ALTERING A PSEUDO-RANDOM BIT SEQUENCE FOR SCAN-BASED BIST

Nur A. Touba[*] and Edward J. McCluskey

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

ABSTRACT

This paper presents a low-overhead scheme for built-in self-test of circuits with scan. Complete (100%) fault coverage is obtained without modifying the function logic and without degrading system performance (beyond using scan). Deterministic test cubes that detect the random-pattern-resistant faults are embedded in a pseudo-random sequence of bits generated by a linear feedback shift register (LFSR). This is accomplished by altering the pseudo-random sequence by adding logic at the LFSR's serial output to “fix” certain bits. A procedure for synthesizing the bit-fixing logic for embedding the test cubes is described. Experimental results indicate that complete fault coverage can be obtained with low hardware overhead. Also, the proposed approach permits the use of small LFSR's for generating the pseudo-random bit sequence. The faults that are not detected because of linear dependencies in the LFSR can be detected by embedding deterministic cubes at the expense of additional bit-fixing logic. Data is presented showing how much additional logic is required for different size LFSR's.

1. INTRODUCTION

In built-in self-test (BIST), internal hardware is used to generate test patterns that are applied to the circuit-under-test and to analyze the output response. A low-overhead approach for BIST in circuits with scan is to use a linear feedback shift register (LFSR) to shift a pseudo-random sequence of bits into the scan chain. When a pattern has been shifted into the scan chain, it is applied to the circuit-under-test and the response is loaded back into the scan chain and shifted out into a serial signature register for compaction as the next pattern is shifted into the scan chain. Figure 1 shows a block diagram for this “test-per-scan” BIST scheme. Unfortunately, many circuits contain random-pattern-resistant (r.p.r) faults [Eichelberger 83] which limit the fault coverage that can be achieved with this approach. Three methods for improving the fault coverage for a test-per-scan BIST scheme are:

1. Modify Circuit-Under-Test: The circuit-under-test is modified by either inserting test points [Eichelberger 83], [Cheng 95], [Touba 96], or by redesigning it [Touba 94], [Chiang 94], [Chatterjee 95] to improve the fault detection probabilities. These techniques generally add extra levels of logic to the circuit which may degrade system performance. Moreover, in some cases it is not possible or not desirable to modify the function logic (e.g., macrocells, cores, proprietary designs).

2. Weighted Pseudo-Random Sequence: Logic is added to change the probability of each bit in the sequence being a ‘1’ or a ‘0’ in a way that biases the patterns that are generated towards those that detect the r.p.r. faults. The weight logic can be placed either at the input of the scan chain [Brglez 89b] or in the individual scan cells themselves [Muradali 90]. Multiple weight sets are usually required due to conflicting input values needed to detect r.p.r. faults [Wunderlich 90]. The weight sets need to be stored and control logic is required to switch between them, so the hardware overhead can be large.

3. Mixed-Mode: Deterministic patterns are used to detect the faults that the pseudo-random patterns miss. Storing deterministic patterns in a ROM requires a large amount of hardware overhead. Koenemann, in [Koenemann 91], proposed a technique based on reseeding an LFSR that reduces the storage requirements. The LFSR that is used for generating the pseudo-random patterns is also used to generate deterministic *test cubes* (test patterns with unspecified inputs) by loading it with computed seeds. The number of bits that need to be stored is reduced by storing a set of seeds instead of a set of deterministic patterns. Hellebrand *et al.*, in [Hellebrand 92], [Venkataraman 93], and [Hellebrand 95a], proposed an improved technique that uses a multiple-polynomial LFSR

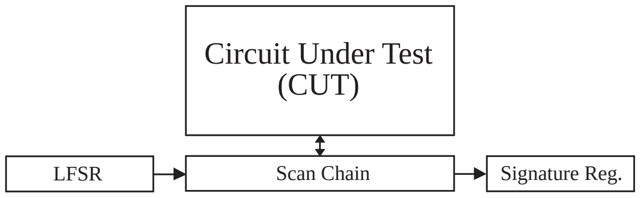


Figure 1 . “Test-Per-Scan” BIST Scheme

[*] Nur A. Touba is now with the Department of Electrical and Computer Engineering, University of Texas, Austin, TX

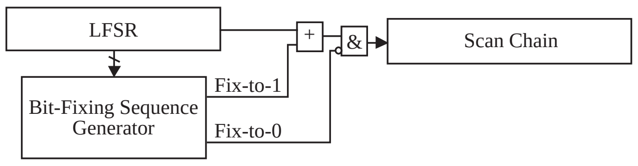


Figure 2 . Logic for Altering the Pseudo-Random Bit Sequence

for encoding a set of deterministic test cubes. By "merging" and "concatenating" the test cubes, they further reduce the number of bits that need to be stored. Even further reduction can be achieved by using variable-length seeds [Zacharia 95] and a special ATPG algorithm [Hellebrand 95b].

