A Statistic-based Approach to Testability Analysis

Chuang-Chi Chiou, Chun-Yao Wang, Yung-Chih Chen
Department of Computer Science, National Tsing Hua University, HsinChu, Taiwan
{mr944333, wcyao, ycchen}@cs.nthu.edu.tw

ABSTRACT

This paper presents a statistic-based approach for evaluating the testability of nodes in combinational circuits. This testability measurement is obtained via Monte Carlo simulation governed by the formulated Monte Carlo model. The Monte Carlo simulation is terminated when the predefined error with respect to the Monte Carlo model, under a specified confidence level, is achieved. We conduct the experiments on a set of ISCAS'85 and MCNC benchmarks. As compared with previous work, our approach more efficiently evaluates the testability with less error rate.

I. INTRODUCTION

Testability analysis in logic circuits usually contains three measurements, controllability, observability, and fault detection probability. Controllability of a node in a circuit is a measurement to show the difficulty to set a value to this node from the primary inputs (PIs). Observability of a node in a circuit is a measurement to show the difficulty to observe the value of a node from the primary outputs (POs). Fault detection probability of a node in a circuit is a measurement to show the difficulty to simultaneously set a value to a node from the PIs (fault activation) and observe the node in the POs (fault effect propagation). Fault detection probability tells designers which nodes in the circuit are hard to test, and therefore design-for-testability (DFT) hardware is mandatory at these locations for achieving higher fault coverage [18]. Furthermore, this measurement could provide guidance for test pattern generation. It reduces the number of backtracking in the process of logic implication in ATPG.

Many efforts have been made in the testability analysis [2], [3], [5]-[9], [11]-[13], [15], [16]. Some work are for controllability and observability [8], [11]-[13], [15], but some work are for fault detection probability [3], [6]. The SCOAP [8] is the first algorithm for computing controllability and observability. It rates each node a number ranging from 0 to infinity, which represents the difficulty in controlling and observing the value of the node. Although SCOAP is efficient in general, it is inaccurate due to neglecting signal correlation between the inputs of a node.

The PREDICT [15], the first exact controllability measurement, tries to split the circuit into many partitions. A partition is known as a supergate, which completely covers the reconvergent fanouts and their reconvergent nodes. However, the algorithm has exponential computation cost with the number of fanouts in each supergate. As to the worst case, the original whole circuit may be a single supergate. Therefore, the PREDICT can compute the exact controllability only for certain circuits.

To deal with the issue of large supergate in the exact approach, cutting algorithms are proposed in [12], [13]. In [12], Savir et al. estimate the controllability lower bound and upper bound of nodes by cutting all multiple fanout branches and initializing each cut line to a controllability value ranging between 0 and 1. Later, Savir proposes an improved cutting algorithm [13] combining the original cutting algorithm [12] and Park-McCluskey technique [11]. The improved algorithm cuts only partial fanout branches, and the remainders are manipulated by the symbolic expression [11]. Although these two cutting algorithms can deal with larger circuits, the computation efforts are still large as well.

The state of the art work more related to ours is the TAIR algorithm [6]. It uses logic reasoning techniques used in ATPG to derive formulae for correcting the fault detection probability obtained by the COP [3]. The algorithm is efficient and improves the accuracy, but it still has 20∼30 percent of error rate on average.

Previous approaches to testability analysis are vectorless. They obtain the results by the symbolic expression and arithmetic operations. The symbolic expression usually has a limitation on scalability, and the arithmetic operations usually do not consider the signal correlation issue. Therefore, previous approaches are either computation intensive or inaccurate.

From the application point of view, exact testability analysis is not a must, approximate values with a small error would be acceptable. Furthermore, the algorithm has to be efficient, especially for large circuits. Therefore, this paper proposes a simulation-based statistic approach based on the Monte Carlo method for testability analysis. Our approach randomizes a large amount of patterns for parallel simulation. Then the backtracing is performed to evaluate the fault detection probability of each nodes according to the simulation results. We introduce a statistical model for error estimation to formulate our stopping condition. As a result, the iterative simulation would continue until the desired accuracy, at a specified confidence level, is achieved.

