which means that x^{a_1}, x^{a_2}, x^{a_k} are linearly independent $\bmod \tilde{P}(x)$ as specified by Lemma 1.1 in the form given by [13], since in the LFSR/SR structure the zeroth cell of the LFSR part is indexed with 0.

2) General LFSM and groups of cells with indexes greater than $d-1$. In this case set $A = \{a_1, a_2, \ldots, a_k\}$ is such that $a_i \geq d$, $1 \leq i \leq k$, that is, no cell lies within the LFSM part of the LFSM/SR. This case is important in TPG since the SR part of an LFSM/SR is many times a scan chain which is responsible for providing all test-phase inputs (primary outputs and internal flip–flops, plus any additional internal test input points) with test values, while the LFSM part is responsible for providing with values only the SR part (and not directly any proper circuit inputs). Then all $\lambda(\;)$ values are the same with $\lambda(a_i) = d-1$, and so all $\mathcal{D}_x$ sets are also the same with $\mathcal{D}_{\lambda(a_i)} = \mathcal{D}_{d-1}$. The $\delta(\;)$ values are $\delta(a_i) = a_{\max} - a_i = a_1 - a_i$. So the formula now becomes $\sum_{a_i \in A} \sum_{j \in D_{\lambda(a_i)}} x^{d-1-j+\lambda(a_i)} = (x^{\lambda(a_1)} + x^{\lambda(a_2)} + \cdots + x^{\lambda(a_k)}) \sum_{j \in \mathcal{D}_{d-1}} x^{d-1-j} = (x^{a_1-a_1} + x^{a_1-a_2} + \cdots + x^{a_1-a_k}) \sum_{j \in \mathcal{D}_{d-1}} x^{d-1-j} = 0 \bmod P(x)$. But since the second term in the above product is of degree less than d and $P(x)$ is an irreducible polynomial, $P(x)$ must divide the first term $x^{a_1-a_1} + x^{a_1-a_2} + \cdots + x^{a_1-a_k}$. But as shown in the case above, this is equivalent to $x^{a_1-a_k} + \cdots + x^{a_j-a_k} + x^{a_{j+1}-a_k} + \cdots + x^{a_k-a_k}$ being divisible by $\tilde{P}(x)$, which is the condition of Lemma 1.1. That is, in this case, all LFSMs with the same characteristic polynomial $P(x)$ have the same behavior as a Type-1 LFSR with characteristic polynomial $P(x)$ (as expected, since every cell j after the $(d-1)$th receives just a shift-by-j version of the characteristic sequence of the LFSM).

IV. Conclusion

We investigated the linear dependencies in an LFSM/SR TPG for an arbitrary LFSM. We obtained a formula that relates the linear dependencies with the characteristic polynomial of the LFSM and that can be computed in very fast polynomial time. The formula, which generalizes a previously known formula for Type-1 LFSRs only, allows the fast determination of linear dependencies in an LFSM/SR structure for any LFSM, including in particular Type-2 LFSRs and CA.

References

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

[2] W.-B. Jone and C. A. Papachristou, "A coordinated approach to partitioning and test pattern generation for pseudoexhaustive testing," in *26th ACM/IEEE Design Automation Conf.*, 1989, pp. 525–530.

[3] H.-J. Wunderlich and S. Hellebrand, "Tools and devices supporting the pseudo-exhaustive test," in *Proc. European Design Automation Conf.*, 1990.

[4] D. Kagaris, F. Makedon, and S. Tragoudas, "A method for pseudo-exhaustive test pattern generation," *IEEE Trans. Computer-Aided Design*, vol. 13, pp. 1170–1178, Sept. 1994.

[5] R. Srinivasan, S. K. Gupta, and M. A. Breuer, "Novel test pattern generators for pseudoexhaustive testing," in *Proc. Int. Test Conf.*, 1993, pp. 17–21.

[6] E. R. Berlekamp, *Algebraic Coding Theory*. Laguna Hills, CA: Aegean Park, 1984.

[7] C. L. Chen, "Linear dependencies in linear feedback shift registers," *IEEE Trans. Computers*, vol. 35, pp. 1086–1088, Dec 1986.

[8] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-in Test for VLSI*. New York: Wiley, 1987.

[9] P. H. Bardell, "Calculating the effects of linear dependencies in m-sequences as test stimuli," *IEEE Trans. Computer-Aided Design*, vol. 11, pp. 83–85, Jan. 1992.

[10] D. Kagaris and S. Tragoudas, "Avoiding linear dependencies in LFSR test pattern generators," *J. Electron. Testing: Theory Applicat.*, vol. 6, pp. 229–241, 1995.

[11] J. Rajski and J. Tyszer, "On linear dependencies in subspaces of LFSR-generated sequences," *IEEE Trans. Computers*, vol. 45, pp. 1212–1221, Oct. 1996.

[12] Z. Barzilai, D. Coppersmith, and A. L. Rosenberg, "Exhaustive bit generation with application to VLSI self-testing," *IEEE Trans. Comput.*, vol. C-32, pp. 190–194, 1983.

[13] D. T. Tang and C. L. Chen, "Logic test pattern generation using linear codes," *IEEE Trans. Comput.*, vol. 33, pp. 845–850, Sept. 1984.

[14] A. Lempel and M. Cohn, "Design of universal test sequences for VLSI," *IEEE Trans. Inform. Theory*, vol. 31, pp. 10–15, Jan. 1985.

[15] P. H. Bardell, "Design considerations for parallel pseudorandom pattern generators," *J. Electron. Testing: Theory Applicat.*, vol. 3, pp. 73–87, Jan. 1990.

[16] J. Rajski, N. Tamarapalli, and J. Tyszer, "Automated synthesis of phase shifters for built-in self-test applications," *IEEE Trans. Computer-Aided Design*, vol. 19, pp. 1175–1188, Oct. 2000.

[17] M. Serra, T. Slater, J. C. Muzio, and D. M. Miller, "The analysis of one-dimensional linear cellular automata and their aliasing properties," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 767–778, July 1990.

[18] C. Cattell and J. C. Muzio, "Synthesis of one-dimensional linear hybrid cellular automata," *IEEE Trans. Computer-Aided Design*, vol. 15, pp. 325–335, Mar. 1996.

[19] V. D. Agrawal, C. R. Kime, and K. K. Saluja, "A tutorial on built-in self-test. part 1: Principles," *IEEE Design Test Computers*, pp. 73–82, Mar. 1993.

[20] H. S. Stone, *Discrete Mathematical Structures and Their Applications*. Chicago, IL: Science Res. Assoc., 1973.