This paper presents a new mixed-mode approach in which deterministic test cubes are embedded in the pseudo-random sequence of bits. Logic is added at the serial output of the LFSR to alter the pseudo-random bit sequence so that it contains patterns that detect the r.p.r. faults. This is accomplished by "fixing" certain bits in the sequence. As illustrated in Fig. 2, logic is added to generate a bit-fixing sequence that alters the pseudo-random sequence by causing certain bits to be fixed to either a '1' or a '0'. A procedure is described for designing the bit-fixing sequence generator in a way that reduces area overhead.

The new test-per-scan BIST scheme presented in this paper is sort of a hybrid approach. It is different from weighted pattern testing because it is not based on probability. It guarantees that certain test cubes will be applied to the circuit-under-test during a specified test length. Also, it doesn't require a multi-phase test in which control logic is needed to switch to different weight sets for each phase. The control is very simple because there is only one phase.

The scheme presented in this paper is also different from previous mixed-mode schemes for test-per-scan BIST. Previous mixed-mode schemes for test-per-scan BIST have been based on storing compressed data in a ROM. In the proposed scheme, no data is stored in a ROM, rather a multilevel circuit is used to dynamically fix bits in a way that exploits bit correlation (same specified values in particular bit positions) among the test cubes for the r.p.r. faults. Small numbers of correlated bits are fixed in selected pseudo-random patterns to make the pseudo-random patterns match the test cubes. So rather than trying to compress the test cubes themselves, the proposed scheme compresses the bit differences between the test cubes and a selected set of pseudo-random patterns. Since there are so many pseudo-random patterns to choose from, a significant amount of compression can be achieved resulting in low overhead.

Schemes based on reseeding an LFSR require that the LFSR have at least as many stages as the maximum number of bits specified in any test cube. A hardware tradeoff that is made possible by the scheme presented in this paper is that a smaller LFSR can be used for generating the pseudo-random bit sequence. This may cause some faults to not be detected because of linear dependencies in the patterns that are generated, but deterministic test cubes for those faults can be embedded at the expense of additional logic in the bit-fixing sequence generator. Data is presented showing how much additional logic is required for different size LFSR's.

The paper is organized as follows: In Sec. 2, the architecture of the bit-fixing sequence generator is described. In Sec. 3, the procedure for designing the bit-fixing sequence generator is presented. In Sec. 4, experimental results are shown for benchmark circuits. Sec. 5 is a conclusion.

2. ARCHITECTURE OF BIT-FIXING SEQUENCE GENERATOR

The purpose of the bit-fixing sequence generator is to alter the pseudo-random sequence of bits that is shifted into the scan chain in order to embed deterministic test cubes in the sequence. This is done by generating a sequence of *fix-to-1* and *fix-to-0* control signals that fix certain bits to either '1' or '0'. The architecture of the bit-fixing sequence generator is shown in Fig. 3. For a scan chain of length *m*, there is a *Mod-(m+1) Counter* that counts the number of bits that have been shifted into the scan chain. After *m* bits, the scan chain is full, so when the counter reaches the *(m+1)* state, the pattern in the scan chain is applied to the circuit-under-test and the response is loaded back into the scan chain. At this point, the LFSR contains the starting state for the next pattern that will be shifted into the LFSR. The *Bit-Fixing Sequence Selection Logic* decodes the starting state in the LFSR and selects the bit-fixing sequence that will be used for the next pattern. The selected bit-fixing sequence identifier is loaded into the *Sequence ID Register.* As the counter counts through the next *m* bits that are shifted into the scan chain, the *Bit-Fixing Sequence Generation Logic* generates the *fix-to-1* and *fix-to-0* control signals based on the bit-fixing sequence identifier stored in the *Sequence ID Register* and the value of the counter (see Fig. 6 for a specific example).

One thing that should be pointed out is that the *Mod-(m+1) Counter* is not additional overhead. It is needed in the control logic for any test-per-scan BIST technique to generate a control signal to clock the circuit-under-test when the scan chain is full. Thus, this scheme takes advantage of existing BIST control logic.

