Multiphase BIST: A New Reseeding Technique for High Test-Data Compression

Emmanouil Kalligeros, Xrysovalantis Kavousianos, *Member, IEEE*, and Dimitris Nikolos, *Member, IEEE*

***Abstract*—In this paper, a new reseeding architecture for scan-based built-in self-test (BIST), which uses a linear feedback shift register (LFSR) as test pattern generator, is proposed. Multiple cells of the LFSR are utilized as sources for feeding the scan chain of the circuit under test in different test phases. The LFSR generates the same state sequence in all phases, keeping that way the implementation cost low. A seed-selection algorithm is furthermore presented that, taking advantage of the multiphase architecture, manages to significantly reduce the number of the required seeds for achieving complete (100%) fault coverage. The proposed technique can be used either in a full BIST implementation or in a test-resource partitioning scenario, since the test-data storage requirements on the tester are very low. When a full BIST implementation is preferable, the multiphase architecture can also be combined with a dynamic reseeding scheme that uses combinational logic instead of a ROM in order to perform the reseedings. This way the implementation area of the BIST circuitry is further reduced. Experimental results demonstrate the advantages of the proposed LFSR reseeding approach over the already known reseeding techniques.**

***Index Terms*—Built-in self-test (BIST), logic circuit testing.**

I. Introduction

THE designers of contemporary systems-on-a-chip (SoCs), in order to reduce the time-to-market and the complexity of their task, make use of various embedded cores, such as memory and processor cores. All these cores are interconnected together with some user-defined logic. However, the continuously increasing density and complexity of such systems make their testing a more and more challenging task.

One of the major problems of testing complex SoCs is that of high test-data volume. The test-data storage requirements on the tester are growing rapidly as the size of the circuit under test (CUT) increases. As a result, test-data compression has become an integral part of a circuit's testing flow, as discussed in [1]. Many works that try to tackle the high test-data volume problem have been recently presented in the open literature. Several different approaches have been proposed, which use, among others, run-length codes and their variants [2]–[4], statistical codes [5]–[7], parallel serial full scan [8], virtual scan chains [9], combinational decompressors [10], linear decompressors [11], [12], ring generators [13], and linear feedback shift registers (LFSRs) along with modified automatic test pattern generator (ATPG) tools [14]. All these works use an external tester to store compressed versions of the test vectors of the CUT, which are then transferred and decompressed on-chip by a small built-in circuit.

Another approach, which, unlike some of the above techniques, requires the structure of the CUT to be known, is to try to embed the test set of the CUT in a longer built-in self-test (BIST) sequence. This way, both the volume and the width of the stored patterns are reduced. BIST [15] is a very effective approach for achieving high rates of test-data compression. It can be combined with various other techniques, but the "optimum results," in terms of test-data reduction, are obtained when combined with reseeding [1]. Reseeding techniques [16]–[32] constitute a very practical and effective solution to the problem of high test-data volume. Recently, commercial test-automation tools that support a reseeding methodology have been announced [33]. Also, as mentioned in [25] and [26], store-and-generate approaches like reseeding are much more flexible, than tailoring the BIST architecture to a given deterministic test set, as in the, otherwise very effective, mapping logic techniques of [34]–[37] ("bit-flipping," "bit-fixing").

Usually, in a reseeding BIST scenario, a mixed-mode approach is adopted, according to which, pseudorandom along with deterministic patterns are applied to the CUT for detecting the random-pattern-testable (easy-to-detect) and the random-pattern-resistant (hard-to-detect) faults, respectively. In some mixed-mode cases, the deterministic patterns are inserted among the pseudorandom ones. The application of deterministic patterns to the CUT is performed by loading, at specific times during testing, new (precalculated) initial states (seeds) to the test pattern generator (TPG). These seeds will expand to deterministic test vectors as the TPG runs. The most acceptable and widely used TPGs, when a mixed-mode approach is used, are LFSRs.

The original idea of encoding test patterns as LFSR seeds by solving systems of linear equations was proposed in [16]. This technique needs an LFSR of length $s_{\max}+20$ for encoding each test cube, where $s_{\max}$ is the maximum number of defined bits in the test cubes testing the hard-to-detect faults of the CUT. The proposed LFSR length $(s_{\max}+20)$ ensures that the probability of not finding a seed for a test cube is less than 10^{-6}. In [17], a method for improving the encoding efficiency of LFSR

Manuscript received July 26, 2003; revised January 2, 2004. This work was supported in part by the Public Benefit Foundation "Alexander S. Onassis" via its scholarships programs, in part by the Research Committee of Patras University, within the framework of the K. Karatheodoris Scholarships Program, and in part by the State Scholarship's Foundation of Greece via its Post-Doctoral research scholarships program. This work was based on "A highly regular multi-phase reseeding technique for scan-based BIST", which appeared in *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, April 2003. This paper was recommended by Associate Editor S. Hellebrand.

E. Kalligeros and D. Nikolos are with the Computer Engineering and Informatics Department, University of Patras, Patra 26500, Greece (e-mail: kalliger@ceid.upatras.gr; nikolosd@cti.gr).

X. Kavousianos is with the Computer Science Department, University of Ioannina, Ioannina 45110, Greece (e-mail: kabousia@cs.uoi.gr).

Digital Object Identifier 10.1109/TCAD.2004.833617

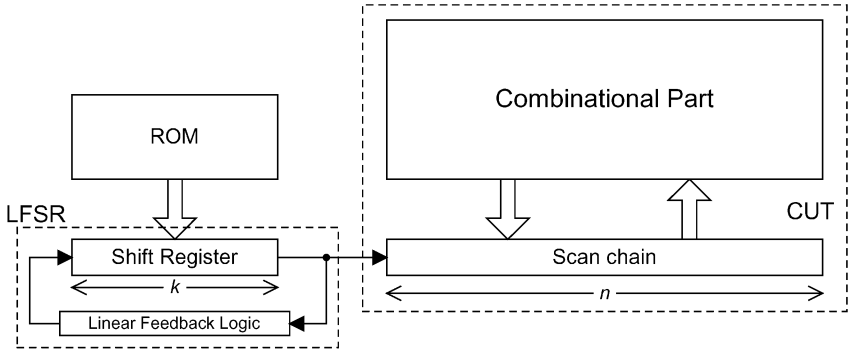

Fig. 1. Classical reseeding scheme for scan-based BIST.

reseeding by using multiple-polynomial LFSRs was proposed. By using 16 polynomials instead of one, the length of the LFSR was reduced to $s_{\max}$. However, the encoding efficiency of both [16] and [17] is limited, since there are many test cubes that contain fewer than $s_{\max}$ bits, which should be encoded using $s_{\max} + 20$ or $s_{\max}$ bits respectively. Two approaches for addressing this problem have been proposed. The first one [18], [19] tries to encode more than one test cubes in just one seed by applying test-cube merging and concatenation, while the other uses variable-length seeds [20]–[22].

In [23] and [24], the reseeding technique is applied to BIST schemes based not on LFSRs but on twisted-ring counters. This approach is as simple to implement as an LFSR-based one and features a very small control logic for controlling the reseeding operation. Its main disadvantage is that a twisted-ring counter cannot offer high encoding efficiency and as a result many seeds are required for fully testing the CUT. In [25], a reseeding scheme based on folding counters (twisted-ring counters with programmable feedback) is presented. Although it manages to significantly reduce the test-data storage requirements, this is done at the expense of a nonstandard configuration of the scan chain of the CUT, which is costly. This problem is solved in [26], where the properties of folding counters are exploited in order to reduce the number of the required seeds (vertical compression), while an LFSR and a small control logic are used for decompressing the seeds into folding counter states (horizontal compression). The method of [26] [two-dimensional (2-D) compression] needs more memory for storing its seeds than that of [25], with the advantage that the nonstandard scan chain configuration of [25] is no longer required. The technique proposed in [27], by using an external tester, reseeds the LFSR dynamically and partially (not all the LFSR register is changed) reducing this way the test-data storage requirements considerably.

The reseeding schemes presented in [28]–[32], instead of loading the seeds from a memory, generate them dynamically during testing, by means of some additional combinational logic. In [28], a very simple scheme with an equally simple but fast reseeding algorithm are presented, which, however, lead to fairly good results, especially as far as the test-sequence length is concerned. In [29], a much more sophisticated reseeding architecture in conjunction with a powerful compression algorithm are proposed. The number of required seeds is significantly reduced, while the proposed architecture, combined with the dynamic reseeding scheme of [28], minimizes the hardware overhead of the BIST circuitry. The authors of [30] use a dynamic scheme similar to that of [28] but with a more efficient reseeding algorithm, which is further improved in [31]. Finally, in [32], the work of [31] is enhanced with a seed-encoding technique based on running the TPG a variable number of clock cycles before loading a new seed.

In this paper, we present a new LFSR reseeding architecture for scan-based BIST that fully exploits the encoding ability of an LFSR seed by using more than one cells of the LFSR for feeding the scan chain of the CUT, in different test phases. This way the number of seeds required for achieving complete (100%) stuck-at fault coverage is significantly reduced. For keeping the hardware overhead of the proposed architecture low, a very regular structure is introduced. This structure can be efficiently combined with the dynamic reseeding scheme, recently proposed in [28]. However, the proposed architecture is generic and can be also combined with a ROM in a full BIST environment or with an external tester in a test resource partitioning implementation. Along with the proposed architecture, an effective seed-selection algorithm is also presented for selecting the seeds and the LFSR cells that will finally feed the scan chain of the CUT.

The remaining of the paper is organized as follows. Section II provides the motivation for this work, Sections III and IV present the proposed architecture and the reseeding algorithm, respectively, while in Section V its applicability to a multiple scan chain environment is discussed. In Section VI, the effectiveness of the proposed technique is evaluated with experimental results and comparisons with previously presented works. Section VII describes how the dynamic reseeding scheme of [28] can be applied to the proposed architecture and hardware overhead results are given for demonstrating the advantages of the dynamic scheme over ROM-based implementations. The paper is concluded in Section VIII.

II. Motivation

The classical reseeding approach for scan-based BIST, assuming a single scan chain, is shown in Fig. 1.

As CUT, we consider a sequential circuit consisting of a combinational part and of n memory elements (flip-flops), which form a scan chain of equal length. The TPG circuit consists of an LFSR with k flip-flop cells $(k < n)$ and a ROM for storing the seeds. The LFSR is periodically loaded with a new seed, starting

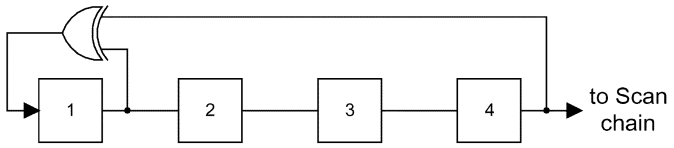

Fig. 2. External-XOR LFSR with characteristic polynomial $x^4 + x + 1$.

from which it generates a predefined number of test vectors. An LFSR cell (usually the last one) is selected for feeding the scan chain of the CUT.

An LFSR seed can be calculated by solving a system of linear equations [16], [18], [27]. Let us assume, for example, that as TPG we use the LFSR of Fig. 2 ($k = 4$) and that the scan chain of the CUT is of length seven ($n = 7$).

For finding a seed so as the LFSR to generate a test vector compatible with a test cube, let say 1x10xx1 (x denotes a don't care), we represent the value of each seed bit with a binary variable a_i. That is, the required seed is initially equal to $\{a_1, a_2, a_3, a_4\}$. For filling the CUT's scan chain the LFSR should pass through seven states, starting from seed $\{a_1, a_2, a_3, a_4\}$ as shown in Fig. 3 (binary expression $a_i a_j \ldots a_m$ denotes the exclusive-OR -XOR- operation of variables $a_i, a_j, \ldots$, and a_m).

In order to find an appropriate seed for test cube 1x10xx1, the resulting binary expressions from the last LFSR cell should be equal to the corresponding care bits of 1x10xx1. Since the last bit of *state* 1, which has been shifted first in the scan chain, is the rightmost binary expression when the scan-in operation is finished, we have:

$$\begin{aligned} a_4 &= 1 \\ a_1 &= 0 \\ a_1 a_4 &= 1 \\ a_1 a_2 a_3 a_4 &= 1. \end{aligned}$$

By solving the above system of linear equations, we get $a_4 = 1, a_1 = 0$, and $a_2 = a_3$, i.e., the required seed is $0a_3a_31$. Indeed, if we replace a_3 with 0, the LFSR will generate vector 1110001, while if we set $a_3 = 1$ vector 1010111 will be generated. Instead of replacing variable a_3 randomly, we could try to encode another test cube in seed $0a_3a_31$ in order to take advantage of the free variable a_3. However, if we consider only the last LFSR cell, the encoding ability of the initial seed is not fully exploited. On the other hand, if we could use some of the other LFSR cells too, more test cubes would be probably compressed to a single LFSR seed.

