AUTOMATED LOGIC SYNTHESIS OF RANDOM PATTERN TESTABLE CIRCUITS

Nur A. Touba and Edward J. McCluskey

Center for Reliable Computing
Departments of Electrical Engineering and Computer Science
Stanford University
Stanford, California 94305-4055

ABSTRACT

Previous approaches to designing random pattern testable circuits use post-synthesis test point insertion to eliminate random pattern resistant (r.p.r.) faults. The approach taken in this paper is to consider random pattern testability during logic synthesis. An automated logic synthesis procedure is presented which takes as an input a two-level representation of a circuit and a constraint on the minimum fault detection probability (threshold below which faults are considered r.p.r.) and generates a multilevel implementation that satisfies the constraint while minimizing the literal count. The procedure identifies r.p.r. faults and attempts to "eliminate" them through algebraic factoring. If that is not possible, then test points are inserted during the synthesis process in a way that minimizes the number of test points that are required. Results are shown for benchmark circuits which indicate that the proposed procedure can generally reduce the random pattern test length by at least an order of magnitude with only a small area overhead.

1. INTRODUCTION

Random pattern testing has a number of well-known advantages: no deterministic test set generation cost, no test pattern storage requirement, higher coverage of non-targeted faults, and high suitability for built-in self test (BIST). The obvious drawback to random pattern testing is that longer test lengths are needed. For some circuits, the test length required to get high fault coverage with random patterns is unacceptably long.

The random pattern test length required to give a particular fault coverage for a circuit depends on the detection probability of each fault in the circuit. The *detection probability* of a fault is equal to the number of input patterns that detect the fault divided by the total number of input patterns, 2^n, where n is the number of primary inputs. Faults with very low detection probabilities are said to be *random pattern resistant (r.p.r.)* because they are hard to detect with random patterns [Eichelberger 83]. A circuit that does not have any r.p.r. faults is *random pattern testable*.

Given a circuit structure that has r.p.r. faults, two general solutions have been proposed. One is to use weighted pattern generation to bias the input patterns towards those that detect the r.p.r. faults. A drawback to this approach is that on-chip weighted pattern generators can be costly to implement, especially when multiple weight sets are required. The other alternative is to modify the circuit structure by adding test points to increase the detection probability of the r.p.r. faults so that they are no longer r.p.r. (i.e., "eliminate" the r.p.r. faults). This requires identifying the r.p.r. faults and trying to add as few test points as possible to eliminate them. Since a single test point can change the detection probability of a number of faults, optimal placement of test points has been an active area of research [Krishnamurthy 87], [Iyengar 89], [Seiss 91], [Savaria 91], [Youssef 93].

Whereas previous methods for designing random pattern testable circuits involve post-synthesis test point insertion, this paper approaches the problem by considering random pattern testability during logic synthesis. An automated logic synthesis procedure is described which takes as an input a two-level representation of a circuit and a constraint on the minimum fault detection probability and generates a multilevel circuit implementation that satisfies the constraint while minimizing the literal count. The minimum fault detection probability constraint essentially defines a threshold below which a fault is considered r.p.r. The central strategy is to identify any r.p.r. faults in the two-level starting point, and then find algebraic factors that eliminate these faults. Once the r.p.r. faults have been eliminated, normal logic optimization using random pattern testability preserving logic transformations (defined in Sec. 3) can then proceed since such transformations will not introduce new r.p.r. faults. Thus, the initial selection of algebraic factors has as its primary goal the elimination of r.p.r. faults, and then once all the r.p.r. faults have been eliminated, subsequent factors are chosen on the basis of reducing the literal count or optimizing for other synthesis criteria (e.g., delay).

As the minimum fault detection probability threshold is increased, a point is reached where some r.p.r. faults cannot be eliminated by algebraic factoring alone. When this is the case, test points are inserted during the synthesis process in order to generate a random pattern testable implementation. Factors are chosen to enable each test point to eliminate several r.p.r. faults so that the

total number of test points needed is minimized. Results are shown which indicate that the minimum fault detection probability can be significantly increased by adding just a few test points during synthesis. Thus, the procedure described in this paper can be used to design circuits which require a much shorter random pattern test length without substantial overhead.

The paper is organized as follows: In Sec. 2, a technique that utilizes special properties of algebraic factorization to efficiently compute fault detection probabilities is described. In Sec. 3, random pattern testability preserving transformations are defined and are shown to be a superset of test-set preserving transformations. In Sec. 4, the proposed logic synthesis procedure is presented. In Sec. 5, an automated procedure is described for inserting test points during the synthesis process. In Sec. 6, results for benchmark circuits are shown and discussed. Section 7 is a summary and conclusion.

The following terminology is used in this paper: A *literal* is a boolean variable or its complement. A *cube* is a set of literals interpreted here as a product of literals. A *cover* is a set of cubes interpreted here as a sum-of-products expression.

2. COMPUTING FAULT DETECTION PROBABILITIES

The proposed logic synthesis procedure involves identifying r.p.r. faults and finding factors that eliminate these faults. This requires computing fault detection probabilities which is an NP-hard problem [Krishnamurthy 86]. Many methods exist for trading off accuracy to reduce computation time. However, during logic synthesis, the structure of the circuit is constantly changing which presents additional difficulty. To cope with this problem, the proposed procedure relies on the following property of algebraic factorization.

Definition 1: The *detecting set* for a fault is the set of input patterns that detect the fault.

Definition 2: Two faults are *equivalent* if they have the same detecting set.

Property 1: Each stuck-at fault in a multilevel circuit derived through algebraic factorization of a two-level circuit is equivalent to some set of stuck-at faults in the original two-level circuit.

Hachtel *et al.* proved this property for algebraic factoring without the use of the complement [Hachtel 92]. Bryan *et al.* showed that this property holds for a constrained version of algebraic factoring with the use of the complement (see [Bryan 90] for details).

