2009 27th IEEE VLSI Test Symposium

Restrict Encoding for Mixed–Mode BIST

Abdul-Wahid Hakmi, Stefan Holst,
Hans-Joachim Wunderlich

Institut für Technische Informatik
Universität Stuttgart
Pfaffenwaldring 47; 70569 Stuttgart, Germany
{hakmi, holst, wu}@iti.uni-stuttgart.de

Jürgen Schlöffel, Friedrich Hapke,
Andreas Glowatz

Mentor Graphics
Development (Deutschland) GmbH
Tempowerkring 1B; 21079 Hamburg, Germany
{Juergen_Schloeffel, Friedrich_Hapke,
Andreas_Glowatz}@mentor.com

***Abstract*— Programmable mixed–mode BIST schemes combine pseudo–random pattern testing and deterministic test. This paper presents a synthesis technique for a mixed–mode BIST scheme which is able to exploit the regularities of a deterministic test pattern set for minimizing the hardware overhead and memory requirements. The scheme saves more than 50% hardware costs compared with the best schemes known so far while complete programmability is still preserved.**

***Keywords*— Deterministic BIST**

I. Introduction

The benefits of built–in self–test strategies are widely recognized. They include reusability in the field, cost reduction for the automatic test equipment (ATE), high coverage of non–target faults and IP protection among others [1].

If a BIST scheme like STUMPS (Self–Test Using MISR and Parallel Shift register sequence generator [2]) is employed, the impact of BIST on the design cost is limited as it is just added externally to the scan chains. Figure 1 shows the basic principle of the STUMPS scheme. The ATE controls the test session and reads back the outcome. The scan chains are loaded during multiple scan clock cycles and with each clock, the shift register SR produces a *vector* of k bits which are shifted into the chains. With t being the maximum scan chain length, t vectors are loaded into the chains to form a complete test pattern. After a system clock cycle, the scan flip–flops have captured the response bits of the core–under–test (CUT) which are unloaded into the MISR with the next t scan clock cycles.

The major drawback of this BIST scheme lies in the limitation of the obtainable fault coverage which can be restored by different means:

- Improving the random pattern testability of the core–under–test (CUT) by inserting test points [3], [4], [5], [6]. This introduces significant design effort, additional validation tasks and may impact circuit performance.

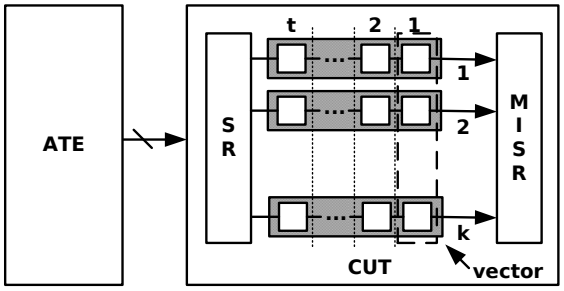

Fig. 1. STUMPS scheme.

- Modifying the pattern generator in order to generate precomputed deterministic patterns [7], [8], [9]. The modified pattern generator may require significant area, and design flow and flexibility are affected, too.
- Reseeding the LFSR and its modifications [10], [11], [12], [13], [14], [15], [16], [17], [18]. The seeds are stored in memory, and this scheme is programmable and rather flexible. However, even the most efficient scheme up to now [19] may require a substantial amount of memory.

Generating a precomputed pattern sequence by reseeding has maximum encoding efficiency of 1, which is the number of encoded care bits divided by the required amount of storage bits [20].

Dictionary based approaches [21], [22], [23], [24] do not have this limitation. Here, some vectors are stored in memory and reused during test. Test vectors and test patterns show regularities which can be exploited by dictionary based approaches rather than by LFSRs for two main reasons:

1) Many vectors are compatible to each other and can be encoded by a small set of dictionary vectors.
2) Some input assignments appear repeatedly in many patterns at the same positions. This is since certain circuit parts get sensitized by these assignments, and multiple patterns that test the faults in these parts keep

1093-0167/09 $25.00 © 2009 IEEE
DOI 10.1109/VTS.2009.43

these assignments constant.

The method presented below identifies a set of useful dictionary entries by solving a clique covering problem. After this, the pattern set is reordered so that the dictionary entries are used in a regular way by solving a traveling salesman problem. If vectors and patterns do not show regularities at some positions, standard reseeding is applied.

