JOURNAL OF ELECTRONIC TESTING: Theory and Applications 18, 315–332, 2002

On-the-Fly Reseeding: A New Reseeding Technique for Test-Per-Clock BIST

EMMANOUIL KALLIGEROS, XRYSOVALANTIS KAVOUSIANOS,
DIMITRIS BAKALIS AND DIMITRIS NIKOLOS
Department of Computer Engineering & Informatics, University of Patras, 26500, Patras, Greece;
Computer Technology Institute, 61 Riga Feraiou Str., 26221, Patras, Greece
kalliger@ceid.upatras.gr
kabousia@ceid.upatras.gr
bakalis@cti.gr
nikolosd@cti.gr

Received June 14, 2001; Revised October 26, 2001

Editors: J.P. Hayes, M. Nicolaidis and C. Metra

Abstract. In this paper we present a new reseeding technique for test-per-clock test pattern generation suitable for at-speed testing of circuits with random-pattern resistant faults. Our technique eliminates the need of a ROM for storing the seeds since the reseeding is performed on-the-fly by inverting the logic value of some of the bits of the next state of the Test Pattern Generator (TPG). The proposed reseeding technique is generic and can be applied to TPGs based on both Linear Feedback Shift Registers (LFSRs) and accumulators. An efficient algorithm for selecting reseeding points is also presented, which targets complete fault coverage and allows to well exploiting the trade-off between hardware overhead and test length. Using experimental results we show that the proposed method compares favorably to the other already known techniques with respect to test length and the hardware implementation cost.

Keywords: built-in self-test, test-per-clock schemes, linear feedback shift registers, accumulator-based test pattern generators, reseeding

1. Introduction

The traditional testing approaches, based on external *Automatic Testing Equipment* (ATE), are becoming more and more unsuitable for *System-on-Chip* (SOC) testing. The reason is twofold: (a) the gap between I/O and internal bandwidth often prevents ATEs from testing SOCs at speed and (b) the number of externally accessible I/O pins, although being fairly high (several hundreds), strongly limits the controllability and observability of the embedded modules.

Built-In Self-Test (BIST) [1–3, 6, 28, 39] has been widely recognized as an effective approach for testing SOCs, since it incorporates in the same IC the *Circuit Under Test* (CUT) and its tester, enabling this way the chip to test itself. The main components of a BIST scheme are the *Test Pattern Generator* (TPG) that produces the test patterns applied to the CUT and the *Test Response Verifier* that compacts the responses of the CUT to a single pattern called signature and compares it with the signature of the fault-free circuit. Minimal test application time and area overhead, negligible, if not zero, performance degradation and at-speed testing are essential for any successful BIST scheme. Furthermore, in most applications, complete (100%) fault coverage is desirable.

Linear Feedback Shift Registers (LFSRs) have been commonly used as pseudorandom test pattern generators in BIST schemes. Their structure is simple, they require very small area overhead and furthermore can be used both for test pattern generation and test response compaction. However, for circuits with random pattern resistant faults, high fault coverage cannot be achieved within an acceptable test length. Several techniques have been proposed to solve this problem [4, 9, 12, 13, 15–17, 20, 21, 23–25, 29, 36–38].

The weighted-random testing techniques [24, 38] although they reduce the test application time, in most cases still require excessively long test sequences for circuits with hard faults [12]. Furthermore, the hardware overhead of the weighted-random pattern generators may be high [19].

Other methods are based on the appropriate selection of the initial seed [13, 21, 25]. Lempel et al. [21] proposed an analytical method for providing a one-seed test sequence from a LFSR with a given feedback polynomial. Since this method uses the theory of discrete logarithms to embed a set of deterministic test patterns in a LFSR sequence, finding one seed for circuits with a lot of inputs or with a high number of random pattern resistant faults is impractical. In [25] several methods to tailor a LFSR to a CUT were presented, that attempt to select the most effective LFSR and initial state for the circuit. The authors of [13] describe a simulation-based method for computing an efficient initial seed of a given primitive polynomial LFSR-based TPG. The latter two techniques, whereas they improve the resulting fault coverage, they are unable to fully test a CUT with many random pattern resistant faults, using an acceptable number of test vectors.

LFSR reseeding [15–17, 20, 23, 29, 37] has been proposed as a possible solution to cope with this drawback. In [20] a test-per-scan technique is presented where a LFSR is used to generate pseudorandom and deterministic patterns, which are encoded as seeds. In [29] a test-per-clock reseeding scheme based on a modified design of an LSSD-based LFSR is described. In this scheme the seeds cannot be predetermined, they are uniformly distributed over the entire LFSR pattern space. This results in long test sequences in the case of circuits with hard-to-detect faults. Test-per-scan techniques for generating test patterns through reseeding of multiple polynomial LFSRs were proposed in [15, 16, 37]. The LFSRs are used to generate both pseudorandom and deterministic patterns. Deterministic patterns are encoded with a seed and a polynomial ID, where the seed specifies the value to be loaded in the register and the polynomial ID selects one of the feedback polynomials. The seeds and the polynomial IDs are stored in a ROM. In [23], a test-per-clock scheme using a Shift Register driven by a LFSR (LFSR/SR), along with a discrete logarithm-based method for predicting bit-patterns in the LFSR/SR sequence were proposed. The authors of [17] describe an efficient algorithm based on the Gauss elimination procedure, for selecting multiple seed-polynomial pairs for LFSR-based BIST schemes.

Several other methods have been proposed for circuits with hard to detect faults [4, 9, 12, 36]. Some of them, [12, 36], are not suitable for at-speed testing since they insert logic between the LFSR and the CUT, while the technique proposed in [4] does not lead to efficient solutions in terms of hardware overhead cost [12]. To reduce the hardware overhead the technique in [9] compromises the fault coverage.

Another easy-to-implement and flexible approach was recently presented in [7, 8]. This technique is based on twisted-ring counters, is as simple to implement as a LFSR-based one, it can be employed both in test-per-clock and test-per-scan schemes and features a very small control logic for controlling the reseeding operation. The main disadvantage of this technique is that, since a twisted-ring counter cannot generate pseudorandom patterns, many seeds and hence many ROM bits and test vectors are required to fully test the CUT.

The drawback of the LFSR-based techniques is that the modifications required so as a register to operate as a LFSR during testing, may result in system performance degradation due to the additional multiplexers in the signal path. The performance degradation and the area required for BIST can be minimized if some of the original building blocks of the circuit are utilized to generate patterns and/or to compact test responses (functional BIST). General-purpose computing structures as well as digital signal processing circuits' datapaths and many other circuits contain accumulator-based units, implementing arithmetic functions such as addition, subtraction and multiplication. The suitability of these modules for test pattern generation and test response compaction in test-per-clock BIST schemes has been investigated in [10, 11, 14, 22, 31–35] and in [5, 22, 26, 27, 30] respectively. Once again, the problem of unacceptably long test sequences for fully testing circuits with random pattern resistant faults occurs. To cope with this problem, four approaches have been proposed [10, 11, 32, 33]. Stroele [33] presents a method for suitably choosing the seed and the constant value of an accumulator such that the cardinality of the test set for a set of hard-to-detect faults to be minimized. However the cardinality of the test set still remains large. The approach of [32] is suitable for easily random testable circuits and it is based on reseeding. Forward and reverse fault simulation are used to find windows of effective test patterns and determine the seeds of the accumulator. The methods described in [10] and [11] are reseeding methods based on Genetic Algorithms and several techniques on the set covering problem respectively and can be applied to TPGs based on functional modules as well as on LFSRs. However, when the functional unit TPG approach is combined with the reseeding method in the way that it is proposed in [10] and [11], one of the main advantages of the functional BIST, i.e. that no delay is inserted in the signal paths of the circuit, due to the required modifications, is canceled. Multiplexers must be added in the input registers of the functional unit used as TPG for accomplishing the reseeding process.

In this paper we present a novel reseeding technique for test-per-clock BIST schemes suitable for at-speed testing. The reseeding takes place on-the-fly by inverting the logic value of some of the bits of the next state of the TPG. Our technique is suitable for LFSR as well as for accumulator-based TPGs and achieves complete fault coverage with shorter test sequences and, in most cases, less hardware overhead than the already known ROM-based reseeding techniques. Furthermore, in the case of accumulator-based TPGs, our method has the additional advantage over [10] and [11] that does not cause system performance degradation.

The remaining of the paper is organized as follows: Sections 2 and 3 present respectively the architecture and the reseeding algorithm for the proposed technique. In Section 4 the effectiveness of the proposed technique is evaluated with experimental results and comparisons are made with previously presented works. Conclusions are given in Section 5.