This property provides some important advantages in computing fault detection probabilities. Cube calculus operations can be used to find the detecting set (represented as a cover) of each fault in the initial two-level circuit. These detecting sets can then be used to quickly compute the detection probability of any fault in *any* multilevel circuit derived through algebraic factorization. Thus, the key feature is that the detecting sets for the initial two-level circuit need only be computed *once*, and then they can be used to compute fault detection probabilities during any stage of the factoring process. This technique will be explained in detail.

2.1 Computing Detecting Sets in a Two-Level Circuit

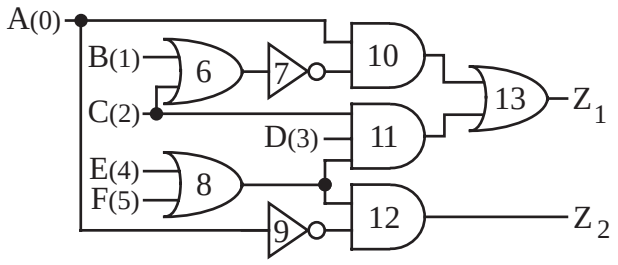
The detecting set for a fault is computed by finding the *faulty logic function*, logic function of the circuit in the presence of the fault, and comparing it with the *fault-free logic function*, logic function of the circuit without any faults. The detecting set is equal to the set of input vectors for which the faulty logic function differs from the fault-free logic function.

Given the cover $\boldsymbol{C}$ corresponding to a single-output two-level circuit (i.e., each cube in $\boldsymbol{C}$ corresponds to an AND gate in the circuit), a cover for the detecting set of each fault in the circuit can be computed using cube calculus operations. Each fault in a two-level circuit, with the exception of the faults at the primary inputs (PI's) and primary output (PO), is equivalent to an input of an AND gate being stuck-at-1 (s-a-1) or the output of an AND gate being stuck-at-0 (s-a-0) and hence causes a faulty logic function in which a cube in $\boldsymbol{C}$ either expands or is removed. The faulty logic function for a s-a-1 fault at the input of an AND gate can be found by expanding the corresponding cube in $\boldsymbol{C}$ by removing the literal that is s-a-1. The detecting set is given by the intersection of the expanded cube with the complement of the cover $\boldsymbol{C}'$ since this gives the input vectors for which the faulty logic function differs from the fault-free logic function. The faulty logic function for a s-a-0 fault at the output of an AND gate can be computed by removing the corresponding cube d in the cover $\boldsymbol{C}$. Let $(\boldsymbol{C} - d)'$ denote the complement of the cover formed by $\boldsymbol{C}$ minus the removed cube d, then the detecting set is given by the intersection of the removed cube d with $(\boldsymbol{C} - d)'$. For a s-a-1 (s-a-0) fault at primary input x, the detecting set is given by the intersection of $x'(x)$ with $\frac{d\boldsymbol{C}}{dx}$ (where $\frac{d\boldsymbol{C}}{dx}$ denotes the boolean difference of the cover $\boldsymbol{C}$ with respect to input x). For the s-a-1 (s-a-0) fault at the PO, the detecting set is simply $\boldsymbol{C}'(\boldsymbol{C})$.

For a multi-output two-level circuit where no AND gate fans out to more than one PO, the detecting set for each fault is computed by treating each PO as a single-output two-level circuit and performing the calculations described above. Then for each fault at a PI that can be detected at more than one PO, its complete detecting set is formed by taking the union of its detecting sets at each PO. The detecting sets are represented as covers, so the union is formed by simply logically ORing the covers together.

2.2 Mapping Faults in Multilevel Circuit to Faults in Two-Level Circuit

As was stated in Property 1, given a stuck-at fault in a multilevel circuit derived through algebraic factorization of a two-level circuit, there exists a set of stuck-at faults in the two-level circuit that is equivalent. One way to determine this "mapping" of faults in the multilevel circuit ("multilevel faults") to faults in the two-level circuit ("two-level faults") is to use a multilevel circuit representation called the *equivalent normal form (ENF)* which is described in [Armstrong 66] (see also [Devadas 92]). The ENF of a circuit is a two-level representation in which each literal in the sum-of-products (SOP) expressions for each PO is annotated by its path through the circuit. It is best explained by looking at an example. In Fig. 1, a multilevel circuit is shown along with its ENF representation. The gates are numbered in topological order, and the ENF is constructed by visiting each gate in ascending order and replacing the gate with a SOP expression for the gate output in terms of the SOP expressions that exist for each of its inputs. When forming the SOP expression for each gate output, De Morgan's laws and distributivity are used without making any Boolean reductions such as $(a+a' \equiv 1)$, $(a+a \equiv a)$, $(a \cdot a \equiv a)$, or $(a \cdot a' \equiv 0)$, and the gate number is appended to the annotation list for each literal. For example, in the circuit in Fig. 1, the SOP expression at the output of gate 8 is $(E_8 + F_8)$, and at the output of gate 11 it is $(C_{11}D_{11}E_{8,11} + C_{11}D_{11}F_{8,11})$. When all of the gates have been visited, a SOP expression with annotated literals will exist for each primary output; this constitutes the ENF of the multilevel circuit. In this paper, two syntactical additions are made to the ENF notation to simplify later definitions: PI's are numbered (using numbers lower than any gate number) and inserted at the beginning of the annotation list for each ENF literal, and a prime sign is placed on a gate number if a logic inversion occurs in the gate.



$$Z_1 = A_{0,10,13}\, B'_{1,6,7',10,13}\, C'_{2,6,7',10,13} + C_{2,11,13}\, D_{3,11,13}\, E_{4,8,11,13} + C_{2,11,13}\, D_{3,11,13}\, F_{5,8,11,13}$$

$$Z_2 = A'_{0,9',12}\, E_{4,8,12} + A'_{0,9',12}\, F_{5,8,12}$$

Figure 1. Example of Multilevel Circuit Represented in ENF

Each ENF literal has the form l_P where the annotated list P specifies a path through the multilevel circuit. If a stuck-at fault occurs at some node in the multilevel circuit, the logic value at that node is fixed to either 0 or 1. This changes the logic function of the circuit by fixing the logic value of each ENF literal whose path goes through that node to either 0 or 1 depending on the inversion parity of the path from the fault site to the primary output. Because it provides a simple relationship between a multilevel fault and the resulting faulty logic function, the ENF representation can be used to map a multilevel fault to an equivalent set of two-level faults. Before looking at an example, inversion parity needs to be defined.

