pp. 909-916, July 1995.

[Keutzer 87] Keutzer, K., "Dagon: Technology Binding and Local Optimization by DAG Matching," *Proc. of 24th Design Automation Conf.*, pp. 341-347, 1987.

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

[Rajski 92] Rajski, J., and J. Vasudevamurthy, "The Testability-Preserving Concurrent Decomposition and Factorization of Boolean Expressions," *IEEE Trans. on Computer-Aided Design*, Vol. 11, No. 6, Jun. 1992, pp. 778-793.

[Tamarapalli 96] Tamarapalli, N., and J. Rajski, "Constructive Multi-Phase Test Point Insertion for Scan-Based BIST," *Proc. of International Test Conference*, pp. 649-658, 1996.

[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 96] Touba, N.A., and E.J. McCluskey, "Test Point Insertion Based on Path Tracing," *Proc. of VLSI Test Symposium*, pp. 2-8, 1996.