An Algorithmic Approach To Optimizing Fault Coverage For BIST Logic Synthesis

Srinivas Devadas
Laboratory for Computer Science
MIT, Cambridge, MA 02139

Kurt Keutzer
Department of EECS
Berkeley, CA 94720

Abstract—

Most approaches to the synthesis of built-in self test (BIST) circuitry use a manual choose-and-evaluate approach, where a particular BIST generator is chosen and then evaluated by fault-simulating the design with the vectors that the chosen generator generates. We develop an algorithmic synthesis-during-test approach in this paper, wherein the tasks of synthesizing the BIST logic and directed test pattern generation (DTPG) are intertwined to maximize the resulting fault coverage.

Our approach is applicable to a variety of BIST strategies including those that use linear- and nonlinear-feedback shift registers. We show how our method can be used to synthesize LFSR polynomials, LFSR seeds, LFSR weights, nonlinear feedback, or bit-fixing logic. Experimental data is presented.

I. Introduction

Built-in Self Test (BIST) has been of interest for some time because it provides in-field testing at request and simplifies fault diagnosis. Recently, with the cost of high-speed VLSI testers *increasing* with each process generation BIST is becoming increasingly attractive because it enables at-speed testing and the cost required is increased silicon area which is *decreasing* with each process generation. There are a few challenges to applying BIST. One fundamental problem in using BIST strategies is finding an acceptable balance between adequate fault-coverage and cost. Cost can correspond to self-test time, circuit area, or circuit performance. In addition, getting adequate fault coverage from a BIST methodology requires manual intervention and experimentation. In this paper we focus on algorithmic techniques for automatically achieving high levels of fault coverage without exceding limits on hardware cost and tester application time.

The design of BIST generators has received considerable attention. They can be categorized in the following groups: exhaustive testing [3], [22], pseudo-random testing [2], weighted random testing [5], [23], hardware generation of deterministic tests [1], [10], [21], and mixed-mode test pattern generation [11], [13]. These techniques offer different fault-coverage/cost tradeoff points; for a review see [2].

Most approaches to the synthesis of built-in self test (BIST) circuitry use a manual choose-and-evaluate approach. In this approach a specific BIST generator is chosen, e.g., LFSR with chosen primitive polynomial, and then evaluated by fault-simulating the design with the vectors that the chosen circuit generates. The results of the particular configuration are manually interpreted and the BIST generator may be accepted or another BIST generator may be tried. In this paper we employ an algorithmic synthesis-during-test approach in this paper, wherein the tasks of synthesizing the BIST logic and directed test pattern generation (DTPG) are intertwined and the particular BIST configuration(s) which provide the best test coverage with least cost can be automatically selected.

The work most closely related to ours is mixed-mode test pattern generation. In these techniques, pseudo-random patterns generated by linear-feedback shift registers (LFSR's) provide coverage of a large fraction of the faults; the remaining faults are targetted using a deterministic test pattern generator. The test vectors obtained from the deterministic test generator are mapped to additional logic attached to the LFSR. This logic can be extra gates as in [20], [8], or reseeding patterns as in [13]. These techniques use LFSR's with specified characteristic polynomials, or select polynomials from a small set, fixed *apriori*. In [13] the selection is performed by trying out each polynomial, and solving a system of linear equations to obtain the seed that will generate the given test vector.

Mixed-mode approaches also modify deterministic, or directed test pattern generators in an attempt to produce tests which can be efficiently encoded in BIST generators. For example, several approaches attempt to maximize the number of unspecified bits in the generated deterministic pattern. In [14], the heuristics used in directed test generation are tuned toward "efficient encodability" of the generated patterns into the chosen LFSR structure. These techniques, while well-motivated, are typically *ad hoc*.

The approach we present in this paper can be used for arbitrary input stimuli generators, including weighted or non-weighted linear- and nonlinear-feedback shift registers. We will focus on scan-based BIST in this paper; our approach is compatible with popular scan methodologies, including scan, partial scan, parallel scan, and boundary scan. An overview of our approach is described immediately below.

A. Overview of Approach

- A choice of the BIST testing methodology is made, which includes the determination of the test application strategy, e.g., scan path, boundary scan, partitioning of inputs, and number of BIST generators, and the type of BIST generator, e.g., linear-feedback shift register, weighted pattern generator, or counters.
- To obviate the problem of manually choosing particu-

lar LFSR configurations we use configuration inputs to enable a generic generator circuit to encode an entire family of pattern generators. The configuration inputs can change the generator function by changing the feedback function, or can change the starting state. For example, configurable feedback in a linear-feedback shift register (LFSR) can encode *all possible* characteristic polynomials. Or, weights in a weighted random pattern generator can be programmed using configuration inputs.

- To obviate the computational problem of generating the entire set of test vectors generated by a LFSR configuration we compute a large subset, or the entire set, of reachable input/state pairs of the configurable generator using implicit state enumeration methods. We will deem the generator circuit to include all the registers that apply stimuli to the device under test (DUT). States of the circuit correspond to test vectors that are applied to the DUT. Reachable input/state pairs embody information regarding the test vectors that *different* generator configurations produce.
- To obviate the problem of fault simulating each test vector in the vector set of each LFSR configuration we instead make a simple modification to a DTPG tool which restricts generated test vectors to lie *within* the computed reachable state-set of some LFSR configuration. This means that any test vector generated can be produced by at least one configuration, and possibly many configurations, of the generator circuit. Of course, the restriction of DTPG to generate patterns that lie within the reachable state set may result in the inability of DTPG to generate tests for non-redundant faults as well. If we generate the entire set of reachable states, and we are unable to produce a test vector for a fault that lies within the set, then it means that no configuration of the chosen generator circuit can possibly generate a test vector for the fault. In order to obtain a test for the fault we have to change the BIST generator by adding configurations, or change the bit-width. For a given configurable generator, it is worthwhile to compute the entire reachable set, or as large a subset as CPU time limits allow, so as to maximize fault coverage. On the other hand, we can limit testing time by limiting the number of iterations in reachability analysis.
- We choose a minimal number of configurations of the generator circuit that collectively produce all, or a chosen subset, of the test patterns. We may also restrict ourselves to selecting a fixed number of configurations (e.g., 1, 2) that maximize fault coverage, so as to limit hardware cost.

