IEEE VLSI Test Symposium (VTS)
Monterey, CA, Apr. 27 -- Mai 1, 1997, Pages 426--431

On Optimizing BIST-Architecture by Using *OBDD*-based Approaches and Genetic Algorithms

Can Ökmen Martin Keim Rolf Krieger Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg i.Br., Germany

Abstract

We introduce a two-staged Genetic Algorithm for optimizing weighted random pattern testing in a Built-In-Self-Test (BIST) environment. The first stage includes the OBDD*-based optimization of input probabilities with regard to the expected test length. The optimization itself is constrained to discrete weight values which can directly be integrated in a BIST environment. During the second stage, the hardware-design of the actual BIST-structure is optimized. Experimental results are given to demonstrate the quality of our approach.*

1. Introduction

A built-in test based on pseudo random patterns generated by a linear feedback shift register (LFSR) is an alternative to external testing. Often the fault coverage achieved by a random test sequence is too low or the length of the test sequence which leads to an acceptable fault coverage is too large. One approach to improve the quality of built-in self test schemes is the optimization of input probabilities for weighted random pattern generation (WRPG). There is a trade-off that has to be optimized: A small number of weight vectors implies a large test application time, a small test application time requires a large number of weight vectors, leading to a high area overhead.

Mainly two different methods for computing weight vectors by optimizing fault detection probabilities are discussed in the literature [2, 9, 11, 16]. Since computing fault detection probabilities is NP-hard, these methods are based on approximated values [15, 18, 20] which may be very inaccurate and thus misdirect the optimization process. We use a numerical optimization technique based on exact fault detection probabilities computed by evaluating the *OBDD* of the fault detection function [13].

Experimental analysis has shown the applicability of our approach and the superior quality of weight vectors based on continuous (exact) weight values. But if the random test is to be implemented as a BIST-scheme, the weighted random patterns must be determined by rounding. However, this may lead to a significant deterioration of the quality of the results.

In this article, we overcome these disadvantages by using *Genetic Algorithms* (GA) instead of numerical methods. Although GAs generally will not yield global optima as well, they turn out to be the better heuristics in many applications. In particular, GAs seem to be predestined for discrete optimization as they do not demand objective functions to have special properties like continuity or differentiability. This merit of GAs allows us to use other than the classical objective functions for improving random BIST. Optimizing classical objective functions, e.g. those introduced in [9, 21], aims at the statistical improvement of random testability, but will not always provide satisfactory results concerning the actual test application time on-chip. We therefore propose an objective function φ_s which measures the quality of the BIST structure. This quality is given by the actual test length or fault coverage, using fault simulation with weighted random patterns generated by the BIST-structure. Thus, our optimizing strategy is as follows:

During the first stage of our GA, we optimize discrete input probabilities using the objective function φ_r presented in [9]. During the second stage, all other parameters affecting the BIST-structure, like feedback connections, will be optimized with respect to φ_s. The output of the GA will not just consist of optimized input probabilities but the complete hardware design of the BIST-structure.

Experimental results show, that our optimized, discrete weights are as good as the exact, continuous weights, concerning test length and fault efficiency. Moreover, using a space efficient BIST-realization based on the ideas of Hartmann [10] optimized by our GA's second stage, we achieve a fault efficiency of at least 99.95% with only two weight vectors for all circuits considered.

This paper is structured as follows: Section 2 presents some definitions and basic concepts. Sections 3 discusses some aspects of random pattern generation on chip. Additionally it shows our target hardware. The optimization of both, input probabilities and BIST-structure using *Genetic Algorithms*, is described in Section 4. In the next section experimental results are given, which emphasize the efficiency and the importance of our approach. Finally, we draw some conclusions.

2. Preliminaries

We use the gate-level realization of a combinational circuit C and the (single) stuck-at fault model. We denote the set of all stuck-at faults of C by F. The detection function of a fault f is defined by $T_f : \mathbb{B}^n \rightarrow \mathbb{B}$ with $T_f(x) = 1$ iff x is a test for f, where n denotes the number of primary inputs of C.

