

On Calculating Efficient LFSR Seeds for Built-In Self Test

C. FAGOT

O. GASCUEL

P. GIRARD

C. LANDRAULT

Laboratoire d'Informatique de Robotique et de Microélectronique de Montpellier,
Université Montpellier II / CNRS

161 rue Ada, 34392 Montpellier Cedex 5 FRANCE

Tél.: (+33) 467 41 85 78 Fax: (+33) 467 41 85 00 Email: <name>@lirmm.fr

Abstract

Linear Feedback Shift Registers (LFSRs) are commonly used as pseudo-random test pattern generators (TPGs) in BIST schemes. This paper presents a fast simulation-based method to compute an efficient seed (initial state) of a given primitive polynomial LFSR TPG. The size of the LFSR, the primitive feedback polynomial and the length of the generated test sequence are a priori known. The method uses a deterministic test cube compression technique and produces a one-seed LFSR test sequence of a predefined test length that achieves high fault coverage. This technique can be applied either in pseudo-random testing for BISTed circuits containing few random resistant faults, or in pseudo-deterministic BIST where it allows the hardware generator overhead area to be reduced. Compared with existing methods, the proposed technique is able to deal with combinational circuits of great size and with a lot of primary inputs. Experimental results demonstrate the effectiveness of our method.

Keywords : BIST, LFSR, Seed, Pseudo-random Testing

1. Introduction

Modern design and package technologies make external testing increasingly difficult and the built-in self test (BIST) has emerged as a promising solution to the VLSI testing problem. BIST is a design for testability methodology aimed at detecting faulty components in a system by incorporating test logic on-chip. The main components of a BIST scheme are the test pattern generator (TPG), the response compactor, and the signature analyzer. The test generator applies a sequence of patterns to the circuit under test (CUT), the responses are compacted into a signature by the response compactor, and the signature is compared to a fault-free reference value.

BIST is well known for its numerous advantages such as at-clock-speed test of modules, no need for automatic test equipment and support during system maintenance [1], while preserving reasonable fault coverage. It also simplifies fault diagnosis and may provide some on-line features [2]. While the cost of high-speed VLSI testers *increases* with each process generation, BIST is becoming increasingly attractive because its cost (additional silicon area) *decreases* with each process generation. Moreover, owing to the emergence of core-based "system-on-a-chip" designs, BIST represents one of the most suitable methods for system testing. The testability of embedded cores may be hampered by limited accessibility, and necessary test information is often hidden in order to protect intellectual property. This kind of problem is solved if the system is equipped with BIST features.

Linear Feedback Shift Registers (LFSRs) are commonly used as pseudo-random test pattern generators in BIST schemes. They have a simple structure requiring very small area overhead, and they can also be used as output response analyzers, thereby serving a dual purpose [3]. However, the quality of the LFSR

pseudo-random test set varies depending on the CUT. High fault coverage in an acceptable test length usually cannot be easily achieved without addressing the problem of random pattern resistant faults. Several methods have been proposed to solve this problem they and can be classified as those that modify the circuit under test and those that modify the pseudo-random input patterns. The first category involves test point insertion techniques that increase the fault detection probabilities in the CUT [4, 5]. The second category consists of techniques (called mixed-mode or pseudo-deterministic BIST techniques) such as weighting the pseudo-random patterns [6, 7], using counter-based schemes [8, 9], or performing bit-fixing (also called *pattern mapping*) [10, 11, 12]. The main drawback of test point insertion techniques is that modifying the CUT may not be acceptable for designers due to possible performance degradation and its impact on design flow. The disadvantages of the second category of techniques are that they require additional area and sometimes delay overhead compared to an LFSR-TPG, although they reduce the size of the test set, in some cases considerably.

A possible solution for alleviating these drawbacks is to act first on the LFSR itself by primarily selecting good seed and/or feedback polynomial. By this way, the set of random resistant faults missed by the LFSR-TPG during test pattern generation is as small as possible, and the cost of the above mixed-mode techniques to improve the fault coverage or reduce the test length is decreased. The problem of finding a good LFSR seed for a given feedback polynomial was first addressed in [13]. Techniques for the generation of test patterns through reseeding of multiple polynomial LFSRs were proposed in [14, 15]. These techniques involve storing LFSR seeds in a ROM instead of storing the deterministic patterns (that detect random resistant faults) themselves. The same LFSR used to generate pseudo-random patterns is loaded with seeds from which vectors fitting test cubes detecting hard-to-test faults are produced. [16] proposed an analytical method providing a one-seed test sequence for random resistant circuits from an LFSR with a given feedback polynomial. This method uses the theory of discrete logarithms to embed a subset of deterministic test patterns in an LFSR sequence.