A New Built-In TPG Method for Circuits With Random Pattern Resistant Faults

Xrysovalantis Kavousianos, Dimitris Bakalis, Dimitris Nikolos, and Spyros Tragoudas

***Abstract*—The partition of the inputs of a circuit under test (CUT) into groups of compatible inputs reduces the size of a test pattern generator and the length of the test sequence for built-in self-test (BIST) applications. In this paper, a new test-per-clock BIST scheme is proposed which is based on multiple input partitions. The test session consists of two or more phases, and a new grouping is applied during each test phase. Using the proposed method a CUT can be tested at-speed and complete fault coverage (100%) is achieved with a small number of test vectors and small area overhead. Our experiments show that the proposed technique compares favorably to the already known techniques.**

***Index Terms*—Built-in self-test, test pattern generators.**

I. Introduction

Built-in self-test (BIST) [1]–[5] is an important promising technique for testing large and complex systems. Minimal test application time,

Manuscript received February 7, 2001; revised October 8, 2001. This work was supported in part by the Computer Technology Institute. This paper was recommended by Associate Editor K.-T. Cheng.

X. Kavousianos, D. Bakalis, and D. Nikolos are with the Computer Engineering and Informatics Department, University of Patras, Patras 26500, Greece (e-mail: kabousia@ceid.upatras.gr; bakalis@cti.gr; nikolosd@cti.gr).

S. Tragoudas is with the Electrical and Computer Engineering Department, Southern Illinois University, Carbondale, IL 62901 USA (e-mail: spyros@engr.siu.edu).

Publisher Item Identifier S 0278-0070(02)05623-3.

area overhead, and test data storage as well as minimal performance degradation and at-speed testing is essential in many BIST applications. In many applications 100% fault coverage is also desirable.

BIST schemes can be classified into two general categories [4], [5]: test-per-scan and test-per-clock. In the test-per-scan scheme a complete or partial scan path is serially filled by the test pattern generator (TPG) [6]–[12], whereas in the test-per-clock scheme a new test vector is applied to the circuit under test (CUT) at each clock cycle [12]–[24]. In this paper, we consider only test-per-clock BIST schemes.

BIST schemes are also classified according to the type of patterns they generate. Pseudorandom BIST schemes [2] have the advantage of very low hardware overhead. However, for circuits with random pattern resistant faults, high fault coverage cannot be achieved within an acceptable test length. Test point insertion, weighted pseudorandom patterns, CUT inputs reduction, and mixed-mode BIST are the main techniques which have been proposed to solve this problem. Although test point insertion methods [13] reduce the test length, they increase the hardware overhead and may also pose additional delays on the critical paths of the system slowing down its performance. Weighted pseudorandom pattern BIST schemes which require very small hardware overhead have been proposed in [14] and [15]. The method of [15] has the additional advantage that it achieves almost complete fault coverage with relatively short test sequences. However, [14] and [15] cannot be easily used for at-speed testing, because some multiplexers should be inserted between the linear feedback shift register (LFSR) outputs and the CUT inputs. Input reduction based only on compatibility analysis of the CUT inputs [16], [17] has the advantage of very small hardware overhead. However, in most cases the test sequence length is very long. In order to reduce the test sequence length, the methods proposed in [18] and [19] insert logic between the TPG outputs and the CUT inputs, which makes at-speed testing difficult. Mixed-mode BIST schemes [12], [20]–[23] impose in the pseudorandom sequence deterministic test vectors for detecting the random pattern resistant faults. The methods proposed in [20] and [22] insert logic between the TPG outputs and the CUT inputs making at-speed testing difficult.

In a test-per-clock BIST scheme, a register must be modified in order to operate as a parallel in—parallel out register during normal mode and as a shift register, LFSR, or two-port register during test mode. This modification may cause a delay in the normal operation of the circuit, if the register is in the critical path. This is the minimal performance degradation that a test-per-clock BIST TPG may cause in the normal operation of the circuit. Since the methods proposed in [14], [15], [18]–[20], and [22] insert logic between the TPG outputs and the CUT inputs, they may further affect the performance of the system in normal operation.

In this paper, we propose a new TPG scheme that ensures: 100% fault coverage, low hardware overhead, short test length, and at-speed testing because it does not insert logic between the input flip–flops and the CUT and minimal performance degradation. The proposed scheme is based on multiple partitions of the CUT inputs into groups. A test session consists of two or more phases and during each phase a new grouping of the inputs is used. The number of groups is the same for all phases. The majority of single stuck-at faults is detected by applying the first grouping. New groupings are repeatedly applied, until the fault coverage reaches 100%. The procedure for the first partition of inputs into groups takes into account all the stuck-at faults, while for the remaining groupings only undetected faults are considered. Our goals, small test set length and low area overhead, are obtained using two grouping algorithms. The first algorithm selects the first grouping of the inputs of the CUT using pseudorandom vectors, while the second one selects the next groupings using pseudorandom test patterns for the easy-to-detect faults and deterministic for the rest.

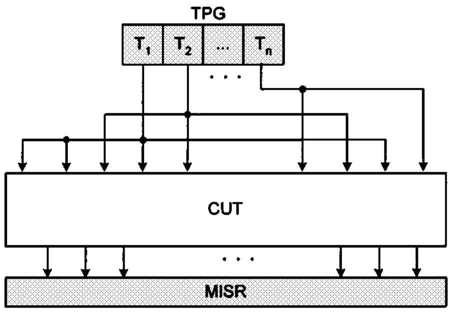

Fig. 1. TPG with single group assignment.

The paper is organized as follows: Section II presents the architecture of the TPG scheme while Section III presents the grouping algorithms. Experimental results are given in Section IV on the ISCAS'85 and the combinational part of the ISCAS'89 benchmarks.

II. Proposed TPG Architecture

Consider a CUT with k inputs. The grouping of the k inputs of the CUT into n groups $(n < k)$ reduces the size of a TPG and the length of the test sequence for BIST applications [16], [17]. As it is shown in Fig. 1, the ith cell of an n-bit TPG, denoted by T_i, drives a group of CUT inputs.

In order to achieve high fault coverage in circuits with many random pattern resistant faults, many groups and hence very long sequences, are needed. The test length can be cut down by reducing the number n of groups, but this results in insufficient fault coverage. To achieve high fault coverage with short test length we propose a new scheme based on multiple partitions of the CUT inputs into groups. Then, the test session consists of two or more phases, each one with a different grouping of inputs. The number of groups is the same for all phases.

To reduce the hardware required for the implementation, the grouping procedure proposed in Section III ensures that there are n inputs of the CUT such that:

1) each one belongs to a different group from the other $n-1$ inputs in all groupings;
2) each one of these n inputs is assigned to the same TPG output in all groupings.

The TPG module can then be implemented by modifying suitably the cells $R_1, \ldots, R_n$ of the k-bit input register $(k > n)$. The proposed scheme is given in Fig. 2. The block phase selection determines the phase while the block assigning logic assigns the TPG outputs to the inputs of the CUT depending on the phase. In the following, we describe the modules of Fig. 2 in more detail.

