DV-TSE: Difference Vector Based Test Set Embedding[†]

Maciej Bellos[1,2], Xrysovalantis Kavousianos[1,3], Dimitris Nikolos[1,2] and Dimitri Kagaris[4]
[1]Dept. Of Computer Engineering & Informatics, University of Patras, 26500 Patras, Greece
[2]Computer Technology Institute, 3 Kolokotroni St., 26221 Patras, Greece
[3]Dept. Of Computer Science, University of Ioannina, 45110 Ioannina, Greece
[4]Dept. of Electrical & Computer Engineering, Southern Illinois University, Carbondale, IL 62901, USA
{bellos, kabousia}@ceid.upatras.gr, nikolosd@cti.gr, kagaris@engr.siu.edu

Abstract

In this paper we present a new test set embedding method for test-per-clock BIST schemes. The method works efficiently with fully specified as well as partially specified test sets and requires a number of clock cycles equal to the size of the test set. The resulting test pattern generation mechanism (TPG) compares favourably in terms of area implementation and test application time to already known approaches.

1. Introduction

Built-In Self-Test (BIST) is an attractive approach of testing large VLSI circuits, since it provides the ability for a circuit to test itself reducing the need for complex external testing equipment. A BIST scheme is comprised from three basic blocks: the test pattern generator (TPG) which generates the test patterns and applies them to the circuit under test (CUT), the Test Response Compactor which produces a signature after compacting the CUT's responses and the BIST controller responsible for producing the control signals which will ensure the correct cooperation of the other two blocks and the CUT.

Several BIST schemes have been proposed so far which can be classified into several categories. Based on the number of cycles required to apply a test pattern to the CUT, BIST schemes can be classified as test-per-clock and test-per-scan. In the test-per-clock scheme a new test pattern is applied to the CUT at each clock cycle, while in the test-per-scan scheme the scan path is serially filled by the TPG. Based on the type of test patterns they generate, BIST schemes can be classified into three main categories: pseudorandom [1], pseudoexhaustive [1] and deterministic. A type of deterministic BIST is test set embedding. In this case the structure of the circuit under test is unknown. This fact makes the design of the TPG even more difficult. However, test set embedding is very attractive for intellectual property cores where no information is given about the internal structure of the cores.

The problem of the Test Set Embedding was studied in [2-8]. In Table 1 we present the main features of each method.

Table 1. Test set embedding method features.

Method	Test sets targeting hard to detect faults	Test sets targeting all stuck-at faults: Fully specified	Test sets targeting all stuck-at faults: Partially specified	Relation of test application time w.r.t. test set size
ROM based	√	√	√	Equal
[2]	√	√		Equal
[3]	√	√		At least two times more
[4]	√	√		Equal
[5, 6]	√		√	Very long test application time
[7]	√	√	√	Very long test application time
[8]	√	√	√	Equal

In this work we propose a new test set embedding method suitable for test-per-clock BIST schemes, which considers fully specified as well as partially specified test sets and reproduces the given test set in as many clock cycles as its size. The approach modifies appropriately the original test set and then uses OR gates to obtain the desired vectors. We will show that the proposed method compares favourably with previous approaches. The rest of the paper is organized as follows. In Section 2 we present the main idea behind the proposed method together with the required test set modifications. Section 3 describes the test set generation mechanism, which is based on the modified test set. In Section 4 we present experimental results considering two kinds of test sets that target all the faults of the circuit under test. The first kind considers fully specified test sets, while the second considers partially specified test sets, i.e. test sets that contain undefined bits. Finally, Section 5 concludes.

2. Test Set Modification

2.1. Basic Idea

Our method takes advantage of the fact that test vectors tend to be correlated. Faults in the CUT that are

[†] This research was financially supported by the Public Benefit Foundation "Alexander S. Onassis" via its scholarships programs, by the Research Committee of Patras University, within the framework of "K. Karatheodoris" scholarships program and by the State Sdcholarship's Foundation of Greece via its Post-doctoral research scholarships program.

structurally related require similar input value assignments in order to be activated and propagated to the CUT's outputs [9]. Therefore, many pairs of test vectors in the test set will differ in a small number of bit positions. Ordering the test vectors in the test set, such that correlated test vectors follow each other, results in a test matrix where neighbouring vectors differ in a small number of bits.