Computing seeds in reseeding techniques is done by solving systems of linear equations whose complexity grows with the number of inputs of the CUT and the number of cares or specified bits of test cubes. Computing an LFSR seed with the technique proposed in [16] is performed by a *test embedding procedure* whose complexity quickly increases with the number of inputs. In reseeding approaches, finding test sequences with acceptable test length and fault coverage is achieved to the detriment of the area overhead required to store seeds (for example, several thousand seeds have to be stored for the biggest ISCAS89 scan circuits [15]). In [16], finding one seed for circuits with a lot of inputs or circuits with a high number of random resistant faults is impractical. For example, only results for circuits whose number

of inputs ranged from 20 to 40 are reported in [16], and the authors mention that test embedding with discrete logarithms is not a viable test option for greater circuits.

In this paper, we propose a fast simulation-based method to compute an efficient seed of a given primitive polynomial LFSR-TPG. We consider that the size of the LFSR (number of stages), its primitive feedback polynomial, and the length of the generated test sequence are *a priori* known. Our method is intended to produce a one-seed test sequence of a given test length that achieves a high stuck-at fault coverage. Moreover, it concentrates on the hardest to detect faults of the CUT. The main feature of the proposed method is that it applies in a lot of BIST situations. It can be used in pseudo-random testing for BISTed circuits that contain few random resistant faults. In this case, it allows the fault coverage achieved with a predetermined pseudo-random test length to be significantly improved. It can also be used in pseudo-deterministic BIST where it allows the hardware generator overhead area to be reduced, because the number of hard faults focused by the deterministic part of the generator is initially reduced. Another important feature is that one-seed test sequences for circuits with a large number of inputs can now be obtained in a reasonable computation time. This feature alleviates problems raised by methods proposed so far.

The rest of the paper is organized as follows. In the next section, we give preliminary details about fault difficulty and test cube quality. In section 3, we present the proposed method for generating efficient one-seed LFSR test sequences. We first present a simple algorithm to compress test cubes generated by an ATPG program, then we describe the method itself. Experiments performed on ISCAS benchmark circuits are presented and discussed in section 4. Concluding remarks are given in section 5.

2. Preliminary details and basic definitions

Let F be the set of modeled faults, *e.g.* stuck-at faults, in the CUT. Let v be an input test pattern generated from a n -stage LFSR, where n is the number of primary inputs of the CUT; v belongs to $\{0,1\}^n$ and is denoted as $v = (v[0], v[1], \dots, v[n-1])$. Let c be a test cube of size n ; c is represented by a vector $(c[0], c[1], \dots, c[n-1]) \in \{0,1,*\}^n$. Let $S(c) = \{i \mid c[i] \neq *\}$ represent the set of specified bits in c , and $|S(c)|$ be the number of specified bits in this test cube. The number of fully specified test patterns that can be generated from a test cube is 2^q , where q is the number of unspecified bits in the test cube, *i.e.*, $q = n - |S(c)|$.

Let $v \in \{0,1\}^n$ be an input test pattern and $c \in \{0,1,*\}^n$ be a test cube. If $c[i]=v[i]$ holds for every $i \in S(c)$, then v is said to be *consistent* with c . Let $c_1 \in \{0,1,*\}^n$ and $c_2 \in \{0,1,*\}^n$ be two test cubes, c_1 and c_2 are said to be *compatible* if and only if $(c_1[i]=* \text{ or } c_2[i]=* \text{ or } c_1[i]=c_2[i])$ holds for every i . Compatibility between c_1 and c_2 is denoted $c_1 \odot c_2$.

When performing pseudo-random testing using an LFSR-TPG, the fault coverage that can be obtained is limited by the presence of random resistant faults in the CUT. To evaluate the detection hardness of these faults, we need to use a measure that represents the *difficulty* for a random pattern to detect a given fault. Computing this difficulty measure (*i.e.* the fault detection probability) is an NP-hard problem [7]. Therefore, we have to be satisfied with approximations. Moreover, it is not really awkward to have an easy-to-test fault estimated as hard-to-test. Conversely, a measure that declares a fault as easy when it is a really hard-to-test fault is problematic, and it is much more preferable to overestimate than to underestimate the fault difficulty.

