Reseeding-based Test Set Embedding with Reduced Test Sequences

E. Kalligeros[1, 2], D. Kaseridis[1, 2], X. Kavousianos[3] and D. Nikolos[1, 2]
[1]*Computer Engineering & Informatics Dept., University of Patras, 26500 Patras, Greece*
[2]*Research Academic Computer Technology Institute, 61 Riga Feraiou str., 26221 Patras, Greece*
[3]*Computer Science Dept., University of Ioannina, 45110 Ioannina, Greece*
kalliger@ceid.upatras.gr, kaserid@ceid.upatras.gr, kabousia@cs.uoi.gr, nikolosd@cti.gr

Abstract

A novel technique for reducing the test sequences of reseeding-based schemes is presented in this paper. The proposed technique is generic and can be applied to test set embedding or mixed-mode schemes based on various TPGs. The imposed hardware overhead is very small since it is confined to just one extra bit per seed plus one very small counter in the scheme's control logic, while the test-sequence-length reductions achieved are up to 44.71%. Along with the test-sequence-reduction technique, an efficient seed-selection algorithm for the test-per-clock, LFSR-based, test set embedding case is presented. The proposed algorithm targets the minimization of the selected seed volumes and, combined with the test-sequence-reduction technique, delivers results with fewer seeds and much smaller test sequences than the already proposed approaches.

1. Introduction

The core-oriented way of designing contemporary Systems-on-a-Chip (SoCs) is placing a severe burden on the external Automatic Test Equipment (ATE) used for their testing. This core-based design style, although reducing the time-to-market and the complexity of the designers' task, leads to much larger and denser circuits which require greater test data volumes and longer test-application times in order to be properly tested. Consequently, the introduction of new techniques that overcome these problems is of great importance.

From the perspective of testing, the cores integrated in a SoC can be classified into two categories: those that are of known structure and those that are IP-protected and practically constitute a black box. For the former, fault simulation and/or test pattern generation can be performed, while the latter are just accompanied by a precomputed set T of test patterns that should be applied to their inputs so as to be tested. Three different approaches can be followed in order to reproduce a test set T that comes with a core of unknown structure: deterministic test set generation, test-pattern compression and test set embedding. In the first approach, an on-chip ROM or a deterministic Test Pattern Generator (TPG) [1] is used for precisely reproducing the test set of the Circuit (or Core) Under Test (CUT). In the second one, compressed versions of the test vectors [2] or the test cubes (test patterns with undefined bits) [3] of T are stored in the tester and decompressed on-chip by means of a small built-in circuit. Contrary to the two aforementioned approaches, test set embedding [4]-[7] encodes the test patterns of T in a longer TPG sequence.

This work was supported in part by the Public Benefit Foundation "Alexander S. Onassis" via its scholarships programs, and in part by the EPEAEK II "Pythagoras" program.

Similarly to Built-In Self-Test (BIST), test set embedding techniques can be classified as test-per-clock [4]-[6] or test-per-scan [7]. When a new pattern is generated at each clock cycle we refer to a test-per-clock scheme, while, in a test-per-scan scheme, the scan chain(s) of the CUT is (are) serially filled by the TPG. Another categorization can be made according to the test patterns of T that are handled. Some approaches operate on test cubes [5]-[7], while others manipulate fully specified patterns [4]. The advantage of test set embedding compared to the deterministic test set generation and the test-pattern compression approaches lies in its reduced hardware and test-data storage requirements, respectively. However, in some cases [5]-[7], this advantage is exchanged with excessively long test sequences. Therefore, test set embedding techniques that combine reduced overhead (hardware and test data storage) with short test sequences are required.

