An Efficient Test Set Embedding Scheme with Reduced Test Data Storage and Test Sequence Length Requirements for Scan-based Testing*

D. Kaseridis[1], E. Kalligeros[1], X. Kavousianos[2] and D. Nikolos[1]

[1]*Computer Engineering & Informatics Dept., University of Patras, 26500, Greece*

[2]*Computer Science Dept., University of Ioannina, 45110, Greece*

kaserid@ceid.upatras.gr, kalliger@ceid.upatras.gr, kabousia@cs.uoi.gr, nikolosd@cti.gr

Abstract

In this paper we present an efficient seed-selection algorithm for reducing the test data storage requirements of scan-based, test set embedding schemes with reseeding. Moreover, a technique for reducing the length of the generated test sequences is introduced. This technique achieves significant savings with minor overhead (one extra bit per seed plus a small counter in the scheme's control logic). Experimental results demonstrate the advantages of the proposed algorithm and the test sequence reduction technique.

1. Introduction

The ever-increasing size and density of contemporary Systems-on-a-Chip (SoCs) are placing a severe burden on the traditional testing approaches based on external Automatic Test Equipment (ATE). The prevalent, core-oriented design style, although reducing the time-to-market and the complexity of the designers' task, leads to circuits with reduced accessibility and increased test data storage and test sequence length requirements. Consequently, the introduction of new, embedded testing solutions 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 pre-computed set T of test patterns that should be applied to their inputs so as to be tested. Various very successful embedded testing techniques for cores of known structure have been recently presented in the literature, some of which have been incorporated in industrial and commercial CAD-tool suites [1], [2].

Three different approaches can be followed in order to reproduce a test set T that comes with a core of

* This research was co-funded by the European Union in the framework of the program "Pythagoras II" of the "Operational Program for Education and Initial Vocational Training" of the 3rd Community Support Framework of the Hellenic Ministry of Education, funded by 25% from national sources and by 75% from the European Social Fund (ESF).

unknown structure: deterministic test set generation, test-pattern compression/decompression and test set embedding. In the first approach, an on-chip ROM or a deterministic Test Pattern Generator (TPG) [3] 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 patterns of T are stored in the tester and decompressed on-chip by means of a built-in circuit [4]-[6]. Contrary to the two aforementioned approaches, test set embedding [7]-[9] encodes the test patterns of T in a longer TPG sequence, thus allowing the exploitation of the test data volume–test sequence length trade-off. Consequently, compared to the other two, the test set embedding approach can achieve smaller hardware overhead and test-data storage results. However, this advantage is often exchanged with excessively long test sequences. Therefore, test set embedding techniques that combine both reduced hardware and test data storage requirements with short test sequences are desirable.

An LFSR-based test set embedding approach with reseeding, featuring the two aforementioned characteristics is proposed in this paper. For minimizing the number of required seeds, a seed-selection algorithm, which makes use of a heuristic criterion similar to that presented in [10] along with two new ones, is proposed. The two new criteria significantly refine the selection process and fairly reduce the selected-seed volumes. Moreover, a technique for reducing the length of the generated test sequences is introduced. The proposed technique partitions the set of vectors generated from each seed into segments and then reorders the seeds according to the number of useful segments they include. The test sequence length savings it achieves are significant, while the imposed overhead is confined to one extra bit per stored LFSR seed plus one very small counter in the scheme's control logic. We note that the proposed test set embedding approach can be implemented either as a full BIST solution or it can be combined with an external tester in a test resource partitioning scenario.

2. Seed-selection algorithm

For presenting the seed-selection algorithm, we consider the classical LFSR-based reseeding scheme, consisting basically of an LFSR, a Bit and a Vector Counter. The LFSR is loaded with a new seed and is let generate states in order to produce L test vectors. That is, each seed is expanded to a window of L vectors, which are serially shifted in the scan chains (through a phase shifter) and applied to the CUT. The corresponding responses are captured by the Test Response Compactor (TRC). The same process is repeated until all the test cubes accompanying the CUT have been covered.