Definition 3: The *inversion parity* of a path through a circuit is even (odd) if the number of logic inversions along the path is even (odd).

The inversion parity for the path specified by an ENF literal's annotated list P starting at PI g or gate output g, which will be denoted *IP(P,g)*, is even (odd) if the number of primed gate numbers greater than g in P is even (odd). For example, if P is the annotated list for the ENF literal $C_{1,4',6,7',9}$, then *IP(P,1)* is even, *IP(P,4)* is odd, *IP(P,6)* is odd, *IP(P,7)* is even, and *IP(P,9)* is even.

Now consider the circuit in Fig. 1. If the output of gate 8 is s-a-1, then the faulty logic function can be constructed by setting each ENF literal whose annotated list includes gate 8 to logic value 1 since the inversion parity along each path from gate 8 to a PO is even. Thus, $E_{4,8,11,13}$ and $F_{5,8,11,13}$ are effectively removed from the sum-of-products expression for Z_1, and $E_{4,8,12}$ and $F_{5,8,12}$ are removed from the sum-of-products expression for Z_2. This faulty logic function is identical to the one that occurs if there were four s-a-1 faults in the two-level circuit at the inputs to the AND gates corresponding to the four literals that were removed. If the output of gate 8 is s-a-0, then the faulty logic function is constructed by setting those same four literals to logic value 0. This is equivalent to s-a-0 faults in the two-level circuit at the inputs to the AND gates corresponding to the four literals. If a s-a-1 fault occurs at the input of gate 6 that comes from PI *C(2)*, then each ENF literal whose annotated list includes both PI 2 and gate 6 is affected; the only such literal is $C'_{2,6,7',10,13}$. Notice that in this case, the inversion parity along the path from the fault site to the PO is odd, therefore the literal $C'_{2,6,7',10,13}$ is s-a-0 which is the opposite polarity from the fault which was s-a-1.

Now, the mapping process will be formally defined. The operation $f_{mult} \rightarrow F_{two}$ maps a stuck-at fault in a multilevel circuit, f_{mult}, to an equivalent set of faults in the two-level circuit, $F_{two} = \{ f_{two,1}, \ldots, f_{two,n} \}$. Each two-level fault, $f_{two,i}$, is a s-a-1 or s-a-0 fault at the input of an AND gate in the two-level circuit described by the ENF SOP expressions. For each PO, let each cube in its ENF SOP expression be ordered and each ENF literal in each cube be ordered, then **s-a-0**[z,i,j], **s-a-1**[z,i,j], and

P[*z,i,j*], will denote the s-a-0 fault, s-a-1 fault, and annotated list, respectively, for the *j*-th ENF literal in the *i*-th cube of the ENF SOP expression for PO *z*. Using this notation, the mapping $f_{mult} \rightarrow F_{two}$ is derived as follows:

if f_{mult} is a s-a-1 (s-a-0) fault at PI *g* or at the output of gate *g*, then

F_{two} = { **s-a-1**[*z,i,j*] | *g* ∈ **P**[*z,i,j*] and *IP* (**P**[*z,i,j*], *g*) is even (odd) }
∪ { **s-a-0**[*z,i,j*] | *g* ∈ **P**[*z,i,j*] and *IP* (**P**[*z,i,j*], *g*) is odd (even) }

if f_{mult} is a s-a-1 (s-a-0) fault at the input of gate *g* that comes directly from PI *f* or gate *f*, then

F_{two} = { **s-a-1**[*z,i,j*] | *f* ∈ **P**[*z,i,j*], *g* ∈ **P**[*z,i,j*], and *IP* (**P**[*z,i,j*], *f*) is even (odd) }
∪ { **s-a-0**[*z,i,j*] | *f* ∈ **P**[*z,i,j*], *g* ∈ **P**[*z,i,j*], and *IP* (**P**[*z,i,j*], *f*) is odd (even) }

2.3 Computing Fault Detection Probabilities for Multilevel Circuit

Given a fault in a multilevel circuit that is derived through algebraic factorization of a two-level circuit, the ENF can be used, as was shown, to map the multilevel fault to an equivalent set of faults in the two-level circuit, $f_{mult} \rightarrow F_{two}$. Assuming that the detecting set for each fault in the initial two-level circuit has been computed, the next step is to compose the detecting set for the multilevel fault from the detecting sets for the two-level faults. If the multilevel fault is at a PI or PO, then the detecting set is just the same as the two-level detecting set for the same fault. For all other faults, the detecting set for the multilevel fault is composed by simply taking the union of the detecting sets for each two-level fault in F_{two}, however, there are two cases where this is not true.

Case 1: If two or more faults in F_{two} are in the same PO function and cause literals in two non-disjoint cubes to be s-a-0. This case involves two overlapping cubes in the cover ***C*** for some PO, which are both removed by the multilevel fault. The two-level detecting sets assumed only one cube would be removed at a time, so their union may understate the actual detecting set. A simple example is a primary output function with two cubes, $a\,b + b\,c$. The detecting set for a literal in cube $a\,b$ being s-a-0 is $a\,b\,c'$ and the detecting set for a literal in cube $b\,c$ being s-a-0 is $a'b\,c$. If literals in both cubes are s-a-0, then the union of the detecting sets suggests $\{a\,b\,c', a'b\,c\}$, however, the full detecting set is $\{a\,b\,c', a'b\,c, a\,b\,c\}$. Not computing the full detecting set for this case always provides a lower bound on the fault detection probability. The full detecting set can be computed by adding in the missing tests. The missing tests can be found by forming the cover ***Q*** consisting of the overlapping cubes, and then taking the intersection of ***Q*** with $(\boldsymbol{C} - \boldsymbol{Q})'$. Note that if a check is being made to see if f_{mult} is r.p.r., then if the lower bound is above the r.p.r. threshold, it is not necessary to compute the full detecting set.