As an example, we assume that after having determined the necessary seed for test cube 1x10xx1, we would like to encode test cube 1x00xx1 in the same seed, if possible. However, none of the resulting systems of linear equations is solvable if we consider only the last LFSR cell (the maximum window of symbolic vectors that could be generated by the last cell was examined). If we could use some of the other cells though, we could get a vector compatible with test cube 1x00xx1 from the inverted output of the first cell, by replacing a_3 with 1. This can be justified by solving the system of linear equations that corresponds to the first LFSR cell for the test cube that is complementary to 1x00xx1 (0x11xx0). The generated bit sequences are shown in Fig. 4.

From the above example, it is obvious that the encoding ability of an LFSR seed would have been better exploited if more than one LFSR cells had been selected for feeding the scan chain of the CUT, in different test phases of course. That is the main idea behind the proposed multiphase LFSR reseeding scheme that will be presented in detail in the following section. It should be underlined that, although the above example was based on an external-XOR LFSR, such LFSRs are not convenient for feeding a CUT's scan chain from various cells, since the phase shift between the bit sequences of any pair of their cells is bounded by the LFSR length. However, as will be seen in the evaluation section of the paper, when internal-XOR LFSRs are used, the seeds' encoding ability improves as the number of utilized cells increases.

III. Proposed Architecture

The overview of the proposed multiphase scan-based architecture is shown in Fig. 5.

The proposed scheme has two modes of operation: 1) the easy-fault-detection mode and 2) the hard-fault-detection mode. Although this distinction appears in every mixed-mode scheme, in the proposed one the operation of the scheme in these two modes differs significantly. During the easy-fault detection, the LFSR runs in autonomous mode and feeds the scan chain of the CUT from a single cell (the last one). The user defines the value of parameter *VectorsForEasyFaults*, which denotes the maximum number of vectors that can be used for detecting the easy faults. Only the Bit and Vector Counters are enabled during the easy-fault-detection mode. The Bit Counter controls the scan-in operation of each produced vector and signals the Vector Counter to increase, while the Vector Counter increases until a number of pseudorandom patterns equal to *VectorsForEasyFaults* have been applied to the CUT. The Reseeding and the Cell Selection Counters are initially reset like the Bit and Vector Counters, but they retain their initial zero value throughout the easy-fault-detection mode (i.e., they are disabled). The scheme's operation switches from the easy-vault-detection mode to the hard-fault-detection one when *VectorsForEasyFaults* pseudorandom patterns have been applied to the CUT, that is when Vector Counter reaches value *VectorsForEasyFaults*-1.

For better explaining the proposed architecture's operation during the hard-fault-detection mode, we assume that a subset of p LFSR cells has been chosen to feed the scan chain (this subset includes the last cell used for generating the initial pseudorandom patterns). The hard-fault-detection part of the test session consists of p test phases, and in each phase, one of the p selected cells is used to feed the scan chain. All phases are identical considering the operation of the LFSR and the reseeding scheme, with the difference that in each phase a different cell of the LFSR is used to feed the scan-chain of the CUT and, therefore, a different pattern sequence is generated. Specifically, in each phase:

1) the same number of test vectors is loaded in the scan chain,
2) the same seeds in the same order and at the same time instants are loaded in the LFSR and, as a result
3) the LFSR passes through the same sequence of states.

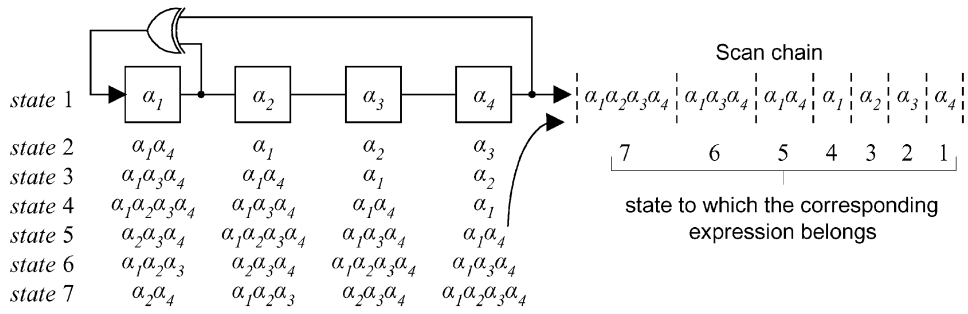

Fig. 3. Generating seven states starting from seed $\{a_1, a_2, a_3, a_4\}$.

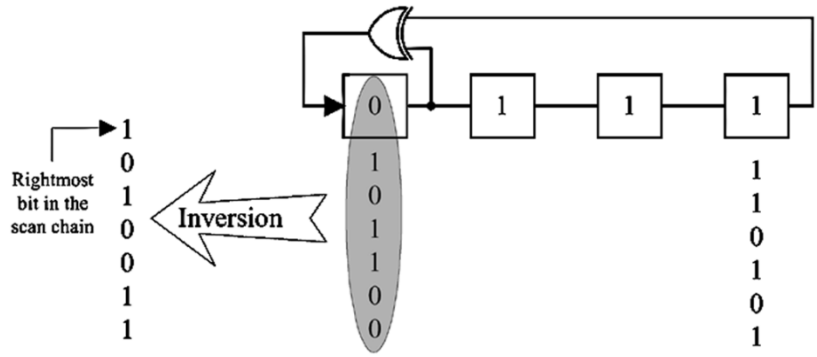

Fig. 4. Using multiple LFSR cells for generating test vectors.

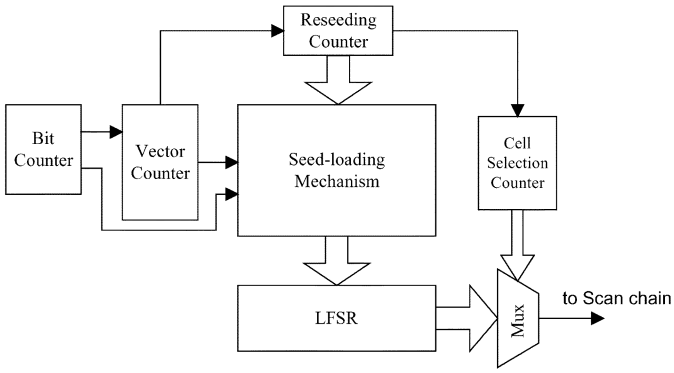

Fig. 5. Multiphase scan-based TPG architecture.

Furthermore, between every two successive reseedings, the same constant, user-defined number of vectors, *VectorsPerSeed*, is loaded into the scan chain. The above properties make the structure of the proposed architecture very regular and easy to implement. The test sequence generated by the multiphase TPG is of the form shown in Fig. 6.

Let us now describe the operation of the proposed architecture in the hard-fault-detection mode more thoroughly. To simplify the description, we have omitted the capture cycle required after a test vector has been shifted in the scan chain. As we previously mentioned, at the end of the easy-fault-detection mode, Vector Counter's value is equal to *VectorsForEasyFaults*-1, while the Reseeding and the Cell Selection Counters are equal to 0 (Bit Counter's value is $n-1$, where n is the scan chain length). In the next clock edge, the first Reseeding is performed. The Vector Counter signals the Reseeding Counter to increase by one and *VectorsPerSeed* test vectors are about to be generated, starting from the seed that has been loaded in the LFSR. A new Reseeding is performed when the Vector Counter has counted *VectorsPerSeed* patterns and, of course, Bit Counter's value is $n-1$ (which means that the last test vector of the previous seed has been loaded in the scan chain). We note that when performing a reseeding, Vector Counter's value is properly reduced, so as after the application of the *VectorsPerSeed* patterns of the new seed, to be equal to *VectorsForEasyFaults*-1. This way the only value of the Vector Counter that needs to be checked in all reseedings is *VectorsForEasyFaults*-1. When all the reseedings of a phase have been performed, the Reseeding Counter signals the Cell Selection Counter to increase and the next phase is initiated (Seed 1 is reloaded in the LFSR). The value of the Cell Selection Counter is increased by one at each new phase, thus enabling a different LFSR cell to feed the scan chain through the multiplexer. As for the Seed-loading Mechanism, it can be a ROM as in the classical reseeding approach, a combinational logic for reducing the hardware requirements of the BIST circuitry as will be described in Section VII or it can even be eliminated and replaced by an external tester in a test resource partitioning scenario.

Assuming that in each phase R reseedings are performed, then the test sequence length is given by the following relation (see Fig. 6):

$$\text{Test Sequence Length} = VectorsForEasyFaults + p \cdot R \cdot VectorsPerSeed. \quad (1)$$

An important feature of the multiphase architecture is that, practically, the hardware overhead of the Seed-loading Mechanism does not depend on the number of phases, since its operation as well as that of the LFSR is the same in all of them. Taking into consideration that, apart from the Seed-loading Mechanism, the hardware overhead of the rest control logic

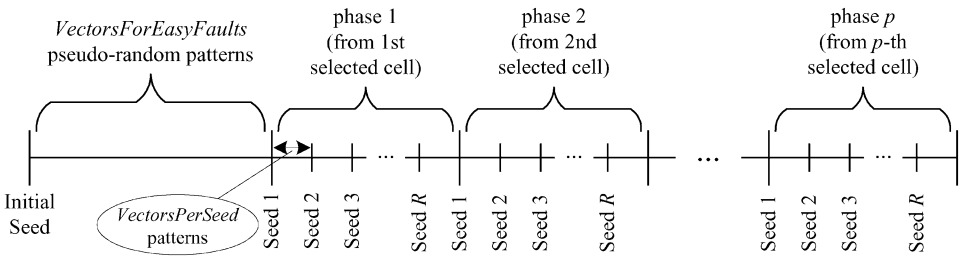

Fig. 6. Test sequence generated by the multiphase TPG architecture.

is negligible, we conclude that the overall hardware overhead of the proposed architecture does not depend on the number of phases included in the test sequence (only the size of the multiplexer and probably that of the Cell Selection Counter increase when a new phase is added).

We have to note that both normal and inverted outputs of the LFSR cells may be used to feed the scan chain, as in the example of the previous section. We should also stress that the proposed architecture does not require any modifications of the scan chain of the CUT, being this way fully compatible with standard scan design.

IV. Reseeding Algorithm

In this section, we present an efficient algorithm for selecting the seeds and the LFSR cells, which will feed the scan chain of the CUT throughout testing. The main goal of this algorithm is to minimize the number of seeds required for fully (100%) testing the CUT. This way, the hardware required for the implementation of the proposed scheme will be minimized as well. The reseeding algorithm consists of three parts: 1) the easy-fault-detection part, which is then followed by some test-cube preprocessing steps; 2) the seed and cell-selection part for detecting the hard faults; and 3) the test sequence-reduction part.

A. Easy-Fault Detection and Preprocessing

The first part of the proposed algorithm is rather straightforward. The user defines the value of parameter *VectorsForEasyFaults* as mentioned above, the LFSR is set to a random initial state and *VectorsForEasyFaults* random patterns are applied to the CUT from the last LFSR cell, for testing the easy-to-detect faults. This pseudorandom part of the test sequence detects the vast majority of the faults of the CUT. The remaining faults are identified as hard-to-detect and multiple test cubes are extracted for each one of them by using the ATALANTA ATPG tool [38].

Only a small subset of the extracted test cubes is selected for participating in the second part of the algorithm. This subset detects all hard faults and has much smaller cardinality than the initial test-cube set. The selection procedure (fault-simulation preprocessing) is fairly fast and targets both the optimization of the seed volume results and the run-time reduction of the second and main part of the reseeding algorithm.

At first, we fault simulate all the test cubes of the hard-to-detect faults that have been extracted with ATALANTA and we record, for each one, the set of hard faults it detects. To each hard fault f_j a fault index $\mathrm{ind}(f_j)$ is also assigned, which is equal to the number of times f_j has been detected after the application of all test cubes. This is also an indication of how "hard" f_j is (f_j is "harder" than f_i if $\mathrm{ind}(f_j) < \mathrm{ind}(f_i)$, since f_i is tested by more test cubes). A quality metric is then calculated for each test cube c, which is equal to the mean value of the indices $\mathrm{ind}(f_j)$ of the faults that test cube c detects. The easier the faults detected by a test cube are, the greater the corresponding metric is.

A minimal set of test cubes has to be selected such that all the hard-to-detect faults of the CUT can be tested by the test cubes of the selected set. For solving this set-covering problem, we use a simple, greedy heuristic algorithm. At each iteration of the algorithm, we select the test cube that detects the largest set of hard-to-detect faults and we eliminate those faults and their corresponding test cubes (except for the selected one), so as not to be considered by the seed-selection procedure. If we have to choose between two test cubes, which detect the same maximum number of hard faults, we select the one with the smaller metric (i.e., the one that detects the hardest faults). By repeating the above described step, we finally end up with a small subset of test cubes that test all the hard-to-detect faults of the CUT.

The role of the fault-simulation preprocessing is to enhance the quality of the test-cube set, by filtering out redundant cubes. In that way, not only the running time of the seed-selection algorithm is significantly reduced, but its efficiency is improved as well, since only test cubes that test many hard-to-detect faults are handled.

