A New Reseeding Technique for LFSR-based Test Pattern Generation[†]

E. Kalligeros, X. Kavousianos, D. Bakalis and D. Nikolos

Dept. of Computer Engineering & Informatics, Univ. of Patras, 26500, Patras, Greece

Computer Technology Institute, 3, Kolokotroni Str., 26221 Patras, Greece

e-mail: kalliger@ceid.upatras.gr, kabousia@ceid.upatras.gr, bakalis@cti.gr, nikolosd@cti.gr

Abstract

In this paper we present a new reseeding technique for LFSR-based test pattern generation suitable for circuits with random-pattern resistant faults. Our technique eliminates the need of a ROM for storing the seeds since the LFSR jumps from a state to the required state (seed) by inverting the logic value of some of the bits of its next state. An efficient algorithm for selecting reseeding points is also presented, which targets complete fault coverage and minimization of the cardinality of the test set and the hardware required for the implementation of the test pattern generator. The application of the proposed technique to ISCAS '85 and the combinational part of ISCAS '89 benchmark circuits shows its superiority against the already known reseeding techniques with respect to the length of the test sequence and, in the majority of cases, the hardware required for their implementation.

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 counting up to several hundreds, strongly limit the controllability and observability of the embedded modules.

Built-In Self-Test (BIST) [1-5] 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, area overhead, and test data storage as well as minimal performance degradation and at-speed testing is essential for any successful BIST scheme. Furthermore, in most applications, complete (100%) fault coverage is desirable.

Linear Feedback Shift Registers (LFSRs) are 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, in circuits with random pattern resistant faults, high fault coverage cannot be achieved with an acceptable test length.

LFSR reseeding [6-12] has been proposed as a possible solution to cope with this drawback. In [6] a test-per-scan technique is presented where an LFSR is used to generate pseudorandom and deterministic patterns which are encoded as seeds. Test-per-scan techniques for generating test patterns through reseeding of multiple polynomial LFSRs were proposed in [7-9]. These techniques involve storing LFSR seeds in a ROM instead of storing the deterministic patterns that detect random pattern resistant faults. 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. In [10] a test-per-clock scheme based on a modified design of an LSSD-based LFSR is described. The proposed scheme is capable of changing seeds by the application of a pair of clock pulses at the time of change. The seeds cannot be predetermined, they are randomly selected and they have the property of being uniformly distributed over the entire LFSR pattern space. In [11], a scheme using a Shift Register driven by an LFSR (LFSR/SR) for the generation of pseudo-deterministic patterns, was proposed. Recently, a test-per-clock technique was presented [12] that, based on Genetic

Algorithms, computes the initial values for several general functional modules and LFSRs, so that they are able to produce test patterns with complete fault coverage.

However all of the above techniques (except for [10], which has the drawback that the seeds cannot be predetermined) suffer from the same problem. In the case of circuits with many random pattern resistant faults, a large number of seeds must be used. Therefore the hardware overhead can be very large if we take into account the necessary control module and the ROM that must be used to store the various seeds.

In this paper we present a novel reseeding technique for LFSR-based test pattern generation suitable for test-per-clock BIST schemes. The LFSR, beginning from an initial state, produces a new test pattern at each clock cycle. A new seed is produced on-the-fly by inverting the logic value of some of the bits of the register. The proposed technique achieves complete fault coverage with shorter test sequences and, in the majority of cases, less hardware overhead than the LFSR-based TPGs given in [12].

The remaining of the paper is organized as follows: Sections 2 and 3 present respectively the architecture and the reseeding algorithm for the proposed TPG. 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 architecture of the proposed TPG is given in Figure 1. The proposed TPG consists of an LFSR with k 2-port register stages R_1, R_2, ..., R_k, the exclusive-OR (XOR) gates of the feedback logic and a number of additional 2-input XOR gates distributed among the stages of the register (the XOR gates drawn using dashed lines in Figure 1). We can see that one of the inputs of

each of the additional XORs is driven by the output of the previous register stage and the other input by the output C_x of a block called Inversion Control Logic. In normal mode of operation the register is loaded from the functional block.

