

Reducing Test Application Time for Built-in-Self-Test Test Pattern Generators *

Ilker Hamzaoglu[†] and Janak H. Patel
 Center for Reliable & High-Performance Computing
 University of Illinois, Urbana, IL 61801

Abstract

This paper presents a new technique, called C-compatibility, for reducing the test application time of the counter-based exhaustive Built-in-Self-Test (BIST) test pattern generators. This technique reduces the test application time by reducing the size of the binary counter used in the test pattern generators. We have incorporated the synthesis algorithm for synthesizing BIST test pattern generators using the C-compatibility technique into ATOM, an advanced ATPG system for combinational circuits. The experimental results showed that the test pattern generators synthesized using this technique for the ISCAS85 and full scan versions of the ISCAS89 benchmark circuits achieve 100% stuck-at fault coverage in much smaller test application time than the previously published counter-based exhaustive BIST test pattern generators.

1 Introduction

Built-in-Self-Test (BIST) is an effective technique for testing VLSI circuits, since it reduces the dependency on expensive external test equipment and it provides at-speed and in-system testing of the circuit-under-test (CUT) [1, 3, 4]. The BIST technique requires an on-chip test pattern generator (TPG) and output response analyzer (ORA) for testing the CUT.

The applicability of the BIST technique to a CUT depends on the ability to design a low-overhead test pattern generator that can achieve very high fault coverage in a small amount of test application time. Different design techniques have been proposed to address this problem, e.g., exhaustive testing [1], pseudo-exhaustive testing [17], pseudo-random testing [1], pseudo-random testing using transformed pseudo-random patterns [18, 19], deterministic test set embedding [2, 5, 16, 20], and counter-based exhaustive

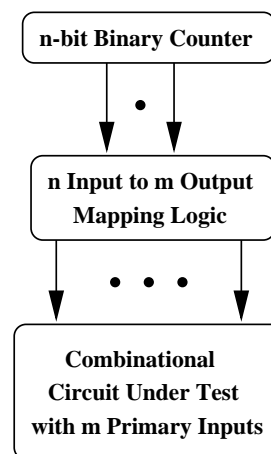


Figure 1: Counter-based BIST Test Pattern Generator

testing [9, 10, 11, 12].

As illustrated in Figure 1, the counter-based exhaustive testing techniques use an n -bit binary counter to generate all possible 2^n n -bit patterns that are then mapped to the primary inputs of the CUT, which has m primary inputs where $n < m$. These techniques try to minimize the size of the binary counter while achieving 100% stuck-at fault coverage. This is achieved by synthesizing the mapping logic using the compatibility relations between the primary inputs of a CUT [9, 11]. The mapping logic used in [11] includes only direct connections or inverters, whereas the one used in [9] also includes binary decoders. A similar technique based on a different compatibility relation is introduced in [6] for reducing the size of the binary counter used for syndrome-testing.

The counter-based exhaustive testing technique does not face the problem of random pattern resistant faults, and it achieves 100% stuck-at fault coverage with a low hardware overhead. However, it requires a relatively high test application time. Therefore, we propose a new technique for reducing the test application time of the counter-based exhaustive BIST test pattern generators by reducing the size of

*This research was supported in part by the Semiconductor Research Corporation under contract SRC 97-DJ-482 and in part by DARPA under contract DABT63-95-C-0069.

[†]Ilker Hamzaoglu is currently with Motorola Labs, Schaumburg, IL.

the binary counter using a small multiple-input combinational circuit in the mapping logic. The synthesis of this multiple-input combinational circuit is based on the new input compatibility relation, called *C-compatibility*, that we propose in this paper.

We have incorporated the synthesis algorithm for synthesizing BIST test pattern generators using the C-compatibility technique into ATOM, an advanced ATPG system for combinational circuits [13, 15]. The experimental results showed that the test pattern generators synthesized using the C-compatibility relation for the ISCAS85 and full scan versions of the ISCAS89 benchmark circuits achieve 100% stuck-at fault coverage in much smaller test application time than the previously published results. This shows that the C-compatibility technique is very effective for synthesizing low-overhead test pattern generators that require a small test application time.