The next section gives an overview of the hardware BIST scheme. Section III shows how to identify the dictionary entries, and section IV introduces the pattern ordering for maximizing regularity. In section V, a test program is synthesized, and section VI shows, that coding efficiency is increased and memory requirement is reduced nearly by a factor of 2 with respect to the best schemes known so far for industrial circuits.

II. Overview

The proposed test compression approach combines partial reseeding of an LFSR with a small dictionary of previously calculated vector values. Figure 2 shows the basic structure containing an LFSR, a phase shifter (PS), the dictionary ROM and a multiplexer and k scan chains. The LFSR is provided with seed information and the ROM is addressed by a bits. For each scan vector, either the output of the LFSR, or a dictionary value from the ROM is selected using the *Slct*–signal.

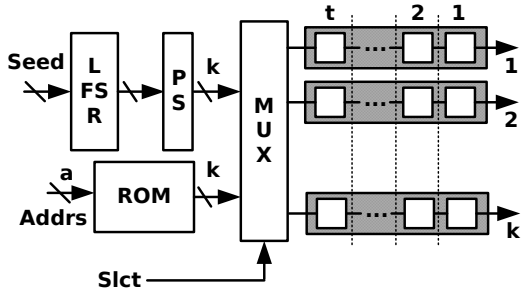

Fig. 2. Combination of LFSR reseeding and a dictionary.

The ROM–addresses and the selection signal for each scan vector are read from a register file which holds the current states of the vectors of one pattern (see figure 3). Each of the t registers is $a + 1$ bits long and a register is updated with new information if the write–enable signal *Wen* is 1. By addressing the status registers with a modulo t counter, the same sequence of addresses and selection signals is generated for each pattern. Therefore, dictionary vectors are continuously injected at the same vector position over multiple patterns. The injection starts and ends by updating the values of the status register accordingly. One continuous sequence of injections at a certain vector position is called a *restrict*. The value or dictionary vector that is injected at this position is called *restrict vector* .

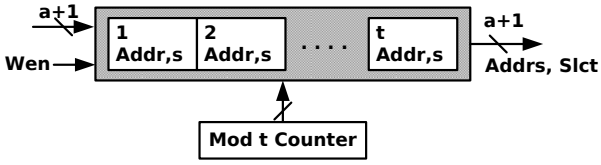

Fig. 3. Status register for generating restrict information.

Figure 4 shows the first 4 patterns of a pattern set. Each pattern consists of 4 vectors. The first two restricts start with the first pattern and continuously inject restrict vectors v_b and v_a into the vector positions 3 and 4 respectively. The third restrict starts at pattern p_2 replacing vector position 1 with the restrict vector v_a. The care-bits in unrestricted vector positions $(v_{11}, v_{12}, \dots)$ are encoded by LFSR–reseeding.

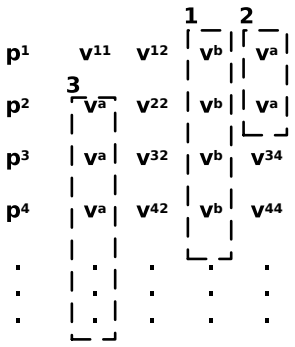

Fig. 4. Start of a pattern application with three restricts.

The amount of information needed to define a restrict is constant and does not depend on the amount of care bits encoded this way. In contrast, the cost of LFSR–reseeding grows linearly in the number of encoded care bits. Maximizing the gain for restrict encoding leads to an optimization problem with the following goals:

- Construct a small dictionary with restrict vector values that can be injected as often as possible.
- Maximize the runlengths of the restricts so that many care bits are covered with constant cost.

For easier discussion, let us first define some more notations.

Let v be a *vector* which contains care and don't care bits. The number of care bits in vector v is noted as $|v|$. Two vectors v_i, v_j are *compatible* ($v_i \sim v_j$), if there is no conflicting care bit at any position. Compatible vectors can be *merged* $v = v_i + v_j$ so that v carries all the care bits of both v_i and v_j. If a vector set V contains only pairwise compatible vectors, the whole set can be merged into one vector which is noted as $v = \Sigma V$.