After the fault-simulation preprocessing step, we check if for each one of the selected test cubes, a seed, for at least one LFSR cell, can be calculated such that the first vector generated from that seed to be compatible with the corresponding test cube. Practically, such a seed always exists according to [26], given that the LFSR length k is in the neighborhood of the maximum number of defined bits of the test cubes ($s_{\max}$).

A final preprocessing step then follows, which uses the information gathered in the preceding checking step. Its purpose is to assign a weight to each of the selected test cubes, which will be used by the seed selection procedure in the second part of the algorithm. The weight of a test cube is equal to the average number of variables that remained undefined (free) after solving the corresponding systems of linear equations from all LFSR cells. That is, if $fv_{c,i}$ is the number of free variables after solving a system of linear equations for finding a seed for test cube c from cell i, then the weight of c is equal to

$$w_c = \left(\sum_{i=1}^{k} \mathrm{fv}_{c,i} \right) \Big/ sl$$

where k is the LFSR length, $fv_{c,i} = 0$, if the system is unsolvable and sl is the number of solvable systems. The smaller the weight is, the more difficult is for the algorithm to encode

another test cube along with c in the same seed, since fewer variables remain free after solving a system for that cube.

B. Selection of Seeds and LFSR Cells for Testing the Hard-to-Detect Faults

The procedure that will be described in this section determines a seed and some LFSR cells for feeding the scan chain of the CUT in order to detect as many hard faults as possible. By repeating the same procedure we derive the final set of seeds and LFSR cells that will be used by the multiphase TPG. The seed-selection algorithm that will be presented tries to encode as many test cubes as possible to just one seed by exploiting the bit sequences produced by more than one cells of the LFSR.

Before proceeding to the description of the algorithm, we should note that although the proposed TPG architecture performs all the reseedings while feeding the scan chain from the same LFSR cell (Fig. 6), the seed-selection algorithm examines the sequence of all LFSR cells when deciding for a new seed. That is, in order to preserve the regularity of the proposed architecture, as it was described in Section III, the test vectors are applied to the CUT in different order from which they have been processed. This regularity is vital in the case that the Seed-loading Mechanism is implemented as combinational logic, as will be explained in Section VII.

The selection of a new seed and of the appropriate LFSR cells is done by solving systems of linear equations based on the feedback structure of the LFSR, as demonstrated in Section II. Initially, the logic value stored in cell q of the LFSR is represented by the binary variable a_q. Therefore, the initial state $\{E_1(1), E_2(1), \ldots, E_k(1)\}$ of the LFSR consists of k variables, $\{a_1, a_2, \ldots, a_k\}$, where k is the LFSR length and $E_1(1) = a_1, E_2(1) = a_2, \ldots, E_k(1) = a_k$. Then, we let the LFSR evolve for $n \cdot VectorsPerSeed$ states (i.e., as if it was generating $VectorsPerSeed$ test vectors), where the ith LFSR state is equal to $\{E_1(i), E_2(i), \ldots, E_k(i)\}$ and each one of the $E_1(i), E_2(i), \ldots, E_k(i)$ is a binary expression containing one or more variables from the set $\{a_1, a_2, \ldots, a_k\}$ (the variables in each binary expression are related together with the modulo-2 addition, i.e., XOR, operation only). We define as $\mathrm{EV}_i(j)$ (Expression Vector) the jth set of binary expressions produced by the ith cell of the LFSR. Each $EV_i(j)$ set has cardinality n. As shown in Fig. 7, $EV_i(j)$ is the jth vector (in reverse bit order) produced by the ith cell of the LFSR, if its initial state is equal to $\{a_1, a_2, \ldots, a_k\}$. In the example of Fig. 3 ($n = 7$), we have $E_4(1) = a_4, E_4(2) = a_3, E_4(3) = a_2, E_4(4) = a_1, E_4(5) = a_1a_4$, $E_4(6) = a_1a_3a_4, E_4(7) = a_1a_2a_3a_4$, and $EV_4(1) = \{E_4(1), E_4(2), E_4(3), E_4(4), E_4(5), E_4(6), E_4(7)\} = \{a_4, a_3, a_2, a_1, a_1a_4, a_1a_3a_4, a_1a_2a_3a_4\}$. As can be seen in Fig. 3, the first symbolic vector shifted in the scan chain of the CUT from the fourth LFSR cell is equal to $EV_4(1)$ in reverse bit order.

Let $t = \{t_n \ldots t_2 t_1\}$ be a test cube detecting fault f, i.e., $t_r \in \{0, 1, \mathrm{x}\}$ with $1 \leq r \leq n$. If the system of linear equations $EV_i(j) = t$, which is

$$\begin{aligned} E_i\big((j-1)\cdot n+1\big) &= t_1 \\ E_i\big((j-1)\cdot n+2\big) &= t_2 \\ &\vdots \\ E_i\big((j-1)\cdot n+n\big) &= t_n \end{aligned}$$

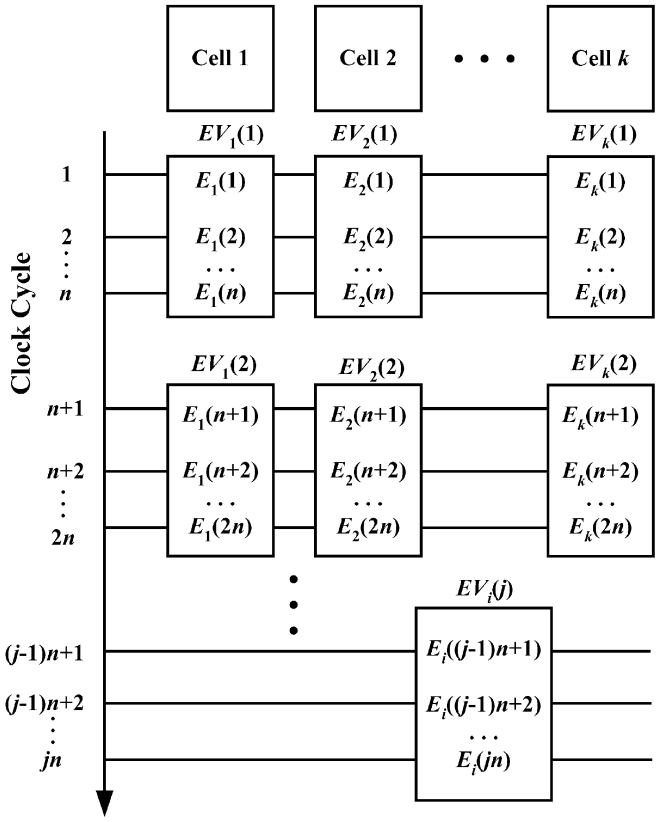

Fig. 7. Expression vector (EV) sets as they are generated by the LFSR.

for $t_r \neq \mathrm{x}$, can be solved, then a test vector detecting fault f can be produced by cell i of the LFSR during the jth n-tuplet of clock cycles after its reseeding. If this system has a solution, then some of the variables $\{a_1, a_2, \ldots, a_k\}$ can be replaced by expressions containing other binary variables (the free variables when solving the system) and constants (0 or 1). If we replace those variables in the initial state of the LFSR $\{E_1(1), E_2(1), \ldots, E_k(1)\}$ we get a seed, starting from which the LFSR will generate a test vector for detecting fault f from cell i, after j n-tuplets of clock cycles. Thus, the first step of the seed-selection procedure is to construct all sets of binary expressions $EV_i(j)$, with $1 \leq i \leq k$ and $1 \leq j \leq VectorsPerSeed$, by simulating the LFSR symbolically.

The seed-selection algorithm tries to encode the test cubes of the hard-to-detect faults in LFSR seeds according to the following rule: the most "difficult" cubes according to their weights w_c (Section IV-A), have to be encoded first. This rule leads to solutions with fewer seeds. By selecting the most "difficult" cubes first, we avoid having to handle them at the last stages of the encoding procedure, where it is not easy to encode, in the same seed, another "difficult" test cube. As a result, a more uniform distribution of the detected hard faults over the reseedings is achieved, since in the last reseedings we can still encode more than one test cube in an LFSR seed. This rule is a generalization of a first-cube selection criterion presented in [39].

Initially, the algorithm selects the test cube t with the smallest weight (the most "difficult" to be encoded along with another one) and attempts to solve one of the systems $EV_i(1) = t$, for all k LFSR cells. Since no significant variable savings can

be achieved by solving the system for different LFSR cells, the first system that can be solved is selected. The selected solution leads to the replacement of some variables as explained above. These variables are replaced in all $EV_i(j)$ sets and, in that way, we get new reduced sets $EV'_i(j)$. Then for each one of the remaining test cubes (t'), the algorithm attempts to solve the systems $EV'_i(j) = t'$, for all i, j. All the systems that can be solved are inserted in the set *SolvableSystems* and, from those, a system that corresponds to the cube with the smallest weight is selected first. The system is solved and some more variables are replaced. This time the replacement of those variables is done only in the systems of the set *SolvableSystems*. It is expected that many of the systems of this set will no longer be solvable, due to the variable replacements. Such systems are dropped from the set. We repeat this selection procedure, until the set *SolvableSystems* becomes empty. During the selection of a system $EV_i(1) = t, EV'_m(j) = t'$, etc., the algorithm also selects cells i, m, etc., for feeding the scan chain.

The successive replacements of the variables a_i with binary expressions leads gradually to their replacement with constant values (0 or 1). In that way, the initial state of the LFSR $\{a_1, a_2, \ldots, a_k\}$ is gradually transformed to a $0/1$ k-tuplet, which is the required seed. Any variables not replaced by constant values are set to a random value. Starting from that seed, we generate all the *VectorsPerSeed* test vectors from each selected cell, we fault simulate them and drop the test cubes of the detected faults. The whole seed-selection procedure is then repeated, targeting a new seed, until complete fault coverage is achieved.

One final remark about the proposed algorithm is that the user can bound the number of cells that will be selected for feeding the scan chain by setting the value of parameter *MaxCellsToSelect*. This constraint has been added in order to have better control of the resulting test sequence length (relation 1, Section III). The algorithm confines its search to only the already selected LFSR cells, if their number becomes equal to *MaxCellsToSelect*. We note that when one test cube has to be chosen among others of the same minimum weight, the one generated by an already selected cell is preferred.

The complexity of the seed-selection algorithm is rather moderate. It should be noted that the procedures for solving systems of linear equations modulo-2, are efficient and fast [27]. In fact, they are much faster than those for solving conventional systems of linear equations. Although in both cases $\mathrm{O(n^3)}$ operations are required [40], for a conventional system these operations are floating point, which are far more time consuming than the simple modulo-2 addition (XOR). Also, the above-described algorithm is significantly speeded up by the fact that the most "difficult" cube is initially selected when searching for a new seed. Consequently, many variables are initially replaced and, as a result, few of the reduced systems $\mathrm{EV}'_i(j) = t'$ can be solved and inserted to the set *SolvableSystems*. The search time of the algorithm is furthermore reduced when, after calculating the first seeds, the number of selected cells becomes equal to the value of parameter *MaxCellsToSelect*. The actual running time of our algorithm, excluding fault simulation, varied for the five larger ISCAS'89 benchmark circuits between some minutes and two hours on a 2.4-GHz Pentium IV system. However, significant execution-time reductions can be achieved by adopting various implementation optimizations.

C. Test Sequence Reduction Procedure

When all the necessary seeds and the cells of the LFSR that will feed the scan chain of the CUT have been determined, the third and final part of the reseeding algorithm is applied. This part attempts to reduce the test sequence length. Let us assume that:

1) p LFSR cells have been selected for finally feeding the scan chain of the CUT (p includes the last cell used in easy-fault-detection mode).
2) R reseedings are performed.

Then, the test sequence length, as explained in Section III, is equal to

$$\text{Test Sequence Length} = VectorsForEasyFaults + p \cdot R \cdot VectorsPerSeed.$$

The optimization procedure attempts to reduce each one of these factors in three steps, by fault simulating the vectors of the test sequence in various permutations:

Step 1) *Reduction of the reseedings (R) and/or the initial vectors (VectorsForEasyFaults):* This step is based on the observation that many of the easy faults, which are detected by the initial pseudorandom patterns, are also detected by subsequent reseedings and, therefore, many of the initial vectors could be eliminated. This optimization step is executed in two substeps.
   1) Starting from the last reseeding, we fault simulate the *VectorsPerSeed* test vectors produced by each seed, from all selected LFSR cells, in reverse seed order. That is, the vectors of the last LFSR seed are first simulated from all p LFSR cells, then those of the second to the last, etc. If at some point during this procedure complete fault coverage is reached, all the initial *VectorsForEasyFaults* patterns are removed from the test sequence along with the test vectors that correspond to the unsimulated seeds, if any. An unsimulated seed is a seed, whose corresponding vectors have not been simulated during this procedure.
   2) If complete fault coverage has not been achieved in the previous substep, we fault simulate in reverse order the initial *VectorsForEasyFaults* patterns until we reach 100% fault coverage. Then, we set the last reversely simulated vector as the initial one (and its corresponding state as the initial state of the LFSR) and we exclude the rest test vectors from the test sequence.