If we consider a Boolean function $g : \mathbb{B}^n \rightarrow \mathbb{B}$ depending on Boolean variables $x_1, \ldots, x_k$, we also interpret these variables as independent random variables defined on the probability space $\Omega = \mathbb{B}^n$. The probability distribution that defines the probability of such a random variable assuming the value 1 is denoted by P_r. We abbreviate these probabilities by $q_i = P_r(x_i = 1)$ and denote q_i, $1 \leq i \leq n$ as *input probability* or *weight value* (*weight* for short). $\vec{q} = \{q_1, ..., q_n\}$ represents the so-called *weight vector*. $P_r(g = 1)$ denotes the probability that the Boolean function g assumes the value 1 for a random argument depending on P_r.

As is well-known, each Boolean function can be represented by a (Reduced) Ordered Binary Decision Diagram (*OBDD*) [7]. If the *OBDD*-representation *OBDD*(g) of a Boolean function g is given, the probability $P_r(g = 1)$ can be determined in linear time [3, 12]. The detection probability of a fault f in a combinational circuit expresses the probability that f will be detected by a randomly selected input vector according to the probability distribution P_r. We denote the detection probability of f as $\delta(f) = P_r(T_f = 1)$.

The *OBDD* representation of the fault detection function can efficiently be constructed [14]. For our application, we developed an accelerated technique for the evaluation: The representation of the fault detection functions as an *OBDD* allows only a recursive computation of the fault detection probabilities. An acceleration can be achieved by sorting the nodes of the *OBDD*s in a topological order and storing the nodes in a list. This has to be done separately for each subset only once. Afterwards, the computation of fault detection probabilities can be done iteratively by traversing the lists. This iterative computation is considerably faster than recursive computation.

3. Aspects of Random BIST

In this section we describe the concepts used in our approach. We start with the definition of discrete weight sets. Furthermore, we introduce our target hardware realization [10]. It turns out, that a given weight vector does not define this BIST-structure in all aspects. For example the characteristic polynomial of the LFSR is not specified. The remaining information is computed using a genetic algorithm, which is introduced in the next section.

If a random test is to be implemented as a BIST-scheme, the random patterns must not be based on exact (continuous) weights. The hardware design of most of the LFSR-based random pattern generators restricts the weights that can be used to those contained in the following set. This set we denote as (a, b)-valued discrete weight set, where $a \geq 2$ and $b \geq 0$:

$$W_{a,b} := \{0, \frac{1}{2^{a+b}}, \frac{1}{2^{a-1+b}}, ..., \frac{1}{2^{2+b}}, \frac{1}{2}, 1 - \frac{1}{2^{2+b}}, 1 - \frac{1}{2^{3+b}}, ..., 1 - \frac{1}{2^{a+b}}, 1\}$$

Obviously, the definition implies that $|W_{a,b}| = 2 \cdot a + 1$.

In [10, 19], procedures for weight optimization are presented which take into account these restriction. The results show, that the use of discrete weights usually reduces the quality of the continous weight vectors in proportion to the number of weights permitted.

3.1. WRPG On-Chip

Using LFSRs [2], only unweighted pattern can be generated. If both unweighted and weighted random patterns are to be applied an additional circuitry for switching from unweighted to weighted random pattern generation is needed. In [10], a space efficient implementation with this property is introduced. Since we optimize this hardware in our experiments, we briefly describe its structure. An outline is given in Figure 1. This BIST-structure contains an LFSR R_1 of the length l_1 and two ordinary shift registers R_2 and R_3 of the lengths l_2 and l_3 respectively. The outputs of the elements in R_3 are connected to those primary inputs of the circuit which according to the given weight vector shall be supplied with constantly 0 or constantly 1 during weighted random pattern generation. Analogously, the elements in R_2 lead to primary inputs to be set to 1 with the probability of $\frac{1}{2^{2+b}}$ or $1 - \frac{1}{2^{2+b}}$, respectively. Primary inputs connected with elements in R_1 will be set to 1 with uniformly distributed probability.

During the first test phase, *Test Phase Select* (TPS) is set to 0, causing R_3 to be provided with the constant value 0. R_2 is fed with the logical AND of values derived from XOR gates, connected to storage elements $r_1, ..., r_{4+2b}$ in R_1. These $(2 + b)$ 2-input-XOR gates help to reduce correlations between elements of R_1 and R_2 [10].

After applying a certain number of weighted random patterns, TPS is switched to 1, causing all values at the output of LFSR R_1 to be shifted through R_2 and R_3. Note, that if there is already a shift register structure on chip, the hardware overhead reduces to a few basic gates.