Case 2: If there is reconvergent fan-out with different inversion parity, then it is possible for two or more faults in F_{two} to be in the same PO function and have opposite polarity, i.e., one or more is s-a-1 (causing cubes to expand) and one or more is s-a-0 (causing cubes to be removed). Then if an expanded cube is non-disjoint from a removed cube, part of the two-level detecting set for the removed cube may not detect the multilevel fault. This case involves a cube that expands such that it partially covers a cube that is removed thereby eliminating some of the tests for the removed cube. In this case, the union of the two-level detecting sets may overstate the detecting set for the multilevel fault. Since tests for the expanded cubes will always detect the multilevel fault, the union of the two-level detecting sets for the s-a-1 faults in F_{two} are a subset of the detecting set for the multilevel fault and hence form a lower bound. The full detecting set can be computed by adding in the missing tests. The missing tests can be found by forming the cover ***E*** consisting of the expanded cubes, and then taking the intersection of $\boldsymbol{E}'$ with the union of the detecting sets for the s-a-0 faults in F_{two}; this gives the tests for the portion of the removed cubes that is not covered by the expanded cubes. As with case 1, if a check is being made to see if f_{mult} is r.p.r., then if the lower bound is above the r.p.r. threshold, it is not necessary to compute the full detecting set.

Note that case 2 will not occur for algebraic factoring without the use of the complement or where the complement is used only if a factor and its complement fan out to different PO's. This type of factoring will avoid reconvergent fan-out with different inversion parity except for the faults at the PI's. However, the detecting sets for faults at the PI's are computed in the two-level circuit as shown in Sec. 2.1, i.e., they are not computed by composing detecting sets. Therefore, for this type of algebraic factoring, case 2 need not be considered.

So, the detecting set for the multilevel fault f_{mult} is composed by taking the union of the two-level detecting sets for each fault in F_{two}. If case 1 or 2 occurs, then some additional calculation may be required to get the full detecting set for f_{mult}. After the detecting set for the multilevel fault f_{mult} has been composed, the last step is to determine the fault detection probability. Since the detecting set is represented as a cover, it is necessary to determine how many input combinations satisfy the cover (i.e., how many minterms are elements of some cube in the cover). This can be computed exactly by using an algorithm such as the one in [Falkowski 90] to make the cover disjoint and then summing up the sizes of each cube, or it can be estimated using the Karp-Luby algorithm [Karp 83] which is a Monte-Carlo algorithm for boolean functions in disjunctive normal form that runs in polynomial time. The number of input combinations that detect the fault is then divided by the total number of input combinations, 2^n, where *n* is the number of primary inputs, to give the fault detection probability.

3. RANDOM PATTERN TESTABILITY PRESERVING TRANSFORMATIONS

The strategy in the proposed logic synthesis procedure is to first factor the initial two-level circuit so that it is random pattern testable, and then use random pattern testability preserving transformations to optimize the circuit without introducing r.p.r. faults.

Definition 4: Let T be a transformation which transforms circuit K_1 into circuit K_2. If the minimum fault detection probability in K_2 is greater than or equal to the minimum fault detection probability in K_1, for some fault class F, then the transformation T is *random pattern testability preserving* for fault class F.

Applying random pattern testability preserving transformations to a circuit that doesn't have any r.p.r. faults will never produce a circuit that has r.p.r. faults. The following theorem states that random pattern testability preserving transformations are a superset of test-set preserving transformations.

Definition 5: If a test set includes a test for each fault in a circuit for some fault class F, then it is a *complete test set* with respect to fault class F.

Definition 6: Let T be a transformation which transforms circuit K_1 into circuit K_2. If any complete test set for K_1 is also a complete test set for K_2, with respect to fault class F, then the transformation T is *test-set preserving* for fault class F.

Theorem 1: If a transformation is test-set preserving for fault class F, then it is also random pattern testability preserving for fault class F.

Proof: Consider faults in fault class F: If a test-set preserving transformation is used to transform circuit K_1 into circuit K_2, then any complete test set for K_1 is also a complete test set for K_2. The detecting set of each fault in K_2 must contain the detecting set of at least one fault in K_1. If this were not the case, then it would be possible to construct a complete test set for K_1 that did not detect some fault in K_2. Therefore, the detection probability for each fault in K_2 is greater than or equal to the detection probability of at least one fault in K_1. Thus, the minimum fault detection probability in K_2 is greater than or equal to the minimum fault detection probability in K_1.

A number of test-set preserving transformations for single stuck-at faults have been identified in [Rajski 92] and [Batek 92]. Tree-covering technology mapping procedures [Keutzer 87], [Detjens 87], are test-set preserving for both single and multiple stuck-at faults [Hachtel 92]. Based on Theorem 1, all of these transformations are random pattern testability preserving as well.

Note that adding an observation point is random pattern testability preserving, however, adding a control point is not. It is possible for a control point to reduce the detection probability for some faults. This will be explained further in Sec. 5.

4. LOGIC SYNTHESIS PROCEDURE

The proposed logic synthesis procedure is described in this section. The procedure generates a multilevel implementation under the constraint that the detection probability for each fault is above a given threshold.

Input: Two-level representation of circuit and minimum fault detection probability threshold

Output: Multilevel circuit with no r.p.r. faults

Step 1: Use a two-level minimizer to form a prime and irredundant cover for circuit

Since algebraic factoring is used, this will ensure that no redundant single or multiple faults will occur in the multilevel implementation [Bryan 90].