The proposed seed-selection algorithm receives as inputs the user-defined parameter L, which represents the size of the window (number of test vectors) 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.

The search space of the seed-selection algorithm is shown in Figure 1. It is (initially) comprised of L symbolic vectors (sv_0, sv_1, ..., sv_{L-1}). Symbolic vector sv_i is the ith vector that would have been shifted in the scan chains of the CUT, if each bit of the initial state of the LFSR were equal to a binary variable a_i. In other words, if the maximum scan chain length is equal to n, symbolic vector sv_i is the union of the sets of linear expressions of variables a_i contained in the scan chains of the CUT after i n-tuplets of clock cycles from the initialization of the LFSR with a new seed (we do not consider the capture cycle).

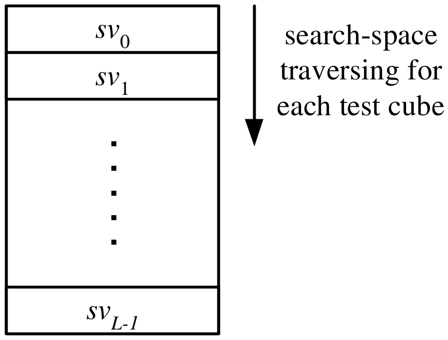

Figure 1. The search space of the seed-selection algorithm

For determining a new seed, the seed-selection algorithm makes use of the well-known concept of solving systems of linear equations [11]. At first, for each window the seed-selection algorithm generates the above described search space by simulating the function of the LFSR and the phase shifter symbolically. Then, for each test cube t of the set T, it traverses the search space row by row and, by solving the corresponding linear systems sv_i=t, tries to verify if a vector compatible to t can be generated at each position. If the corresponding system is solvable, then such a vector exists. In order to be generated, the initial LFSR state should be updated 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). We then say that test cube t has been covered at the ith window position.

The seed-selection algorithm examines, at each step, all possible linear systems for all test cubes and chooses one in order to be solved. After variable replacement, the selected test vector is removed from T and the search space is regenerated using the new initial LFSR state. We note that the replaced variables are not contained in the symbolic vectors of the regenerated search space. The above-described procedure is repeated by selecting a new test cube to be covered at some window position at each step of the algorithm, until no system is solvable for any of the remaining test cubes of T. At this point a new seed has been determined. The seed-selection algorithm continues to generate seeds this way until all the test cubes of T have been covered.

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 three basic criteria for this purpose. The first one is similar to the one proposed in [10] but since this criterion is not elaborate enough, we refine the selection process with two additional ones. These two new criteria significantly improve 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.

According to the first criterion, after traversing the search space for all test cubes, the algorithm selects the solvable system that corresponds to the "hardest" among the test cubes of T at each step. As "hardest", we consider the test cube containing the maximum number of defined bits. Such cubes are always more difficult to be encoded along with others in the same seed, since, due to the increased number of defined bits they contain, fewer variables remain free after solving a system for them. Therefore, we avoid letting such "hard" test cubes to be covered at the last stages of the encoding procedure, since it is difficult to encode, in the same seed, another "hard" test cube (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).

Since at each step there are more than one linear systems that correspond to test cubes with the same maximum number of defined bits, a second criterion is needed in order to select one of them. According to that, among the systems selected by the first criterion, we choose the one that its solution leads to the replacement of the fewest variables a_i in the L-vector window (search space). In that way, at each step there will be more variables available in the linear expressions of the symbolic vectors of the window and therefore more systems will be solvable in the following steps of the algorithm, making it possible to encode more test cubes in just one seed. Thus this second, new criterion enhances the encoding ability of each one of the selected seeds.

Finally, in the case that after the application of the two above criteria there are more than one test cubes that can be selected, a third criterion is necessary in order to choose just one of them. According to that, among the systems that correspond to test cubes with the same maximum number of defined bits and require the replacement of the same minimum number of variables in order to be solved, the one which is nearest to the first vector of the window is selected. This third selection criterion (the second new one) targets the minimization of the test sequences resulting after the application of the test sequence reduction technique that will be presented in the following section.

