A Seed Selection Procedure for LFSR-Based Random Pattern Generators

Kenichi ICHINO, Ko-ichi WATANABE, Masayuki ARAI,
Satoshi FUKUMOTO and Kazuhiko IWASAKI
Graduate School of Engineering, Tokyo Metropolitan University, Hachioji, Tokyo 192-0397, Japan
e-mail: ichino@info.eei.metro-u.ac.jp

Abstract- We propose a technique of selecting seeds for the LFSR-based test pattern generators that are used in VLSI BISTs. By setting the computed seed as an initial value, target fault coverage, for example 100%, can be accomplished with minimum test length. We can also maximize fault coverage for a given test length. Our method can be used for both test-per-clock and test-per-scan BISTs. The procedure is based on vector representations over $GF(2^m)$, where m is the number of LFSR stages. The results indicate that test lengths derived through selected seeds are about sixty percent shorter than those derived by conventionally selected seeds for a given fault coverage. We also show that seeds obtained through this technique accomplish higher fault coverage than the conventional selection procedure. In terms of the c7552 benchmark, taking a test-per-scan architecture with a 20-bit LFSR as an example, the number of undetected faults can be decreased from 304 to 227 for 10,000 LFSR patterns using our proposed technique.

1. Introduction

The cost of testing VLSIs has been increasing with the rapid advances in semiconductor technology [1]. With increasing VLSI clock frequencies and scales, we must use more expensive testers over longer periods of time. Designs that provide testability such as scan design and built-in self-tests (BISTs), have been studied to reduce the amount of time we have to use these expensive testers. In terms of BISTs, a device under test integrates both a test pattern generator (TPG) and a test response compactor (TRC) [2]. As a result, we may be able to reduce the cost of including a tester [3-5].

A linear feedback shift register (LFSR) is generally used for a TPG, and a multiple input signature register (MISR) is used for a TRC. A pseudo random pattern generator (PRPG) using an LFSR internally inputs test vectors to the circuit under test (CUT). However, to achieve high single stuck-at fault coverage, such as 95%, requires a long time for testing. Many studies have been done and these have reported on ways to overcome this difficulty.

Weighted pseudo random testing puts test vectors into the CUT, where each bit of the vectors has weighted probabilities of “0” or “1”, so that it can detect more faults than the original pattern [6-7]. Another technique to improve testability and observability is to insert control and observation points [8-9]. Inserting test points however, changes the CUT structure resulting in signal delay in the test. Dostie et al. also proposed a scan BIST structure to enhance CUT testability [10].

Although these techniques are useful, it is quite difficult for us to accomplish high fault coverage with limited test length. Consequently, mixed mode BISTs have been proposed, combining pseudo-random tests along with testcube information [3, 11, 12]. Reseeding of LFSRs has also been proposed. This can utilize encoded testcubes because the number of specified bits in a testcube is generally smaller than the length of the scan chain. Using this method, we require an LFSR with (s + 20) bits to encode a testcube for s specified bits with a probability of $1 - 10^{-6}$. The MP-LFSR decodes testcubes using multiple polynomials [13]. This method requires us to use an (s + 4)-bit MP-LFSR to encode a testcube with s specified bits. Bit flipping BISTs [14] and bit-fix BISTs [15] find a test pattern from the LFSR that is similar to the testcubes. Then, the test vector from the LFSR is flipped or has some bits fixed to it to match the testcube. These techniques require additional circuits. For reseeding BISTs, bit-flipping BISTs and bit-fix BISTs, many faults can easily be detected using pseudo-random tests with, for example, 10,000 LFSR patterns. Then, random-pattern-resistant faults are detected by the reseeded, bit-flipped or bit-fixed vectors. Additional seeds can be added by an external tester or an internal read only memory (ROM) and additional flipping or fixing functions can be implemented in an internal programmable logic array. Consequently, test costs increase when many random-pattern-resistant faults remain, because we require a longer test application time or a large number of additional circuits. It is quite important to reduce the number of undetected faults, after pseudo-random testing. One possible way to do this is to select the “seed” very carefully. Several procedures to select seeds have already been studied [16-21]. Bayraktaroglu et al. examined the PRPG structure and selection approaches [16]. Lempel et al. proposed an LFSR seed-selecting algorithm that used the theory of discrete logarithms [17]. The technique Fagot et al. used [18] estimates test quality by using the Hamming distance between the testcube and the test pattern that is output from the LFSR for a bit-flipping BIST. In Fagot et al.’s later study [19], fault simulation computes an efficient LFSR seed which outputs the test sequence including a testcube. To reduce test application time, Stroele et al. used a reseeding method and reverse order simulation [20]. Basturkmen et al. proposed a BIST based on the Markov source [21] and it can achieve high fault coverage.

