RP-SYN: Synthesis of Random Pattern Testable Circuits with Test Point Insertion

Nur A. Touba, *Member, IEEE*, and Edward J. McCluskey, *Life Fellow, IEEE*

***Abstract*—An automated logic synthesis procedure, called RP-SYN, is described for synthesizing random pattern testable circuits. RP-SYN takes as an input a two-level description of a circuit and a constraint on the minimum fault detection probability (threshold below which faults are considered random-pattern-resistant), and generates a multilevel implementation which satisfies the constraint while minimizing the literal count. RP-SYN identifies random-pattern-resistant faults and eliminates them through testability-driven factoring combined with test point insertion. By moving the task of test point insertion from the back-end into the synthesis process, RP-SYN reduces design time and enables better optimization of the resulting implementation. Results are shown for benchmark circuits which indicate that RP-SYN can generally reduce the random pattern test length by at least an order of magnitude with only a small area overhead.**

***Index Terms*—Built-in self-test (BIST), computer-aided design, design for testability, fault coverage, integrated circuit testing, logic optimization, logic synthesis, logic transformations, pseudo-random testing, random pattern testability, test points.**

I. Introduction

CONSIDERING testability requirements during synthesis (as opposed to the traditional approach of making back-end modifications after an implementation has already been generated), can reduce design time, design mistakes, and test overhead. This paper describes an automated logic synthesis procedure, called RP-SYN, that considers random pattern testability requirements during the synthesis process and generates optimized random pattern testable implementations. RP-SYN moves the task of test point insertion from the back-end into the synthesis process to enable better optimization of the resulting implementation. RP-SYN is implemented in TOPS, Stanford CRC's synthesis-for-test tool.

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

The random pattern test length needed to achieve a particular fault coverage for a circuit depends on the detection probability of the faults 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 [14]. 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, there are two possible solutions. One is to modify the test pattern generator so that it generates patterns that detect the r.p.r. faults, and the other is to modify the circuit structure 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). The test pattern generator can be modified by adding logic to weight the patterns [27], [30], [37]; correlate the patterns [26]; map the patterns [7], [33], [34], [36]; or reseed the generator [18], [19], [38]. For on-chip generation, these approaches generally require significantly more overhead than modifying the circuit structure itself. This paper focuses on techniques for modifying the circuit structure to make it random pattern testable.

Two general techniques have been proposed for modifying the circuit structure to eliminate r.p.r. faults. The first is "post-synthesis" test point insertion. The circuit is synthesized and then test points are inserted afterwards to eliminate the r.p.r. faults. Since test points add area and performance overhead, it is important to carefully select the location of each test point in order to use as few test points as possible. Test point placement has been an active area of research [6], [9], [20], [25], [29], [31], [35]. The second technique that has been proposed is to consider random pattern testability during logic synthesis [8], [10], [32]. Logic transformations are performed to improve the random pattern testability of the resulting circuit.

Chiang and Gupta [10] presented a logic synthesis method that starts from a two-level circuit and performs algebraic factoring to generate a multilevel implementation. Instead of selecting the algebraic factors based on literal count reduction (as in [3]), they use a special cost function that estimates the impact of each factor on the detection probabilities of the faults. This cost function guides the factoring process such that the resulting implementation is more random pattern testable than literal count-based factoring.

Manuscript received August 25, 1998. This work was supported in part by the Ballistic Missile Defense Organization, Innovative Science and Technology (BMDO/IST) Directorate and administered through the Department of the Navy, Office of Naval Research under Grant N00014-92-J-1782, by the National Science Foundation (NSF) under Grant MIP-9107760, and by the Advanced Research Projects Agency (ARPA) under Prime Contract DABT63-94-C-0045. This paper was recommended by Associate Editor T. Cheng

N. A. Touba was with the Center for Reliable Computing at Stanford University. He is now with the Computer Engineering Research Center, Department of Electrical and Computer Engineering, University of Texas, Austin, TX 78712-1084 USA.

E. J. McCluskey is with the Center for Reliable Computing, Departments of Electrical Engineering and Computer Science, Stanford University, Stanford, CA 94305 USA.

Publisher Item Identifier S 0278-0070(99)05688-2.

Chatterjee *et al.*, [8] presented multilevel logic transformations which improve random pattern testability by introducing XOR gates. They use statistical estimation of the fault detection probabilities as used in STAFAN [21] to form a cost function that estimates the impact of each candidate transformation on the sum of the detection probabilities of the r.p.r. faults. The cost function guides the synthesis procedure to perform transformations that improve the random pattern testability.

