An Efficient PRPG Strategy by Utilizing Essential Faults

Li-Ren Huang, *Jing-Yang Jou, and Sy-Yen Kuo

Department of Electrical Engineering
National Taiwan University
Taipei, Taiwan, R.O.C.

Abstract

One major drawback of the LFSR-based BIST is its low fault coverage. To obtain the complete fault coverage, multiple seeds and multiple polynomials are usually required. One way to find the seeds and polynomials for the LFSR was utilizing the Gauss-elimination procedure. In this approach, the test patterns which are generated by LFSR are modeled as a set of multi-variable linear equations. It is created from a given deterministic test set. The corresponding seed and polynomial are then obtained from the solution of this equations set. However, given the original deterministic test set without don't cares, it were not acceptable on the random pattern resistant circuits. In this paper‡, we allow the test patterns to have don't care values. With an intelligent heuristic of further utilizing the essential faults, this approach becomes much more efficient even for the random pattern resistant circuits. The experimental results on the ISCAS_85 and the ISCAS_89 benchmarks show that a significant improvement can be obtained both on the hardware overhead and the test length.

1 Introduction

With ever increasing chip density, built-in self-test (BIST) has become one of the preferred test methods. The linear feedback shift register (LFSR) based BISTs in generating pseudorandom pattern are very attractive due to their low hardware costs [1]. But there exist random pattern resistant circuits which cannot be detected within a reasonable amount of time [2]. Furthermore, to achieve complete fault coverage, it will usually lead to large hardware overhead. The test application time and the hardware overhead thus become the two critical problems for pseudorandom testing to achieve high fault coverage.

To deal with these problems, weighted random methods [2, 3] which are based on the numerical optimization [3] or the pre-analysis of deterministic tests [2] are proposed. Another multiple seed LFSR for uniform random testing is illustrated in [4]. It requires two separate clocks to achieve the reseeding operation. Hellebrand et al. [5] used a reseeding technique for multiple polynomial LFSRs in which the test vectors are encoded as the polynomial indexes and seeds.

A novel application of the Gauss-elimination procedure for the multiple seed and multiple polynomial LFSR was proposed in [9]. In this approach, the pseudorandom test patterns are modeled as a set of multivariable equations. Selection of seeds and creation of the equations set are guided by a deterministic test set. The corresponding polynomial are then found from this equations set by the Gauss-elimination procedure. However, this approach was not very successful on random pattern resistant circuits. It is because that all known values (0 and 1) are required in the original deterministic test set. So only restricted deterministic tests could be included in an equations set, and then also limit the fault efficiency of the obtained LFSR. In this paper, we will introduce an improvement on the strategy of pattern match in [9] by including the don't care values in the test patterns of the deterministic test set. More deterministic tests can thus be included in an equations set and the hardware overhead is far reduced. The experiments show that the memory overhead can be improved significantly by over 43% on average compared to [9].

2 The PRPG Modeling

The programmable multiple seed and multiple polynomial LFSR which was proposed in [9] is used in this paper. It is based on the general external_XOR LFSR model as shown in Figure 1. This architecture maintains the characteristics of the general LFSR model that each cell (framed by dash line) contains a shift register S as well as an XOR gate. Shift registers form the basis of LFSR and XOR gates are provided for each feedback links. For the programmable LFSR, additional AND gates and D flip-flops are used to set the multiple polynomials. The enable signal of each D flip-flop is connected to the instruction register and the setting operations are dynamically controlled by the test instructions. Each AND gate which passes the value of corresponding feedback is controlled by the content which is latched in the D flip-flop. In order to generate a random test cycle, an appropriate

‡This research was supported by the National Science Council, Taiwan, R.O.C., under Grants NSC84-2215-E-002-032

*Jing-Yang Jou is with the Department of Electronic Engineering, National Chiao Tung University, Hsin-Chu, Taiwan, R.O.C.