The rest of this paper is structured as follows. Section II gives the background of our approach. Section III describes the proposed approach for analyzing the testability. Section IV shows the experimental results. Section V concludes this paper.

II. BACKGROUND

In this section, we review the background of the Monte Carlo method. The *parallel pattern simulation* [17] and *critical path tracing* [1] technique are also introduced. The Monte Carlo method is used to ensure that the testability value has a desired accuracy within a confidence level. The latter techniques assist us in evaluating the testability.

A. Monte Carlo Method

We first briefly describe the major components of the Monte Carlo simulation approach. They comprise the foundation of most Monte Carlo applications. These components are as follows:

i. Random number generator: a source of random numbers such that the generated numbers are uniformly distributed on the unit interval.
ii. Sampling rules: some defined rules that evaluate the result of each sampling.
iii. Scoring: accumulate each outcome into overall scores for the quantities of interest.
iv. Error estimation: an estimation of the statistical error as a function of the number of trials.

This work was supported in part by the National Science Council of R.O.C. under Grant NSC96-2220-E-007-009

0-7695-3117-2/08 $25.00 © 2008 IEEE
DOI 10.1109/ISQED.2008.89

B. Parallel Pattern Simulation and Critical Path Tracing

Parallel pattern simulation is a mechanism that can simultaneously simulate x patterns, where x could be up to 16,777,216 (2^{24}) by using a *large number package*, e.g., GMP library [20]. This simulation is very efficient but spends a large memory space to record the data as compared to the traditional logic simulation. There is a trade-off between the computation time and memory space. Thus, we should determine a proper value of x to ensure that simulation would end up in a reasonable time with acceptable memory consumption.

The critical path tracing technique is based on the concept of critical values, first defined in [19]. A wire w has a critical value v in the test vector t if and only if t detects w the *s-a-*$\overline{v}$ fault. According to the definition of criticality, we can derive critical value of each line from the POs to the PIs by one pass of backtracing. Although the backtracing is efficient, it could merely provide an approximate result due to not considering the self-masking and multiple path sensitization effects. But in general, the error caused by the backtracing is very small and can be ignored. This is because the self-masking and multiple path sensitization infrequently occur in the circuits.

III. Our Approach

In this paper, we assume that the networks only consist of AND, OR, and NOT gates for simplicity. Complex gates can be decomposed into these three types of gates. The four components of Monte Carlo method associated with our work are described in the following four subsections in detail.

A. Random Pattern Generator (RPG)

The architecture of our approach is shown in Fig. 1. Two elements are included, one is a random pattern generator (RPG) and the other is the circuit under test (CUT) S with M outputs. If the CUT S is an N-input circuit, then the RPG also produces N outputs. The RPG is with a parameter r which indicates that the RPG can generate patterns with 2^r bits by using the GMP library [20]. These parallel random patterns are used as the simulation patterns for the CUT S.

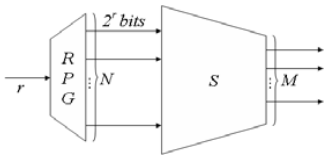

Fig. 1. The architecture of our approach.

B. Sampling Rules

After a trial of parallel pattern simulation, each node of network is also associated with a vector of 2^r bits. We denote this vector and its inverse on the wire w as $V(w)$ and $\overline{V(w)}$. The 1's controllability and 0's controllability of the wire w are denoted as $C_1(w)$ and $C_0(w)$, respectively. $|V(w)|$ represents the number of 1 in the vector $V(w)$. Thus, the 1's controllability and 0's controllability of the wire w can be approximated as Equation (1) and Equation (2) based on this simulation.

$$C_1(w) = \frac{|V(w)|}{2^r} \tag{1}$$

$$C_0(w) = 1 - C_1(w) \tag{2}$$

Then the critical path tracing technique is performed to determine the *criticality vector* of each wire from the POs to the PIs. The criticality vectors of POs are all initialized to 1. This is because the change of all these bits definitely causes POs change. The criticality vector of the wire w is denoted as $Cr(w)$ and it can be obtained by using the rules as shown in Equation (3), Equation (4), and Equation (5). Equation (3) is for AND gate shown in Fig. 2(a), Equation (4) is for OR gate shown in Fig. 2(b), and Equation (5) is for NOT gate shown in Fig. 2(c).