Step 2) *Reduction of VectorsPerSeed*: This reduction step is based on the observation that the number of test vectors generated between two successive reseedings

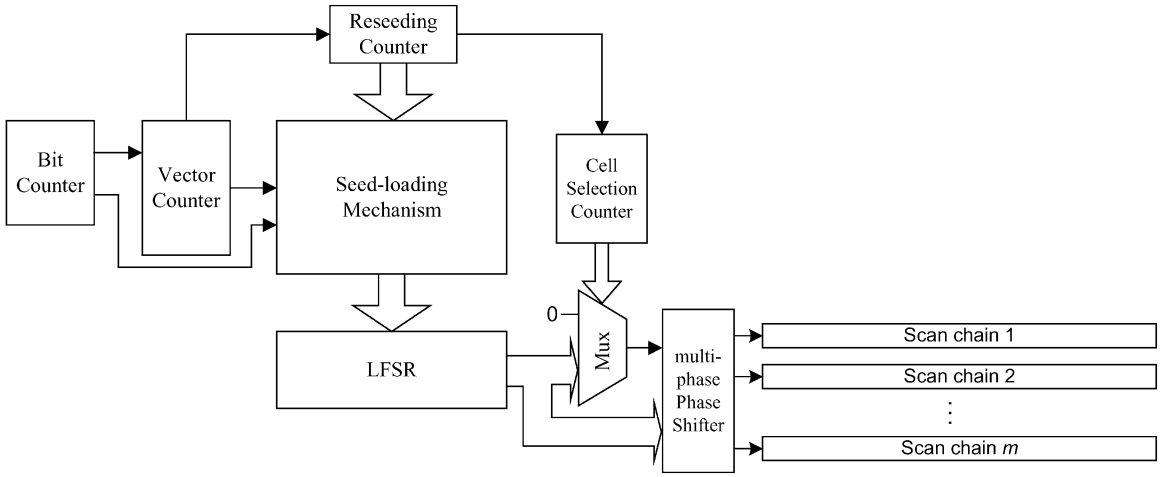

Fig. 8. Proposed multiphase architecture for multiple scan chains.

has been set arbitrarily by the user and may be reduced, without any drop in the fault coverage. The following tasks are executed during this step.

1) The remaining from Step 1 test vectors for testing the easy faults are fault simulated and the detected faults are dropped. If Step 1 has eliminated all the initial vectors, we proceed directly to task 2.
2) For all the seeds, we fault simulate the first vector generated by each of the p selected LFSR cells and we drop the detected faults. We repeat the same procedure for the second vector, the third, etc., until at the ith vector, complete fault coverage is achieved. We then set $VectorsPerSeed = i$.

Step 3) *Reduction of the selected LFSR cells* (p): This step fault simulates the test vectors produced by all the selected cells of the LFSR and records the faults detected by each cell, as well as the total number of times a fault has been detected by different LFSR cells. If all the faults tested by cell i have been totally detected more than once, then cell i can be removed without any loss in the fault coverage. For this reduction step we should note that 1) no extra fault simulation is required, since all the necessary information can be gathered when executing Step 2 and 2) if Step 1 has not totally eliminated the initial pseudorandom vectors, the last LFSR cell (from which they are generated) cannot be removed.

V. Application of the Multiphase Technique to Multiple Scan Chain Architectures

So far, we have presented the multiphase architecture and the corresponding reseeding algorithm considering circuits with a single scan chain. We will now explain how the multiphase technique can be applied in a multiple scan chain environment. We should note beforehand that neither the basic structure (Fig. 5) of the proposed architecture nor the reseeding algorithm (and its complexity) are modified in the multiple scan chain case.

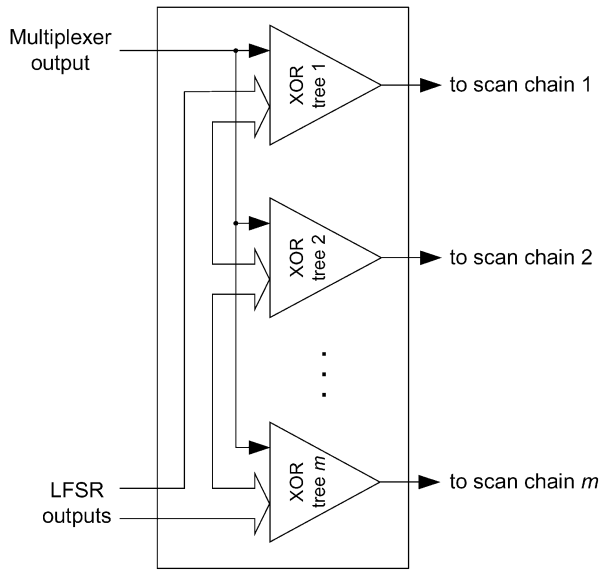

Fig. 9. Multiphase Phase Shifter.

A. Multiphase Architecture for Multiple Scan Chains

The architecture of the proposed multiphase scan-based TPG, for the case of a CUT with more than one scan chains, is shown in Fig. 8.

The main difference from the architecture proposed for a CUT with a single scan chain, is the multiphase Phase Shifter module, which is inserted between the multiplexer and the scan chains. The purpose of this module is twofold: it minimizes the linear dependencies among the bit sequences shifted in the scan chains, as well as, by receiving the output of the multiplexer, it feeds the scan chains of the CUT with different shifted versions of the LFSR's m-sequence. A more detailed diagram of the multiphase Phase Shifter is shown in Fig. 9.

The multiphase Phase Shifter is initially calculated as a normal phase shifter, as proposed by the authors of [41]. Then, an extra input is added to each XOR tree, which is driven by the multiplexer of the multiphase architecture. At each test phase, a different LFSR cell is selected to drive through the multiplexer that extra input. This way the shifting function of each of the Phase Shifter's outputs is altered and a different part of the

LFSR's m-sequence is generated. This is essentially the same as using different LFSR cells in the single scan chain case presented in Section III. In the case of multiple scan chains, the multiplexer can also drive the XOR trees' extra input with the constant 0 value. When this happens, the Phase Shifter operates as if the additional input was not present and, therefore, as it was originally designed to shift the m-sequences. This is done for exploiting the potential of the original phase shifter too. We should note that when the output of an LFSR cell is fed to the XOR trees, there is some probability that the resulting interchain separation for some of the scan chains will be less than the maximum scan chain length. This may reduce the encoding ability of the scheme when that specific LFSR cell drives the XOR trees, but it is highly unlikely that the same will be true for many LFSR cells. We should also mention that the additional XOR trees' input does not affect their propagation delay when the number of the rest of the inputs is not equal to a power of 2. In the opposite case, one additional level of two-input XOR gates is added to the trees.

As far as the operation of the scheme is concerned, it is the same as that described in Section III with the difference that, during the easy-fault-detection mode and the first reseeding phase, the output of the multiplexer is not driven by the last LFSR cell, but by the logic value 0 (i.e., the initially calculated phase shifter is used). This is done in order to ensure the high efficiency of the pseudorandom sequence, which tests the easy-to-detect faults of the CUT.

B. Reseeding Algorithm in the Multiple Scan Chain Case

Handling the additional constant 0 value of the multiplexer is what differentiates the reseeding algorithm for the multiple scan chain case from that for the single one. As explained above, the constant 0 value is used in the easy-fault-detection mode and is also considered during seed selection. That is, it is utilized exactly as the preselected last LFSR cell in the single scan chain case. So, when a number of LFSR cells has been selected by the reseeding algorithm, one of them (the first one) corresponds to the constant 0 value. As for the linear system-solving part of the seed-selection procedure, since the outputs of the LFSR cells are driven through the XOR trees to the scan chains of the CUT, the resulting binary expressions are of the form presented in Sections II and IV. Consequently, the resulting linear systems can be treated as already described. Of course, an $EV_i(j)$ set is now constructed taking into consideration all the outputs of the multiphase Phase Shifter, so as to correspond to a test vector shifted in the scan chains of the CUT.

Another point that is worth discussion concerns the random shift of the LFSR's m-sequence when an unconsidered, during the original shifter's calculation, LFSR cell is driven through the multiplexer to the XOR trees, during the original shifter's calculation. The basic idea of the multiphase technique (for either single or multiple scan chains) is to exploit various shifted versions of the same m-sequence available in an LFSR state sequence. Although the exact shift is not known at each test phase (which is also the case when driving a single scan chain from various LFSR cells), it can be utilized to our benefit by determining the binary variables' values by means of solving systems of linear equations. That is, the randomness of the shifting is overbalanced by the derivation of the variables' values via the solution of linear systems. This way, the multiphase technique enhances the encoding ability of an LFSR seed, as will be presented in the experimental results section that follows.

TABLE I
EXPERIMENTAL RESULTS FOR THE ISCAS BENCHMARK CIRCUITS, ASSUMING A SINGLE SCAN CHAIN

Circuit	Scan Elements	s_{max}	LFSR length	# Source cells	*Vectors PerSeed*	# Seeds	# Vectors
c2670	233	41	66	12	20	27	8980
c7552	207	100	100	16	15	34	11715
s420	34	20	25	13	50	5	5954
s641	54	20	22	3	1250	2	12450
s713	54	20	22	3	1250	2	12450
s838	66	36	45	25	5	12	4625
s953	45	15	15	5	700	1	8352
s1196	32	17	17	7	300	2	8058
s1238	32	17	17	8	200	2	7840
s5378	214	17	19	5	55	10	12760
s9234	247	47	44	14	25	82	37675
s13207	700	22	22	18	100	21	46400
s15850	611	37	47	19	40	66	56160
s38417	1664	86	86	32	70	116	262416
s38584	1464	54	51	17	80	23	38947

VI. EVALUATION AND COMPARISONS

For evaluating the effectiveness of the proposed technique, we implemented the algorithm described in Section IV in the C programming language and we performed a series of experiments using the ISCAS'85 [42] and the ISCAS'89 [43] benchmark circuits. Only circuits with undetected faults after the application of 10 000 pseudorandom patterns have been considered. The size of the LFSRs used was determined by the maximum number of defined bits $(s_{\max})$ that the test cubes, detecting the hard-to-detect faults of the CUT, contained. According to [26], an LFSR of size k, with $s_{\max} - 5 \leq k \leq s_{\max} + 2$, suffices for encoding test cubes with $s_{\max}$ defined bits. For a few benchmark circuits, in order to boost the encoding efficiency of the seed-selection algorithm, we used LFSRs with greater length than $s_{\max} + 2$ (see Table I). This way more variables remained in the produced expressions after the selection of the first, most "difficult" test cube. The primitive polynomials of the LFSRs were generated with the tools that can be found in [44]. We note that in our experiments, we used internal-XOR LFSRs. For each one of the smaller circuits, which have few hard-to-detect faults, 20 experiments were performed, starting from a random initial seed each time. For the circuits containing many hard faults (c2670, c7552, s838, s9234, s13207, s15850, s38417, s38584) we ran up to five experiments with different values of parameters *VectorsPerSeed*, *MaxCellsToSelect* and, in some cases, with different LFSR lengths. For such circuits, the selection of the initial seed is not critical for the performance of the algorithm. The value of parameter *VectorsForEasyFaults* varied from 5000 to 10 000 (in most cases it was set to 10 000). The test sets we have used in our experiments can be found in [45].

In Table I, the best results obtained from the above experiments, for the single scan chain case, are reported (results for circuits with multiple scan chains will be provided at the end of this section). The names of the benchmark circuits as well as the length of the corresponding scan paths (primary inputs +

internal flip-flops) are given in the first two columns. The subsequent columns present the maximum number of defined bits in the test cubes of the CUT ($s_{\max}$), the length of the LFSR used as TPG, the number of LFSR cells which were selected for feeding the scan chain (source cells), the value of parameter *VectorsPerSeed* and the number of calculated seeds. The total number of vectors for achieving 100% stuck-at fault coverage is shown in the right-most column of Table I. Please note that although the outputs of the benchmark circuits are not included in the scan element volumes that appear in the second column of Table I, they have been taken into consideration when performing our experiments.

Concerning the results of Table I, it is worth noting that, as mentioned earlier, in most cases the LFSR length is in the neighborhood of $s_{\max}$ as it was reported in [26]. Consequently, the high encoding efficiency of the proposed technique that will be verified by subsequent comparisons is due to the better exploitation of the LFSR seeds, which are used for feeding the scan chain of the CUT from multiple LFSR cells. We also observe that for circuits containing many hard-to-detect faults, we do not need to set *VectorsPerSeed* to a high value in order to achieve high compression rates. This favors the running time of the algorithm, which remains low.

There are three parameters that affect the performance of the seed-selection algorithm: 1) *MaxCellsToSelect*; 2) *VectorsPerSeed*; and 3) the LFSR length. The LFSR length is directly connected to the number of free variables when solving a system for encoding a test cube to an LFSR seed and, as a result, a larger LFSR offers higher encoding ability. However, this is a well-known property and can be applied to the vast majority of LFSR reseeding works. Thus, it would be more interesting to investigate the behavior of the algorithm, with respect to the values of parameters *MaxCellsToSelect* and *VectorsPerSeed*.

