A Gauss-Elimination Based PRPG for Combinational Circuits

Li-Ren Huang and Sy-Yen Kuo

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

Ing-Yi Chen

Department of Electronic Engineering
Chung Yuan Christian University
Chungli, Taiwan, R.O.C.

Abstract

A new algorithm for the reseeding of multiple polynomial LFSR for pseudorandom test pattern generation(PRPG) is proposed in this paper[‡]*. It is based on the Gauss-elimination procedure and the deterministic test set generated by an ATPG software system for combinational circuits. In addition to the general LFSR model, we also provide two further improvements,* ms1p *and* 1smp, *to minimize the hardware overhead. Experimental results were obtained on ISCAS_85 benchmark circuits to demonstrate the effectiveness of this methodology. Complete fault coverage is achieved in all circuits. Low hardware overhead is also maintained with a reasonable test length.*

1 Introduction

The BIST with pseudorandom patterns is very attractive due to their low hardware cost, and has been offered as a test solution for large scale VLSI circuits, such as microprocessors, MCMs. But there exist circuits that contain random pattern resistant faults and cannot be detected with a reasonable amount of time [2, 3]. Weighted random methods [1, 2, 3] which are based on the numerical optimization [1] or the pre-analysis of deterministic tests [2, 3] are proposed for the PRPG(pseudo-random pattern generation) and good results were obtained. A 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] and S. Venkataraman et al. [6] use a reseeding technique for multiple polynomial LFSRs in which the test vectors are encoded as the polynomial index and seeds.

There are some major problems on the PRPG with LFSR. One of the problems is the fault coverage. To achieve complete fault coverage will usually lead to large overhead and complex hardware as experienced in the early researches [3]. However, complete fault coverage is achieved in many recent works and has become an essential criteria for the pseudorandom test. The test application time and the hardware overhead are the other two critical problems.

The ATPG(Automatic Test Pattern Generation) algorithms [9] analyze the characteristics of a circuit and the potential faults under the fault model to derive the appropriate test set. Very high fault coverage and minimum test set are the major advantages of this method. But the implementation of ATPG algorithm always leads to a large area and is not applicable for large scale VLSI circuits.

In this paper, we start from a deterministic test set generated by ATPG software and propose a new application of the Gauss-elimination procedure for the multiple seed and multiple polynomial LFSR structure. The selection of seeds and polynomials is made by the Gauss-elimination process to solve the random pattern sequences as a set of multi-variable linear equations.

2 Reassignable LFSR

A general LFSR consists of shift registers and XOR-gates with feedback links of the corresponding bits. Based on the general LFSR structure, we propose a reassignable LFSR which employs multiple seeds and polynomials to perform the pseudorandom test pattern generation. This approach is shown in Fig. 1.

For each test generation cycle, the appropriate seed-polynomial pair must be selected from memory and be loaded into LFSR. The selected polynomial is then shifted by the LFSR 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 seed. Following this, the test vectors will be

[‡]This research was supported by National Science Council, R.O.C., under Grants NSC84-2215-E-002-032 and NSC83-0404-E-033-005

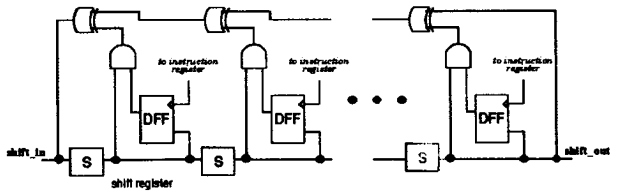

Figure 1: The reassignable LFSR model.

generated in a pre-determined cycle under this seed-polynomial pair.

The reseeding and multiple polynomial concept can be not only a 1-*to*-1 but also an *m-to*-1 relationship in our approach. The *m-to*-1 relationship means there are m seeds with only one polynomial or one seed with multiple polynomials. The advantage of *m-to*-1 approach is the space saving on the storage for both the seeds and the polynomials.