In this work, we propose a procedure that selects the LFSR seed to improve random-pattern-test quality. The technique is based on vector representation over $GF(2^m)$. We also propose a technique of selecting a seed that can achieve target fault coverage with minimum test length for a given polynomial. We propose an additional technique of seed selection that can achieve maximum fault coverage for a given test length and for a given polynomial. In other words, the number of undetected faults is reduced to a minimum. Although the above techniques require prior off-line calculation, they considerably improve pseudo-random testing without the need to change circuits in the CUT. When these techniques are applied to a mixed-mode BIST such as reseeding, bit-flipping or bit-fixing, the amount of additional circuits is reduced. The proposed seed-selection techniques are able to be applied not only to typical LFSRs but also to other types of PRPGs such as CA based PRNGs [22], GLFSRs [23] and PRPGs with phase shifters [24].

2. Definitions

A PRPG based on an LFSR generates a test pattern sequence after setting a seed. We first applied the method to the test-per-clock BIST structure in Fig. 1, where the CUT is a combinational circuit. In a later section, we will consider test-per-scan BISTs. Once an LFSR seed is established, then the succeeding output test sequence is uniquely determined. In this manuscript, we consider single stuck-at faults.

The LFSR in Fig. 1 is based on an m-bit primitive polynomial and it outputs (2^m-1) different patterns to the CUT. A binary vector from the m-stage LFSR is regarded as an element α^i ($0 \le i \le 2^m-2$) from $GF(2^m)$, where α is a primitive element from $GF(2^m)$ and the i denotes an index.

If the LFSR outputs α^0 as the first test vector, then it outputs α^1, α^2, α^3, to α^{2m-2}. For seed α^1, the LFSR first outputs α^1, then outputs α^2, α^3, to α^{2m-2}, α^0. For seed α^i, the j-th test pattern is denoted by α^{i+j-1}.

Let F denote a set of stuck-at faults in the CUT. Let $|\cdot|$ represent the number of elements in a set. Then a fault coverage of $C\%$ by a test pattern sequence means that $|F|\times C/100$ faults are detected by this sequence. Moreover, let F_α denote the set of faults that are detected by test pattern α.

Function $L(\alpha^i, n_{fix})$ returns the test length whereby n_{fix} faults are detected for seed α^i, where n_{fix} denotes the target number of detected faults. The test pattern sequence can then be represented by $\alpha^i, \alpha^{i+1}, \alpha^{i+2}, \cdots, \alpha^{i+L(\alpha^i, n_{fix})-1}$. Although $n_{fix} = |F|$ is desirable, it is often realistic to set a slightly smaller value than $|F|$ for n_{fix}.

When a sequence of test patterns is represented by $\alpha^i, \alpha^{i-1}, \alpha^{i-2}, \cdots$ for seed α^i, we call it a reverse order pattern. Function $L_R(\alpha^{i+l}, n_{fix})$ returns the test length where n_{fix} faults are detected for the reverse order patterns from seed α^{i+l}. Thus, the sequence can be represented by $\alpha^i, \alpha^{i-1}, \alpha^{i-2}, \cdots, \alpha^{i-L_R(\alpha^i, n_{fix})+1}$.

Functions $L(\alpha^i, n_{fix})$ and $L_R(\alpha^{i+l}, n_{fix})$ can be computed by fault simulations. For a given CUT, n_{fix}, and LFSR, there is at least one seed that detects n_{fix} faults with a minimum test length. We call this the minimum test length seed α^{i*}, and its index is termed $i*$, that is,

$$i^* = \arg\min_i \{L(\alpha^i, n_{fix})\}. \tag{1}$$

3. LFSR Seed and Output Test Pattern Sequence

In this section, we assume that the test-per-clock BIST has been used in the s386 circuit, which is one of the ISCAS'89 benchmark circuits. This circuit has 13 inputs. Figure 2 shows test lengths where 100% fault coverage is achieved as a function of each seed index. The number of detectable faults for the s386 is 384. For example, $L(\alpha^0, 384) =$ 1783. The relationship between seeds and test lengths when using LFSR test-per-clock testing is in Fig. 2. That is to say, when an index i is increased to $(i+1)$, the test length $L(\alpha^{i+1}, n_{fix})$ becomes

(a) $L(\alpha^{i+1}, n_{fix}) = L(\alpha^i, n_{fix}) - 1$, or

(b) $L(\alpha^{i+1}, n_{fix}) \ge L(\alpha^i, n_{fix})$.

Figure 3 is a simplified example of Fig. 2. For simplicity, let the number of detectable faults be 100 for Fig. 3. The test length for seed α^0 is 8 as Fig. 3 shows, that is, $L(\alpha^0, 100) = 8$.