For each pattern that is shifted into the scan chain, the bit-fixing sequence generator is capable of generating one of 2^n different bit-fixing sequences where *n* is the size of the *Sequence ID Register*. A deterministic test cube for an r.p.r. fault can be shifted into the scan chain by generating an appropriate bit-fixing sequence for a pseudo-random pattern generated by the LFSR. The bit-fixing sequence fixes certain bits in the pseudo-random pattern such that the resulting pattern that is shifted into the scan chain detects the r.p.r. fault. The bit-fixing sequence generator

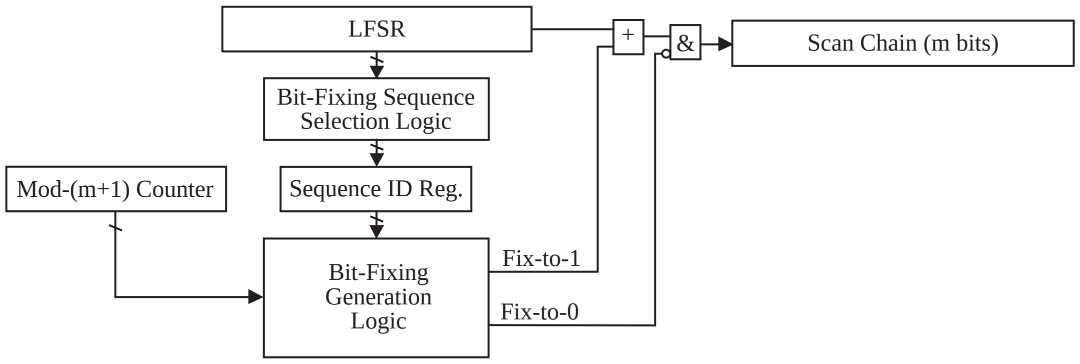


Figure 3. Architecture of Bit-Fixing Sequence Generator

must be designed so that it generates enough deterministic test cubes to satisfy the fault coverage requirement. The key to minimizing the area overhead for this approach is careful selection of the bit-fixing sequences that are generated.

One characteristic of the test cubes for r.p.r. faults is that subsets of them often have the same specified values in particular bit positions (this will be referred to as "bit correlation"). For example, the test cubes 11011, 11X00, and 1X0X0, are correlated in the 1st, 2nd, and 3rd bit positions, but not in the 4th and 5th. That is because all of the specified bits in the 1st and 2nd bit positions are 1's, and all of the specified bits in the 3rd bit position are 0's. However, the 4th and 5th bit positions have conflicts because some of the specified values are 1's and some are 0's. Note that the unspecified values (X's) don't matter. The reason why a significant amount of bit correlation often exists among the test cubes for the r.p.r. faults is probably due to the fact that several r.p.r. faults may be caused by a single random pattern resistant structure in the circuit. For example, if there is a large fan-in AND gate in a circuit, then that may cause all of the input stuck-at 1 faults and the output stuck-at 0 fault of the gate to be r.p.r. Many of the specified values in particular bit positions of the test cubes for these r.p.r. faults will be the same. Thus, there will be a significant amount of bit correlation among the test cubes. This phenomenon is seen in weighted pattern testing where biasing certain bit positions results in detecting a significant number of r.p.r. faults.

In the scheme presented in this paper, bit correlation among the test cubes for the r.p.r. faults is used to minimize both the number of different bit-fixing sequences that are required and the amount of decoding logic. A procedure for designing the bit-fixing sequence generator is described in the next section.

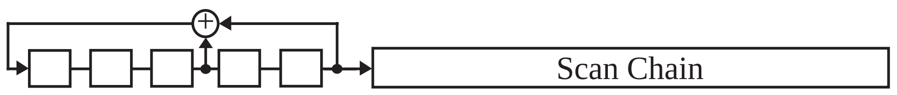
3. DESIGNING BIT-FIXING SEQUENCE GENERATOR

For a given LFSR and circuit-under-test, this section describes an automated procedure for designing a bit-fixing sequence generator to satisfy test length and fault coverage requirements. The bit-fixing sequence generator is designed to alter the pseudo-random bit sequence generated by the LFSR to achieve the desired fault coverage for the given test length (number of scan patterns applied to the circuit-under-test).

3.1 Obtaining Test Cubes

The first step is to simulate the *r*-stage LFSR for the given test length *L* to determine the set of pseudo-random patterns that are applied to the circuit-under-test. For each of the *L* patterns that are generated, the starting *r*-bit state of the LFSR is recorded (i.e., the contents of the LFSR right before shifting the first bit of the pattern into the scan chain). Fault simulation is then performed on the circuit-under-test for the pseudo-random patterns to see which faults are detected and which are not. The pattern that *drops* each fault from the fault list (i.e., detects the fault for the first time) is recorded. The faults that are not detected are the faults that require altering of pseudo-random bit sequence. The pseudo-random bit sequence must be altered to generate test cubes that detect the undetected faults. An automatic test pattern generation (ATPG) tool is used to obtain test cubes for the undetected faults by leaving unspecified inputs as *X's*.

