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 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 (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

[†] We thank the European Social Fund (ESF), Operational Program for Educational and Vocational Training II (EPEAEK II), and particularly the Program PYTHAGORAS, for funding the above work.

with a new seed (we do not consider the capture cycle).

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 vector by vector 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. 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 that are presented in Table 1. 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.

Table 1. Seed-selection algorithm's criteria

Criterion	Description
1st	Select the solvable systems that correspond to the test cubes containing the maximum number of defined bits
2nd	If there are more than one solvable systems selected be the 1st criterion, choose the one that its solution leads to the replacement of the fewest variables α_i in the L-vector window
3rd	If there are more than one solvable systems selected be the 2nd criterion, select the one that is nearest to the first vector of the window

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 1). 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. 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.

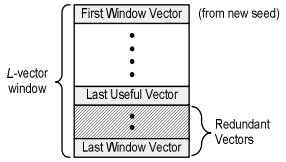

Figure 1. A window of L states

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 is loaded in the scan chains of the CUT. In that way the number of redundant vectors in each window will be equal to zero. Assuming that a Vector Counter is used for controlling the generation of 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.

Such an approach would require a component (e.g. a state machine), which would be used to generate the required number of test vectors for every window in the test sequence. A counter, named Load Counter, could be a simple implementation of that machine. 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, 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 of the "maximum reduction" approach). Unfortunately, there will be cases that the volume of data that should be stored in order to achieve a certain level of accuracy, may increase to such an amount that no gain would be feasible compared to the "maximum reduction" approach.

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 2. The useful vectors of the window are included in the first k segments, where the kth 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 the one that the "maximum reduction" approach drops.

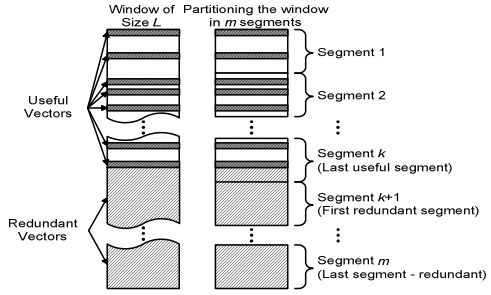

Figure 2. The proposed window segmentation technique

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. In order to reduce the test-data storage requirements and achieve at the same time a satisfactory level of accuracy using a simple implementation, we choose to trigger the counter only once for each reseeding. Therefore, only one extra bit is needed to be stored along with every selected seed.

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 was chosen in order to eliminate this problem since 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.

Taking everything into consideration, the steps of the proposed seed-rearrangement procedure are: At first, the seeds are arranged in descending order according to the number of useful segments their windows include. 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. 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).

The architecture that handles the operation of the proposed segmentation-rearrangement scheme is shown in Figure 3. 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. 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.

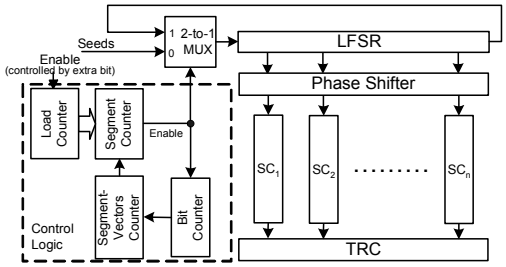

Figure 3. The proposed test-sequence reduction scheme

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 shifter was calculated according to the work of [13], using 2 XOR gates for every output of it.

In Table 2 we present the results of the proposed technique for 32 and 64 scan chains. 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. 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. Furthermore, for assessing the effectiveness of the segmentation-rearrangement technique, in the rightmost column of Table 2 we provide the percentage of the managed test-sequence-length reductions over those that can be achieved by the "maximum reduction" approach (Section 3).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).

In Table 3 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

Table 2. 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

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.

Two kinds of comparison are presented in Table 3. In columns 3 to 5 we compare the two techniques with respect to the length of the resulting test sequences, while in the next 3 columns we compare them as far as the imposed hardware overhead is concerned. The hardware overhead was calculated according to the number of transistors that each approach requires in order to implement the control logic and to store the required test-data. Therefore, for the case of [9] the hardware overhead 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 required ROM for storing its necessary control bits. On the other hand, the hardware overhead of our 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.

Table 3. Test-sequence length and hardware overhead comparisons

		Test sequence length			Hardware overhead		
Scan Chains	**Circuit**	**[9] (#vec.)**	**Proposed (#vec.)**	**Reduct. (%)**	**[9] (#trans.)**	**Proposed (#trans.)**	**Reduct. (%)**
32 scan chains	s9234	135765	46312	65.89	9424	7082	24.85
	s13207	152596	34040	77.69	5428	3737	31.15
	s15850	222336	30095	86.46	7352	6792	7.62
	s38417	625273	97736	84.37	44896	47640	-6.11
	s38584	383009	36775	90.40	5884	5300	9.93
64 scan chains	s13207	75047	34573	53.93	19409	4174	78.49
	s15850	179580	20004	88.86	19420	7504	61.36
	s38417	616835	95078	84.59	52524	48754	7.18
	s38584	291425	37566	87.11	18323	5926	67.66

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. 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.

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, no. 6, 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 et al. , "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.