Various optimizations have been adopted for reducing the execution time of the seed-selection algorithm. The "always select the hardest cube" nature of the algorithm allows the test cubes of T to be grouped in cube-groups. Each cube-group contains only test cubes with the same number of defined bits. The groups are sorted in descending order starting from the one that contains the cubes with the maximum number of defined bits. In the process of selecting a new seed, the algorithm considers only the test cubes of a group and only when no cube belonging to that group can be further encoded, proceeds to the next group. This reduces significantly the number of traverses of the search space and thus, the number of linear systems solved at each step of the algorithm. Also, when a test cube is considered, the positions in the search space that a linear system can be solved for that cube are marked, so that if the same cube has to be re-examined (after the selection of another cube in the same group), only these positions are checked. These run-time optimizations combined with the fact that the procedures for solving systems of binary linear equations are much faster than those for solving conventional systems of linear equations [10], lead to run-times in the range of some minutes 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 test vectors for each selected seed. Only some of the vectors of each window are actually being used for reproducing the test cubes of set T. One can easily understand that, if the last vector of a window is not a useful one, i.e. no test cube has been selected by the algorithm in order to be covered at that position, then all vectors from the last useful one to the last window vector are redundant (Figure 2). On the other hand, the useless vectors between two successive useful ones are necessary since they connect the two useful vectors in the test vector sequence. Therefore, they cannot be removed without reseeding the LFSR (so as to bridge the gap created in its state sequence by the removed vectors). Moreover, as more seeds are selected by the algorithm, fewer cubes are encoded in new seeds' windows, leaving more useless states at the end of those windows. Due to the above-mentioned reasons we conclude that usually there will be a significant number of final-redundant test vectors 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.

The most efficient way, in terms of test-sequence length, for eliminating those redundant final vectors is to stop the expansion of each vector-window after the clock cycle, in which the last useful vector was loaded in the scan chains of the CUT. In that way the number of redundant vectors in each window (the useless vectors at the end of the window) will be equal to zero. Assuming that a Vector Counter is used for counting the vectors 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, a different approach has to be followed.

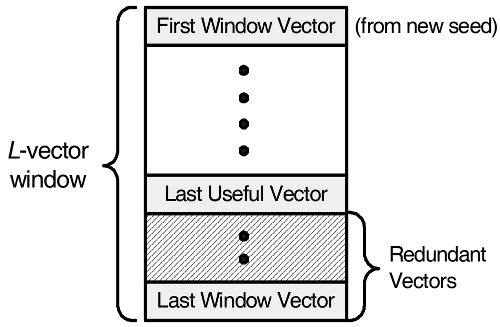

Figure 2. A window of *L* states

Such an approach would require a component (e.g. a state machine), which could generate the required number of test vectors (the initial value of the Vector Counter) for every window in the test sequence. Since the value that would be loaded in the Vector Counter at each reseeding is generated by that state machine, the corresponding initialization values do not have to be stored. A counter could be a simple implementation of that machine. Due to its functionality we call it Load Counter. As far as this counter is concerned, the only information that should be kept in a ROM is some trigger bits. Since one such bit can be used for triggering (or not) Load Counter only once (bit=1 => Load Counter proceeds to its next value, bit=0 => Load Counter maintains its previous value), the volume of stored data is proportional to the accuracy provided by the counter's values, i.e. how close are these values to the best possible ones (those required for achieving “maximum reduction”). If for example, at the end of seed's *i* window there are five useful vectors less than those at the end of the window of seed *i*-1, Load Counter should be decreased five times in order for the best result to be achieved. Unfortunately, in this case, the volume of data that should be stored may increase to such an amount that no gain would be feasible compared to the “maximum reduction” approach (for which the initial values of Vector Counter are stored).