A *pattern* $p = (v_1, v_2, \dots, v_t)$ is a sequence of vectors. The vector at position i of pattern p is noted as $p(i) = v_i$. Let P be a pattern set with $|P| = q$. The *vector set* V_P of the pattern

set P is the set of all vectors present in $p_1, \ldots, p_q \in P$. The *cardinality* of a vector v with respect to P is $c_P(v) = n$ with n being the number of vectors in $p_1, \ldots, p_q \in P$ that are equal to v, q is the total number of patterns. The *vector slice set* V_i of the pattern set P and vector position $1 \leq i \leq t$ is equal to $\{p(i)|p \in P\}$.

The *compatibility graph* $G(E, V_P, w)$ of a vector set V_P is a node–weighted graph with

$$(v_i, v_j) \in E \Leftrightarrow v_i \sim v_j \qquad \forall v_i, v_j \in V_P$$

and the weight of a node v_i:

$$w(v_i) = c_P(v_i) \cdot |v_i| \qquad \forall v_i \in V_P$$

III. Generation of the Restrict Vector Dictionary

The goal is to find a minimum set RV of fully specified restrict vectors, so that a large number of care bits can be encoded by replacing the original vectors by compatible restrict vectors. More formally, the sum over the weights of vectors restrictable with a set RV

$$\sum_{v \in V_P} w(v)[\exists v_r \in RV \text{ with } v \sim v_r]$$

should be large.

The final restrict vector set is determined in a two-step process. The first step constructs a candidate set RV_c of restrict vectors. The second step selects the final restrict vectors $RV \subseteq RV_c$ as described in the next section.

RV_c is constructed by partitioning the compatibility graph $G(E, V_P, w)$ into cliques. Each clique contains pairwise compatible vectors which are merged into a single restrict vector candidate $v_r \in RV_c$.

The clique partitioning problem has exponential complexity and the following simple heuristic is used:

COMPUTE-RESTRICTS:

1) Let $RV_c = \emptyset$.
2) C=FIND-NEXT-CLIQUE.
3) Let $V_P = V_P - C$ and add $v_r = \Sigma C$ to RV_c.
4) Unless $V_P = \emptyset$ goto step 2.
5) Return RV_c.

FIND-NEXT-CLIQUE:

1) Choose a $v \in V_P$ with $w(v)$ maximum, let $C = \{v\}$.
2) Let $N = \{v'|v' \in V_P - C, (v, v') \in E \ \forall v \in C\}$ be the set of all common neighbors. Unless $N = \emptyset$, add v' with largest $w(v')$ to C and repeat step 2.
3) Return C.

After clique partitioning, every $v \in V_P$ is compatible with at least one of the $v_r \in RV_c$. The next section will use some of these candidates for restricting and therefore determine the subset of RV_c, that has to be stored in a ROM.

IV. Test Set Ordering

The patterns are ordered so that the candidate restrict vectors in RV_c are able to cover large amounts of care bits in single runs. Then, the set of restricts are determined by scanning through the ordered pattern list. This is similar to static test pattern compaction [25], [26], [27] where multiple compatible patterns are merged to shorten the test set. In this approach, multiple compatible vectors are jointly encoded with minimum information to improve coding efficiency.

A gain function expresses the benefit of sorting two patterns adjacent to each other. This benefit increases with the number of compatible restrict vectors shared by these two patterns. However, each vector v can be compatible to multiple restrict vector candidates $v_r \in RV_c$ and it is not yet clear, which restrict vector is likely to be used at the end. The following procedure determines for each vector v a single restrict value v_r that is most capable in restricting large amounts of care bits.

Each vector position $1 \leq i \leq t$ is considered separately. The weight of a restrict vector with respect to a vector position i is defined as the overall weight of their compatible vectors in V_i:

$$w_i(v_r) = \sum_{v \in V_i} w(v)[v_r \sim v]$$

The more care bits a certain restrict vector can cover at position i, the higher is the weight. The restrict vectors with higher weight should have priority over restrict vectors with lower weight because it is more likely, that they are able to cover more care bits in longer runs. Thus, each vector $v \in V_i$ is associated with the compatible restrict vector v_r of highest weight $w_i(v_r)$. This single restrict vector associated with v is noted as $r(v)$.

Each pattern now has t restrict vectors associated with it; one at each vector position. The benefit gained by putting two patterns p_m, p_n in the pattern set adjacent to each other depends on the following factors:

- The number of vector positions for which these two patterns share the same restrict vector ($r(p_n(i)) = r(p_m(i))$).
- The amount of care bits covered by these common restrict vectors.