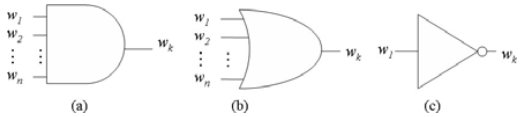

Fig. 2. (a)Backtracing rule for AND gate. (b)Backtracing rule for OR gate. (c)Backtracing rule for NOT gate.

$$\forall j \in \{1, 2, \cdots, n\} : Cr(w_j) = \left[\bigcap_{i=1, i\neq j}^{n} V(w_i) \right] \cap Cr(w_k) \tag{3}$$

$$\forall j \in \{1, 2, \cdots, n\} : Cr(w_j) = \left[\bigcap_{i=1, i\neq j}^{n} \overline{V(w_i)} \right] \cap Cr(w_k) \tag{4}$$

$$Cr(w_1) = Cr(w_k) \tag{5}$$

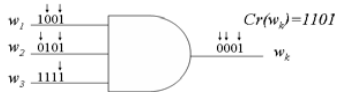

Fig. 3. An example to explain the backtracing rule for AND gate.

We use an AND gate example shown in Fig. 3 to explain why the backtracing rules for deriving the criticality vector is valid. The arrows drawn in the figure indicate the critical bits, which are 1 in the criticality vector. Suppose the criticality vector of the wire w_k is 1101, then three arrows are drawn above the 1^{st}, 3^{rd}, and 4^{th} bits. This means that the change of these critical bits will cause at least one PO change. Then we can use a chaining effect to show how to evaluate the criticality vector of the wire w_1. The change of the bits in the wire w_1, that causes the change of the wire w_k and these bits of the criticality vector in the wire w_k are also critical, will cause the change of the POs as well. The effect of one fanin can be propagated to gate output only when the other fanins of the gate are 1 (non-controlling value of AND gate). Therefore, the criticality vector of the wire w_1 can be obtained by the following operation.

$$Cr(w_1) = V(w_2) \cap V(w_3) \cap Cr(w_k) = 0101 \cap 1111 \cap 1101 = 0101$$

Next we give an example shown in Fig. 4 to demonstrate how backtracing rules proceed. Firstly, the vectors of the PIs are generated by the RPG. In this example, the parameter r of RPG is set to 2, thus 2^2-bit vectors are generated. Afterward, parallel pattern simulation is performed to evaluate the vector of each wire. Secondly, the bits in the criticality vector of the PO are all initialized to 1. Then backtracing rules are used to determinate the criticality vectors of each wire from the POs to the PIs. For instance,

$$\begin{aligned} Cr(g \to h) &= \overline{V(a \to h)} \cap \overline{V(f \to h)} \cap Cr(h) \\ &= 0101 \cap 1111 \cap 1111 = 0101 \end{aligned}$$

$$\begin{aligned} Cr(e \to g) &= V(c \to g) \cap V(d \to g) \cap Cr(g \to h) \\ &= 0011 \cap 0101 \cap 0101 = 0001 \end{aligned}$$

When encountering fanout nodes, OR operation is performed to collect the critical bits of fanout wires.

$$Cr(c) = Cr(c \to f) \cup Cr(c \to g) = 0100 \cup 0101 = 0101$$

After obtaining the criticality vector of each wire, the fault detection probability of a wire could be also evaluated. The probability of detecting *s-a-0* fault of a wire is the probability that the wire is 1 and it can simultaneously propagate the fault effect to at least one PO.