The synthesis methods in [8] and [10] use only testability-driven factoring without test point insertion. As a result, they are not always able to sufficiently reduce the random pattern test length. This is borne out both in practice and in theory. There are many cases where r.p.r. faults cannot be eliminating by factoring alone. In fact, it can be proven that many logic functions do not have random pattern testable implementations. A simple example is a decoder where for any implementation the probability of detecting a stuck-at-zero (s-a-0) fault on an output is 2^{-n}, where n is the number of inputs. In the case where a logic function does not have a random pattern testable implementation, it is necessary to add test points. The synthesis procedure described in this paper, RP-SYN, combines both random pattern testability-driven factoring and test point insertion to solve this problem (preliminary results were presented in [32]). Instead of synthesizing the circuit and then adding test points afterwards to sufficiently reduce the random pattern test length, RP-SYN inserts test points during the synthesis process in a way that enhances testability-driven factoring and minimizes overhead.

RP-SYN takes as an input a two-level representation of a circuit and a constraint on the minimum fault detection probability and generates a multilevel implementation that satisfies the constraint while minimizing the literal count and the number of test points. 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. If it is not possible to eliminate all of the r.p.r. faults by factoring alone, then test points are inserted during the synthesis process in a way that minimizes the number of test points that are required. Once the r.p.r. faults have been eliminated, normal logic optimization using random pattern testability preserving logic transformations (defined in Section IV) 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 or power).

Whereas the procedures in [8] and [10] estimate fault detection probabilities when making decisions during the synthesis process, RP-SYN uses an efficient technique (described in Section III) that exploits properties of algebraic factoring to compute the exact fault detection probabilities. An important advantage of this approach is that the amount of testability-driven factoring (as opposed to area-driven or delay-driven factoring) is reduced to only that which is required to eliminate the r.p.r. faults. Once the r.p.r. faults have been eliminated, subsequent factoring can focus exclusively on other synthesis criteria such as area or delay. Moreover, when test points are required, they are inserted during the synthesis process and, thus, accounted for during the testability driven-factoring. This is a very important feature for reducing the amount of testability-driven factoring. If "post-synthesis" test point insertion is used, as in the other techniques, then a lot of unnecessary testability-driven factoring may be performed during synthesis in an attempt to improve the random pattern testability when afterwards a test point ends up being used anyway thereby obviating the need for the inefficient testability-driven factoring.

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, RP-SYN can be used to synthesize circuits which require much shorter random pattern test lengths without substantial overhead.

This work is organized as follows: Section II explains some basic definitions and terminology used in this paper. In Section III, a technique that utilizes properties of algebraic factorization to efficiently compute fault detection probabilities is described. In Section IV, random pattern testability preserving transformations are defined and their relationship to testability preserving and test-set preserving transformations is shown. In Section V, the RP-SYN procedure is outlined step by step. In Section VI, the task of inserting test points during the synthesis process in described in detail. In Section VII, results for benchmark circuits are shown and discussed. Section VIII is a summary and conclusion.

II. Basic Definitions and Terminology

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. An *algebraic expression* is a cover in which no one cube contains another (e.g., $a+ab$ is not an algebraic expression). FG is an *algebraic product*, if F and G are algebraic expressions with no input variables in common; otherwise, FG is a Boolean product. For example, $(w+x)(y+z) = wy+wz+xy+xz$ is an algebraic product, but $(x+y)(x+z) = x+xy+xz+yz$ and $(x+y)(y'+z) = xy'+xz+yz$ are Boolean products [4]. If $F = QD + R$, where F, Q, D, and R are algebraic expressions and QD is an algebraic product, then Q and D are *algebraic factors* of F [4]. Boolean factors are much more computationally expensive to identify than algebraic factors and, thus, algebraic factorization is commonly used in multilevel synthesis.

III. Computing Fault Detection Probabilities

The operations in RP-SYN involve identifying r.p.r. faults and finding factors that eliminate these faults. This requires computing fault detection probabilities which is an NP-hard problem [24]. 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, RP-SYN 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 [17].

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 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.

A. 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 to 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-one (s-a-1) or the output of an AND gate being 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 $d\boldsymbol{C}/dx$ (where $d\boldsymbol{C}/dx$ 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 multioutput 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.

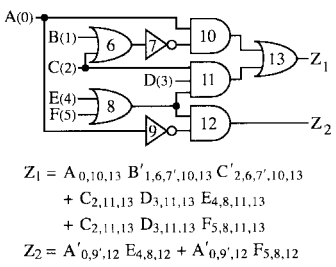

Fig. 1. Example of multilevel circuit represented in ENF.