In Sections II and III we describe our approach. In Section II, we describe how we utilize configuration inputs to encode a family of BIST generators, and we describe a method to compute the set of reachable states for all generator configurations. In Section III we describe how generator configurations are selected using a directed test pattern generator. Applications of the synthesis approach to different generic generators such as LFSR's and nonlinear-feedback shift registers are described in Section IV. Experimental results are reported in Section V.

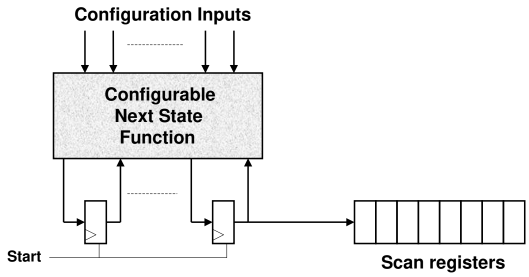

Fig. 1. Generic Configurable Generator and Associated Scan Path

II. Generator Circuits

In this section we will focus on the encoding of circuits and reachable state computation. The actual synthesis of BIST generators is the subject of Section III.

A. *Encoding Circuits Using Configuration Inputs*

In order to achieve our goal of automating the choice of a BIST generator, we need to find a way of exploring the fault coverage of a wide range of generator configurations in a computationally efficient manner. Initially, this may seem like an impossible task. For example, if a linear-feedback shift register (LFSR) is used as the basis of the generator then a large number of different polynomials can be generated depending on the configuration of the LFSR. For each configuration of the LFSR the fault coverage must be evaluated.

To obviate the problem of manually considering each configuration of the LFSR we use additional logic to encode a variety of different generator circuits in a single configurable generator. In Figure 1, we show a generic configurable generator with an associated scan path in the DUT used for test application. The circuit of Figure 1 corresponds to a generic finite state machine. The next state function can be broadly categorized as a linear-feedback function, non-linear feedback function, or a weighted linear-feedback function. A family of next state functions belonging to the same category are encoded using the configuration inputs. The **Start** signal places the flip-flops in a known state. Specific examples of configurable next state functions are given in Section IV.

We emphasize that we are using configuration inputs as means of encoding various generators into one circuit, and the

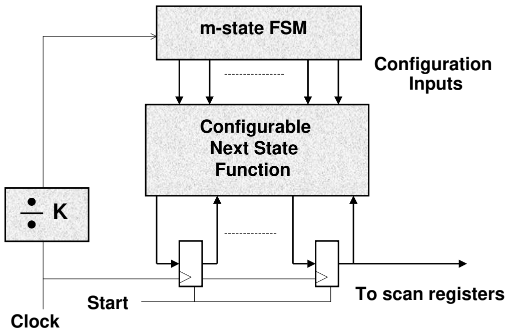

Fig. 2. BIST Generator with Fixed Number of Configurations

eventual realization of the synthesized BIST circuit will not necessarily have configuration inputs, although a straightforward realization could correspond to the configurable generator and a ROM that stores different configurations. As an extreme case, if exactly one configuration is selected, then the values of the configuration inputs are set to particular values, and the logic can be simplified accordingly. If a few configurations are selected, and if any configuration input has a constant 0/1 value over all the configurations, that input and associated logic can be removed by applying the constant 0/1 value. Once the circuit is configured, it will cycle through a fixed number of (possibly repeating) states, before it is configured again. In Figure 2, we show a logic-level realization of the BIST generator which can be obtained once the configurations are selected. The m-state FSM produces the settings of the configuration inputs every cycle, however, it is clocked every K LFSR clock cycles, where K and m, the number of selected configurations, are determined during synthesis (cf. Section III). K controls testing time, and m affects hardware cost.

The **Start** signal may be asserted every K clock cycles, if we wish the generator to begin from the same state for each configuration. We can also use configuration inputs to place the generator in particular states. The generator can then be configured with a particular "seed" or starting state.

Having obviated the problem of manually considering each LFSR configuration we must now find a way of efficiently determining the fault coverage of a particular configuration. Explicitly generating the vectors from a configuration and fault-simulating them is prohibitively expensive. Instead, we look at the problem in a different way. We consider the BIST circuitry as a finite-state machine and then determine the *set of reachable states* of this finite-state machine. This may not seem to simplify the problem at all since the set of reachable states of the BIST circuitry is precisely the set of test vectors that the BIST circuit can generate. However, the work of [9] allows us to quickly compute and

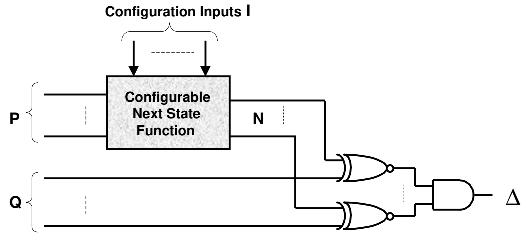

Fig. 3. The Transition Relation of a Sequential Circuit

compactly represent the set of states reachable by a finite-state machine, and thereby allows us to compactly represent all test vectors generated by a BIST configuration.