A simple contrived design example will be used to illustrate the procedure described in this paper. A bit-fixing sequence generator will be designed to provide 100% fault coverage for a test length of 12 patterns ($L=12$) generated by a 5-stage LFSR ($r=5$) and shifted into a 12 bit scan chain ($m=12$). Figure 4 shows the 12 patterns that are generated by the LFSR and applied to the circuit-under-test through the scan chain. For each pattern, the starting state of the LFSR is shown and the number of faults that are dropped from the fault list is shown. Five of the patterns drop faults while the other 7 do not. The pseudo-random patterns detect 16 out of 20 possible faults giving a fault coverage of 80%. An ATPG tool is used to obtain test cubes for the 4 undetected faults. The bit-fixing sequence generator must be designed so that it alters the pseudo-random bit sequence in a way that all 4 test cubes are generated in the scan chain.



Starting State		Pattern Shifted into Scan Chain	Num. of Faults Dropped
0 1 0 1 1	→	0 1 0 0 0 0 1 0 1 0 1 1	7
1 1 0 1 0	→	1 1 1 1 1 0 0 1 1 0 1 0	5
1 1 0 0 0	→	0 1 0 1 1 1 0 1 1 0 0 0	2
0 0 0 0 1	→	1 1 0 1 0 0 1 0 0 0 0 1	0
1 1 1 0 0	→	1 1 0 0 0 1 1 1 1 1 0 0	1
0 1 1 1 0	→	0 0 0 0 1 0 1 0 1 1 1 0	0
0 1 0 0 1	→	1 1 1 0 0 1 1 0 1 0 0 1	0
0 0 0 1 1	→	0 1 1 1 0 1 1 0 0 0 1 1	0
0 0 1 0 1	→	0 1 0 0 1 0 0 0 0 1 0 1	0
1 0 0 1 1	→	0 0 0 1 1 1 1 1 0 0 1 1	1
1 1 0 1 1	→	0 0 1 0 1 0 1 1 1 0 1 1	0
0 0 1 0 0	→	1 0 0 1 1 0 1 0 0 1 0 0	0
			16/20
			Fault Coverage = 80%

Test Cubes for Undetected Faults:
1 1 1 X 0 0 X X X X 0 0
1 0 1 X 1 0 X X X X 0 X
0 0 0 X X 1 X X X X 0 0
0 1 X X 0 1 X X X X 1 0

Figure 4 . Design Example (r=5, L=12, m=12): Obtaining the Test Cubes

Starting LFSR states for Patterns that Drop Faults: 01011, 11010, 11000, 11100, 10011

F′ = (01011 + 11010 + 11000 + 11100 + 10011)′

Largest Implicant in **F**′: 00XXX

	Starting LFSR State		Corresponding Scan Pattern
Patterns Decoded by 00XXX:	0 0 0 0 1	→	1 1 0 1 0 0 1 0 0 0 0 1
	0 0 0 1 1	→	0 1 1 1 0 1 1 0 0 0 1 1
	0 0 1 0 1	→	0 1 0 0 1 0 0 0 0 1 0 1
	0 0 1 0 0	→	1 0 0 1 1 0 1 0 0 1 0 0

Consider all 4 Test Cubes		Eliminate One Test Cube
1 1 1 X 0 0 X X X X 0 0	Test Cubes	1 1 1 X 0 0 X X X X 0 0
1 0 1 X 1 0 X X X X 0 X		1 0 1 X 1 0 X X X X 0 X
0 0 0 X X 1 X X X X 0 0		
0 1 X X 0 1 X X X X 1 0		0 1 X X 0 1 X X X X 1 0
0 (last bit) ↓	Bits to Fix	1 (third bit) ↓ 0 (last bit) ↓
1 1 0 1 0 0 1 0 0 0 0 0	Resulting Scan Patterns	**1 1 1** 1 **0 0** 1 0 0 0 **0 0**
0 1 1 1 **0 1** 1 0 0 0 **1 0**		**0 1** 1 1 **0 1** 1 0 0 0 **1 0**
0 1 0 0 1 0 0 0 0 1 0 0		0 1 1 0 1 0 0 0 0 1 0 0
1 0 0 1 1 0 1 0 0 1 0 0		**1 0 1** 1 **1 0** 1 0 0 1 **0** 0
1 Test Cube Embedded		3 Test Cubes Embedded

Figure 5 . Design Example: Finding Decoding Function and Set of Bits to Fix for the New *Sequence ID Register* Bit

3.2 Embedding Test Cubes

Once the set of test cubes for the undetected faults has been obtained, the bit-fixing sequence generator is then designed to embed the test cubes in the pseudo-random bit sequence. The test cubes are embedded in a way that guarantees that faults that are currently detected by the pseudo-random bit sequence will remain detected after the test cubes are embedded. This is done by only altering patterns that don't drop any faults. As long as the patterns that drop faults are not altered, the dropped faults are guaranteed to remain detected. This ensures that fault coverage will not be lost in the process of embedding the test cubes.