2. The Proposed Architecture

The general scheme of our TPG is shown in Fig. 1. It consists of a register R that holds the current state of the TPG and the Next State Specification Logic (NSSL) which determines the next state. The outputs of the register are fed to the CUT as well as to the NSSL. The key point of the proposed architecture is to invert, at specific cycles of the Test Pattern Generation, some of the outputs of the register before feeding them back to the NSSL. This is shown in Fig. 2.

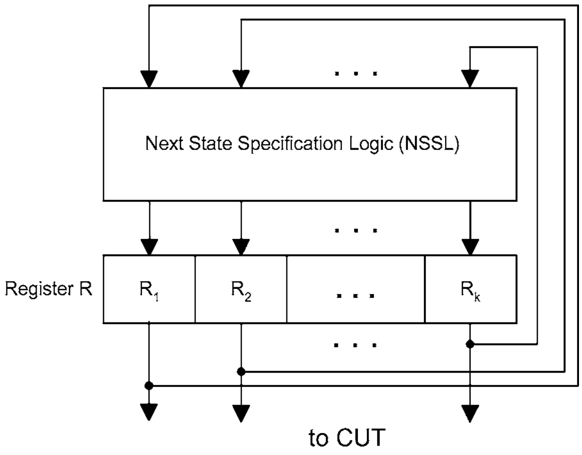

Fig. 1. General scheme of TPG.

The outputs of register R drive the NSSL inputs through XOR gates, which form the Inversion Module. The Inversion Control Module is responsible for controlling the inversion operation by setting the inputs of the XOR gates to the logic value 1. As will be shown by experimental results, not all k inputs of the NSSL need to be inverted in order for the necessary seeds to be produced. Therefore m XORs, with $m < k$, are sufficient for producing all the seeds needed to completely test the circuit under test.

This inversion operation is equivalent to a reseeding operation of the TPG since the new state produced by the NSSL is different from the state that would have been produced if no inversion had taken place. Note

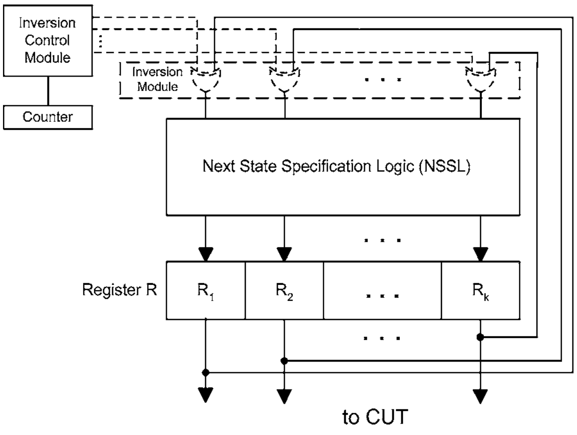

Fig. 2. Proposed TPG architecture.

that the Inversion Module is not placed between the TPG and the CUT and that the new state of the register is produced while the previous vector (state) is applied to the CUT. Since generally the delay of the CUT is much larger than the delay of the NSSL, we conclude that the proposed architecture can perform at-speed testing.

Many well-known TPGs follow the general scheme of Fig. 1. Among them the LFSR and the accumulator have been chosen in order to evaluate the effectiveness of the proposed scheme of Fig. 2. In the sequel we analyze these two cases.

2.1. LFSR-Based TPG

The architecture of the proposed LFSR-based TPG is given in Fig. 3. It consists of a LFSR (k 2-port register cells R_1, $R_2, \ldots,$ R_k and the Linear Feedback Logic), the Inversion Module (the XOR gates drawn using dashed lines in Fig. 3), the Inversion Control Module and a counter. The NSSL is formed by the Linear Feedback Logic of the LFSR and the direct connections between cells R_{i-1} and R_i ($i \neq 1$). We can see that one of the inputs of each of the XORs of the Inversion Module is driven by the output of the previous register cell and the other input by an output C_x of the Inversion Control Module.

In normal mode of operation the register is loaded from the functional block. In test mode, for $C_1 = C_2 = \cdots = C_k = 0$, the LFSR, after its initialization, changes state at each clock cycle, according to its feedback structure. Reseeding can take place in clock time t_i by setting, in clock time t_{i-1}, the lines $C_1, C_2, \ldots, C_k$ to the suitable values. If the bit j of the state that will be loaded in the register in time t_i has to be inverted, then a 2-input XOR gate must exist before the cell R_j of the LFSR (one of the dashed XOR gates of Fig. 3). In time t_{i-1} the control line C_j that drives that XOR gate is set to 1 by the Inversion Control Module and the value of bit j is inverted before being stored to the R_j stage.

The Inversion Control Module is responsible for generating the required signals on the control lines C_x. It receives the output of a counter, which counts the vectors generated by the LFSR, and sets each control line C_x to either 0 or 1 depending on the number of the current vector. We note that the same counter can be used for the generation of the test-end signal. During the normal operation, the values of the control lines are don't cares since the register is loaded with values from the functional block.

Example 1. Consider that the CUT has 4 inputs and that we have a 4-bit LFSR with initial seed 1010. The sequence that is generated by the LFSR is shown at the left side of Fig. 4 (the LFSR implements the characteristic polynomial $x^4 + x + 1$). Consider also that the easy faults of the circuit are detected by the first 3 vectors of that sequence, while the remaining faults, which are hard-to-detect, need the vectors $0x01$, 1000 and 1111 in order to be tested (x denotes a don't care value). We observe that, without reseeding, the first 12 vectors are required for testing the circuit. If during the generation of state 3 of the LFSR, we invert the bit that will be stored to the third cell (B$_3$), state 0001, instead of 0011, will be generated (see the right side of Fig. 4). This state detects the fault that needs vector $0x01$ to be tested. At the next clock cycle the required vector 1000 will be produced. During the generation of vector 5, with the inversion of the third and the fourth bit of

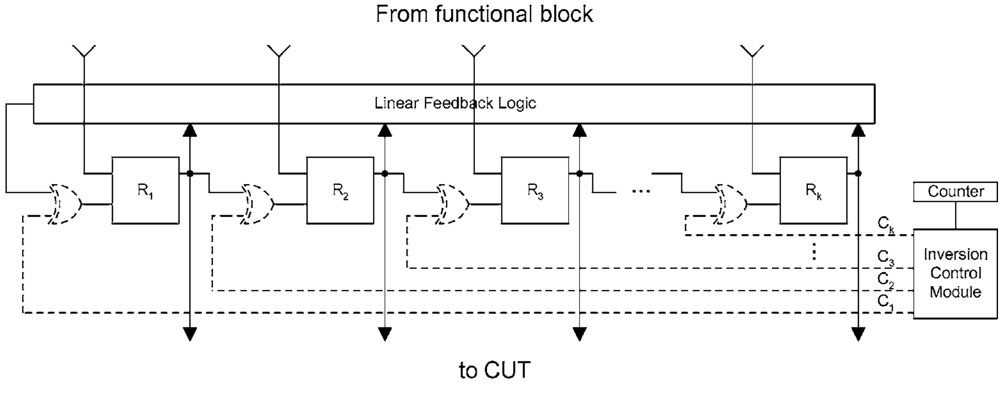

Fig. 3. The proposed LFSR-based TPG scheme.

Vector	LFSR Sequence without Reseeding			
	B_1	B_2	B_3	B_4
0	**1**	**0**	**1**	**0**
1	**1**	**1**	**0**	**1**
2	**0**	**1**	**1**	**0**
3	0	0	1	1
4	1	0	0	1
5	0	1	0	0
6	0	0	1	0
7	**0**	**0**	**0**	**1**
8	**1**	**0**	**0**	**0**
9	1	1	0	0
10	1	1	1	0
11	**1**	**1**	**1**	**1**
12	0	1	1	1
13	1	0	1	1
14	0	1	0	1

Vector	LFSR Sequence with Reseeding				
	B_1	B_2	B_3	B_4	
0	**1**	**0**	**1**	**0**	
1	**1**	**1**	**0**	**1**	
2	**0**	**1**	**1**	**0**	
3	**0**	**0**	**0**	**1**	Reseeding
4	**1**	**0**	**0**	**0**	
5	**1**	**1**	**1**	**1**	Reseeding

Fig. 4. Example LFSR sequence.