Specifically, we are interested in the set of reachable states of the generic BIST generator shown in Figure 1. [1] We wish to determine the set of test vectors generated by each BIST generator configuration. There may be exponentially many configurations, and exponentially many reachable states for any given configuration.

B. Reachable State Computation

Methods for the implicit enumeration of reachable states of a sequential circuit have been proposed [9], [7]. The main idea behind implicit enumeration is use representations of a set of vectors that are more efficient than an explicit listing. For example, the set of vectors $\{000, 001, 010, 011\}$ can be represented as $0 - -$. Generally speaking, a set of vectors can be viewed as the ON-set of a single-output function, and the function can represented as a list of cubes, reduced ordered Binary Decision Diagrams (ROBDD's) [6], or multilevel circuit.

Implicit state enumeration methods use a breadth-first search procedure to compute the set of reachable states. This involves converting the circuit C into a transition relation [9], which relates the circuit inputs, present state variables and next state variables. These methods typically use ROBDD's as the underlying representation of sets.

The transition relation $\Delta(I,\ P,\ Q)$ is a single-output function illustrated in Figure 3. We convert the next state function of the circuit C into a relation. I is the set of (configuration) inputs to the circuit, P is the set of present state variables, and Q is a set of "pseudo" next state variables. $\Delta(I = i, P = p, Q = q) = 1$ if applying i when the machine is in state p produces the combination q on the next state variables N (cf. Figure 3), otherwise it is 0.

In conventional reachable state computation, we are only concerned with reachability, and not with the input combinations that lead to particular states. The inputs are existentially

[1] Note that the scan registers are included in the circuit. This is because we want the states of the BIST generator to correspond to the test vectors applied to the DUT. However, the logic of the DUT is not included.

quantified from the transition relation Δ. We compute

$$\delta(P, Q) \;=\; S_I\,\Delta(I, P, Q),$$

where S_I coresponds to the existential quantification of all the I inputs of Δ. We will call δ the smoothed transition relation.

Given a function $f(x_1,\ x_2,\ \ldots,\ x_n)$, the **existential quantification** of f with respect to x_1 is computed as follows.

$$S_{x_1}\,f \;=\; f(0,\ x_2,\ \ldots,\ x_n) \vee f(1,\ x_2,\ \ldots,\ x_n).$$

If we wish to existentially quantify several inputs, we simply quantify them one at a time.

$$S_{x_1x_2}\,f \;=\; S_{x_1}\,(S_{x_2}f) \;=\; S_{x_2}\,(S_{x_1}\,f).$$

We begin with the given starting state, call it C_0, and compute C_∞ the entire set of reachable states, which is a set of combinations of the present state variables P. We describe the implicit method in Appendix VII-A.

C. Reachable Input/State Pair Computation

We cannot directly use the method described in Appendix VII-A for our application. This is because our generator circuit has no real inputs, but instead has configuration inputs which we will set to particular values that remain constant for many clock cycles, or all through self-test, when we use the circuit as a BIST generator. We wish to obtain information regarding the reachability of states under particular configurations of C which means we should retain the configuration inputs during enumeration.

We will compute reachable input/state pairs. Each input/state pair simply corresponds to an assignment to the configuration inputs and memory elements of the generator circuit. Reachable input/state pair enumeration begins with the given starting state C_0 and obtains the entire set of reachable input/state pairs $C_\infty(I, P)$ as described in Appendix VII-B. Each input/state pair is a setting of the configuration inputs and present state variables.

D. Example

We illustrate the differences between the computation procedures using a simple example. For the sake of clarity we will explicitly list states and input combinations. In reality, each set of states is represented as a list of cubes, or ROBDD.

Consider the State Transition Table with four states, and a single input shown in Figure 4(a). The transition relation Δ is simply a five-input, single-output function whose ON-set contains with the terms listed in Figure 4(a). The smoothed transition relation δ is a four-input, single-output function with the terms listed in Figure 4(b).

Assume the state 00 is the starting state. Conventional breadth-first enumeration using the smoothed transition relation δ will result in the following sets of states.

I	PS	NS
0	00	01
1	00	10
0	01	11
1	01	01
0	10	11
1	10	10
0	11	11
1	11	11

(a)

PS	NS
00	01
00	10
01	11
01	01
10	11
10	10
11	11

(b)

Fig. 4. (a) State Transition Table or Transition Relation and (b) Smoothed Transition Relation

$$C_0 = \{00\}$$

$$C_1 = \{01,\ 10\}$$

$$C_2 = \{01,\ 10,\ 11\}$$

$$C_3 = \{01,\ 10,\ 11\}$$

The reachable set of states is $C_0 \cup C_1 \cup C_2$.

Our procedure works with the transition relation Δ. In effect, it computes the set of states assuming the (configuration) input is set to 1, and the set of states assuming the (configuration) input is set to 0.

$$C_0 = \{\langle 0, 00\rangle,\ \langle 1, 00\rangle\}$$

$$C_1 = \{\langle 0, 01\rangle,\ \langle 1, 10\rangle\}$$

$$C_2 = \{\langle 0, 11\rangle,\ \langle 1, 11\rangle\}$$

$$C_3 = \{\langle 0, 11\rangle,\ \langle 1, 11\rangle\}$$

The reachable set is $C_0 \cup C_1 \cup C_2$.

