University of Nebraska - Lincoln
DigitalCommons@University of Nebraska - Lincoln

CSE Conference and Workshop Papers | Computer Science and Engineering, Department of

2005

Efficient Test Compaction for Pseudo-Random Testing

Sheng Zhang
University of Nebraska-Lincoln, szhang@cse.unl.edu

Sharad C. Seth
University of Nebraska - Lincoln, seth@cse.unl.edu

Bhargab B. Bhattacharya
Indian Statistical Institute, Kolkata, India, bhargab@isical.ac.in

Follow this and additional works at: http://digitalcommons.unl.edu/cseconfwork

Part of the Computer Sciences Commons

Zhang, Sheng; Seth, Sharad C.; and Bhattacharya, Bhargab B., "Efficient Test Compaction for Pseudo-Random Testing" (2005). *CSE Conference and Workshop Papers*. 7.
http://digitalcommons.unl.edu/cseconfwork/7

This Article is brought to you for free and open access by the Computer Science and Engineering, Department of at DigitalCommons@University of Nebraska - Lincoln. It has been accepted for inclusion in CSE Conference and Workshop Papers by an authorized administrator of DigitalCommons@University of Nebraska - Lincoln.

Efficient Test Compaction for Pseudo-Random Testing

Sheng Zhang, Sharad C. Seth
Department of Computer Sci. and Eng.
University of Nebraska-Lincoln
Lincoln, NE 68588-0115, USA
{szhang, seth}@cse.unl.edu

Bhargab B. Bhattacharya
ACM Unit
Indian Statistical Institute
Calcutta – 700 108, India
bhargab@isical.ac.in

ABSTRACT

Compact set of 3-valued test vectors for random pattern resistant faults are covered in multiple test passes. During a pass, its associated test cube specifies certain bits in the scan chain to be held fixed and others to change pseudo -randomly. We propose an algorithm to find a small number of cubes to cover all the test vectors, thus minimizing total test length. The test-cube finding algorithm repeatedly evaluates small perturbations of the current solution so as to maximize the expected test coverage of the cube. Experimental results show that our algorithm covers the test vectors by test cubes that are one to two orders of magnitude smaller in number with a much smaller increase in the percentage of specified bits. It outperforms comparable schemes reported in the literature.

Keywords: Test compaction, test-data compression, pseudo-random testing, built-in testing.

1. Introduction

In this paper we provide an algorithmic solution to a test compaction problem. The traditional (static) test compaction problem for combinational logic is to find a minimal cover of a given set of 3-valued *test vectors* T = {T_1, T_2, …, T_n} by a set of 3-valued *test cubes* C = {C_1, C_2, …, C_m}, such that for every test vector T_i, there is a *compatible* test cube C_j that covers (includes) all the bits (0 or 1) of T_i. Here, compatibility means the absence of a 0-1 conflict in any position of T_i and C_j.

We consider a more general form of test compaction problem in which certain bits (0 or 1) of a test vector are allowed to be covered by x-values in the compatible test cube. We denote by $\delta(T_i, C_j)$ the number of such bits or the *distance* of T_i from C_j. Note that if $\delta(T_i, C_j) = d$ and if every x-bit in C_j is randomly set to 0 or 1 with equal probability, while keeping the remaining bits at their fixed value, the probability of *covering* T_i by one such random pattern will be precisely 2^{-d}. As more such patterns are applied, the cumulative probability of covering T_i will rise, with the expected number of patterns to cover T_i being 2^d. Thus, the generalization leads to a hybrid form of test application scheme in which $O(2^{dmax})$ random patterns are applied corresponding to each test cube so as to cover all compatible test vectors that are at most distance *dmax* from the cube. The parameter *dmax* may be used to optimize the expected test length.

As an example, suppose the set T consists of the four test vectors shown in the top part of Table 1. The bottom part shows three test cubes and the right part shows the distances of the test vectors to these test cubes. It will be seen that if *dmax* is 4, C_1 is sufficient to cover the four test vectors. However, if dmax is reduced to 3, no single cube can be the cover but either {C_1, C_3} or {C_2, C_3} will suffice. Indeed, {C_2, C_3} will still be a cover if *dmax* is reduced to 2. If we apply precisely 2^{dmax} number of test patterns corresponding to the *dmax* value of the cover, then we will need 1*16, 2*8, and 2*4 test patterns respectively for the covers {C_1}, {C_1, C_3}, and {C_2, C_3}. A more complex testing scheme might allow application of a variable number of tests for test cubes in the cover, in which case, the test length for the cover {C_1, C_3} can be reduced to from 2*8 (16) to 1*8+1*4 (12).