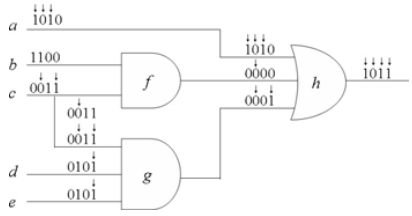

Fig. 4. An example of demonstrating backtracing rules.

We denote the *s-a-0* and *s-a-1* fault detection probability of wire w as $S_0(w)$ and $S_1(w)$. Hence the fault detection probability of wire w could be expressed as Equation (6) and Equation (7). Finally, the fault detection probability according to this set of parallel random patterns could be obtained. For instance in Fig. 4.

$$S_0(w) = \frac{|V(w) \cap Cr(w)|}{2^r} \tag{6}$$

$$S_1(w) = \frac{\left|\overline{V(w)} \cap Cr(w)\right|}{2^r} \tag{7}$$

C. Scoring

With the sampling rules, lots of sampling data could be collected to approximate the exact result. The probability density function (pdf) of 100,000 sample data about the fault detection probability of a certain wire in the benchmark C432 is shown in Fig. 5.

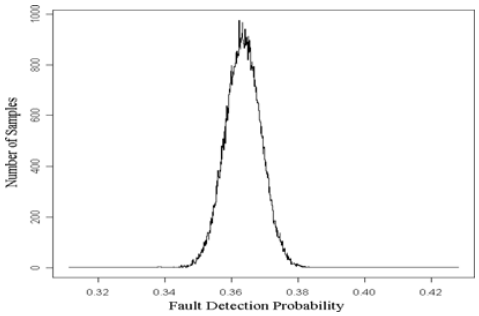

Fig. 5. The distribution of fault detection probability of a wire.

In Fig. 5, the x axis represents the fault detection probability of a wire in each sampling and the y axis represents the number of samplings that have this value. For example, (0.36, 750) represents that 750 out of 100,000 samplings are with the fault detection probability of 0.36. According to Fig. 5, the distribution of the samplings obviously shapes as a bell which indicates the distribution is a normal distribution. Therefore, an approximated result could be derived from averaging the sampling data. Note that the accuracy of our approach is strongly related to the amount of sampling data.

D. Error Estimation

We exploit the *Confidence Interval* to estimate the error of the fault detection probability of a wire as comparing with the true mean of the normal distribution. Suppose that we have N sampling data of the fault detection probability of a wire w, then we could compute the sample standard deviation of this wire labeled as $sd(w)$. And the parameter of the predefined confidence level is α as shown in Fig. 6. Then we can look up $t_{\frac{\alpha}{2}}$ from the t distribution table [10] with $(N-1)$ degrees of freedom. Hence, we have $(1-\alpha)\times 100\%$ confidence that the estimated error of the fault detection probability of w, $e(w)$, could be expressed as Equation (8) [10].

$$e(w) = \frac{t_{\frac{\alpha}{2}} \times sd(w)}{\sqrt{N}} \tag{8}$$

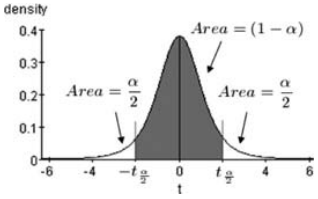

Fig. 6. Confidence level.

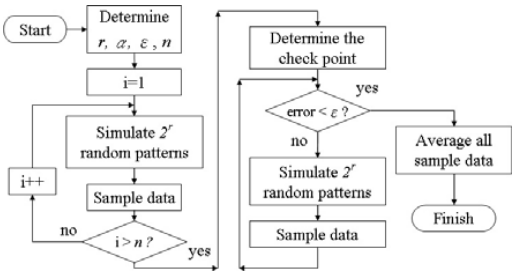

Fig. 7. The flow chart of our approach.