In this paper, a novel approach for significantly reducing the test sequences of reseeding-based schemes is presented. This test-sequence-reduction approach is generic in the sense that it can be applied to test-per-clock or test-per-scan, test set embedding or mixed-mode schemes, based on various TPGs (Linear Feedback Shift Registers-LFSRs, cellular automata etc.). Also, it can be used either in a BIST implementation or in a test-resource partitioning scenario, where the seeds are supplied by an external ATE. The proposed technique is based on partitioning the vector window of each seed into segments. The seeds are then reordered according to the number of useful segments their windows include. The overhead imposed by the proposed technique is confined to one extra bit per stored LFSR seed plus one very small counter in the scheme's control logic. In addition to the test-sequence-reduction technique, a very efficient seed-selection algorithm is presented for the case of test-per-clock, LFSR-based, test set embedding. For minimizing the number of selected seeds, apart from three heuristic criteria that have been presented in [8], two new criteria are introduced, which significantly refine the selection process and fairly reduce the selected seed volumes. We note that the seed-selection algorithm requires that test set T consists of test cubes. As explained in [9], much smaller test-data-storage results can be achieved if, instead of fully specified patterns, test cubes are handled. The seed-selection algorithm is presented first in Section 2, while the test-sequence-reduction technique is explained in Section 3. Both of them are evaluated with experimental results and comparisons in Section 4 and the paper is concluded in Section 5.

2. Seed-selection algorithm

For presenting the seed-selection algorithm, we consider the classical LFSR-based reseeding scheme, consisting basically of an LFSR of length n and a Vector Counter (n is the number of primary inputs of the CUT). The LFSR is loaded

serially with a new seed and is let generate states for L-1 clock cycles. That is, each seed is expanded to a window of L vectors (including the new seed), which are applied to the CUT and the corresponding responses are captured by the Test Response Compactor (TRC). The same process is repeated until all the test cubes accompanying the CUT are covered.

The seed-selection algorithm receives as inputs the size L of the window that each seed is expanded to and a test cube set T. Its goal is to select a number of LFSR seeds so as each test cube of T to be compatible with at least one of the vectors generated when the selected seeds are expanded to the corresponding vector-windows. The set of chosen seeds should be as small as possible.

For determining a new seed, the seed-selection algorithm makes use of the well-known concept of solving systems of linear equations [10]. That is, if each bit of the initial state (seed) of the LFSR were replaced by a binary variable, then each one of the L window-states would be equal to a symbolic vector. A symbolic vector is a vector, the bits of which are linear expressions comprising binary variables (those constituting the initial LFSR state) and/or a binary constant. Let $sv_i = t$ be the linear system which results from equating the defined bits of test cube t with the corresponding linear expressions of the ith symbolic vector of the window (sv_i). If this system can be solved, then a test vector compatible to test cube t can be generated at the ith window position (or, in other words, t can be covered or encoded at the ith window position). This is achieved by updating the initial LFSR state according to the solution of the system (i.e., assuming Gauss-Jordan elimination, all the variables belonging in the pivot columns of the system should be replaced by linear expressions of the free variables). The seed-selection algorithm examines all possible linear systems and chooses one in order to be solved. After variable replacement, the L symbolic vectors are regenerated starting from the updated initial LFSR state. The difference from the previous step is that the replaced variables are not included in the expressions of the symbolic vectors any more. A new test cube is selected to be covered at some window position and the above-described process is repeated until no system can be solved for any of the remaining test cubes. At this point a new seed has been determined. If any of the variables of the initial LFSR state have not been replaced by constant values during the linear-system solving process, they are set to a random value.

Since at each step of the algorithm, linear systems corresponding to more than one test cubes will be solvable at more than one positions of the examined window, a set of heuristics should be defined for selecting the system that will be actually solved. The proposed seed-selection algorithm utilizes the three basic criteria of the algorithm proposed in [8] for mixed-mode BIST, which are further enhanced with two additional ones. These two new criteria significantly refine the selection process, improving that way the encoding ability of the proposed algorithm and thus lead to better results in terms of the required seed volumes and the resulting test-sequence lengths. In the following, we briefly discuss the three criteria described in [8] and then we present in detail the two new ones.