The higher the number of patterns detecting a given fault, the easier this fault is detected. Let C be a set of test cubes generated by an ATPG program so that each fault in the CUT is detected by at least one test cube of C . Now, let C_f be the set of test cubes of C that detect the fault f . Let c be a cube that belongs to C_f ; the more unspecified bits (represented by $*$) in c , the more test patterns are consistent with c , and the easier f is regarding c . A simple estimation of the difficulty δ_f of a fault f is:

$$\delta_f = \min_{c \in C_f} (|S(c)|)$$

which is equal to n minus the maximum number of unspecified bits in a cube belonging to C_f . Note that this measure fulfills the above requirement. A cube with few specified bits cannot detect a hard fault and, therefore, δ_f is inevitably high. On the other hand, δ_f is high, even if f is easy to detect, when C_f (as generated by the ATPG program) only contains cubes with few $*$.

In our method, a “quality measure” for the test cubes is required. The quality of a test cube is evaluated with respect to the number of hard faults it detects. The quality of a test cube c is represented by a vector $Q_c = (q_c[1], q_c[2], \dots, q_c[n])$, where $q_c[i]$ is equal to the number of faults of difficulty i detected by c . More formally:

$$q_c[i] = \left| \left\{ f \text{ detected by } c / \delta_f = i \right\} \right|$$

For example, consider $n = 5$ and the test cube c with $Q_c = (26, 12, 14, 0, 1)$; this means that c detects 26 faults of difficulty 1, 12 faults of difficulty 2, ..., and 1 fault of difficulty 5. In order to compare the quality of two test cubes c and c' , we use a lexicographical order on their quality vectors. We have:

$$Q_c > Q_{c'} \Leftrightarrow \exists i \leq n \text{ such as :}$$

$$(q_c[i] > q_{c'}[i]) \text{ and } (\forall i < j \leq n, q_c[j] = q_{c'}[j])$$

For example, if we consider two test cubes c and c' for which $Q_c = (26, 12, 14, 0, 1)$ and $Q_{c'} = (30, 11, 10, 0, 1)$, we have $Q_c > Q_{c'}$ (c is of better quality than c') since $q_c[5] = q_{c'}[5]$, $q_c[4] = q_{c'}[4]$, but $q_c[3] > q_{c'}[3]$.

3. Finding an efficient one-seed LFSR test sequence

An n -stage LFSR with a primitive feedback polynomial generates a cyclic sequence containing all the non-zero binary n -tuples. Changing the polynomial changes the position of each n -tuple in the sequence, while the LFSR seed specifies the starting position for scanning the sequence. Consequently, the generated test sequence depends on the LFSR feedback polynomial and on the LFSR seed. In this section, we describe a fast simulation-based algorithm to compute an efficient seed of an *a priori* given primitive polynomial LFSR-TPG. We consider that the length l of the generated test sequence is set. The proposed method is thus intended to produce a one-seed test sequence of a given length that achieves the highest possible fault coverage. This method is divided into two parts. First (Section 3.1), a compressed set of test cubes C_{comp} is produced from a set of deterministic test cubes C generated by an ATPG program. This compression is specially useful when there are many deterministic test cubes with a lot of unspecified bits. Second (Section 3.2), we select the best one-seed test sequence (in terms of fault coverage) among those containing the best test cubes of C_{comp} .

3.1 Compressing the initial set of test cubes

Let c_1 and c_2 be two compatible test cubes $\in \{0,1,*\}^n$. The *merged cube* of c_1 and c_2 is $c = c_1 \cap c_2$. To obtain this merged cube c , we check the compatibility of c_1 and c_2 and report in c every bit specified in c_1 or in c_2 .

Algorithm-1 given below, iteratively merges compatible cubes. It first selects the best cube c from C , according to the quality measure Q . Then, it searches the cube x , compatible with c , which provides the best improvement of c when merged with this one. The process is repeated until c cannot be merged with any cube from C , and c is then stored in C_{comp} . Then we start again with the best cube that remains in C , and continue until C is empty. To measure the improvement provided by x to c , we consider the set of faults detected by x or c , and we compute $Q_{x \cup c}$. The cube x is preferred to x' when $Q_{x \cup c} > Q_{x' \cup c}$.

Algorithm-1: Constructing a compressed set of test cubes

Input: C = a set of test cubes.

Output: C_{comp} = set of compressed test cubes.