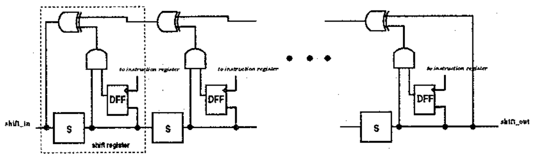

Figure 1: **The programmable LFSR model.**

seed-polynomial pair must be selected from memory and loaded into the LFSR. The selected polynomial is first shifted by the shift register and latched into the D flip-flop of each LFSR cell to establish the feedback links. The same shift operation is performed repeatedly to load the new seed. The test vectors will then be generated in a pre-deterministic cycle.

The hardware overhead of this approach consists of two parts. First part is the constant portion of every cell in the general LFSR model. It is proportional to the number of Primary Inputs (PIs) of the circuit under test, and is independent of the circuit as well as the test patterns. The other part is the size of the memory used for the storage of seeds and polynomials and is one of the key performance index for this approach. In order to achieve a highly efficient testing, we must make the memory size as small as possible to reduce the hardware overhead.

The test generation procedure for a general LFSR model can be illustrated as following. Test vectors are generated by loading the LFSR with pre-selected seed as the initial state. Then the LFSR will generate l pseudorandom patterns under the feedback link assignments by the selected polynomial. All bits except the most significant bit (MSB) of each test vector are shifted from the previous vector. The MSB is set by the XOR operations of the corresponding feedback bits of the previous vector. Under this model, we can represent the test vectors as a multi-variable linear equations set as following:

$$\begin{array}{llllll}
v_0^0x_0 & +\, v_1^0x_1 & +\, v_2^0x_2 & + \dots & +\, v_{n-1}^0x_{n-1} = & v_0^1 \\
v_0^1x_0 & +\, v_1^1x_1 & +\, v_2^1x_2 & + \dots & +\, v_{n-1}^1x_{n-1} = & v_0^2 \\
\vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\
v_0^{l-2}x_0 & +\, v_1^{l-2}x_1 & +\, v_2^{l-2}x_2 & + \dots & +\, v_{n-1}^{l-2}x_{n-1} = & v_0^{l-1}
\end{array}$$

Assume the LFSR is n bits width. Each test vector is represented by $v(i) := (v_0^i, v_1^i \dots, v_{n-1}^i)$. The variables $x_0, x_1, \dots x_{n-1}$ represent the corresponding feedback links of each bit and form the polynomial vector $p := (p_0, p_1 \dots, p_{n-1})$ in which $\forall i \in [0, n-1], p_i = x_i$. The solution of the above variables is exactly the specified feedback link assignments we want, the polynomial of LFSR.

3 The Seed-Polynomial Generation

The seed-polynomial generation algorithm proposed in this paper is basically based on that in [9]. A pattern is selected from a deterministic test set first to be the seed. Then the algorithm emulates the shift operation of LFSR and set the MSB to unknown temporarily. After each shift, the algorithm tries to find a match pattern from the deterministic test set. If it is found, all unknowns could be decided. This procedure is repeated until n patterns to be emulated. Then those patterns are modeled as a set of multi-variable linear equations set and solved by the Gauss-elimination procedure to find the polynomial. However, the algorithm in [9] is not efficient for the random pattern resistant circuits. Because it does not utilize the don't cares in the deterministic test set. Therefore, a large number of shifts may result before two match patterns can be found. The generated pseudorandom pattern test set can only inherit limited benefit from the deterministic test set. In this paper, we will allow don't cares to exist in the test patterns of the deterministic test set to obtain a more efficient pattern match algorithm.

In order to obtain a test efficient LFSR, we must find the appropriate seeds and the corresponding polynomials to generate a set of pseudorandom test patterns to achieve the goal of complete fault coverage. Therefore, two problems should be considered. One is to get a highly efficient test pattern set with complete fault coverage (the pattern emulation). The other one is to find the corresponding seeds and polynomials which can generate such a random pattern set (the seed-polynomial generation).