In order to achieve the above we can use the notion of the Hamming distance of two vectors, which is the number of bit positions in which the two vectors differ. Specifically we can order the test vectors in such a way that the Hamming distances between successive vectors are reduced. Let the precomputed test set be $T=\{t_1, t_2, t_3, \dots, t_n\}$. Having T as a basis we can derive the difference matrix T_{diff}, where each vector, except for the first, is the bit-wise exclusive OR (XOR) of consecutive vector pairs of T in the following manner: $t_1, t_1 \oplus t_2, t_2 \oplus t_3, \dots, t_{n-1} \oplus t_n$. The number of '1's in the difference matrix is expected to be much smaller than that in the initial test depending on the amount of correlation between the test vectors. The difference matrix was at first used in [9] as an intermediate matrix in a compressing technique proposed for reducing the amount of test data stored on a tester and transferred to the CUT during testing. In our method the difference matrix is used as an intermediate matrix serving a completely different purpose. Its task is to reduce the hardware required for the implementation of the test pattern generator.

Test data can be fully specified, that is, all the bits of the test vectors have one of the two possible values (0 or 1), or can be partially specified, that is, they can contain don't care or undefined bits (x). The latter can be assigned values as the test engineer deems appropriate. The proposed method is applicable to both kinds of test data. The original test set T_{orig} undergoes a number of preprocessing steps to create a modified version T_{mod}. T_{mod} is then used to construct the pattern generation mechanism.

2.2. Test set preprocessing steps

The modification of the given test, T_{orig}, is carried out through a number of preprocessing steps. Consider the general case of a test set that contains undefined bits. The existence of undefined bits provides flexibility in value assignment so as to minimize the TPG hardware. The goal of the preprocessing steps is to create columns where one of the two binary values has much more occurrences than the other. This test set will eventually lead to a rather simple pattern generation mechanism. The preprocessing steps are shown below:

Step 1. Determine if there are any possible constant columns and remove them.

Step 2. Reorder the test vectors so as to reduce the sum of Hamming distances among test pairs.

Step 3. Identify possible identical / complementary columns and assign the proper values to the undefined bits.

Step 4. If there are undefined bits left, assign the undefined bits of each column to the value that has the most occurrences in the column.

Step 5. Remove identical vectors. The result is the fully specified test set T_{full}.

Step 6. Create the test set that contains the difference vectors, named T_{diff}.

Step 7. Combine T_{full} and T_{diff} by choosing appropriate columns from each test set so as to create T_{mod}.

2.3. Preprocessing step description

In this section we will provide a thorough explanation of the above preprocessing steps.

Step 1 removes, if there are any, some of the undefined bits of the T_{orig}. The first case examined is the existence of columns with '1's and 'X's, or '0's and X's. Upon identification of such a column, the column is labelled as constant and not considered in the creation of T_{mod} since it can be easily generated. Obviously, test sets that detect all the faults of the circuit do not contain such columns.

The next step (Step 2) is to reorder the test vectors so as to reduce the sum of Hamming distances of consecutive test vector pairs. In this way, by considering the difference vectors, we can eventually reduce the numbers of '1's in the resulting set. The reduction of the sum of Hamming distances among test vector pairs is equivalent to the Traveling Salesman Problem, which is known to be NP-complete, so a heuristic is used. At first, a weighted graph is constructed from the given test set. In this graph each node represents a test vector and each edge connecting two nodes represents the Hamming distance of the corresponding test vector pair. When undefined bits are present in any vector of the test pair, the Hamming distance is calculated in a slightly different way. The undefined bit is assumed to have the same value as the corresponding bit of the other test vector and thus the Hamming distance of the pair does not increase.

Having constructed the graph, the selection of a test vector pair is based on a greedy approach, according to which the test vector pair with the smallest Hamming distance is chosen at each step. After each selection, the undefined bits that may exist must be assigned to values that will not contradict the Hamming distance value that was used in the selection. Therefore, each time the combination X, 1 (X, 0) appears in the corresponding bit positions it is converted to 1, 1 (0, 0). Figure 1 shows the effect of the above conversions on two vectors. With these assignments in mind, the graph is updated with the new values of the Hamming distances between the remaining test vectors and then the pair with the smallest Hamming distance is chosen again. The procedure goes on until there are no nodes left in the graph.

$$\begin{array}{c} 1\,x\,0\,x\,0\,1\,1\,1 \\ x\,0\,0\,1\,0\,1\,x\,1 \end{array} \xrightarrow[(x,\ 1 = 1,\ 1)]{(x,\ 0 = 0,\ 0)} \begin{array}{c} 1\,0\,0\,1\,0\,1\,1\,1 \\ 1\,0\,0\,1\,0\,1\,1\,1 \end{array}$$

Figure 1. Undefined bits assignments.