Suppose that the set of faults detected by 7 patterns from α^1 to α^7 include all the faults detected by α^0, that is

$$F_{\alpha^0} \subseteq F_{\alpha^1} \cup F_{\alpha^2} \cup \cdots \cup F_{\alpha^7} = F, \tag{2}$$

then $L(\alpha^1, 100) = 7$. Similarly, the set of faults detected by 6 patterns from α^2 to α^7 includes all the faults detected by α^1, that is

$$F_{\alpha^1} \subseteq F_{\alpha^2} \cup F_{\alpha^3} \cup \cdots \cup F_{\alpha^7} = F, \tag{3}$$

then, $L(\alpha^2, 100) = 6$.

Suppose the set of faults detected by 4 patterns from α^4 to α^7 does not include all the faults detected by α^3, that is,

$$F_{\alpha^3} \not\subset F_{\alpha^4} \cup F_{\alpha^5} \cup \cdots \cup F_{\alpha^7}, \tag{4}$$

then the test length for seed α^4, $L(\alpha^4, 100)$, gets longer. In the example in Fig. 3, the test length for α^4 is 10.

When $L(\alpha^i, n_{fix}) < L(\alpha^{i+1}, n_{fix})$ holds, we call $L(\alpha^i, n_{fix})$ the minimal test length l_{min} and call $\alpha^{i_{min}}$ the minimal test-length seed.

The minimum test length $l*$ is the minimum value of all the minimal test lengths. In Fig. 3, seed α^3 is the minimal test-length seed and the corresponding minimal test length is 5.

In Fig. 2, there are 13 minimal test lengths, where the minimal test lengths from the left are 1648, 2104, 1298, 1466, 2505, 2211, 2823, 2283, 2497, 1866, 1615, 2000, and 1867. Thus, the minimum test length $l*$ is 1298.

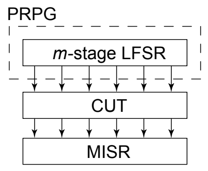

Fig. 1 Test-per-clock BIST architecture using LFSR.

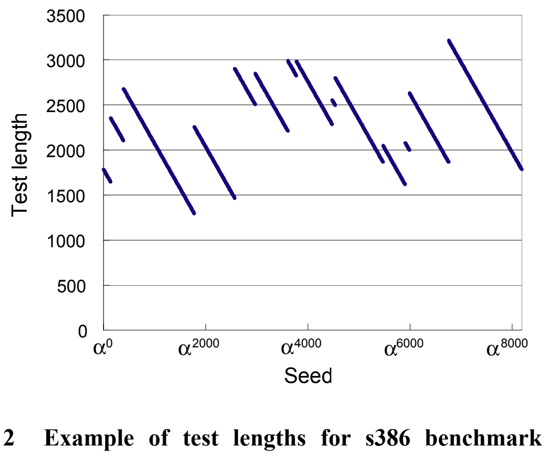

Fig. 2 Example of test lengths for s386 benchmark using test-per-clock LFSR (polynomial: 20033)

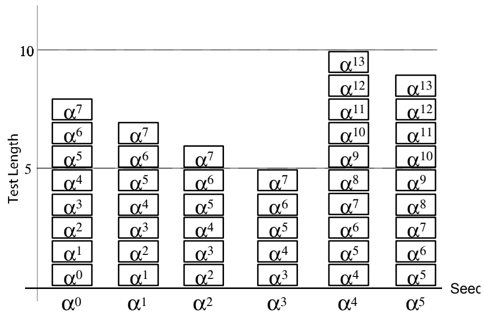

Fig. 3 Illustration of relationship between seeds and test lengths.

4. Seed-Selection Algorithm for Test-per-clock BIST

We will describe the algorithm used to select the minimum test-length seed by using an example. The algorithm consists of two steps. "Step A" was used to find the minimal test length and "Step B" was used to find if there was a shorter minimal test length.

4.1. Step A

For current index i, we need to find the "next" index of minimal test length seed i_{min} that is greater than i. First, calculate test length l for seed α^i using $L(\alpha^i, n_{fix})$. Next, apply the reverse order pattern [20] and calculate $L_R(\alpha^{i+l-1}, n_{fix})$. The value obtained is the minimal test length l_{min}, because the following equations hold:

$$F_{a^i} \cup F_{a^{i+1}} \cup \cdots \cup F_{a^{i+l-1}} = F_{a^{i+l-l_{min}}} \cup F_{a^{i+l-l_{min}+1}} \cup \cdots \cup F_{a^{i+l-1}} \quad (5)$$

and

$$F_{a^{i+l-l_{min}}} \not\subset F_{a^{i+l-l_{min}+1}} \cup F_{a^{i+l-l_{min}+2}} \cup \cdots \cup F_{a^{i+l-1}} . \quad (6)$$

The index i_{min} that attains the minimal test length l_{min} is

$$i_{min} = i + l - l_{min} . \quad (7)$$

Figure 4 shows an example for Step A. First, we calculate the test length for seed α^0, that is, $L(\alpha^0, n_{fix})$ in Step-A-1. In this example, $L(\alpha^0, n_{fix})$ = 8. Next, we calculate $L_R(\alpha^7, n_{fix})$ and the value of 5 we obtain is the minimal test length in Step-A-2. In this example, the index i_{min} is calculated as $i_{min} = 0 + 8 - 5 = 3$ in Step-A-3. In short, seed α^3 achieves the target fault coverage with a minimal test length of 5.