The rest of the paper is organized as follows. Section 2 describes the input compatibility relations previously introduced in the literature and the C-compatibility relation we introduce in this paper. Section 3 presents the synthesis algorithm that we propose to synthesize BIST test pattern generators using the C-compatibility relation. The experimental results are given in Section 4. Section 5 presents the conclusions.

2 Input Compatibility Relations

In this section, we first present the definitions of the input compatibility relations previously introduced in the literature. We then present the new input compatibility relation we introduce in this paper which allows the synthesis of low-overhead BIST test pattern generators that require much smaller test application time than the previously published counter-based exhaustive test pattern generators.

Two inputs of a combinational circuit are said to be *directly compatible* if they can be shorted together without introducing any redundant stuck-at fault in the circuit. Similarly, if two inputs of a combinational circuit can be shorted together via an inverter without introducing any redundant stuck-at fault in the circuit, these two inputs are called *inversely compatible* [11]. Two inputs that are neither directly nor inversely compatible are called *incompatible* inputs.

A set of inputs are said to form a *compatibility class* if all the inputs in the set can be shorted together without introducing any redundant stuck-at fault in the circuit. Since the same (the opposite in case of inversely compatible inputs) logic value can be applied to all the inputs in a compatibility class during testing without introducing any redundant stuck-at fault in the circuit, the size of the binary counter used in

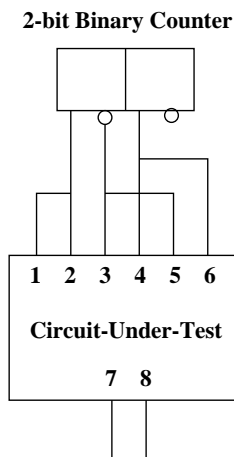


Figure 2: Direct and Inverse Compatibility Example

the counter-based exhaustive test pattern generators is equal to the number of compatibility classes.

Example: Consider the combinational circuit presented in Figure 2. Suppose that the inputs 1 and 2, 3 and 5, and 4 and 6 are directly compatible; moreover, inputs 1 and 2 are inversely compatible with inputs 3 and 5. Therefore, the inputs 1, 2, 3, and 5 form a compatibility class, and the inputs 4 and 6 form another compatibility class. As illustrated in the figure, since there are two compatibility classes, all the testable faults in this circuit can be tested by using a 2-bit binary counter.

Two inputs of a circuit are said to be *decoder-compatible (d-compatible)* if all the detectable stuck-at faults in the circuit can be tested by a test set in which no test vector requires both inputs to be one [9]. This definition can be generalized to include any number of inputs, i.e., inputs I_1, I_2, \dots, I_n of a circuit are said to be d-compatible if all the detectable stuck-at faults in the circuit can be tested by a test set in which every test vector requires at most one of these inputs to be one. Decoder-compatibility can be used to further reduce the size of the binary counter obtained by using the direct and inverse incompatibility of the inputs, because even if the inputs I_1, I_2, \dots, I_n are not directly or inversely incompatible, it is possible that these inputs are d-compatible. Therefore, the values of the inputs I_1, I_2, \dots, I_n in each test vector can be generated using a binary counter with $\log_2 n$ stages, thus reducing the size of the binary counter used in the test pattern generator.

In this paper, we introduce a new compatibility relation between inputs of a circuit called *combinational compatibility (C-compatibility)*. Inputs I_1, I_2, \dots, I_n of a circuit, T , are said to be *Cn-compatible* with another input of T , I_x , if the output of an n-input 1-output

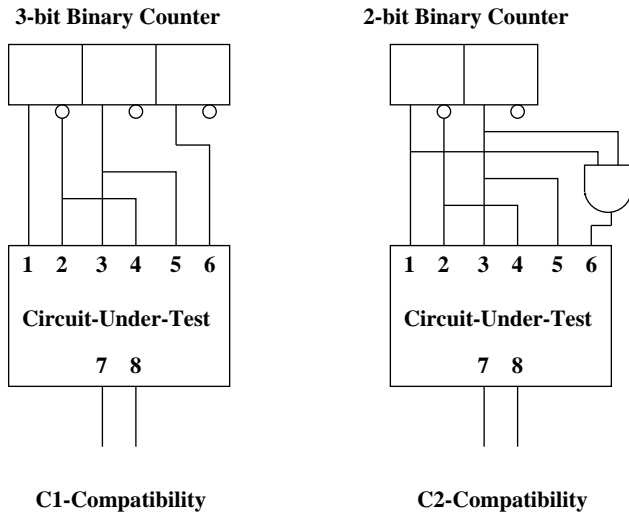


Figure 3: C-Compatibility Example

combinational circuit whose inputs are I_1, I_2, \dots, I_n can be connected to I_x without introducing any redundant stuck-at fault in T .