3.1 Pattern Emulation

In the following discussion, we will use '*known*' value to represent the deterministic boolean value 0 or 1, and '*don't care*' to represent the don't care value. It should be noted that the *unknown* (u) is different to the *don't care* (x). An *unknown* means that the value of this bit is uncertain yet. It could be 0, 1, or don't care. On the other hand, *don't care* means the value is unrestricted. We do not mind whether it is a 0 or 1.

Since the pattern emulation procedure is based on the shift and match operations, the match condition should be defined before we discuss the procedure in detail.

Definition 1 :
Two vectors are **compatible** *if they are identical bit by bit except the don't care or unknown values.*

For example, vector (uu10x0x) and vector (10100xx) are compatible.

Definition 2 :
The **match length** *of a vector is defined as the number of right-shift operations necessary to make it compatible to the next consecutive vector in the test set.*

For example, the match length of the vector (01001101) to vector (uuu01001) is 3. The benefit of including don't cares is that the match condition becomes looser and the match pattern is then easier to find. In other words, more don't cares appearing in the ATPG test set means a higher probability to find the match vectors. And more patterns in the deterministic test set can be embedded into the random vectors. As a result, higher fault coverage could be obtained. One way to have more don't cares in the original deterministic test set is to utilize the concept of *essential faults.* All the bits which are useless to the detection of essential faults can be reset to don't care. The *essential faults* is defined as below:

Definition 3 :
Given a fault list F of a circuit under test and a corresponding test set T, the **essential faults** *of a pattern t in T are those faults which can be detected by t only and cannot be detected by any other patterns in T.*

This heuristic has two major advantages. One is with respect to the pattern utility. The faults which can be detected by more than one pattern also have a higher probability being detected by random patterns. On the contrary, the essential faults can be treated as random pattern resistant faults. So the essential faults are detected by the embedded deterministic test patterns while the remaining faults can be detected by the emulated random vectors. The other one is with respect to the match efficiency. This approach significantly reduces the number of shifts to find a compatible pattern. It is because that don't cares are allowed and therefore a compatible pattern is easier to find. Obviously, much more deterministic test patterns will also be embedded in a random pattern emulation cycle.

The detail procedure of the pattern emulation is presented step by step in Figure 2. Two vector sets are used here. The set A represents the ATPG test set. The resulting pseudorandom vector set is represented by the set V. After selecting a vector with minimum match length from A as the initial state (seed) of the LFSR, we generate a new vector by a right shift and set its most significant bit to 'u'. Then we search an compatible vector from A as the next vector. If this vector is found, the matching is successful. We set all the 'u's in the emulated random vectors by it. It includes the setting of all the unknown values in v^k as well as the propagation of the settings to those unknown values in the previous random vectors in V. Otherwise, the matching fails and the shift as well as the search operation are then continued until the compatible vector is found.

It is well known that n linear independent vectors are necessary and sufficient to span an unique n-dimensional linear space. Thus, we emulate at most n random vectors to find the corresponding polynomial of LFSR. If the number of total random vectors in V does not exceed n, we continue the emulation cycle.

After we got n random vectors and set all unknowns, some don't care values may still existed in them. Since the Gauss-elimination procedure can not be applied on the don't care values, further settings of those don't care bits are required. Many different methods can be employed for this task. A simple weight-based

```
Procedure pattern-emulation

    $A$ := the ATPG test set all with known values
    $V$ := the emulating test set
    $a^k$ := a pattern in the ATPG test set
      ($a_i^k$ is the ith bit of $a^k$, $a_0^k$ is the most significant bit)
    $v^k$ := a pattern in the emulating random pattern set
      ($v_i^k$ is the ith bit of $v^k$, $v_0^k$ is the most significant bit)
$V \leftarrow \emptyset$;
do
  select a pattern $a^x$ from $A$ with the min. match length;
  $V \leftarrow V + a^x$;
  $A \leftarrow A - a^x$;
  $v^0 \leftarrow a^x$;
  $k \leftarrow 0$;
  while $|V| \leq n$
    $k \leftarrow k+1$;
    generate a pattern $v^k$ by right-shift a bit from $v^{k-1}$
      and set MSB to 'u';
    if a pattern $a^y$ from $A$ which is compatible to $v^k$ is found
      then set all 'u's of $v^k$ by $a^y$;
              propagate the settings into $V$;
      endif
      $V \leftarrow V + v^k$;
  endwhile
  set the remaining don't cares by bit_offset;
endwhile ! inconsistency_chk(V)
```