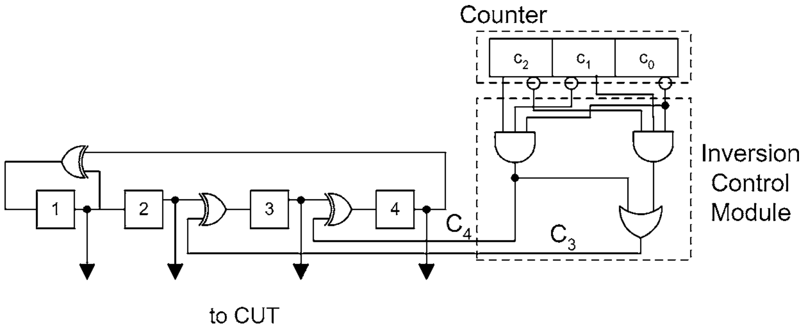

Fig. 5. Example LFSR-based TPG.

the LFSR (B_3 and B_4 respectively), vector 1111 will be derived. We can see that the reseeded LFSR covers all faults within 6 clock cycles, while without reseeding 12 cycles were needed. The implementation of the proposed TPG, that is the vector counter, the Inversion Control Module and the LFSR along with the XORs that perform the inversions is shown in Fig. 5.

We presented the proposed scheme using a LFSR with the feedback implemented with external XOR gates. It is obvious that our scheme can also be used when the XOR gates of the feedback are located between the stages of the register, as shown in Fig. 6 (the LFSR implements the characteristic polynomial $x^6 + x + 1$).

2.2. Accumulator-Based TPG

The architecture of the proposed accumulator-based TPG is shown in Fig. 7. The accumulator consists of a k-bit register and a circuit performing addition or subtraction. In the sequel we consider only the addition but the applicability of the proposed scheme is straightforward for the subtraction operation too. The Inversion Control Module is placed between the inputs of the adder and the outputs of the register.

It has been reported in [34] that accumulators with stored carry feedback perform better than the accumulators without carry feedback. For that reason we suppose that the carry out of the adder is stored in a flip-flop and is used in the next clock cycle. After being initialized, the accumulator adds in every clock cycle the constant value, the current contents of the register and the content of the flip-flop and stores the result back to the register.

Example 2. Consider that the circuit under test has 4 inputs and that we have a 4-bit accumulator with initial seed 0111 and constant value 0111. The sequence of test vectors that are generated by the accumulator

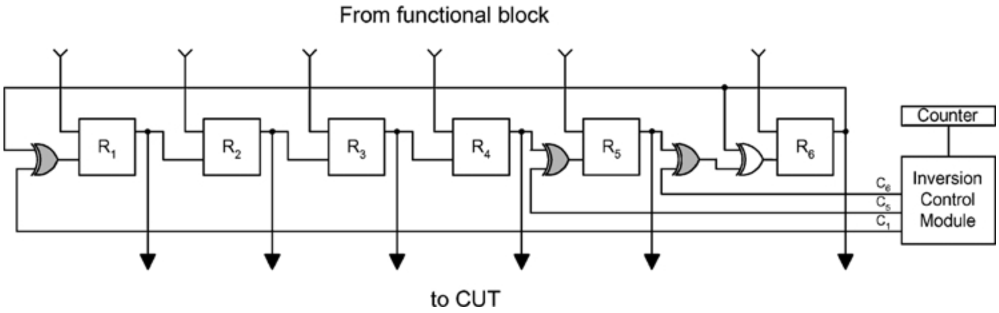

Fig. 6. The proposed TPG using a LFSR with internal XORs.

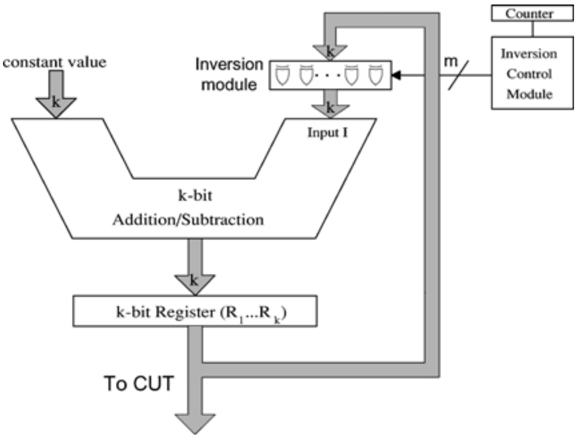

Fig. 7. The proposed accumulator-based TPG circuit.

is given at the left side of Fig. 8. Consider that our circuit has 3 hard-to-detect faults that are tested by the vectors $x011$, 0001 and 1001. Also consider that all the easy to detect faults are detected by the first two vectors of the sequence generated by the accumulator. We can see that, without reseeding, the first twelve vectors are required for testing our circuit. Now suppose that we invert the value of R_2 line during the addition for the generation of vector 2. Then vector 0011 will be generated which covers the desirable vector $x011$ (see the right side of Fig. 8). In the same way, the inversion of the value of lines R_2 and R_4 at the next addition, generates the vector 0001. The next vector that will be generated by the accumulator is the vector 1001. We can easily see that all the faults are now detected by the first 5 vectors produced by the reseeded accumulator. The implementation logic, that is the counter, the Inversion Control Module and the Inversion Module are shown in Fig. 9 (lines I_1 to I_4 are the adder inputs). We assume that in normal mode the counter remains at the 00 state (vector 0), so $C_2 = C_4 = 0$.

We note that when the accumulator is used to test more than one circuits, the inversion control logic can be easily designed in such a way so as to generate the m control signals taking into account all the under test circuits.

It is obvious that the selection of the points at which the TPG will be reseeded, is crucial to the hardware overhead imposed by the proposed architecture. In the following section we present an efficient algorithm that selects the reseeding points and the proper seed at each point, so as to minimize the overall hardware overhead.

3. Reseeding Algorithm

According to the proposed method, the test sequence consists of the parts P_0, P_1, P_2, P_3, … as shown in Fig. 10. Each one of these parts is comprised of successive vectors produced by the TPG, while the first vector of each part, except P_1, is a new seed produced by inverting some of the bits of the register (shaded areas in Fig. 10).

The flowchart of the algorithm is given in Fig. 11. Its main objective is to select the parts P_i and the corresponding seeds effectively, in order to minimize the cardinality of the test set and the required hardware, that is, the Inversion Control Module and the XOR gates needed for realizing the inversions. The outline of the algorithm of Fig. 11 is similar to that of the algorithm we recently presented in [18] for accumulator-based TPGs. However, the algorithm proposed in this paper uses a different cost function in the heuristic process of selecting the best points for reseeding and a different approach for determining the maximum length of each P_i part ($i > 0$), thus leading to better results as far as

Vector	Accumulator Sequence without Reseeding
	$R_4R_3R_2R_1$
0	**0 1 1 1**
1	**1 1 1 0**
2	0 1 0 1
3	1 1 0 1
4	0 1 0 0
5	1 1 0 0
6	**0 0 1 1**
7	1 0 1 1
8	0 0 1 0
9	1 0 1 0
10	**0 0 0 1**
11	**1 0 0 1**
12	0 0 0 0
13	1 0 0 0
14	1 1 1 1
.	.
.	.
.	.

Vector	Accumulator Sequence with Reseeding	
	$R_4R_3R_2R_1$	
0	**0 1 1 1**	
1	**1 1 1 0**	
2	**0 0 1 1**	Reseeding
3	**0 0 0 1**	Resseding
4	**1 0 0 1**	

Fig. 8. Example accumulator sequence.

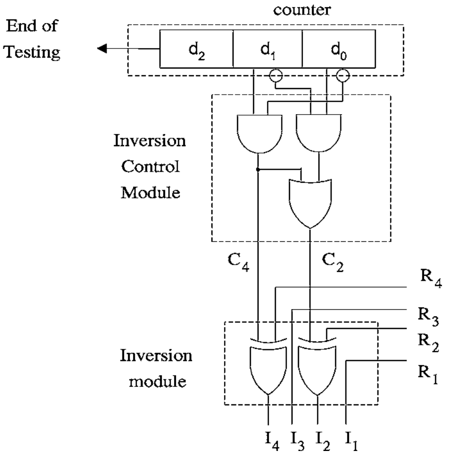

Fig. 9. Inversion and Inversion Control Modules of the example accumulator-based TPG.

both the hardware overhead and the cardinality of the test set are concerned.

The initial state of the TPG is set to a random value and the first vectors produced constitute the set P_0, which is capable of detecting all easy-to-detect faults. Specifically, successive test vectors are generated until the last T of them fail to detect any additional faults. All vectors generated, excluding the last T, form the set P_0. We note that P_0 detects the vast majority of the faults of the CUT. Let f_0 be the faults of the CUT, which cannot be detected by the vectors of set P_0. We consider them as hard-to-detect faults. For each hard-to-detect fault we extract Q test vectors using a deterministic test pattern generation tool. Each test vector in the sequence is modified to a test cube with don't care bits.