In test mode, for $C_1 = C_2 = \ldots = C_k = 0$, 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, we insert before the cell R_j of the LFSR, a 2-input XOR gate (one of the dashed XOR gates of Figure 1). In time t_{i-1} the control line that drives that XOR gate (C_j) is set to 1 by the Inversion Control Logic and the value of bit j is inverted before stored to the R_j stage.

The "Inversion Control Logic" is responsible for generating all 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 care since the register is loaded with the values from the functional block. As will be shown by experimental results, not all k bits of the LFSR need to be inverted in order for the necessary seeds to be produced. Therefore m XORs $(m < k)$ are sufficient for producing all the seeds needed to completely test the circuit under test.

For example, consider that the CUT has 4 inputs and we have a 4-bit LFSR with initial seed 1010. The sequence that is generated by the LFSR is shown in Figure 2 (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-
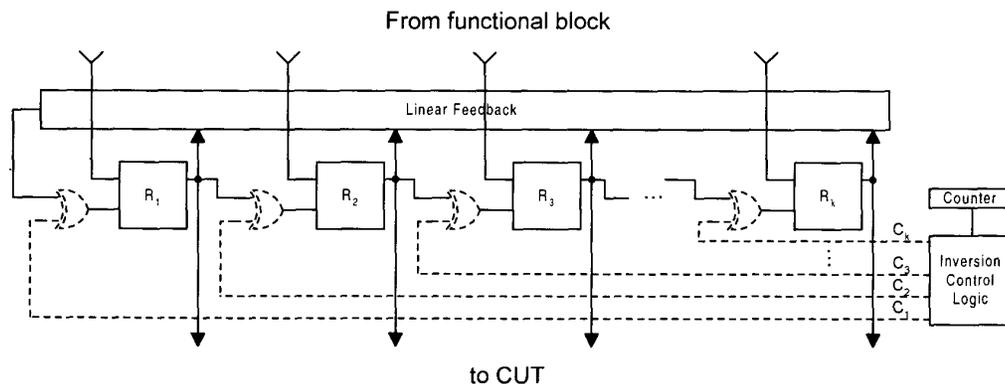
From functional block

Figure 1. The proposed TPG scheme

81

detect, need the vectors 0x01, 1000 and 1111 in order to be tested. We observe that, without reseeding, the first 12 vectors are sufficient to test the circuit. If, during the generation of vector 3, we invert the bit of the LFSR that will be stored to the third cell (B_3), the vector 0001, instead of the vector 0011, will be generated. This vector covers the fault that needs the vector 0x01 to be tested. In the same way, with the inversion of the third and the fourth bit of the LFSR (B_3 and B_4 respectively) during the generation of vector 5, vector 1111 is derived. Since vector 4 is 1000, 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 Logic and the LFSR along with the XORs that perform the inversions is shown in Figure 3.

Vector	LFSR Sequence without Reseeding				LFSR Sequence with Reseeding				
	B_1	B_2	B_3	B_4	B_1	B_2	B_3	B_4	
0	1	0	1	0	1	0	1	0	
1	1	1	0	1	1	1	0	1	
2	0	1	1	0	0	1	1	0	
3	0	0	1	1	0	0	0	1	Reseeding
4	1	0	0	1	1	0	0	0	
5	0	1	0	0	1	1	1	1	Reseeding
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					

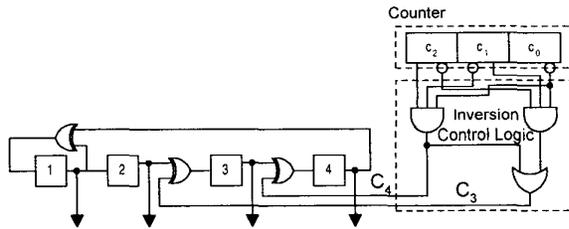**Figure 2.** Example sequence

to CUT

Figure 3. Example TPG

It is obvious that the selection of the points at which the LFSR 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.

We presented the proposed scheme using an LFSR with the feedback implemented by external XOR gates. It is obvious that our scheme can also be used when the XOR gates of the feedback are distributed between the stages of the register.