Figure 2: **The pattern emulation procedure with essential test set.**

method is adopted here. For each bit position, we calculate the number of ones and zeros for all the test patterns in the deterministic test set. Then we define a value called *offset* for each bit. It is the absolute value of the difference between the number of 1s and the number of 0s of each bit position for the deterministic test set. If the number of 1s (0s) is larger, 1 (0) *dominates* this bit. The 1 is chosen as the default dominate value when offset is 0. Following is an example.

```
x0xxxx1xx0
x0xxx0xxxx   1's count  : 1112002101
x0x1xxxxx0   0's count  : 0400120003
11110x11x1   offset     : 1312122102
x0xxx0xxx0   dominate   : 1011001110
```

The bit with the largest offset should be set first. The setting value is also propagated into the entire random pattern set. This operation is conducted until all the don't care values are set.

After we get sufficient number of random vectors, the first phase of the Gauss-elimination procedure is performed to check the existence of a solution for this random pattern set. The inconsistency on the random pattern set means that no polynomial exists for the LFSR to generate the specified random pattern sequence. In such a case, we give up this seed as well as the associated random vector set. The emulation process is restarted all over again by using another seed. The pattern emulation procedure is completed successfully if the resulting random vector set passes the inconsistency check. Then the resulting vector set is returned to the seed-polynomial generation procedure to find the corresponding polynomial.

3.2 Seed-Polynomial Generation

When the random vector emulation is completed without the inconsistency, we get a linear multi-variable equation set from those pseudo-random vectors. The solution of this equation set will be the polynomial of LFSR and the first vector will be the seed. In this subsection, the entire procedure from ATPG test set to the complete fault coverage LFSR implementation will be discussed.

Before we start the seed and polynomial generation procedure, a reset operation is first performed on the original deterministic test set to reduce the number of known values. It is called *Essential Raising* [8]. That is, we change the value of a bit in a deterministic test pattern from known value to don't care, if the essential fault detectability of this pattern is maintained. Otherwise, the original known value is retained. This operation is also performed at the end of each pseudorandom pattern emulation cycle. The reason why we use this reset operation instead of regenerating a new test set as the deterministic test set is to better utilize the existing original one. Since the original test set may be provided by the IC vendor instead of being generated by ourselves.

```
Procedure seed-polynomial generation
    A denotes the ATPG test set
    V denotes the random test set
Step1 : V=pattern-emulation();
Step2 : p=Gauss-elimination(V);
Step3 : Perform fault simulation to determine the test length;
Step4 : Drop the detected faults and the patterns with no contribution to fault coverage;
Step5 : Essential raising for A;
Step6 : Repeat step2 to step5 until the complete fault coverage is achieved;
```

After pattern emulation procedure returns the resultant random vector set which has no inconsistency (step1), the second phase of Gauss-elimination procedure is now performed to solve the set of equations to obtain the polynomial (step2). Then a set of pseudorandom patterns, for example 5,000 vectors, are generated. The number of patterns are called the *simulation length.* Fault simulation is then performed on the generated pseudorandom patterns to find the minimum test length (step3), which is the number of patterns applied before the fault coverage starts to increase very slowly. Next, we drop the detected faults from the fault list and remove the patterns which cannot detect any fault left in the fault list from A (step4). It should be noted that the detected faults are not only covered by the selected deterministic test patterns but also by the generated pseudorandom patterns. Finally, the essential raising operation is applied again to the deterministic test patterns to include more don't care values (step5). Using the obtained seed and polynomial, we can construct an LFSR to generate the same pseudorandom pattern sequence as V. Step 1 to 5 are repeated until the complete fault coverage is achieved (step6).