The goal in designing the bit-fixing sequence generator is to embed the test cubes with a minimal amount of hardware. A hill-climbing strategy is used in which one bit at a time is added to the *Sequence ID Register* based on maximizing the number of test cubes that are embedded each time. Bits continue to be added to the *Sequence ID Register* until a sufficient number of test cubes have been embedded to satisfy the fault coverage requirement. Complete fault coverage can be obtained by embedding test cubes for all of the undetected faults.

For each bit that is added to the *Sequence ID Register*, the first step is to determine for which patterns the bit will be active (i.e., which patterns it will alter). In order not to reduce the fault coverage, it is important to choose a set of patterns that don't currently drop any faults in the circuit-under-test. In order to minimize the *Bit-Fixing Sequence Selection Logic*, it is important to choose a set of patterns that are easy to decode. The set of patterns for which the new *Sequence ID Register* bit will be active are decoded from the starting state of the LFSR for each pattern. Let $\mathbf{F}$ be a Boolean function equal to the sum of the minterms corresponding to the starting state for each pattern that drops faults. Then an implicant in $\mathbf{F}'$ corresponds to a set of patterns that don't drop faults and can be decoded by an n-input AND gate where n is the number of literals in the implicant. A binate covering procedure can be used to choose the largest implicant in $\mathbf{F}'$ (see [Touba 95]). The largest implicant requires the least logic to decode and corresponds to the largest set of pseudo-random patterns that don't drop any faults and thus is most desirable. These are the patterns that will activate, and hence be altered by, the new *Sequence ID Register* bit.

In the design example, there are 5 starting LFSR states that correspond to the patterns that drop faults. They are listed at the top of Fig. 5. The function $\mathbf{F}$ is formed, and the largest implicant in the complement of $\mathbf{F}$ is found. The largest implicant is *00XXX*. Whenever the first two bits in a starting state of the LFSR are both '0', then the new *Sequence ID Register* bit is activated. Thus, there are 4 patterns for which the new *Sequence ID Register* bit will be activated.

After the set of patterns that activate the new *Sequence ID Register* bit have been determined, the next step is to determine which bits in the patterns will be fixed when the new *Sequence ID Register* bit is activated. The goal is to fix the bits in a way that embeds as many test cubes as possible. The strategy is to find some good candidate sets of bits to fix and then compute how many test cubes would be embedded if each were used. The candidate that embeds the largest number of test cubes is then selected.

The candidate sets of bits to fix are determined by looking at bit correlation among the test cubes. For example, if the two test cubes *1010X* and *00X11* are to be embedded, then fixing the 2nd bit position to a '0', the 3rd bit position to a '1', and the 5th bit position to a '1' would help to embed both test cubes in the pseudo-random bit sequence. But, fixing the 1st bit to a '1' or fixing the 4th bit to a '0' would only help to embed the first test cube; it would prevent the second test cube from being embedded. The reason for this is that the two test cubes have conflicting values in the 1st and 4th bit. So given a set of test cubes to embed, the best bits to fix are the ones in which there are no conflicting values among the test cubes. The procedure for selecting the set of bits to fix is as follows (the procedure is illustrated for the design example at the bottom of Fig. 5):

1. <u>Place all test cubes to be embedded into the initial set of test cubes.</u>

Begin by considering all of the test cubes that need to be embedded.

> [In the design example in Fig. 5, all 4 test cubes are considered initially.]

2. <u>Identify bits where there are no conflicting values among the test cubes.</u>

Look at each bit position. If one or more test cubes has a '1' and one or more test cubes has a '0' in the bit position then there is a conflict. If all of the test cubes have either a '1' ('0') or an 'X', then the bit can be fixed to a '1' ('0').

> [In the design example in Fig. 5, when all 4 test cubes are considered, only the last bit position has no conflicting values. All 4 of the test cubes have either a '0' or an 'X' in the last bit position.]