After determining the part P_0, we have to select the rest of the sets P_i so as to cover the remaining hard-to-detect faults. We denote as SL_i the maximum length of set P_i, with $i \geq 1$. In other words, a reseeding operation can be performed after at most SL_i vectors from the previous reseeding or from the last vector of set P_0. The criteria we use to specify a reseeding point and how SL_i is determined are critical for the performance of our algorithm.

We will at first discuss how the algorithm selects, among the SL_i vectors of set P_i, the most suitable point for reseeding. Let HV be the set of all the test cubes extracted for testing the hard-to-detect faults. First, all SL_i states of a set P_i are generated by the TPG and examined in order to find out if some of them match

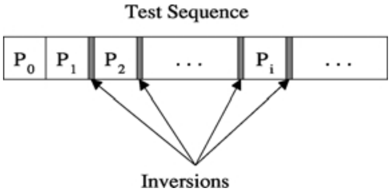

Fig. 10. Test sequence.

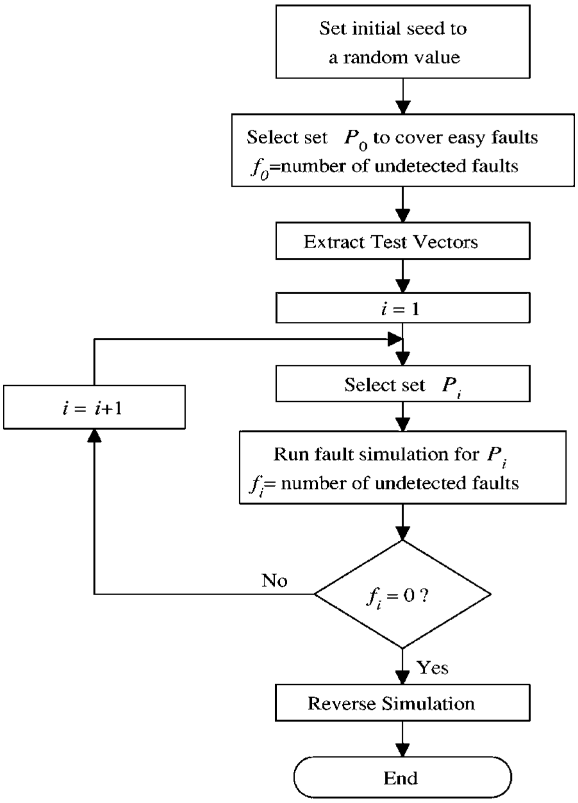
Set initial seed to a random value
Select set P_0 to cover easy faults
f_0=number of undetected faults
Extract Test Vectors
$i = 1$
Select set P_i
$i = i+1$
Run fault simulation for P_i
f_i= number of undetected faults
No
$f_i = 0$?
Yes
Reverse Simulation
End

Fig. 11. The proposed algorithm's flowchart.

any test cubes of *HV* without a reseeding. If this is the case, we select the last TPG state that is compatible with a test cube as the first vector of the next set P_{i+1}. If there is no TPG state that matches a test cube of *HV* without reseeding, we select as the next seed of the TPG the test vector of *HV* that requires from a TPG state s_i within SL_i the fewest bit inversions in order to be matched. As reseeding point we select the state s_i (s_i is the first vector of P_{i+1}). If two or more vectors require the same minimum number of bit inversions, we choose the one that needs the smallest number of additional XOR gates in order to be generated in the TPG sequence.

In order to check how many and which bit inversions are required for a LFSR state s_i to match a test cube $v \in HV$, we use the following procedure: we compare v against s_i bit-by-bit, starting from the least significant one (LFSR stage 1). If at some point, let us say at the j-th bit, v and s_i do not match, that is their j-th bits are both defined and complementary, then, for s_i to match v, an inversion must occur at the $j-1$ bit of state s_{i-1} (stage k is considered to be the previous of stage 1). Thus we invert bit $j-1$ of s_{i-1}, we re-produce the i-th LFSR state (s_i') and we follow the same procedure for s_i' and v starting from bit $j+1$. The same method is used for the case of accumulator-based TPGs with the difference that if vector v and state s_i do not match at bit j, we invert the j-th bit of s_{i-1}.

Example 3. Consider the case of a 4-bit adder-based accumulator used as TPG and assume that $s_i = 0111$, $v = 10x1$, $s_{i-1} = 0100$, constant value $= 0011$ and that the carry input added to s_{i-1} is equal to 0. Considering as least significant bit the rightmost one, we observe that with respect to the first two bits, state s_i matches v. For complementing the third bit of s_i, the third bit of s_{i-1} must be inverted. Thus, s_{i-1} becomes 0000 and $s_i' = 0011$. Again, the fourth bit of s_{i-1} must be inverted and therefore s_{i-1} becomes 1000 and, with 2 bit inversions, s_i finally matches test cube v. So, in order for the i-th state of the TPG to match test cube v, we should invert bits 3 and 4 of state $i - 1$, before feeding them to the adder.

The value of SL_i, $i > 0$, can be determined in two ways. According to the first one, we use a user-defined parameter called *MAXVECTORS*, which declares the maximum acceptable number of test vectors required to fully test the CUT. For each set P_i ($i > 0$), SL_i can be derived by the formula:

$$SL_i = MV_{i-1}/f_{i-1} \tag{1}$$

where f_{i-1} is the number of the remaining hard-to-detect faults after $i - 1$ reseedings and MV_{i-1} is the number of vectors which can be applied for detecting these f_{i-1} faults. That is:

$$MV_{i-1} = MAXVECTORS - \sum_{j=0}^{i-1} |P_j|$$

When the value of SL_i is derived by the above formula, the final test length strongly depends on the value of *MAXVECTORS*.

If the circuit under test has random-pattern resistant faults, then the vectors generated between two successive reseedings can detect very few if not zero additional faults than those detected by just the two seed vectors. In such cases, we observed that if we choose the value of SL_i directly equal to 1, 3 or 5 (SL_i is constant for all P_i sets with $i > 0$), we can get smaller test sets with less hardware overhead. The latter can be attributed to the fact that the tool that synthesizes the control logic exploits the fact that the distances between the reseedings are small, to make groupings in the control logic and therefore the final hardware overhead is much smaller than in the case that SL_i is calculated by relation (1).

After determining a new part P_i of our test set, with the value of SL_i derived either by relation (1) or defined by the user, we run fault simulation so as to drop all faults detected by P_i. At this step of our algorithm we also update the set *HV* by throwing out all the vectors that test the faults detected by set P_i and we calculate the values f_i and MV_i (when SL_i is not user-defined) for the next part P_{i+1}.

Some of the easy-to-detect faults that are tested with the test vectors of set P_0 can also be detected by some test vectors of the sets $P_1, P_2, \ldots$ Therefore some of the first vectors of the test sequence may be redundant. In order to minimize the cardinality of the test set, reverse simulation [32] is performed after determining all the P_i sets. That is, we fault simulate all the test vectors in reverse order and if, at some point (let us say at vector v), we reach 100% fault coverage, we exclude the remaining vectors from the test sequence and we set v as the new initial seed.

4. Experimental Results

In order to evaluate the effectiveness of the proposed technique, we implemented the algorithm described in Section 3 in C programming language and performed several simulations for both LFSR-based and accumulator-based TPGs. The ISCAS '85 and the combinational part of ISCAS '89 benchmarks circuits were used as CUT. The primitive polynomials for the LFSRs used in the experiments were taken from [6].

Table 1. Results for 3 different values of *MAXVECTORS* for the LFSR-based TPGs.

Circuit	Number of primary inputs	*MAX-VECTORS*	Inverting XORs	Number of test vectors	Hardware overhead (gate equiv.)
c880	60	1000	15	472	52
		1500	14	1117	46
		1900	8	980	29[a]
c1355	41	700	15	1046	24[a]
		1000	10	1416	15
		1200	13	1449	18
c1908	33	2000	19	2825	54
		3000	17	2896	56
		4000	6	3107	33
c2670	233	5000	83	2553	425
		10000	87	6035	440
		12000	82	1281	414
c7552	207	5000	187	4151	1014
		10000	188	8271	1057
		12000	181	7614	1051
s420	34	8000	21	2505	150
		10000	21	4133	158
		12000	21	9276	149
s641	54	2000	20	1722	63
		3000	19	2355	67
		4000	18	3128	69
s713	54	2000	18	2082	61
		3000	16	1666	65
		4000	16	2839	69
s820	23	450	14	458	211
		500	17	496	208
		520	18	515	198
s838	66	8000	57	1702	422
		10000	58	3537	423
		12000	58	1935	437
s953	45	3000	16	2807	84
		5000	8	2733	87
		7000	6	4210	58
s1196	32	10000	8	6767	77
		15000	4	11553	49
		18000	5	10419	53[a]
s1238	32	5000	18	4914	99
		8000	6	7246	80
		9000	9	7257	77[a]
s1423	91	1000	32	737	111
		1300	24	1096	99
		1500	26	1155	101

(*Continued on next page.*)

Tables 1 and 2 present results for the LFSR-based TPGs, while Tables 3 and 4 present results for the accumulator-based TPGs. In Tables 1, 3 and 2, 4 we present results for the cases that *MAXVECTORS* (maximum number of vectors to test the CUT) and SL_i (maximum number of vectors between two consecutive

Table 1. (*Continued*).