The most important of the three heuristic criteria presented in [8] concerns the selection of a new system to be solved, at each step of the algorithm. The algorithm examines the linear systems for all the test cubes of set T, in all the L positions of the examined window. The system that is actually selected is the one that leads to the replacement of the fewest variables in the initial window state. The reason is that with more variables in the linear expressions of the symbolic vectors, more systems will be solvable at each step and thus more test cubes will be encoded in the windows of the selected seeds. The above rule is always held except for the case of the selection of the first test cube for each seed. This cube is covered at the first state of the examined window, i.e., its defined bits determine the values of the corresponding bits of the new seed. The test cube containing the maximum number of defined bits is the first to be selected for each new seed. This second criterion leads to better results in terms of the required seed volumes, due to the fact that cubes containing many defined bits (those selected first for each seed) are not left to be covered at the last stages of the seed-selection process. Such cubes are difficult to be encoded together in the same seed's window due to the great number of defined bits they include. Thus, if only cubes with many defined bits were left to be encoded in the final stages of the seed-selection process, many seeds would be needed in order to cover them (there may be cases in which only one cube would be encoded in one seed). For the same reason, the third criterion presented in [8] splits the test cubes of T into two different groups. The first one (high priority) consists of the cubes of T that contain many defined bits, while the other group (low priority) includes the remaining cubes. At each step of the algorithm the cubes of high priority are targeted first and only if no such cube can be covered, the algorithm proceeds to the cubes of the low-priority group. To sum up, for each new seed, the test cube containing the maximum number of defined bits is covered first at the first window position, while the remaining cubes are divided into two groups. Then the algorithm runs iteratively, examining the high priority cubes first, and selects at each step the linear system that eliminates the fewest variables in the initial LFSR state.

Although the application of the above heuristics leads to good results in terms of the resulting seed volumes, the main selection criterion is not elaborate enough. That is, at each step of the algorithm there are many solvable linear systems that require the same minimum number of variables to be replaced in the initial LFSR state. According to the seed-selection algorithm of [8], among those systems, the first one is selected. However, this is essentially a random choice that does not improve the algorithm's efficiency in any way. For that reason, this selection is refined with two new criteria. Their aim is to favor the selection of test cubes that are more difficult to be encoded along with others in the same seed's window.

As explained above, an indication of this cube-encoding difficulty is given by the number of defined bits a test cube contains. Therefore, this number is chosen to be the first of the two new selection criteria. Specifically, from the systems that require the elimination of the same minimum number of variables in order to be solved, those corresponding to the test cube with the maximum number of defined bits are preferred. From the preferred systems, the one, which is nearest to the initial state in the L-state window, is selected. This last choice favors the test-sequence-reduction procedure that will be presented in the following section. We should note that the number of defined bits is a "global" metric of how difficult is for a cube to

be encoded in the same seed's window along with another one. That is, independently of the test cubes that have already been encoded in the current window, if cube t_1 has fewer defined bits than t_2, then the probability that t_1 can be encoded along with another cube t_3 is higher than the probability of t_2 being able to be encoded together with t_3.

Since there may be minimum-variable-eliminating systems that correspond to more than one test cubes with the same maximum number of defined bits, a second criterion for selecting the proper system in such cases is used. According to this criterion, the systems corresponding to the cube, which can be covered at the fewest positions within the L-state window, are preferred. As above, from those systems, the one that is higher in the examined window is selected. In contrast with the previous one, this encoding-difficulty metric can be characterized as "local". That is, according to this criterion, the selection between test cubes t_1 and t_2 with the same number of defined bits totally depends on the test cubes that have been previously encoded in the same window. In other words, a different test-cube encoding history may lead to the selection of t_1 instead of t_2 or the opposite.

With the addition of the two above-described heuristic criteria, the encoding ability of the seed-selection algorithm has been significantly improved. Specifically, the resulting sets of seeds were up to 25% smaller than those derived after the application of the algorithm of [8] in the test set embedding case.

We should note that the running time of the seed-selection algorithm is rather moderate. It mainly depends on the length of the LFSR used, i.e., on the number of primary inputs of the CUT. For reducing the execution time, we mark during each step the window positions where a system cannot be solved for each test cube, in order not to try to solve it again in the following steps. We also stop solving a system, when, during its solution, more variables are replaced compared to the, up to that point, best system (the one that eliminates the fewest variables). These two run-time optimizations combined with the fact that the procedures for solving binary linear systems are much faster than those for solving conventional systems of linear equations [11], lead to run-times in the range of some seconds to few hours in a Pentium 4, 2.6 GHz workstation, for each of the experiments that will be presented in Section 4.