Note that in our methodology, we do not need the entire set of reachable input/state pairs. Any subset will suffice; however, larger sets will imply greater freedom in test generation, implying higher fault coverage, or smaller BIST circuitry, or both.

Another important point is that reachable input/state pair computation will depend on the test application strategy. Assume that tests are applied to the DUT every p clock cycles, where p can be between 1 and the length of the scan chain. Then, we are only interested in retaining those states that are reachable in $p \times t$ clock cycles, $t \geq 1$ from the starting state.

Having found a method for implicitly representing the set of all test vectors that can be generated by a BIST configuration we need to use this information to assess fault coverage. To avoid the computationally complex problem of fault simulating this entire set of vectors what we do instead is test each vector generated by our automatic test pattern generator for membership in the vectors of a BIST configuration. Moreover, this actually allows us to tune our vector generation to generate vectors within one or more BIST configurations. This aspect will be clarified in Section III.

III. SELECTION OF GENERATOR CONFIGURATIONS

We require the use of a directed test pattern generator that uses conventional branching and backtracking search over input (or intermediate) variables in an attempt to obtain a test vector for a chosen fault. Ideally, the test pattern generator program should be modified slightly for our purpose. We will refer to the test generation program as DTPG in the sequel.

A. Test Generation

Given the set (or subset) of reachable input/state pairs $C_\infty(I, P)$, we compute $E(P) = S_I \; C_\infty(I, P)$. I corresponds to the configuration inputs of the BIST generator circuit, and each test vector corresponds to an assignment of $\{0, 1, X\}$ to variables in P. (Recall that we included the test-application/scan registers in the reachable input/state pair computation.) $E(P)$ thus corresponds to all the test vectors that can be generated by any configuration of the BIST generator.

The DTPG tool should only produce tests that lie within $E(P)$. We obtain test vectors that satisfy this property for as many faults as possible. Obviously, this restriction could result in less than maximum fault coverage. However, if $C_\infty(I, P)$ corresponds to the entire set of reachable states, and if DTPG cannot produce a test for a fault that lies within $E(P)$ given sufficient time, it tells us that no configuration of the chosen configurable BIST generator can possibly generate a test for the fault. In order to detect the fault the generator has to be changed; either more configurations have to be added, or the bit-width increased.

During DTPG, we check the partial input assignment for consistency against $E(P)$, and backtrack on inconsistencies. Alternately, we can generate test vectors and delete those that lie outside $E(P)$. The first technique is more efficient; however, it does require a small modification to DTPG.

Assume that we have a set of test vectors T such that each $t \in T$ is also in $E(P)$. We now wish to determine which configurations of the generator circuit will produce a particular test vector t. We are guaranteed that at least one configuration exists, thanks to the fact that $t \in E(P)$. For each t, we compute

$$F^t(I) \; = \; S_P(t \cdot C_\infty(I, P)).$$

That is, we intersect t with the set of reachable input/state pairs, and then existentially quantify the variables in P. This will produce the set of configurations $F^t(I)$ of the generator circuit that eventually produce t (assuming the generator is clocked enough times). As stated above, $t \in E(P) \; \Rightarrow \; F^t(I) \neq \phi$.

Now, we have, for each t, a set of configurations $F^t(I)$ of the generator circuit that eventually produce t. Our goal is to choose a minimum number of configurations that collectively generate all $t \in T$. We have two different formulations of this problem depending on whether the number of configurations of the generator circuit is small, or large.

B. Set Covering

If the number of configurations is small, we create a matrix where the rows are the test vectors, and the columns are the different configurations. The $F^t(I)$ functions tell us if a particular configuration generates a particular test vector or not. We will say that a configuration *covers* a test vector, if the configuration can generate it. We select a minimum number of configurations such that each test vector is covered by at least one configuration. This is a set covering problem for which exact algorithms exist that work well in practice [17], [16].

If we wish to limit ourselves to a certain number of configurations, say m, to limit hardware cost, then we can recast the problem as one of weighted covering to maximize fault coverage. The weight of each test vector corresponds to the number of faults it detects. We select a certain number of configurations that maximizes the cumulative weight of the covered test vectors. Note that the cumulative weight of test vectors is not necessarily the sum of the weights, since multiple test vectors may detect the same fault. The cumulative weight can be computed correctly by maintaining a list of detected faults for each test vector.

We realize the BIST generator as in Figure 2, and we will determine the minimum value of K as follows. Beginning with the starting seed, for each selected configuration, we determine the number of clock cycles required to generate any test vectors in T that are covered by the configuration. We make this determination simply by simulating the generator under the selected configuration. Denote the number of cycles for configuration i as K_i. K is set to $MAX_i \; K_i$. Note that K will be limited by the maximum number of iterations in the reachable input/state pair computation. This is because each covered test vector has to correspond to the state in some reachable input/state pair. We can therefore limit testing time by limiting the number of iterations in reachable input/state pair computation.

C. Clique Covering

The number of configurations may be exponentially many, but implicit reachable input/state pair computation can still provide us with the $F^t(I)$ functions. In the case of a large number of configurations, we formulate the configuration selection problem as one of clique covering.

A clique is a fully connected subgraph of nodes where each node in the subgraph is connected to every other node via an edge. We will consider each single node as a clique of size 1. If two nodes have an edge between them, then the pair corresponds to a clique of size 2.