A solution to this problem would be to assign each value of Load Counter to a group of test vectors instead of just one. In order to realize that, we segment each window into a number of equal-sized groups of test vectors (segments). The partitioning of a window into segments is shown in Figure 3. The useful vectors of the window are included in the first *k* segments, where the *k*th segment contains the last useful vector. *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 vectors) can be dropped during test generation. Furthermore, with proper selection of the segment size (parameter *Segment_Size*), the distance between the last useful vector and the end of the last useful segment can be minimized. Both the above reasons assure that this segmentation approach eliminates the majority of redundant vectors that a window includes, having as upper limit (of the eliminated redundant vectors) those dropped by the “maximum reduction” approach. Since Load Counter values now correspond to the segments of each window, Vector Counter has to be split in two separate counters, Segment Counter and Segment-Vectors Counter. The former counts the segments that should be generated for each window and is the one that is loaded with Load Counter's value, while the latter counts the test vectors of a segment (from *Segment_Size*-1 to 0). The combination of these two counters substitutes the Vector Counter of the classical reseeding approach.

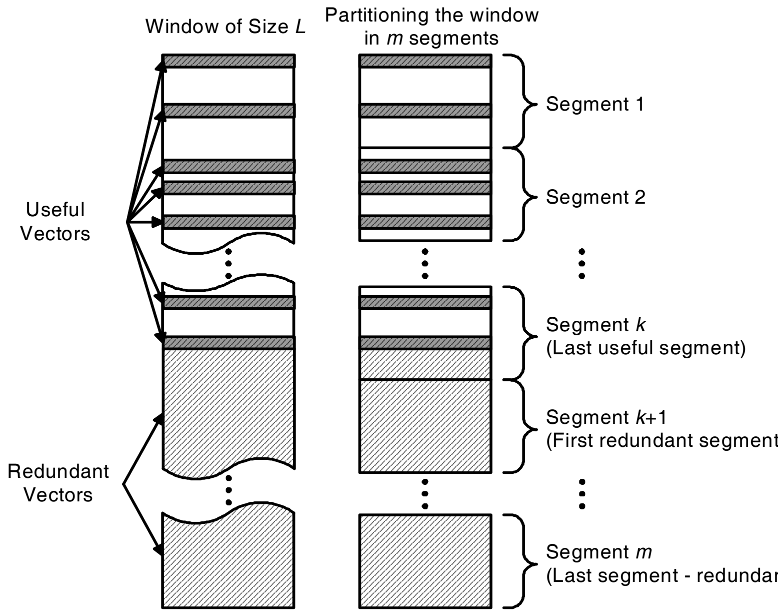

Figure 3. The proposed window segmentation technique

After partitioning each seed's window into segments, two issues remain to be resolved. The first one has to do with the frequency with which Load Counter will be triggered, or, in other words, with the number of extra bits that will be stored. There are two extreme approaches that can be followed: a) a single trigger per multiple reseedings or b) multiple triggers per one reseeding. Since the first approach lacks accuracy and the second one has increased storage requirements, we choose the solution of triggering the counter once for each reseeding. This solution combines the advantages of easy implementation, reduced test data storage requirements (one extra bit is stored per seed) and satisfactory accuracy (as earlier mentioned, the accuracy can be further improved by adjusting the size of the segments).

The second issue concerns the functionality of Load Counter. If from seed i more useful segments have to be generated compared to seeds i-1 and i+1, then Load Counter should first increase (from seed i-1 to seed i) and then decrease (from seed i to seed i+1). Seed reordering helps to eliminate this problem. We remind that the seeds are independent of each other and can be reordered in any suitable way. Therefore, we rearrange the seeds, in descending order, according to the number of useful segments they include. However, 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 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 4.