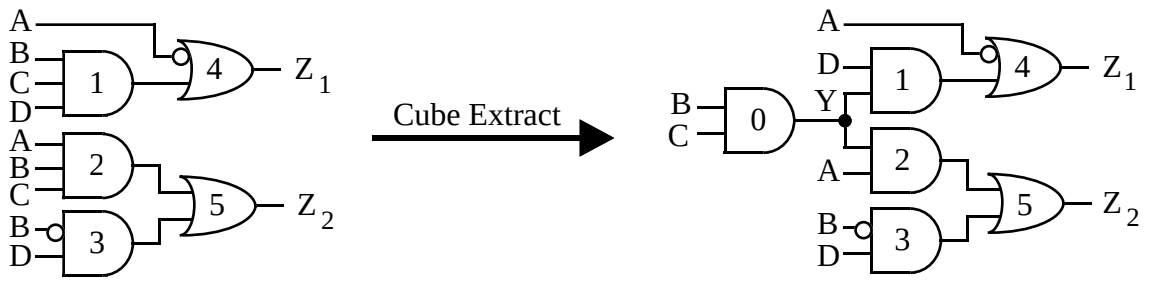
Step 2: Identify r.p.r. faults

This is done by computing the fault detection probabilities (using the method described in Sec. 2) and comparing them with the given threshold. If the detection probability for a fault is below the threshold, then the fault is marked as r.p.r.

Step 3: Identify algebraic factors that eliminate r.p.r. faults

The two types of algebraic factors are kernels and common cubes [Brayton 87]. Factoring out a common cube affects the detection probability of faults associated with each instance of the cube. Consider the example of extracting a common cube shown in Fig. 2. The detecting sets for each fault associated with the common cube are listed. In the original network, some of the faults associated with the common cube had a detection probability of $\frac{1}{16}$. For example, the s-a-1 fault on the input of gate 2 coming from primary input B can only be detected by one input combination, $AB'CD$. However, after the common cube is extracted, all of the faults associated with the common cube have a detection probability of at least $\frac{2}{16}$. The technique described in Sec. 2 can be used to quickly check what the resulting detection probabilities for faults associated with a common cube would be if the cube were factored out; by so doing, cube factors that eliminate r.p.r. faults can be identified.

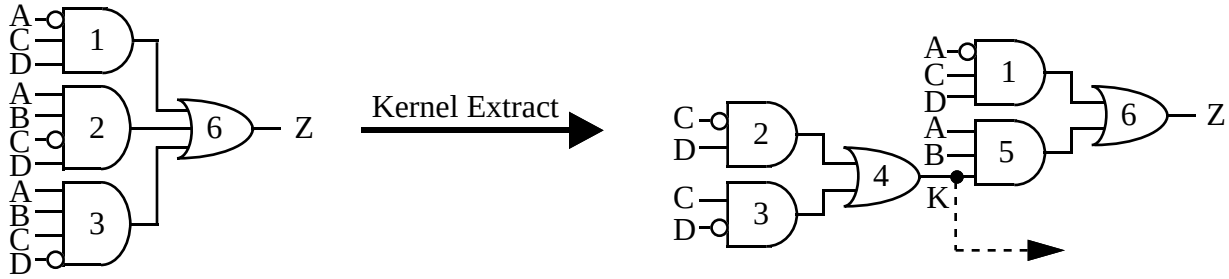
A kernel k of an expression f is the quotient of f and a cube d which is called the co-kernel; $k = f / d$ or $f = d\,k$. Factoring out a kernel affects the detection probability for faults associated with each instance of co-kernel d, and if the kernel k is common to multiple expressions, then the detection probability of faults associated with each instance of the kernel k are also affected. Consider the example in Fig. 3. In the original network, all of the faults associated with the co-kernel have a detection probability of $\frac{1}{16}$, however, after the kernel K is extracted, all the faults associated with the co-kernel have a detection probability of $\frac{2}{16}$. If the kernel K is common to other



DS [B_1, s-a-1] = A B′ C D
DS [C_1, s-a-1] = A B C′ D
DS [B_1C_1, s-a-0] = A B C D
DS [B_2, s-a-1] = A B′ C D′
DS [C_2, s-a-1] = A B C′
DS [B_2C_2, s-a-0] = A B C

DS [B_0, s-a-1] = A B′ C
DS [C_0, s-a-1] = A B C′
DS [Y, s-a-1] = A B′ + A C′
DS [Y, s-a-0] = A B C

Figure 2. Example of Effect of Extracting a Common Cube on Detecting Sets



DS [A_2, s-a-1] = A′ B C′ D
DS [B_2, s-a-1] = A B′ C′ D
DS [A_2B_2, s-a-0] = A B C′ D
DS [A_3, s-a-1] = A′ B C D′
DS [B_3, s-a-1] = A B′ C D′
DS [A_3B_3, s-a-0] = A B C D′

DS [A_5, s-a-1] = A′ B C′ D + A′ B C D′
DS [B_5, s-a-1] = A B′ C′ D + A B′ C D′
DS [A_5B_5, s-a-0] = A B C′ D + A B C D′
DS [K, s-a-1] = A B C D + A B C’ D′
DS [K, s-a-0] = A B C′ D + A B C D′

Figure 3. Example of Effect of Extracting a Kernel on Detecting Sets

expressions, then it may fan out which would increase the observability of faults associated with it thereby affecting their detection probabilities. Again, the technique in Sec. 2 can be used to quickly check how the affected detection probabilities would change if a kernel were extracted and therefore kernel factors that eliminate r.p.r. faults can be identified.

During the normal kernel and cube extraction procedures in MIS [Brayton 87], kernels and common cubes are enumerated and chosen on the basis of literal count reduction. This same enumeration process can be used to find kernels and common cubes so that each can be checked to see which, if any, r.p.r. faults would be eliminated if it were extracted.

Step 4: Extract a set of factors that eliminate all r.p.r. faults and minimize literal count as much as possible

Given the list of factors and the r.p.r. faults that each eliminates, a set of these factors is extracted such that all r.p.r. faults are eliminated, and as a secondary goal, the literal count is reduced as much as possible. Note that some factors which actually increase the literal count may in fact be chosen to satisfy the primary criteria of eliminating all r.p.r. faults. Of course, in some cases it may not be possible to eliminate all r.p.r. faults by algebraic factoring alone. In those cases, test points will need to be inserted. This is discussed in Sec. 5 where an automated procedure for inserting test points is presented as an extension to the techniques used in this step.