We create a graph $G(V, E)$, given the test vectors T and the $F^t(I)$ functions for each $t \in T$. Each vertex $v \in V$ corresponds to a test vector t. There is an edge between vertices v_1 and v_2 if and only if the corresponding $F^t(I)$ functions have a non-empty intersection. Clearly, a collection of test vectors corresponding to a set of vertices which do not belong to a common clique cannot be generated by the same configuration. If a set of vertices is a clique, then we still have to check that the entire set of $F^t(I)$ functions have a non-empty intersection. This is because even if functions f, g, and h are such that $f \cdot g$, $g \cdot h$, and $f \cdot h$ are all non-empty, $f \cdot g \cdot h$ may be empty. We will define a legal clique to further satisfy the property that the intersection of all the corresponding $F^t(I)$ functions is non-empty. Given a legal clique l, denote the intersection of the $F^t(I)$'s as h^l.

Since clique covering is NP-hard [12], we use heuristic algorithms to quickly find a minimal set of cliques. We can generate all or a large number of maximal legal cliques. A legal clique is maximal, if the addition of any vertex to the set of vertices in the clique, results in a set that is either not a clique, or not a legal clique. We then select a minimal set of legal cliques. Given a set of legal cliques, $l_1, l_2, \ldots, l_m$, we have a set of configurations that generate all the test vectors in T, namely $h^{l_1}, h^{l_2}, \ldots, h^{l_m}$. [2]

In order to limit the configurations, we may select a given number of configurations, say m, and maximize the fault coverage. In this case, we will need a list of detected faults for each test vector t. These lists are merged to generate a list of detected faults for each maximal legal clique. We select m such cliques to maximize the total fault coverage.

We determine the value of K in the generator realization as described in Section III-B. Each h^{l_i} covers the test vectors corresponding to the clique l_i. Using simulation, we determine K_i for each h^{l_i}, and then determine K as $MAX_i\ K_i$. As before, we can limit K by limiting the number of iterations in reachable input/state pair computation, and therefore limit testing time.

D. Post Processing

After we have chosen the m configurations, and computed K, we can fault-simulate the DUT with *all* the vectors produced by each configuration $1, 2, \ldots, m$. A given configuration i may include several test vectors in T, but other vectors may be generated before the vectors in T are generated.

It is possible that later configurations do not detect any additional faults, because we are now applying many more test vectors in the initial configurations. As we perform fault-simulation, we discard any configuration that does not result in increased fault coverage.

[2]In reality, each h^{l_i} could be a set. Since each configuration within a h^{l_i} is equally desirable, we may pick any one arbitrarily. Therefore, for convenience we assume that each h^{l_i} corresponds to exactly one configuration.

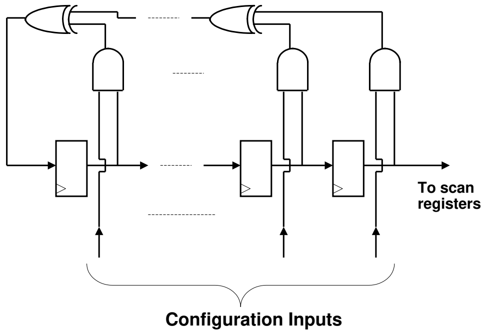

Fig. 5. Configurable LFSR

IV. Synthesis Applications

We describe how the general strategy outlined in the previous two sections can be applied to various BIST logic synthesis problems.

A. Linear-Feedback Shift Registers

A.1 LFSR Polynomial Synthesis

It is easy to configure the feedback of a linear-feedback shift register; this will allow the choice of any characteristic polynomial for the LFSR. A configurable k-bit LFSR is shown in Figure 5 with an associated scan path. It is easy to see that the values of the configuration inputs are simply the coefficients of the feedback (characteristic) polynomial of the LFSR.

We will compute the reachable input/state pairs over the set of configuration inputs and registers shown in Figure 5, as well as the scan path registers. While the number of registers is large, the logic is rather simple. In effect, the scan path state corresponds to sequences of states in the k-bit LFSR. We will begin with a given starting state for the LFSR, that is not the all zeroes state.

After computing a set of reachable input/state pairs, we can select a minimal number of configurations, say m, which correspond to a minimal number of characteristic polynomials that will generate all the test vectors obtained from DTPG. Note that these vectors all lie within $E(P)$. The LFSR is then realized as shown in Figure 2. We will determine the value of K as described in Section III-B. The **Start** signal will place the LFSR in the given starting state, every K clock cycles.

A.2 LFSR Seed Synthesis

In addition to using configurable feedback, we can also use additional configuration inputs to "seed" the LFSR. Using k configuration inputs, we can put the LFSR in any possible state. For example, we can assume if all the configuration inputs are set to 0, that the LFSR should function normally, i.e., its next state is determined from the present state and feedback logic. For any other setting of configuration inputs, the configurable LFSR moves to the state given by the setting.

Note that if we choose a primitive polynomial for the configurable feedback, then we will cycle through $2^k - 1$ states of the LFSR. (If there is an associated scan chain, with additional registers, we will not necessarily run through all combinations of scan chain states.) Seeding the LFSR is useful when k is large, and it is not feasible to run through 2^k cycles. For each seed, we can clock the LFSR for a fixed number of cycles, call it K, and then reseed it. We elaborate on this below.

Assume that we have chosen a fixed primitive polynomial, but the LFSR has seed inputs. When we compute the set of reachable input/state pairs with a primitive polynomial, the number of iterations required will be $O(2^k)$, since certain starting seeds will have a period of $O(2^k)$. If this is too large, we will limit the number of iterations in reachability analysis to $K << 2^k$. This means that, for each starting seed, we only have states that are reachable within K clock cycles from that seed in our reachable set.