Table 1: Test Vectors and Test Cubes

	x_1	x_2	x_3	x_4	x_5	$\delta(-,C_1)$	$\delta(-,C_2)$	$\delta(-,C_3)$
T_1:	x	x	0	1	0	3	2	∞
T_2:	x	0	1	0	1	4	∞	2
T_3:	0	x	1	1	1	3	3	1
T_4:	0	x	1	x	1	2	3	0
C_1:	0	x	x	x	x			
C_2:	x	x	x	1	x			
C_3:	0	x	1	x	1			

From the previous discussion and example, we can now state a precise formulation of the test compaction problem considered in this paper.

> **(Test Compaction Problem):** Given a set of 3-valued *test vectors* T = {T_1, T_2, …, T_n} and a positive integer *k*, find the smallest set of 3-valued *test cubes* C = {C_1, C_2, …, C_m}, such that for every test vector T_i, there is a compatible test cube C_j satisfying the condition that there are at most *k* bit positions where T_i is 0 or 1 and C_j is x.

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

This test compaction problem can be shown to be NP-complete by reduction from the minimum set cover problem [1], hence efficient polynomial-time algorithms must be found that provide good approximations.

The general problem of circuit-specific customization of pseudo-random sequence generators has been vigorously studied for the last twenty years. Proposed solutions involve reseeding of LFSR [2, 3, 4, 5, 6, 7, 8, 9, 10], weighted random-pattern generation (WRPG) [11, 12, 13, 14, 15, 16], and embedding of deterministic patterns in the pseudo-random sequence by using bit-fixing or bit-flipping logic, sometimes in combination with Markov sources [17, 18, 19, 20]. Along this spectrum of approaches, our contribution falls under WRPG because the 0, 1, and x values in a cube can be mapped to the very simple weight set, {0, 1, 0.5}.

The primary contribution of this paper is a new algorithm to solve the test set compaction problem described above. Previously proposed algorithms [11, 13, 15, 16] to solve this problem all follow a greedy approach in which an initial test cube is defined in some fashion and updated by selecting another test vector from among the uncovered test vectors. In contrast, our algorithm considers all the uncovered test vectors simultaneously, as it modifies the initial test cube bit by bit to improve its expected coverage. This use of the global context and of a precise coverage metric allows our algorithm to produce solutions that are generally superior to even those obtained with more than three weights. Further, while the focus in this paper is on test-length reduction, our experimental results show that the logic overhead in all cases is quite small.

2. Test Generation Framework

In applying the test compaction approach, we must consider the source and nature of the original set of test vectors. We assume that the test vectors are produced by a deterministic test generation system to have the desired fault coverage. Further, the test generator must maximize "don't cares" during test generation, otherwise, it may not be possible to achieve good test compaction due to the "redundant" or "contradictory" information contained in the test set [21]. On the other hand, if the don't cares are not filled in, the fault simulation steps may not be able to drop many faults that would otherwise have been detected, thus leading to an impractically large number of test generation steps. Practical solutions to this dilemma require that the ATPG be run only on a small fraction of random-pattern-resistant (RPR) faults identified either by fault simulating a certain number of random patterns or by testability analysis. We follow the first alternative, and include it in the initial step of our overall test-generation framework described below:

1. Identify a set of RPR faults by fault-simulating a fixed number (256, 512, or 1024 in our experiments) of pseudorandom patterns.

2. For the identified RPR faults, generate a compact set of 3-valued test vectors using an ATPG tool, ignoring the faults that are either proven redundant or aborted by the ATPG tool.

3. Find the next test cube by the algorithm described in Section 3 using, as input, the current set of uncovered test vectors (i.e. those associated with RPR faults that are not yet detected).

4. Generate a *cube-contained* pseudorandom test-vector sequence of a fixed length (256, 512, or 1024 in our experiments) to be applied during the current pass, where cube contained means that the binary (0-1) bits of the test cube remain fixed during this pass while the x bits change randomly.

5. Fault-simulate the pseudorandom test-vector sequence on the as-yet undetected RPR faults and update the fault set.

6. Repeat Steps 3 through 5 until all RPR faults are covered.

7. Let S represent the test obtained by concatenating individual test sequences determined as above. Augment S with the pseudorandom sequence used in Step 1 to identify RPR faults.