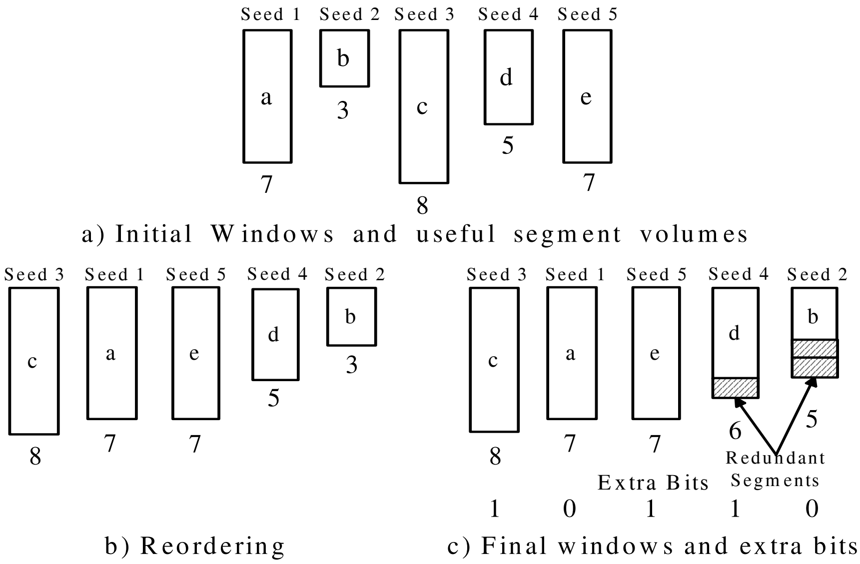

Figure 4. Rearrangement technique: a) Initial Windows, b) Windows after the reordering, c) Final windows and extra bits

At first, the seeds are arranged in descending order according to the number of useful segments their windows include (Figure 4.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 4.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 4.c windows c, a and e). Finally the procedure runs over the resulting windows and calculates the value of the extra bit of each seed (one=next seed's window requires one segment less, zero=next seed's window requires the same number of segments). Although the existence of a single extra bit per seed decreases somehow the effectiveness of the segment-partitioning scheme (due to the redundant-segments allowed in some windows), with proper selection of the segment size the number of allowed redundant segments can be reduced. This way, test-sequence length reductions that are very close to those of the "maximum reduction" approach can be achieved, as will be demonstrated in Section 4.

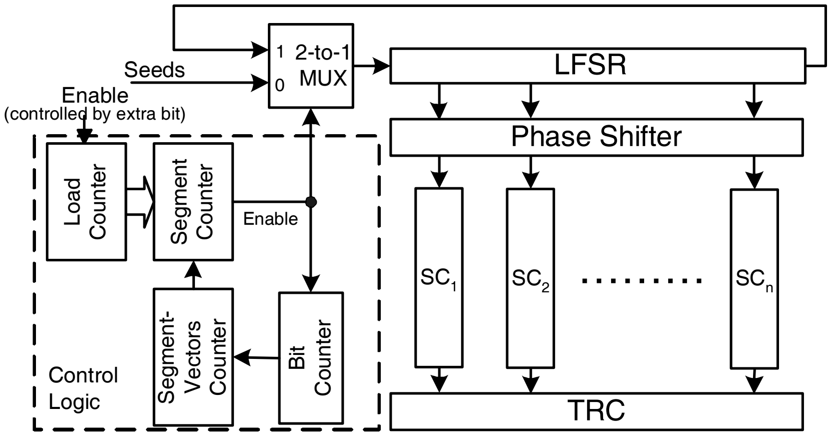

Figure 5. The proposed test-sequence reduction scheme

The architecture that handles the operation of the proposed scheme is shown in Figure 5. 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. If this bit is equal to 1, the counter is triggered and its value decreases by one (just after its previous value has been loaded in the Segment Counter), while if it is 0, Load Counter remains unchanged. 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. In order to actually control the generation of the test vectors of a window, three counters are needed: Bit Counter, Segment Counter and Segment-Vectors Counter. Bit Counter controls the scan-in operation of each vector's bits in the scan chains of the CUT. Segment-Vectors Counter controls the generation of the test vectors of a single segment, while Segment Counter is responsible for counting the required number of segments for each window and thus is initialized for each seed with the value of Load Counter. As previously mentioned, 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 (Bit 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 Bit Counter is enabled again (due to the initialization of Segment Counter to a value different from 0). The above-described process 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, their combined size cannot be much greater than the size of the Vector Counter of the classical reseeding approach. In fact it can be proven that their combined length can be 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 evaluate 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 the larger ISCAS'89 benchmark circuits with many hard-to-detect faults, assuming 32 and 64 scan chains. The required test sets were obtained by using the Atalanta ATPG tool [12]. The characteristic polynomials of the LFSRs were selected to be primitive and the Phase Shifters were calculated according to the work of [13], using 2 XOR gates for every output of the shifter.