In general, we choose an appropriate seed-polynomial pair each time. However, we can just rescan a new seed only. At the next test generation cycle, the previous polynomial remains unchanged and the feedback link assignments are still determined by the contents of the D_latches. This is the case of m seeds with *one* polynomial(*ms1p*). It makes the implementation of LFSR from a general structure to a fixed, well designed architecture to reduce the hardware overhead. The required seeds can be stored in the local memory of the BIST structure or shifted from the ATE outside the circuit. An example *ms1p* LFSR is shown in Fig. 2.

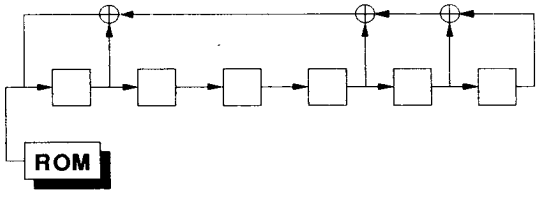

Figure 2: Example ms1p LFSR.

Otherwise, we can shift the new seed at the beginning of test. We only change the polynomial of the LFSR at the next test generation cycle. The last vector in the previous test generation cycle is then used as the seed of the current cycle. That is the case of *one* seed with m polynomials(*1smp*). A simple example is shown in Fig. 3.

More advantage is obtained due to the saving of shifting operations(for both seeds and polynomials). Furthermore, only the bits with different setting among test generation cycles are required, the widths of the stored polynomials can be significantly reduced.

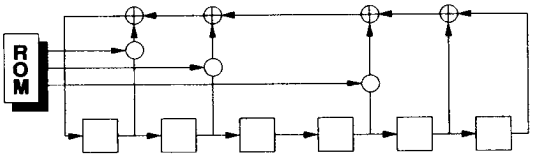

Figure 3: Example 1smp LFSR.

The hardware overhead of this approach can be evaluated in two parts. First part is the constant portion due to every cell in the general LFSR model. It is proportional to the number of PIs(primary inputs) of the circuit under test, and is independent of the circuit as well as the test patterns. The other part is the memory size for the storage of seeds and polynomials and is one of the main issues for this approach. We must restrict the memory size as small as possible to reduce the hardware overhead.

3 FLA : Feedback Link Assignment

The selection of seeds and polynomials aims at not only achieving complete fault coverage but also incurring minimum hardware overhead. A Gauss-elimination based algorithm for seed-polynomial selection is discussed here.

3.1 Gauss-Elimination

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 of each test vector are shifted from the previous vector. The most significant bit 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:

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

Assume n is the number of bits in the LFSR. Each test vector is represented by $v(i) := (v_0^i, v_1^i \ldots, v_{n-1}^i)$. The variables $x_0, x_1, \ldots x_{n-1}$ represent the corresponding feedback links of each bit and form the polynomial vector $p := (p_0, p_1 \ldots, p_{n-1})$ 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. The Gauss-elimination method [8] in linear equation system with some modifications to handle the boolean values and the module-2 add operations is appropriate for solving such multi-variable linear equations.

The complexity of this procedure while applying to an N-input circuit is $O(N^2)$ for the module-2 add operations. In the worst case, all m deterministic test vectors must be evaluated. The computing time becomes $m \times O(N^2)$. For ISCAS_85 benchmark circuits, the maximum number of deterministic tests used is 106(c1908) and the maximum number of input is 206(c7552). It can be seen that it is very efficient in the physical experiments and is comparable to the weight assignment or reseeding scheme in other existing approaches.

3.2 FLA Algorithm

A known deterministic test set that provides complete fault coverage of detectable faults is first used to guide the seed-polynomial selection procedure. Ideally, we want to generate a pseudorandom test set which has the same fault coverage as the deterministic test set. And then we solve these random vectors backward to find the specified feedback link assignments. The first one of these random vectors is chosen as the seed. The algorithm for feedback link assignments is described step by step in the following.