An alternative to Step 7 (not followed here) would be to fault simulate S on all non-RPR faults (most of whom ought to be covered by the definition of RPR faults) and treat the uncovered RPR faults as the starting point for another test generation process described in Steps 2 through 6.

3. Finding Test-Cubes

Coverage Metric. The test-cube finding algorithm uses a coverage metric for a test cube based on the distance function defined earlier. It is not hard to prove that if $\delta(T_i, C_j) = d_i$ and l patterns are generated that pseudo-randomly fill the x bits of C_j, while holding its fixed bits at their specified values, then the probability of covering T_i by C_j is $1-(1-2^{-d_i})^l$. For moderately small values of d_i, this coverage probability rapidly approaches 1 for $l \geq 2^{d_i}$ (see Table 2).

Table 2: Test length vs. coverage

		l:			
		2^{d_i}	$2*2^{d_i}$	$4*2^{d_i}$	$6*2^{d_i}$
d_i:	6	.635	.867	.982	.998
	7	.634	.866	.982	.998
	8	.633	.865	.982	.998

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

From the table we see that a test vector at distance d_i from a cube is almost certainly covered by a cube-contained random-pattern sequence of length $4*2^{d_i}$. By summing up the coverage probabilities of individual test vectors we get the overall coverage metric (*weight*) of a test cube by a test sequence of length l as follows:

$$WTC(C_j) = \sum_{i:T_i \text{ is coverd by } C_j} 1-(1-2^{-d_i})^l \qquad (1)$$

When l is expressed as 2^d, for some d, a test vector T_i with $d_i > d+10$ contributes negligibly small amount to the sum and can be ignored in the computation of WTC.

Algorithm. The basic idea of the test-cube finding algorithm is to start with a maximally specified cube that is compatible with all the test vectors and in successive steps modify it so as to maximize its WTC coverage of the test vectors. We use a greedy algorithm in which several trial candidates for the test cube are generated from the current cube and the one that provides the maximum positive gain is selected to replace the current cube. The algorithm terminates when none of the candidates can improve on the WTC value of the current cube, i.e., a local maxima has been reached. A key feature of the algorithm is the way in which candidate cubes are obtained from the current test cube by considering incremental changes in each vector position. We explain the steps of the algorithm next by means of examples.

Consider, again, the four test vectors introduced in Section 1 and repeated in Table 3 for ease of reference. The initial cube C_0 is maximally specified, i.e. whenever only one binary value occurs in a column, the column in C_0 is set to that value and if both 0 and 1 occur or all values are x in the column, then C_0 is set to x. These rules insure that C_0 is a compatible with all the test vectors and, among all cubes that have this property, it has a minimal distance to any test vector. We will denote this process of deriving a cube from a collection of rows as *collapsing* the rows.

For this example, we assume that 4 random patterns are generated for each cube, therefore, the WTC computation is carried out according to Eq (1) for l=4. Using the test vector distances shown, this yields WTC = 1.39 for C_0 as shown in Table 4.

Candidate cubes are generated from this initial cube by considering single *compatible* changes in each position: if the original cube value is binary, we consider changing it only to x. On the other hand, if the original value is x, we consider changing it to both 0 and 1. After each change, we eliminate originally compatible rows (test vectors) in that column that now have an increased distance to the modified cube. For example, in column x_1, when the 0 in C_0 is changed to x, the distance of rows T_3 and T_4 will increase by 1, therefore we eliminate these rows and derive the new cube by collapsing the remaining rows. The resulting cube is C_1 = x 0 x x x (Table 3) and WTC evaluates to 0.34 (Table 4). As the change in column x_1 does not improve WTC, C_1 is not a candidate to replace C_0

Table 3: Test vectors and test cubes

	x_1	x_2	x_3	x_4	x_5
T_1:	x	x	0	1	0
T_2:	x	0	1	0	1
T_3:	0	x	1	1	1
T_4:	0	x	1	x	1
C_0:	0	0	x	x	x
C_1	x	0	x	x	x
C_2	0	x	x	1	x
C_3	0	0	1	x	1

Table 4: Test-cube coverage

	$\delta(-,C_0)$	$\delta(-,C_1)$	$\delta(-,C_2)$	$\delta(-,C_3)$
T_1:	3	3	2	∞
T_2:	3	3	∞	1
T_3:	3	4	2	1
T_4:	2	4	2	0
WTC:	1.39	0.34	2.05	**2.88**

Now consider the change 0->x in column x_2. In this case, we delete row T_2 and get the cube C_2 = 0 x x 1 x by collapsing the remaining rows. C_2 is now incompatible with row T_2 in column x_4, which results in an infinite distance of this row from the cube. The other row distances are all 2 and the resulting WTC (C_2) = 2.05. As this improves the original WTC, C_2 stays as a candidate to replace C_0.