Apart from the greedy approach other approaches [10, 11] can be used as well. Both [10] and [11] examine the case where the vectors can be inverted, which provides more flexibility in achieving less transitions.

However, this would require additional logic that would control the inversions of the vectors. In [10], a Double Spanning tree, a Minimum Spanning tree Maximum Matching and a greedy approach where implemented and the experimental results showed that the latter was the most efficient. In [11] a genetic algorithm was used on partially specified test sets and was found to be more efficient than the greedy approach.

Step 2 may not necessarily remove all the undefined bits from the test vectors, so Step 3 is performed. If we consider the test set column-wise, we can assign values to the undefined bits so as to maximize the existence of identical or complementary columns. If there are such columns then the undefined bit in a position will obtain the value of the corresponding position in the identical / complementary column. This step can also be applied if we partition the reordered test set in phases of equal length, thus increasing the probability of finding identical / complementary columns.

If after all the above steps there are still some undefined bits in Step 4, we assign them in a manner that will maximize or minimize the presence of one of the two values (0 or 1). More specifically, if the number of ‘0’s (‘1’s) is larger than the number of ‘1’s (‘0’s) then the remaining undefined bits are assigned the value 0 (1). The idea behind this approach is to bias the number of occurrences of one of the two values so as to achieve as less transitions (1->0 or 0->1) as possible when the difference vectors will be created. This also complies with the fact that the undefined bits were considered to have the same value when the reordering step was applied.

Step 5 checks if the result of all of the assignments has yielded some identical vectors. In this case those vectors are removed because their generation serves no purpose and will lead to higher area implementation.

It is obvious that when a fully specified test set is provided we do not have to perform the steps that assign values to undefined bits, that is, Steps 1 and 4. Step 5 is also not performed since in a fully specified test set there are no identical vectors. The only steps performed are the reordering of the test vectors according to the Hamming distances of the vector pairs and the identification of the identical or complementary columns with minor differences. One such difference is that once the graph is constructed it will never be updated with new values, since the Hamming distances do not change.

With the above preprocessing steps we have created a fully specified version, T_{full}, of the initial test set. Based on this test set, Step 6 creates the difference test vectors in T_{diff} according to the following manner: v_1, $v_1 \oplus v_2$, $v_2 \oplus v_3$, … , $v_{n-1} \oplus v_n$, where v_i belongs to T_{full}.

The creation of T_{mod}, performed in Step 7, is achieved by using T_{full} and T_{diff}. We examine both sets column-wise. For each column we count the number of ‘0’s and ‘1’s and we assign a weight equal to the smaller of the two values. This is done for both T_{full} and the T_{diff}. We then choose the column from T_{full} only if it has a weight smaller than the weight of the corresponding column from T_{diff} plus a threshold, that is:

$$W_{i,\,full} < W_{i,\,diff} + \text{Threshold},\ 0 \le i \le n$$

In the above n is the number of the primary inputs of the circuit and $W_{i,\,full}$, $W_{i,\,diff}$ are the weights of column i of the fully specified and the difference test sets, respectively. The threshold value takes into account how the test set will be reproduced from the TPG mechanism described in the next section and is estimated experimentally.

3. Test Set Generation Mechanism

Having produced the modified test set T_{mod} we can now construct the test pattern generation mechanism or test pattern generator (TPG). The TPG uses the T_{mod} test set, which contains columns either from the T_{full} or the T_{diff} test sets, and reproduces the T_{full} test set in as many clock cycles as the number of its test vectors.