A. Input Register

We denote the k-input register of the CUT as $R_1 \ldots R_n M_1 \ldots M_{k-n}$. In normal mode the input register receives inputs from the previous functional units and feeds them to the CUT. The $R_1 \ldots R_n$ cells are modified so that in test mode they implement either an LFSR (with primitive characteristic polynomial) or a binary counter. This part of the input register constitutes block TPG in Fig. 2. The other part of the input register $(M_1 \ldots M_{k-n})$ is modified to a single-clock two-port register. The single-clock two-port register is implemented with multiplexed flip–flops, which are controlled by a test/normal mode signal. In test mode, the TPG module produces n-bit vectors, and the assigning logic generates vectors each having $k-n$ bits. Hence, k-bit vectors are generated and applied to the CUT. The inputs of the assigning logic module are driven from the

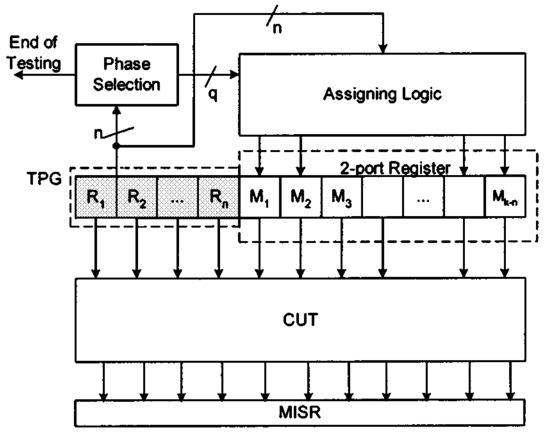

Fig. 2. The proposed scheme.

flip–flops of the TPG. Hence, while a test vector is applied to CUT, the input of each R_i receives the new value, corresponding to the new state of the TPG. This value is also spread through the assigning logic to the inputs of the cells M_j, which belong to the same group with R_i. These logic values are stored in the cells $R_1 \ldots R_n M_1 \ldots M_{k-n}$ at the next clock edge and hence appear at the inputs of the CUT at the next clock period. Then, taking into account that the delay of the CUT is always greater than that of the assigning logic module, we conclude that the test vectors are available at clock rate. Since the test vectors are available at clock rate and our technique does not insert any logic between the input register and the CUT inputs, we conclude that our method is suitable for at-speed testing. We consider that a register drives the inputs of the CUT; therefore, the multiplexers required for the selection between normal operation and test mode are placed at the inputs of the flip–flop of the input register (as in [16, Fig. 7]).

B. Phase Selection Module

The test session consists of one or more phases, where in each phase we have a different grouping of the inputs. During a phase, the TPG module of Fig. 2 generates all possible vectors. Fig. 3 presents the block diagram of the phase selection module.

Let q be the number of phases. The *last TPG state* block identifies the last state of the TPG and triggers a t-bit counter, $t = \lceil \log_2(q) \rceil$, to increase. The decoder transforms the binary output of the counter into the signals $P_1, P_2, \ldots, P_q$, which determine the current phase of the test session. During phase i, only P_i has the value 1. The counter is initialized at the $00\ldots0$ state and is incremented at the beginning of each new phase until it reaches state q. When the counter reaches state q, the end of the test session is signaled by the *end of testing* block.

The area overhead of the phase selection module is very small. As it is shown in experimental results, in most of the cases the number of the phases is less than 16. Therefore, no more than four cells are required for the implementation of the counter and the decoder is very small. Since the length n of the TPG is generally low (12 or less in most experiments), a very small number of gates is sufficient to identify the last state of the TPG. The end of testing block is also very small.

We have to note that when the test session consists of only one phase, the phase selection module is reduced to the last TPG state block which signals the end of testing mode.

C. Assigning Logic Module

In each phase of the test session the cells $M_1 \ldots M_{k-n}$ of the input register are divided into n groups $G_1, \ldots, G_n$. Cell R_i of the TPG also belongs to group G_i. The cells of group G_i have identical logic values stored during each phase. The assigning logic module is responsible for assigning each cell M_j to the appropriate cell of the TPG module, in each phase. When the test session consists of more than one phase, multiplexers can be used to accomplish this assignment. If cell M_j is a member of p different groups during different phases, then a $p \rightarrow 1$ multiplexer is necessary to select the value to be stored to the cell. If the cell is a member of the same group during all phases, then no multiplexer is necessary. The number and the size of the multiplexers can be reduced applying several simplifications.

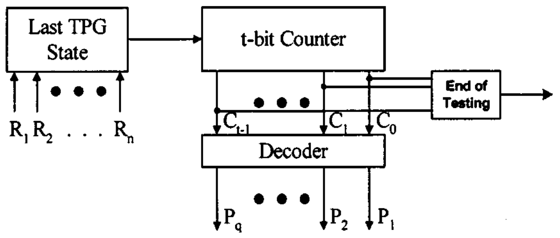

Fig. 3. Phase selection module.

1) Suppose that cells M_i and M_j of the two-port register belong to a common group, during each phase. Then a common multiplexer is sufficient to drive both cells.
2) Consider a test session with m phases and a cell M_j that belongs to x different groups $(x \ll m)$. Then less area overhead can be achieved if a $x \rightarrow 1$ multiplexer along with some glue logic for the selection signals is used to drive this cell instead of a $m \rightarrow 1$ multiplexer.

Many such simplifications can be performed due to the small number of outputs of the TPG module and the small number of phases. The algorithm presented in Section III minimizes the hardware overhead of the assigning logic module, by reducing the number of different outputs of the TPG that correspond to an input of the two-port register during all phases. In that way, a small number (even zero in many cases) of smaller multiplexers is required for many inputs of the two-port register.[1]

III. Grouping Procedure

The objective of the proposed method is to find for a given length n of the TPG, q different sets S_i, with $1 \leq i \leq q$, each one consisting of n groups G_j^i, with $1 \leq j \leq n$, of the primary inputs of the CUT. In set S_i group G_j^i consists of inputs of the CUT that in phase i of the test session receive the same logic value as the input of the CUT driven by the cell R_j of the TPG. The proposed method selects n cells of the input register to construct the TPG module and considers the rest $k-n$ cells of the input register as the two-port register module.

First of all, we have to select a value for n. The selection depends on the specific CUT. The value of n must be as small as possible, in order to bound the number of test vectors generated by the TPG. At the same time it is preferable for the first grouping to cover the majority of easy-to-detect faults, in order to reduce the process time. For that reason we apply random patterns to the circuit until there is a number of consecutive $p = 100$ vectors that do not detect any new faults. The number of these vectors minus the p last ones is considered as the number of vectors, s, detecting all easy faults. Then the first grouping must apply $2^n - 1$ vectors with $2^n - 1 \geq s$; therefore, $n = \lceil \log_2(s+1) \rceil$. Our experiments have verified that this value is a good initial selection for n. Other values close to n (greater or smaller) can be also considered for achieving better hardware overhead or shorter test sequences (as we can see at the experimental results) depending on the designer's specifications.