The choice of the window size *L* of the seed-selection algorithm significantly affects both the number of final selected seeds and the length of the resulting test sequences (before the application of segmentation-rearrangement technique). By increasing the size of the window, the number of required seeds initially reduces due to the existence of more test vectors in each window. However, as the value of *L* increases, after a point we get diminishing returns (Saturation Point in Figure 6). The primary target of the selection process of the window size *L* was the reduction of the required seeds. For that reason, the window size *L* was chosen to be at the right of the saturation point (Figure 6).

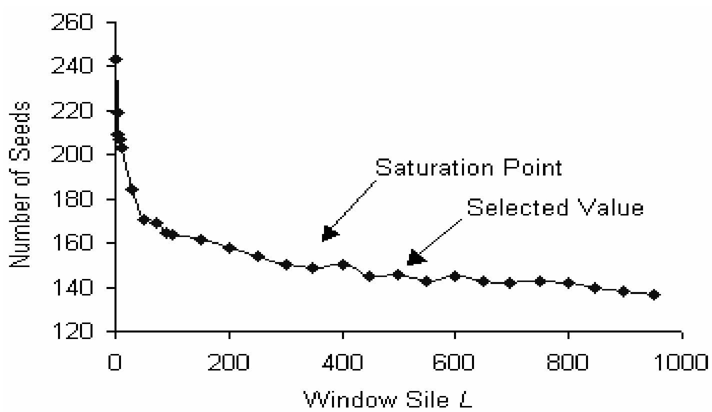

Figure 6. Number of seeds versus windows size *L* for s9234.

In Table 1 we present the results of the proposed technique for 32 and 64 scan chains, having determined the window size L as explained above. Columns 6 to 8 give the results after the application of the seed-selection algorithm with the last one (the column labelled "Test-seq. length (unreduced)") referring to the test-vector sequences for which full size windows are used for each seed (i.e., before the application of the segmentation-rearrangement technique). Moreover, in columns 9 to 13 we present the results of the segmentation-rearrangement technique. In contrast to the window size parameter L, the value of *Segment_Size* parameter can be easily determined by using a very quick, brute-force procedure. This procedure tests all possible values for the *Segment_Size* and chooses the best of them, with respect to the final test-sequence length. That is, 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 useful segments. 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 11, while the reduction achieved compared to the unreduced test sequences of column 8 is given in column 12. As can be seen, the gain is up to 42,9% while the average test-sequence-length reduction reaches 30.05% and 29.91% for 32 and 64 scan chains, respectively. For the case of the s38584, assuming either 32 or 64 scan chains, this gain is small due to the fact that the seed-selection algorithm achieves high levels of compression, by encoding many test cubes in each window, and therefore less redundant test vectors remain at the end of the windows. As a result, the segmentation-rearrangement technique achieves small reductions for this circuit.

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 (94.12% and 94.62% on average for 32 and 64 scan chains respectively). This is valid even for the small-gain case of s38584, where a significant percentage of the (few) redundant vectors is eliminated from the final test sequence (83.14% for 32 scan chains and 85.79% for the case of 64 scan chain).

In Table 2 we compare the proposed technique against the Reconfigurable Interconnection Network (RIN) approach of [9], which has been shown to be the most successful test set embedding technique in the literature, in terms of the required test-data storage. Since, according to this approach no reseedings are performed, two strategies are proposed for declustering the care bits of the test cubes: scan cell reorganization and the insertion of an extra level of multiplexers between the outputs of RIN and the inputs of scan chains of the CUT (Interleaving Multiplexers). Due to the fact that scan cell reorganization is not a preferable approach, in the comparisons we considered only the strategy of the extra interleaving level.