3. Test-sequence-reduction scheme

As it has been explained in the previous section, the seed-selection algorithm assumes a window of L successive LFSR states for each selected seed. Only some of the states of each window are actually being used for reproducing the test cubes of set T. One can easily understand that, if the last state of a window is not a useful one, i.e. no test cube has been selected by the seed-selection algorithm to be covered at that state, then all the states from the last useful one to the last window state are redundant (Figure 1). On the other hand, the useless states between two successive useful ones are necessary since they connect the two useful states in the LFSR sequence. Therefore, they cannot be removed without reseeding the LFSR (so as to bridge the gap created in the state sequence by the removed states). Moreover, as more seeds are selected by the seed-selection algorithm, more variables need to be replaced in order for the remaining test cubes to be covered and thus fewer cubes are encoded in each seed, leaving more useless states at the end of the corresponding windows. Due to the above-mentioned reasons we conclude that, usually, there will be a significant number of final-redundant LFSR states in each window. This fact negatively affects the required test application time. This problem is much more important in the case of test set embedding since the increased number of seeds, compared to the case where the CUT is of known structure, leads to much longer test sequences.

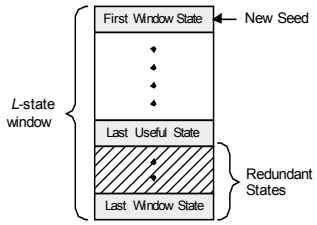

Figure 1. A window of L states

The most efficient way, in terms of test-sequence length, for eliminating those redundant final window-states is to stop the expansion of each seed after the clock cycle, in which the last useful state was generated by the LFSR. In that way the number of redundant states in each window (the useless states at the end of the window) will be equal to zero. Assuming that a Vector Counter is used for counting the states of each window, this "maximum reduction" approach requires Vector Counter to be initialized in a different value at each reseeding and consequently, the initialization values of the counter should be stored along with the corresponding seeds. Therefore, excessive test data storage may be required, especially when a long Vector Counter is needed. In order to overcome this inefficiency an intermediate approach is proposed.

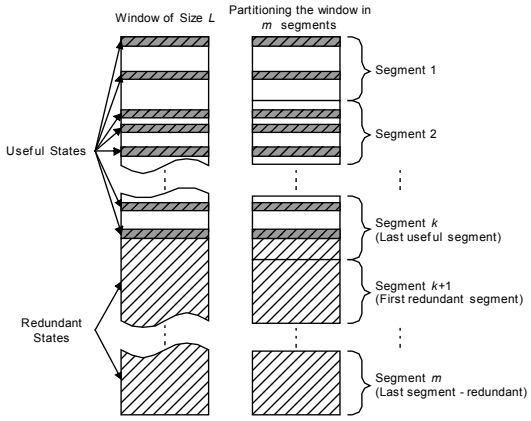

Figure 2. The proposed window segmentation technique

According to this approach, each window is segmented into a number of equal-sized groups of LFSR states. The generation of the states of each group (segment) is controlled by a counter called Segment-Vectors Counter, which counts from *Segment_Size*-1 to 0. The partitioning of a window into segments is shown in Figure 2. The useful states of the window are included in the first k segments, where the kth segment contains the last useful state. k is, most of the times, smaller than m (the total number of segments a window has been partitioned to) and thus the last m-k segments (those containing redundant states) can be dropped during test generation. Furthermore, with proper selection of the value of parameter *Segment_Size*, the distance between the last useful state and the end of the last

useful segment can be minimized. Both the above reasons assure that this intermediate approach eliminates the majority of redundant states that a window includes, having as upper limit (of the eliminated redundant states) those that the "maximum reduction" approach drops.

Having partitioned each window into segments, a low hardware-overhead solution for generating the useful segments of each window is required. Seed reordering could be helpful for coming up with such a solution. We remind that the selected seeds are independent of each other and therefore can be rearranged in any suitable order. The main idea behind the adopted approach is that if we order the seeds according to the number of useful segments they include and if the useful-segment volumes of two successive windows differ at most by one, then only a single extra bit per seed is necessary for indicating this relation. A zero value of that bit corresponds to the same number of segments between the current seed and its successor while a value equal to one indicates a one segment difference. The problem is that there will be cases for which the difference in the number of useful segments between two successive (ordered) seeds will be greater than one. In such cases, some of the useless segments should be maintained in the window with the smaller number of useful segments. The proposed seed rearrangement procedure is better explained with the example of Figure 3.