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 Figure 4. Each one of these parts is comprised of successive vectors produced by the LFSR, while the first vector of each part, except P_1, is a new seed produced by inverting some of the bits of the LFSR (shaded areas in Figure 4).

Test Sequence

Figure 4. Test sequence
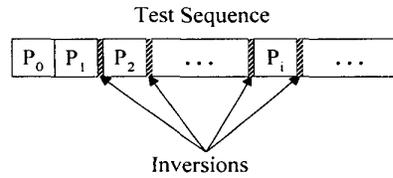
The flowchart of the algorithm is given in Figure 5. Its main objective is to select the parts P_i and the corresponding seeds effectively, in order to minimize the required hardware, that is, the Inversion Control Logic and the XOR gates needed for realizing the inversions. The initial state of the LFSR is set to a random value.
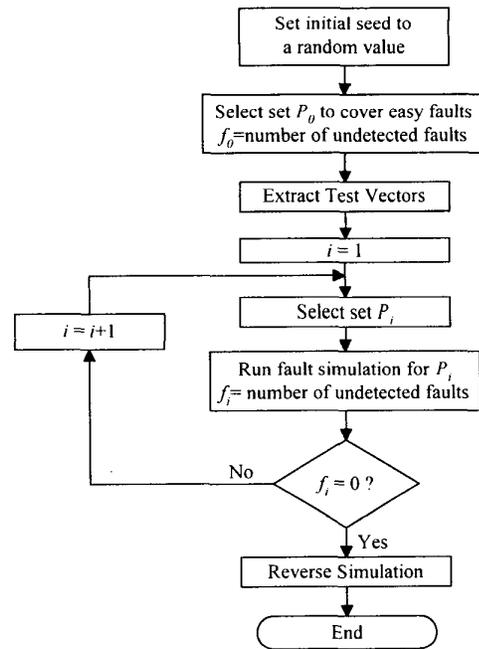

Figure 5. The proposed algorithm's flowchart

Let *MAXVECTORS* be a user-defined parameter, which declares the maximum acceptable number of test vectors required to fully test the CUT. The first vectors produced by the LFSR 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. Let f_0 be the faults of the CUT which are not detectable by the vectors of set P_0. We consider them as hard-to-detect faults. We set $MV_0 = MAXVECTORS - |P_0|$. MV_0 is the number of vectors that can be used for testing the f_0 hard-to-detect faults of the CUT. For each hard-to-detect fault we extract Q test vectors using a random test pattern generation program. In the case that for a fault f the random test pattern generation program fails to give at least one test vector, test vectors are derived using a deterministic test pattern generation tool. Each test vector in the sequence is modified to a test cube with as many don't care bits as possible.

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* the maximum length of each one of the sets P_i, with $i \geq 1$. That means that a reseeding operation can be performed after at most *SL* 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* is determined are critical for the performance of our algorithm.

We will at first discuss how the algorithm selects, among the *SL* 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* states of a set P_i are generated by the LFSR and examined in order to find out if some of them match any test cubes of *HV* without any reseeding. If this is the case, we select the last LFSR state that is compatible with a test cube as the first vector of the next set P_{i+1}. If there is no LFSR state that matches a test cube of *HV* without reseeding, we select as the next seed of the LFSR the test vector of *HV* that needs the less bit inversions in order, for an LFSR state s_i within *SL*, to match this test vector. As reseeding point we select the state s_i (s_i is the first vector of P_{i+1}). For example, if $HV = \{1x10, 110x\}$ and $SL = 2$ with $s_1 = 0110$ and $s_2 = 0011$, then for s_1 to match $1x10$ we need 1 bit inversion, for s_1 to match $110x$ we need 2 bit inversions, for s_2 to match $1x10$ we need 2 bit inversions and for s_2 to match $110x$ we need 3 bit inversions. Therefore we choose the test vector 1110 ($1x10$) as the new seed and the state s_1 as the reseeding point. If two or more vectors need the same minimum number of bit inversions, we choose the one that requires the smallest number of additional XOR gates in order to be formed by an LFSR state.