[1]In the case of implementing the multiplexing logic using logic gates, further area reduction can be achieved on the logic circuit by synthesis tools, which has not been examined in this paper.

The proposed grouping procedure consists of four main steps.

Step 1) The objective of the first step is to derive the first grouping, set S_1, in such a way that the majority of the easy-to-detect faults to be detected under this grouping. For that reason an algorithm, Algorithm A, is used which does not guarantee 100% fault coverage but runs very fast. We then run fault simulation for all vectors that can be applied to the CUT under this grouping and discard the detected faults from the fault list.

Step 2) At the second step, test vectors for the undetected faults are extracted via an ATPG tool. The ATPG tool is utilized in order to extract no less than one and no more than T (a given parameter) test vectors for every nonredundant fault, whenever that is possible. Multiple test vectors per fault are needed in order to ensure alternatives and produce better groupings on subsequent steps. The larger T is, the better the remaining sets S_i, $i > 1$, but the larger the execution time of the ATPG tool is. A typical value used in our experiments for T is 30. Initially, the ATPG tool runs fault simulation using random vectors and collects all vectors detecting at least one of the targeted faults. If for some faults the random vectors fail to produce at least one test vector, a deterministic ATPG algorithm is employed.

Step 3) In this step, the n cells $R_1, R_2, \ldots, R_n$ forming the TPG are selected. The selection of the $R_1, R_2, \ldots, R_n$ is very crucial, hence we developed an effective heuristic procedure.

Step 4) The last stage of the method has the objective to form sets S_i for $i > 1$. For that reason a heuristic algorithm, Algorithm B, is developed which receives a set of undetected faults and test vectors and creates a grouping detecting as many of these faults as possible. This algorithm is iteratively executed until all nonredundant faults have been covered.

In the following, we present the details of Algorithm A, the TPG cells selection, and Algorithm B.

A. Algorithm A

The vector $11\ldots1$ is applied to the CUT under any possible grouping, regardless of whether the TPG is implemented as a counter or an LFSR. When the TPG is implemented as a counter the vector $00\ldots0$ is also always applied. For that reason, depending on the used TPG, we drop at the beginning the faults detected by these vectors.

We then consider one group containing all inputs of the CUT and we attempt to split it into n groups $G_1, G_2, \ldots, G_n$ in $n-1$ repetitions. In each repetition m, with $m \in [1, \ldots, n-1]$, one of the groups $G_1, G_2, \ldots, G_m$ is selected and split into two groups. In order to select a good group and a good way of splitting it, the algorithm applies various random solutions until it cannot find a better solution for a predefined number of successive attempts. Specifically, each one of the m groups is repeatedly divided into two random parts while in the same time the rest $m-1$ groups remain unaffected. Then the 2^{m+1} vectors produced by the $m+1$ groups are applied to the CUT and the additional fault coverage is calculated. Among all the random groupings produced by the algorithm, the one with the largest additional fault coverage is selected.

In every repetition, the 2^{m+1} test vectors produced by the selected $m+1$ groups are always a superset of the 2^m test vectors produced by the original m groups. For that reason, in each repetition, we execute fault simulation applying only the new vectors for the undetected faults. This leads to a significant reduction of the execution time of Algorithm A.

Algorithm A achieves very high fault coverage of the easy-to-detect faults, even though the produced set S_1 is formed randomly in a very short time. However, due to its random nature, it cannot guarantee complete fault coverage.

B. Selection of TPG Cells

Hereafter, we denote as $I = \{I_1 \ldots I_k\}$ the set of inputs of the CUT, $\mathrm{IR} = \{\mathrm{IR}_1, \mathrm{IR}_2, \ldots, \mathrm{IR}_n\}$ the set of inputs driven by the cells $R_1, R_2, \ldots, R_n$ of the TPG, and $\mathrm{IM} = \{\mathrm{IM}_1, \mathrm{IM}_2, \ldots, \mathrm{IM}_{\mathrm{k-n}}\}$ the set of inputs driven by the cells $M_1, M_2, \ldots, M_{k-n}$ of the two-port register. Obviously, we have $I = \mathrm{IR} \cup \mathrm{IM}$. The selection of cells $R_1, R_2, \ldots, R_n$ is equivalent to the selection of inputs $\mathrm{IR}_1, \mathrm{IR}_2, \ldots, \mathrm{IR}_\mathrm{n}$.

According to the first grouping the set I has been split into n groups $G_1^1, \ldots, G_n^1$. From each group we select one representative so as to form set IR. In order to select the representatives, a weight $\mathrm{WR}(I_a, I_b)$ for every pair of inputs I_a, I_b is calculated. For each fault f of the so far undetected faults F of the circuit, and for every pair of inputs I_a, I_b we calculate the percentage $p_f(I_a, I_b)$ of its test vectors that have complementary logic values at these inputs. The higher the percentage $p_f(I_a, I_b)$ is, the more difficult is the fault to be detected if inputs I_a, I_b are put in the same group. However, since the test vectors, used by our method, for each fault may represent only a small portion of the test vectors which detect this fault, the previous argument may be misleading if we base our decision on the exact value of $p_f(I_a, I_b)$. Hence, for each range of values of $p_f(I_a, I_b)$ we assign a weight. When $p_f(I_a, I_b)$ is less than 50%, less than half of the produced test vectors require complementary values at inputs I_a, I_b. In this case, this fault can be easily detected even if inputs I_a, I_b receive the same value and therefore we assign to $w_f(I_a, I_b)$ the value of zero. On the other hand, when $p_f(I_a, I_b)$ is approaching 100%, then almost all test vectors require inputs I_a, I_b to receive complementary values; therefore, the probability that the fault will remain undetected if inputs I_a, I_b receive always the same logic value is very large. In this case $w_f(I_a, I_b)$ is given a large value. Specifically, we assign weights in the following way:

$$w_f(I_a, I_b) = \begin{cases} 0, & p_f(I_a, I_b) < 50\% \\ 2, & 50\% \le p_f(I_a, I_b) \le 70\% \\ 8, & 70\% > p_f(I_a, I_b) \le 90\% \\ 16, & 90\% < p_f(I_a, I_b) \le 100\%. \end{cases}$$

Then the weight $\mathrm{WR}(I_a, I_b)$ is estimated as the sum of $w_f(I_a, I_b)$ for all faults, that is $\mathrm{WR}(I_a, I_b) = \sum_{f \in F} w_f(I_a, I_b)$. The value of $\mathrm{WR}(I_a, I_b)$ is an indication of the impact of assigning inputs I_a, I_b in the same group. When $\mathrm{WR}(I_a, I_b)$ is large, we should avoid putting I_a and I_b in the same group. In this case, we prefer to assign these inputs in set IR to ensure that they will not be assigned in the same group in any of the future groupings.