Consequently, for a desired error ϵ in the fault detection probability estimation, and for a given confidence level $(1-\alpha)\times 100\%$, we have to repeat the the simulation until

$$\frac{t_{\frac{\alpha}{2}} \times sd(w)}{\sqrt{N}} < \epsilon \tag{9}$$

Equation (9) is called the stopping condition of the Monte Carlo method. However, it is too time-consuming to check whether every wire stuck-at fault meets the stopping condition defined in Equation (9). As a result, a proper check point should be determined such that nearly all of the other wires would meet the stopping condition as well while the error of the checked wire satisfies the stop condition. We observe the error estimation formula in Equation (9) and realize that only the sample standard deviation $sd(w)$ of each wire differs from each other. Thus, we think that the wire with the largest standard deviation $sd(w)$ is the most inaccurate. Accordingly, several initial samplings are needed to find the wire with the largest standard deviation. This wire will be regarded as the check point. If the error of the check point satisfies Equation (9), the simulation is terminated.

E. Overall flow

The flow chart of our approach is shown in Fig. 7. At first, the parameter r of RPG, confidence level α, error ϵ, and the number of initial samplings n are specified. Then n samplings are applied to calculate the fault detection probability of each wire. These results are used to locate the wire with the largest standard deviation, and therefore this wire is selected as the check point. After n samplings, we can estimate the error by using Equation (9) and then terminate the simulation if the error is smaller than ϵ. If the error is still larger than or equal to ϵ, the simulation continues. This iterative process is ceased when the error of the check point is smaller than ϵ. In the end, we average all the sampling data on each wire to get the estimation of the fault detection probability of each wire.

IV. Experimental Results

The experiments use a set of *ISCAS'85* and *MCNC* benchmarks to evaluate the performance of our algorithm within SIS [14] environment. These benchmarks are in BLIF format. Each circuit is first optimized by using *script.boolean* script, then decomposed into AND and OR gates by using the command *tech_decomp -a 1000 -o 1000* in SIS. In the experiments, only single stuck-at fault detection

TABLE I

CPU TIME COMPARISON AMONG THE EXACT, TAIR, AND OUR APPROACH.

Circuits	# of gates	Total faults	CPU time (Sec.)		
			Exact	TAIR	Ours
C432	286	618	255	0.10	0.18
C499	567	1548	1115	0.32	0.42
C880	423	1222	713	0.18	0.33
C1355	682	1548	1100	0.31	0.44
C1908	770	1462	283	0.32	0.45
*C2670	1076	2232	780	0.40	0.86
C3540	1530	3760	7754	2.27	0.93
C5315	2447	5324	1249	0.84	1.62
*C6288	3540	9796	15623	1.60	2.78
*C7552	3281	7012	4894	1.70	3.20
***des**	**2260**	**10812**	**11886**	**7.30**	**3.98**
example2	212	954	14.03	0.22	0.37
i6	276	1314	23.15	0.50	0.55
i7	344	1710	39.34	0.87	0.71
*i8	637	3106	1053	1.90	0.95
*i9	332	1696	333	0.90	0.60
*i10	1450	6730	5612	2.90	2.11
pair	1022	4796	853	0.95	1.48
rot	444	2004	136.45	0.29	0.72
t481	414	2144	66.9	2.34	0.32
average CPU time ratio			***2338.39***	***1.14***	***1***

TABLE II

ERROR RATE DISTRIBUTION OF TAIR AND OUR APPROACH.