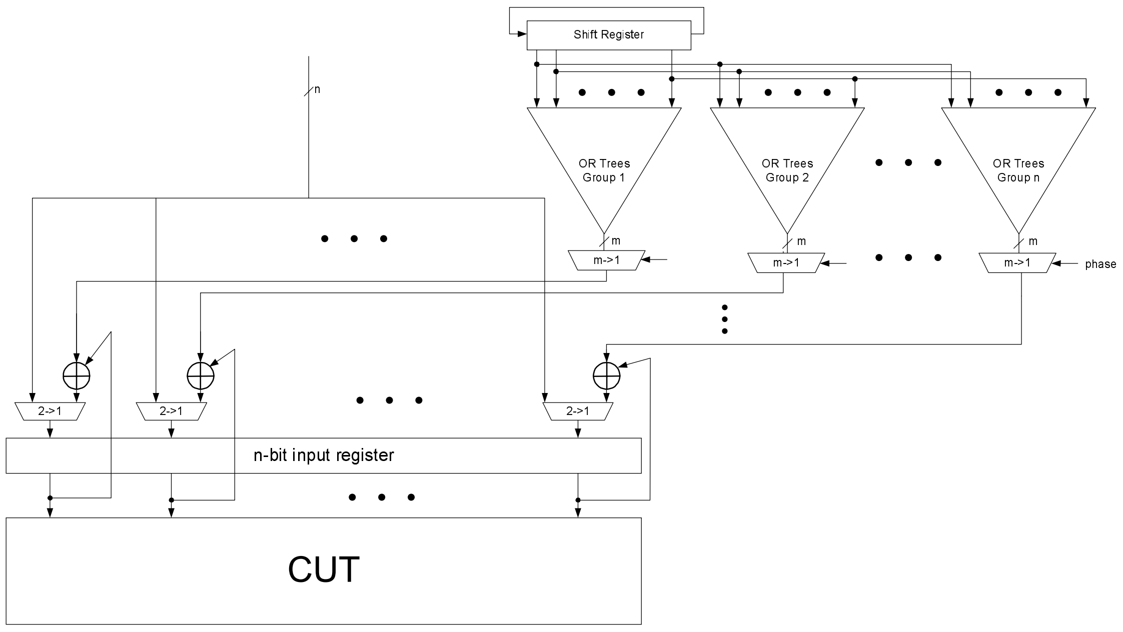

Figure 2. Proposed test pattern generation mechanism.

Figure 2 depicts the blocks the proposed TPG consists of. There is a cyclic shift register that feeds *n* groups of trees constructed from OR gates, which have the task of producing the columns of the T_{mod} test set. Each group of OR trees contains *m* OR trees, where *m* is the number of phases the test set was split into. The outputs of the OR trees of each group are driven to m->1 multiplexers, which have the task of selecting the proper OR tree. The control signals of the multiplexers are driven by a binary counter that counts up to *m*. Obviously if there are no phases, the m->1 multiplexers and the binary counter do not exist. Finally the outputs of the multiplexers are driven to XOR gates, whose other inputs are fed from the corresponding flip-flop of the input register of the CUT. The number of the XOR gates required is determined during the production of T_{mod} and is equal to the number of columns taken from the T_{diff} test set.

Before we explain how the TPG produces the test vectors, we will show how the OR trees are constructed. The key to their construction is the initial value of the cyclic shift register. In our case the register contains the pattern 100…00 and at each clock cycle it shifts the '1' to the right by one position. This ensures that at each clock cycle there can be at most one '1' in the inputs of the OR tree. Consider as an example that we have to produce the sequence (test matrix column) '10011'. In order to do so, we must use a cyclic shift register with 5 flip-flops initialized to '10000' (Figure 3). The OR tree used will be driven by some of the 5 flip-flops, more specifically from D_0, D_3 and D_4. It is obvious that at clock cycles 0, 3 and 4 the output of the OR tree will be equal to '1' while in the rest of the cycles the output is '0' since all the its inputs are driven by '0'.

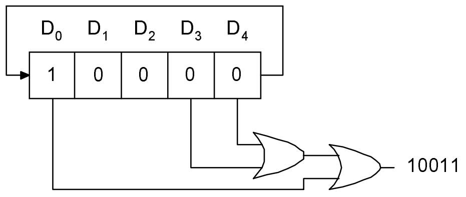

Figure 3. OR tree construction example.

From the example it is obvious that the construction of the OR trees is straightforward and fast. Using the columns of T_{mod} we construct the appropriate trees and the m->1 multiplexers forward the correct tree output to the XOR gates or the flip-flops of the input register.