Begin

$C_{comp} \leftarrow \emptyset$;

While $C \neq \emptyset$ **do**

$c \leftarrow$ best cube from C ;

$C \leftarrow C - \{c\}$;

$D \leftarrow \{x \in C / x \odot c\}$;

While $D \neq \emptyset$ **do**

$x \leftarrow$ cube of D providing the best increasing of Q_c ;

$c \leftarrow$ merged cube of c and x (i.e., $c \cup x$);

$C \leftarrow C - \{x\}$;

$D \leftarrow \{x \in C / x \odot c\}$;

$C_{comp} \leftarrow C_{comp} \cup \{c\}$;

Return C_{comp} ;

End

3.2 Computing an efficient LFSR seed

In the C_{comp} set constructed as described above, the test cubes have a wide quality range. Remember that test cubes having the highest quality are those that detect the highest number of hard faults. Therefore, it is important to find a one-seed LFSR test sequence that is able to produce test patterns consistent with these test cubes (called *best test cubes* hereafter). We propose a fast simulation-based approach in which the aim is to study the highest possible number of test sequences containing the best test cubes of C_{comp} , and to select the one that achieves the highest fault coverage (its first vector thus giving the best LFSR seed). This approach is based on two complementary algorithms. Algorithm-2 swiftly simulates all test sequences containing a pattern consistent with a selected test cube of C_{comp} , and selects the seed of the test sequence that achieves the highest fault coverage. This algorithm is described in section 3.2.1. To obtain the best one-seed LFSR test sequence, Algorithm-3 uses Algorithm-2 with the best test cubes of C_{comp} . Algorithm-3 is detailed in section 3.2.2.

3.2.1 Algorithm-2: a fast simulation-based procedure

In order to address all the test sequences containing a selected test cube $c \in C_{comp}$, we should study the set of test sequences in which the first vector is consistent with c , plus those in which the second vector is consistent with c , and so on until all test sequences in which the last vector is consistent with c have been studied. However, an exhaustive simulation of all these test sequences would take a prohibitive amount of CPU time. For this reason, we propose the following alternative. As shown on the Figure 1, we first simulate all vectors of an LFSR test sequence Ω in which v is the first vector (the seed) and is determined so that it is consistent with c . This first step requires l simulations where l is the test sequence length. We then consider a new LFSR test

sequence Ω' in which v is the second pattern. As the feedback polynomial of the LFSR is known, we use the reciprocal feedback polynomial of the LFSR to calculate the first vector of Ω' . As Ω' is also of length l , the last vector of Ω is removed in Ω' . There are therefore $l-2$ common vectors between Ω and Ω' , so that only one additional simulation is needed to calculate the fault coverage achieved with Ω' . Of course, faults detected by the last vector of Ω (and not detected by any vector of Ω') are deleted from the list of faults detected by the new sequence Ω' .

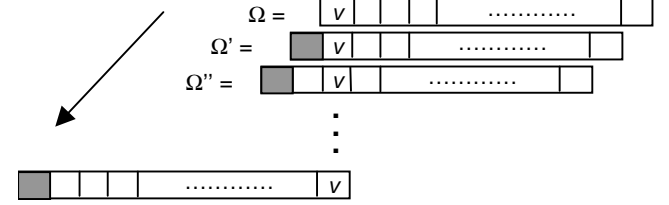


Figure 1: Fast simulation of l test sequences of length l

Next, the same process is used to compute Ω'' , and is repeated until the test sequence in which v is the last vector has been evaluated. Algorithm-2 given below details the whole process, and additional comments are then reported. In this algorithm, PREVIOUS(x) is a function that returns the previous vector of x in the test sequence generated by the given LFSR. PREVIOUS uses the reciprocal characteristic polynomial of the LFSR and is used $l-1$ times to reach the test sequence in which v is the last vector.

Algorithm-2: Fast simulation of l test sequences embedding a vector v

Input: v = a vector; l = the test sequence length.

Output: $bestSeed$ = seed of best sequence embedding v .

$bestCover$ = fault coverage provided by $bestSeed$.