Circuits	0%	0∼5%	5∼10%	10∼20%	20∼30%	30%↑
	TAIR (%) / Ours (%)					
C432	14 / 10	5 / 80	3 / 1	17 / 2	9 / 1	52 / 6
C499	0 / 0	14 / 43	0 / 35	2 / 3	24 / 0	60 / 19
C880	8 / 0	17 / 86	11 / 6	23 / 6	11 / 0	30 / 2
C1355	0 / 1	15 / 41	0 / 36	2 / 4	24 / 0	59 / 19
C1908	0 / 1	13 / 83	5 / 7	15 / 3	13 / 4	54 / 2
*C2670	0 / 5	16 / 65	3 / 2	12 / 2	7 / 2	62 / 24
C3540	16 / 4	9 / 66	10 / 8	16 / 6	12 / 2	37 / 14
C5315	10 / 1	7 / 95	6 / 1	17 / 1	15 / 1	46 / 1
*C6288	1 / 1	2 / 99	1 / 0	2 / 0	5 / 0	88 / 0
*C7552	1 / 2	5 / 85	3 / 2	7 / 3	10 / 3	74 / 7
*des	0 / 0	30 / 97	8 / 3	15 / 0	6 / 0	41 / 0
example2	44 / 5	17 / 92	3 / 2	22 / 1	4 / 0	10 / 0
i6	62 / 0	13 / 100	20 / 0	5 / 0	0 / 0	0 / 0
i7	63 / 0	13 / 100	6 / 0	11 / 0	2 / 0	5 / 0
*i8	2 / 2	25 / 96	10 / 2	19 / 1	6 / 0	38 / 1
*i9	0 / 0	16 / 100	28 / 0	9 / 0	18 / 0	30 / 0
*i10	1 / 3	31 / 75	9 / 9	10 / 5	9 / 1	41 / 8
pair	15 / 0	17 / 96	14 / 2	20 / 1	8 / 0	26 / 1
rot	20 / 2	23 / 88	10 / 4	16 / 2	9 / 2	22 / 2
t481	17 / 5	2 / 41	3 / 5	4 / 4	15 / 4	58 / 41
average	***14 / 2***	***15 / 82***	***8 / 6***	***12 / 2***	***10 / 1***	***41 / 7***

probability is considered. The fault detection probability for each stuck-at fault is obtained by three different methods for comparison. The *exact* method uses the Binary Decision Diagram (BDD) [4] to obtain the exact result. The *TAIR* result is derived by the TAIR algorithm [6] provided by Chang et al. For our approach, we conduct some preliminary experiments to find the parameters needed in our method. It is suggested to set $r = 13$, $\alpha = 0.001$, $\epsilon = 0.005$ and $n = 10$. This means that our result has only 0.005 error with 99.9% confidence level. These three methods are implemented on a 1350 MHz Sun Fire V490 workstation with 8 GB memory. The experimental results are summarized in Table I and Table II.

In Table I, the first column shows the name of each circuit. The circuits marked with "*" mean that they cannot be represented by BDD. Therefore, the results of these circuits are approximated by the simulation of a large amount of patterns, specifically, 2^{20} patterns in our experiments. The second column shows the number of gates in each circuit, and the third column shows the number of total faults in each circuit. We compare the CPU time among the Exact, TAIR, and our approach. For example, *des* benchmark has 2260 gates and 10812 faults. The Exact and TAIR methods cost 11886 and 7.30 seconds, while ours spends 3.98 seconds. The last row shows the ratio of the average CPU time of these methods. According to Table I, we can know that the CPU time of the Exact method grows rapidly with the circuit size, but that of TAIR and our methods almost grow linearly to the circuit size.

Table II shows the detailed error rate distribution of TAIR and our approach. We divide the error rate into six levels, and they are 0%, $0\sim5\%$, $5\sim10\%$, $10\sim20\%$, $20\sim30\%$, and over 30%, respectively. The 0% column shows the percentage of fault detection probability of TAIR and our approach that are the same as the exact results. The $0\sim5\%$ column shows the percentage of TAIR and our results that are within $0\sim5\%$ error rate as being compared to the exact results, and so forth. The last row summarizes the distribution of average error rate of TAIR and ours. We observe that on avergae the error rate of our estimation is within 5% for $(2+82)\%$ of faults as comparing to the exact results. But the TAIR method has only $(14+15)\%$ of faults that are within 5% error rate. Furthermore, 41% of faults have more than 30% error rate by the TAIR.