In Fig. 10(a) and (b), the seed volume and the test sequence length results for a series of experiments for s5378 are presented, respectively. 7000 pseudorandom patterns were initially generated by the LFSR and then the seed-selection algorithm was applied for calculating the seeds for the remaining hard-to-detect faults. *VectorsPerSeed* was set to 1, 5, 15, 20, 30, 50, 100, 200, and 400 for each one of the values 5, 10, 15, and 19 of parameter *MaxCellsToSelect* (36 experiments were totally performed). The LFSR length was equal to 19.

In Fig. 10(a), we observe that, as was expected, the encoding efficiency of the seed-selection algorithm improves as the value of parameter *VectorsPerSeed* increases. The main characteristic of the proposed TPG architecture however, is that of feeding the scan chain of the CUT from multiple LFSR cells. In general, the total number of selected seeds is reduced when the algorithm is allowed to handle more LFSR cells (parameter *MaxCellsToSelect*). Specifically, an increase in the value of *MaxCellsToSelect* can affect the results of the seed-selection algorithm, in different ways according to value of parameter *VectorsPerSeed*. As can be seen in Fig. 10(a), when *VectorsPerSeed* is very small (1 to 15), there is no significant benefit, since the search space of the algorithm is confined by parameter *VectorsPerSeed*. Especially for *VectorsPerSeed* $= 1$, the number of resulting seeds is constant and equal to 30 for all values of parameter *MaxCellsToSelect*. For *VectorsPerSeed* $= 15$ to approximately 25, the reduction in the seed volume, caused by a *MaxCellsToSelect* increase, does not bring about a respective increase in the test sequence length, due to the relative small value of *VectorsPerSeed*. That is, *VectorsPerSeed* is high enough so as not to decisively bound the search space of the algorithm, as well as small enough in order to allow an increase in the value of *MaxCellsToSelect* to be compensated by the resulting decrease in the number of selected seeds. For example, for *VectorsPerSeed* $= 20$, we get 23 seeds and 9520 vectors for *MaxCellsToSelect* $= 5$ and 20 seeds with 9880 vectors for *MaxCellsToSelect* $= 10$, that is, with a 3.8% increase in the test sequence length, we have a 13% gain in test-data storage. For greater values of *VectorsPerSeed*, the gain in seed volume due to an increase in the value of parameter *MaxCellsToSelect*, is "paid" by additional test vectors (the classical "test-data storage—test sequence length" tradeoff).

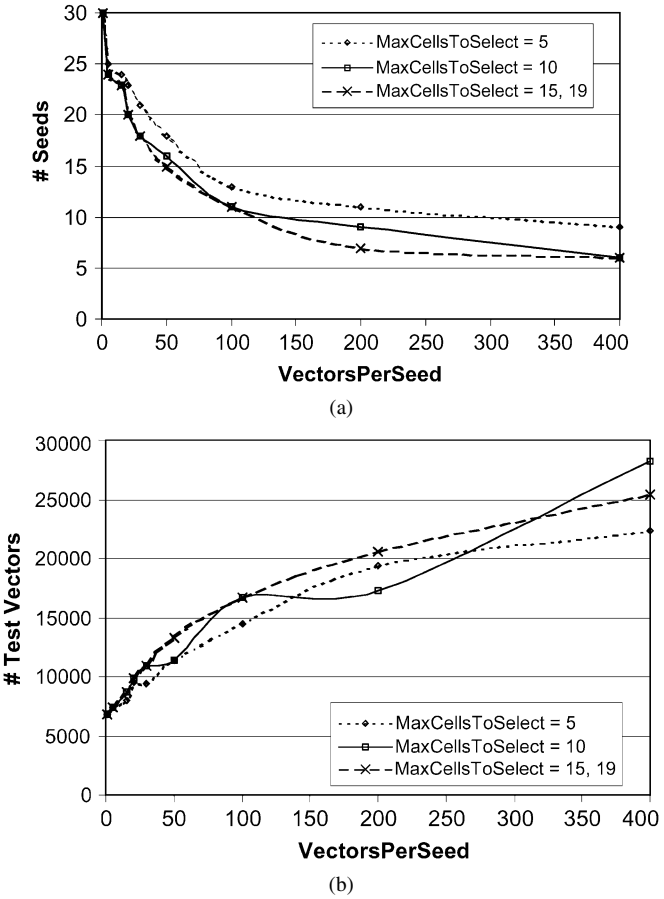

Fig. 10. (a) Seed volume results for various values of parameters *VectorsPerSeed* and *MaxCellsToSelect*. (b) Test sequence length results for various values of parameters *VectorsPerSeed* and *MaxCellsToSelect*.

For both parameters *VectorsPerSeed* and *MaxCellsToSelect*, there is a saturation point such that an increase of their values after that point slightly contributes to the seed volume results. For the experiments presented in Fig. 10(a), the saturation point of parameter *VectorsPerSeed* seems to be approximately 200 for *MaxCellsToSelect* $= 15$ and 19 and greater than 200 for the rest values of *MaxCellsToSelect*. In general, the saturation point of *VectorsPerSeed* gets higher as *MaxCellsToSelect* decreases (the same holds for *MaxCellsToSelect* as the value of *VectorsPerSeed* decreases). For parameter *MaxCellsToSelect*,

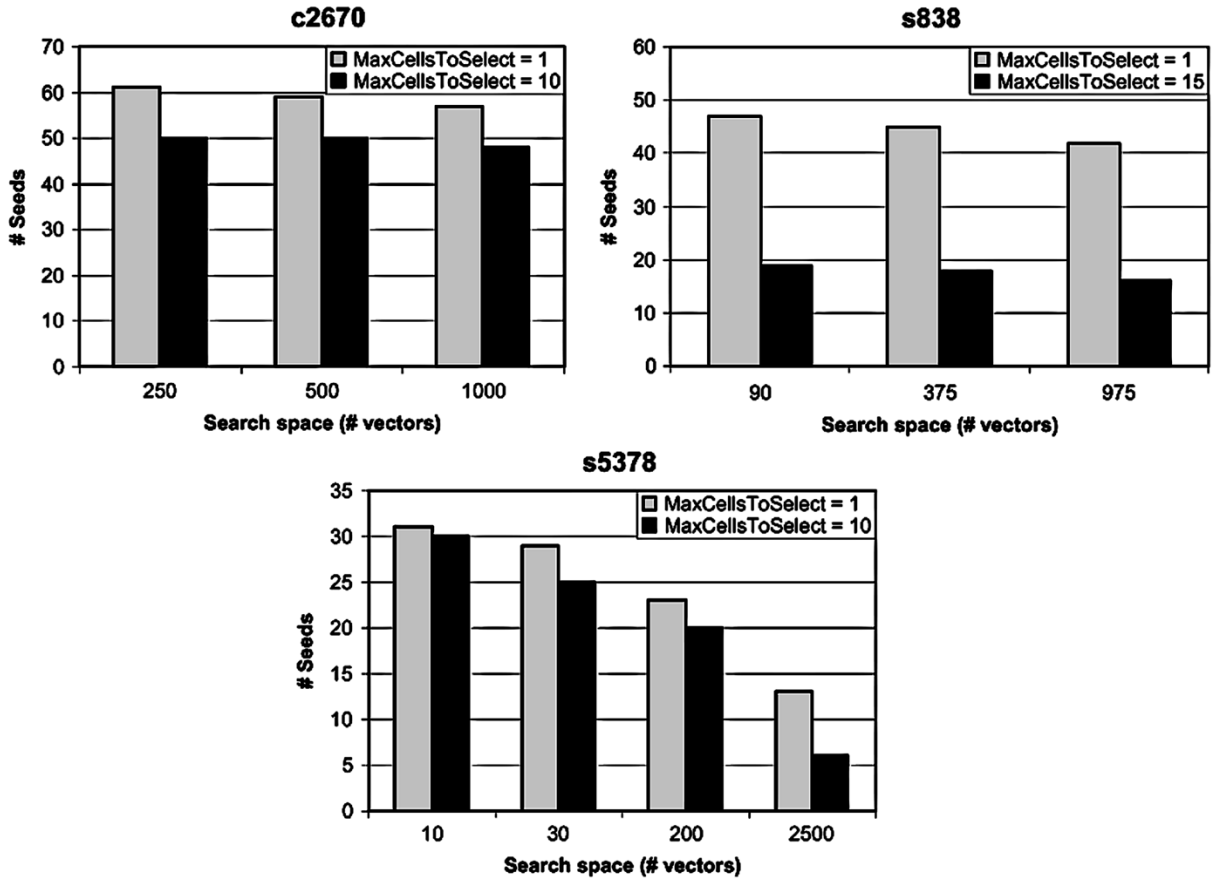

Fig. 11. Seed volume comparisons for *MaxCellsToSelect* $= 1$ and 10 or 15.

the saturation starts from value 10, while for values greater than 15 any further seed reduction cannot be achieved [the curves of "*MaxCellsToSelect* $= 15$" and "*MaxCellsToSelect* $= 19$" coincide in Fig. 10(a)]. We should also note that, for both the considered parameters, when the saturation point is exceeded, any further increase in their value does not affect the resulting test-sequence length significantly. In the case that *MaxCellsToSelect* is saturated, the seed-selection algorithm simply does not pick any additional cells. That is why the curves of "*MaxCellsToSelect* $= 15$" and "*MaxCellsToSelect* $= 19$" coincide also in Fig. 10(b). Furthermore, any additional vectors per seed, when the value of the corresponding parameter is saturated, usually cause a decrease in the number of utilized cells, which keeps the test sequence length at similar levels to those for smaller values of parameter *VectorsPerSeed*. This explains the fact that the "# Test Vectors" curves in Fig. 10(b) seem to converge to an upper limit. The anomalies of the "*MaxCellsToSelect* $= 10$" curve in Fig. 10(b) (it should be close to the "*MaxCellsToSelect* $= 15$" and above the "*MaxCellsToSelect* $= 5$" curve) are mainly due to the test sequence reduction procedure. More of the initial pseudorandom patterns are eliminated for *VectorsPerSeed* $= 50$ and 200 than in the corresponding cases for the rest values of *MaxCellsToSelect*. Finally, for *VectorsPerSeed* $= 400$ the algorithm takes advantage of the big pattern window (it manages to keep the number of utilized cells equal to 10), reducing the seed count a little more [Fig. 10(a)] and, as a result, the test sequence length increases again.

For demonstrating the benefit from the use of multiple LFSR cells instead of 1, we have also performed some experiments for c2670, s838, and s5378 with *MaxCellsToSelect* $= 1$. While keeping the size of the search space (i.e., the product of parameters *MaxCellsToSelect* and *VectorsPerSeed*) constant, we have conducted the same experiments setting *MaxCellsToSelect* $= 10$ for c2670 and s5378 and *MaxCellsToSelect* $= 15$ for s838. The results are shown in Fig. 11. The gain in seed volume is obvious. For c2670, there is a constant reduction of about 16.5% (approximately ten seeds), for s838, the mean gain is equal to 60.5%, while for s5378, the gain gradually increases to reach 54% for $\text{search space} = 2500$ vectors. We should mention that for c2670 and s838, the test sequence reduction procedure did not manage to reduce the test sequences for the experiments with $\text{MaxCellsToSelect} = 1$, which means that, due to the increased seed volume, they are much longer than the corresponding test sequences for greater values of *MaxCellsToSelect*. We furthermore note that for the above experiments, in order to demonstrate the advantage of using multiple LFSR cells more clearly, smaller values than 10 000 vectors have been used for parameter *VectorsForEasyFaults* (2000, 5000, and 7000 for c2670, s838, and s5378, respectively), while for c2670 an LFSR of length 55 has been used. Similar experiments with similar results have been also performed for the multiple scan chain case.

The most recent and efficient mixed-mode BIST reseeding techniques in the open literature are those presented in [25], [26], [30], [31], and [32]. As explained in the introduction, the

TABLE II
TEST-DATA STORAGE COMPARISONS

Circuit	2-D Compression [26]			Dynamic reseeding of [30]			Dynamic reseeding of [31]			Dynamic reseeding of [32]			Proposed technique		
	LFSR length	# Seeds	ROM bits	LFSR length	# Seeds	ROM bits	LFSR length	# Seeds	ROM bits	LFSR length	# Seeds	ROM bits	LFSR length	# Seeds	ROM bits
c2670	37	28	1036	-	-	-	-	-	-	-	-	-	66	27	1782
c7552	133	36	4788	-	-	-	-	-	-	-	-	-	100	34	3400
s420	16	10	160	-	-	-	-	-	-	-	-	-	25	5	125
s641	16	4	64	-	-	-	-	-	-	-	-	-	22	2	44
s713	16	4	64	-	-	-	-	-	-	-	-	-	22	2	44
s838	33	26	858	-	-	-	-	-	-	-	-	-	45	12	540
s953	10	2	20	-	-	-	-	-	-	21	9	189	15	1	15
s1196	10	3	30	-	-	-	-	-	-	15	13	195	17	2	34
s1238	14	3	42	-	-	-	15	7	105	15	12	180	17	2	34
s5378	14	14	196	61	20	1220	61	7	427	61	26	1586	19	10	190
s9234	40	95	3800	80	62	4960	80	57	4560	80	82	6560	44	82	3608
s13207	18	58	1044	45	26	1170	45	12	540	45	30	1350	22	21	462
s15850	30	112	3360	150	38	5700	150	35	5250	-	-	-	47	66	3102
s38417	42	267	11214	200	62	12400	-	-	-	-	-	-	86	116	9976
s38584	49	59	2891	200	36	7200	200	16	3200	-	-	-	51	23	1173

folding counter approach of [25] requires a costly, nonstandard scan chain configuration. This problem is solved in [26] and therefore, we compare the proposed method against the 2-D Compression approach of [26] and the dynamic reseeding techniques of [30]–[32].