Circuit	Number of primary inputs	*MAX-VECTORS*	Inverting XORs	Number of test vectors	Hardware overhead (gate equiv.)
s5378	214	6000	34	7646	173
		8000	41	9909	178
		10000	27	8802	180
s9234	247	3000	137	11950	1060
		5000	138	11159	1030
		10000	135	13771	1016

[a] Denotes the best results between Tables 1 and 2 in terms of hardware overhead. Among results with similar hardware overhead, the one with the shortest test sequence is chosen.

reseedings) are used as user-defined parameters respectively. In the case that *MAXVECTORS* is used as a user-defined parameter, SL_i is given by relation (1). Each result presented is the best out of 10 trials. For each trial the initial seed was randomly selected. In the fourth and the fifth column of these tables we give the number of the XOR gates that must be inserted in order to produce the seeds and the total number of test vectors required to achieve complete (100%) fault coverage for single stuck-at faults respectively. The sixth column shows the hardware overhead required by the inverting XORs and the Inversion Control Module. The hardware overhead is given in terms of gate equivalents, assuming that 1 gate equivalent corresponds to a 2-input NAND gate. The test vector counter has not been taken into account in the derivation of the hardware overhead, since, in the majority of the LFSR and accumulator-based TPG schemes, such a counter is used. Also, for the LFSR-based TPG, the cost of the modification of a register to a shift register has not been taken into account in the hardware overhead, since the same modifications are required by any test-per-clock LFSR-based TPG and by the twisted-ring counters approach of [7] to which we compare. We note that for the majority of the experiments we performed, we set parameter T (has been defined in Section 3) to the value 500. In some cases where we wanted to produce smaller test sequences (e.g. s820 for LFSRs), T was set to 300.

From Tables 1 and 3 we can see that, in general, the total number of vectors required for testing the CUT (fifth column), does not exceed the value of parameter *MAXVECTORS*. In some cases this is not true as a consequence of the fact that the value that has been given to *MAXVECTORS* is small compared to the vectors needed by the algorithm in order to detect the easy faults. Since the majority of the circuits used as benchmarks have random-pattern resistant faults, from Tables 1–4 we can easily verify that by using SL_i as the user-defined parameter, we get, in most cases, better results with respect to the hardware overhead and the cardinality of the test set. We should finally note that when SL_i is user-defined, the test length strongly depends on the number of vectors needed for detecting the easy-to-detect faults. Hence we cannot assert that for greater values of SL_i we get longer test sequences.

As the value of *MAXVECTORS* increases, more efficient solutions in terms of hardware overhead are obtained. This is not true in many cases of Tables 1 and 3, since, each experiment was performed using a random initial seed. If for the same circuit we had used the same initial seed, then the results would have been like those depicted in Fig. 12 for the s1196 benchmark

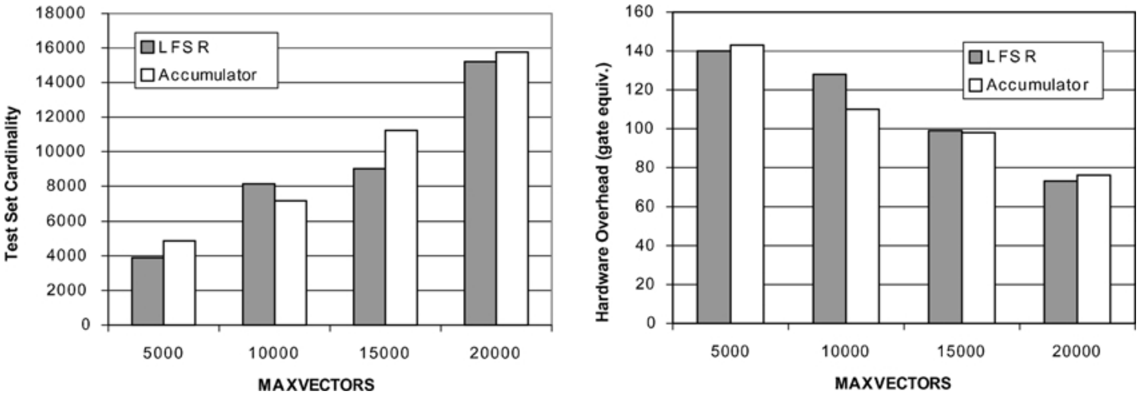

Fig. 12. Hardware overhead—test length trade-off for s1196.

Table 2. Results for 4 different values of SL_i for the LFSR-based TPGs.

Circuit	Number of primary inputs	SL_i	Inverting XORs	Number of test vectors	Hardware overhead (gate equiv.)
c880	60	1	20	720	42
		3	11	868	40
		5	12	550	45
		10	11	719	36
c1355	41	1	29	1180	42
		3	14	1239	18
		5	13	1295	18
		10	10	1186	15
c1908	33	1	23	2880	49
		3	17	2405	58
		5	13	3327	25[a]
		10	25	1799	82
c2670	233	1	74	1002	373[a]
		3	84	1192	408
		5	81	949	411
		10	83	1158	425
c7552	207	1	184	3958	1012[a]
		3	181	3808	1016
		5	187	3384	1045
		10	191	4058	1055
s420	34	1	28	1876	125[a]
		3	28	953	152
		5	29	1541	143
		10	25	1219	148
s641	54	1	15	1565	48
		3	20	1705	63
		5	19	1084	61[a]
		10	19	2143	58
s713	54	1	23	2148	67
		3	17	1826	62
		5	19	1963	59[a]
		10	21	1659	66
s820	23	1	18	498	182[a]
		3	19	524	194
		5	17	494	186
		10	13	760	174
s838	66	1	57	1625	423
		3	60	1844	426
		5	63	1929	420
		10	61	1223	423[a]
s953	45	1	11	3147	54[a]
		3	13	4963	64
		5	16	3436	83
		10	13	3576	83
s1196	32	1	16	6266	89
		3	16	4063	76
		5	15	4728	86
		10	18	3880	116
s1238	32	1	20	7788	83
		3	17	5151	102
		5	16	7356	91
		10	14	4773	97
s1423	91	1	39	1102	71[a]
		3	35	1185	83
		5	30	1085	95
		10	32	942	97
s5378	214	1	43	9222	188
		3	36	8469	143[a]
		5	32	7403	164
		10	34	7473	176
s9234	247	1	144	11207	948[a]
		3	138	14658	995
		5	136	12726	1039
		10	138	18498	1013

[a] Denotes the best results between Tables 1 and 2 in terms of hardware overhead. Among results with similar hardware overhead, the one with the shortest test sequence is chosen.

circuit. These results were taken by using the same random initial state in all four *MAXVECTORS* cases. In this figure, the trade-off between the cardinality of the test set and the hardware overhead of the application of the proposed technique is obvious. By increasing the value of parameter *MAXVECTORS*, smaller TPGs that require more test application time for fully testing the CUT are derived. The designer can choose the solution that better suits to his/her case.

Applying their reseeding method, the authors of [10] have come to the conclusion that the efficiency of the reseeding technique in accumulator-based TPGs is as high as its efficiency in the classical LFSR-based TPGs. From Tables 1–4 we reach to the same conclusion. For similar values of the input parameters, we obtain similar results for both the required test vectors and the imposed hardware overhead (see for example c1355, s5378, s9234). Therefore, Tables 1–4 further validate

Table 3. Results for 3 different values of *MAXVECTORS* for the accumulator-based TPGs.