B. 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 [1] (see also [13]). 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.

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 zero or one. 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 zero or one 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 one 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 zero . 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 $f_{\text{mult}} \rightarrow F_{\text{two}}$ maps a stuck-at fault in a multilevel circuit, f_{mult}, to an equivalent set of faults in the two-level circuit, $F_{\text{two}} = \{f_{\text{two},1}, \cdots, f_{\text{two},n}\}$. Each two-level fault, $f_{\text{two},i}$, 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 $\boldsymbol{s\text{-}a\text{-}0}[z,i,j]$, $\boldsymbol{s\text{-}a\text{-}1}[z,i,j]$, and $\boldsymbol{P}[z,i,j]$, will denote the s-a-0 fault, s-a-1 fault, and annotated list, respectively, for the jth ENF literal in the ith cube of the ENF SOP expression for PO z. Using this notation, the mapping $f_{\text{mult}} \rightarrow F_{\text{two}}$ 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

$$\begin{aligned} F_{\text{two}} = \{&\boldsymbol{s\text{-}a\text{-}1}[z,i,j] | (g \in \text{A[z,i,j]}) \\ &\text{and } \text{IP(A[z,i,j],g) is even (odd)}\} \\ \cup\ \{&\boldsymbol{s\text{-}a\text{-}0}[z,i,j] | (g \in \text{A}[z,i,j]) \\ &\text{and } \text{IP(A[z,i,j],g) is odd (even)}\}. \end{aligned}$$

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

$$\begin{aligned} F_{\text{two}} = \{&\boldsymbol{s\text{-}a\text{-}1}[z,i,j] | (f \in \text{A}[z,i,j]), (g \in \text{A}[z,i,j]), \\ &\text{and } \text{IP(A[z,i,j],f) is even (odd)}\} \\ \cup\ \{&\boldsymbol{s\text{-}a\text{-}0}[z,i,j] | (f \in \text{A}[z,i,j]), (g \in \text{A}[z,i,j]), \\ &\text{and } \text{IP(A[z,i,j],f) is odd (even)}\}. \end{aligned}$$

C. 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, $f_{\text{mult}} \rightarrow F_{\text{two}}$. 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 nondisjoint cubes to be s-a-0. This case involves two overlapping cubes in the cover $\boldsymbol{C}$ 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, $ab+bc$. The detecting set for a literal in cube ab being s-a-0 is abc' and the detecting set for a literal in cube bc being s-a-0 is $a'bc$. If literals in both cubes are s-a-0, then the union of the detecting sets suggests $\{abc', a'bc\}$, however, the full detecting set is $\{abc', a'bc, abc\}$. 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 $\boldsymbol{Q}$ consisting of the overlapping cubes, and then taking the intersection of $\boldsymbol{Q}$ 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 nondisjoint 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 $\boldsymbol{E}$ 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 Section II-A, 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 [15] to make the cover disjoint and then summing up the sizes of each cube, or it can be estimated using the Karp–Luby algorithm [22] 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.

IV. Random Pattern Testability Preserving Transformations

Before describing the RP-SYN procedure, random pattern testability preserving transformations need to be defined. Logic transformations can be classified based on their effect on the testability properties of the resulting circuit. This section defines three classes of transformations (testability preserving, random pattern testability preserving, and test-set preserving) and shows that they are related in the following way:

testability preserving
$\supset$ random pattern testability preserving
$\supset$ test-set preserving.

Definition 4: Let T be a transformation which transforms circuit K_1 into circuit K_2. If K_2 is testable for fault class F provided K_1 is testable for fault class F, then the transformation T is testability preserving for fault class F.

In the case of single stuck-at faults, any transformation that does not introduce redundancy into the circuit is testability preserving.

Definition 5: 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.

While testability preserving transformations ensure that no redundant faults are introduced into the circuit, random pattern testability preserving transformations ensure that no r.p.r. faults are introduced into the circuit. Applying random pattern testability preserving transformations to a circuit that does not have any r.p.r. faults will never produce a circuit that has r.p.r. faults. The strategy in RP-SYN 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. Note that a redundant fault has a detection probability of zero and, therefore, can be thought of as a special type of r.p.r. fault. It is easy to show that random pattern testability preserving transformations are a subset of testability preserving transformations.

Now consider test-set preserving transformations which are defined as follows.

Definition 6: 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 7: 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.

The following theorem states that random pattern testability preserving transformations are a superset of test-set preserving transformations.