Procedure Feedback Link Assignments

$A :=$ *the ATPG test set*
$V :=$ *the random test set*

Step1 : *choose* $a \in A$, *which can detect the hard-to-detect faults and has the highest fault coverage*
$x \leftarrow 1; v^0 \leftarrow a;$

Step2 : *for all* $i \in [1, n-1]$
$v_i^x \leftarrow v_{i-1}^{x-1};$
endfor
$v_0^x \leftarrow$ *'X';*

Step3 : *for all* $a^y \in A, a^y \neq a$
match=true;
for all $i \in [0, n-1]$
if $(v_i^y \neq$ *'X') and* $(v_i^x \neq v_i^y)$
then match=false;
endfor
if match=true
then goto Step4
endfor
$x \leftarrow x + 1;$
goto Step2;

Step4 : *chk=inconsistency_checking(V);*
if chk=true
then goto Step2;

Step5 : *p=Gauss_elimination(V);*

Step6 : *perform fault simulation to determine test length*

Initially, we choose a test vector which can detect the hard-to-detect faults and has the highest fault coverage from the deterministic test set as an initial state of LFSR. The definition and the selection procedure of hard-to-detect fault in [3] is used here. Then we emulate the shift operation of the LFSR, and set the most significant bit to 'unknown'. After each shift operation, we search the deterministic test set to find a vector which is equivalent to the current selected vector bit by bit except the 'unknown' bits. If this vector is found, we use it to set all the 'unknown' bits of the current random vector. We can get a set of multi-variable linear equations from the vector set generated above, and the most significant bit of each vector is set to the equation value of its former vector. The Gauss-elimination procedure is performed first for the inconsistency check of the random vector set. To explain the inconsistency of the test vector, we will define the linear dependence for the test vector first.

Definition 1 : Linear Dependence of Test Vectors
Vector v^i *and the corresponding random vector space* V *of the random tests is said to be linearly dependent if there exist a finite number of distinct nonzero elements* $v^1, v^2, ..., v^k$ *in* V *such that* $v^i \oplus v^1 \oplus v^2 \oplus ... \oplus v^k = 0$ *in which* 0 *is the zero vector.*

From the definition above we can then define the inconsistency of the test vectors as follows.

Definition 2 : Inconsistency of Test Vectors
Vector v^i *is said to be inconsistent to the random vector space* V *if* v^i *is linearly dependent to* V *and* $v_0^{i+1} \neq 0$ *after elimination.*

The procedure *inconsistency_checking(V)* in Step4 can be treated as a pre-processing to ensure the existence of the solution for these linear equations. If the inconsistency does occur, the shift and search operations are continued until the next vector is found. Otherwise, the Gauss-elimination procedure is then performed again to solve the feedback link assignments. It is shown in Step5 and p is the resultant polynomial vector. For each iteration, we drop the detected faults by fault simulation and remove the patterns that cover the same fault set from the deterministic test set. This

procedure is executed for the remaining vectors of the deterministic test set until complete fault coverage is achieved.

This method has two main advantages. First, the fault coverage can be guaranteed to be equal to that of the deterministic test set. Physically, even in random pattern resistant circuit we can use only a subset of the deterministic test set to generate a random test set which has complete fault coverage. The second advantage is that there is no restriction on the circuit under test. We have focused only on the vectors and do not care what the circuit is. It makes this method applicable to general circuits and saves the analysis of the circuit characteristics.

4 Experimental Results

There are two heuristics used in our experiments. First, we perform the Gauss-elimination process to solve the polynomial whenever an equivalent vector is found. This makes that two deterministic test vectors be used at the same time. The results of the first heuristic are shown under the column 2-S in Table 1.