Based on these factors, the *similarity* of a pair of patterns $p_m, p_n \in P$ is defined as:

$$s(p_m, p_n) = \sum_{i=1}^{t} (|p_m(i)| + |p_n(i)|) \cdot y(p_m(i), p_n(i)) \quad \text{with}$$

$$y(v_j, v_k) = \begin{cases} 2 & \text{if } |v_j| \cdot |v_k| > 0 \text{ and } r(v_j) = r(v_k), \\ 1 & \text{if } |v_j| \cdot |v_k| = 0, \\ -1 & \text{otherwise.} \end{cases}$$

The first term weights the outcome of y by the number of care bits at each position. y is positive only if the two vectors are compatible. It evaluates to 2, if both vectors have care bits and are associated with the same restrict vector.

For a given pattern set P and with the metric defined above, a similarity graph $S(P, s)$ is constructed. This graph is edge weighted, undirected, and complete. Each node represents a pattern and the edges between the patterns are weighted with the similarity of the two adjacent nodes. Figure 5 shows the graph S for ordering the patterns in fig. 4. For sake of simplicity, each vector has exactly one care bit and the vectors v_{11} through v_{44} unrestricted in fig. 4 have all different restrict vectors associated with them. The similarity between p_1 and p_2 is $s(p_1, p_2) = -2 - 2 + 4 + 4 = 4$ since the first two vector positions have different restrict values and the latter two positions have matching restrict values. The patterns p_1 and p_4 have lower similarity $s(p_1, p_2) = -2-2+4-2 = -2$ because only one vector position show matching restrict values.

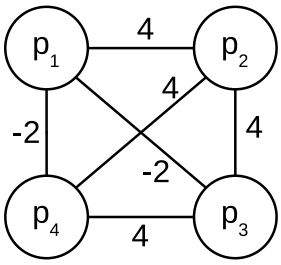

Fig. 5. A similarity graph.

The patterns are ordered by finding a maximum weight path in S such that every node is visited only once. It can be easily verified, that all maximum weight paths in figure 5 $(p_1, p_2, p_3, p_4), (p_1, p_2, p_4, p_3), (p_3, p_4, p_2, p_1), (p_4, p_3, p_2, p_1)$ preserve the restricts shown in figure 4, while other paths like (p_1, p_3, p_2, p_4) result in less care bits to be covered by restricts.

The graph traversal described above is an instance of the travelling salesman problem, so the best solution cannot be found in polynomial time. The following greedy heuristic however, provides sufficient results. For the starting point of this path we use a special pattern p_0. This pattern has by definition no care bits and no compatible restrict vectors.

1) Let $L = (p_0)$ be a pattern list and $p = p_0$.
2) Find a $p' \in P$ with $s(p, p')$ maximum. Let $p = p'$, $P = P - \{p\}$ and append p to list L.
3) Unless $P = \emptyset$, goto step 2
4) Return L.

Now, a set R_c of *restrict candidates* is generated based on the ordering. From now on, we consider P to be the sorted pattern list L. A restrict candidate $r = (s, e, i, v_r) \in R_c$ is a four tuple consisting of a starting index s, an ending index e, a vector position i and the restrict vector v_r. The following procedure scans through the list of vectors at a certain position $1 \leq i \leq t$ and generates restrict candidates in a greedy manner. The candidates are determined by successively computing the intersections of the sets of restrict values compatible to the considered vectors.

1) Let starting index $s = 1$.
2) Let ending index $e = s$.
3) Let T contain all the restrict values compatible with the ith vector in the sth pattern: $T = \{v_r \in RV_c | p_s(i) \sim v_r\}$.
4) Intersect T with the restrict values compatible with the ith vector of the next pattern: $T' = T \cap \{v_r \in RV_c | p_{e+1}(i) \sim v_r\}$
5) If $T' \neq \emptyset$ then $T = T'$, $e = e + 1$ and goto step 4.
6) Now, T holds the last non–empty intersection. Add the restrict $(s, e, i, v_r \in T)$ to R_c.
7) Set $s = e + 1$ and if $s < |P|$, goto step 3.
8) Return R_c.

The cost of a restrict is constant. Let c be the number of bits to be stored for one restrict and l the encoding efficiency of the LFSR. Only those restricts from R_c are selected that provide a gain in test data storage: $R = \{r \in R_c | c \cdot l < |r|\}$ with $|r|$ being the number of care bits covered by the restrict r.