When $n = 1$, the combinational circuit degenerates to a 1-input 1-output combinational circuit, i.e., to a single line or an inverter. Therefore, in this case, the input I_1 is said to be *C1-compatible* with input I_x . Similarly, when $n = 2$, the combinational circuit degenerates to a 2-input 1-output combinational circuit. Therefore, in this case, the inputs I_1 and I_2 are said to be *C2-compatible* with input I_x . Similarly, *C3-compatibility* can be defined for $n = 3$, and so on.

C-compatibility is a more general compatibility relation between the inputs of a circuit than the previously introduced compatibility relations. The previously introduced direct, inverse and decoder-compatibilities are special cases of the C-compatibility. The direct and inverse compatibilities correspond to C1-compatibility. The decoder-compatibility corresponds to C-compatibility with a multiple output combinational circuit, where the combinational circuit is a decoder.

Therefore, C-compatibility provides more opportunities for reducing the size of the binary counter used in the test pattern generator. This in turn reduces the test application time. For example, even if the three inputs I_1, I_2 and I_3 of a circuit are pairwise incompatible and they are not d-compatible, any two of these inputs can be C2-compatible with the third one; thus, reducing the size of the binary counter from three to two, which reduces the test application time by 50%.

Example: Consider the combinational circuit presented in Figure 3. Suppose that the inputs 2 and 4, and 3 and 5 are directly compatible; moreover, input 1

-
1. Generate a complete partially specified test set for the given circuit
 2. Find the C1-compatibility, i.e., direct and inverse compatibility, classes using this test set
 3. Find the necessary input assignments for each testable fault
 4. Find the C1-compatibility classes by using test pattern generation only when necessary; i.e., avoid using test pattern generation for the inputs that can be proven to be incompatible using the necessary input assignments
 5. Find the C2-compatible inputs by considering all possible 2-input combinational functions for each input pair and by using test pattern generation only when necessary
-

Figure 4: Synthesis Algorithm

is inversely compatible with inputs 2 and 4. Therefore, the inputs 1, 2, and 4 form a compatibility class, the inputs 3 and 5 form another compatibility class, and the input 6 itself forms another compatibility class. As illustrated in the figure, since there are three compatibility classes, the counter-based exhaustive testing technique using only C1-compatibility relation, i.e., direct and inverse compatibilities, requires using a 3-bit binary counter. On the other hand, as illustrated in the figure, if the inputs 1 and 3 are C2-compatible with the input 6, then the counter-based exhaustive testing technique using both C1-compatibility and C2-compatibility relations requires using a 2-bit binary counter. Thus, C2-compatibility relation reduces the test application time by 50%.

3 Synthesis Algorithm

In this section, we will describe the algorithm that we use to synthesize the BIST test pattern generators using the C-compatibility relation. The overall synthesis algorithm is presented in Figure 4. In this paper, we only used the C1-compatibility and C2-compatibility relations for synthesizing test pattern generators. However, it is possible to extend the synthesis algorithm to use the C_n -compatibility relations, where $n > 2$.

3.1 Finding C1-compatible Inputs

The synthesis algorithm finds the C1-compatibility classes using a common greedy heuristic. The heuristic technique considers each input one at a time. The first input is assigned to the first compatibility class. If the next input is C1-compatible with the first one, it is also assigned to the same compatibility class; otherwise, it is assigned to a new compatibility class. If the next input is C1-compatible with all the inputs in an existing compatibility class, then it is added to that class; otherwise, it is assigned to a new compatibility class. This step is repeated for every input in the

	x_1	x_2	x_3	x_4
t_1	X	1	0	1
t_2	1	0	1	1
t_3	0	0	0	0
t_4	1	1	X	0
t_5	X	0	X	X

Figure 5: Partially Specified Test Set Matrix

circuit.