To illustrate a change in the opposite direction, i.e. from x to 0 or 1, consider the example of changing the cube value in column x_3 from x to 1. After deleting the first row (because it is now incompatible), and collapsing the remaining rows, the new candidate cube is C_3 = 0 0 1 x 1 and WTC(C_3) = 2.88. Therefore, C_3 is a better candidate for replacement of C_0 than C_2.

When all the remaining changes are considered, it is found that none of them improves on C_3; therefore, C_3 is used to replace C_0 in the first iteration of the algorithm. At this point, four pseudorandom patterns will be applied to the circuit using the selected cube and fault simulated to determine which faults are actually detected. After deleting the corresponding rows, another iteration of cube-finding algorithm will ensue. For the example, we assume that the fault simulation eliminated

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

all but the first row. Then, the next iteration will trivially produce the cube C_4 = xx010 corresponding to the first test pattern. Any pseudorandom pattern contained in this cube will cover T_1, because its distance to the cube is 0. Hence, the algorithm will find $\{C_3, C_4\}$ as the cover for the four test vectors.

Figure 2 formally summarizes the above discussion in the form of an algorithm. Note that the Eval function in the algorithm encapsulates the steps of deleting rows for which the bit-change increases the distance from the cube, collapses the remaining compatible rows to define C_{new} and computes CM_{new} as the coverage value of C_{new}.

```
NextTestCube(T, I)
 C = Collapse(T);  \* Initial cube
 CM = WTC(C); CMold = CM; \* Coverage metric
 loop forever {
        for i = 1, n {       \* change bit i and evaluate
                if C[i] is 0 or 1 then
                        C*[i] = x;
                        Eval(C*)
                else
                        C*[i] = 0; Eval(C*);
                        C*[i] = 1; Eval(C*);
        }
        if CM>CMold then \* update if better cube found
                C = Cnew; CM = CMnew
        else break loop   \* otherwise stop search
 }
```

Figure 2: Algorithm to find the next cube

4. Experimental Results

The results of our implementation are shown in Table 5. The first three columns of the table show the circuit name, the number of circuit inputs, and the number (L) of pseudo-random patterns applied for each test cube. The fourth column shows the number of three-value test vectors obtained from the ATPG tools to cover the RPR faults identified by fault-simulating L pseudo-random patterns. The average number of specified bits (ASB) in these tests is shown as the percentage of the number of circuit inputs in the fifth column. The remaining columns indicate the performance of the cube-contained pseudo-random testing proposed in this paper. Columns 6 through 8 respectively record the number of cubes, number of tests (product of the number of cubes and L), and the percentage ASB in the test cubes. Columns 9 and 10 indicate the performance of the cube-finding algorithm in terms of the average and maximum numbers of iterations of the outer loop required to find a cube. The last two columns indicate the lack of fault coverage, which results from the ATPG tool aborting for some faults. Comparing these two columns, it is clear that even some of ATPG-aborted faults are fortuitously covered by our pseudo-random sequence.

Comparing columns 4 and 6, we note that our algorithm is able to cover the original ATPG tests by test cubes that are typically an order to two orders of magnitude smaller in number. The corresponding increase in the percentage ASB (col. 5 vs. col. 8) is much smaller in every case. For example, for s38417, with L=256, the original 1,986 tests were covered by 20 test cubes while the % ASB went up from 2.1 to 23.8.

In Table 6 we compare our best test-length results with the best published in the literature. Our cubes are directly comparable with the three-weight sets in [11, 13, 11, 15, 16]. However, reference [15] also considers more than three weights in their algorithm and their best results (shown in parentheses) come from 9-weights. References [21, 22] allow multiple arbitrary weights. Our algorithm consistently performs significantly better than others' for all but chkn-ML, the multi-level synthesized circuit, for which our test length is only slightly worse. It should be noted, however, that the chkn-ML circuits in the two cases were obtained by different synthesis tools (*Sis* vs. *Pendulum*) and the results in [11] are averages over ten runs.

The WTC measure, as defined in Eq. (1), aims only to optimize the test length and does not include a parameter, (such as the number of fixed bits in a cube) to reflect the cost of hardware implementation. We have not yet carried out detailed experiments to compute the exact area overhead but a limited experiment with s38584 circuit shows the overhead of the mapping logic to be less than 6.7% of the combinational part of the benchmark circuit.