After DTPG produces the test vectors, we will select a minimal number of seeds that produce the test vectors within K clock cycles. In the general case, we can select a primitive polynomial by implementing configurable feedback in the BIST generator, in addition to choosing seeds.

It is interesting to compare the above synthesis problem with the one solved by Hellebrand *et. al* [13]. Hellebrand *et. al* also synthesize seeds that generate chosen test vectors. However, their approach determines a k-bit seed that produces a given s-bit test vector when the LFSR is seeded and clocked for s cycles. In our problem a selected seed can produce more than one test vector *within* K clock cycles, where K can be significantly greater than s.

A.3 Other Variations

The inputs may be partitioned, for example into control and data input sets. In the BIST methodology, it may be that multiple LFSR's, are used to drive the different sets of inputs. Given a clocking methodology for the different LFSR's, we compute the set of reachable input/state pairs for the parallel machine corresponding to the various LFSR's and associated test application registers.

B. *Synthesis of Weighted LFSR's*

A weighted LFSR is shown in Figure 6. The block marked **Logic** is used to generate signals whose signal probabilities are closer to 0 or 1, rather than the 0.5 signal probabilities of the (unweighted) LFSR outputs. For example, the logic can correspond to the AND'ing of adjacent LFSR outputs, as in [5].

We assume a fixed polynomial and synthesize weights. The same method as that outlined in Section IV-A.1 is used, except that the configuration inputs correspond to weights rather than polynomial coefficients. We select a minimal number of weight-input combinations to cover the test vectors.

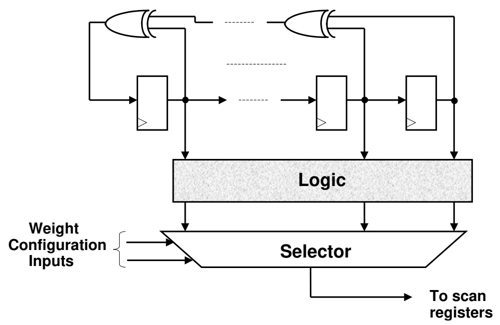

Fig. 6. LFSR with Configurable Weights

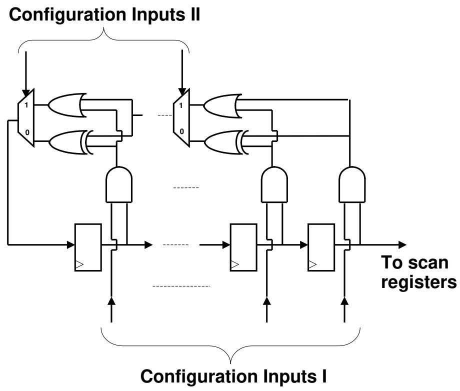

Fig. 7. Configurable Nonlinear-Feedback Shift Register

C. *Nonlinear-Feedback Shift Register Synthesis*

Our techniques can be applied unchanged to nonlinear-feedback shift registers, or autonomous finite state machine generators. Reachability analysis and configuration selection proceed exactly as before. The main decision to be made is to determine the configurability of the feedback function. For example, we can use the configurable nonlinear-feedback shift register of Figure 7, and determine a minimal number of configurations for maximal fault coverage. The realization of the configured generator may end up being an LFSR, if we happen to only choose 0 values for the second set of configuration inputs to the generator (marked Configuration Inputs II in the figure).

D. Bit-Fixing Logic

Bit-fixing logic can be used to map pseudo-random vectors generated by the LFSR to deterministic test vectors (e.g., [8]). We can synthesize bit-fixing logic by adding a configurable logic block to the outputs of the LFSR. The configurable logic is similar to the **Logic** block in Figure 6, except it will typically have multiple outputs.

V. Experiments

A. LFSR Polynomial Synthesis

We first present results on the synthesis of LFSR polynomials. We chose a few large ISCAS-89 benchmark circuits [4] for our experiments. The configurable LFSR topology of Figure 5 was chosen; the number of stages in the LFSR was fixed at 32 for all the circuits.

There are redundant faults in many of the ISCAS-89 circuits, which we ignored in our experiments. For convenience, we simply report fault coverage for the non-redundant faults. Our goal is configure chosen BIST generators to achieve maximal fault coverage, while taking into account hardware cost, and testing time restrictions.

In Table I, we report the results we obtained for LFSR polynomial synthesis. Given k stages in the LFSR, reachability analysis takes $O(2^k)$ iterations to converge from the chosen starting state; hence we limited the number of iterations in reachability analysis, K, to 10,000. We used ROBDD's to represent the set of reachable input/state pairs of the configurable LFSR plus the scan chain. For each circuit, we report the number of inputs to the circuit, and the number of test vectors generated by DTPG that lie within the reachable set $E(P)$. We ran two experiments for each circuit. First, we restricted the selection process to select exactly one polynomial ($m = 1$). The fault coverage achieved is reported under the column $m = 1$. In the second experiment, we did not restrict m. The number of polynomials selected, and the fault coverage achieved is given under the column $m = \infty$. We also report the CPU times required for reachability analysis, and for DTPG. The CPU times are on a HyperSparc with 256MB of memory. In the CPU time column, m stands for minutes, and s for seconds. Reachability analysis was performed within the SIS [18] framework. The DTPG tool used was TEGUS [19].

High fault coverages are achieved, with reasonable hardware cost. Increasing the number of polynomials increases the fault coverage, in some cases, significantly. It is possible that if the configurable feedback LFSR's were run for 10^5 (or 10^6) cycles higher fault coverages would be achieved, however, in the current implementation, this would increase the reachability analysis time by one (or two) orders of magnitude, and also increase storage requirements.