For each set P_i ($i > 0$), SL_i can be derived by the formula

$$SL_i = MV_{i-1} / f_{i-1} \qquad (1),$$

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. When the value of *SL* is derived by the above formula, the final test length is strongly dependent 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* directly equal to 1, 3 or 5, 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* is calculated by relation (1). When the value of *SL* is directly defined by the user, the parameter *MAXVECTORS* has no impact on the performance of our algorithm.

After determining a new part P_i of our test set, with the value of *SL* 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, if *SL* is not user-defined, we calculate the values f_i and MV_i 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, ... Therefore some of the first vectors of the test sequence can be redundant. In order to minimize the cardinality of the test set, reverse simulation [13] is performed after determining all the P_i sets and the initial seed is adjusted so as to exclude these redundant test vectors.

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 comparison reasons, we used the ISCAS '85 and the combinational part of ISCAS '89 benchmarks circuits.

In tables 1 and 2 we present results for the cases that *MAXVECTORS* (maximum number of vectors to test the CUT) and *SL* (maximum number of vectors between the previous and the next reseeding) are used as user-defined parameters respectively. In the case that *MAXVECTORS* is used as a user-defined parameter, *SL* is given by relation (1). Each result presented is the best out of 10 trials. In the third and the fourth 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 respectively. The fifth column shows the hardware overhead required by the inverting XORs and the Inversion Control Logic. The hardware overhead is given in terms of gate equivalents, assuming that 1 gate equivalent is a 2-input NAND gate. The test vector counter has not been taken into account in the derivation of the hardware overhead since every LFSR-based TPG use such a counter. Also the cost of the modification of a register to an LFSR has not been taken into account in the hardware overhead, since the same modifications are required by any test-per-clock LFSR-based TPG. In Tables 1 and 2 the best results with respect to the number of test vectors or the hardware overhead are shaded.

Table 1. Results for 3 different values of *MAXVECTORS*

Circuit	MAX VECTORS	Inverting XORs	Number of Test Vectors	Hardware Overhead (gate equiv.)
c880	1000	15	472	52
	1500	14	1117	46
	1900	8	980	29*
c1355	700	15	1046	24*
	1000	10	1416	15
	1200	13	1449	18
c1908	2000	19	2825	54
	3000	17	2896	56
	4000	6	3107	33
c2670	5000	83	2553	425
	10000	87	6035	440
	12000	82	1281	414
c7552	5000	187	4151	1014
	10000	188	8271	1057
	12000	181	7614	1051
s420	8000	21	2505	150
	10000	21	4133	158
	12000	21	9276	149
s641	2000	20	1722	63
	3000	19	2355	67
	4000	18	3128	69
s713	2000	18	2082	61
	3000	16	1666	65
	4000	16	2839	69
s820	450	14	458	211
	500	17	496	208
	520	18	515	198
s838	8000	57	1702	422
	10000	58	3537	423
	12000	58	1935	437
s953	3000	16	2807	84
	5000	8	2733	87
	7000	6	4210	58
s1196	10000	8	6767	77
	15000	4	11553	49*
	18000	5	10419	53
s1238	5000	18	4914	99
	8000	6	7246	80
	9000	9	7257	77*
s1423	1000	32	737	111
	1300	24	1096	99
	1500	26	1155	101
s5378	6000	34	7646	173
	8000	41	9909	178
	10000	27	8802	180
s9234	3000	137	11950	1060
	5000	138	11159	1030
	10000	135	13771	1016

*: The best results between Table 1 and Table 2

Table 2. Results for 4 different values of *SL*

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

*: The best results between Table 1 and Table 2

Table 3. Comparison of the proposed technique with the results of [12] for LFSRs (parallel reseeding)

Circuit	Number of Test Vectors			Hardware Overhead			
	Proposed technique	LFSR-based TPG of [12]	Reduction	Proposed technique (gate equiv.)	LFSR-based TPG of [12]		
					ROM bits	Control Logic	Multiplexers (gate equiv.)
c880	980	1829	46.4 %	29	0	0	0
c1355	1046	1334	21.6 %	24	0	0	0
c1908	3327	3759	11.5 %	25	0	0	0
c2670	1002	10206	90.2 %	373	33 x 233	H	280
c7552*	3958	-	-	1012	-	-	-
s420	1876	10843	82.7 %	125	9 x 34	H	41
s641	1084	2430	55.4 %	61	6 x 54	H	65
s713	1963	2759	28.6 %	59	7 x 54	H	65
s820	498	527	5.5 %	182	34 x 23	H	28
s838	1223	9273	86.8%	423	43 x 66	H	79
s953	3147	4834	34.9 %	54	4 x 45	H	54
s1196	11553	18776	38.5 %	49	4 x 32	H	38
s1238	7257	7713	5.9 %	77	5 x 32	H	38
s1423	1102	1308	15.7 %	71	4 x 91	H	109
s5378*	8469	-	-	143	-	-	-
s9234*	11207	-	-	948	-	-	-