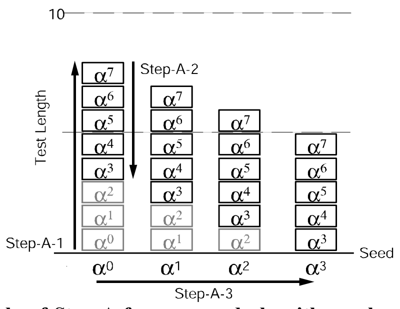

Fig. 4 Example of Step A for proposed algorithm, where minimal test length found.

4.2. Step-B

In Step-B, we try to find a minimal test length shorter than that found through the seed detected by Step-A.

The test length for the seed of index $i_{min} + 1$ is longer than l_{min}. First, we set the current index $i = i_{min} + 1$, and calculate the test length l for index i using $L(\alpha^i, n_{fix})$ in Step-B-1. Next, we calculate the index whereby it is possible to achieve target fault coverage using the shorter test length in Step-B-2. In Step-B-2, we use the properties of (a), and (b) in the previous section.

This is because the following expressions hold:

$$L(\alpha^i, n_{fix}),\ L(\alpha^{i+1}, n_{fix}),\ \cdots,\ L(\alpha^{i+l-l_{min}-1}, n_{fix}) > l_{min}, \quad (8)$$

and

$$L(\alpha^{i+l-l_{min}}, n_{fix}) \geq l_{min}. \quad (9)$$

This equation indicates that the seeds for the index between i and $i + l - l_{min}$ are not candidates for minimal test lengths that are shorter than the l_{min} calculated in the previous Step-A. Thus, the current index i is updated as

$$i \leftarrow i + l - l_{min} + 1 . \quad (10)$$

Then, calculate the test length l for the new seed α^i using $L(\alpha^i, n_{fix})$ in Step-B-3.

If the test length l is shorter than l_{min}, then the minimal test length can be calculated using Step-A. Otherwise, try to find the next possible index that has a shorter test length by repeatedly applying Step-B-2.

The above calculation is continued while index i is shorter than $2^m - 1$.

Figures 5 and 6 have examples of Step-B. In both figures, seed α^3 is a minimal test length seed, that is i_{min} = 3, and its test length l_{min} = 5.

In Fig. 5, the test length for seed α^4 is calculated using $L(\alpha^4, n_{fix})$ at step-B-1. In this figure, $L(\alpha^4, n_{fix})$ = 10. As the figure shows, when the minimal test length is 5,

$$L(\alpha^4, n_{fix}) > \cdots > L(\alpha^9, n_{fix}) \geq 5 . \quad (11)$$

Therefore seeds from α^4 to α^9 are definitely not candidates for minimum test length. Seeds that may accomplish shorter test lengths than 5 are calculated at step-B-2. Here, the seed is α^{10}. The test length l for seed α^{10} is calculated by $L(\alpha^{10}, n_{fix})$. In the figure, the test length l is 4. If we use a seed whose index is more than 10, there is a possibility that the test length will be shorter than 4. In this case, Step-A is applied.

In Fig. 6, Step-B-1 and Step-B-2 work the same as in Fig. 5. That is, $L(\alpha^4, n_{fix})$ = 10. Then, the candidate seed is α^{10}. The test length for seed α^{10} is calculated using $L(\alpha^{10}, n_{fix})$. In Fig. 6, $L(\alpha^{10}, n_{fix})$ = 7. In this case, Step-B-2 is repeated until the next possible seed that can attain a shorter test length is calculated.

Since the number of inputs for the s386 benchmark is 13, an exhaustive search for the minimum test length is possible to compute for a given LFSR. Using this approach, the process time required was about 40 minutes on an AT-compatible PC (Pentium 3, 1 GHz clock with 256-MB memory). In contrast, the calculation time for the proposed method is about 4 seconds.

Table 1 shows test lengths for several possible seeds. The test length of a minimum test length seed is 1293. The test length for seed $00\cdots01$ is 1783. When the bits of seed are all 1, the test length is 2230. It is evident that our proposed method is effective in greatly reducing test length.