3. <u>Compute the number of test cubes that would be embedded by fixing this candidate set of bits.</u>

For each pattern that activates the new *Sequence ID Register* bit, fix the set of bits that was determined in step 2. Count the number of test cubes that are embedded in the resulting patterns.

> [In the design example in Fig. 5, when the last bit position is fixed to a '0' in the 4 scan patterns that activate the new Sequence ID Register bit, it enables the test cube *01XX01XXXX10* to be embedded in the pseudo-random pattern *011101100011*.]

4. If the number of test cubes embedded is larger than that of the best candidate, then mark this as the best candidate.

The goal is choosing the set of bits to fix is to embed as many test cubes as possible.

5. Remove the test cube that will eliminate the most conflicts.

One test cube is removed from consideration in order to increase the number of bits that can be fixed. The test cube that is removed is chosen based on reducing the number of conflicting bits in the remaining set of test cubes.

[In the design example in Fig. 5, if the third test cube is eliminated from consideration, the remaining 3 test cubes have two specified bit positions where there are no conflicts. The third bit can be fixed to a '1' in addition to fixing the last bit to a '0'.]

6. If the number of test cubes that are embedded by the best candidate is greater than the number of test cubes that remain, then select the best candidate. Otherwise, loop back to step 2.

The next candidate set of bits to fix will only help to embed the remaining set of test cubes, and therefore has limited potential. If it is not possible for the next candidate to embed more test cubes than the best candidate, then the best candidate is selected as the set of bits to fix.

7. Eliminate as many fixed bits as possible without reducing the number of embedded test cubes.

In order to minimize hardware area, it is desirable to fix as few bits as possible. It may be possible to embed the test cubes without fixing all of the bits in the selected set. An attempt is made to reduce the number of fixed bits by eliminating one bit at a time and checking to see if the same test cubes are embedded.

The bit-fixing sequence generator is designed so that when the new *Sequence ID Register* bit is activated, the set of bits selected by the procedure above is fixed. The pseudo-random patterns that are altered to embed each test cube are added to the set of patterns that drop faults (one pattern per embedded test cube). This is done to ensure that those patterns are not further altered such that they would no longer embed the test cubes. If the fault coverage is not sufficient after adding the new *Sequence ID Register* bit, then another *Sequence ID Register* bit is added to embed more test cubes.

In the design example in Fig. 5, when all 4 test cubes are considered, the only specified bit position where there are no conflicts is the last bit position which can be fixed to a '0'. Fixing this bit enables one test cube to be embedded. However, when one of the test cubes is eliminated from consideration then the remaining 3 test cubes have two specified bit positions where there are no conflicts. Fixing these two bits enables all 3 of the remaining test cubes to be embedded. Thus, this is the selected set of bits to fix when the new *Sequence ID Register* bit is activated. There is still one test cube that has not been embedded. Since complete fault coverage is required, another bit must be added to the *Sequence ID Register*. The three pseudo-random patterns in which the three test cubes were embedded are added to the set of patterns that drop faults, and the procedure for adding a new *Sequence ID Register* bit is repeated.

3.3 Synthesizing Bit-Fixing Sequence Generation Logic

When enough bits have been added to the *Sequence ID Register* to provide sufficient fault coverage, the remaining task is to synthesize the *Bit-Fixing Sequence Generation Logic*. The *Bit-Fixing Sequence Generation Logic* generates the *fix-to-1* and *fix-to-0* control signals to fix the appropriate bits in the sequence depending on which *Sequence ID Register* bits are active. For each *Sequence ID Register* bit that is active, control signals are generated when certain states of the counter are decoded.

The process of constructing the *Bit-Fixing Sequence Generation Logic* is best explained with an example. The *Bit-Fixing Sequence Generation Logic* for the design example is shown in Fig. 6. The first bit in the *Sequence ID Register* is activated whenever the first two bits in the starting seed for a pattern are both '0'. This condition is decoded using a two-input AND gate and loading the *Sequence ID Register* right before shifting a new pattern into the scan chain. When the first bit in the *Sequence ID Register* is active, it fixes the 1st bit shifted into the scan chain to a '0' and the 10th bit shifted into the scan chain to a '1'. This is done by generating a *fix-to-0* signal when the counter is in the "cnt-1" state and a *fix-to-1* signal when the counter is in "cnt-10" state. The second bit in the *Sequence ID Register* is activated whenever the 3rd and 4th bit in the starting seed for a pattern are both '1'. When the second bit in the *Sequence ID Register* is activated, it fixes the 2nd bit shifted into the scan chain to a '0'. This is done by generating a *fix-to-0* signal when the counter is in "cnt-2" state.