At first, among the inputs belonging to G_1^1 and G_2^1 we select the inputs $I_{a1} \in G_1^1$, and $I_{a2} \in G_2^1$ with the maximum weight $\mathrm{WR}(I_{a1}, I_{a2})$. We symbolize I_{a1} and I_{a2} as IR_1 and IR_2, respectively. Then, we select, as representative of G_3^1, the input $I_{a3} \in G_3^1$ which has the larger sum $\mathrm{WR}(I_{R1}, I_{a3}) + \mathrm{WR}(I_{R2}, I_{a3})$. We symbolize I_{a3} by IR_3. In the same way we select, as the representative of group G_p^1, the input $I_{ap} \in G_p^1$ which has the maximum value of $\sum_{i=1}^{p-1} \mathrm{WR}(\mathrm{IR_i}, I_{ap})$.

C. Algorithm B

Algorithm B receives as inputs: 1) a list of undetected faults; 2) a set of multiple test vectors for them; 3) the set $\mathrm{IR}_1, \mathrm{IR}_2, \ldots, \mathrm{IR}_\mathrm{n}$; and 4)

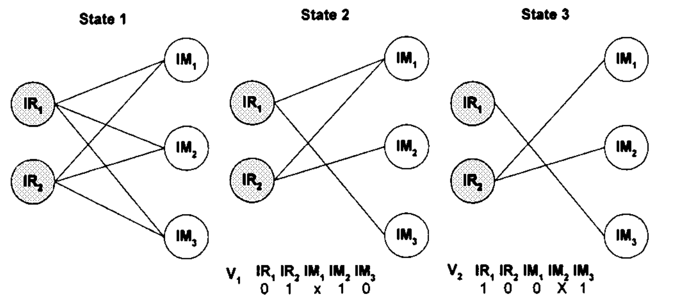

Fig. 4. Graph reduction example.

the groupings constructed so far, and creates a new grouping detecting as many faults as possible.

Consider a graph where nodes correspond to inputs and the edge connecting two nodes represents the ability to put these inputs in the same group. The nodes of the graph are divided into two sets: N_R, the nodes corresponding to inputs $\mathrm{IR}_1, \mathrm{IR}_2, \ldots, \mathrm{IR}_\mathrm{n}$, and N_M, the nodes corresponding to inputs $\mathrm{IM}_1, \mathrm{IM}_2, \ldots, \mathrm{IM}_{\mathrm{k-n}}$. Initially, all nodes of set N_R are connected to all nodes of set N_M. This means that each input of set N_M can be put in the same group with each input of set N_R. The objective is to reduce the edges of the graph in order each node of set N_M to be connected with exactly one node of set N_R. Then, all nodes of set N_M connected to node IR_i, along with node IR_i, construct a single group. We note that each time the state of the graph represents all permissible groupings of inputs.

The reduction of the graph edges is done as follows: Firstly, the algorithm selects a test vector t. Let the logic values of the bits corresponding to inputs IM_j and IR_i be complementary in t. Then in order for t to be applied under any grouping, represented by the state of the graph, inputs IM_j and IR_i must not be assigned in the same group. Therefore, we remove the edge connecting the corresponding nodes. It is obvious that after the removals imposed by vector t, each node of set N_M must be connected to at least one node of N_R else the graph will not represent a valid grouping. In the latter case, we reject the test vector. This procedure is repeated until either all faults have been covered or there is not any other test vector of a so far undetected fault that can be produced by the grouping. Then, if a node of N_M is connected to p nodes of N_R, with $p > 1$, the hardware overhead required by each connection is estimated and the $p-1$ connections with the larger hardware overhead are removed.

Fig. 4 presents a very simple example of the edge reduction procedure. In this example, a counter TPG is considered. Assume that initially we have sets $N_R = \{\mathrm{IR}_1, \mathrm{IR}_2\}$ and $N_M = \{\mathrm{IM}_1, \mathrm{IM}_2, \mathrm{IM}_3\}$. In state 1 all nodes of N_R are connected to all nodes of N_M. For the vector $(\mathrm{IR}_1, \mathrm{IR}_2, \mathrm{IM}_1, \mathrm{IM}_2, \mathrm{IM}_3) = 01 \times 10$ the connections $\mathrm{IR}_1 - \mathrm{IM}_2$ and $\mathrm{IR}_2 - \mathrm{IM}_3$ have to be removed because the corresponding bits in the test vector are complementary. Then we get state 2. In the same way for the vector $(\mathrm{IR}_1, \mathrm{IR}_2, \mathrm{IM}_1, \mathrm{IM}_2, \mathrm{IM}_3) = 100 \times 1$ we get state 3 and the resulting groups are $\{\mathrm{IR}_1, \mathrm{IM}_3\}$ and $\{\mathrm{IR}_2, \mathrm{IM}_1, \mathrm{IM}_2\}$. The vectors produced by this grouping are (00000), (01110), (10001), (11111); therefore, the required test vectors will be applied to the CUT.

A very crucial factor for the efficiency of the algorithm is the selection of the test vectors used for the graph edge removal. Each test vector and each undetected fault is associated with a cost. The cost of test vector v indicates the possibility to fail covering other test vectors if v is selected. Then, based on the costs of their test vectors, the faults are classified into easy and hard and a cost is given to each one of them. The details are given in the Appendix. A very good approach is to select test vectors beginning from the hard faults because easy faults can be covered easily.

The first step of the algorithm is the initialization of the graph and the calculation of the costs of all undetected faults and the costs of their test vectors. Then the algorithm executes repeatedly the following two steps until the fault list becomes empty.

1) The test vectors list is checked for test vectors which do not make any reductions of the edges of the graph. These vectors are produced by any grouping represented by the current state of the graph, therefore the faults that are detected by these vectors are discarded from the fault list.
2) The hardest fault is selected, that is the fault with the maximum cost. If the fault has at least one test vector that can be applied under a grouping represented by the state of the graph, then the one with the smallest cost (less harmful) is selected. The graph reductions are performed and the fault is discarded permanently from the fault list, else the fault is temporarily discarded from the list of the faults.

When the list becomes empty, all faults discarded temporarily are appended again and new test vectors are extracted via the application of a few random vectors. This is done because some of the undetected faults may be quite easy and have more test vectors than the so far extracted ones. If new test vectors are extracted the algorithm makes a new effort to cover them by executing again the above procedure, else the repetition terminates and the algorithm initiates a new repetition for the construction of the next grouping.

The algorithm terminates when a repetition fails to cover any more faults, or the edges of the graph cannot be furthermore reduced. In the first case, the edges can be further reduced using hardware minimization criteria. For that reason, we examine the previous groupings and try to make the most frequent assignments in the new grouping. In this way, the necessary multiplexers can be minimized. Specifically, suppose that node IM_j is connected with p nodes of the set N_R. We select to retain the connection which assigns input IM_j to the group of input IR_i in which was mostly assigned in the previous groupings. The rest of the $p-1$ connections are removed.