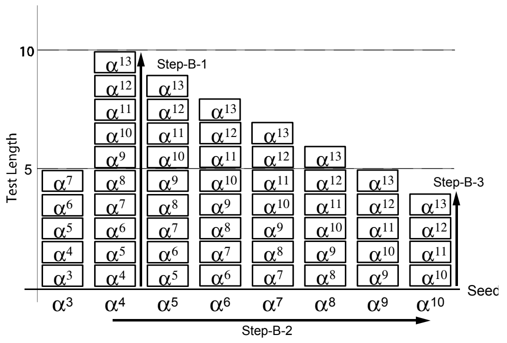

Fig. 5 Example of Step B, where new seed using Step-B-2, α^{10}, is better than current seed, α^4.

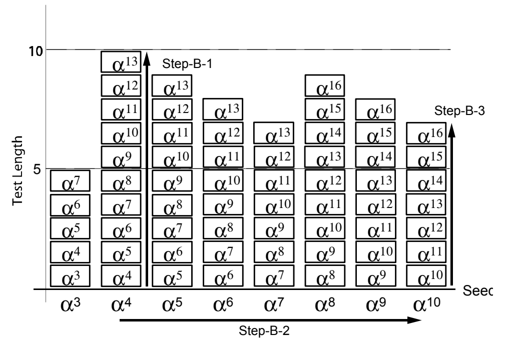

Fig. 6 Example of Step B, where new seed using Step-B-2, α^{10}, is worse than current seed, α^4.

TABLE 1 TEST LENGTH FOR SEVERAL POSSIBLE SEEDS.

Type of seed	Test length
Proposed method	1293
00⋯01	1783
All 1	2230
0101⋯	1971
1010⋯	2949

5. SEED-SELECTION ALGORITHMN FOR TEST-PER-SCAN BISTS

The test-per-scan BIST in Fig. 7 is said to be more practical. The test pattern from the PRPG changes depending on the seeds, similar to the test-per-clock architecture in Fig. 1.

Consider an m-bit LFSR and one k-bit scan chain. The PRPG outputs k bit test patterns $p_0, p_1, \cdots, p_{2^m-2}$.

Let β_j denote the seed that generates test pattern p_j. Seed β_j is expressed as follows:

$$\beta_j = \alpha^{(j\times k)\bmod(2^m-1)}. \qquad (12)$$

To put it differently, the seeds are expressed as $\beta_0 = \alpha^0$, $\beta_1 = \alpha^k$, $\beta_2 = \alpha^{2k}, \cdots$.

It is desirable for ($2^m - 1$) and k to be prime. Otherwise, we need to add dummy flip-flops to the scan chain to satisfy this condition. There is an example in Fig 8, where five seeds from β_0 to β_4 output a test pattern sequence whose test length is 10. The seed β_0 corresponds to output test pattern p_0, and then, test patterns $p_1, p_2, \cdots, p_9$ are generated. The seed β_1 corresponds to output test pattern p_1, and so on. We can obtain a seed selection procedure that yields minimum test length for test-per-scan testing by using the same technique as that used for test-per-clock testing. However, it is difficult for test-per-scan testing to accomplish high target fault coverage such as 99.99%. The scan BIST structure makes it much more practical to select a seed that can detect as many faults as possible for fixed test length, l_{fix}.

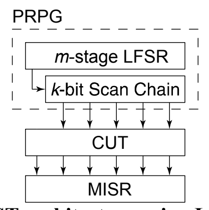

Fig. 7 Test-per-scan BIST architecture using LFSR

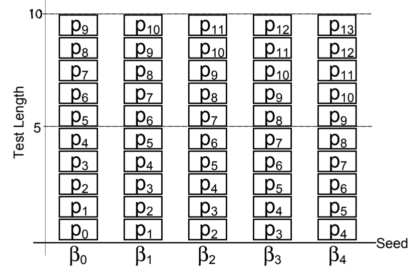

Fig. 8 Example of test patterns in test-per-scan BISTs.

Let us introduce function $N(\beta_j, l_{fix})$, which derives the number of detected faults for seed β_j and fixed test length l_{fix}. Thus, the best seed for test-per-scan testing, β_{j*} is expressed as:

$$\beta_{j*} = \arg\max_{\beta_j}\{N(\beta_j, l_{fix})\}. \qquad (13)$$

In the following, we term j as the seed number for β_j. Therefore, β_{j*} is the optimum seed number whose corresponding seed β_{j*} is the maximum number of detected faults with fixed test length. Function $N(\beta_j, l_{fix})$ can be computed by fault simulation the same as $L(\alpha^i, n_{fix})$ and $L_R(\alpha^{i+l}, n_{fix})$.

By slightly modifying the algorithm described in the previous section, the seed can be selected for test-per-scan BISTs. To put it another way, in this section the target number of detected faults is dynamically updated for the given fixed test length l_{fix}. This technique selects a seed that maximizes the number of detected faults with a limited length of test patterns.

The algorithm consists of Step-A*, which calculates the tentative target number of detected faults with test length l_{fix}, and Step-B*, which examines whether there is a seed that can detect more faults.