*: The authors of [12] have not given results for these circuits

Table 4. Comparison of the proposed technique with the results of [12] for LFSRs (serial reseeding)

Circuit	Test application time			Hardware Overhead			
	Proposed technique	LFSR-based TPG of [12]	Reduction	Proposed technique (gate equiv.)	LFSR-based TPG of [12]		
					ROM bits	Control Logic	
						Random Logic	bit counter (gate equiv.)
c880	980	1829	46.4 %	29	0	0	0
c1355	1046	1334	21.6 %	24	0	0	0
c1908	3327	3759	11.5 %	25	0	0	0
c2670	1002	17862	94.4 %	373	33 x 233	H	27
c7552*	3958	-	-	1012	-	-	-
s420	1876	11140	83.2 %	125	9 x 34	H	20
s641	1084	2748	60.6 %	61	6 x 54	H	20
s713	1963	3130	37.3 %	59	7 x 54	H	20
s820	498	1275	60.9 %	182	34 x 23	H	16
s838	1223	12068	89.9 %	423	43 x 66	H	23
s953	3147	5010	37.2 %	54	4 x 45	H	20
s1196	11553	18900	38.9 %	49	4 x 32	H	16
s1238	7257	7868	7.8 %	77	5 x 32	H	16
s1423	1102	1668	33.9 %	71	4 x 91	H	20
s5378*	8469	-	-	143	-	-	-
s9234*	11207	-	-	948	-	-	-

*: The authors of [12] have not given results for these circuits

From Table 1 we can see that, in general, the total number of vectors required for testing the CUT (fourth column), do not exceed the parameter *MAXVECTORS*. In some cases (c1355, c1908, s820, s5378, s9234) this is not true as a consequence of the fact that *MAXVECTORS* is small compared to the vectors needed by the algorithm in order to detect the easy faults. We can also observe that as the value of *MAXVECTORS* increases, less XOR gates and a greater number of vectors are required to fully test the CUT. The exceptions of this rule are due to the random selection of the initial seed and the reverse simulation process. Since the majority of the circuits used as benchmarks have random-pattern resistant faults, from Tables 1 and 2 we can easily verify that using *SL* 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* 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* we get longer test sequences.

In Tables 3 and 4 we compare the proposed scheme against the LFSR-based TPG scheme of [12]. Among the results given in Tables 1 and 2, those with the least hardware overhead were chosen. Also, among results with similar hardware overhead, we chose the one requiring the fewer test vectors. Columns 2 and 3 of Table 3 present the number of test vectors required by the two techniques. It is obvious that our method outperforms the one proposed in [12] (the reduction percentages are given in column 4).

We consider that in both the proposed and the LFSR-based TPG of [12], the initialization of the LFSR takes place by resetting the register. The LFSR-based TPG

85

approach of [12] requires a ROM for storing the seeds and a *control logic for controlling the reseeding operation*. The ROM bits required for each circuit are shown in the sixth column of Table 3 as a product Y x Z, where Y is the number of the required seeds minus 1 and Z the word-length of the ROM. The area overhead of the control logic cannot be calculated, since not enough information has been given in [12], and is therefore represented with letter H in the seventh column.