Computing the C1-compatibility classes requires checking the compatibility of a set of inputs. The synthesis algorithm first uses the heuristic technique described in [9, 11] for checking the compatibility of a given set of inputs. The heuristic technique first involves generating a complete partially specified stuck-at test set for the given circuit. In order to generate a test set in which each test vector has many unspecified inputs, during test generation, the inputs that are not assigned a logic value by the test generator are left unspecified, and the dynamic compaction algorithm is not used. The redundant vector elimination algorithm [14] is used to remove all the redundant vectors from this test set.

The resulting partially specified test set can be represented as a two-dimensional matrix, where each row is a test vector and each column is the set of values that will be assigned to a single input of the circuit. A set of columns in this matrix are compatible if for every row, the corresponding logic values of all the columns, except the ones with a don't care (X) logic value, are the same. An example test set matrix is shown in Figure 5 in which only the columns x_1 and x_3 are compatible. All the other columns are incompatible; e.g., the columns x_1 and x_2 are incompatible, since their logic values for the second row are different and neither of them is a don't care.

If a set of columns in a test set matrix are compatible, then the corresponding circuit inputs are compatible. However, if a set of columns are incompatible, this does not show that the corresponding inputs are incompatible, because it is possible that in a different test set, the columns corresponding to these inputs may be compatible. Therefore, this heuristic technique is pessimistic in identifying the compatible inputs; i.e., it may fail to prove the compatibility of compatible inputs.

After quickly identifying a large number of C1-compatibility classes by using this heuristic technique for checking the compatibility of a set of inputs, the

synthesis algorithm then tries to find more such classes by using a more computationally expensive technique, similar to the one presented in [11], for checking the compatibility of a given set of inputs. This technique checks the compatibility of a set of inputs by using test generation to determine whether shorting these inputs introduces any redundant stuck-at faults in the circuit. Since the computationally inexpensive technique usually identifies a large number of C1-compatibility classes, this computationally expensive technique needs to explore a small search space.

This technique checks the compatibility of a set of inputs as follows. It first tries to prove the incompatibility of these inputs without doing explicit test generation. This is achieved by using the necessary assignments for detecting all the testable faults in the circuit. Therefore, after using the computationally inexpensive heuristic technique for identifying C1-compatibility classes, the synthesis algorithm determines the necessary assignments for detecting each testable fault in the circuit using the unique sensitization and improved unique sensitization [13, 15] techniques. For each fault, only the necessary assignments corresponding to primary inputs are stored. If the necessary primary input assignments for a fault are the same as or covered by the necessary primary input assignments of another fault, the necessary assignments for this fault are not stored. Before using explicit test generation, the computationally expensive technique checks these necessary assignments to find out whether it can prove the incompatibility of the given inputs. For example, if the necessary primary input assignments of a fault require the value 1 for the input I_1 and the value 0 for the input I_2 , then this proves that the inputs I_1 and I_2 cannot be directly compatible, because if these two inputs are shorted together, at least this fault will become redundant.

If the incompatibility of the given set of inputs cannot be proved using the necessary primary input assignments, then the computationally expensive technique uses test generation to check the compatibility of these inputs. It maintains a complete test set during this process. When trying to decide the compatibility of a set of inputs, it checks the test set to identify the test vectors that have incompatible values for these inputs. For example, if the synthesis algorithm is checking for direct compatibility of the inputs I_1 and I_2 , a test vector has incompatible values for these two inputs if the value of one of the inputs is 1 and the value of the other one is 0. The faults detected only by the vectors that have incompatible values for the given inputs may potentially become redundant because of shorting

these inputs together. Therefore, the technique carries out test generation only for these faults to determine whether they actually become redundant or not. If any one of these faults is proven to be redundant, then it concludes that the given inputs are incompatible. If, on the other hand, all of these faults are detected, the inputs are declared to be C1-compatible.