In Step-A*, the number of faults detected with seed β_j is first calculated for test length l_{fix} by means of function $N(\beta_j, l_{fix})$. We consider the number obtained, n_t, as the tentative target number of detected faults for the following steps. Next, the minimal test length l_{min} for n_t is calculated using $L_R(\beta_{j+l_{fix}-1}, n_t)$. The seed number j_{min} corresponding to the minimal test length is $j + l_{fix} - l_{\min}$. Step-A* is repeated, until the test length l of seed $\beta_{j_{\min}+1}$ is longer than l_{fix}.

When the test length l of seed $\beta_{j_{\min}+1}$ is longer than l_{fix}, we go to Step-B*. In Step-B*-1, the current seed number j is updated by $j+1$, and we calculate test length $l = L(\beta_j, n_t)$. In Step-B*-2 we update the current seed number j using $j + l - l_{fix} + 1$. In Step-B*-3, we calculate test length $l = L(\beta_j, n_t)$. When the test length l is longer than $l_{fix} - 1$, we return to Step-B*-2. If test length l is ($l_{fix} - 1$), we go to Step-A* and the tentative target number of detected faults is re-calculated.

For an m-bit LFSR, these steps are repeated until the seed number j reaches ($2^m - 2$). At the end of the algorithm, n_t is n*, the maximum number of faults detected with fixed test length l_{fix}. The corresponding seed number j is $j*$.

Figure 9 has an algorithm to select seeds that can detect the maximum number of faults through a PASCAL-like description.

```
procedure SEARCH_JSTAR
begin
  $j$ := 0;
  repeat
    repeat
      $n_t$ := $N(\beta_j, l_{fix})$;
      $l_{min}$ := $L_R(\beta_{i+L_{fix}-1}, n_t)$;
      $j$ := $j + l_{fix} - l_{min}$;
      $j_{min}$ := $j$;
      $j$ := $j+1$;
      $l$ := $L(\beta_j, n_t)$
    until( $l < l_{fix}$ and $j < 2^n - 1$ );
    repeat
      $j$ := $j + l - l_{fix} + 1$;
      $l$ := $L(\beta_j, n_t)$
    until( $l > l_{fix} - 1$ and $j < 2^n - 1$)
  until( $j < 2^n - 1$ );
  $n^* := n_t$;
  $j^* := j_{min}$
end.
```

Fig. 9 Algorithm for selecting seeds to detect maximum number of faults using fixed test length.

6. Experimental Results

The procedure in the previous section applied to benchmark circuits. The computer we used in this section is an IBM e-server pSeries 690 with single processor class nodes. We considered the BIST structure in Fig. 7, that is, a test-per-scan BIST using an LFSR and one scan chain. We used one hundred 16-bit and twenty 20-bit LFSRs because of computational requirements. These LFSRs were applied to the ISCAS'85 circuits and the combinational circuits of the ISCAS'89 benchmark. We used the 16-bit seed "0000000000000001" and the 20-bit seed "00000000000000000001" and these seeds were expressed as β_0.

6.1. Minimum Test Length for 100% fault coverage

First, we selected a seed that could achieve the minimum length to cover 100% of faults using a test-per-scan BIST with a 16-bit LFSR. Table 2 lists the results for some of the ISCAS'85 and ISCAS'89 circuits. We selected seeds using 100 primitive polynomials for each circuit. The first column has the circuit name, the next, "N_{ph}", is the number of polynomials out of 100 which could achieve 100% fault coverage. The third and fourth columns show the average and minimum test lengths using seed β_0 for N_{ph} polynomials. The fifth and sixth columns have the average and minimum test lengths using our proposed method for N_{ph} polynomials. The CPU time is in the seventh column.

For example, for the c432 benchmark, 100% fault coverage was achieved for all 100 polynomials. If we select β_0 as the seed, the average test length is 1,056, and the minimum test length is 338. Using the technique we propose, the average test length is 260, and the minimum test length is 193. The CPU time for one polynomial was 31 seconds.

Similarly, for the c880 benchmark, 100% fault coverage can be achieved for 94 polynomials. If we select β_0 as the seed, the average test length is 9,992, and the minimum test length is 508 for these 94 polynomials. Using the method we propose, the average test length is 3,841, and the minimum test length is 301. The CPU time for one polynomial is 51 seconds. There was no polynomial that attained 100 % fault coverage for the c2670 and c7552 benchmarks.

We could reduce the test lengths to 60% compared with those using seed β_0 by applying our procedure, except for the c2670 and c7552 benchmarks.