In Table II, the test-data storage comparisons are presented. A dash (-) in the comparison tables means that no result has been provided by the authors of the referenced paper for the corresponding benchmark circuit. The results of Table II validate that the encoding ability of an LFSR seed is enhanced when it is utilized by multiple LFSR cells. Compared to the 2-D Compression technique of [26], which requires the smallest test-data storage among all the reseeding techniques in the open literature, we observe that the gain in ROM bits is significant. This gain is achieved regardless of the fact that, for all circuits except for c7552, a larger LFSR is required by the proposed technique. In most cases, such a larger LFSR is necessary due to the greater maximum number of defined bits $(s_{\max})$ in the test cubes we have used in our experiments (we can estimate the value of $s_{\max}$ for the cubes of [26] from the length of the corresponding LFSRs). The advantage of the proposed scheme over that of [26] is that by adjusting parameters *VectorsPerSeed* and *MaxCellsToSelect* as well as the LFSR length, various test-data storage and test sequence length results can be derived. For example, if we can tolerate an increase in the test sequence length, we can "allow" the seed-selection algorithm to handle more *VectorsPerSeed* and/or *MaxCellsToSelect* in order to achieve better compression. On the contrary, in the approach of [26], an LFSR is used in order to encode initial folding counter states that will generate specific folding counter sequences. Thus, an increase in the LFSR length is not directly linked with better compression results but just with more options on the folding counter sequences that will be generated. Also, in [26], the number of vectors applied to the CUT starting from a new seed is fixed, so no "test-data storage—test sequence length" tradeoff is possible via any parameter adjustment.

As for the techniques of [30]–[32], although they use much larger LFSRs (2.2 times on average), they cannot match the results of the proposed method. The average test-data reduction of the proposed technique over those of [30]–[32] is 53.5%, 43.8%, and 75.8%, respectively. We should note that the technique of [32] encodes some seeds as Bit-Counter states by using some combinational logic. This logic may be of significant size if the number of encoded seeds is high. However, in the comparisons we have taken into account, only the seeds that are loaded in the LFSR.

TABLE III
TEST SEQUENCE LENGTH COMPARISONS (# TEST VECTORS APPLIED TO THE CUT)

Circuit	2-D Compression [26]	Dynamic reseeding of [30]	Dynamic reseeding of [31]	Proposed technique
c2670	16552	-	-	8980
c7552	17488	-	-	11715
s420	10350	-	-	5954
s641	10220	-	-	12450
s713	10220	-	-	12450
s838	11742	-	-	4625
s953	10092	-	-	8352
s1196	10099	-	-	8058
s1238	10099	-	832	7840
s5378	13010	27648	1920	12760
s9234	33560	76800	4992	37675
s13207	50658	60416	2688	46400
s15850	78544	45056	3904	56160
s38417	454555	89088	-	262416
s38584	96435	76800	2624	38947

As far as the rest of the implementation logic is concerned, the proposed technique, compared to that of [26], requires an additional counter (the Cell Selection Counter), which is relatively small (up to five bits for the experiments presented in Table I) and the multiplexer, while the technique of [26] incorporates an additional comparator. Thus, the area overhead of the TPGs of the two techniques, excluding the ROM, is similar. Compared to the techniques of [30]–[32], the proposed one requires additionally the Reseeding Counter, the Cell Selection Counter and the multiplexer, but the use of much smaller LFSRs overbalances the additional hardware overhead of the extra counters and the multiplexer.

From the test sequence length comparisons presented in Table III, the dynamic reseeding technique of [32] has been omitted, since no test sequence length results have been provided by the authors of this work. As can be seen in Table III,

the proposed technique leads in most cases to shorter test sequences than those of [26] and [30]. This is mainly due to the fact that the proposed seed-selection algorithm achieves high test-data compression by using relatively few *VectorsPerSeed*. We remind that two out of the four parameters that affect the test sequence length of the proposed technique (relation 1, Section III) are the number of reseedings R and the number of test vectors generated between reseedings (*VectorsPerSeed*). The high value of parameter *VectorsPerSeed* for some of the smaller circuits is overcompensated by the very small number of seeds required for fully testing them.

As for the test sequence reduction procedure, its main contribution is that it reduces the initial *VectorsForEasyFaults*. This reduction may be significant for circuits that contain just a few hard-to-detect faults, but for bigger circuits with many hard faults it is not so important, since the part of the initial pseudorandom patterns that is eliminated is usually a small portion of the whole test sequence. We should mention that the regularity of the proposed architecture limits the reduction of *VectorsPerSeed* and the number of cells p, which were selected for feeding the scan chain of the CUT. If, for example, *VectorsPerSeed* is set to 50 and there is just one hard-to-detect fault that is tested by vector 50, while all the others are tested by vectors 1 to 25, all 50 vectors must be applied for all R seeds and from all p cells. A similar example can also be provided for demonstrating the limitations of the LFSR cell reduction procedure. Specifically, for the experiments of Table I, the test sequence reduction procedure achieved a 17.27% average reduction of the initial pseudorandom sequence, a 12.89% reduction of the selected cells, a 6.51% reduction of *VectorsPerSeed* and a 15.59% reduction of the total test sequence. However, the same reductions for the five larger ISCAS'89 and the two ISCAS'85 benchmark circuits that contain many hard-to-detect faults are 16.52%, 3.92%, 0.18%, and 7.12%, respectively. As we have expected there was no seed reduction in any experiment. We should also note that in the experiments we performed, our main target was to reduce the number of the seeds. For many circuits, with just a few more seeds, we could get much smaller test sequences.

The technique of [31] requires impressively shorter test sequences, compared to any other reseeding technique in the open literature. This can be explained by observing Fig. 12. In these two diagrams, we present, for both the proposed multiphase technique and that of [31], the number of initial pseudorandom patterns applied to the CUT (upper diagram) as well as the number of test cubes, which test the remaining hard-to-detect faults (lower diagram). For the proposed technique, we provide the length of the initial pseudorandom sequence after the test sequence reduction procedure and the number of extracted test cubes after the fault-simulation preprocessing step. The number of initial pseudorandom patterns for the dynamic reseeding technique of [31] can be easily derived by subtracting from the test sequence length results of the regular mixed-mode technique provided in [31], the number of seeds for detecting the corresponding hard faults (one seed per pattern is assumed).

From Fig. 12, it is obvious that for all circuits, a much smaller initial pseudorandom sequence has been used in the experiments of [31] than in those of the proposed technique. Also, after the application of the initial pseudorandom patterns, much fewer test cubes are extracted in [31] for the compared circuits that contain many hard-to-detect faults (s9234, s13207, s15850, s38584). Therefore, the great difference in their test sequence length results can be primarily attributed to the great difference of the initial conditions of the performed experiments (initial pseudorandom patterns volume and number of test cubes used by the reseeding algorithm) that favors the technique of [31]. For the two circuits (s1238 and s5378) that fewer test cubes are handled by the proposed algorithm, the difference in the test sequence lengths is due to the difference in the initial test sequences combined with the high value of parameter *VectorsPerSeed* we have chosen for optimizing the seed volume results (our primary objective was to reduce the test-data volume). We should mention, however, that a highly compressed test-cube set by the ATPG tool, may be good in terms of test sequence length, but requires LFSRs of greater length in order to be encoded. It also confines the solution space of the seed-selection algorithm (since its test cubes probably contain few don't care bits). Therefore, a less compressed test-cube set may lead to better test-data storage results. Unfortunately, we were not able to confirm this assertion, since ATALANTA does not have any test-cube compression (fault merging) capabilities.

Let us now present some experimental results for the larger of the ISCAS'89 benchmark circuits, assuming that the CUT contains multiple scan chains. For conducting the corresponding experiments, we have used the same LFSRs and parameter values as those for the single scan chain case of Table I. The phase shifters have been obtained by using the fast synthesis algorithm presented in [41] and the number of XOR taps of the XOR trees of each phase shifter (excluding the extra input) has been chosen to be equal to 3. The total number of seeds and the final test sequence lengths after the application of the test sequence reduction procedure, for various scan chain volumes, are shown in Table IV.

As we have expected, the results for the single and the multiple scan chain cases are similar. In fact, for all circuits, with the exception of s15850, the results for multiple scan chains are, in most cases, better than those for a single chain. This can be attributed to the existence of the phase shifter. For s38417, the multiple scan chains results are significantly better for all the scan chain volume cases than that for a single scan chain. The reason is the same, but, probably, when a lot of test cubes are considered as in this case, the improvement of the encoding ability of the scheme, due to the existence of the phase shifter, is more visible in the final seeds results. On the other hand, from the results of s15850, we deduce that there are circuits for which the application of the multiphase technique in a multiple scan environment does not offer the same encoding ability as in the single scan case (the difference is minor, however). We also observe that among the experiments for the same circuit, the seed-volume results are essentially the same. This demonstrates the robustness of the proposed multiphase scheme and reseeding algorithm. Any differences can be primarily attributed to different initial conditions (the number of test cubes remained after the fault-simulation preprocessing step). Finally, we should note that the discrepancy in the test sequence length results of

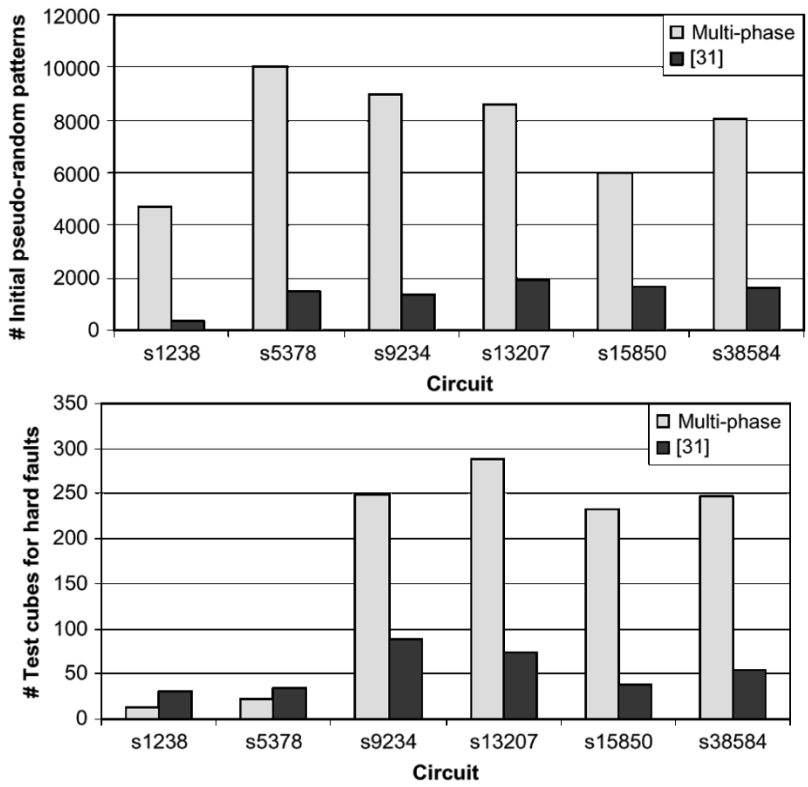

Fig. 12. Initial condition comparisons between the proposed technique and that of [31].

TABLE IV
EXPERIMENTAL RESULTS FOR MULTIPLE SCAN CHAINS FOR THE LARGER ISCAS'89 BENCHMARK CIRCUITS

Circuit	Multiple scan chains			Single scan chain	
	# Scan chains	# Seeds	# Vectors	# Seeds	# Vectors
s9234	5	77	35428	82	37675
	10	78	36595		
	15	74	36021		
	20	78	35246		
s13207	8	21	43912	21	46400
	16	21	42979		
	24	22	45110		
	32	20	45047		
s15850	8	70	62975	66	56160
	16	70	60117		
	24	70	61934		
	32	71	63125		
s38417	8	102	226006	116	262416
	16	95	213105		
	24	102	228799		
	32	101	228075		
	64	104	235618		
s38584	8	22	43365	23	38947
	16	21	41812		
	24	22	45022		
	32	22	42845		
	64	21	42188		

s38584 between the single and multiple scan chain cases is due to the high efficiency of the test sequence reduction procedure in the single scan chain case (for the corresponding experiment).