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

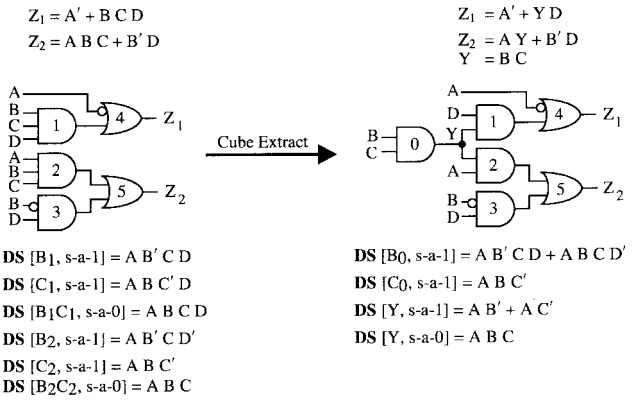

Fig. 2. Example of extracting a common cube.

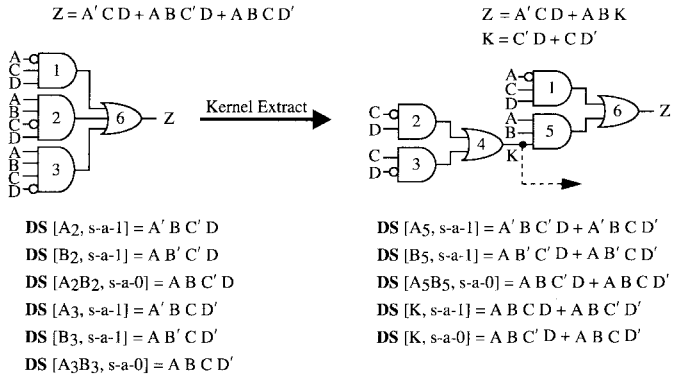

Fig. 3. Example of extracting a kernel.

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 [2] and [28]. Tree-covering technology mapping procedures [12], [23], are test-set preserving for both single and multiple stuck-at faults [17]. 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 Section VI.

V. Logic Synthesis Procedure

This section describes the RP-SYN procedure step by step. 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 [5].

Step 2: Identify r.p.r. faults.

This is done by computing the fault detection probabilities (using the method described in Section III) and comparing them to 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 [3]. 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 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 2/16. The technique described in Section III 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 algebraic expression F is the quotient of F and a cube D which is called the co-kernel; $K = F/D$ or $F = DK$. 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 1/16, however, after the kernel K is extracted, all the faults associated with the co-kernel have a detection probability of 2/16. If the kernel K is common to other 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 Section III 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 [3], 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 reduce 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 selected such that all r.p.r. faults are eliminated, and as a secondary goal, the literal count is reduced as much as possible. This is essentially a weighted covering problem where the constraint is the r.p.r. faults and the cost is the literal count. One of the many heuristic procedures for solving covering problem can be used (e.g., [11]). Note that some factors that actually increase the

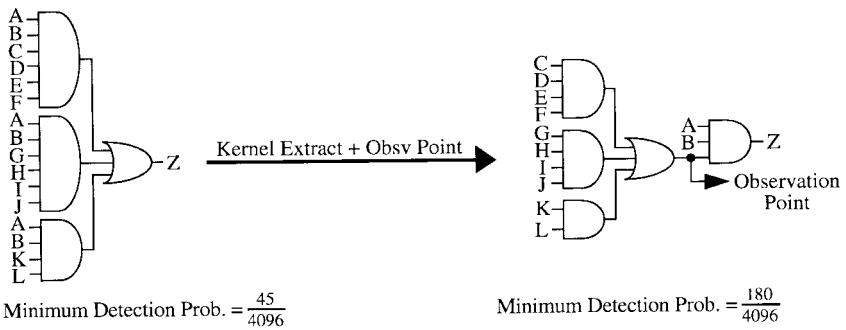

Fig. 4. Example of kernel extraction with an observation point.

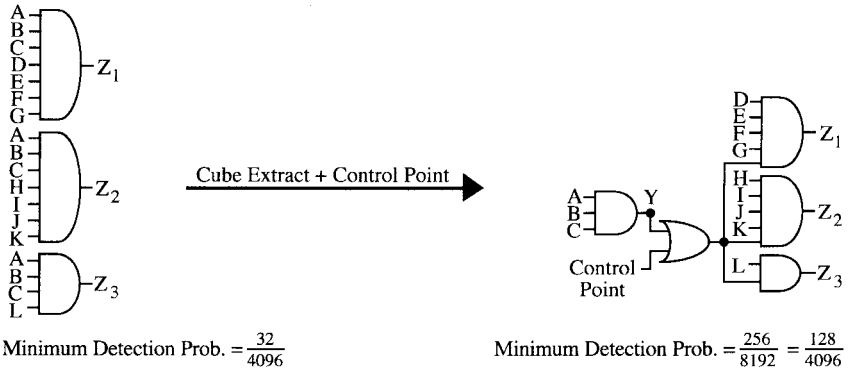

Fig. 5. Example of cube extraction with a control point (same form applies if Y is a kernel).