Begin

$s \leftarrow v$;

 Simulation of the test sequence Ω whom seed is s ;

$ListSimul$ stores the simulation results;

$cover \leftarrow$ number of faults detected by Ω ;

$bestCover \leftarrow cover$;

For $i \leftarrow 1$ to $l-1$ **do**

$s \leftarrow$ PREVIOUS(s);

$Simul \leftarrow$ list of faults detected by s ;

$NewFaults$ = faults in $Simul$ not in $ListSimul$;

 Update $ListSimul$ using $Simul$;

 Keep and remove $LostFaults$ from $ListSimul$;

$cover \leftarrow cover - |LostFaults| + |NewFaults|$;

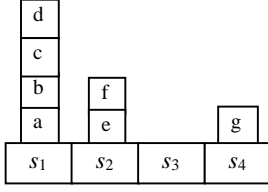
If $cover > bestCover$ **then**

$bestCover \leftarrow cover$;

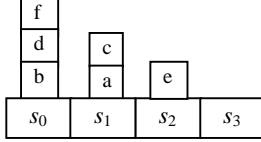
$bestSeed \leftarrow s$;

End

For the sake of efficiency, we need the data structure $ListSimul$ that stores, for every pattern p , the faults it detects and which are not detected by any pattern p' that appears before p in the sequence. Moreover, this data structure has to be updated at each iteration. Let $l=4$ and let s_i denote the i -th vector in the sequence. Moreover, assume that s_1 detects the fault set $\{a,b,c,d\}$, $s_2 : \{b,c,e,f\}$, $s_3 : \{a,e\}$ and $s_4 : \{a,f,g\}$. $ListSimul$ can be represented as follows:



LostFaults corresponds to the last heap and is equal to {g}. Assume now that $s_0 = \text{PREVIOUS}(s_1)$ detects the fault set {b,d,f}. When updated, *ListSimul* becomes :



Using this storing approach, only $2l-1$ simulations are needed to evaluate l sequences with length l . Indeed, l simulations are needed to evaluate the first sequence, and $l-1$ simulations are needed to evaluate the next sequences. Note that performing a complete simulation of each sequence containing a given vector (full simulation approach) would need l^2 simulations to obtain the same result. For some circuits (e.g. c2670), each simulation requires about 1 second CPU; with $l=10000$ the full simulation approach yields over 3 years of computations, whereas no more than 6 hours are required for the solution we propose.

3.2.2 Algorithm-3: the main procedure

The main procedure in which Algorithm-1 and Algorithm-2 discussed above are used is presented in this subsection and summarized in Algorithm-3. This algorithm is simply a generalization of Algorithm-2 to all the best test cubes of C_{comp} . Nevertheless, two points have to be clarified in this algorithm.

The first one relates to the way we determine a vector v that is consistent with c (a selected test cube among the best test cubes of C_{comp}) and which is further used in Algorithm-2. This vector is first determined so that the specified bits in c are reported in the same bit position in v (consistency). Next, the remaining bits in v are arbitrarily set at the logic “1” value. This solution experimentally provides better results than randomly setting the remaining bits at either “0” or “1”, or at the logic “0” value. Only one vector is thus produced from a selected test cube c of C_{comp} . The quality of the fully specified patterns obtained from C_{comp} is determined by simulation, and these patterns are sorted according to the criterion. The first motivation for this limitation is of course simulation time, which otherwise would be too high, due to the exponential number of vectors consistent with a given test cube. The second motivation is that it is easier to find previous patterns of a fully specified vector by using the reciprocal characteristic polynomial of the LFSR, than to find the previous patterns of a test cube by solving linear equations as in [14].

The second point relates to the number of best test cubes of C_{comp} needed for testing in order to obtain good results. This number is not so important for the following reason: as patterns selected during this process have decreasing qualities and hence detect a decreasing number of hard faults, the probability of getting a test sequence that provides better fault coverage than a previously studied sequence decreases. Experimental results reported in the next section demonstrate that test sequences of high quality can be obtained with a reasonable computation time.

Algorithm-3: Finding an efficient LFSR seed

Input: C = a set of test cubes provided by an ATPG tool.
 l = the test sequence length.

Output: *bestSeed* = the most efficient LFSR seed

Begin

$bestCover \leftarrow 0\%$;

$C_{comp} \leftarrow$ merged cubes of C (Algorithm-1);

$V \leftarrow$ cubes of C with unspecified bits set to value 1;

Simulate patterns of V ;

While high fault coverage is not reached **do**

$v \leftarrow$ best pattern of V according to quality vectors;

$V \leftarrow V - v$;

$seed \leftarrow$ seed of the best test sequence Ω embedding v (Algorithm-2);

If fault coverage of $\Omega > bestCover$ **then**

$bestCover \leftarrow$ fault coverage achieved with Ω ;

$bestSeed \leftarrow seed$;

If fault coverage = 100% **then return** $bestSeed$;

Return $bestSeed$;

End

4. Experimental results

Experimental evaluation of the proposed method was conducted using the standard sets of ISCAS’85 and ISCAS’89 benchmark circuits. It was assumed that the flip-flops in the ISCAS’89 circuits were configured as part of the LFSR during testing so that the circuits are tested like combinational circuits. Note that only detectable stuck-at faults were considered in the fault coverage calculations.