VII. DYNAMIC RESEEDING SCHEME

In this section, we demonstrate how the proposed architecture can be combined with the dynamic reseeding scheme of [28], in a full BIST environment, for further reducing the hardware requirements of the BIST circuitry. According to the dynamic reseeding approach, the reseeding operation, that is, the alteration of the normal state sequence $A \rightarrow B$ of the LFSR to the sequence $A \rightarrow B'$, with $B' \neq B$, is performed dynamically when the LFSR passes from state A to state B, by inverting the logic value of the bits of state B which are complementary to those of B'. The required inversions are performed by means of additional XOR gates controlled by a combinational module, as will be explained in the following section. When dynamic reseeding is adopted, the flexibility of a ROM-based approach (store-and-generate) is exchanged with reduced BIST implementation area.

The dynamic reseeding scheme was first proposed for circular BIST in [46]. A more efficient approach, which uses decoding of the pattern counter states to trigger the inversions, was presented for both LFSR- and accumulator-based test-per-clock BIST schemes in [47]–[49]. Later, it was applied to scan-based BIST schemes as well [28], [29]. At the same time period with the publication of [29], two scan-based techniques based on dynamic reseeding were presented in [30] and [31], the latter of which was enhanced in [32]. All three techniques of [30]–[32] perform the necessary inversions by using $2 \rightarrow 1$ multiplexers instead of XOR gates.

The rest of this section is organized as follows. For the shake of completeness, the basic concepts of dynamic reseeding [28] are presented in Section VII-A. In Section VII-B, an explanation of why the dynamic reseeding scheme is preferable to a ROM is provided, as well as why it suits the proposed multiphase architecture well. Finally, in Section VII-C the effectiveness of the dynamic reseeding scheme in terms of area overhead is evaluated with synthesis results.

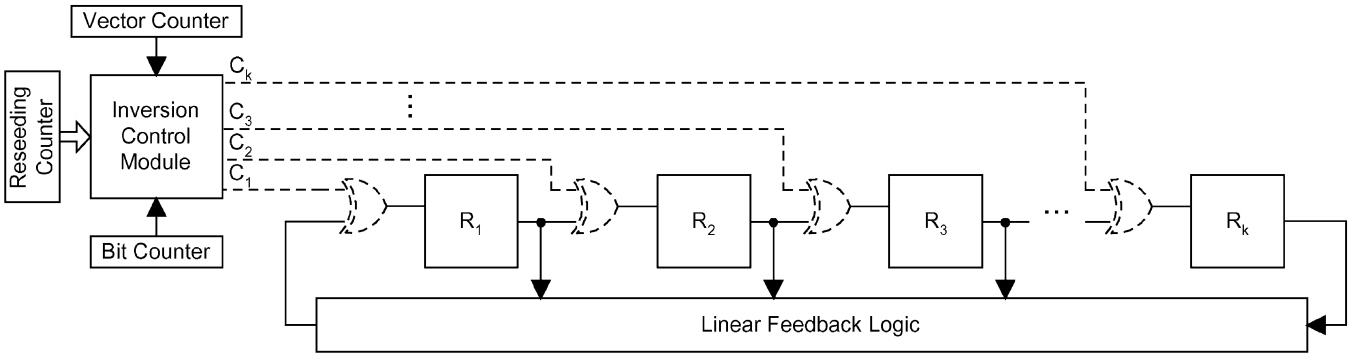

Fig. 13. Dynamic reseeding scheme.

A. Dynamic Reseeding

The dynamic reseeding scheme along with the counters of the multiphase architecture that control it are shown in Fig. 13. The reseeding operation is performed by inverting, at certain clock cycles, the outputs of some of the LFSR cells before being stored to their adjacent cells. This is achieved by means of additional XOR gates, which are placed between the cells of the LFSR, as shown in Fig. 13 (these XOR gates are drawn using dashed lines). We observe that one of the inputs of each of the additional XORs is driven by the output of the previous LFSR cell, while the other one is driven by an output C_i of the inversion control module, which is a combinational module as mentioned earlier. C_i is responsible for controlling all the inversion operations of cell i, with $1 \leq i \leq k$. When the multiphase architecture of Figs. 5 or 8 is implemented using the dynamic reseeding scheme of Fig. 13, the Seed-loading Mechanism of the former is comprised of the Inversion Control Module and the additional XOR gates.

We assume that the LFSR changes state at clock times $t_1, t_2, \ldots, t_j, \ldots$. The time interval T, with $t_{j-1} < T \leq t_j$, actually represents the clock cycle j, where, between t_{j-1} and t_j, a new state of the LFSR is generated. This new state is stored in the LFSR flip-flops exactly at clock time t_j. For $C_i = 0$, no inversions occur and the LFSR changes state at each clock time according to its feedback structure. If a reseeding must take place at clock time t_j (when Bit Counter $= n - 1$ and Vector Counter $=$ *VectorsForEasyFaults* $- 1$), some of the control lines C_i must be set to the logic value 1 in the time interval between t_{j-1} and t_j. During this interval (i.e., clock cycle j) the state of the Reseeding Counter (rc_{j-1}) is captured by the Inversion Control Module and a proper subset S_j of the lines $C_1, C_2, \ldots, C_k$ is activated, while the rest control lines are left to 0. In that way, at clock time t_j, the flip-flops $R_1, R_2, \ldots R_k$ receive the new state, which is inverted at the bit positions that correspond to the lines of S_j and, therefore, a new seed is stored in the LFSR register. After the reseeding is performed, the control lines of S_j are reset by the Inversion Control Module and the LFSR resumes its normal operation.

Let us now demonstrate how the seed calculated in Section II $(0\underline{1}11)$ will be loaded in the LFSR if it is to replace state $0\underline{1}01$ (we assume that 0101 is the state that would have been loaded in the LFSR if it was let run in autonomous mode). What we should do is to invert the third bit of the LFSR (since the new seed and the state to be replaced differ in the third bit) before it is stored in the third cell. This can be done by using a XOR gate between the second and the third LFSR cell and by activating its control line in the clock cycle before state 0101 is stored in the LFSR register. This is shown in Fig. 14.

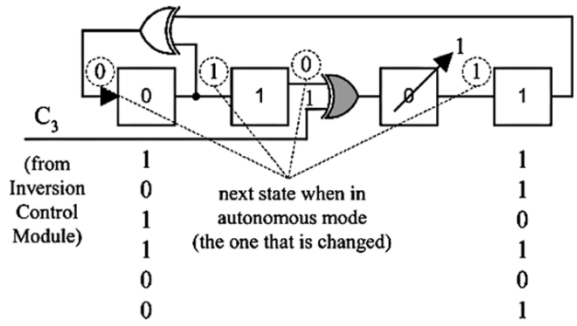

Fig. 14. Example of the application of the dynamic reseeding scheme.

As can be seen, the Inversion Control Module synchronizes the reseedings according to the values of the Bit and the Vector Counters and loads the appropriate seed according to the value of the Reseeding Counter. Concerning the inverting XOR gates, such a gate is usually used in more than one reseeding, while there may be cases in which less than k XORs may be sufficient for producing all the necessary seeds. That is why in Fig. 13 these gates are drawn using dashed lines. It is also obvious that, although for the above description we have used an external-XOR LFSR, the dynamic technique can be applied to internal-XOR LFSRs in a straightforward manner.

B. Effectiveness of the Dynamic Reseeding Scheme

However, the question stil remains. Why should someone choose the dynamic reseeding scheme instead of a ROM or, equivalently, why the dynamic reseeding scheme leads to better hardware overhead results than a ROM? There are two major advantages of the dynamic reseeding approach over the implementations based on a ROM. The first one is that, at each reseeding, only the LFSR cells that their values need to be inverted are handled. When a ROM is used, due to its regular structure, some area is required for every stored bit. On the contrary, in the case of dynamic reseeding, logic needs to be added only if a bit's value has to be inverted during a reseeding. Also, various LFSR cells share common parts of that logic and, as a result, a single copy is generated for them by the synthesis tool. Consequently, the hardware overhead requirements of the inversion control module are further reduced.

The second advantage of the dynamic reseeding scheme is not inherent like the previous one and relies on the fact that there are more ways for reducing its hardware overhead than the classical LFSR length and seed minimization, which are also valid in the case of a ROM. The most profitable way has to do with the periodicity of the reseedings. It is obvious that when there are no separate Vector and Reseeding Counters but just a single pattern counter which controls the reseedings [28], [30]–[32], the hardware overhead of the Inversion Control Module can be reduced when the pattern distance between subsequent reseedings is constant and equal to a power of 2. This way the logic that decodes the reseeding states of the pattern counter is minimized. In the proposed multiphase architecture we have chosen to use separate Vector and Reseeding Counters, so as both, the reseeding counter to be as small as possible, as well as to be able to define the size of the pattern window between reseedings freely. Another way would be to try to minimize the number of inversions required for performing the reseedings, either by selecting the most appropriate seeds (those that require the fewer inversions) [28], or by taking advantage of the undefined variables after having selected a seed. Such optimization steps have not been adopted in the proposed method due to the "seed volume reduction" objective of the seed-selection algorithm and the existence of the test sequence reduction procedure, respectively.

Although the dynamic reseeding scheme suits the proposed multiphase architecture well, it is not straightforward to assume the same for every reseeding technique. For example, let us consider the 2-D Compression approach of [26], which is the best reseeding technique in the literature, in terms of test-data storage requirements. If the dynamic reseeding scheme had been used in conjunction with the 2-D Compression technique, then two different sets of inversions would have had to be implemented for each reseeding. The reason is that each seed s_i has to be loaded $n+1$ times in the LFSR (n is the scan chain length) before the next one is loaded. This means that state s_i' must be inverted in some bit positions n times so as to be equal to seed s_i and once in some other bit positions so as to be equal to s_{i+1}, which is the next seed to s_i. s_i' is the state of the LFSR that will be modified by reseeding, after seed s_i has expanded to the required folding counter state. The proposed architecture would have had the same problem if each seed had been used from all selected source cells before proceeding to the next one. That is why we have chosen to apply all reseedings while feeding the scan chain from a single cell. However, if we adopt a similar approach for the 2-D Compression technique, its folding controller needs to be modified (an additional reseeding counter should be added that will also control the index counter). From the above, we conclude that the proposed multiphase architecture can be implemented equally easily when it is combined either with the dynamic reseeding scheme or with a ROM.

C. Evaluation of the Dynamic Reseeding Scheme

In this section, for demonstrating the effectiveness of the dynamic reseeding scheme, we provide hardware overhead comparisons for both ROM and dynamic reseeding implementations of the proposed multiphase architecture, for the experiments presented in Section VI for the single scan chain case (Table I).

TABLE V
HARDWARE OVERHEAD COMPARISONS BETWEEN ROM-BASED AND DYNAMIC RESEEDING IMPLEMENTATIONS OF THE PROPOSED ARCHITECTURE

Circuit	ROM-based approach		Dynamic reseeding scheme		Reduction (%)
	ROM bits	h/w overhead (ROM & MUXes - gate equivalents)	Additional XORs	h/w overhead (Inv. Control Module & XORs - gate equivalents)	
c2670	1782	525	66	300	42.86
c7552	3400	970	100	520	46.35
s420	125	61	24	48	21.96
s641	44	37	19	26	29.41
s713	44	37	19	26	29.41
s838	540	189	45	123	35.13
s953	15	22	11	16	28.28
s1196	34	29	15	22	22.49
s1238	34	29	16	23	19.03
s5378	190	70	19	48	31.72
s9234	3608	955	44	485	49.25
s13207	462	142	22	86	39.25
s15850	3102	832	47	440	47.11
s38417	9976	2597	86	1380	46.87
s38584	1173	354	51	214	39.57

Our goal is to validate that when a full BIST solution is targeted, the dynamic reseeding approach constitutes the best solution in terms of area required for implementing the BIST circuitry. The corresponding results are shown in Table V.

For the calculation of the hardware overhead of the dynamic reseeding scheme we have used a commercial synthesis tool for synthesizing the Inversion Control Module and, to the synthesis results, we have added the hardware overhead of the XOR gates that perform the inversions. We should note that one gate equivalent corresponds to a two-input NAND gate. For translating the ROM bits of the ROM-based approach to gate equivalents, we have taken into account the estimation of [50] that, on average, 0.25 gates are required for each memory cell of a ROM. Also, the hardware overhead of the multiplexers required for loading a seed from the ROM to the LFSR has been calculated.

As can be seen in Table V, the hardware overhead gain is significant even for circuits with few hard-to-detect faults that need a small number of seeds in order to be tested. We should mention that since, in most cases, the number of additional XORs is equal to the LFSR length and, as a result, equal to the number of multiplexers used in the ROM-based case for loading the seeds, the area gain is mainly due to the effectiveness of the dynamic reseeding scheme. This is more evident in the case of circuits with many hard-to-detect faults (c2670, c7552, s838, s9234, s13207, s15850, s38417, and s38584), where the area gain concerning the Seed-loading Mechanism of the multiphase scheme ranges from 35.13% to 49.25%. Given that the proposed multiphase technique requires significantly less storage than the most current reseeding techniques in the open literature (Section VI), we conclude that, when combined with dynamic reseeding, it constitutes a very cheap BIST reseeding solution in terms of area overhead (the size of the synthesized combinational logic is directly analogous to the size of the corresponding ROM).