According to [9], the number of tristate gates in each multiplexer of the RIN is equal either to the number of configurations or to the number of LFSR cells, whichever is smaller. By doing so, the authors ensure the usage of the minimum overall number of tristate gates. Although this is true, in the case that the number of configurations is bigger than the size of the LFSR, some extra gates will be needed in every MUX of the RIN, in order to actually connect each configuration control line D_i with a tristate gate that is used in more than one configurations. That is, for every tristate gate of the RIN, which is used more than once, an extra gate (having as inputs the respective configurations' control lines) is needed in order to control this tristate gate. If we assume that each of these extra gates require at least 4 transistors, then the hardware overhead of the RIN will be equal to or even greater than the case in which a single tristate gate (4 transistors) is used for every configuration in each MUX of the RIN. Consequently, the easiest and fairest way to compare the two techniques is to assume that the number of tristate buffers in each MUX is equal to the number of configurations' control lines (D_i), as shown in Figure 2.b of [9].

Table 1. The results of the proposed technique.

					Seed-selection algorithm			Segmentation-rearrangement technique				
Scan Chains	**Circuit**	**Scan elements**	**LFSR length**	**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**
32 scan chains	s9234	247	44	1190	500	146	73000	8	6	46312	36.56	97.55
	s13207	700	24	2217	350	129	45150	5	7	34040	24.61	92.55
	s15850	611	39	2391	300	157	47100	5	6	30095	36.10	97.79
	s38417	1664	85	6322	300	548	164400	2	8	97736	40.55	99.56
	s38584	1464	56	8317	500	84	36775	25	5	36775	12.44	83.14
64 scan chains	s13207	700	24	2217	370	126	46620	11	6	34573	25.84	94.33
	s15850	611	39	2391	200	162	32400	3	7	20004	38.26	98.55
	s38417	1664	85	6322	300	555	166500	1	9	95078	42.90	99.80
	s38584	1464	56	8317	500	86	43000	18	5	37566	12.64	85.79

Taking into consideration all the above, we have calculated the hardware overhead of [9] as the overall number of transistors that this approach requires. In other words, the overhead presented in Table 2 is equal to the sum of the transistors required for the implementation of the tristate-buffer-based MUXes of the RIN and the interleaving level, as well as the ROM for storing some necessary control bits. We have assumed that for the implementation of each tristate buffer 4 transistors are needed as mentioned in [9], while the area that each ROM bit occupies was considered to be equal to that of 1 transistor [14]. Moreover, for every configuration control line (D_i) we have calculated the overhead of an inverter required for generating the signal $\overline{D_i}$, which, combined with the D_i, controls the inputs *enable* and *enable* of the assumed tristate buffers, respectively. The hardware overhead of the proposed approach was calculated as the sum of the transistors required for the implementation of the phase shifter plus the transistors that correspond to the ROM bits that should be stored (as above, 1 transistor per bit was assumed). The hardware overhead of the phase shifters is equal to 16·*Number of scan chains*, since 2 standard transmission-gate-based XORs were used for the realization of each of their XOR trees. Such a XOR gate requires 8 transistors for its implementation. We note that in our ROM-bit results we have also calculated the extra bit that is stored along with each seed. We should stress that, for the calculation of the hardware overhead of the approach of [9] we did not take into account neither the excessive wiring of the RIN and the interleaving level, nor the decoder needed for driving the RIN. As for the rest of the control logic of the two compared schemes, it is very small and imposes similar area overhead and, for that reason, it has not been considered in the comparisons.

Two kinds of comparison are presented in Table 2. In columns 3 to 5 we compare the two techniques with respect to the imposed hardware overhead, while in the next 3 columns we compare them as far as the length of the resulting test sequences is concerned. As can be seen from this table, the proposed approach requires substantially smaller test sequences than those of [9]. Specifically, our technique is better in terms of test-sequence length in all cases, requiring on average 80.96% and 78.62% fewer test vectors, for the 32 and 64 scan chains respectively. As for the hardware overhead comparisons, the RIN approach requires significantly fewer ROM bits but this is done at the expense of the insertion of two levels of MUXes between the LFSR and the scan chains of the CUT. Consequently, these MUXes, the size of which is proportional to the number of the scan chains, require for their implementation significantly more transistors compared to those needed by both the phase shifter and the stored data bits of our approach. Therefore, as far as the hardware overhead comparisons are concerned, even with much smaller test sequences, the proposed technique is, in all but one case, more efficient than that of RIN approach. Only for s38417 in the case of 32 scan chains, the proposed approach imposes more hardware overhead. However, in this case the test sequence length saving is equal to 84.37%. On average, compared to the technique of [9] the proposed one requires 13.49% and 53.67% less hardware overhead for 32 and 64 scan chains respectively.