Step 5: Optimize with random pattern testability preserving logic transformations

As was shown in Sec. 3, random pattern testability preserving logic transformations may be performed without concern of introducing new r.p.r. faults.

5. TEST POINT INSERTION DURING SYNTHESIS

As the minimum fault detection probability threshold is increased for a circuit, a point is reached where some r.p.r. faults cannot be eliminated through algebraic factoring. When this is the case, test points are inserted in order to generate an implementation that satisfies the minimum detection probability constraint. Test inputs (for control points) and/or test outputs (for observation points) are added in such a way that they can be used during testing to increase fault detection probabilities, but during normal operation the test inputs can be set to a specific logic value that allows the circuit to operate as intended. The advantage of adding test points during synthesis is that factors can be specially chosen so that a single test point can eliminate a number of r.p.r. faults. In post-synthesis test point insertion, factoring has already been completed so it is fortuitous if a test point can be placed so as to eliminate multiple r.p.r. faults.

Test point insertion is performed during step 4 of the procedure in Sec. 4. During that step, an attempt is made to find a set of factors that eliminate all r.p.r. faults. If it is found that some r.p.r. faults cannot be eliminated with factoring alone then one or more test points must be inserted to eliminate these faults. One cause of r.p.r. faults are cubes with large fan-in which result in poor observability at their inputs and poor controllability at their outputs, so test points are needed to "break up" these cubes. By finding common factors among large fan-in cubes, a single test point can be inserted to break up several large fan-in cubes thus eliminating a number of r.p.r. faults. Examples of factors that enable this are shown in Figs. 4, 5, and 6. In Fig. 4, the general form can be seen for extracting a kernel and adding an observation point at the output of the kernel. Since a kernel breaks up multiple cubes, this type of factoring increases the effectiveness of a single observation point. Extracting a common cube *c* and adding an observation point at its output does not help, however, because the controllability at the output of *c* is not improved so the fault detection probabilities associated with the cubes for which *c* is a fan-in are not improved. In Fig. 5, the general form can be seen for extracting either a cube or a kernel factor *Y* and adding a control point at its output. The control point improves the controllability at the output of *Y* and thus improves the observability of the inputs of each cube for which *Y* is a fan-in. In Fig. 6, the general form can be seen for extracting either a cube or a kernel factor *Y* and adding both a control point and an observation point. In the example in Fig. 4, a small fan-in cube provided good controllability at the output of the kernel, and in the example in Fig. 5, a small fan-in cube provided good observability at the output of the extracted cube *Y*, however, in the example of Fig. 6, all of the cubes have large fan-in, so both a control point and an observation point are required.

Extracting a common cube and adding a control point and observation point is an effective technique for eliminating all of the r.p.r. faults with few test points. The reason for this is that it is often possible to find a common cube among many large fan-in cubes, thus enabling a single control and observation point to break up several large fan-in cubes. In some circuits, only a few common cubes need to be factored out and augmented with test points to break up all the large fan-in cubes and eliminate all r.p.r. faults. In other circuits, good kernels exist such that only observation points are needed to

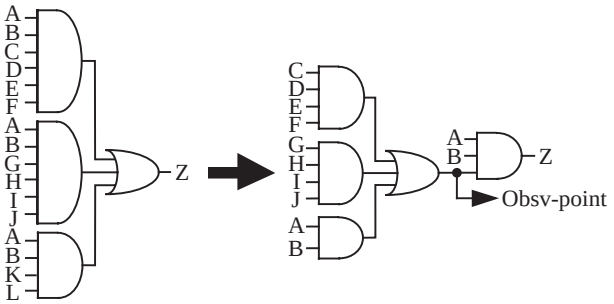


Figure 4. Example of Kernel Extraction with Observation Point

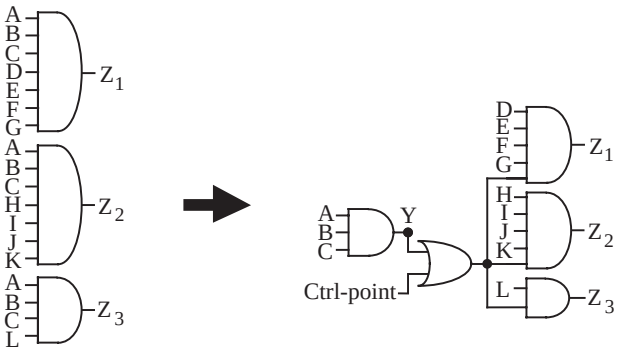


Figure 5. Example of Cube Extraction with Control Point (same form applies if Y is a kernel)

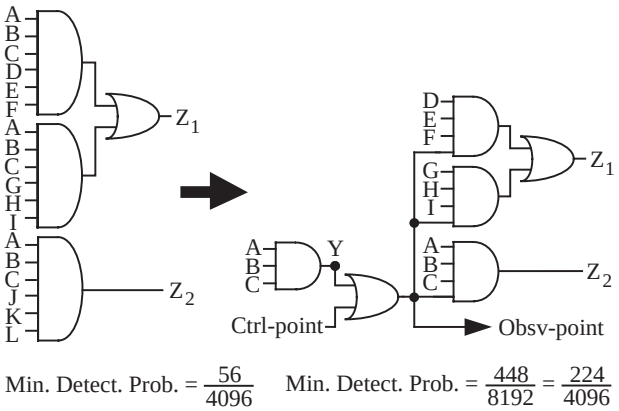


Figure 6. Example of Cube Extraction with Control and Observation Point (same form applies if Y is a kernel)

eliminate all r.p.r. faults. So, it is advantageous to consider all of the options when inserting test points during synthesis.