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 need to be inserted. This is discussed in Section VI where an automated procedure for inserting test points is presented as an extension to the techniques used in this step. When test points are required, the factoring described in Section VI is done first before the factoring described in this step.

Step 5: Optimize with random pattern testability preserving logic transformations.

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

VI. Test Point Insertion During Synthesis

When the minimum fault detection probability threshold is such that some r.p.r. faults cannot be eliminated through algebraic factoring, then test points need to be 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 RP-SYN procedure in Section V. 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. 4–6. 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

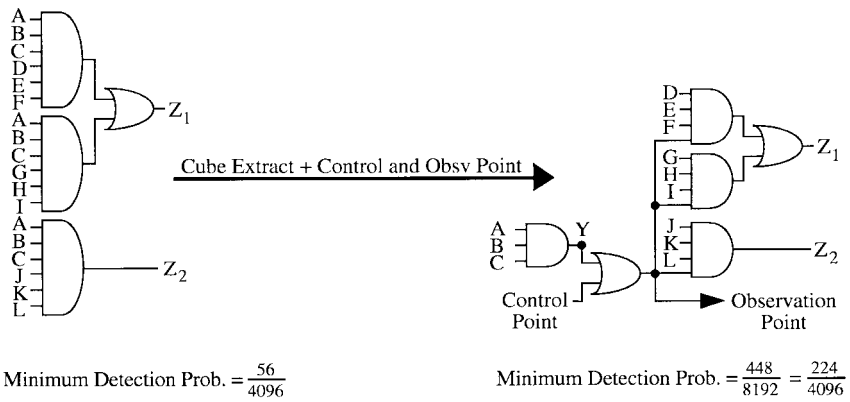

Fig. 6. Example of cube extraction with control and observation point (same form applies if Y is a kernel).

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

In RP-SYN, 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 Section III. 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 two, 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 [16], 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 is not 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.

VII. Experimental Results

RP-SYN has been built on top of SIS 1.1 (an updated version of MIS [3]) and used to generate multilevel implementations for several benchmark circuits that have long random pattern test lengths. Results are shown in Tables I and II.

TABLE I
COMPARISON OF TEST LENGTH RESULTS FOR BENCHMARK CIRCUITS

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

In Table I, 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 by RP-SYN with different minimum fault detection probability constraints. Three things are shown under each of the following 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 two.

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 five 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 by RP-SYN, 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 random pattern test length of the multilevel circuit implementations generated by RP-SYN are compared to those generated by the algebraic and rugged scripts. The test length reduction factor is shown and is computed as follows:

$$\text{Test Length Reduction Factor} = (\text{script test length})/(\text{RP-SYN test length}).$$

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 of ten with only one test output and in some cases one test input. In comparing the implementations generated by the algebraic script versus the rugged script, the rugged script produced smaller implementations, however, in some cases the 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 with using only testability-driven factoring, i.e., without inserting test points (no test inputs or test outputs) was found for each circuit. For most circuits, no appreciable improvement was obtained compared with the algebraic script (because there were some r.p.r. faults could not be eliminated by factoring alone), hence, results for this case are shown only for *duke2*, *in2*, *misg*, and *vg2*.