4 Experimental Results

We conducts entensive experiments on ISCAS_85 and ISCAS_89 benchmarks. The original deterministic test set for ISCAS_85 benchmarks used here is produced by PODEM algorithm [10] and additional compaction is applied on it to further reduce its test length [8]. The deterministic test set for ISCAS_89 benchmarks is provided by the COMPACTEST [7]. For comparison, we also apply the algorithm of [9] to the ISCAS_89 benchmarks to demonstrate the improvements of our approach. Our experimental results are collected in Table 1 and some advanced comparisons of these two approaches are listed in Table 2. For simplicity, we only list those circuits that are not very efficient in [9]. Table 1 lists the results of our new approach. For each circuit, the number of PIs and the corresponding number of tests in the deterministic test set are given first. Next column shows the number of seed-polynomial pairs used. The data format of this column is $p+k$. p represents the number of seed-polynomial pairs required for the programmable LFSR, and k is the number of the rest deterministic test vectors which cannot be generated by LFSR. They are those patterns whose match length is larger than n. Thus, one LFSR (one seed plus one polynomial) is required to generate each of such patterns, and it is worthless obviously. All of them (seeds, polynomials, and rest vectors) should be stored in the memory. Therefore, the exact number of the vectors required to be stored in memory becomes $2p+k$. Following is the ratio of memory hardware between our approach and the ATPG. This value is obtained by dividing the number of vectors required by the seed-polynomial pairs $(2p+k)$ in column 2 by the number of ATPG test vectors in column 3.

Memory overhead only considers the storage space for the test vectors. However, the physical hardware overhead for the programmable LFSR consists of not only the memory overhead but also the hardware for the LFSR itself. We use an approximate estimation to evaluate the number of gates required in the ROM implementation of ATPG and in [9]. We assume that it needs 4 gates for each memory cell. Under this assumption, the cost of each general programmable LFSR cell is 10 gates. There are 4 gates for each shift register cell S, 4 gates for each D flip-flop, one AND gate, and one XOR gate. A standard LFSR cell will cost 5 gates (4 gates for the shift register and one XOR gate). Therefore, the overall gate count of our approach can be evaluated as below:

$$\begin{cases} n \times (4 \times (p+k)+5), & if \quad p=1 \\ n \times (4 \times (2p+k)+10), & otherwise \end{cases}$$

where n denotes the number of PIs or the width of a pattern. It assumes that $p+k$ seed-polynomial pairs

Table 1: **Experimental results of our new approach.**

Circuit	PIs	AT PG	S-P pair	Mem. ratio	Gate count		GC ratio	% Imp.	Test length	Sim.
					ATPG	Our				
c2670	157	44	7+1	0.34	27632	10990	0.40	60.23%	7300	5K
c7552	206	73	9+1	0.26	60152	17716	0.29	70.55%	31282	10K
s526	24	55	2+1	0.09	5280	720	0.14	86.36%	3614	2K
s713	54	30	3	0.20	6480	1836	0.28	71.67%	2069	2K
s838	67	76	6+1	0.17	20368	4154	0.20	79.61%	17526	10K
s953	45	87	3	0.07	15660	1530	0.10	90.23%	7146	3K
s1196	32	138	3	0.04	17664	1088	0.06	93.84%	7991	3K
s5378	214	111	3	0.05	95016	7276	0.08	92.34%	8400	5K
s9234	247	150	11+1	0.15	148200	25194	0.17	83.00%	108638	15K
s13207	700	237	4+1	0.04	663600	32200	0.05	95.15%	43543	15K
s15850	611	123	7+1	0.12	300612	42770	0.14	85.77%	34121	10K
s38417	1664	95	12	0.25	632320	176384	0.28	72.11%	126615	15K
s38584	1464	125	3+1	0.05	732000	49771	0.07	93.20%	18333	10K
Average			11.43*	0.14	195018.86	26945.86	0.17	82.77%	31210.43	