Circuit	Number of primary inputs	*MAX-VECTORS*	Inverting XORs	Number of test vectors	Hardware overhead (gate equiv.)
c880	60	1000	12	999	33[a]
		1500	15	1634	27
		1900	9	1341	39
c1355	41	700	24	1102	38
		900	18	1072	28
		1100	8	1090	13[a]
c1908	33	2000	16	3413	35
		3000	14	3738	26
		4000	11	3321	33[a]
c2670	233	5000	103	2233	581
		10000	93	5557	561
		12000	90	6507	532
c7552	207	3000	187	5175	1097
		4000	189	3791	1106
		5000	194	5454	1081
s420	34	2000	29	3185	172
		4000	25	3442	189
		5000	25	4397	201
s641	54	3000	18	2668	63
		5000	14	3537	70
		6000	14	4180	61
s713	54	4000	17	2757	71
		7000	12	3227	59
		9000	14	2877	58
s820	23	3000	15	4490	48
		5000	5	4031	51
		10000	5	4986	36
s838	66	4000	65	3839	788
		6000	66	5690	782
		7000	66	6270	810
s953	45	3000	14	4552	64
		4000	14	3817	60
		9000	5	6346	59
s1196	32	5000	11	4806	68
		7000	12	6669	65
		10000	9	7400	76
s1238	32	5000	15	5702	87
		6000	16	6032	65
		7000	13	5652	80
s1423	91	2000	16	1515	48[a]
		3000	13	2131	53
		4000	12	3392	45
s5378	214	6000	39	8541	191[a]
		8000	42	7350	238
		10000	34	9109	202
s9234	247	3000	141	13784	1198
		5000	145	10223	1235
		10000	150	12570	1187

[a] Denotes the best results between Tables 3 and 4 in terms of hardware overhead. Among results with similar hardware overhead, the one with the shortest test sequence is chosen.

the statement that reseeding with accumulator-based TPGs is equally efficient to reseeding with LFSR-based TPGs.

In Tables 5 and 6 we compare the results of the proposed technique for LFSR-based TPGs against the mixed-mode twisted-ring counter approach of [7], the results obtained for LFSRs in [10] using genetic algorithms and the LFSR-based TPG scheme of [17]. Among the results given in Tables 1 and 2, we chose those with the smallest hardware overhead. Also, among results with similar hardware overhead, we chose the one requiring the fewest test vectors. We note that a dash (-) in the tables means that no results have been provided by the authors of the referenced paper for the corresponding benchmark circuit.

In Table 5 we compare the four techniques with respect to the number of test vectors they require for fully testing the CUT. We note that in [7] and in [17] the seeds are loaded serially in the register. The same assumption was made for [10] (no hint was given by its authors about this matter) since this approach is commonly used and results in the minimum hardware overhead. Thus, the test application time for the techniques of [10] and [17] is calculated by the formula:

$$\textit{Test Application Time}$$
$$= (\textit{Test Vectors} - \textit{Seeds}) x f_1 + (\textit{Seeds x PI}) x f_2,$$

where *Test Vectors* and *Seeds* are the number of the test vectors and seeds needed by each technique for fully testing the CUT, *PI* is the number of primary inputs of the corresponding circuit, f_1 is the functional frequency of the CUT and f_2 is the scan-in frequency of the seeds. For the technique of [7], the corresponding

Table 4. Results for 4 different values of SL_i for the accumulator-based TPGs.

Circuit	Number of primary inputs	SL_i	Inverting XORs	Number of test vectors	Hardware overhead (gate equiv.)
c880	60	1	12	1514	26
		3	11	1219	32
		5	12	1760	30
		10	17	895	40
c1355	41	1	15	1434	19
		3	13	1321	19
		5	16	1364	28
		10	12	1272	17
c1908	33	1	24	3890	40
		3	16	3321	50
		5	18	4568	41
		10	14	3275	54
c2670	233	1	107	1249	542[a]
		3	104	2081	576
		5	100	1507	598
		10	103	1558	611
c7552	207	1	192	3763	1019[a]
		3	190	3241	1109
		5	197	4592	1114
		10	187	4273	1074
s420	34	1	30	1492	183[a]
		3	31	1895	206
		5	30	3114	191
		10	31	3025	197
s641	54	1	21	1915	65
		3	23	2592	62
		5	23	2001	75
		10	23	1567	72[a]
s713	54	1	24	1608	74
		3	19	956	73[a]
		5	23	1056	75
		10	20	991	74
s820	23	1	12	2599	53[a]
		3	10	3749	39
		5	10	3882	35
		10	13	4998	48
s838	66	1	66	1592	765[a]
		3	65	3222	820
		5	66	2037	773
		10	64	1964	795
s953	45	1	13	1519	82[a]
		3	13	4956	52
		5	14	3796	81
		10	14	4188	80
s1196	32	1	14	4081	70[a]
		3	16	4846	83
		5	15	6187	66
		10	15	5574	87
s1238	32	1	17	4498	72[a]
		3	15	4675	88
		5	17	4544	95
		10	16	5971	63
s1423	91	1	48	1275	110
		3	31	3059	66
		5	17	2569	43
		10	27	2043	54
s5378	214	1	46	10144	165
		3	42	11399	163
		5	50	6316	224
		10	39	6947	218
s9234	247	1	148	10138	1186[a]
		3	143	16822	1163
		5	143	16471	1192
		10	141	13557	1191

Table 4. (*Continued*).

[a] Denotes the best results between Tables 3 and 4 in terms of hardware overhead. Among results with similar hardware overhead, the one with the shortest test sequence is chosen.

test application time is derived by the formula:

$$\begin{aligned} &\textit{Test Application Time} \\ &\quad = 3xSeeds\ xPIxf_1 + Seeds\ x\ PIxf_2, \end{aligned}$$

which characterizes the operation of the proposed TPG. However, in Table 5 we have not taken into account clock cycles needed for loading the seeds, since this loading may be done using a greater clock frequency than that used for applying the test vectors to the CUT. From Table 5 it is obvious that, although we do not take into account the reseeding time, our method favorably compares to those of [7, 10] and [17] (the reduction percentages are given in columns 4, 6 and 8). Specifically our method requires on average 67.2%, 40.3% and 47.3% fewer test vectors compared to [7, 10] and [17] respectively. The above percentages will become

Table 5. Test vector comparisons for the case of LFSR-based TPGs.

		Twisted-ring counters [7]		LFSR-based TPGs of [10]		Gauss elimination [17]	
Circuit	Proposed technique (test vectors)	Test vectors	Reduction (%)	Test vectors	Reduction (%)	Test vectors	Reduction (%)
c880	980	–	–	1829	46.4	1596	38.6
c1355	1046	–	–	1334	21.6	1447	27.7
c1908	3327	–	–	3759	11.5	3659	9.1
c2670	1002	58930	98.3	10206	90.2	7300	86.3
c7552	3958	76447	94.8	–	–	31282	87.3
s420	1876	10816	82.7	10843	82.7	5775	67.5
s641	1084	11458	90.5	2430	55.4	2345	53.8
s713	1963	11296	82.6	2759	28.6	2069	5.1
s820	498	–	–	527	5.5	6036	91.7
s838	1223	15742	92.2	9273	86.8	17526	93.0
s953	3147	10810	70.9	4834	34.9	7146	56.0
s1196	10419	11152	6.6	18776	38.5	7991	−23.3
s1238	7257	10864	33.2	7713	5.9	8185	11.3
s1423	1102	–	–	1308	15.7	2993	63.2
s5378	8469	10642	20.4	–	–	8400	−0.8
s9234	11207	–	–	–	–	108638	89.7

Table 6. Hardware overhead comparisons for the case of LFSR-based TPGs.

		Twisted-ring counters [7]			LFSR-based TPGs of [10]			Gauss elimination [17]		
Circuit	Proposed technique (gate equiv.)	ROM bits (gate equiv.)[a]	Control logic (gate equiv.)	Bit counter (gate equiv.)	ROM bits (gate equiv.)[a]	Control logic	Bit counter (gate equiv.)	ROM bits (gate equiv.)[a]	Control logic	P-LFSR + bit counter (gate equiv.)
c880	29	–	–	–	0	0	0	30	H	284
c1355	24	–	–	–	0	0	0	0	0	0
c1908	25	–	–	–	0	0	0	17	H	165
c2670	373	4019	38	27	1922	H	27	757	H	1052
c7552	1012	5486	38	27	–	–	–	880	H	938
s420	125	60	22	20	77	H	20	51	H	169
s641	61	108	26	20	81	H	20	68	H	257
s713	59	95	22	20	95	H	20	54	H	257
s820	182	–	–	–	196	H	16	23	H	117
s838	423	462	31	23	710	H	23	182	H	314
s953	54	56	22	20	45	H	20	45	H	218
s1196	53	88	26	16	32	H	16	32	H	157
s1238	77	64	26	16	40	H	16	48	H	157
s1423	71	–	–	–	91	H	20	68	H	424
s5378	143	0	14	27	–	–	–	214	H	969
s9234	948	–	–	–	–	–	–	1297	H	1114

[a] We have taken into account the assumption made in [17], that, on average, 0.25 gates are required for each memory cell of a ROM.

larger if we take into account the reseeding times required by those three techniques. We remind that the proposed technique produces the seeds on-the-fly.