5. Conclusion

We presented a new algorithm for solving a version of the test compaction problem that has received much attention from researchers in the last two decades because it provides a practical solution to covering random-pattern resistant fault in the pseudo-random testing environment. Our experimental results on a variety of benchmark circuits demonstrate significant improvement in test length over comparable published results.

Our metric to evaluate a test cube currently accounts for only the test length but can be easily modified to include a measure that appropriately weights the cube coverage with the number of its fixed bits to represent the logic cost.

Further performance improvement may be possible if the greedy nature of the algorithm was changed to allow

Proceedings of the 14th Asian Test Symposium (ATS '05)
1081-7735/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

for occasional acceptance of intermediate solutions that do not improve the expected coverage, e.g. in a simulated-annealing type of implementation. This is left as a future area of research. Our formulation is also amenable to future genetic-algorithm implementation.

Acknowledgment: This work was partly supported by a grant from Intel Corporation. The authors acknowledge S. Patil, S. Mitra, and V. Gangaram for their feedback on an earlier version of the paper.

Table 5: Performance of cube-contained pseudo-random testing

Circuit	# Inputs	L	Orig. Tests		Cube-contained testing			Ave Iters.	Max Iters.	ATPG Undet.	Ours Undet.
			# Tests	% ASB	# Cubes	# Tests	% ASB				
c2670	233	256	88	16.2	12	3072	22.4	18	43	0	0
		512	75	16.6	9	4608	22.3	16	39	0	0
		1024	71	16.9	8	8192	20.9	15	34	0	0
c7552	207	256	191	23.5	18	4608	46.1	34	73	4	0
		512	160	25.2	14	7168	43.1	24	62	2	0
		1024	146	26.3	12	12288	46.5	22	50	0	0
s13207	700	256	1154	2.3	10	2560	12.7	18	45	1	1
		512	1017	2.3	8	4096	11.0	14	38	1	1
		1024	886	2.3	7	7168	11.2	15	50	1	1
s15850	611	256	775	2.7	11	2816	18.3	22	74	0	0
		512	615	2.9	9	4608	18.9	22	55	0	0
		1024	484	3.1	6	6144	19.5	27	54	0	0
s38417	1664	256	1986	2.0	20	5120	23.8	144	458	0	0
		512	1824	2.1	15	7680	25.4	133	392	0	0
		1024	1685	2.2	11	11264	27.5	133	369	0	0
s38584	1464	256	2087	0.9	8	2048	17.9	45	140	0	0
		512	1573	1.0	7	3584	14.6	22	52	0	0
		1024	1241	1.0	6	6144	13.7	18	53	0	0
chkn-TL	29	256	266	63.5	15	3840	57.1	11	10	0	0
		512	239	66.9	11	5632	55.5	9	12	0	0
		1024	218	69.6	10	10240	54.4	5	11	0	0
chkn-ML	29	256	122	59.2	9	2304	62.9	7	12	0	0
		512	106	62.7	7	3584	63.8	5	12	0	0
		1024	91	66.8	5	5120	55.2	5	8	0	0

Table 6: Number of Weights/Cubes and Test Length Comparison with the Best Published Results

Circuit	Ours	Kapur [15]	Waic [22]	Pome [13]	Pate [11]	Reeb [21]	Wang [16]	Li [23]
c2670	12/**3072**	15/4608 (5/3840)	8/5888	15/30675	-	2/9600	8/8K	-
c7552	18/**4608**	21/6656 (7/6400)	10/9728	36/73728	-	6/17046	9/6.7K	-
S13207	10/**2560**	-	-	-	-	3/7026	6/71.6k	20.4k
s15850	11/**2816**	-	-	-	-	3/7985	21/87.0K	13.9K
s38417	20/**5120**	-	-	-	-	11/42606	26/86.0K	48.5K
s38584	8/**2048**	19/18.7K	-	-	-	3/12581	8/49.1K	33.2K
chkn-TL	15/**3840**	-	-	-	20/20000	-	-	-
chkn-ML	9/2304	-	-	-	20/**1900**	-	-	-

1081-7735/05 $20.00 © 2005 IEEE

IEEE COMPUTER SOCIETY

References

[1] M. R. Garey and D. S. Johnson, *Computers and intractability. A guide to the theory of NP-completeness*. Oxford, UK: Freeman, 1979.