In the proposed logic synthesis procedure, test point insertion is performed by first identifying factors that allow a single test point to eliminate several r.p.r. faults that require test points (i.e., that cannot be eliminated by factoring alone). These factors can be found by again enumerating the kernels and common cubes and computing the relevant fault detection probabilities to determine which r.p.r. faults each factor plus a control point or observation point or both will eliminate. Once all of these factors have been identified, a set of them are chosen and augmented by the appropriate test points such that all r.p.r. faults requiring test points are eliminated using as few test points as possible.

Computing the fault detection probabilities when identifying which r.p.r. faults each factor plus test point eliminates is complicated by the fact that test points change the two-level detecting sets. An observation point adds a new PO, so its ENF and two-level detection sets must be computed in order to use the technique described in Sec. 2. Control points pose a more difficult challenge because the ENF and two-level detecting sets change for each PO that the control point has a path to. This presents two problems: (1) adding a control point can lower the detection probability for any fault that has a path to some primary output that the control point has a path to, and (2) recomputing the two-level detecting sets each time a control point is considered can be computationally expensive. The first problem need not be a major concern during the factor selection process. In most cases, adding control points will not significantly lower any fault detection probabilities. At one logic value, the control point has no effect on the circuit so the detection probabilities remain the same. At the other logic value, the control point can raise or lower the detection probabilities for some faults. Thus adding a control point can reduce the detection probability for a fault by no more than a factor of 2, but it can increase the detection probability many times over. *After* a control point is added, fault detection probabilities can be verified to make sure that none of them have slipped below the minimum detection probability threshold. In the rare event that this has occurred, the procedure can backtrack and find an alternative. Regarding the second problem of recomputing two-level detecting sets, only the two-level detecting sets that are needed to check if any r.p.r. faults are eliminated by the control point need to be recomputed during the factor selection process.

Another important issue is minimizing the number of test inputs and test outputs that are needed to support the test points. Each test input and test output has some overhead associated with it. Each test input requires larger input patterns to be generated, and each test output requires more output response analysis. Observation points can be "condensed" using techniques such as those in [Fox 77], to reduce the number of test outputs. If at-speed testing is to be used, care must be taken in designing the condensation network so that the delay isn't longer than a clock period. When the logic synthesis procedure adds an observation point, a check can be made to see if it can be condensed without significantly reducing any fault detection probabilities. Multiple control points can be derived from the same test input. When the logic synthesis procedure adds a new control point, a check can be made to see if it can be derived from one of the previously added primary inputs without significantly reducing any fault detection probabilities.

6. RESULTS

The logic synthesis procedure described in this paper has been incorporated into SIS 1.1 (an updated version of MIS [Brayton 87]) and used to generate multilevel implementations for several benchmark circuits that have long random pattern test lengths. The results are shown in Table 1. Under the first major heading, information is given about each benchmark circuit: name, number of primary inputs, and number of primary outputs. Under the next three major headings, results are given for the multilevel circuits generated using two scripts that are distributed with SIS, **script.algebraic**, which uses only algebraic transformations, and **script.rugged**, which uses both boolean and algebraic transformations, and the multilevel circuits generated using the proposed logic synthesis procedure with different minimum fault detection probability constraints. Three things are shown under each of these major headings:

P_{det} (log_2) - Lowest fault detection probability for any fault in the circuit. It is computed exactly and expressed as a log base 2.

Test Length - Test length which was obtained by averaging the number of random patterns needed to reach 100% fault coverage for 50 simulation experiments using LFSR's with 5 different characteristic polynomials and 10 different seeds. The number of stages in the LFSR was equal to the number of PI's.

lits - Factored form literal count for the circuit.

For the multilevel circuits generated using the proposed procedure, the number of test inputs and test outputs that were added to the circuit (necessitated by test points) are listed under the columns labeled *Test PI* and *Test PO*. Control points were derived from the same test input when possible, and observation points were condensed when possible. The area for the condensation network is included in the literal count. Under the last two major headings, the multilevel circuit implementations generated by the proposed procedure are compared with those generated by the algebraic and rugged scripts. The first

Table 1. Results for Benchmark Circuits

Circuit			Algebraic Script			Rugged Script			Proposed Procedure					Comp Alg		Comp Rug	
Name	PI	PO	P_{det} ($\log_2$)	Test Length	Lits	P_{det} ($\log_2$)	Test Length	Lits	P_{det} ($\log_2$)	Test Length	Lits	Test PI	Test PO	TLen Red	Area Ovrhd	TLen Red	Area Ovrhd
chkn	29	7	-22	19M	422	-23	33M	332	-19	1.1M	426	0	1	17	0.9%	30	28%
									-17	900K	442	0	1	21	4.7%	36	33%
									-15	120K	467	0	1	158	11%	275	41%
duke2	22	29	-15	48K	425	-15	76K	420	-14.7	35K	426	0	0	1.3	0.2%	2.1	1.4%
									-12	7K	426	0	1	6.8	0.2%	10	1.4%
									-11	3K	431	1	1	16	1.4%	25	2.6%
exep	30	63	-22	4.6M	566	-20	2.1M	545	-16	260K	597	1	1	17	5.5%	8.1	9.5%
									-14	67K	635	1	1	69	12%	31	17%
									-11	16K	663	2	1	287	17%	131	22%
gary	15	11	-13	23K	501	-13	20K	374	-12	11K	559	0	1	2.1	12%	1.8	49%
									-11	4.2K	578	0	1	5.4	15%	4.7	55%
									-9	1.8K	605	1	1	12	21%	11	62%
in2	19	10	-13	20K	419	-13	14K	301	-12.4	12K	429	0	0	1.6	2.4%	1.2	43%
									-11	3K	492	0	1	6.6	17%	4.6	63%
in7	26	10	-17	100K	126	-17	110K	116	-13	15K	137	1	1	6.6	8.7%	7.3	18%
									-10	8K	167	1	1	12	33%	14	44%
misg	56	23	-13	12K	101	-13	23K	95	-12	9K	101	0	0	1.3	0%	2.5	6.3%
									-11	5K	112	0	1	2.4	11%	4.6	18%
									-8	0.6K	122	0	1	20	21%	38	28%
vg2	25	8	-15.2	145K	97	-15.2	148K	88	-14.5	40K	97	0	0	3.6	0%	3.7	10%
									-11	6K	121	1	1	24	25%	24	38%
x1dn	27	6	-17.5	210K	101	-17.5	420K	90	-15	130K	113	1	1	1.6	12%	3.2	26%
									-13	69K	125	1	1	3.0	24%	6.1	39%
									-11	2.3K	149	2	1	91	48%	182	61%
x2dn	82	56	-15.5	165K	206	-15.5	140K	194	-10	3.8K	214	1	1	43	3.9%	36	10%
									-8	1.6K	234	2	1	103	14%	87	21%
x6dn	39	5	-15	44K	381	-14	29K	333	-13	9K	398	0	1	4.8	4.4%	3.2	20%
									-11	4.3K	423	0	1	10	11%	6.7	27%