This final set of restricts R addresses only a subset of the restrict vector candidates RV_c. This subset is now the final set of restrict vectors $RV \subseteq RV_c$ which has to be stored in the dictionary. The size of this dictionary is

$$dcost = |RV| \cdot k,$$

and the status register, which has to store the t addresses has the size of

$$scost = \lceil log_2(|RV| + 1) \rceil \cdot t.$$

The overall gain is then estimated as the sum over the gains of the restricts minus the costs mentioned above:

$$\sum_{r \in R} (|r| - c \cdot l) - dcost - scost$$

V. Test Program Encoding and Decompression

The compressed test data consists of a sequence of actions. An action is either an update for the status register (fig. 3) or a reseeding of the LFSR. Each action needs to be performed at a certain shift cycle. For an update of the restrict information in the status register, this shift cycle is determined by the pattern indices and the vector positions of the associated restrict in R. For an LFSR seed update, this shift cycle is determined by the runlengths of the previous seeds.

design	k	t	q	care bits
p35k	23	127	8506	1213293
p45k	97	333	8791	210789
p77k	13	305	5883	199670
p81k	8	504	30706	1534646
p89k	18	963	13514	541166
p100k	18	792	3437	72541
p141k	24	486	10984	483332
p239k	40	541	6034	156695
p259k	40	541	8482	216530
p267k	45	494	14123	594530
p269k	45	494	14615	603409
p279k	55	416	17575	561088
p286k	55	416	25645	794879
p330k	64	317	17322	863544
p388k	50	546	14109	454866
p418k	64	831	33704	1016554
p483k	71	900	22168	441510

TABLE I

SCAN CHAIN CONFIGURATION AND PATTERN SET CHARACTERISTICS

Each action corresponds to a command in a test program. Each command consists of two parts: The action to be performed and the number of shift cycles to the next command, called *delay*. The delay determines the control flow and a simple state machine can be used for performing the actions by executing the test program.

For example, the command that sets restrict 3 in figure 4 is issued in the 5th shift clock cycle. In this cycle, the modulo t counter that addresses the status register is 0 and the command updates the ROM–address for the first vector position to v_a. The delay part of this command is 7, since this is the number of shift cycles to the next update (release of restrict 2 in pattern p_3, vector position 4).

VI. EXPERIMENTAL RESULTS

Experiments were performed on industrial designs provided by NXP.

Table I shows the characteristics of the designs and their pattern sets. The name of the design itself corresponds to the number of logic gates of the circuit. Column k shows the number of scan chains in the design, the maximum length of a chain is denoted in column t. Deterministic patterns are generated for random pattern resistant faults by a commercial ATPG. The compaction effort of the ATPG was set to low to provide enough space for random fill in this mixed-mode BIST scheme. The number of patterns is shown in column q followed by the total number of care bits in the pattern set.

The pattern sets are now encoded with the proposed approach. The results are shown in table II.

The first part of the table deals with the restricts only. The first column shows the percentage of care bits covered by dictionary vectors. Then, the total number of restricts are given followed by the total number of commands used for these restricts. Column *tpcost* shows the amount of storage spend on these commands in bits. Column *dcost* shows the size of the dictionary in bits. The size of the dictionary is very small compared to the size of the test program. The next column *scost* reports the size of status register in bits. The coding efficiency for the care bits covered by restricts is given in column *ceff.*. All three costs are considered while computing the encoding efficiencies. Over half of the care bits in the test set can be encoded with efficiencies of 1.5 up to 3.27. These efficiencies are higher than any efficiency achievable by a linear decompressor.

The second part of the table shows the results for encoding the remaining care bits with reseeding. For reseeding, a 128bit LFSR was used in combination with a randomly generated phase shifter. Here the column *cost* shows the amount of seed information stored to encode the unrestricted care bits. This seed information is comprised of the seed bits themselves and the additional delay information in each seeding command.