TABLE II
COMPARISON OF AREA AND DELAY RESULTS FOR BENCHMARK CIRCUITS

Circuit Name	Rugged Script			Delay Script			RP-SYN				Cmp Rug		Cmp Delay	
	CPU Time	Area	Delay	CPU Time	Area	Delay	Pdet (log_2)	CPU Time	Area	Delay	Area Ratio	Delay Ratio	Area Ratio	Delay Ratio
chkn	333	388.8	17.02	36	481.1	14.56	-19	473	445.0	13.24	1.14	.78	.92	.91
							-17	536	471.2	14.78	1.21	.87	.98	1.02
							-15	668	497.2	15.17	1.28	.89	1.03	1.04
duke2	19	471.4	17.07	55	662.1	9.94	-14.7	13	437.6	10.61	.93	.62	.66	1.07
							-12	116	455.6	11.57	.97	.68	.69	1.16
							-11	286	473.7	12.51	1.00	.73	.72	1.26
exep	29	589.7	10.84	85	761.4	10.12	-16	96	632.0	10.64	1.07	.98	.83	1.05
							-14	110	674.7	10.65	1.14	.98	.89	1.05
							-11	325	703.3	11.22	1.19	1.04	.92	1.11
gary	61	474.2	23.37	52	574.9	14.12	-12	25	564.2	10.10	1.19	.43	.98	.72
							-11	60	581.5	11.42	1.23	.49	1.01	.81
							-9	178	609.5	15.82	1.29	.68	1.06	1.12
in2	77	357.2	21.99	42	607.8	11.00	-12.4	7	420.4	10.55	1.18	.48	.69	.96
							-11	139	548.9	11.16	1.54	.51	.90	1.01
in7	2	131.8	12.11	19	187.0	8.81	-13	8	149.4	10.94	1.13	.90	.80	1.24
							-10	62	165.3	11.59	1.25	.96	.88	1.32
misg	1	94.2	5.44	5	114.1	6.38	-12	2	87.7	5.75	.93	1.06	.77	.90
							-11	2	91.4	6.15	.97	1.13	.80	.96
							-8	7	96.3	6.83	1.02	1.26	.84	1.07
vg2	1	112.8	7.65	10	196.3	7.67	-14.5	3	109.0	7.91	.97	1.03	.56	1.03
							-11	19	122.5	9.49	1.09	1.24	.62	1.24
x1dn	1	116.9	7.91	8	163.8	7.75	-15	265	124.4	11.59	1.06	1.47	.76	1.50
							-13	323	135.9	11.63	1.16	1.47	.83	1.50
							-11	382	156.3	13.12	1.34	1.66	.95	1.69
x2dn	2	210.7	6.42	9	220.8	6.60	-10	5	229.2	9.34	1.09	1.45	1.04	1.42
							-8	57	245.2	9.93	1.16	1.55	1.11	1.50
x6dn	22	377.7	16.12	81	482.6	10.03	-13	143	459.4	10.71	1.22	.66	.95	1.07
							-11	211	472.1	13.53	1.25	.84	.98	1.35

In Table II, results are shown comparing the area and delay of the multilevel circuits synthesized by RP-SYN versus those synthesized using *script.rugged* (area optimized) and *script.delay* (delay optimized). The CPU time required for each synthesis procedure running on an UltraSPARC 2 is shown. The circuits were mapped using *lib2.genlib*, and the resulting area and delay are shown. Under the last two major headings, the ratio of the area and delay of the circuits generated by RP-SYN with those generated by the rugged and delay scripts are shown.

As can be seen in Table II, the execution time for RP-SYN is roughly comparable with the other synthesis procedures. It is surprising to see that in a few cases, the area of the random pattern testable circuits generated by RP-SYN is actually less than those generated by the rugged script. It is also interesting to see that in many cases the delay of the random pattern testable circuits generated by RP-SYN is less than those generated by the rugged script. This implies that while the testability-driven factoring increases the literal count, it tends to reduce delay. However, the main purpose of the rugged script is to minimize area, so perhaps it is not fair to compare the delay. Thus, a comparison is also made with the delay script. It is surprising to see that in a few cases, the circuits generated by RP-SYN have even less delay then those generated by the delay script. In most cases the delay is larger, but note that the area is generally much less.

In Table III, a comparison is made between RP-SYN and post-synthesis test point insertion. The results for post-synthesis test point insertion were obtained by first synthesizing the circuit and then inserting a sufficient number of test points to achieve the desired minimum detection probability. As can be seen from the results, by combining factoring with test point insertion, RP-SYN is able to significantly reduce the number of test points required.

TABLE III
COMPARISON WITH POST-SYNTHESIS TEST POINT INSERTION

Circuit			Post-Synthesis TPI			RP-SYN		
Name	PI	PO	Pdet (log_2)	Test PI	Test PO	Pdet (log_2)	Test PI	Test PO
chkn	29	7	-15	1	2	-15	0	1
duke2	22	29	-11	3	2	-11	1	1
exep	30	63	-11	4	3	-11	2	1
gary	15	11	-9	1	3	-9	1	1
in2	19	10	-11	3	5	-11	0	1
in7	26	10	-10	1	1	-10	1	1
misg	56	23	-8	1	1	-8	0	1
vg2	25	8	-11	2	0	-11	1	1
x1dn	27	6	-11	3	4	-11	2	1
x2dn	82	56	-8	5	3	-8	2	1
x6dn	39	5	-11	2	5	-11	0	1