Circuit	#inps	#faults	Length	FC in	CPU Time
C880	60	1760	1K	99.5	1328.41
C1355	41	2702	3K	100	1893.71
C1908	33	3805	4K	100	2025.41
C2670	233	4995	5K	91.4	25582.15
C3540	50	6824	4.5K	100	27479.24
C5315	178	10568	5K	100	3800.18
C7552	206	14865	8K	97.8	165930.26
S208	18	436	750	99.3	191.67
S208_89	19	416	750	98.8	182.64
S298	17	596	200	100	11.2
S344	24	670	200	100	5.25
S382	24	764	250	100	6.38
S386	13	772	2K	100	22.91
S420	34	916	1K	85.4	570.30
S420_89	35	840	1K	94.0	536.07
S444	24	866	300	100	38.37
S510	25	1020	500	100	12.58
S526	24	1051	4K	99.7	2405.71
S641	54	1274	10K	99.5	10191.26
S713	54	1353	10K	99.5	11086.12
S820	23	1640	5K	98.9	3367.31
S832	23	1647	5K	98.8	3400.50
S838	66	1876	10K	75.5	12460.77
S838_89	67	1676	10K	86.2	11926.16
S953	22	1860	5K	98.6	7395.07
S1196	31	2390	10K	99.9	15279.51
S1423	91	2820	3K	99.6	7285.97
S1488	14	2976	3K	100	146.19
S1494	14	2972	1K	99.1	1570.39
S9234	247	17350	1K	84.9	26817.68

Table 1: Intrinsic results on Benchmark circuits

Experiments were conducted using algorithms described in section 3. The size of each LFSR is equal to the number of primary inputs in each circuit. The characteristic polynomial of each LFSR is a primitive feedback polynomial. Fault simulations in each circuit were performed using a home fault simulation tool based on the critical path tracing algorithm proposed in [17]. Deterministic test cubes were generated using the ATPG tool of the SUNRISE Test System.

A first sample of intrinsic results is given in Table 1. The first column contains the circuit names. The second and third columns give the number of primary inputs (#inps) and the number of detectable faults (#faults) in each circuit. The fourth column reports the chosen length of the generated test sequence (Length), and column five (FC in %) gives the fault coverage obtained with the proposed method on each circuit. The fifth column gives the CPU time (in seconds, on a Pentium 350) needed for the complete process of generating a one-seed LFSR test sequence (construction of a compressed set of test cubes and computation of an efficient LFSR seed). For each circuit, 20 test cubes from C_{comp} are evaluated in Algorithm-3. However, we observed that it is usually enough to use the 5 or 6 best test cubes of C_{comp} to obtain the same result (seed), which illustrates the efficiency of our quality measure.

The results in Table 1 show the performances obtained with the proposed method to compute an LFSR seed. For test sequence lengths no greater than 10K, the average fault coverage is 96.8%, while the better and the worst are respectively 100% (in many cases) and 75.5%. They also demonstrate that circuits with a lot of inputs and circuits of great size can be dealt with efficiently. Thus, the fault coverage achieved for the s9234 (247 inputs and a size of 4505 gates equivalent) is 84.9% with a test sequence length of 1K, for a computational time of no more than 7.5 hours. This circuit has too much input to be dealt by [16]. Note that for the biggest circuits, the applicability of the method depends on two user specified constraints. The first one is the computational effort one is willing to spend and the second is the length of the test sequence one is willing to allow. The results presented in Table 1 can be further improved by focusing on these parameters.

Circuit	Test Length = 1K		Test Length = 10K	
	AvRand	OurSeed	AvRand	OurSeed
C2670	86.0	91.2	87.0	91.6
C5315	99.7	99.9	100	100
C7552	93.9	97.2	95.7	98.0
S420	69.1	85.4	84.7	92.7
S420_89	85.8	94.0	92.3	96.4
S641	96.8	98.4	98.5	99.5
S838	50.1	72.2	58.5	75.5
S838_89	77.6	85.4	80.7	86.2
S1494	97.5	99.1	100	100
S9234	78.4	84.9	89.8	92.8