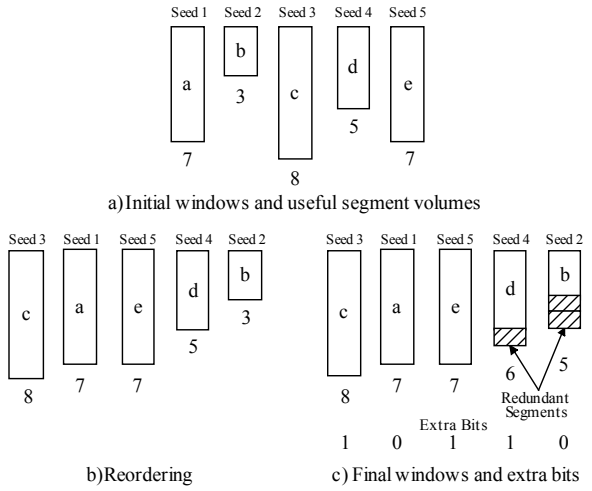

Figure 3. Rearrangement technique: a) Initial windows, b) Windows after the reordering, c) Final windows and extra bits

At first, the seeds are arranged in descending order according to the number of useful segments their windows include (Figure 3.b). After that, if there is any difference in the number of required segments between two successive windows, let say W_i and W_{i+1}, that is larger than one, then a number of redundant segments should be allowed in W_{i+1}, so as this difference to be reduced to one (Figure 3.c windows d and b). On the other hand, if the above difference is smaller than or equal to one then the number of segments remains unchanged (Figure 3.c windows c, a and e). As a final step, the value of the extra bit is calculated for each seed (one=next seed's window requires one segment less, zero=next seed's window requires the same number of segments). Although the described procedure decreases the effectiveness of the segment-partitioning scheme (due to the redundant segments allowed in some windows), the extra storage required for its implementation is cut down to just one bit per seed. Furthermore, by properly selecting *Segment_Size* we can reduce the number of allowed redundant segments, achieving test-sequence-length reductions that are very close to those of the "maximum reduction" approach, as will be demonstrated in Section 4.

For maintaining the necessary number of segments for each window, a down counter, called Load Counter, is used. This counter is initially loaded with the maximum number of segments required among all the windows, which is actually the number of segments of the first window in the rearranged order. The value of the extra bit of each seed determines the operation of Load Counter. An 1 triggers the counter decreasing its value by one, while a 0 leaves it unchanged (Load Counter maintains its previous value). That is, before being triggered, Load Counter contains the number of vector-segments that should be applied to the CUT, starting from the current seed. We note that Load Counter is triggered at most once for each seed, depending on the value of the corresponding extra bit.