VIII. CONCLUSIONS

There are two new and important features in RP-SYN that distinguish it from other methods for obtaining random pattern testable implementations. The first is that properties of algebraic factoring are used to simply fault detection

probability calculations. This enables faster and more accurate identification of r.p.r. faults, thereby, avoiding unnecessary testability-driven factoring and test point insertion. The second feature is that, when necessary, test points are inserted during the synthesis process. All previous techniques use post-synthesis test point insertion. Inserting test points during synthesis increases the effectiveness of testability-driven factoring; factors can be chosen to minimize the number of test points that are required. Moreover, the impact of the test points on area, delay, power, etc., is known during the synthesis process thereby permitting better optimization to ensure that the resulting circuit will satisfy design requirements. Inserting test points at the back-end runs the risk of causing a circuit to not meet specifications and require redesign.

RP-SYN 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). However, control circuits are an important application because they can contain large fan-in cubes that cause r.p.r. faults. Note that RP-SYN can be used for nonflattenable circuits by partitioning the circuit into flattenable logic blocks which are logically isolated during testing.

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

One way to improve the results is to combine the transformations described in [8] with those in RP-SYN. RP-SYN can be used to insert the necessary test points and factor the circuit and then the multilevel transformations described in [8] can be used to further optimize the circuit.

REFERENCES

[1] D. B. Armstrong, "On finding a nearly minimal set of fault detection tests for combinational logic networks," *IEEE Trans. Elect. Comput.*, vol. EC-15, pp. 66–73, Feb. 1966.

[2] M. J. Batek and J. P. Hayes, "Test-set preserving logic transformations," *Proc. 29th Design Automation Conf.*, 1992, pp. 454–457.

[3] R. K. Brayton, R. Rudell, A. Sangiovanni-Vincentelli, and A. R. Wang, "MIS: A multiple-level logic optimization system," *IEEE Trans. Computer-Aided Design*, vol. 6, pp. 1062–1081, Nov. 1987.

[4] R. K. Brayton, G. D. Hachtel, and A. L. Sangiovanni-Vincentelli, "Multilevel logic synthesis," *Proc. IEEE*, vol. 78, pp. 264–300, Feb. 1990.

[5] M. J. Bryan, S. Devadas, and K. Keutzer, "Testability-preserving circuit transformations," *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1990, pp. 456–459.

[6] A. J. Briers and K. A. E. Totton, "Random pattern testability by fast fault simulation," *Proc. Int. Test Conf.*, 1986, pp. 274–281.

[7] M. Chatterjee and D. K. Pradhan, "A novel pattern generator for near-perfect fault-coverage," *Proc. VLSI Test Symp.*, 1995, pp. 417–425.

[8] M. Chatterjee, D. K. Pradhan, and W. Kunz, "LOT: Logic optimization with testability—New transformations using recursive learning," *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1995, pp. 318–325.

[9] K.-T. Cheng and C. J. Lin, "Timing-driven test point insertion for full-scan and paritial-scan BIST," *Proc. Int. Test Conf.*, 1995, pp. 506–514.

[10] C.-H. Chiang and S. K. Gupta, "Random pattern testable logic synthesis," *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1994, pp. 125–128.

[11] O. Coudert, "On solving covering problems," *Proc. 33rd Design Automation Conf.*, 1996, pp. 197–202.

[12] E. Detjens, G. Gannot, R. Rudell, A. Sangiovanni-Vincentelli, and A. Wang, "Technology mapping in MIS," *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1987, pp. 116–119.

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

[14] E. B. Eichelberger and E. Lindbloom, "Random-pattern coverage enhancement and diagnosis for LSSD logic self-test," *IBM J. Res. Dev.*, vol. 27, no. 3, pp. 265–272, May 1983.

[15] B. J. Falkowski, I. Schafer, and M. A. Perkowski, "A fast computer algorithm for the generation of disjoint cubes for completely and incompletely specified Boolean functions," *Proc. 33rd Midwest Symp. Circuits and Systems*, 1990, pp. 1119–1122.

[16] J. R. Fox, "Test-point condensation in the diagnosis of digital circuits," *Proc. Inst. Elect. Eng.*, vol. 124, no. 2, pp. 89–94, Feb. 1977.

[17] G. D. Hachtel, R. M. Jacoby, K. Keutzer, and C. R. Morrison, "On properties of algebraic transformations and the synthesis of multifault-irredundant circuits," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 313–320, Mar. 1992.