Table 2. Hardware overhead and test-sequence length comparisons.

		Hardware overhead			**Test sequence length**		
Scan Chains	**Circuit**	**[9] (#trans.)**	**Proposed (#trans.)**	**Reduct. (%)**	**[9] (#vec.)**	**Proposed (#vec.)**	**Reduct. (%)**
32 scan chains	s9234	9424	7082	24.85	135765	46312	65.89
	s13207	5428	3737	31.15	152596	34040	77.69
	s15850	7352	6792	7.62	222336	30095	86.46
	s38417	44896	47640	-6.11	625273	97736	84.37
	s38584	5884	5300	9.93	383009	36775	90.40
64 scan	s13207	19409	4174	78.49	75047	34573	53.93

chains	s15850	19420	7504	61.36	179580	20004	88.86
	s38417	52524	48754	7.18	616835	95078	84.59
	s38584	18323	5926	67.66	291425	37566	87.11

5. Conclusion

An efficient LFSR-based test set embedding approach with reseeding has been proposed in this paper. It features an effective seed-selection algorithm that minimizes the test data storage requirements, as well as a technique for reducing the resulting test sequences. The latter achieves significant test sequence length savings (30% on average), while the overhead imposed is confined to one extra bit per stored LFSR seed plus one very small counter in the scheme's control logic. The proposed approach compares favorably against the most recent and efficient test set embedding technique in the literature.

References

[1] B. Koenemann et al., "A SmartBIST variant with guaranteed encoding", Proc. of ATS, 2001, pp. 325-330.

[2] J. Rajski et al., “Embedded deterministic test”, IEEE TCAD, vol. 23, May 2004, pp. 776-792.

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

[4] A. Jas et al., "Scan Vector Compression/Decompression Using Statistical Coding", Proc. of VTS, 1999, pp. 114-120.

[5] K. Chakrabarty, V. Iyengar, A. Chandra, ”Test resource partitioning for System-on-a-Chip”, Kluwer, 2002.

[6] I. Bayraktaroglu and A. Orailoglu, “Concurrent application of compaction and compression for test time and data volume reduction in scan designs”, IEEE TC, vol. 52, Nov. 2003, pp.1480-1489.

[7] D. Kagaris & S. Tragoudas, "On the design of optimal counter-based schemes for test set embedding", IEEE TCAD, Feb. 1999, pp. 219-230.

[8] S. Swaminathan & K. Chakrabarty, "On using twisted-ring counters for test set embedding in BIST", JETTA, vol. 17, Dec. 2001, pp. 529-542.

[9] L. Li & K. Chakrabarty, "Test set embedding for deterministic BIST using a reconfigurable interconnection network", IEEE TCAD, Sept. 2004, pp. 1289-1305.

[10] E. Kalligeros, X. Kavousianos & D. Nikolos, "Multiphase BIST: A new reseeding technique for high test-data compression", IEEE TCAD, Oct. 2004, pp. 1429-1446.

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

[12] H. K. Lee and D. S. Ha, “Atalanta: an Efficient ATPG for Combinational Circuits,” TR, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, 1993.

[13] J. Rajski et al., “Automated synthesis of phase shifters for Built-In Self-Test applications”, IEEE TCAD, vol. 19, Oct. 2000, pp. 1175-1188.
[14] L. Huang et al., "Gauss-Elimination-Based Generation of Multiple Seed-Polynomial Pairs for LFSR", IEEE TCAD, vol. 16, Sept 1997, pp. 1015-1024.