	restricts							reseeding			total
design	%cb	#restr.	#comm.	tpcost	dcost	scost	ceff.	%cb	cost	ceff.	ceff.
p35k	72	9283	16226	326922	10649	1778	2.60	28	466186	0.70	1.51
p45k	68	1354	2438	48199	2231	3663	2.66	32	76270	0.88	1.62
p77k	74	1225	2269	42231	91	3355	3.27	26	58176	0.86	1.92
p81k	79	17505	32957	579811	64	7560	2.08	21	349635	0.88	1.64
p89k	68	3699	6952	143166	108	11556	2.39	32	197100	0.86	1.54
p100k	58	442	803	16564	108	7128	1.78	42	34220	0.88	1.25
p141k	71	3441	6325	127596	312	5832	2.60	29	154944	0.87	1.67
p239k	46	1218	2290	42567	280	5951	1.50	54	94464	0.88	1.09
p259k	50	1726	3223	59952	320	5951	1.65	50	122400	0.87	1.15
p267k	73	5177	9394	189782	405	6422	2.22	27	187775	0.84	1.55
p269k	74	5612	10127	188978	360	6422	2.31	26	182700	0.83	1.59
p279k	66	4839	8803	168990	825	5408	2.14	34	211104	0.87	1.45
p286k	69	6268	11338	224069	990	5408	2.38	31	280080	0.87	1.56
p330k	71	9802	18391	333448	704	4438	1.84	29	273416	0.88	1.41
p388k	62	2984	5495	110828	450	6552	2.43	38	195696	0.86	1.45
p418k	75	5990	10969	226362	512	10803	3.24	25	280430	0.88	1.96
p483k	68	2347	3899	81103	426	10800	3.27	32	165590	0.84	1.71

TABLE II

RESTRICT ENCODING RESULTS

Method	Average Efficiency	Maximum Efficiency
Continuous reseeding	0.47 [18]	$\leq$1.00 [20]
Programmable D-BIST	0.80 [19]	1.19 [19]
Proposed Scheme	1.53	1.96

TABLE III

COMPARISON TO PREVIOUS METHODS

As expected, coding efficiency for reseeding alone is below 1. But less than half of all care bits in the test set need to be encoded in this way.

The overall coding efficiency remains greater than 1 in all cases, as shown in the very last column.

Table III compares the average coding efficiencies of continuous reseeding[18], programmable deterministic BIST [19] and the proposed mixed-mode BIST scheme. As these experiments were dealing with different sets of industrial circuits only average values can be reported here. Depending on the phase shifter configuration, the efficiency of continuous reseeding [18] can be higher than 0.47, but it can never exceed 1.0 [20]. In [19] efficiencies up to 1.19 were shown, and for the proposed scheme we observe an improvement by a factor of two compared to previous approaches.

VII. CONCLUSIONS

A new synthesis technique for a mixed–mode BIST scheme was presented which is able to exploit the regularities of a deterministic test pattern set for minimizing the hardware overhead and memory requirements. The scheme saves more than 50% hardware costs compared with the best schemes known so far while complete programmability is still preserved.

VIII. ACKNOWLEDGEMENT

This work has been supported by the BMBF within the project MAYA (Project ID 01M3172). The authors would like to thank Melanie Elm for her help with implementing the reseeding part.

REFERENCES

[1] H.-J. Wunderlich, "Bist for systems-on-a-chip," *Integration, the VLSI Journal*, vol. 26, no. 1-2, pp. 55–78, December 1998.

[2] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudorandom Techniques*. John Wiley & Sons, 1987.

[3] J. Hayes and A. Friedman, "Test point placement to simplify fault detection," *IEEE Transactions on Computers*, vol. C-33, pp. 727–735, July 1974.

[4] B. Seiss and P. Trouborst, "Test point insertion for scan-based bist," in *IEEE European Test Conference (ETC)*, April 1991, pp. 253–262.

[5] M. J. Geuzebroek, J. T. van der Linden, and A. J. van de Goor, "Test point insertion for compact test sets," in *IEEE International Test Conference (ITC)*, 2000, pp. 506–514.

[6] H. Vranken, F. Meister, and H. J. Wunderlich, "Combining deterministic logic bist with test point insertion," in *IEEE European Test Workshop (ETW)*, May 2002, pp. 389–394.

[7] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in *International Conference on Computer-Aided Design (ICCAD), November 10-14, 1996, San Jose, CA*, 1996, pp. 337–343.

[8] V. Gherman, H.-J. Wunderlich, H. P. E. Vranken, F. Hapke, M. Wittke, and M. Garbers, "Efficient pattern mapping for deterministic logic BIST," in *Proceedings 2004 International Test Conference (ITC 2004), October 26-28, 2004, Charlotte, NC, USA*, 2004, pp. 48–56.

[9] N. A. Touba and E. J. McCluskey, "Altering a pseudo-random bit sequence for scan-based BIST," in *Proceedings IEEE International Test Conference 1996, Washington, DC, USA, October 20-25, 1996*, 1996, pp. 167–175.