Mem. ratio : Memory overhead ratio compared to ATPG.
S-P pair : Seed-polynomial pairs to achieve complete FC.
GC ratio : Gate count ratio compared to ATPG.
% Imp. : Percentage improvement for gate count.
Sim. : Simulation length for random pattern.

are required. When $p = 1$, one LFSR is sufficient. We can implement a standard LFSR (5 gates) and store the seed as well as the rest k vectors in memory $(4 \times (p + k))$. Otherwise, we need a programmable LFSR (10 gates) plus the stored seeds, polynomials, and the rest vectors $(4 \times (2p + k))$. A similar formula $n \times 4 \times m$ is used to compute the gate count for memory required by the ATPG approach, where m represents the number of patterns.

Column 6 and 7 show the gate counts required for the ATPG test set and ours respectively. The ratio of the gate counts is obtained by the same way as the memory ratio and are presented in column 8. Following is the percentage improvement for ours against the ATPG one. The pseudorandom test length and the simulation length are in the last two columns. The ratio of the storage space are between 0.04 (s13207) to 0.34 (c2670). The percentage improvement for the gate count is from 60.23% to 95.15%.

We make a comparison between these two cases in terms of the seed-polynomial pairs, the gate count, and the test length. The results are shown in Table 2. Column 2 to column 4 compare the number of required seeds and polynomials. It can be seen that at least 25% improvement and 43.54% improvements on average has been obtained. Also, we can get over 19% and 43.08% on average improvements in terms of the gate count as shown in column 7. The last three columns show the comparisons for the test length. Note that four of them show a negative improvement. It is because we apply a larger simulation length to further reduce the hardware overhead. However, 27.19% improvement on average can be obtained. Figure 3 shows the hardware cost of c7552 for different simulation lengths. The lines marked as '5k' and '10k' represent the case under the 5,000 and the case under 10,000 simulation length respectively. Several points with 3-tuple data shows the percentage hardware overhead, the test length, and the fault coverage respectively. It can be seen that although the complete fault coverage can be achieved by a shorter test length (21,426) under the case of the 5,000 simulation length. The corresponding hardware cost is higher (0.33). While the case of 10,000 simulation length achieves the same test performance by a longer test length (31,282) but lower hardware cost (0.26). The trade-off between the test length and the hardware cost depends on the requirements of the different applications.

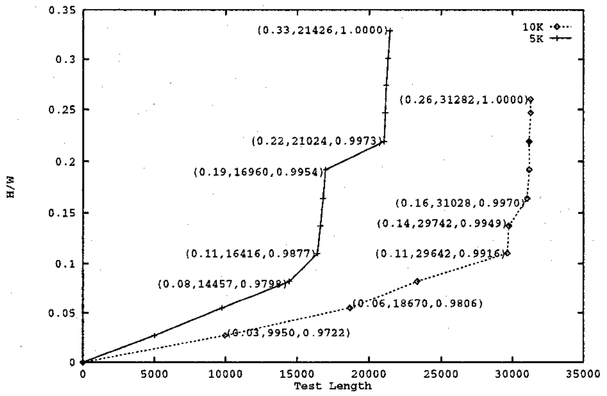

Figure 3: The hardware cost of c7552 for different simulation lengths.