3.2 Finding C2-compatible Inputs

After finding the C1-compatibility classes, the synthesis algorithm tries to reduce the size of the binary counter used in the test pattern generator by finding the C2-compatible inputs. In this process, it uses one representative input from each C1-compatibility class. It checks each pair of these inputs for a possible C2-compatibility with another representative input. Therefore, the size of the search space that needs to be explored in this process is proportional to the number of C1-compatibility classes. Since the number of C1-compatibility classes of a given combinational circuit is usually much less than the number of its primary inputs, the size of this search space is relatively small.

For a given integer n , the total number combinational functions with up to n inputs is given by the expression 2^{2^n} . Thus, there are $2^{2^2} = 16$ combinational functions with up to 2 inputs. Two of these are 0-input functions, i.e., constant 0 and constant 1, four of them are 1-input functions, i.e., x , \bar{x} , y , \bar{y} , and the remaining ten are 2-input functions, i.e., $x AND y$, $\bar{x} AND y$, $x AND \bar{y}$, $\bar{x} AND \bar{y}$, $x OR y$, $\bar{x} OR y$, $x OR \bar{y}$, $\bar{x} OR \bar{y}$, $x XOR y$, $x XNOR y$.

The 1-input combinational functions are used for finding C1-compatible inputs, and the 2-input combinational functions are used for finding C2-compatible inputs. Therefore, for each input pair, ten different gate types are tried for deciding whether these two inputs are C2-compatible with another input.

Given an input pair (I_a, I_b) and another input I_c , for each gate type, the synthesis algorithm first checks the necessary primary input assignments in order to prove that I_a and I_b cannot be C2-compatible with I_c . For example, if the necessary primary input assignments of a fault requires the value 1 for the input I_a , the value 1 for the input I_b , and the value 0 for the input I_c , then this proves that the inputs I_a and I_b cannot be C2-compatible with I_c for the $x AND y$ gate type. Because if the output of a 2-input AND gate, whose inputs are connected to I_a and I_b , is connected to I_c , at least this fault will become redundant.

If the C2-incompatibility of three inputs cannot be proved using the necessary primary input assignments, then the synthesis algorithm uses test genera-

tion to check the C2-compatibility of the three inputs. It maintains a complete test set during this process. When trying to decide the C2-compatibility of two inputs with a third one, for each gate type, it checks the test set to identify the test vectors that have incompatible values for these three inputs when this gate type is used. For example, if the synthesis algorithm is checking the C2-compatibility of the inputs I_1 and I_2 with the input I_3 using the gate type $x AND y$, a test vector has incompatible values for these three inputs under this gate type if the value of either I_1 or I_2 is 0 and the value of I_3 is 1.

The faults detected only by the vectors that have incompatible values for the three inputs may potentially become redundant because of connecting the output of the 2-input gate, whose inputs are connected to the first two inputs, to the third input. Therefore, the synthesis algorithm carries out test generation only for these faults to determine whether they actually become redundant or not. If any one of these faults is proven to be redundant, then it concludes that these two inputs are not C2-compatible with the third one. If, on the other hand, all of these faults are detected, the inputs are declared to be C2-compatible.

4 Experimental Results

We have incorporated the synthesis algorithm into ATOM, an advanced ATPG system for combinational circuits [13, 15]. ATOM and the synthesis algorithm are implemented in C++. ATOM is tested using the ISCAS85 and full scan versions of the ISCAS89 benchmark circuits [7, 8] on a 200 MHz Pentium Pro PC with 128MB RAM running Linux 2.0.0 using GNU CC version 2.8.0. In all the experiments, a backtrack limit of 9 is used in ATOM. All the test sets generated by ATOM have 100% fault coverage.

The performance results of ATOM are compared against the best published results in the literature, and the comparison of the performance results is shown in Table 1. The experimental results for the largest three ISCAS89 circuits were not reported in [11, 12], as indicated by the “-” sign in the table. The percentage of test application time reduction for a circuit is computed as $(Original\ Test\ Cycles - Reduced\ Test\ Cycles) / (Original\ Test\ Cycles)$ where $Original\ Test\ Cycles$ is computed as $2^{Counter\ Size\ Reported\ in\ [11, 12]}$ and $Reduced\ Test\ Cycles$ is computed as $2^{Counter\ Size\ Obtained\ Using\ C-compatibility}$. The original test cycles for the benchmark circuits for which a counter size is not reported in [11, 12] are computed by using the counter sizes that are obtained by only using the C1-compatibility relation. In [11, 12], the