TABLE 2 Average and Minimum Test Lengths for Iscas'85 / '89 (100 16-bit Primitive Polynomials)

	N_{ph}	Seed β_0		proposed method		CPU time/ polynomial
		avrg.	min.	avrg.	min.	
c432	100	1056	338	260	193	31
c499	100	1041	508	400	301	38
c880	94	9992	3388	3841	2041	51
c1355	100	2612	1485	1157	901	47
c1908	97	10350	4332	4336	2892	50
c2670	0	-	-	-	-	-
c3540	85	26437	11143	9734	4316	67
c5315	94	3442	1576	1316	983	196
c6288	100	129	55	45	39	272
c7552	0	-	-	-	-	-
s444	100	536	136	141	94	25
s526	91	15863	4707	6427	3082	23
s820	92	20779	10386	11641	6960	27
s832	92	20594	10386	11250	6960	28
s953	68	34314	12261	19267	9076	62

6.2. Maximum Number of Faults Detected by the Selected Seeds

One hundred percent fault coverage cannot be achieved with a hundred 16-bit primitive polynomials for the c2670 and c7552 benchmarks. Consequently, we tried to detect as many faults as possible by selecting the seeds for c7552. In other words, we selected seeds and polynomials so that the number of undetected faults was minimized, where

$$undetected\,faults = detectable\,faults - detected\,faults. \quad (14)$$

Table 3 lists the results for the c7552 benchmark with a hundred 16-bit primitive polynomials. The first column is the fixed test length l_{fix}. The second and third columns show the average and the minimum number of undetected faults using seed β_0. The fourth and fifth columns have the average and minimum number of undetected faults using the proposed method. The sixth column shows the CPU time for one polynomial.

For l_{fix} = 1,000 and seed β_0 the average number of undetected faults is 461, and the minimum is 404. In contrast, the average and minimum number of undetected faults using the proposed technique are 388 and 345, respectively. As shown in Table 3, the number of undetected faults is reduced for l_{fix} = 1,000. The average CPU time for one polynomial is 472 seconds.

For seed β_0 with 10,000 patterns, the minimum number of undetected faults is 238 for a hundred 16-bit primitive polynomials. Using our seed technique, the minimum is 234. In this case, we could hardly find any improvement, because the fixed test length was too long for the length of the LFSR sequence.

We selected seeds for the c7552 benchmark with twenty 20-bit primitive polynomials. The results are in Table 4. For seed β_0 with 10,000 patterns, the minimum number of undetected faults is 270. Compared with 16-bit polynomials, the minimum number of undetected faults has increased. In this case, trying many polynomials such as 100 for seed β_0 is more effective than increasing the number of polynomial bits. Using our seed technique, the average is 243, and the minimum is 227 obtained through only 20 polynomials.

TABLE 3 AVERAGE AND MINIMUM NUMBER OF UNDETECTED FAULTS FOR C7552 BENCHMARCK WITH 100 16-BIT PRIMITIVE POLYNOMIALS.

test length	number of undetected faults				process time/ polynomials
	Seed β0		proposed method		
	avrg.	min.	avrg.	min.	
1000	461	404	388	345	about 700 (sec)
5000	352	297	300	261	
10000	310	238	268	234	

TABLE 4 AVERAGE AND MINIMUM NUMBER OF UNDETECTED FAULTS FOR C7552 BENCHMARCK WITH TWENTY 20-BIT PRIMITIVE POLYNOMIALS.

test length	number of undetected faults				process time/ polynomials
	Seed β0		proposed method		
	avrg.	min.	avrg.	min.	
1000	464	418	369	359	about 4300 (sec)
5000	351	319	277	259	
10000	304	270	243	227	

7. CONCLUSION

In this paper, we proposed a technique for selecting the LFSR seeds used in VLSI BISTs, for a given primitive polynomial, to improve test quality, such as, decreasing test length and increasing the number of detected faults. First, we presented an algorithm for a test-per-clock BIST from which a seed was derived that has a minimum test length to cover the target fault. Next, we presented an algorithm for a test-per-scan BIST from which a seed was derived that detected the maximum number of faults with a fixed test length.

We applied the techniques to the ISCAS'85 benchmark circuits and combinational parts of the ISCAS'89 benchmark circuits. The experimental results, using the first algorithm obtained a test length that provided 100% fault coverage. This reduced test lengths down to 60% for seed $\beta_0 = 00\cdots01$.

We applied the second algorithm to the c7552 benchmark circuit using a 20-bit LFSR. For a test length of 10,000 LFSR patterns, the minimum number of undetected faults was 227, while that for seed β_0 was 270. This means that 43 more faults could be detected by only changing the seed.

Although this method requires off-line calculation, the test-application time can be reduced for a given fault coverage, or the number of undetected faults can be reduced without the need for additional hardware. The fewer undetected faults remaining, the fewer additional circuits required. Thus, the proposed seed selection algorithms can be applied not only to conventional BISTs but also mixed-mode BISTs such as those with reseeding, bit-flipping, and bit-fixing.