VIII. CONCLUSION

We have described a novel multiphase reseeding architecture for scan-based BIST. Multiple cells of the LFSR, which are used as TPG, feed the scan chain of the CUT in different test phases. Since the operation of the LFSR is identical in all of them, i.e.,

the LFSR passes through the same state sequence, the implementation cost of the proposed architecture remains low. Furthermore, the encoding ability of an LFSR seed is enhanced when the bit sequences generated by more than one LFSR cells are considered during its calculation. Thus, the reseeding algorithm that accompanies the multiphase architecture manages to reduce the number of necessary seeds for fully testing the CUT significantly, compared to the already known reseeding techniques. As a result, the proposed architecture is suitable when either a full BIST or a test resource partitioning approach is selected. When BIST is the desired solution, the designer can choose between the flexibility that an on-chip ROM offers and the area gain that can be achieved when the proposed architecture is combined with the dynamic reseeding scheme described in Section VII. In any case, the same generic multiphase architecture can be used, the efficiency of which has been demonstrated by experimental results and comparisons against some of the most effective reseeding techniques that have been presented so far in the open literature.

REFERENCES

[1] E. J. McCluskey, D. Burek, B. Koenemann, S. Mitra, J. Patel, J. Rajski, and J. Waicukauski, "Test data compression (ITC 2002 roundtable)," *IEEE Design Test Comput.*, vol. 20, pp. 76–87, Mar./Apr. 2003.

[2] A. Jas and N. A. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based designs," in *Proc. Int. Test Conf.*, Oct. 1998, pp. 458–464.

[3] A. Chandra and K. Chakrabarty, "Test data compression for system-on-a-chip usingGolomb codes," in *Proc. 18th IEEE VLSI Test Symp.*, Apr./May 2000, pp. 113–120.

[4] ——, "Frequency-directed run length (FDR) codes with application to system-on-a-chip test data compression," in *Proc. 19th IEEE VLSI Test Symp.*, Apr.–May 2001, pp. 42–47.

[5] A. Jas, J. Ghosh-Dastidar, and N. A. Touba, "Scan vector compression/decompression using statistical coding," in *Proc. 17th IEEE VLSI Test Symp.*, Apr. 1999, pp. 114–120.

[6] C. V. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," in *Proc. IEEE Int. Test Conf.*, Oct. 2002, pp. 321–330.

[7] A. Jas, J. Ghosh-Dastidar, M.-E. Ng, and N. A. Touba, "An efficient test vector compression scheme using selective Huffman coding," *IEEE Trans. Computer-Aided Design*, vol. 22, pp. 797–806, June 2003.

[8] I. Hamzaoglu and J. H. Patel, "Reducing test application time for full scan embedded cores," in *Proc. Int. Symp. Fault Tolerant Comput.*, June 1999, pp. 260–267.

[9] A. Jas and N. A. Touba, "Virtual scan chains: A means for reducing scan length in cores," in *Proc. 18th IEEE VLSI Test Symp.*, Apr./May 2000, pp. 73–78.

[10] S. M. Reddy, K. Miyase, S. Kajihara, and I. Pomeranz, "On test data volume reduction for multiple scan chain design," in *Proc. 20th IEEE VLSI Test Symp.*, Apr./May 2002, pp. 103–108.

[11] I. Bayraktaroglu and A. Orailoglu, "Test volume and application time reduction through scan chain concealment," in *Proc. Design Automation Conf.*, June 2001, pp. 151–155.

[12] W. Rao, I. Bayraktaroglu, and A. Orailoglu, "Test application time and volume compression through seed overlapping," in *Proc. Design Automation Conf.*, June 2003, pp. 732–737.

[13] J. Rajski, J. Tyszer, M. Kassab, N. Mukherjee, R. Thompson, K.-H. Tsai, A. Hertwig, N. Tamarapalli, G. Mrugalski, G. Eider, and J. Qian, "Embedded deterministic test for low cost manufacturing test," in *Proc. IEEE Int. Test Conf.*, Oct. 2002, pp. 301–310.

[14] P. Wohl, J. A. Waicukauski, S. Patel, and M. B. Amin, "Efficient compression and application of deterministic patterns in a logic BIST architecture," in *Proc. Design Automation Conf.*, June 2003, pp. 566–569.

[15] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-in Test for VLSI: Pseudorandom Techniques*. New York: Wiley, 1987.

[16] B. Koenemann, "LFSR-coded test patterns for scan design," in *Proc. Eur. Test Conf.*, Apr. 1991, pp. 237–242.

[17] S. Hellebrand, S. Tarnick, B. Courtois, and J. Rajski, "Generation of vector patterns through reseeding of multiple-polynomial linear feedback shift registers," in *Proc. IEEE Int. Test Conf.*, Sept. 1992, pp. 120–129.

[18] 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. Comput.*, vol. 44, pp. 223–233, Feb. 1995.

[19] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern generation for a deterministic BIST scheme," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 1995, pp. 88–94.

[20] N. Zacharia, J. Rajski, and J. Tyszer, "Decompression of test data using variable-length seed LFSRs," in *Proc. 13th IEEE VLSI Test Symp.*, Apr./May 1995, pp. 426–433.

[21] N. Zacharia, J. Rajski, J. Tyszer, and J. Waicukauski, "Two dimensional test data decompressor for multiple scan designs," in *Proc. IEEE Int. Test Conf.*, Oct. 1996, pp. 186–194.

[22] J. Rajski, J. Tyszer, and N. Zacharia, "Test data decompression for multiple scan designs with boundary scan," *IEEE Trans. Comput.*, vol. 47, pp. 1188–1200, Nov. 1998.

[23] K. Chakrabarty, B. T. Murray, and V. Iyengar, "Built-in test pattern generation for high-performance circuits using twisted-ring counters," in *Proc. 17th IEEE VLSI Test Symp.*, Apr. 1999, pp. 22–27.

[24] K. Chakrabarty and S. Swaminathan, "Built-in testing of high-performance circuits using twisted-ring counters," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 2000, pp. 72–75.

[25] S. Hellebrand, H.-G. Liang, and H.-J. Wunderlich, "A mixed mode BIST scheme based on reseeding of folding counters," in *Proc. IEEE Int. Test Conf.*, Oct. 2000, pp. 778–784.

[26] H.-G. Liang, S. Hellebrand, and H.-J. Wunderlich, "Two-dimensional test data compression for scan-based deterministic BIST," in *Proc. IEEE Int. Test Conf.*, Oct./Nov. 2001, pp. 894–902.

[27] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial LFSR reseeding," in *Proc. IEEE Int. Test Conf.*, Oct./Nov. 2001, pp. 885–893.

[28] E. Kalligeros, X. Kavousianos, and D. Nikolos, "A ROMless LFSR reseeding scheme for scan-based BIST," in *Proc. 11th Asian Test Symp.*, Nov. 2002, pp. 206–211.

[29] ——, "A highly regular multi-phase reseeding technique for scan-based BIST," in *Proc. 13th ACM Great Lakes Symp. VLSI*, Apr. 2003, pp. 295–298.

[30] A. A. Al-Yamani and E. J. McCluskey, "Built-in reseeding for serial BIST," in *Proc. 21st IEEE VLSI Test Symp.*, Apr./May 2003, pp. 63–68.

[31] A. A. Al-Yamani, S. Mitra, and E. J. McCluskey, "BIST reseeding with very few seeds," in *Proc. 21st IEEE VLSI Test Symp.*, Apr./May 2003, pp. 69–74.

[32] A. A. Al-Yamani and E. J. McCluskey, "Seed encoding with LFSR's and cellular automata," in *Proc. Design Automation Conf.*, June 2003, pp. 560–565.

[33] C. Barnhart, B. Keller, and B. Koenemann, "Logic DFT and test resource partitioning for 100 M gate ASICs," in *Proc. Int. Workshop Test Resource Partition.*, Oct. 2000.

[34] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, Nov. 1996, pp. 337–343.

[35] G. Kiefer and H.-J. Wunderlich, "Using BIST control for pattern generation," in *Proc. IEEE Int. Test Conf.*, Nov. 1997, pp. 347–355.

[36] G. Kiefer, H. Vranken, E. J. Marinissen, and H.-J. Wunderlich, "Application of deterministic logic BIST on industrial circuits," in *Proc. IEEE Int. Test Conf.*, Oct. 2000, pp. 105–114.

[37] N. A. Touba and E. J. McCluskey, "Bit-fixing in pseudorandom sequences for scan BIST," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 545–555, Apr. 2001.

[38] H. K. Lee and D. S. Ha, "Atalanta: An efficient ATPG for combinational circuits," Dept. Elect. Eng., Virginia Polytechnic Inst. State Univ., Blacksburg, Tech. Rep. 93-12, 1993.

[39] E. Kalligeros, X. Kavousianos, D. Bakalis, and D. Nikolos, "An efficient seeds selection method for LFSR-based test-per-clock BIST," in *Proc. Int. Symp. Quality Electron. Design*, Mar. 2002, pp. 261–266.

[40] G. H. Golub and C. F. Van Loan, *Matrix Computations*, 2nd ed. Baltimore, MD: Johns Hopkins Univ. Press, 1989.

[41] J. Rajski, N. Tamarapalli, and J. Tyszer, "Automated synthesis of phase shifters for built-in self-test applications," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1175–1188, Oct. 2000.

[42] F. Brglez, P. Pownall, and R. Hum, "Accelerated ATPG and fault grading via testability analysis," in *Proc. IEEE Int. Symp. Circuits Syst.*, June 1985, pp. 695–698.

[43] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," in *Proc. IEEE Int. Symp. Circuits Syst.*, May 1989, pp. 1929–1934.
[44] A primitive polynomial search program [Online]. Available: http://users2.ev1.net/~sduplichan/primitivepolynomials/primitive-polynomials.htm
[45] TCA Lab web page [Online]. Available: http://tca-lab.ceid.upatras.gr/cubes.zip
[46] N. A. Touba, "Obtaining high fault coverage with circular BIST via state skipping," in *Proc. 15th IEEE VLSI Test Symp.*, Apr./May 1997, pp. 410–415.
[47] X. Kavousianos, D. Bakalis, and D. Nikolos, "A novel reseeding technique for accumulator-based test pattern generation," in *Proc. 11th Great Lakes Symp. VLSI*, Mar. 2001, pp. 7–12.
[48] E. Kalligeros, X. Kavousianos, D. Bakalis, and D. Nikolos, "A new reseeding technique or LFSR-based test pattern generation," in *Proc. 7th Int. On-Line Test. Workshop*, July 2001, pp. 80–86.
[49] ——, "On-the-fly reseeding: A new reseeding technique for test-per-clock BIST," *J. Electron. Test.: Theory Applicat.*, vol. 18, no. 3, pp. 315–332, 2002.
[50] L. R. Huang, J. Y. Jou, and S. Y. Kuo, "Gauss-elimination-based generation of multiple seed-polynomial pairs for LFSR," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 1015–1024, Sept. 1997.

Emmanouil Kalligeros was born in Athens, Greece, in 1976. He received the Diploma in computer engineering and informatics and the M.Sc. degree in 1999 and 2001, respectively, from the University of Patras, Patras, Greece, where he is currently pursuing the Ph.D. degree.

His research interests include built-in self-test, test-data compression, and low-power testing.

Xrysovalantis Kavousianos (S'97–M'02) received the B.E. degree in computer engineering and informatics and the Ph.D. degree from the University of Patras, Patras, Greece, in 1996 and 2000, respectively.

He is currently a Lecturer in the Computer Science Department, University of Ioannina, Ioannina, Greece. His main interests are in the fields of very large scale integration design and digital testing. He is currently working in the areas of built-in-self-test, low power testing, and on-line testing.

Dimitris Nikolos (M'95) received the B.Sc. degree in physics, the M.Sc. degree in electronics, and the Ph.D. degree in computer science, from the University of Athens, Athens, Greece, in 1979, 1981, and 1985, respectively.

He is currently a Full Professor in the Computer Engineering and Informatics Department, University of Patras, Patras, Greece, and Head of the Technology and Computer Architecture Laboratory. He has authored or coauthored more than 130 scientific papers and holds one U.S. patent. His main research interests include fault-tolerant computing, computer architecture, VLSI design, and test and design for testability.

Prof. Nikolos has served as the Program Co-chairman of five IEEE Int. On-Line Testing Workshops (1997–2001). He also served on the program committees for the IEEE International Symposium on Defect and Fault Tolerance in VLSI systems (1997–1999), the Third and Fourth European Dependable Computing Conferences, and the DATE Conferences (2000–2004). He was a Guest Co-editor for the June 2002 special issue of the *Journal of Electronic Testing*, *Theory and Applications (JETTA)*, which was devoted to the 2001 IEEE International On-Line Testing Workshop. He also was co-recipient of the Best Paper Award for his work "Extending the Viability of IDDQ Testing in the Deep Submicron Era," which was presented at the Third IEEE International Symposium on Quality Electronic Design (2002).