Circuit	No. of Inputs	Counter Size			Test Application Time Reduction	Synthesis Time (secs)	
		[11, 12]	C1-comp.	C2-comp.		C1-comp.	C2-comp.
c432	36	12	10	9	87.5%	1.7	5.4
c499	41	9	11	9	0%	10.7	13.0
c880	60	13	13	10	87.5%	4.2	48.6
c1355	41	11	11	10	50%	30.6	141.3
c1908	33	13	13	12	50%	33.9	258.8
c2670	233	22	22	20	75%	57.9	2083.1
c3540	50	17	18	15	75%	43.4	294.1
c5315	178	13	16	11	75%	133.2	659.5
c6288	32	6	8	6	0%	98.8	137.2
c7552	207	28	26	23	96.9%	837.7	6117.6
s208	19	11	11	9	50%	0.2	1.7
s298	17	7	7	7	0%	0.2	0.3
s344	24	7	7	6	50%	0.3	0.5
s349	24	7	7	6	50%	0.3	0.5
s382	24	7	8	7	0%	0.2	0.4
s386	13	11	11	10	50%	0.5	0.5
s400	24	7	7	7	0%	0.2	0.3
s420	35	19	19	13	98.4%	0.5	13.7
s444	24	8	8	7	50%	0.2	0.3
s510	25	8	8	8	0%	0.6	0.6
s526	24	13	13	12	50%	0.5	0.7
s526n	24	-	13	12	50%	0.5	0.5
s641	54	15	15	10	96.9%	1.2	2.8
s713	54	15	15	10	96.9%	1.4	3.6
s820	23	13	13	12	50%	3.4	3.5
s832	23	13	13	12	50%	3.7	3.8
s838	67	35	35	13	99.9%	2.3	130.3
s953	45	16	16	12	93.8%	2.3	2.6
s1196	32	15	17	14	50%	2.7	5.3
s1238	32	15	17	15	0%	3.4	7.4
s1423	91	13	14	12	50%	6.4	77.2
s1488	14	12	12	10	75%	6.4	7.8
s1494	14	12	12	10	75%	6.5	6.7
s5378	199	16	19	16	0%	28.4	162.4
s9234	247	30	30	22	99.6%	100.3	2443.4
s13207	700	27	27	17	99.9%	129.9	209.1
s15850	611	31	31	26	96.9%	154.1	1010.0
s35932	1763	-	6	6	0%	594.0	643.0
s38417	1664	-	30	25	96.9%	1529.1	16,948
s38584	1464	-	31	25	98.4%	1033.7	2134.9

Table 1: Synthesis Results

execution times for synthesizing the test pattern generators were not reported. The synthesis times for the C-compatibility technique presented in the table include all the test generation and fault simulation times wherever applicable. The synthesis times presented in the column headed C2-comp. also include the execution times for finding the C1-compatibilities.

Since reducing the size of the binary counter reduces the test application time exponentially, i.e., reducing the size by 1 reduces the test application time by 50%, reducing the size by 2 reduces the test application time by 75%, and so on, we have achieved significant test application time reduction for the counter-based exhaustive test pattern generators. The experimental results show that the test pattern generators synthesized using the C-compatibility technique require much smaller test application time for the ISCAS85 and full scan versions of the ISCAS89 benchmark circuits than the previously published counter-based exhaustive BIST test pattern generators.

Since reducing the size of the binary counter by 1 using a 2-input combinational circuit effectively replaces a flip-flop with a 2-input gate, reducing the test application time by using the C1-compatibility and C2-compatibility relations does not introduce an additional hardware overhead. In other words, the counter-based exhaustive test pattern generators synthesized using the C-compatibility technique achieve much smaller test application time than the previously published ones with a similar hardware overhead. Thus, the experimental results show that the C-compatibility technique is very effective for synthesizing low-overhead test pattern generators that require a small test application time.