[2] S. Hellebrand, J. Rajski, S. Tarnick, S. Venkataraman, and B. Courtois, "Built-in test for circuits with scan based on reseeding of multiple-polynomial linear feedback shift registers," *IEEE Transactions on Computers*, vol. 44 (2), pp. 223-233, 1995.

[3] S. Hellebrand, B. Reeb, S. Tarnick, and H.-J. Wunderlich, "Pattern generation for a deterministic BIST scheme," in *Proc. IEEE/ACM International Conference on Computer-Aided Design* 1995, pp. 88-94.

[4] C. Fagot, O. Gascuel, P. Girard, and C. Landrault, "On calculating efficient LFSR seeds for built-in self test," in *Proc. European Test Workshop* 1999, pp. 7-14.

[5] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial LFSR reseeding," in *Proc. International Test Conference* 2001, pp. 885-893.

[6] C. V. Krishna and N. A. Touba, "Reducing test data volume using LFSR reseeding with seed compression," in *Proc. International Test Conference* 2002, pp. 321-330.

[7] B. B. Bhattacharya, S. C. Seth, and S. Zhang, "Low-energy BIST design for scan-based logic circuits," in *Proc. 16th International Conference on VLSI Design* 2003, pp. 546-551.

[8] A. A. Al-Yamani, S. Mitra, and E. J. McCluskey, "BIST reseeding with very few seeds," in.*Proc. 21stVLSI Test Symposium, 2003* 2003, pp. 69-74.

[9] A. A. Al-Yamani and E. J. McCluskey, "Built-in reseeding for serial BIST," in *Proc. 21st VLSI Test Symposium* 2003, pp. 63-68.

[10] P. Wohl, J. A. Waicukauski, S. Patel, and M. B. Amin, "Efficient compression and application of deterministic patterns in a logic BIST architecture," in *Proc. ACM IEEE Design Automation Conference* 2003, pp. 566-569.

[11] S. Pateras and J. Rajski, "Cube-Contained Random Patterns and their Application to the Complete Testing of Synthesized Multi-le," in *Proc. InternationalTest Conference*, 1991, pp. 473-482.

[12] E. B. Eichelberger and E. Lindbloom, "Random-pattern coverage enhancement and diagnosis for LSSD logic self-test," *IBM Journal of Research and Development*, vol. 27 (3), pp. 265-272, 1983.

[13] I. Pomeranz and S. M. Reddy, "3-weight pseudo-random test generation based on a deterministic test set for combinational and sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 12 (7), pp. 1050-1058, 1993.

[14] H. Wunderlich, "On computing optimized input probabilities for random tests," in *Proc. 24th ACM/IEEE Design Automation Conference*, 1987, pp. 392-398.

[15] R. Kapur, S. Patil, T. J. Snethen, and T. W. Williams, "Design of an efficient weighted random pattern generation system," in *Proc. International Test Conference*, 1994, pp. 491-500.

[16] S. Wang, "Low hardware overhead scan based 3-weight weighted random BIST," in *Proc. International Test Conference* 2001, pp. 868-877.

[17] N. A. Touba and E. J. McCluskey, "Transformed pseudo-random patterns for BIST," in *Proc. 13th IEEE VLSI Test Symposium*, 1995, pp. 410-416.

[18] N. A. Touba and E. J. McCluskey, "Bit-fixing in pseudorandom sequences for scan BIST," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20 (4), pp. 545-555, 2001.

[19] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in *Proc. IEEE/ACM Inter. Conf. on Computer-Aided Design*, 1996, pp. 337-343.

[20] G. Kiefer and H.-J. Wunderlich, "Using BIST control for pattern generation," in *Proc. International Test Conference*, 1997, pp. 347-355.

[21] B. Reeb and H.-J. Wunderlich, "Deterministic pattern generation for weighted random pattern," in *Proc. European Design and Test Conference* 1996, pp. 30-36.

[22] J. A. Waicukauski, E. Lindbloom, E. B. Eichelberger, and O. P. Forlenza, "A method for generating weighted random test patterns," *IBM J. Res. Dev.*, vol. 33 (2), pp. 149-161, March, 1989.

[23] W. Li, C. Yu, S. M. Reddy, and I. Pomeranz, "A scan BIST generation method using a markov source and partial bit-fixing," in *Proc. ACM IEEE Design Automation Conference* 2003, pp. 554-559.

Proceedings of the 14th Asian Test Symposium (ATS ’05)
1081-7735/05 $20.00 © 2005 IEEE
IEEE COMPUTER SOCIETY