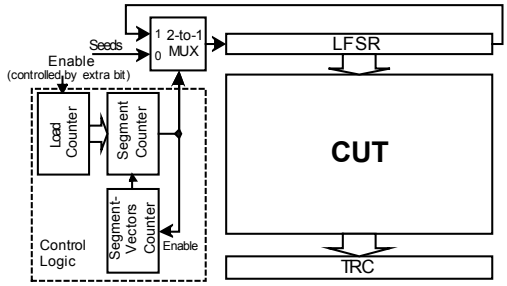

Figure 4. The proposed test-sequence-reduction scheme

The architecture that handles the operation of the proposed scheme is shown in Figure 4. As previously mentioned, Load Counter is controlled by the value of the extra bit of each seed and is responsible for maintaining the required number of segments for each window. In order to actually control the generation of the patterns of a window, two counters are needed. The first one is the Segment-Vectors Counter, which (as has already been mentioned) counts from *Segment_size*-1 to 0 and controls the generation of the vectors of a segment. The second counter has length equal to that of Load Counter, and is called Segment Counter. Segment Counter is responsible for counting the required number of segments for each window and thus it is initialized for each seed with the value of Load Counter. As can be easily seen, Segment and Segment-Vectors Counter constitute a combined counter. Segment Counter's value is decreased by one every time Segment-Vectors Counter signals that *Segment_Size* patterns have been applied to the CUT. That is, for every state of Segment Counter a full count down of Segment-Vectors Counter is carried out. When Segment Counter becomes equal to zero, the vectors of the current window have been generated and the expansion of the current seed stops (Segment-Vectors Counter is disabled). In order to generate the next window the following steps have to be carried out: the next stored seed is loaded in the LFSR, Segment Counter is loaded with current Load Counter's value, Load Counter is triggered (or not) according to the value of the seed's extra bit and Segment-Vectors Counter is enabled again (due to the initialization of Segment Counter to a value different from 0). The above-described procedure is repeated until all the seeds have been expanded to their corresponding vector-segments. We stress that Segment

Counter does not count down from #*Segments*-1 to 0 for each seed but from #*Segments*, since its zero value triggers the loading of the next seed.

A final comment that should be made about the proposed scheme is that, since Segment and Segment-Vectors Counters are combined to generate the vectors of a seed's window, the sum of their lengths cannot be much greater than the length of the Vector Counter of the classical reseeding approach. In fact it can be proven that their combined size is at most one bit larger than that of Vector Counter. Consequently, extra hardware overhead in the Control Logic of the proposed scheme is imposed only by the addition of Load Counter, which, as will be seen in the evaluation section, is very small (its length is equal to that of Segment Counter).

4. Evaluation and comparisons

In order to validate the effectiveness of the proposed scheme, we implemented the seed-selection and segmentation-rearrangement algorithms in C programming language and we conducted a series of experiments on both the ISCAS'85 and the (combinational part of) ISCAS'89 benchmark circuits. The corresponding test sets were obtained by using the Atalanta ATPG tool. We only considered circuits that are not completely tested with the application of 10000 pseudorandom patterns. The characteristic polynomials of the LFSRs were selected to be primitive.

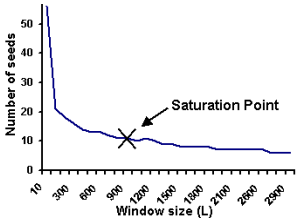

Figure 5. Number of seeds versus window size for s1238

The choice of the window size parameter L of the seed-selection algorithm significantly affects both the number of final selected seeds and the length of the resulting test sequences. Increasing the size of the window may reduce the required seeds due to the existence of more useful LFSR states in each window and thus the final test-sequence length may also be reduced. However, the number of required seeds is gradually saturated, as the value of parameter L increases. That is, although larger windows are used by the seed-selection algorithm, the number of required seeds will not be reduced at the same rate (or even at all after some point), having as a result larger test sequences. This is shown in Figure 5 (the saturation point is marked with an "X"). In order to achieve a good balance between the number of seeds and the test-sequence length (before the application of the segmentation-rearrangement technique), the size of the seed-selection algorithm's window was selected to be near the saturation point. However, since our primary target was to reduce the number of required seeds, the selected window size L was chosen to be after that point.

In Table 1 we present the results of the proposed technique, having determined the window size L as explained above. In columns 4 to 6 the results of the seed-selection algorithm are given. The last column concerning the seed-selection algorithm (the column labeled "Test-seq. length (unreduced)") refers to the test sequences before the application of the segmentation-rearrangement technique of Section 3. Their length is equal to *Number of Seeds*·L. In columns 7 to 11 we present the results of the segmentation - rearrangement technique. In contrast with the window size parameter L, the selection of the value of parameter *Segment_Size* for this technique can be easily performed by using a very fast, brute-force procedure. This procedure tests all possible segment sizes and chooses the best one, with respect to the final test-sequence length, i.e., it chooses the *Segment_Size* that achieves the best balance between the number of allowed redundant segments in the seeds' windows and the number of redundant vectors included in the last useful segment of each window. The running time of this procedure is very low. In fact, it was lower than two seconds for each of the experiments of Table 1, in a Pentium 4, 2.6 GHz workstation. The final test-sequence length after the application of the segmentation-rearrangement procedure is shown in column 9, while the reduction achieved compared to the unreduced test sequences of column 6 is given in column 10. As can be seen, in most cases the gain is significant and the average test-sequence-length reduction reaches 20.17%. We should mention that this reduction is achieved in spite of the fact that the