[18] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," *IEEE Trans. Computers*, vol. 44, pp. 223–233, Feb. 1995.

[19] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern generation for a deterministic BIST scheme," *Proc. Int. Conf. Computer-Aided Design (ICCAD)*, 1995, pp. 88–94.

[20] V. S. Iyengar and D. Brand, "Synthesis of pseudo-random pattern testable designs," *Proc. Int. Test Conf.*, 1989, pp. 501–508.

[21] S. K. Jain and V. D. Agrawal, "Statistical fault analysis," *IEEE Design Test Comput.*, vol. 2, pp. 38–44, Jan. 1985.

[22] R. M. Karp and M. Luby, "Monte-Carlo algorithms for enumeration and reliability problems," *Proc. Annu. Symp. Foundations Computer Science*, 1983, pp. 56–64.

[23] K. Keutzer, "Dagon: Technology binding and local optimization by DAG matching," *Proc. 24th Design Automation Conf.*, 1987, pp. 341–347.

[24] B. Krishnamurthy and I. Tollis, "Improved techniques for estimating signal probabilities," *Proc. Int. Test Conf.*, 1986, pp. 244–251.

[25] B. Krishnamurthy, "A dynamic programming approach to the test point insertion problem," *Proc. 24th Design Automation Conf.*, 1987, pp. 695–704.

[26] S. Paternas and J. Rajski, "Generation of correlated random patterns for the complete testing of synthesized multi-level circuits," *Proc. 28th Design Automation Conf.*, 1991, pp. 347–352.

[27] I. Pomeranz and S. M. Reddy, "3-weight pseudo-random test generation based on a deterministic test set for combinational and sequential circuits," *IEEE Trans. Computer-Aided Design*, vol. 12, pp. 1050–1058, July 1993.

[28] J. Rajski and J. Vasudevamurthy, "The testability-preserving concurrent decomposition and factorization of Boolean expressions," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 778–793, June 1992.

[29] Y. Savaria, M. Youssef, B. Kaminska, and M. Koudil, "Automatic test point insertion for pseudo-random testing," *Proc. Int. Symp. Circuits and Systems*, 1991, pp. 1960–1963.

[30] H. D. Schnurmann, E. Lindbloom, and R. G. Carpenter, "The weighted Random test-pattern generator," *IEEE Trans. Computers*, vol. C-24, pp. 695–700, July 1975.

[31] B. H. Seiss, P. M. Trouborst, and M. H. Schulz, "Test point insertion for scan-based BIST," *Proc. European Test Conf.*, 1991, pp. 253–262.

[32] N. A. Touba and E. J. McCluskey, "Automated logic synthesis of random pattern testable circuits," *Proc. Int. Test Conf.*, 1994, pp. 174–183.

[33] ———, "Transformed pseudo-random patterns for BIST," *Proc. VLSI Test Symp.*, 1995, pp. 410–416.

[34] ———, "Synthesis of mapping logic for generating transformed pseudo-random patterns for BIST," *Proc. Int. Test Conf.*, 1995, pp. 674–682.

[35] ———, "Test point insertion based on path tracing," *Proc. VLSI Test Symp.*, 1996, pp. 2–8.

[36] ———, "Altering a pseudo-random bit sequence for scan-based BIST," *Proc. Int. Test Conf.*, 1996, pp. 167–175.

[37] H.-J. Wunderlich, "Self-test using unequiprobable random patterns," *Proc. Symp. Fault-Tolerant Computing*, 1987, pp. 258–263.

[38] N. Zacharia, J. Rajski, and J. Tyszer, "Decompression of test data using variable-length seed LFSR's," *Proc. VLSI Test Symp.*, 1995, pp. 426–433.

Nur A. Touba (S'88–M'96) received the B.S. degree in electrical engineering from the University of Minnesota, Twin Cities, where he graduate *summa cum laude* in 1990, and the M.S. and Ph.D. degrees in electrical engineering from Stanford University, Stanford, CA, in 1991 and 1996, respectively.

Since 1996, he has been an Assistant Professor at the University of Texas at Austin. His research interests are in VLSI testing, computer-aided design, and fault-tolerant computing.

Dr. Touba received a National Science Foundation (NSF) Early Faculty CAREER Award in 1997. He serves on the Technical Program Committees of the International Test Conference, International Conference on Computer-Aided Design, International Conference on Computer Design, and International Test Synthesis Workshop.

Edward J. McCluskey (S'51–M'55–SM'59–F'65–LF'94), for a photograph and biography, see p. 768 of the June 1999 issue of this TRANSACTIONS.