B. LFSR Polynomial/Seed Synthesis

We report results in Table II on LFSR seed and polynomial synthesis. We chose the 8 best polynomials for each of the circuits, where best corresponds to the polynomials that generated the most number of test vectors. For each circuit, we then created an LFSR with configurable feedback such that any of the 8 polynomials can be chosen; this can be done using 3 configuration inputs. We also added 32 configuration inputs for reseeding the 32-stage LFSR; these inputs can place the LFSR in any starting state. We again performed reachability analysis for 10,000 iterations. After DTPG, we chose a minimal number of polynomial/seed pairs to cover all the test vectors. In Table II we report results on LFSR polynomial/seed synthesis. We report the number of polynomials chosen (from a maximum of 8 available polynomials), the number of seeds, the fault coverage achieved, and the CPU time required for reachability analysis and DTPG. As can be seen, $> 95\%$ fault coverage is obtained for all circuits. These results are better than those obtained in Table I because the set of reachable states computed for the configurable generator is larger. The reachable set is larger because we can start from different seeds. Of course, the reseeding of the LFSR will require increased hardware cost.

We are currently experimenting with weighted LFSR's and nonlinear-feedback shift registers.

VI. Conclusions, Limitations, and Ongoing Work

Traditional approaches to the synthesis of BIST logic use an explicit choose-and-evaluate approach, where a specific BIST generator is chosen, and then evaluated using fault-simulation. This traditional approach requires manual intervention to achieve a balance between fault coverage, test time and hardware overhead and this manual intervention is an obstacle to broader usage of BIST. In this paper we have presented an algorithmic approach to the synthesis of BIST logic which aims to automate these decisions. We hope that improving the automation of BIST synthesis will broaden its usage.

There are several ways the presented methodology can be improved upon. The main problem we encountered in reachable state computation was the large number of iterations required for convergence. Typically, this number grows exponentially with k, the number of flip-flops in the BIST generator (excluding the scan flip-flops). We had to limit the number of iterations, implying that the computed set of states was only a subset of the reachable set. Techniques such as iterative squaring [7] have been proposed to decrease the number of iterations in circuits containing counters. Variants or generalizations of iterative squaring have to be developed to alleviate the iteration count problem.

ROBDD's are currently the representation of choice for implicit state enumeration of finite state machines. BIST generators form a restricted class of finite state machines, and it may be worthwhile to tailor *set representations* for this class of machines. We are exploring the use of convex polyhedra for use in reachable input/state pair computation.

A primary feature of our approach is its generality. This approach can be applied to linear-feedback or nonlinear-

CKT	#inputs	#tests	$m = 1$		$m = \infty$		CPU time	
			#poly	fcov	#poly	fcov	state	DTPG
s5378	214	119	1	98.9%	4	99.6%	11.0m	4.1m
s9234	247	144	1	91.0%	7	91.9%	16.2m	2.4m
s13207	700	245	1	95.1%	5	97.9%	16.9m	2.7m
s38417	1664	133	1	94.5%	5	95.4%	25.3m	3.8m
s38584	1464	166	1	98.5%	6	99.3%	21.6m	117s

TABLE I
LFSR POLYNOMIAL SYNTHESIS

CKT	#inputs	#tests	fcov	#poly/seed	CPU time	
					state	DTPG
s5378	214	137	99.6%	8	9.7m	4.0m
s9234	247	148	95.6%	24	16.2m	2.2m
s13207	700	273	98.7%	15	14.9m	2.3m
s38417	1664	150	98.5%	19	19.4m	3.1m
s38584	1464	175	99.3%	12	19.2m	89s

TABLE II
LFSR POLYNOMIAL/SEED SYNTHESIS

feedback shift registers, weighted or nonweighted, with or without reseeding, and with or without bit-fixing. However, we have focussed only on scan-based BIST in this paper and the choice of the type of configurable BIST generator style of configurability in the generator has to be made before the algorithmic steps of state computation, DTPG, and covering are applied. We believe that there will be further improvements if we extend our approach to examine whether for a given DUT a configurable LFSR, a weighted LFSR, a nonlinear-feedback shift register, or a combination of all three the best option. The general challenge is to develop a constructive BIST synthesis approach that determines the best configurable generator for a given DUT.

ACKNOWLEDGEMENTS

S. Devadas was supported in part by a grant from Schlumberger Foundation. We thank Prof. Tim Cheng for interesting discussions regarding BIST synthesis.

REFERENCES

[1] S. B. Akers and W. Jansz. Test Set Embedding in a Built-In Self Test Environment. In *Proceedings of the Int'l Test Conference*, pages 257–263, August 1989.

[2] P. H. Bardell, W. H. McAnney, and J. Savir. *Built-In Test for VLSI: Pseudorandom Techniques*. John Wiley and Sons, New York, New York, 1987.

[3] Z. Barzilai, D. Coppersmith, and A. L. Rosenberg. Exhaustive Generation of Bit Patterns with Applications to VLSI Self-Testing. *IEEE Transactions on Computers*, C-32(2):190–194, February 1983.

[4] F. Brglez, D. Bryan, and K. Kozminski. Combinational Profiles of Sequential Benchmark Circuits. In *Proceedings of the International Symposium on Circuits and Systems*, pages 1929–1934, Portland, Oregon, May 1989.

[5] F. Brglez, C. Gloster, and G. Kedem. Hardware-Based Weighted Random Pattern Generation for Boundary Scan. In *Proceedings of the Int'l Test Conference*, pages 264–274, August 1989.