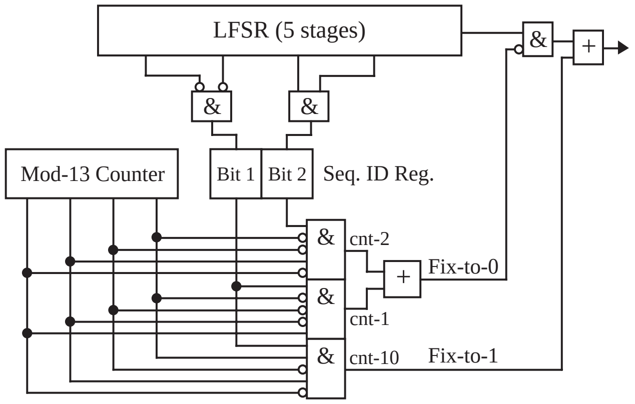


Figure 6. Design Example: Bit-Fixing Sequence Generation Logic Prior to Multilevel Logic Optimization

Table 1 . Results for 100% Fault Coverage for a Test Length of 10,000 Pseudo-Random Patterns

Circuit			Reseeding [Hellebrand 95b]		Bit-Fixing Sequence Generator		
Name	Scan Size	Max. Num Specified Bits	LFSR Size	ROM Bits	LFSR Size	Seq ID Reg Size	Literal Count
s420	34	20	20	250	20	1	27
					14	3	70
					10	4	70
s641	54	22	22	183	22	2	63
					14	4	87
					9	6	109
s838	66	36	36	1623	36	5	168
					14	7	176
					12	7	199
s1196	32	17	17	267	17	4	67
					14	4	71
					12	8	102
s5378	214	19	27	726	19	3	163
					14	4	174
					12	9	367
C2670	233	48	60	3412	48	4	328
					16	5	334
					10	12	427
C7552	207	100	100	5241	100	7	741
					36	8	782
					17	13	828

When constructing the *Bit-Fixing Sequence Generation Logic*, the states of the counter can be decoded by simply using n-input AND gates where n is equal to the number of bits in the counter. However, once the logic has been constructed, it should be minimized using a multilevel logic optimization tool. The don’t care conditions due to the unused states of the counter can be used to minimize the logic, but more importantly, the logic can be factored. Because the number of inputs to the logic is small, factoring is very effective for significantly minimizing the *Bit-Fixing Sequence Generation Logic.*

Since the *Bit-Fixing Sequence Generation Logic* is synthesized from a two-level starting point, it can be made prime and irredundant using synthesis procedures such as those described in [Bostick 87] and [Rajski 92]. If the full don't care set is used (i.e., all input combinations that don't occur during BIST are don't cares), then the resulting logic will be 100% tested for single stuck-at faults by the patterns applied during BIST [Bartlett 88].

4. EXPERIMENTAL RESULTS

The procedure described in this paper has been implemented in Stanford CRC’s synthesis-for-test tool, TOPS, and was used to design bit-fixing sequence generators for ISCAS 85 [Brglez 85] and ISCAS 89 [Brglez 89a] benchmark circuits that contain random-pattern-resistant faults. The primary inputs and flip-flops in each circuit were configured in a scan chain. The bit-fixing sequence generators were designed to provide 100% fault coverage of all detectable single stuck-at faults for a test length of 10,000 patterns.

The results are shown in Table 1. The size of the scan chain is shown for each circuit followed by the maximum number of specified bits in any test cube contained in the test set reported in [Hellebrand 95b]. If the characteristic polynomial of the LFSR is primitive (as is the case in these results) and the number of stages in the LFSR is greater than or equal to the number of specified bits in a test cube for a fault, then the LFSR is guaranteed to be capable of generating patterns that detect the fault. If the number of stages in the LFSR is less than the number of specified bits in a test cube for some fault, then it may not be possible for the LFSR to generate a pattern that detects the fault due to linear dependencies in the LFSR. Results are shown for the bit-fixing sequence generator required for different size LFSR's. For each different size LFSR, the number of bits in the *Sequence ID Register* is shown along with the factored form literal count for the multilevel logic required to implement the bit-fixing sequence generator. For each circuit, results were shown for an LFSR with as many stages as the maximum number of specified bits. These LFSR's are guaranteed to be capable of generating patterns to detect all of the faults. For the smaller LFSR's, there are some faults that are not detected because of linear dependencies in the LFSR. Extra test cubes must be embedded in order to detect those faults thereby resulting in an increase in the area of the bit-fixing sequence generator. As can be seen, in some cases adding just a small amount of logic to the bit-fixing sequence generator permits the use of a much smaller LFSR. Consider *C2670*, using a 16-stage LFSR instead of a 48-stage LFSR only requires an additional 6 literals. However, in some cases there is a large increase in the amount of logic required for using a smaller LFSR. Consider *s5378*, using a 12-stage LFSR instead of a 14-stage LFSR increases the amount of logic in the bit-fixing sequence generator by more than a factor of two.