[10] B. Koenemann, "LFSR-coded test patterns for scan designs," in *Proceedings of the European Test Conference, Munich, Germany*, 1991, pp. 237–242.

[11] 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 Trans. Computers*, vol. 44, no. 2, pp. 223–233, 1995.

[12] J. Rajski, J. Tyszer, and N. Zacharia, "Test data decompression for multiple scan designs with boundary scan," *IEEE Trans. Computers*, vol. 47, no. 11, pp. 1188–1200, 1998.

[13] C. V. Krishna, A. Jas, and N. A. Touba, "Test vector encoding using partial LFSR reseeding," in *Proceedings IEEE International Test Conference 2001, Baltimore, MD, USA, 30 October - 1 November 2001*, 2001, pp. 885–893.

[14] A. A. Al-Yamani, S. Mitra, and E. J. McCluskey, "BIST reseeding with very few seeds," in *21st IEEE VLSI Test Symposium (VTS 2003), 27 April - 1 May 2003, Napa Valley, CA, USA*, 2003, pp. 69–76.

[15] W. Rao, I. Bayraktaroglu, and A. Orailoglu, "Test application time and volume compression through seed overlapping," in *Proceedings of the 40th Design Automation Conference, DAC 2003, Anaheim, CA, USA, June 2-6, 2003*, 2003, pp. 732–737.

[16] C. V. Krishna and N. A. Touba, "3-stage variable length continuous-flow scan vector decompression scheme," in *22nd IEEE VLSI Test Symposium (VTS 2004), 25-29 April 2004, Napa Valley, CA, USA*, 2004, pp. 79–86.

[17] B. Koenemann, C. Barnhart, B. Keller, T. Snethen, O. Farnsworth, and D. Wheater, "A SmartBIST variant with guaranteed encoding," in *Proceedings 10th Asian Test Symposium*, 2001, pp. 325–330.

[18] E. H. Volkerink and S. Mitra, "Efficient seed utilization for reseeding based compression," in *21st IEEE VLSI Test Symposium (VTS 2003), 27 April - 1 May 2003, Napa Valley, CA, USA*, 2003, pp. 232–240.

[19] A.-W. Hakmi, H.-J. Wunderlich, C. G. Zoellin, A. Glowatz, F. Hapke, J. Schloeffel, and L. Souef, "Programmable deterministic built-in self-test." in *Proceedings of IEEE International Test Conference 2007 (ITC 2007) , San-Jose, CA, USA, October 22-24*, 2007.

[20] K. J. Balakrishan and N. A. Touba, "Relating entropy theory to test data compression," in *Proceedings IEEE European Test Symposium*, 2004, pp. 187–192.

[21] L. Li, K. Chakrabarty, and N. A. Toube, "Test data compression using dictionaries with selective entries and fixed-length indices," *ACM Transactions on Design Automation of Electronic Systems*, vol. 8, no. 4, pp. 470–490, October 2003.

[22] A. Würtenberger, C. S. Tautermann, and S. Hellebrand, "Data compression for multiple scan chains using dictionaries with corrections," in *IEEE International Test Conference (ITC)*, 2004, pp. 926–935.

[23] E. H. Volkerink, A. Khoche, and S. Mitra, "Packet-based input test data compression techniques," in *IEEE International Test Conference (ITC)*, 2002, pp. 154–163.

[24] M. Tehranipoor, M. Nourani, and K. Chakrabarty, "Nine-coded compression technique for testing embedded cores in socs," *IEEE Transactions on VLSI Systems*, vol. 13, no. 6, pp. 719–731, June 2005.

[25] S. Kajihara, I. Pomeranz, K. Kinoshita, and S. Reddy, "On compacting test sets by addition and removal of test vectors," in *VLSI Test Symposium, 1994. Proceedings., 12th IEEE*, Apr 1994, pp. 202–207.

[26] I. Hamzaoglu and J. H. Patel, "Test set compaction algorithms for combinational circuits," in *ICCAD '98: Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*. New York, NY, USA: ACM, 1998, pp. 283–289.

[27] X. Lin, J. Rajski, I. Pomeranz, and S. Reddy, "On static test compaction and test pattern ordering for scan designs," in *Test Conference, 2001. Proceedings. International*, 2001, pp. 1088–1097.