REFERENCES

[1] *The Semiconductor Industry Association, International Technology Roadmap For Semiconductors*, 1999 Edition, 1999.
[2] P. Bardel, W. H. McAnney, and J. Savir, *Built-in test for VLSI*, John Wiley and Sons, 1987.
[3] G. Kiefer, H. Vranken, E. J. Marinissen, and H-J. Wunderlich, "Application of Deterministic Logic BIST on Industrial Circuits," Proc. Int'l Test Conf., pp. 105-114, 2000.
[4] L. Y. Ungar, and T. Ungar, "Economics of Built-in Self-Test," IEEE Design and Test of Computers, Vol. 18, No. 5, pp. 70-79, 2001.
[5] G. Hetherington, T. Fryars, N. Tamarapalli, M. Kassab, A. Hassan, and J. Rajski, "Logic BIST for Large Industrial Designs: Real Issues and Case Studies," Proc. Int'l. Test Conf., pp. 283-291, 2000.
[6] J. Savir, "On Chip Weighted Random Patterns," Proc. Asian Test Symp., pp. 344-352, Nov. 1997.
[7] R. Kapur, S. Patil, T. J. Snethen, and T. W. Williams, "Design of an Efficient Weighted Random Pattern Generation System," Proc. Int'l. Test Conf., pp. 491-500, 1994.
[8] N. A. Touba, and E. J. McCluskey, "Test Point Insertion Based on Path Tracing," Proc. 14th VLSI Test Symp., pp. 2-8, 1996.
[9] M. Nakano, S. Kobayashi, K. Hatayama, K. Iijima, and S. Terada, "Low-overhead Test Point Insertion for Scan-Based BIST," Proc. Int'l Test Conf., pp. 348-357, Sep. 1999.
[10] B. N. Dostie, D. Burek and A. Hassan, "Scan-BIST: A Multifrequency Scan-Based BIST Method," IEEE Design & Test of Computers., Vol. 11, No. 1, pp. 7-17, 1994.
[11] G. Kiefer and H-J. Wunderlich, "Deterministic BIST with Scan Chains," Proc. Int'l Test Conf., pp. 1057-1064, 1998.
[12] D. Das and N. A. Touba, "Reducing Test Data Volume Using External/LBIST Hybrid Test Patterns," Proc. Int'l Test Conf., pp. 115-122, 2000.
[13] S. Hellebrand, S. Tarnick, J. Rajski and B. Courtois, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," Proc. Int'l. Test Conf., pp. 120-129, 1992.
[14] H. J. Wunderlich, and G. Kiefer, "Bit-Flipping BIST," Proc. Int'l Conf. on Comput.-Aided Design, pp. 337-343, 1996.
[15] N. A, Touba, and E. J. McCluskey, "Altering a Pseudo-Random Bit Sequence for Scan-Based BIST," Proc. Int'l. Test Conf., pp. 167-175, 1996.
[16] I. Bayraktaroglu, K. Udawatta, and A. Orayloglu, "An Examination of PRPG Selection Approaches for Large, Industrial Design," Proc. Asia Test Symp., pp. 440-444, 1998.
[17] M. Lempel, S. K. Gupta, and M. A. Breuer, "Test Embedding with Discrete Logarithms," IEEE Transactions on Comput.-Aided Design, Vol. 14, No. 5, pp. 554-566, 1995.
[18] C. Fagot, O. Gascuel, P. Girard, and C. Landrault, "A Ring Architecture Strategy for BIST Test Pattern Generation," Proc. Asia Test Symp., pp. 418-423, 1998.
[19] C. Fagot, O. Gascuel, P. Girard and C. Landrault, "On Caluclating Efficient LFSR Seeds for Built-In Self Test," Proc. European Test Conf., pp. 4-14, 1999.
[20] A. P. Stroele, and F. Mayer, "Methods to Reduce Test Application Time for Accumulator-Based Self-Test," Proc. 15th VLSI Test Symp., pp. 48-53, 1997.
[21] N. Z. Basturkmen, S. M. Reddy, and I. Pomeranz, "Pseudo Random Patterns Using Markov Sources for Scan BIST," Proc. Int'l. Test Conf., pp. 1013-1021, 2002.
[22] P. D. Hortensius, R. D. Mcleod, W. Pries, D. M. Miller, and H. C. Card, "Cellular Automata-Based Pseudorandom Number Generators for Built-In Self-Test," IEEE Transactions on Comput.-Aided Design, Vol. 8, No. 8, pp. 842-859, 1989.
[23] D. K. Pradhan, and M. Chatterjee, "GLFSR – A New Test Pattern Generator for Built-In-Self-Test," Proc. Int'l. Test Conf., pp. 481-490, 1994.
[24] J. Rajski, N. Tamarapalli, and J. Tyszer, "Automated Synthesis of Large Phase Shifters for Built-In Self-Test," Proc. Int'l. Test Conf., pp. 1047-1056, 1998.