IV. Experimental Results

In order to validate the effectiveness of the proposed scheme, we have performed several experiments. Table I gives experimental results on the ISCAS'85 benchmarks. The first and the second column reports the circuit name and the number of its inputs, while the third column shows the TPG length used for the experiment. Columns 4 and 5 present the number of phases and the total number of test vectors, respectively, required for achieving complete 100% fault coverage. The area overhead column gives, in gate equivalents (GEs), the hardware required for the implementation of the assigning logic module and the phase selection module. We assume that each n-input NAND or NOR gate is $0.5\,n$ gate equivalents, and each D flip–flop is equal to 3.5 gate equivalents. We also assume that the $m \rightarrow 1$ multiplexer is implemented by the use of m transmission gates and is equal to $0.5\,m$ gate equivalents. Decoding logic is not used in the multiplexers because the control lines are provided directly by the decoder shown in Fig. 3.

Table I shows that when the TPG size decreases, the number of phases required for complete fault coverage increases as well as the hardware overhead while the total number of vectors decreases. Therefore, there is a tradeoff between the area overhead and the test length. A BIST designer can select between a TPG with small size that leads to a solution with less test vectors and more area overhead and a larger TPG that leads to a solution with more test vectors and less area overhead.

The CPU time required for selecting the number of phases and the groupings of the inputs of the CUT in each phase was measured on a 500-MHz Intel Pentium-III processor system with 100-MHz system

TABLE I
RESULTS FOR 100% FAULT COVERAGE ON THE ISCAS'85 BENCHMARK CIRCUITS WITH HARD TO DETECT FAULTS

Circuit	Number of Inputs	TPG Length	Phases	Test vectors	Hardware (GEs)
c880	60	5	8	248	159.5
		6	6	378	112.5
		7	6	762	116
		8	5	1275	93
c1355	41	5	15	465	281
		6	12	756	237
		7	10	1270	192
		8	4	1020	68
		9	3	1533	56.5
		10	1	1023	0
c1908	33	7	11	1397	147
		8	7	1785	91
		9	5	2555	65.5
c2670	233	8	14	3570	464.5
		9	12	6132	425.5
		10	9	9207	304
c7552	207	8	12	3060	719
		9	11	5621	632.5
		10	9	9207	528

TABLE II
RESULTS FOR 100% FAULT COVERAGE ON THE ISCAS'89 BENCHMARK CIRCUITS WITH HARD TO DETECT FAULTS

Circuit	Number of Inputs	TPG Length	Phases	Test vectors	Hardware (GEs)
s420	34	6	10	630	141.5
		7	11	1397	112.5
s641	54	5	8	248	106.5
		6	8	504	93
		7	6	762	82.5
s838	66	8	14	3570	288.5
s9234	247	10	16	16386	790
		11	11	22517	632.5
s13207	700	11	7	14329	456.5
		12	7	28665	429.5
s15850	611	9	12	6132	909.5

bus speed and 128 MBs of main memory and was always less than 30 min. The largest portion is due to the large number of random solutions applied in algorithm A. This CPU time does not include the ATPG component, which was not implemented with efficiency considerations in mind since it can be substituted by any commercial ATPG tool.

In Table II, we present experimental results for the combinational part of ISCAS'89 benchmark circuits with random pattern resistant faults. In this case, the outputs of the flip–flops used for storing the internal state of the sequential circuit are considered as primary inputs, and the inputs of the flip–flops are considered as primary outputs.

Among the already known test pattern generation schemes for test-per-clock BIST the schemes proposed in [14], [15], [18]–[20], and [22] insert logic between the TPG outputs and the CUT inputs. This makes at-speed testing of the CUT difficult and may affect the performance of the system in normal operation. The hardware overhead of the method proposed in [16] and [17] is minimal, however the test lengths in most cases are very long. The methods proposed in [12], [21], [23], and [24] as well as our method are suitable for at-speed testing.

In Tables III and IV, we compare our method against [12], [21], [23], and [24] for the ISCAS'85 and ISCAS'89 benchmark circuits with random pattern resistant faults. Our method as well as MFBIST [21] can provide several solutions which tradeoff the test length and the hardware requirements. Among the derived solutions the best, with respect to test length, are given. We note that a dash (-) in Tables III and IV means that no results have been provided by the authors of the referenced paper for the specific benchmark circuit.

The comparison presented in Table III is based on the number of test vectors required for fully testing the CUT. We note that in [12] and in [23] the seeds are loaded serially in the register. The same assumption was made for [24] (no hint was given by its authors about this) since this approach is commonly used and results in the minimum hardware overhead. Thus, the test application time for the techniques of [23] and [24] can be calculated by the formula

$$\text{Test Appl. Time} = (\text{Test Vectors} - \text{Seeds}) \times f_1 + (\text{Seeds} \times PI) \times f_2$$

where *test vectors* and *seeds* are the number of the test vectors and seeds needed by each technique for fully testing the CUT, PI is the number of primary inputs of the corresponding circuit, f_1 is the functional frequency of the CUT, and f_2 is the scan-in frequency of the seeds. For the technique of [12], the corresponding test application time is derived by the formula

$$\text{Test Application Time} = \text{Test Vectors} \times f_1 + \text{Seeds} \times PI \times f_2$$

where test vectors $= 3 \times$ Seeds $\times PI$. For our method as well as MFBIST the test application time is given by the formula

$$\text{Test Application Time} = \text{Test Vectors} \times f_1.$$

Although in Table III we do not take into account the reseeding time, our method favorably compares with those of [12], [21], [23], and [24].

In Table IV comparisons based on hardware overhead are given. Note that the implementation of the MFBIST [21] test pattern generation scheme requires a quantity H1 in addition to the equivalent gates listed in Table IV. H1 accounts for a ROM component, a shift register with parallel load, a Control-PLA, and other combinational logic. Reference [21] does not give enough information to calculate the hardware overhead of H1. From Table IV we can see that our approach in many cases is less hardware intensive than MFBIST. The hardware H2 and H3 required for the implementation of the control logic of the TPG schemes presented in [23] and [24] cannot be calculated, because of the lack of information provided in the papers. Even if we do not take into account the hardware symbolized by H1, H2, and H3 we can see that in most cases our method requires less hardware overhead than the other techniques.

The logic required for the modification of an k-bit register into a scan register in [12], [21], [23], [24] is equivalent to that required in our scheme for the modifications of a part of a k-bit register to two-port register and of the rest to an LFSR. Both modifications have not been included in the hardware given in Table IV.

V. CONCLUSION

A new TPG scheme for circuits with random pattern resistant faults has been proposed. This scheme is based on multiple groupings of the inputs of the CUT. The number of groups n is constant. During each test phase a new grouping is applied and the inputs belonging to the same group are driven by the same output of an n-stage primitive LFSR or counter. Using the proposed method we achieve complete fault coverage with short test lengths and low hardware overhead while supporting at-speed testing. The short test length implies that the method is attractive to low energy applications. Our experimental results have shown that our approach compares favorably to the already known test-per-clock built-in TPGs.