Having explained how the OR trees are constructed, we can now show how the TPG works. At first the flip-flops, which during test mode will receive the value of the XOR gates, are initialized to '0', therefore not altering the first value arriving to the XOR gates. As mentioned earlier, the columns fed to the XOR gates are the ones taken from T_{diff} and their first bit is the corresponding bit of the first vector of T_{full}. At the same time, the cyclic shift register and the binary counter are initialized to 100…00 and 0…0, respectively. Then at each clock cycle the counter increases by one, applying a new test vector to the CUT. When the counter reaches the value *m* it produces a clock signal for the cyclic shift register, so as to feed the OR trees with new values. When the '1' in the shift register reaches the rightmost position and the counter reaches value *m*, the process terminates. The production of the vectors is done in an interleaved manner identical to the one used in [12], which reduces the transitions in the OR trees and therefore the overall power consumption of the TPG.

4. Experimental Results and Comparisons

In the general case the core provider supplies the core together with its test set. The test set can be fully specified, i.e. compacted, or partially specified, that is it contains undefined bits. For that reason fully specified as well as partially specified test are used for comparing the efficiency of the proposed method against the already known methods. The test sets used in our experiments concerned the ISCAS '85 benchmark circuits.

Among the methods that have been proposed for test set embedding [2-8], only the methods given in [2, 4, 8] can be used for test sets targeting all the single stuck-at faults of the circuit and ensure that a test set with cardinality *n* is generated in *k* clock cycles, where *k* is equal or close to *n*. The methods given in [5, 6] are based on the manipulation of the large number of undefined bits appearing in the test sets for the hard to detect faults and usually lead to long test application times. They cannot be used in the case of fully specified test sets that target all the faults of the CUT. In the case of the partially specified test sets the above methods lead to very long test application times.

Table 2. Area implementation for fully specified test sets.

			Phase Shifters [8]		LFSRom [4]		Proposed		Savings %	
Circuit	**Inputs**	**Vectors**	**Phases**	**Area**	**Phases**	**Area**	**Phases**	**Area**	**Proposed vs. [8]**	**Proposed vs. [4]**
c432	36	48	2	381.8	2	342	2	329.6	13.67	3.63
c499	41	52	2	455	2	392.4	2	390.2	14.24	0.56
c880	60	49	2	573.2	1	463.4	1	458.2	20.06	1.12
c1355	41	85	4	685.6	3	596	3	578.8	15.58	2.89
c1908	33	111	4	708.8	3	621.2	3	571.6	19.36	7.98
c2670	233	100	4	3506.4	3	2647.6	3	2700.6	22.98	-2.00
c3540	50	143	8	1406.4	4	1085.2	4	1034.4	26.45	4.68
c5315	178	112	4	3284.4	3	2434.8	3	2462.4	25.03	-1.13
c6288	32	27	3	209.6	1	172.6	1	171.4	18.23	0.70
c7552	207	178	8	5670	4	4128.4	4	4170.8	26.44	-1.03

Table 3. Area implementation for partially specified test sets.

	Phase Shifters [8]			Enhanced-LFSRom				Proposed			Savings %		
						Area							
Circuit	Vectors	Phases	Area	Vectors	Phases	Random	Bias	Vectors	Phases	Area	Proposed vs. [8]	Proposed vs. Random	Proposed vs. Bias
c432	95	4	418	95	3	546.4	284.2	79	3	278	33.49	49.12	2.18
c499	55	2	382.4	55	2	202.4	191	55	2	187.2	51.05	7.51	1.99
c880	62	2	632.4	62	2	603.8	498	59	2	474.2	25.02	21.46	4.78
c1355	137	4	1004.6	137	4	601.6	576	137	4	541.8	46.07	9.94	5.94
c1908	123	4	696	123	4	596	591.4	122	3	533.6	23.33	10.47	9.77
c2670	124	4	2639	124	3	3230.4	2022.2	124	3	2068.6	21.61	35.96	-2.29
c3540	157	8	1301.8	157	4	1184.6	1105.8	154	4	1004.2	22.86	15.23	9.19
c5315	120	4	3022.8	120	3	2546.8	2470	120	3	2501.8	17.24	1.77	-1.29
c6288	51	2	343	51	2	216.2	183.2	50	2	170.4	50.32	21.18	6.99
c7552	231	8	5592.6	231	4	5234	4313.8	230	4	3959.8	29.20	24.34	8.21