Results for the reseeding method presented in [Hellebrand 95b] are shown in Table 1 for comparison. The size of the LFSR and the number of bits stored in a ROM are shown. Note that the reseeding method requires that the LFSR have at least as many stages as the maximum number of specified bits in any test cube. It is difficult to directly compare the two methods because they are implemented differently (ROM versus multilevel logic) and require very different control logic. The reseeding method requires that the LFSR have programmable feedback logic and parallel load capability as well as additional control logic for loading the seeds from the ROM.

5. CONCLUSIONS

There are three new and important features that distinguish the mixed-mode scheme presented in this paper from other mixed-mode schemes for test-per-scan BIST. The first is that test cubes for the random-pattern-resistant faults are embedded in the pseudo-random bit sequence. Since there are so many possible pseudo-random patterns in which to embed each test cube, the bit-fixing required to embed a set of test cubes can be correlated in certain bit positions to minimize hardware. The second feature is that a one-phase test is used. Having only one phase simplifies the BIST control logic. The third feature is that smaller LFSR's can be used. There is a tradeoff between the size of the LFSR and the amount of bit-fixing logic, so the LFSR size can be chosen to minimize the overall area. These three features make the scheme presented in this paper an attractive option for BIST in circuits with scan.

One way to achieve an even greater overhead reduction may be to combine the bit-fixing technique described in this paper with reseeding techniques. By reseeding the LFSR with just a few selected seeds to generate some of the least correlated test cubes that require a lot of bit-fixing to embed, it may be possible to significantly reduce the complexity of the bit-fixing sequence generator. This idea is currently being investigated.

Another way to reduce the overhead would be to develop a special ATPG procedure that finds test cubes for each r.p.r. fault in a way that maximizes the bit correlation among the test cubes. This would reduce the amount of bit-fixing that is required to embed the test cubes. The results in [Hellebrand 95b] indicate that modifying the ATPG procedure can make significant difference. This idea is also being investigated.

Note that while the experimental results presented in this paper were for single stuck-at faults, the approach of embedding deterministic test cubes in a pseudo-random sequence works for other fault models (e.g., multiple stuck-at faults and bridging faults) as well.

ACKNOWLEDGMENTS

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

REFERENCES

[Bartlett 88] Bartlett, K.A., R.K. Brayton, G.D. Hachtel, R.M. Jacoby, C.R. Morrison, R.L. Rudell, A. Sangiovanni-Vincentelli, and A.R. Wang, "Multilevel Logic Minimization Using Implicit Don't Cares," *IEEE Transactions on Computer-Aided Design*, Vol. 7, No. 6, pp. 723-740, Jun. 1988.

[Bostick 87] Bostick, D., G.D. Hachtel, R. Jacoby, M.R. Lightner, P. Moceyunas, C.R. Morrison, and D. Ravenscroft, "The Boulder Optimal Logic Design System," *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 62-65, 1987.

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

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

[Brglez 89b] Brglez, F., C. Gloster, and G. Kedem, "Hardware-Based Weighted Random Pattern Generation for Boundary Scan," *Proc. of International Test Conference*, pp. 264-274, 1989.

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

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

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

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

[Hellebrand 92] Hellebrand, S., S. Tarnick, J. Rajski, and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *Proc. of International Test Conference*, pp. 120-129, 1992.

[Hellebrand 95a] Hellebrand, S., 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*, Vol. 44, No. 2, pp. 223-233, Feb. 1995.

[Hellebrand 95b] Hellebrand, S., B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern Generation for a Deterministic BIST Scheme," *Proc. of International Conference on Computer-Aided Design (ICCAD)*, pp. 88-94, 1995.

[Koenemann 91] Koenemann, B., "LFSR-Coded Test Patterns for Scan Designs," *Proc. of European Test Conference*, pp. 237-242, 1991.

[Muradali 90] Muradali, F., V.K. Agarwal, B. Nadeau-Dostie, "A New Procedure for Weighted Random Built-In Self-Test," *Proc. of International Test Conference*, pp. 660-668, 1990.

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

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

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

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

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

[Wunderlich 90] Wunderlich, H.-J., "Multiple Distributions for Biased Random Test Patterns," *IEEE Transactions on Computer-Aided Design*, Vol. 9, No. 6, pp. 584-593, Jun. 1990.

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