5 Conclusions

In this paper, we presented a new technique, called C-compatibility, for reducing the test application time of Built-in-Self-Test (BIST) test pattern generators. We have incorporated the synthesis algorithm for synthesizing BIST test pattern generators using the C-compatibility technique into ATOM. The experimental results showed that the test pattern generators synthesized using this technique for the ISCAS85 and full scan versions of the ISCAS89 benchmark circuits achieve 100% stuck-at fault coverage in much smaller test application time than the previously published counter-based exhaustive BIST test pattern generators.

References

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

[2] V. K. Agarwal and E. Cerny, "Store and Generate Built-in Testing Approach," *Int. Symp. on Fault-Tolerant Computing*, pp. 35-40, June 1981.

[3] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A Tutorial on Built-in Self Test (Part 1)," *IEEE Design and Test of Computers*, pp. 73-82, March 1993.

[4] V. D. Agrawal, C. R. Kime and K. K. Saluja, "A Tutorial on Built-in Self Test (Part 2)," *IEEE Design and Test of Computers*, pp. 69-77, June 1993.

[5] S. B. Akers and W. Jansz, "Test Set Embedding in a Built-in-Self-Test Environment," *Proc. Int. Test Conf.*, pp. 257-263, October 1989.

[6] Z. Barzilai, J. Savir, G. Markowsky, and M. G. Smith, "The Weighted Syndrome Sums Approach to VLSI Testing," *IEEE Trans. on Computers*, vol. C-30, no. 12, pp. 996-1000, December 1981.

[7] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Designs and a Special Translator in Fortran," *Proc. Int. Symp. on Circuits and Systems*, June 1985.

[8] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," *Proc. Int. Symp. on Circuits and Systems*, pp. 1929-1934, May 1989.

[9] K. Chakrabarty, B. T. Murray, J. Liu, and M. Zhu, "Test Width Compression for Built-In Self Testing," *Proc. Int. Test Conf.*, pp. 328-337, October 1997.

[10] K. Chakrabarty and B. T. Murray, "Design of Built-In Test Generator Circuits Using Width Compression," *IEEE Trans. on Computer-Aided Design*, pp. 1044-1051, vol. 17, no. 10, October 1998.

[11] C. Chen and S. K. Gupta, "A Methodology to Design Efficient BIST Test Pattern Generators," *Proc. Int. Test Conf.*, pp. 814-823, October 1995.

[12] C. Chen and S. K. Gupta, "Efficient BIST TPG Design and Test Set Compaction via Input Reduction," *IEEE Trans. on Computer-Aided Design*, pp. 692 -705, vol. 17, no. 8, August 1998.

[13] I. Hamzaoglu and J. H. Patel, "New Techniques for Deterministic Test Pattern Generation," *Proc. IEEE VLSI Test Symp.*, pp. 446-452, April 1998.

[14] I. Hamzaoglu and J. H. Patel, "Test Set Compaction Algorithms for Combinational Circuits," *Proc. Int. Conf. on Computer-Aided Design*, pp. 283-289, November 1998.

[15] I. Hamzaoglu and J. H. Patel, "New Techniques for Deterministic Test Pattern Generation", *Journal of Electronic Testing: Theory and Applications*, vol. 15, no. 1/2, pp. 63-73, October 1999.

[16] M. Lempel, S. K. Gupta, and M. A. Breuer, "Test Embedding with Discrete Logarithms," *IEEE Trans. on Computer-aided Design*, vol. 14, no. 5, pp. 554 -566, May 1995.

[17] E. J. McCluskey, "Verification Testing - A Pseudoexhaustive Test Technique," *IEEE Trans. on Computers*, vol. C-33, no. 6, pp. 541-546, June 1984.

[18] N. A. Touba and E. J. McCluskey, "Transformed Pseudo-Random Patterns for BIST," *Proc. IEEE VLSI Test Symp.*, pp. 410-416, April 1995.

[19] N. A. Touba and E. J. McCluskey, "Synthesis of Mapping Logic for Generating Transformed Pseudo-Random Patterns for BIST," *Proc. Int. Test Conf.*, pp. 674-682, October 1995.

[20] S. Venkataraman, J. Rajski, S. Hellebrand, and S. Tarnick, "An Efficient BIST Scheme Based on Reseeding of Multiple Polynomial Linear Feedback Shift Registers," *Proc. Int. Conf. on Computer-Aided Design*, pp. 572-577, November 1993.