TABLE III
TEST VECTOR COMPARISONS

Circuit	Proposed technique (test vectors)	MFBIST [21]		[23]		[12]		LFSR-based TPGs [24]	
		Test Vectors	Reduction	Test Vectors	Reduction	Test Vectors	Reduction	Test Vectors	Reduction
c880	248	1140	78.2%	1596	84.5%	-	-	1829	86.4%
c1355	465	2460	81.1%	1447	68.6%	-	-	1334	65.9%
c1908	1397	2937	52.4%	3659	61.8%	-	-	3759	62.8%
c2670	3570	5592	36.2%	7300	51.1%	58930	93.9%	10206	65.0%
c7552	3060	16K	80.1%	31282	90.2%	76447	96.0%	-	-
s420	630	1632	61.4%	5775	89.1%	10816	94.2%	10843	94.2%
s641	248	648	61.7%	2345	89.4%	11458	97.8%	2430	89.8%
s838	3570	3696	3.4%	17526	79.6%	15742	77.3%	9273	61.5%
s9234	16K	30K	46.6%	108638	84.9%	-	-	-	-
s13207	14K	44K	68.2%	43543	67.1%	64600	77.8%	-	-
s15850	6132	37K	83.8%	34121	82.0%	-	-	-	-

TABLE IV
HARDWARE OVERHEAD COMPARISONS

Circuit	Proposed Technique (GE)	MFBIST [21]	[23]			[12]			LFSR-based TPGs [24]		
		Hardware Overhead	ROM bits (GE)[a]	Control Logic	P-LFSR + Bit Counter (GE)	ROM bits (GE)[a]	Control Logic (GE)	Bit Counter (GE)	ROM bits (GE)[a]	Control Logic	Bit Counter (GE)
c880	159.5	0 + H1	30	H2	284	-	-	-	0	0	0
c1355	281	0 + H1	0	0	0	-	-	-	0	0	0
c1908	147	63 + H1	17	H2	165	-	-	-	0	0	0
c2670	464.5	577.5 + H1	757	H2	1052	4019	38	27	1922	H3	27
c7552	719	1333.5+ H1	880	H2	938	5486	38	27	-	-	-
s420	141.5	10.5 + H1	51	H2	169	60	22	20	77	H3	20
s641	106.5	178.5 + H1	68	H2	257	108	26	20	81	H3	20
s838	288.5	294 + H1	182	H2	314	462	31	23	710	H3	23
s9234	790	766.5 + H1	1297	H2	1114	-	-	-	-	-	-
s13207	456.5	157.5 + H1	1225	H2	3114	4375	32	34	-	-	-
s15850	909.5	556.5 + H1	1986	H2	2723	-	-	-	-	-	-

[a] *We have taken into account the assumption made in [23], that, on average, 0.25 gates are required for each memory cell of a ROM*

APPENDIX
COST CALCULATION OF THE UNDETECTED FAULTS AND THEIR TEST VECTORS

At first, a weight is assigned to each edge of the graph which represents the possibility this edge to be removed during the execution of the algorithm. All edge's weights are initialized to zero. Then for each pair $(\mathrm{IM}_j, \mathrm{IR}_i)$ we increase the weight of the edge $\mathrm{IM}_j \rightarrow \mathrm{IR}_i$ by w/t, where t is the number of the test vectors of fault f which can be applied under the current state of the graph and w the number of them which remove edge $\mathrm{IM}_j \rightarrow \mathrm{IR}_i$.

The weight of the edges will be used for the estimation of the costs of test vectors and faults. There are some useful properties regarding the weights of the edges.

1) Consider that all edges of node IM_j have large weights. Then many test vectors will require the removal of these edges and considering that at least one of them must not be removed we come to the deduction that there is a strong possibility that some faults may be left uncovered. For that reason, we preserve these edges as long as possible, by avoiding selecting test vectors removing them. A good estimation of this property is given by

$$\mathrm{RemoveFreedom}(\mathrm{IM}_j) = \sum_{IR_i \in R} \frac{1}{\mathrm{Weight}\ (\mathrm{IM}_j \rightarrow \mathrm{IR}_i)}$$

where R is the set of nodes IR_i connected via an edge with IM_j. Smaller values of $\mathrm{RemoveFreedom}(\mathrm{IM}_j)$ indicate greater number of faults that will remain uncovered. This is due to the fact that if a test vector removing an edge of node IM_j is selected, then a large number of test vectors will not be produced by the groupings represented by the state of the graph.

2) Suppose that some edges of node IM_j have large weights, while the others have small but not zero. In this case, there are far more test vectors removing edges with large weights than vectors removing edges with small weights. Therefore, the edges with large weights are more likely to be removed during the execution of the algorithm. Hence, the best policy is to prefer selecting vectors that remove the edges with large weights and preserve edges with the smaller weights. The reason is that edges with large weights are expected to be removed by many test vectors. If we remove all edges with small weights, taking into account that at least one edge cannot be removed (the last one), then it is obvious that this edge will have large weight. This means that many faults may become uncovered. A good estimation of this property is given by

$$\mathrm{RemoveDanger}(v, \mathrm{IM}_j) = \sum_{IR_i \in R^*} \frac{1}{\mathrm{Weight}\ (\mathrm{IM}_j \rightarrow \mathrm{IR}_i)}$$

where v is a test vector and R^* is the set of all nodes IR_i which satisfy the condition *the selection of test vector* v *removes the edge* $\mathrm{IM}_j \rightarrow \mathrm{IR}_i$. The larger the $\mathrm{RemoveDanger}(v, \mathrm{IM}_j)$ is, the larger is the number of faults which will become undetected if vector v is selected.

3) When an edge of node IM_j has zero weight, then a vector for removing this edge does not exist. Therefore, no matter what removals are going to be made at the node, this edge will never be removed. For that reason, such a node should not be taken into account in our decisions since always guarantees a valid solution.

Based on the above observation we define the cost of a test vector as

$$\mathrm{VectorCost}(v) = \sum_{IM_j \in M} \frac{\mathrm{RemoveDanger}\ (v, \mathrm{IM}_j)}{\mathrm{RemoveFreedom}\ (\mathrm{IM}_j)}$$

where M is the set of all nodes IM_j excluding those fulfilling property 3. The larger the cost of a vector v is, the less test vectors can be produced by the grouping if v is selected.

The cost of a fault f, with V_f denoting the set of its test vectors, is then estimated as

$$\mathrm{FaultCost}(f) = \sum_{v \in V_f} \mathrm{WeightVector}(v).$$

A larger fault cost indicates that it is harder for the algorithm to cover it.

References

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

[2] P. H. Bardell, W. H. McAnney, and J. Savir, *Built-In Test for VLSI: Pseudo-Random Techniques*. New York: Wiley, 1987.

[3] V. Agrawal, C. Kime, and K. Saluja, "A tutorial on built-in self-test part 1: Principles," *IEEE Design Test Computers*, pp. 73–82, Mar. 1993.