Table 2: Comparison with average randomly selected seed coverage

In order to compare our method with random selection of seeds, we conducted two sets of experiments (Tables 2 and 3). In the first set (Table 2), we measured the average stuck-at fault coverage (over 10000 trials) of pseudo-random test sequences with 1000 and 10000 test sequence lengths, and ran our method with the same sequence lengths. Circuits hard to test or having numerous inputs were chosen for this comparison. Note that the average fault coverage was evaluated by performing Algorithm-2 with a randomly chosen vector, for computation time convenience. Indeed, for some circuits (*e.g.* c7552 or s9234) the time needed to fully simulate over 10000 pseudo-random test

sequences is excessively high. For each tested length (“Test Length = 1K” and “Test Length = 10K”), we report the average (AvRand) stuck-at fault coverage (in %) achieved with random seeds. The fault coverage achieved with the proposed method (20 test cubes from C_{comp} evaluated in Algorithm-3) is given in Columns 2 and 4 (OurSeed). These results show that in all cases, the fault coverage achieved with the one-seed LFSR test sequence generated by the proposed method is much higher than the average fault coverage obtained randomly. For test sequence length 1000, the average fault coverage of the proposed method is 90.8%, while the average random seed selection fault coverage is 83.5%. For test sequence length 10000, the average fault coverage of the proposed method is 93.3%, while the average random seed selection fault coverage is 88.7%. For very hard-to-test circuits, the difference between our results and random seed selection is even more significant.

Circuit	Time	#Rand	MaxRand	OurSeed
C2670	4533	42	86.3	91.2
C5315	11474	43	99.8	99.9
C7552	19184	45	94.6	95.3
S420	625	51	75.8	81.4
S420_89	583	50	88.9	94.0
S641	813	37	98.0	98.4
S838	2067	76	54.6	69.7
S838_89	1654	61	79.6	85.4
S1494	1100	33	98.5	98.8
S9234	12393	21	79.6	84.4

Table 3: Comparison with best randomly selected seed coverage for a computation time

In Table 3, we compare the fault coverage provided by our method (column “OurSeed”, with 10 test cubes from C_{comp} evaluated in Algorithm-3) and the best fault coverage of a randomly selected seed test sequence (column “MaxRand”). For these experiments, the test length is 1000, and the computational time (in seconds and in second column) required to find the best randomly selected seed is equal to that used by our method. The third column of Table 3 indicates the number of random test sequences fully simulated during the accorded time. Through this experiment, it appears that the results provided by our method are always better than those of a random seed selection. In average, the fault coverage provided by the proposed method is 89.8%, while the fault coverage of the best random seed selection is 85.5%. For some circuit, the difference between the results is still much more important. For example, this difference is about 15% for the s838. It also appears that for the same computational time, it is not possible to fully simulate a high number of test sequences with a random selected seed (only 46, on average).

In the introduction of this paper, we pointed out that the main feature of the proposed method is that it can be applied in a lot of BIST situations. It can be used in pseudo-random testing for BISTed circuits that contain few random resistant faults. In this case, it allows to significantly improve the fault coverage achieved with a predetermined pseudo-random test length. The results listed in Tables 1, 2 and 3 confirm this assertion. It can also be used in pseudo-deterministic BIST, where it allows the hardware generator overhead area to be reduced, since the number of hard faults to target with the deterministic part of the generator is initially reduced. In order to demonstrate the effectiveness of our method in pseudo-deterministic BIST, we implemented two existing BIST scheme, [12] and [11], in which LFSR plus mapping logic are combined to produce test sequences providing 100% stuck-at fault coverage with test

sequence length *a priori* known (1000 here). Our results are reported in Table 4. We first considered a one-seed LFSR test sequence produced by our method, and computed the number of deterministic test cubes (Column OurSeed) needed to be implemented in order to achieve 100% fault coverage with [12] or [11]. Next, we considered a test sequence with the best randomly chosen seed found in the previous experiment (see Table 3); for this second test sequence, we also computed the number of deterministic cubes (Column MaxRand) that have to be implemented. Finally, we estimated the overhead area (Gates Equivalent) of the additional mapping logics for both test sequence using [12] and [11]. For each of these mixed-mode techniques we computed the ratio between the area overhead needed using our proposed seed and the random seed (columns [11]Ratio and [12]Ratio).