[6] R. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[7] J. Burch, E. Clarke, K. McMillan, and D. Dill. Sequential Circuit Verification Using Symbolic Model Checking. In *Proceedings of the* 27^{th} *Design Automation Conference*, pages 46–51, June 1990.

[8] M. Chatterjee and D. K. Pradhan. A New Pattern Biasing Technique for BIST. In *Proceedings of the VLSI Test Symposium*, pages 417–425, April 1995.

[9] O. Coudert, C. Berthet, and J. C. Madre. Verification of Sequential Machines Using Boolean Functional Vectors. In *IMEC-IFIP Int'l Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, November 1989.

[10] W. Daehn and J. Mucha. Hardware Test Pattern Generators for Built-In Test. In *Proceedings of the Int'l Test Conference*, pages 315–319, 1981.

[11] C. Dufaza and G. Cambon. LFSR Based Deterministic and Pseudo-random Test Pattern Generator. In *Proceedings of the* 2^{nd} *European Test Conference*, pages 27–34, 1991.

[12] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. W. H. Freeman and Company, 1979.

[13] 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 Transactions on Computers*, 44(2):223–233, February 1995.

[14] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich. Pattern Generation for a Deterministic BIST Scheme. In *Proceedings of the Int'l Conference on Computer-Aided Design*, pages 88–94, November 1995.

[15] S. Hellebrand and H.-J. Wunderlich. Mixed Mode Test Pattern Generation Using Embedded Processors. In *Proceedings of the Int'l Test Conference*, pages 195–204, October 1996.

[16] S. Liao and S. Devadas. Solving Covering Problems Using LPR-Based Lower Bounds. In *Proceedings of the* 34^{th} *Design Automation Conference*, pages 117–120, June 1997.

[17] R. Rudell and A. Sangiovanni-Vincentelli. Exact Minimization of Mutiple-Valued Functions for PLA Optimization. In *Proceedings of the Int'l Conference. on Computer-Aided Design*, pages 352–355, 1986.

[18] E. M. Sentovich, K. J. Singh, C. Moon, H. Savoj, R. K. Brayton, and A. Sangiovanni-Vincentelli. Sequential Circuit Design Using Synthesis and Optimization. In *Proceedings of the Int'l Conference on Computer Design: VLSI in Computers and Processors*, pages 328–333, October 1992.

[19] P. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli. Combinational Test Pattern Generation Using Satisfiability. *IEEE Transactions on Computer-Aided Design*, 15(9):1167–1176, September 1996.

[20] N. A. Touba and E. J. McCluskey. Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST. In *Proceedings of the Int'l Test Conference*, pages 674–682, 1995.

[21] K-H. Tsai, J. Rajski, and M. Marek-Sadowska. Scan Encoded Test Pattern Generation for BIST. In *Proceedings of the Int'l Test Conference*, pages 548–556, November 1997.

[22] L. T. Wang and E. J. McCluskey. Circuits for Pseudo-Exhaustive Test Pattern Generation. In *Proceedings of the Int'l Test Conference*, pages 25–37, 1986.

[23] H.-J. Wunderlich. Multiple Distributions for Biased Random Test Patterns. In *Proceedings of the Int'l Test Conference*, pages 236–244, September 1988.

VII. Appendix

A. Reachable State Computation

We begin with the given starting state, call it C_0. C_0 is a setting of the present state variables P, and C_i is typically a set of states, represented as a single-output function over the present state variables, namely, $C_i(P)$; $C_i(P = p) = 1$ if the present state p belongs to the i^{th} state set, otherwise it is 0.

In order to obtain $C_{i+1}(P)$ from $C_i(P)$ we perform the operations below.

$$E_{i+1}(P, Q) \ = \ C_i(P) \cdot \delta(P, Q),$$

i.e., the current set of states is intersected with the smoothed transition relation.

$$C'_{i+1}(Q) \ = \ S_P \ E_{i+1}(P, Q),$$

where S_P corresponds to the existential quantification of all of the present state variables P. Finally, $C_{i+1}(P)$ merely corresponds to renaming the Q inputs of $C'_{i+1}(Q)$, i.e., converting the next states to present states.

All the reachable states have been found when, for some t,

$$C_{t+1}(P) \subseteq \bigcup_{i=1}^{t} C_i(P).$$

The entire set of reachable states is

$$C_\infty(P) \ = \ \bigcup_{i=1}^{t} C_i(P).$$

B. Reachable Input/State Pair Computation

We will compute reachable input/state pairs. Each input/state pair simply corresponds to an assignment to the configuration inputs and memory elements of the generator circuit. Reachable input/state pair enumeration proceeds as follows. We begin with the given starting state, call it C_0. The i^{th} set of states is represented as a function of the configuration inputs and present state variables, namely, $C_i(I, P)$. In order to obtain $C_{i+1}(I, P)$ from $C_i(I, P)$ we perform the following operations.

$$E_{i+1}(I, P, Q) \ = \ C_i(I, P) \cdot \Delta(I, P, Q).$$

$$C'_{i+1}(I, Q) \ = \ S_P \ E_{i+1}(I, P, Q).$$

Finally, $C_{i+1}(I, P)$ merely corresponds to renaming the Q variables of $C'_{i+1}(I, Q)$.

All the reachable input/state pairs have been found when, for some t,

$$C_{t+1}(I, P) \subseteq \bigcup_{i=1}^{t} C_i(I, P).$$

The entire set of reachable input/state pairs is

$$C_\infty(I, P) \ = \ \bigcup_{i=1}^{t} C_i(I, P).$$