[4] H.-J. Wunderlich, "BIST for systems-on-a-chip," *Integration, VLSI J.*, vol. 26, no. 1-2, pp. 55–78, Dec. 1998.

[5] M. Bushnell and V. Agrawal, *Essentials of Electronic Testing*: Kluwer, 2000.

[6] K.-T. Chen and C.-J. Lin, "Timing driven test point insertion for full-scan and partial-scan BIST," in *Proc. Int. Test Conf.*, 1995, pp. 506–514.

[7] A. Stroele and H.-J. Wunderlich, "TESTCHIP: A chip for weighted random pattern generation, evaluation, and test control," *IEEE J. Solid-State Circuits*, vol. 26, pp. 1056–1063, July 1991.

[8] 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. Comput.*, vol. 44, pp. 223–233, Feb. 1995.

[9] H.-J. Wunderlich and G. Kiefer, "Bit-flipping BIST," in *Proc. Int. Conf. Computer-Aided Design*, 1996, pp. 337–343.

[10] S. Hellebrand, H.-G. Liang, and H.-J. Wunderlich, "A mixed mode BIST scheme based on reseeding of folding counters," in *Proc. Int. Test Conf.*, 2000, pp. 778–784.

[11] N. A. Touba and E. J. McCluskey, "Bit-fixing in pseudorandom sequences for scan BIST," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 545–555, Apr. 2001.

[12] K. Chakrabarty, B. T. Murray, and V. Iyengar, "Built-in test pattern generation for high-performance circuits using twisted-ring counters," in *Proc. IEEE VLSI Test Symp.*, 1999, pp. 22–27.

[13] N. A. Touba and E. J. McCluskey, "Test point insertion based on path tracing," in *Proc. VLSI Test Symp.*, 1996, pp. 2–8.

[14] J. Hartmann and G. Kemnitz, "How to do weighted random testing for BIST," in *Proc. Int. Conf. Computer-Aided Design*, 1993, pp. 568–571.

[15] C. Okmen, M. Keim, R. Krieger, and B. Becker, "On optimizing BIST-architecture by using OBDD-based approaches and genetic algorithms," in *Proc. VLSI Test Symp.*, 1997, pp. 426–431.

[16] C.-A. Chen and S. K. Gupta, "A methodology to design efficient BIST test pattern generators," in *Proc. Int. Test Conf.*, 1995, pp. 814–823.

[17] ——, "Efficient BIST TPG design and test set compaction via input reduction," *IEEE Trans. Computer-Aided Design*, vol. 17, pp. 692–705, Aug. 1998.

[18] K. Chakrabarty, B. Murray, J. Liu, and M. Zhu, "Test width compression for built-in self test," in *Proc. Int. Test. Conf.*, 1997, pp. 327–337.

[19] I. Hamzaoglu and J. Patel, "Reducing test application time for built-in-self-test test pattern generators," in *Proc. VLSI Test Symp.*, 2000, pp. 369–375.

[20] N. A. Touba and E. J. McCluskey, "Synthesis of mapping logic for generating transformed pseudo-random patterns for BIST," in *Proc. Int. Test Conf.*, 1995, pp. 674–682.

[21] M. F. Alshaibi and C. R. Kime, "MFBIST: A BIST method for random pattern resistant circuits," in *Proc. Int. Test Conf.*, 1996, pp. 176–185.

[22] C. Fagot, P. Girard, and C. Landrault, "On using machine learning for logic BIST," in *Proc. Int. Test Conf.*, 1997, pp. 338–346.

[23] L. R. Huang, J. Y. Jou, and S. Y. Kuo, "Gauss-elimination-based generation of multiple seed-polynomial pairs for LFSR," *IEEE Trans. Computer-Aided Design*, vol. 16, pp. 1015–1024, Sept. 1997.

[24] S. Chiusano, P. Prinetto, and H. J. Wunderlich, "Non-intrusive BIST for systems-on-a-chip," in *Proc. Int. Test Conf.*, 2000, pp. 644–651.

BDS: A BDD-Based Logic Optimization System

Congguang Yang and Maciej Ciesielski

***Abstract*—This paper describes a novel logic decomposition theory and a practical logic synthesis system, *BDS*. It is based on a new binary decision diagrams (BDD) decomposition technique which supports all types of decomposition structures, including AND, OR, XOR, and complex MUX, both algebraic and Boolean. As a result, the method is very efficient in synthesizing both AND/OR and XOR-intensive functions. It also has a capability to handle very large circuits, as it employs the BDD decomposition in the partitioned Boolean network environment. The experimental results show that BDD-based logic decomposition is a promising alternative to the existing logic optimization approaches. In particular, it offers a superior runtime advantage over traditional logic synthesis systems.**

***Index Terms*—BDD, logic optimization, synthesis.**

I. Introduction

Traditional logic optimization methodology, based on algebraic factorization [1], [2], has gained tremendous success and emerged as a dominant method in logic synthesis. However, while near optimal results can be obtained for AND/OR-intensive functions of control and random logic, results are far from satisfactory for arithmetic and XOR-intensive logic functions, which can be more compactly represented as a combination of AND/OR and XOR expressions. Although logic optimization methods based on Boolean factorization can potentially offer better results than algebraic methods, they failed to compete with algebraic techniques due to their high computational complexity. We believe that this failure of Boolean optimization techniques is caused by inappropriate data structure used to represent Boolean functions. The predominant cube representation used by those techniques naturally favors algebraic-based methods and is not suitable for Boolean operations. Consequently, Boolean operations such as MUX and XOR received less attention from the onset of logic synthesis research.

We believe that logic synthesis methods will keep evolving with the emergence of newer and more efficient logic representations, and in particular with the accumulation of expertise in binary decision diagrams (BDDs). This paper presents the first results of research that address this new opportunity. It presents a novel theory and a set of efficient techniques for logic decomposition based on BDD representation. We show that logic optimization can be efficiently carried out through an iterative BDD decomposition and manipulation. Our approach proves to be very efficient for both AND/OR- and XOR-intensive functions. To the best of our knowledge, this is the first unified logic optimization methodology that allows one to optimize such diverse classes of logic functions. We also present a practical and complete BDD-based logic optimization system, *BDS*, that can handle arbitrarily large circuits. It employs the BDD decomposition techniques in the partitioned Boolean network environment.

Manuscript received July 13, 2001. This work was supported in part by the National Science Foundation under Contract CCR-9901254. This paper was recommended by Associate Editor E. Macii.

C. Yang is with Chameleon Systems, Inc., San Jose, CA 95134 USA (e-mail: cyang@chameleonsystems.com).

M. Ciesielski is with the Department of Electrical and Computer Engineering, University of Massachusetts at Amherst, Amherst, MA 01003-4410 USA (e-mail: ciesiel@ecs.umass.edu).

Publisher Item Identifier S 0278-0070(02)05630-0.