Table 2: **Comparisons between [9] and our new approach.**

Circuit	S-P pair			Gate count			Test length		
	K.	E.	Imp.	K.	E.	Imp.	K.	E.	Imp.
c2670	15	7+1	50.00%	20410	10990	46.15%	61400	7300	88.11%
c7552	16	9+1	40.63%	28428	17716	37.68%	61800	31282	49.38%
s526	3+1	2+1	28.57%	912	720	21.05%	2574	3614	-40.40%
s713	3+2	3	25.00%	2268	1836	19.05%	4864	2069	57.46%
s838	8+9	6+1	48.00%	7370	4154	43.64%	10947	17526	-60.10%
s953	4+1	3	33.33%	2070	1530	26.09%	5036	7146	-41.90%
s1196	4	3	25.00%	7991	1088	86.38%	8722	7991	8.38%
s5378	6	3	50.00%	12412	7276	41.38%	18650	8400	54.96%
s9234	19+6	11+1	47.73%	45942	25194	45.16%	226835	108638	52.11%
s13207	6+1	4+1	30.77%	43400	32200	25.81%	83989	43543	48.16%
s15850	20	7+1	62.50%	103870	42770	58.82%	149357	34121	77.15%
s38417	33+6	12	66.67%	495872	176384	64.43%	321177	126615	60.58%
s38584	13+1	3+1	77.78%	172752	49776	71.19%	85012	18333	78.43%
Average	23.86*	11.43*	43.54%	67472.07	26945.86	43.08%	74495.50	31210.43	27.19%

K. : Known test set.
E. : Essential test set.
Imp. : Improvement.

5 Conclusions

A pseudorandom test pattern generation method based on the Gauss-elimination procedure was illustrated in [9]. The general programmable LFSR provides a multiple seed and multiple polynomial structure to perform the functions of both the deterministic test pattern embedding and the random pattern generation. But the requirement of known values in the deterministic test set limits its performance for some circuits, especially for the random pattern resistant circuits.

In this paper, the concept of the *essential fault* is utilized to introduce the *don't cares* in the deterministic test set. An intelligent heuristic is then proposed to obtain an efficient pattern emulation procedure and a seed-polynomial generation procedure. The experimental results show that a high performance testing has been achieved for all cases including the random pattern resistant ones. Over 60% improvement was obtained while compared with the ROM implementation of the ATPG test set in terms of hardware overhead. Also over 43% improvement on average was obtained compare to that of [9]'s approach.

The future work includes the optimization of the pattern emulation procedure to maximize the embedding of the ATPG test vectors in each emulation cycle. As shown in [6], employing appropriate ATPG test set or ATPG algorithm to generate the test set specifically for the proposed BIST architecture can further improve the results. Obviously, this area deserves further studies.

References

[1] P. Nagvajara, "Pseudorandom Testing for Boundary-Scan Design with Built-In Self-Test," *IEEE Design & Test of Computers*, pp. 58-65, Sep. 1991.

[2] I. Pomeranz et al., "3_Weight Pseudo-Random Test Generation Based on a Deterministic Test Set for Combinational and Sequential Circuits," *IEEE Trans. Computer-Aided-Design*, Vol. 12, No. 7, pp.1050-1058, July 1993.

[3] H. J. Wunderlich, "On Computing Optimized Input Probabilities for Random Tests," *Design Automation Conf.*, pp. 392-298, June 1987.

[4] J. Savir et al., "A Multiple Seed Linear Feedback Shift Register," *IEEE Trans. Computers*, pp.250-252, VOL. 41, NO. 2, Feb. 1992.

[5] S. Hellebrand et al., "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Trans. on Computer*, pp.223-233, VOL. 44, NO.2, Feb. 1995.

[6] S. Hellebrand el. al., "Pattern Generation for a Deterministic BIST Scheme," *Int. Conf. on Computer-Aided-Design*, pp. 88-94, Nov. 1995.

[7] I. Pomeranz et al., "COMPACTEST: A Method to Generate Compact Test Sets for Combinational Circuits," *IEEE Trans. Computer-Aided Design*, Vol. 12, No. 7, pp. 1040-1049, July 1993.

[8] Jau-Shien Chang and Chen-Shang Lin, "Test Set Compaction for Combinational Circuits," *First Asia Test Symposium Japan*, pp.20-25, 1992.

[9] L. R. Huang, S. Y. Kuo, and I. Y. Chen, "A Gauss-Elimination Based PRPG for Combinational Circuits," *Europe Design & Test Conf.*, pp. 212-216, Mar. 1995.

[10] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Computers*, Vol. C-30, No. 3, pp. 215-222, Mar. 1981.