V. CONCLUSIONS

From the application point of view, rapid testability analysis with small error is desired. In this paper, we propose a simulation-based statistic approach based on the Monte Carlo method for testability analysis. Experimental results show that our method efficiently acquires much more accurate results than TAIR on the fault detection probability. Also, our method provides accurate controllability and observability at the same time. As a result, our method could be used in the applications of DFT and ATPG.

REFERENCES

[1] M. Abramovici, P. R. Menon, and D. T. Miller, "Critical path tracing - an alternative to fault simulation," in *Proc. 20th Conf. on Design Automation*, pp. 214-220, 1983.

[2] V. D. Agrawal and S. C. Seth, "Probabilistic testability," in *Proc. Int. Conf. on Computer Design*, pp. 562-565, 1985.

[3] F. Brglez, "On testability of combinational networks," in *Proc. Int Symp. Circuits and Systems*, pp. 221-225, 1984.

[4] R. E. Bryant, "Graph-based algorithms for boolean function manipulation," *IEEE Trans. on Computers*, pp. 677-691, Aug. 1986.

[5] S. Chakravarty and H. B. Hunt III, "On computing signal probability and detection probability of stuck-at faults," *IEEE Trans. Comput.*, vol. 39, pp. 1369-1377, Nov. 1990.

[6] S. C. Chang, W. B. Jone and S. S. Chang, "TAIR: testability analysis by implication reasoning," *IEEE Trans. Computer-Aided Design.*, vol. 19, pp. 152-160, Jan. 2000.

[7] S. K. Jain and V. D. Agrawal, "Statistical fault analysis," *IEEE Design Test Comput.*, vol. 2, pp. 38-44, Feb. 1985.

[8] L. H. Goldstein, "Controllability/observability analysis of digital circuits," *IEEE Trans. Circuits Syst.*, vol. CAS-26, pp. 685-693, Sept. 1979.

[9] R. K. Gaede, M. R. Mercer, and B. Underwood, "Calculation of greatest lower bounds obtainable by the cutting algorithm," *in Proc. Int. Test Conf.*, pp. 498-505, 1986.

[10] I. R. Miller, J. E.Freund and R. Johnson, "Probability and statistics for engineers," Englewood Cliffs, NJ: Prentice Hall, 1990.

[11] K. P. Parker and E. J. McCluskey, "Probabilistic treatment of general combinational networks," *IEEE Trans. Comput.*, vol. C-24, pp. 668-670, 1975.

[12] J. Savir, G. S. Ditlow, and P. H. Bardell, "Random pattern testability," *IEEE Trans. Comput.*, vol. C-33, pp. 79-90, Jan. 1984.

[13] J. Savir, "Improved cutting algorithm," *IBM J. Res. Develop.*, vol. 34, no. 2/3, pp. 381-388, Mar.-May 1990.

[14] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May 1992.

[15] S. C. Seth, L. Pan, and V. D. Agrawal, "PREDICT: Probabilistic estimation of digital circuit testability," in *Proc. 15th Int. Fault-Tolerant Computer Symp.*, pp. 220-225, June 1985

[16] S. C. Seth and V. D. Agrawal, "An exact analysis for efficient computation of random-pattern testability in combinational circuits," in *Proc. 16th Int. Fault-Tolerant Computer Symp.*, pp. 318-323, 1986.

[17] O. Song and P. R. Menon, "Parallel pattern fault simulation based on stem faults in combinational circuits," in *in Proc. Int. Test Conf.*, pp. 706-711, 1990.

[18] H. C. Tsai, K. T. Cheng, C. J. Lin, and S. Bhawmik, "A hybrid algorithm for test point selection for scan-based BIST," in *Proc. Design Automation Conf.*, pp. 478-483, 1997.

[19] D. T. Wang, "An algorithm for the generation of test sets for combinational logic network," *IEEE Trans. Comput.*, Vol. C-24, No. 7, pp. 742-746, July 1975.

[20] http://www.swox.com/gmp