Circuit	Mixed-mode BIST			
	MaxRand	OurSeed	[11]Ratio	[12]Ratio
C2670	116	90	0.96	0.78
C5315	17	13	0.45	0.69
C7552	157	143	0.95	0.84
S420	71	52	0.92	0.85
S420_89	40	19	0.32	0.14
S641	16	15	0.90	1.33
S838	218	172	0.92	0.75
S838_89	109	95	0.79	0.73
S1494	24	14	0.92	0.76
S9234	708	598	0.72	0.83
Average	147.6	121.1	0.78	0.77

Table 4: Comparative results for a pseudo-deterministic BIST scheme

These results (Table 4) highlight the advantages of our method in the context of mixed-mode BIST techniques, where it reduces the hardware generator overhead area by a quarter (on average). For some circuits (e.g. s420_89), the area of the mapping logic may be more than 70% smaller with our solution.

5. Conclusion

In this paper, we propose an efficient method to compute the seed of an LFSR-TPG with a given characteristic polynomial and a given test length. The results demonstrate that our technique can be successfully applied either in pseudo-random testing for BISTED circuits that contain few random resistant faults, or in pseudo-deterministic BIST where it allows the hardware generator overhead area to be reduced. Moreover, our technique is able to deal with combinational circuits of great size and with a lot of primary inputs. Further studies will be conducted to adapt it to the test-per-scan scheme, to develop it for delay faults testing, and to avoid the arbitrarily completion of compressed cubes with logic value 1.

References

- [1] H.J. Wunderlich and Y. Zorian, *Built-In Self Test (BIST): Synthesis of Self-Testable Systems*, Kluwer Academic Publishers, 1997.
- [2] S.K. Gupta and D.K. Pradhan, *Utilization of On-Line (concurrent) Checkers during Built-In Self Test and Vice-versa*, IEEE Trans. on Computers, vol. 45, n° 1, January 1996.
- [3] A. Krasniewski and S. Pilarski, *Circular Self-Test Path: a Low Cost BIST Technique for VLSI Circuits*, IEEE Trans. on Computer-Aided Design, vol. 8, n°1, pp. 46-55, January 1989.
- [4] K.T. Cheng and C.J. Lin, *Timing Driven Test Point Insertion for Full-Scan and Partial-Scan BIST*, IEEE Int. Test Conf., pp. 506-514, 1995.
- [5] N. Tamarapalli and J. Rajski, *Constructive Multi-Phase Test Point Insertion for Scan-Based BIST*, IEEE Int. Test Conf., pp. 649-658, 1996.
- [6] F. Muradali, V.K. Agarwal and B. Nadeau-Dostie, *A New Procedure for Weighted Random Built-In Self-Test*, IEEE Int. Test Conf., pp. 660-669, 1990.
- [7] H.J. Wunderlich, *Multiple Distributions for Biased Random Test Patterns*, IEEE Trans. on Computer-Aided Design, vol. 9, n°6, pp. 584-593, June 1990.
- [8] S.B. Akers and W. Jansz, *Test Set Embedding in a Built-In Self-Test Environment*, IEEE Int. Test Conf., pp. 257-263, 1989.
- [9] D. Kangaris and S. Tragoudas, *Generating Deterministic Unordered Test Patterns with Counter*, IEEE VLSI Test Symp., pp. 374-379, 1996.
- [10] M. Chatterjee and D.K. Pradhan, *A Novel Pattern Generator for Near-Perfect Fault Coverage*, IEEE VLSI Test Symp., pp. 417-425, 1995.
- [11] C. Fagot, O. Gascuel, P. Girard and C. Landrault, *A Ring Architecture Strategy for BIST Test Pattern Generation*, IEEE Asian Test Symposium, pp. 418-423, 1998.
- [12] N.A. Toubia and E.J. McCluskey, *Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST*, IEEE Int. Test Conf., pp. 674-682, 1995.
- [13] B. Koenemann, *LFSR-Coded Test Patterns for Scan Designs*, IEEE Euro. Test Conf., pp. 237-242, 1991.
- [14] S. Hellebrand, S. Tarnick, J. Rajski and B. Courtois, *Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers*, IEEE Int. Test Conf., pp. 120-129, 1992.
- [15] S. Venkataraman, J. Rajski, S. Hellebrand and S. Tarnick, *An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers*, IEEE Int. Conf. on Computer-Aided Design, pp. 572-577, 1993.
- [16] M. Lempel, S.K. Gupta and M.A. Breuer, *Test Embedding with Discrete Logarithms*, IEEE VLSI Test Symp., pp. 74-78, 1994.
- [17] M. Abramovici, P.R. Menon and D.T. Miller, *Critical Path Tracing: An Alternative to Fault Simulation*, IEEE Design & Test of Computers, vol. 1, n° 1, February 1984.