As far as the hardware overhead comparison is concerned, we assume that for all four schemes, the initialization of the TPGs takes place by resetting their registers. Also, all of them, except for the proposed one, need a bit counter for controlling the serial loading of the seeds in the registers. The twisted-ring counter-based scheme of [7] imposes very small hardware overhead for the control module but it has to make use of many ROM bits. As a result, in the cases of circuits with many random pattern resistant faults (c2670, c7552, s838), despite of the very small control logic needed, extensive use of ROM must be made. For the rest of the cases the hardware overhead imposed by the proposed technique is similar to that of [7] without taking into consideration the ROM bits required for storing the seeds. The genetic algorithm approach of [10], although it reduces the number of the required seeds, it still needs a ROM, which imposes considerable hardware overhead. Also, the overhead H of the logic that controls the TPG scheme must be added to the overall hardware overhead required by this method. Since not enough information has been given in [10], we are unable to calculate this hardware overhead. Finally, the Gauss elimination approach of [17] uses a programmable LFSR (P-LFSR) that requires additionally to the logic of the register, two gates and one D flip-flop per LFSR stage. Thus, although it achieves to significantly reduce the use of ROM, this additional hardware overhead is big enough to make the approach of [17] worse than the proposed technique (except for the case of c1355 and possibly that of s820). From Table 6, excluding circuits c880, c1355 and c1908, we can see that our technique requires on average 34.9%, 21.9% and 59.1% less area overhead than the techniques given in [7, 10] and [17] respectively. Note that the above results would have been even more favorable to our technique if we could estimate the hardware overhead of the control logic (H) for the techniques proposed in [10] and [17]. Experiments with the small and relatively "easy-to-test" benchmark circuits c880, c1355 and c1908 show that the adoption of the proposed technique leads to significantly smaller test sequences with negligible additional hardware overhead.

The comparison of the proposed technique with the techniques given in [10] and [11] for adder-based TPGs is made in Tables 7 and 8. In this case we consider parallel loading of the seeds in the TPG registers of [10] and [11] since this results in minimum test application time as well as minimum hardware overhead. This

Table 7. Test vector comparisons for the case of accumulator-based TPGs.

		Adder-based TPGs of [10]		Adder-based TPGs of [11]	
Circuit	Proposed technique (test vectors)	Test vectors	Reduction (%)	Test vectors	Reduction (%)
c880	999	2104	52.5	3935	74.6
c1355	1090	1151	5.3	1816	40.0
c1908	3321	3773	12.0	3845	13.6
c2670	1249	10179	87.7	168072	99.3
c7552	3763	–	–	286725	98.7
s420	1492	5510	72.9	111899	98.7
s641	1567	4475	65.0	69473	97.7
s713	956	9082	89.5	–	–
s820	2599	5311	51.1	7075	63.3
s838	1592	6694	76.2	833217	99.8
s953	1519	7871	80.7	16855	91.0
s1196	4081	10000	59.2	–	–
s1238	4498	7356	38.9	15551	71.1
s1423	1515	3100	51.1	6916	78.1
s5378	8541	–	–	22848	62.6
s9234	10138	–	–	182100	94.4

Table 8. Hardware overhead comparisons for the case of accumulator-based TPGs.

Circuit	Proposed technique (gate equiv.)	Adder-based TPGs of [10] ROM bits (gate equiv.)[a]	Adder-based TPGs of [10] Control logic	Adder-based TPGs of [10] Multiplexers (gate equiv.)	Adder-based TPGs of [11] ROM bits (gate equiv.)[a]	Adder-based TPGs of [11] Control logic	Adder-based TPGs of [11] Multiplexers (gate equiv.)
c880	33	0	0	0	0	0	0
c1355	13	0	0	0	0	0	0
c1908	33	0	0	0	0	0	0
c2670	542	3728	H	559	1981	H	559
c7552	1019	–	–	–	3830	H	497
s420	183	102	H	82	51	H	82
s641	72	108	H	130	54	H	130
s713	73	108	H	130	–	–	–
s820	53	23	H	55	0	0	0
s838	765	330	H	158	2277	H	158
s953	82	45	H	108	0	0	0
s1196	70	48	H	77	–	–	–
s1238	72	48	H	77	16	H	77
s1423	48	91	H	218	0	0	0
s5378	191	–	–	–	214	H	514
s9234	1186	–	–	–	5558	H	593

[a] We have taken into account the assumption made in [17], that, on average, 0.25 gates are required for each memory cell of a ROM.

is due to the fact that in both cases (serial or parallel seed loading) n multiplexers, where n is the length of the register, are required, while in the serial case an additional bit counter is needed. With respect to the number of test vectors required for fully testing the CUT (see Table 7), the proposed method leads to substantially better results. In particular, it needs on average 57,1% and 77,4% fewer test vectors than the approaches of [10] and [11] respectively.

As far as the hardware overhead is concerned, as we mentioned above, both [10] and [11] require multiplexers in order for the seeds to be loaded in the TPG registers. Therefore, the proposed technique (see Table 8), in most cases, requires less hardware overhead than that of [10] and [11], even if we do not take into account the hardware overhead imposed by the control logic (we exclude the small combinational circuits c880, c1355 and c1908). If we also consider the area for the controlling logic (H), then the area overhead of the proposed technique is expected to be even smaller than the area overhead of [10] and [11]. We note that the method of [11], compared to that of [10], manages to test more circuits with only one seed (s820, s953, s1423) and in general reduces the storage requirements by reducing the number of the seeds that need to be stored. However, this is achieved at the expense of the test application time, which is significantly lengthened.

We have to remind that the hardware required for the reseeding applied in [10] and [11] may affect system performance while our method does not affect it, since the additional logic is added to a non-critical path of the system.

We finally note that the algorithm described in Section 3 is fairly simple and needs for each simulation from very few minutes (c880, c1355, s641) to approximately one hour for the bigger circuits (c7752, s9234), in a Pentium III, 500 MHz system, excluding the run-time for the ATPG. Although a home ATPG tool was used, it can be easily replaced by any other, more efficient, commercial tool.

5. Conclusion

Reseeding has been proposed as an effective technique for testing circuits with random-pattern resistant faults, since it can achieve complete fault coverage within an acceptable number of test vectors. In this paper a new reseeding technique for test pattern generation, suitable for test-per-clock, at-speed testing, BIST schemes, was proposed. The generation of the seeds is performed on-the-fly by the inversion of the logic values of some of the bits of the TPG's next state. Experimental results on the ISCAS '85 and the combinational part of ISCAS '89 benchmark circuits for both LFSR and accumulator based TPGs, showed that the proposed technique requires fewer test vectors and, in most cases, less hardware for its implementation than the other already known LFSR- and accumulator-based reseeding techniques, for complete testing of the circuits under test.

Acknowledgments

This research was financially supported by the Public Benefit Foundation "Alexander S. Onassis" via its scholarships programs and by the Research Committee of Patras University, within the framework of "K. Karatheodoris" scholarships programs.