There are two ways for reseeding the LFSR in [12], parallel reseeding and serial. If the seeds are loaded in parallel, then another set of multiplexers, besides those used for modifying a register to an LFSR, is needed. The hardware overhead for these multiplexers is given in column 8 of Table 3. We observe that in some cases (s641, s713, s953, s1423), this area overhead is greater than the total area required by our technique (column 5, Table 3), while in others (c2670, s1196, s1238), the difference is small enough to conclude that the addition of the ROM and the required control logic will result in greater hardware overhead than the one shown in column 5. As far as c880, c1355 and c1908 are concerned, we observe that the adoption of the proposed technique leads to significantly smaller test sequences with negligible additional hardware overhead. We also note that the use of the extra set of multiplexers imposes further delay in the critical path of the circuit under test thus causing further system performance degradation.

In the case of serial reseeding of the LFSR, instead of the multiplexers, a bit counter is required. This approach also causes an increase to the test application time of each circuit. The test application time is equal to $NV - NS + NS \times s(LFSR)$, where NV is the number of test vectors, NS the number of the seeds and $s(LFSR)$ the size of the LFSR. The clock cycles needed to test each benchmark circuit, are given in column 3 of Table 4. We observe that in several cases (c2670, s641, s713, s820, s1423), the increase in the test application time is significant, while the use of a ROM and the required control logic still imposes considerable hardware overhead.

5. Conclusions

Reseeding has been proposed as an effective technique for testing circuits with random-pattern resistant faults, since it can achieve complete fault coverage with an acceptable number of test vectors. In this paper a new reseeding technique for LFSR-based test pattern generation, suitable for test-per-clock 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 LFSR's next state. Experimental results on the ISCAS '85 and the combinational part of ISCAS '89 benchmark circuits showed that the proposed technique requires less test vectors and, in the majority of cases, less hardware overhead than the other LFSR-based reseeding techniques for achieving complete fault coverage.

References

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, New York, NY, 1990.

[2] P. H. Bardell, W. H. McAnney and J. Savir, *Built-In Test for VLSI: Pseudo-Random Techniques*, Johh Wiley & Sons, New York, NY, 1987.

[3] V. Agrawal, Ch. Kime and K. Saluja, "A tutorial on Built-In Self-Test Part 1: Principles", *IEEE Design & Test of Computers*, March 1993, pp. 73-82.

[4] V. Agrawal, Ch. Kime and K. Saluja, "A tutorial on Built-In Self-Test Part 2: Applications", *IEEE Design & Test of Computers*, June 1993, pp. 69-77.

[5] H. J. Wunderlich, "BIST for systems-on-a-chip", *Integration, The VLSI Journal*, vol. 26, (no.1-2), Elsevier, December 1998, pp. 55-78.

[6] B. Koenemann, "LFSR-Coded Test Patterns for Scan Design", Proc. of European Test Conference, Munich, Germany, April 1991, pp 237-242.

[7] S. Hellebrand, S. Tarnick, B. Courtois and J. Rajski, "Generation of Vector Patterns through Reseeding of Multiple-Polynomial Linear Feedback Shift Registers", Proc. of International Test Conference, Baltimore, MD, USA, Sept. 1992, pp. 120-129.

[8] S. Venkataraman, J. Rajski, S. Tarnick and S. Hellebrand, "An Efficient BIST Scheme based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers", Proc. of International Conference on Computer-Aided Design, Santa Clara, CA, USA, Nov. 1993, pp. 572-577.

[9] 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 Transactions on Computers*, Vol. 44, No. 2, February 1995, pp. 223-233.

[10] J. Savir and W. H. McAnney, "A Multiple Seed Linear Feedback Shift Register", *IEEE Trans. on Computers*, vol. 41, no. 2, February 1992, pp. 250-252.

[11] S. K. Mukund, E. J. McCluskey and T. R. N. Rao, "An Apparatus for Pseudo-Deterministic Testing", Proc. of 13th VLSI Test Symposium, Princeton, NJ, USA, April-May 1995, pp. 125-131.

[12] S. Chiusano, P. Prinetto and H. J. Wunderlich, "Non-Intrusive BIST for Systems-on-a-Chip", Proc. of International Test Conference, Atlantic City, NJ, USA, Oct. 2000, pp. 644-651.

[13] A. P. Stroele and F. Mayer, "Methods to Reduce Test Application Time for Accumulator-based Self-Test", Proc. of VLSI Test Symposium, Monterey, CA, USA, April-May 1997, pp. 48-53.