Also the test application time required by the method of [7] is more than 1000 times larger than that required by the ROM based test set embedding [8]. Therefore we do not compare our method with the ones given in [5-7]. In [8] it has been shown that the phase shifter based method is more efficient, with respect to the hardware required for its implementation, than the ROM based test set embedding as well as the method given in [2]. Therefore we will compare the proposed method against the ones presented in [4] and [8]. The proposed method uses the greedy heuristic so as to reorder the test vectors. The use of the genetic algorithm of [11] would possibly lead to slightly better experimental results.

We first determined the value of the threshold that is used in the selection of the columns. After a number of experiments, a value of 5 (bits) was adopted for the threshold and is a little larger than a gate equivalent. Then we determined the number of phases. In the phase shifter based method [8], every column of a given test matrix should be found as a subsequence of the m-sequence of some LFSR of appropriate length and characteristic polynomial. In order to be sure to find a subsequence of length l we have to use a characteristic polynomial of degree at least equal to l. The above implies that the LFSR size must be greater than or equal to $l = \left\lceil \frac{\text{Test Set Cardinality}}{\text{Count of Phases}} \right\rceil$, where $\lceil x \rceil$ denotes the smallest integer greater than or equal to x. In the phase shifter based method we cannot reduce the number of phases because then the LFSR size increases and the search time becomes unacceptably long. On the other hand, in the proposed method, as well as the method given in [4], the design time is negligible, thus we can examine a number of choices for the phases the test set can be split into in order to find the value leading to the least hardware overhead. In Figure 4 we give the hardware required for the implementation of the proposed method with respect to the number of phases used for circuits c499 and c1355. As we can see for each circuit there is a number of phases giving the least hardware overhead. In our experiments we have used the number of phases that results in the least hardware overhead.

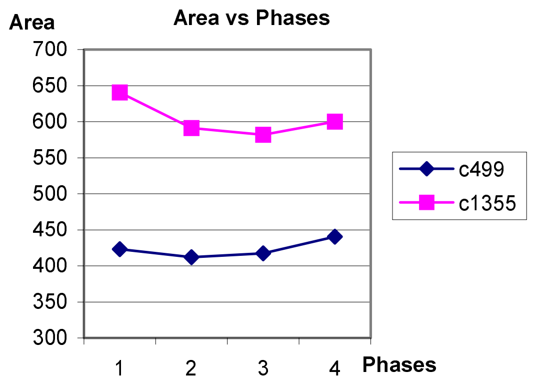

Figure 4. Impact of the number of phases on implementation area.

The experimental results concerning fully specified test sets are shown in Table 2. The fully specified test sets are the same with the test sets used in [8]. We have also used the same synthesis tools and the same implementation library as in [8]. As it can be seen the proposed method is significantly better than that of [8]. The difference becomes larger for longer test sets and circuits with a lot of inputs. Furthermore it is better than the method of [4] in all but 3 cases (c2670, c5315 and c7552).

LFSRom was proposed only for fully specified test sets. We have enhanced LFSRom so as it can be applied to partially specified test sets. In the case of partially specified test sets, we have to remove the undefined bits from the test set. At first we identify the cases were there could be identical / complementary columns and we assign the proper values to the appropriate undefined bits. The remaining bits are assigned using two different policies. The first called "Random" assigns the undefined bits randomly with probability 0.5 to one of the two values. The second called "Bias" assigns the undefined bits of each column with the value that would minimize the occurrences of the '1's ('0's) if their population is smaller than the population of '0's ('1's). The "Bias" assignment policy is actually Step 4 of the preprocessing steps of the proposed method (see Section 2.2.).

Table 3 contains the experimental results for the phase shifter based [8], the two assignment policies for the enhanced-LFSRom and the proposed method. For each method we also provide the number of phases the test set was split into. It is obvious that the proposed method is far more superior than [8] and the enhanced-LFSRom when the "Random" assignment policy is used. The proposed method is still superior in all but two cases (c2670 and c5315) when the "Bias" policy is used.