			2-S				n-S			
circuit	PIs	# of ATPG pts	# of s-p pairs	% of memory overhead	test length	simulation length	# of s-p pairs	% of memory overhead	test length	simulation length
c432	36	29	1	0.00	320	1000	1	0.00	320	1000
c499	41	52	1	0.00	719	1000	1	0.00	679	1000
c880	60	18	2	22.22	4203	3000	2	22.22	1596	1000
c1355	41	84	1	0.00	1447	2000	1	0.00	1641	2000
c1908	33	106	2	3.77	3751	3000	2	3.77	3659	3000
c2670	157	44	19	86.36	33000	5000	15	68.18	61400	10000
c3540	50	88	2	4.55	7304	4000	2	4.55	6592	4000
c5315	178	44	1	0.00	2396	3000	1	0.00	1843	2000
c6288	32	13	1	0.00	50	1000	1	0.00	43	1000
c7552	206	73	16	43.84	60350	10000	20	54.79	32800	5000

Table 1: Results with dynamic simulation length.

For each circuit, the number of PIs(primary inputs) and the number of tests in the deterministic test set(# of ATPG patterns) are given first in Table 1. The deterministic test set which is used here was generated by the PODEM [9] and an additional compaction [10] to reduce the test length. None of the don't care values is used in our experiments. The first column under the 2-S method is the number of seed-polynomial pairs used. The physical memory size required is twice of this value. The next column shows the percentage of the storage overhead compared with the necessary space for the ATPG tests. Following is the pseudorandom test length. The simulation length of each test generation cycle is in the last column.

In the 2-S experiments, we always obtain less than N linear independent equations. This will lead to more than one different random vector space be spanned. Thus we can only find a solution randomly. In order to decide unique as well as more suitable feedback link assignments, the second heuristic is then used. Extended from the previous method, we continue the shift and search operations after the first vector was found until the N linear vectors are found. The results are shown under the column n-S in Table 1.

Complete fault coverage is achieved in all cases. The hardware overhead for the storage space of the seeds and polynomials is between 0% to 25% except the c2670 and c7552. The test length is restricted to a reasonable size which reduces the test application time. However, hardware overheads of c2670 and c7552 are also restricted to 86.36% and 54.79% respectively. We apply different simulation lengths in the experiments. Although larger test length will increase the number of detected faults, it will also lead to an unacceptable test application time and must be considered at the same time to balance the test length reduction and the hardware optimization.

As described in section 2, a major advantage of our approach is the *m-to-1* relationship of seed and polynomial. Experiments are provided only for those circuits which require more than one seed-polynomial pair. And for simplicity, we perform n-S heuristic only in the following experiments. Simulation results are collected in Table 2. First two columns are the circuit name and the simulation length. The first case is *m-s1p* under the column **ms1p**. The number of required seeds is given first. The other two columns are the percentage memory overhead and the physical test length same as above.

		ms1p			1smp			
circuit	simulation length	# of seeds	% of memory overhead	test length	# of polynomials	% width of polynomial	% of memory overhead	test length
c880	1000	2	11.11	1706	2	34/60	6.30	1704
c1908	3000	2	1.89	3748	2	8/33	0.46	4034
c2670	5000	30	68.18	26667	23	157/157	52.27	29868
c3540	4000	2	2.27	5187	2	19/50	0.86	5642
c7552	10000	30	41.10	85047	27	206/206	36.99	95419

Table 2: Results for *m-to-1* seed-polynomial relationship.

Experimental results for *1smp* are in the column **1smp** in Table 2. First column shows the number of required polynomials. The polynomial width reduction is shown in the column `% width of polynomial` by a factor representation of the width of stored polynomial and the number of PIs. Thus, the percentage memory overhead is multiplied by the percentage width of polynomial as shown. The last column is the physical test length.

Obviously, the percentage memory overhead is reduced further. For c880, c1908, and c3540, the required memory overhead is reduced to less than 12% and even under 7% in the case of *1smp*. About 53% and 37% memory overheads are required for c2670 and c7552 respectively. But the large number of polynomials leads none of improvement for polynomial resetting.