References

1. M. Abramovici, M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, New York: Computer Science Press, 1990.
2. V. Agrawal, C. Kime, and K. Saluja, "A Tutorial on Built-In Self-Test Part 1: Principles," *IEEE Design & Test of Computers*, pp. 73–82, March 1993.
3. V. Agrawal, C. Kime, and K. Saluja, "A Tutorial on Built-In Self-Test Part 2: Applications," *IEEE Design & Test of Computers*, pp. 69–77, June 1993.
4. M.F. AlShaibi and C.R. Kime, "MFBIST: A BIST Method for Random Pattern Resistant Circuits," in *Proc. of International Test Conference*, 1996, pp. 176–185.
5. D. Bakalis, D. Nikolos, and X. Kavousianos, "Test Response Compaction by An Accumulator Behaving as a Multiple-Input Non-Linear Feedback Shift Register," in *Proc. of International Test Conference*, 2000, pp. 804–811.
6. P.H. Bardell, W.H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudo-Random Techniques*, New York: Johh Wiley & Sons, 1987.
7. K. Chakrabarty, B.T. Murray, and V. Iyengar, "Built-in Test Pattern Generation For High-Performance Circuits Using Twisted-Ring Counters," in *Proc. of 17th IEEE VLSI Test Symposium*, 1999, pp. 22–27.
8. K. Chakrabarty and S. Swaminathan, "Built-in Testing of High-Performance Circuits Using Twisted-Ring Counters," in *Proc. of IEEE International Symposium on Circuits and Systems*, 2000, pp. 72–75.
9. M. Chatterjee and D. Pradhan, "A Novel Pattern Generator for Near-Perfect Fault Coverage," in *Proc. of 13th IEEE VLSI Test Symposium*, 1995, pp. 417–425.
10. S. Chiusano, P. Prinetto, and H.J. Wunderlich, "Non-Intrusive BIST for Systems-on-a-Chip," in *Proc. of International Test Conference*, 2000, pp. 644–651.
11. S. Chiusano, S. Di Carlo, P. Prinetto, and H.J. Wunderlich, "On Applying the Set Covering Model to Reseeding," in *Proc. of Design, Automation & Test in Europe Conference*, 2001, pp. 156–160.
12. C. Fagot, P. Girard, and C. Landrault, "On Using Machine Learning for Logic BIST," in *Proc. of International Test Conference*, 1997, pp. 338–346.
13. C. Fagot, O. Gascuel, P. Girard, and C. Landrault, "On Calculating Efficient LFSR Seeds for Built-In Self Test," in *Proc. of IEEE European Test Workshop*, 1999, pp. 7–14.
14. S. Gupta, J. Rajski, and J. Tyszer, "Arithmetic Additive Generators of Pseudo-Exhaustive Test Patterns," *IEEE Trans. on Computers*, vol. 45, no. 8, pp. 939–949, Aug. 1996.
15. S. Hellebrand, S. Tarnick, B. Courtois, and J. Rajski, "Generation of Vector Patterns Through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," in *Proc. of International Test Conference*, 1992, pp. 120–129.
16. S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-In Test for Circuits with Scan Based on Reseeding of Multiple-Polynomial Linear Feedback Shift Registers," *IEEE Trans. on Computers*, vol. 44, no. 2, pp. 223–233, Feb. 1995.
17. L.R. Huang, J.Y. Jou, and S.Y. Kuo, "Gauss-Elimination-Based Generation of Multiple Seed-Polynomial Pairs for LFSR," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 16, no. 9, pp. 1015–1024, Sep. 1997.
18. X. Kavousianos, D. Bakalis, and D. Nikolos, "A Novel Reseeding Technique for Accumulator-Based Test Pattern Generation," in *Proc. of 11th ACM Great Lakes Symposium on VLSI*, 2001, pp. 7–12.
19. G. Kiefer and H. Wunderlich, "Using BIST Control for Pattern Generation," in *Proc. of International Test Conference*, 1997, pp. 347–355.
20. B. Koenemann, "LFSR-Coded Test Patterns for Scan Design," in *Proc. of European Test Conference*, 1991, pp. 237–242.
21. M. Lempel, S.K. Gupta, and M.A. Breuer, "Test Embedding with Discrete Logarithms," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 14, no. 5, pp. 554–566, May 1995.
22. N. Mukherjee, M. Kassab, J. Rajski, and J. Tyszer, "Arithmetic Built-In Self-Test for High Level Synthesis," in *Proc. of 13th IEEE VLSI Test Symposium*, 1995, pp. 132–139.
23. S.K. Mukund, E.J. McCluskey, and T.R.N. Rao, "An Apparatus for Pseudo-Deterministic Testing," in *Proc. of 13th IEEE VLSI Test Symposium*, 1995, pp. 125–131.
24. F. Muradali, V.K. Agarwal, and B. Nadeau-Dostie, "A New Procedure for Weighted Random Built-In Self-Test," in *Proc. of International Test Conference*, 1990, pp. 660–669.
25. I. Pomeranz and S.M. Reddy, "On Methods to Match a Test Pattern Generator to a Circuit-Under-Test," *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 6, no. 3, pp. 432–444, Sep. 1998.
26. J. Rajski and J. Tyszer, "Test Responses Compaction in Accumulators with Rotate Carry Adders," *IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12, no. 4, pp. 531–539, April 1993.
27. J. Rajski and J. Tyszer, "Accumulator-Based Compaction of Test Responses," *IEEE Trans. on Computers*, vol. 42, no. 6, pp. 643–650, June 1993.

28. J. Rajski and J. Tyszer, *Arithmetic Built-In Self-Test for Embedded Systems*, Upper Saddle River, New Jersey: Prentice Hall PTR, 1998.
29. J. Savir and W.H. McAnney, "A Multiple Seed Linear Feedback Shift Register," *IEEE Trans. on Computers*, vol. 41, no. 2, pp. 250–252, Feb. 1992.
30. A.P. Stroele, "Test Response Compaction Using Arithmetic Functions," in *Proc. of 14th IEEE VLSI Test Symposium*, 1996, pp. 380–386.
31. A.P. Stroele, "Arithmetic Pattern Generators for Built-In Self-Test," in *Proc. of International Conference on Computer Design*, 1996, pp. 131–134.
32. A.P. Stroele and F. Mayer, "Methods to Reduce Test Application Time for Accumulator-based Self-Test," in *Proc. of 15th IEEE VLSI Test Symposium*, 1997, pp. 48–53.
33. A.P. Stroele and F. Mayer, "Test Length Reduction for Accumulator-based Self-Test," in *Proc. of International Symposium on Circuits and Systems*, 1997, pp. 2705–2708.
34. A.P. Stroele, "BIST Pattern Generators Using Addition and Subtraction Operations," *Journal of Electronic Testing: Theory and Applications*, vol. 11, no. 1, pp. 69–80, Aug. 1997.
35. A.P. Stroele, "Synthesis for Arithmetic Built-In Self-Test," in *Proc. of 18th IEEE VLSI Test Symposium*, 2000, pp. 165–170.
36. N.A. Touba and E.J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," in *Proc. of International Test Conference*, 1995, pp. 674–682.
37. S. Venkataraman, J. Rajski, S. Tarnick, and S. Hellebrand, "An Efficient BIST Scheme based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers", in *Proc. of International Conference on Computer-Aided Design*, 1993, pp. 572–577.
38. J.A. Waicukauski, E. Lindenbloom, E.B. Eichelberger, and O.P. Forlenza, "A Method for Generating Weighted Random Patterns," *IBM Journal of Research and Development*, vol. 33, no. 2, pp. 149–161, March 1989.
39. H.J. Wunderlich, "BIST for Systems-on-a-Chip," *Integration, The VLSI Journal*, vol. 26, no. 1-2, pp. 55–78, Dec. 1998.

Emmanouil I. Kalligeros received the Diploma degree in 1999 and the M.Sc. degree in 2001 in Computer Engineering, both from the Department of Computer Engineering and Informatics at the University of Patras in Greece. He is now pursuing his Ph.D. in the area of Built-In Self-Test at the same department. His research interests also include delay fault testing and VLSI design.

Xrysovalantis I. Kavousianos received the Ph.D. degree in 2000 from the Department of Computer Engineering and Informatics at the University of Patras, Greece in the area of On-Line Testing. He is currently teaching VLSI Design in this department. He holds a Diploma degree in Computer Engineering from the same department. In 1997 he was awarded a scholarship from the Technical Chamber of Greece due to his excellent student records. He is also a member of the IEEE. His other research interests include VLSI design and Low-Power Testing.

Dimitris N. Bakalis received the Diploma degree in 1995 and the M.Sc. degree in 2000 in Computer Engineering, both from the Department of Computer Engineering and Informatics at the University of Patras in Greece. He is currently pursuing his Ph.D. degree at the same department in the area of Built-In Self-Test. His other research interests include Low Power VLSI design and test. He is a student member of the IEEE.

Dimitris Nikolos received the B.Sc. degree in Physics, the M.Sc. degree in Electronics and the Ph.D. degree in Computer Science, all from the University of Athens. He is currently a professor in the Department of Computer Engineering and Informatics of the University of Patras, director of the Hardware and Computer Architecture division and head of the Technology and Computer Architecture Laboratory. He has served as program co-chairman of the five (1997–2001) IEEE Int. On-Line Testing Workshops. He has also served on the program committees of the IEEE Int. Symposium on Defect and Fault Tolerance in VLSI systems (1997–1999), of the 3rd and 4th European Dependable Computing Conference and of the DATE (2000–2002) Conference. His main research interests are fault-tolerant computing, computer architecture, VLSI design, test and design for testability. Prof. Nikolos has authored or co-authored more than 110 scientific papers and was co-recipient of the Best Paper Award for his work "Extending the Viability of IPPQ Testing in the Deep Submicron Era" presented at the 3rd IEEE Int. Symposium on Quality Electronic Design (ISQED 2002). He is a member of the IEEE.