From Table 2 and Table 3, we can observe that in the case of partially specified test vectors, the benefits of the existence of undefined bits overcome the drawback of the larger test vector population leading to less area overhead in almost all cases, when the proposed method is considered. In contrast, the other two methods had an increase in area implementation in nearly half the cases. More specifically in Table 4 we show the effect of the partially specified test sets against the fully specified test sets when the proposed method is considered.

Table 4. Impact of partially specified vs. fully specified test sets on the proposed method.

		Fully Specified Test Sets		Partially Specified Test Sets		Partially Specified vs. Fully Specified Test Sets
Circuit	**Inputs**	**Vectors**	**Area**	**Vectors**	**Area**	**Area Savings**
c432	36	48	329,6	79	278	18,56%
c499	41	52	390,2	55	187,2	108,44%
c880	60	49	458,2	59	474,2	-3,37%
c1355	41	85	578,8	137	541,8	6,83%
c1908	33	111	571,6	122	533,6	7,12%
c2670	233	100	2700,6	124	2068,6	30,55%
c3540	50	143	1034,4	154	1004,2	3,01%
c5315	178	112	2462,4	120	2501,8	-1,57%
c6288	32	27	171,4	50	170,4	0,59%
c7552	207	178	4170,8	230	3959,8	5,33%

5. Conclusions

We have presented a new method, which provides a solution for the problem of test set embedding. The method is based on the notion of difference vectors and it uses a cyclic shift register in combination with OR networks and XOR gates. We perform a number of preprocessing steps on the given test set and from the resulting test data we construct the test pattern generation mechanism. The resulting TPG is far superior when compared with the phase shifter based approach [8] and is better than the LFSRom approach of [4] for fully specified test sets and the enhanced-LFSRom for partially specified test sets.

6. References

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

[2] L. F. C. Lew Yan Voon, C. Dufaza and C. Landrault, "BIST Linear Generator Based on Complemented Outputs", Proc. IEEE VLSI Test Symposium, 1992, pp. 137-142.

[3] G. Edirisooriya and J. P. Robinson, "Design of Low Cost ROM Based Test Generators", Proceedings of VLSI Test Symposium, 1992, pp. 61-66.

[4] C. Dufaza, C. Chevalier and L. F. C. Lew Yan Voon, "LFSROM. A Hardware Test Pattern Generator for Deterministic ISCAS85 Test Sets", Proceedings of the 2nd Asian Test Symposium (ATS'93), Beijing, China, November 16-18, 1993, pp. 160-165.

[5] D. Kagaris, S. Tragoudas and A. Majumdar, "On the Use of Counters for Reproducing Deterministic Test Sets", IEEE Trans. Comp., Vol 45, No. 12, Dec. 1996, pp. 1405-1419.

[6] D. Kagaris and S. Tragoudas, "On the Design of Optimal Counter-Based Schemes for Test Set Embedding", IEEE Trans. CAD, Vol. 18, No. 2, Feb. 1999, pp. 219-230.

[7] S. Swaminathan and K. Chakrabarty, "On Using Twisted-Ring Counters for Test Set Embedding in BIST", JETTA, Vol.17, No. 6, Dec. 2001, pp. 529-542.

[8] M. Bellos, D. Kagaris and D. Nikolos, "Test Set Embedding Based on Phase Shifters", Lecture Notes in Computer Science No. 2485, Edited by Andrea Bondavalli and Pascale Thevenod-Fosse, (Proceedings of 4th European Dependable Computing Conference, EDCC-4, Toulouse, France, October 2002), Springer – Verlag, pp. 90-101.

[9] Jas and N. A. Touba, "Test vector decompression via cyclical scan chains and its application to testing core-based designs", Proceedings of the 1998 International Test Conference, October 18-23, 1998, pp. 458-464.

[10] R. Murgai, M. Fujita and A. Oliveira, "Using Complementation and Resequencing to Minimize Transitions", Design Automation Conference, 1998, pp. 694-697.

[11] N. Drechsler and R. Drechsler, "Exploiting Don't Cares During Data Sequencing Using Genetic Algorithms", ASP Design Automation Conference, 1999, pp. 303-306.

[12] M. Bellos, D. Kagaris and D. Nikolos, "Low Power Test Set Embedding Based On Phase Shifters", Proceedings of the 2003 IEEE Computer Society Annual Symposium on VLSI (ISVLSI 2003), Tampa, Florida, USA, February 20-21, 2003, pp. 155-160.