Table 1. The results of the proposed technique

			Seed-selection algorithm			Segmentation-rearrangement technique				
Circuit	**Number of primary inputs**	**Test set (*T*) size (#vectors)**	**Window size (*L*)**	**Number of seeds**	**Test-seq. length (unreduced)**	***Segment_Size***	**Segment Counter length**	**Test-seq. length (reduced)**	**Test-seq. length gain (%)**	**% of "Max. reduction" gain**
c2670	233	533	1400	15	21000	108	4	15984	23.89	84.59
c7552	207	607	500	55	27500	8	6	23112	15.96	90.64
s420	34	96	4000	6	24000	1964	2	17676	26.35	76.32
s641	54	173	1220	4	4880	405	2	3645	25.31	62.88
s713	54	172	2895	3	8685	965	2	5790	33.33	73.16
s838	66	196	2100	13	27300	351	3	15093	44.71	86.73
s953	45	184	1000	6	6000	265	3	3975	33.75	75.99
s1196	32	216	2800	6	16800	948	2	13272	21.00	70.52
s1238	32	228	1900	7	13300	625	2	11250	15.41	52.73
s5378	214	926	2000	9	18000	656	2	11808	34.40	83.90
s9234	247	1190	900	27	24300	31	5	21731	10.57	84.31
s13207	700	2217	1800	5	9000	450	3	8550	5.00	83.49
s15850	611	2391	1400	9	12600	116	4	12180	3.33	60.26
s38417	1664	6322	1664	21	34944	14	7	34510	1.24	57.64
s38584	1464	8317	1464	6	8784	732	2	8052	8.33	78.21

seed-selection algorithm targets the minimization of the required seed volumes and not that of the test-sequence length. This seed-volume-minimization objective of the seed-selection algorithm explains the small reductions for s13207, s15850 and 38417. Test cubes were encoded even in the last states of the seeds' windows for these circuits, thus leaving very few redundant states that could be dropped.

For assessing the effectiveness of the segmentation-rearrangement technique, in the rightmost column of Table 1 we provide the percentage of the managed test-sequence-length reductions over those that can be achieved by the "maximum reduction" approach (Section 3). That is, if the application of the segmentation-rearrangement technique leads to a test-sequence-length reduction of X vectors, while the "maximum reduction" technique achieves a Y-vector reduction, then this percentage is equal to $(X/Y)\cdot 100$. As can be seen, the proposed test-sequence-reduction technique manages to drop most of the windows' redundant vectors (74.76% on average), while, even for the small-gain cases of s13207, s15850 and s38417, a significant percentage of the (few) redundant vectors is eliminated from the final test sequences (83.49, 60.26 and 57.64 respectively).

Table 2. Test-sequence length and ROM bits comparisons

Circuit	Test-sequence length			ROM bits		
	[6] with Mintest	[6] with Atalanta	Proposed technique	[6] with Mintest	[6] with Atalanta	Proposed technique
c2670	-	-	15984	-	-	3510
c7552	-	-	23112	-	-	11440
s420	21114	18768	17676	306	272	210
s641	35316	52974	3645	324	486	220
s713	47088	47088	5790	432	432	165
s838	272118	131670	15093	2046	990	871
s953	32760	36855	3975	360	405	276
s1196	31200	35360	13272	480	544	198
s1238	52000	33280	11250	800	512	231
s5378	1652508	-	11808	3852	-	1935
s9234	6113250	4034745	21731	12350	8151	6696
s13207	1961400	-	8550	1400	-	3505
s15850	8219783	-	12180	6721	-	5508
s38417	105249664	-	34510	31616	-	34965
s38584	94337232	-	8052	32208	-	8790