column gives the test length reduction factor, and the second gives the percentage of area overhead. They are computed as follows:

$$\text{Test Length Reduction Factor} = \frac{\text{(script test length)}}{\text{(procedure test length)}}$$

$$\text{\% Area Overhead} = \frac{\text{(procedure lits) - (script lits)}}{\text{(script lits)}} \times 100$$

The minimum fault detection probability constraints were chosen for the proposed procedure to show a range of area versus test length reduction tradeoffs. For almost all the circuits, implementations were found which reduced the test length by at least a factor 10 with only one test output and in some cases one test input. In some circuits, very little area overhead was required to achieve a significant reduction in random pattern test length. In comparing the implementations generated by the algebraic script versus the rugged script, the rugged script produced smaller implementations, however, in some cases the required test length is longer. This is due to the fact that the rugged script uses boolean transformations which may generate circuit structures that have faults with lower detection probabilities than the original starting point.

The minimum fault detection probability constraint that could be satisfied without inserting test points (i.e., no test inputs or test outputs) was found for each circuit. For most circuits, no appreciable improvement was obtained compared with the algebraic script, hence results for this case are shown only for *duke2*, *in2*, *misg*, and *vg2*.

By carefully choosing kernels to increase controllability, the proposed procedure was able to use only observation points to significantly reduce the random pattern test length for many of the circuits. The observation points were condensed so that only one test output was needed. When control points were required, it was often possible to share test inputs. In almost all cases, only one test input was needed.

7. SUMMARY AND CONCLUSIONS

An automated logic synthesis procedure for designing random pattern testable circuits was presented in this paper. The procedure relies on properties of algebraic factoring to simplify fault detection probability

calculations. By so doing, algebraic factors that eliminate r.p.r. faults can be quickly identified. If it is not possible to eliminate some r.p.r. faults via algebraic factoring alone, then test points are inserted during synthesis. This enables factors to be chosen so as to minimize the number of test points that are required. Results show significant decreases in random pattern test length with few test inputs and outputs and small area overhead.

This method for synthesizing random pattern testable circuits requires a two-level representation as a starting point thereby limiting its application to control circuits and other circuits that can be flattened (i.e., two-level representation is not exponential). Control circuits are an important application because they can contain large fan-in cubes that cause r.p.r. faults. Note that this method can be used for non-flattenable circuits by partitioning the circuit into flattenable logic blocks which are logically isolated during testing.

Another limitation of this method is that it uses only algebraic transformations. However, in a large system design, this method need only be used to generate the logic blocks for which other synthesis methods do not produce random pattern testable implementations. Thus, the area overhead associated with algebraic transformations is incurred for only a portion of the overall design.

An area for future research is to extend the techniques described in this paper to allow for a multilevel starting point and the use of boolean factoring.

ACKNOWLEDGEMENTS

The authors would like to thank Dr. LaNae Avra and Siyad Ma for their helpful comments and suggestions. This work was supported in part by the Innovative Science and Technology Office of the Strategic Defense Initiative Organization and administered through the Office of Naval Research under Contract No. N00014-92-J-1782, and by the National Science Foundation under Grant No. MIP-9107760.

REFERENCES

[Armstrong 66] Armstrong, D.B., "On Finding a Nearly Minimal Set of Fault Detection Tests for Combinational Logic Networks," *IEEE Trans. Elect. Computers*, Vol. EC-15, pp. 66-73, Feb. 1966.

[Batek 92] Batek, M.J., and J.P. Hayes, "Test-Set Preserving Logic Transformations," *Proc. of the 29th Design Automation Conference*, pp. 454-457, 1992.

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

[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.

[Devadas 92] Devadas, S., and K. Keutzer, "Synthesis of robust delay-fault-testable circuits: Practice," *IEEE Transactions on Computer-Aided Design*, vol. 11, pp. 277-300, Mar. 1992.

[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.

[Falkowski 90] Falkowski, B.J., I. Schafer, and M.A. Perkowski, "A Fast Computer Algorithm for the Generation of Disjoint Cubes for Completely and Incompletely Specified Boolean Functions," Proc. of 33rd Midwest Symposium on Circuits and Systems, pp. 1119-1122, 1990.

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

[Hachtel 92] Hachtel, G.D., R.M. Jacoby, K. Keutzer, and C.R. Morrison, "On Properties of Algebraic Transformations and the Synthesis of Multifault-Irredundant Circuits," *IEEE Trans. on Computer-Aided Design*, Vol. 11, No. 3, pp. 313-320, Mar. 1992.

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

[Karp 83] Karp, R.M., and M. Luby, "Monte-Carlo Algorithms for Enumeration and Reliability Problems," *Proc. of Annual Symposium on Foundations of Computer Science*, pp. 56-64, 1983.

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

[Krishnamurthy 86] Krishnamurthy, B., and I. Tollis, "Improved Techniques for Estimating Signal Probabilities," *Proc. International Test Conference*, pp. 244-251, 1986.

[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.

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

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

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

[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.