For the circuit with 0% memory overhead, a fixed LFSR implementation is provided and is the same as a regular LFSR without any additional hardware. Less polynomial resetting elements in reassignable LFSR is required for c880, c1908, and c3540 to get the second polynomial. They both lead to a very small test hardware. In the worst case of random pattern resistant circuits c2670 and c7552, the whole reassignable LFSR is needed and becomes twice of the regular LFSR as in [5] and [6].

Our approach is also comparable to the weighted random method in [3]. It needs 3_gate module to generate specified weight and always lead to a larger hardware in general cases. Besides, incomplete fault coverage is achieved for random pattern resistant circuits c2670 and c7552. It gets only 95.74% fault coverage for c2670 for example. In this case, only 2 polynomials are required for *1smp* LFSR in our approach and only 4.55% memory storage overhead is introduced. Similarly, for c7552, 98.26% fault coverage is achieved by 5 polynomials with 6.85% memory overhead in our approach.

In summary, the experimental results show that except the c2670 and the c7552, much better results are obtained in all cases. Although some improvements have been made on the random pattern resistant circuits, the overall performance is still not satisfactory. This is due to the long distance between two matching vectors in the *FLA* algorithm. Since a deterministic value is used in the algorithm, a more restricted search space will limit the flexibility of this algorithm. The other limitation is the LFSR model used here, the external-XOR LFSR. It will generate a more regular test sequence to cover restricted number of faults.

5 Conclusions

A pseudorandom test pattern generation method based on the Gauss-elimination procedure has been illustrated in this paper. The pattern generator uses a general LFSR model with multiple seeds and polynomials to generate the pseudorandom tests. We use a deterministic test set to guide the selection of appropriate seeds and polynomials. Experiments were performed to show that the goal of complete fault coverage, minimum hardware overhead, and small test length are achieved on most ISCAS_85 benchmark circuits.

We not only can guarantee complete fault coverage but also can obtain a complete fault coverage random test set from an incomplete fault coverage ATPG test set. Besides, the absence of circuit analysis makes our approach applicable to all kinds of circuits. Further research is being conducted on the hardware optimization of the random pattern resistant circuits by including the don't care value and the internal-XOR LFSR model. Application to sequential circuits and more physical experiments on VLSI circuit or MCM die testing are also future works.

References

[1] H. J. Wunderlich, "On Computing Optimized Input Probabilities for Random Tests," *In Proc. of the Design Automation Conf.*, pp. 392-298, 1987.

[2] J. Hartmann et al., "How to do Weighted Random Testing for BIST ?" *In Proc. of the Int. Conf. on Computer-Aided-Design*, pp.568-571, 1993.

[3] 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 of Integrated Circuits and Systems*, Vol. 12, No. 7, pp.1050-1058, July 1993.

[4] J. Savir et al., "A Multiple Seed Linear Feedback Shift Register," *IEEE Trans. Computers*, pp.250-252, 1992.

[5] S. Hellebrand et al., "Generation of Vector Patterns through Reseeding of Multiple Polynomial LFSRs," *IEEE Int. Test Conf.*, pp.120-129, 1992.

[6] S. Venkataraman et al., "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers," *In Proc. of the Int. Conf. on Computer-Aided-Design*, pp.572-577, 1993.

[7] S. J. Upadhyaya et al., "On-chip Test Generation for Combinational Circuits by LFSR Modification," *In Proc. of the Int. Conf. on Computer-Aided-Design*, pp.84-87, 1993.

[8] Michael D. Greenberg, "Advanced Engineering Mathematics," *Prentice-Hall Inc.*, 1988.

[9] P. Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Trans. on Computers*, pp.215-222, Mar. 1981.

[10] J. Chang et al., "Test Set Compaction for Combinational Circuits," *Asia Test Symposium Japan*, Nov. 1992.