In Table 2 we compare the proposed technique against the Twisted-Ring Counters approach of [6], which is, in terms of the required test-data storage, the most successful test-per-clock, test set embedding approach in the literature. Although (only) the Atalanta ATPG tool was used for generating the test sets for our experiments, we compare the results of Table 1 against those presented in [6] for (uncompacted) test sets obtained with both the Mintest and Atalanta ATPG tools. A dash (-) in the comparison table means that no result has been provided by the authors of [6] for the corresponding benchmark circuit. Two kinds of comparison are presented in Table 2. In the first three columns we compare the two techniques with respect to the length of the resulting test sequences. In the next three columns we present the test data storage comparisons (in ROM bits). We remind that the number of bits that need to be stored for our technique is equal to *Number of seeds* · (*Number of primary inputs*+1), with the 1 corresponding to the extra bit that is stored along with each seed. We should also note that the control logic for the two compared schemes is very small and imposes similar area overhead (our approach requires on average 11.78% fewer flip-flops), and, for that reason, it is not considered in the comparisons.

As can be seen from Table 2, the proposed approach requires substantially smaller test sequences than those of [6]. Specifically, our technique is better in terms of test-sequence length in all cases, requiring on average 85.39% and 74.07% fewer test vectors than the approach of [6], for the Mintest and Atalanta cases respectively. As far as the test-data storage comparisons are concerned, the effectiveness of the seed-selection algorithm is reflected in the ROM-bits results. Even with much smaller test sequences, the proposed technique is, in the majority of cases, better than that of Twisted-Ring Counters. Only for s13207 and s38417 in the Mintest case, the proposed approach requires more ROM bits to be stored (3 more seeds are needed for s13207 and 2 for s38417). However, in these two cases the test-sequence-length savings are 99.56% and 99.97% respectively. On average, compared to the technique of [6], the proposed one requires 27.79% and 39.94% less test-data storage for the results concerning the test sets obtained with Mintest and Atalanta respectively.

5. Conclusions

A test-sequence-reduction technique that can be applied to a variety of reseeding TPG schemes has been proposed. The presented technique is based on seed-windows' segmentation and rearrangement and imposes very small hardware overhead. The combination of the proposed technique with an efficient seed-selection algorithm that has been introduced for LFSR-based, test-per-clock, test set embedding schemes, leads to results with significantly reduced test-sequence length and test-data storage requirements. Future work will be concentrated in scan-based schemes.

References

[1] C. Dufaza et al., "LFSROM: A hardware test pattern generator for deterministic ISCAS85 test sets", Proc. of ATS, 1993, pp. 160-165.

[2] A. Jas and N. A. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based designs", Proc. of ITC, 1998, pp. 458-464.

[3] W. Rao et al., "Test application time and volume compression through seed overlapping", Proc. of DAC, 2003, pp. 732-737.

[4] L. F. C. Lew Yan Voon et al., "BIST linear generator based on complemented outputs", Proc. of VTS, 1992, pp. 137-142.

[5] D. Kagaris and S. Tragoudas, "On the design of optimal counter-based schemes for test set embedding", *IEEE Trans. on CAD*, vol. 18, Feb. 1999, pp. 219-230.

[6] S. Swaminathan and K. Chakrabarty, "On using twisted-ring counters for test set embedding in BIST", *JETTA*, Kluwer Academic Publishers, vol. 17, no. 6, Dec. 2001, pp. 529-542.

[7] L. Li and K. Chakrabarty, "Test set embedding for deterministic BIST using a reconfigurable interconnection network", *IEEE Trans. on CAD*, vol. 23, Sept. 2004, pp. 1289-1305.

[8] E.Kalligeros et al., "An efficient seeds selection method for LFSR based test-per-clock BIST", Proc. of ISQED, 2002, pp. 261-266.

[9] E. J. McCluskey et al., "Test data compression (ITC'02 roundtable)", *IEEE Design Test Comp.*, vol. 20, Mar./Apr. 2003, pp. 76-87.

[10] B. Koenemann, "LFSR-coded test patterns for scan design", Proc. of ETC, 1991, pp. 237-242.

[11] E. Kalligeros et al., "Multiphase BIST: A new reseeding technique for high test-data compression", *IEEE Trans. on CAD*, vol. 23, Oct. 2004, pp. 1429-1446.