The hardware is to a great degree determined by a given weight vector $\vec{q}$ with $q_i \in W_{2,b}$, $1 \leq i \leq n$. Therefore, we refer to it as a $(2, b)$-valued Weighted Random Pattern Generator ($(2, b)$-valued WRPG for short). Nevertheless, results obtained performing a weighted random built-in self test often do not come up to expectations, even if the input probabilities they are based on are of high quality. This can be explained by problems related to the following questions:

Which polynomial should be selected for R_1?
Where should the connections be placed in R_1 for R_2?
How to initialize the BIST-structure?

Often it is impossible to solve these problems efficiently by hand, so an algorithm, that finds a good solution, is strongly

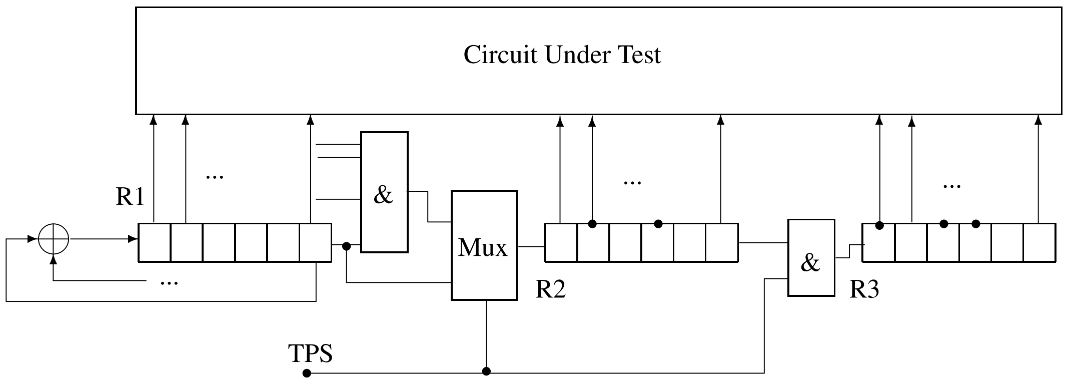

Figure 1. BIST-structure of Hartmann.

necessary. These problems get even worse, if only a small part of the maximum period (m-sequence) of an LFSR is used for weighted random pattern generation. To overcome these problems, we let a genetic algorithm search for solutions.

4. Genetic Algorithm

Genetic Algorithms (GA) [8] represent an alternative to classical numeric optimization methods, e.g. the well known *hill-climbing*-procedure that demands an objective function that is differentiable and strictly convex [21]. Using discrete weight values will cause the objective function to lose the property of being continuous, so it will no longer be suitable for numeric optimization. In such cases, probabilistical approaches like GAs may help.

We present a two-staged GA for optimizing (a, b)-valued WRPGs with regard to the quality of the pseudo random tests they produce.

Definition 1 *With* $q_j \in W_{2,b}$, $t_j \in \mathbb{B}$, $1 \leq j \leq n$, $v_j \in \{1, ..., l_1\}$, $1 \leq j \leq 4 + 2b$ *and* $r_j \in \{1, ..., l_1\}$, $1 \leq j \leq 2$, *we define a* $(2, b)$*-valued individual as a vector*

$$\vec{c} = q_1, ..., q_n, t_1, ..., t_n, v_1, ..., v_{4+2b}, r_1, r_2$$

In $\vec{c}$, $\vec{q}$ represents a $(2, b)$-valued discrete weight vector, $\vec{t}$ an initial state of a $(2, b)$-valued WRPG, $\vec{v}$ denotes the storage elements whose outputs feed the XOR/AND gates to perform weighting and $\vec{r}$ denotes the storage elements which have a feedback connection. Note that the design of a $(2, b)$-valued WRPG is uniquely determined by a $(2, b)$-valued individual. In other words, we encoded not only the weight vector into an individual, but also the complete description of the target hardware. By this, the best individual at the end of the algorithm not only determines an optimized weight vector, but also a hardware realization of that vector. Additionally, this hardware generates test patterns that detect more faults than any other realization, based on any individual ever created.

To minimize hardware costs, we restrict $\vec{r}$ to a pair of storage elements. Hence, the characteristic polynomial associated with LFSR R_1 will be of the shape $f(x) = 1 + x^{r_1} + x^{r_2}$ for a $(2, b)$-valued WRPGs defined by an $(2, b)$-valued individual. We denote a set of N $(2, b)$-valued individuals as a $(2, b)$-valued population of size N.

4.1. Objective Functions

Within the scope of our two-staged GA, we intend to optimize individuals with respect to the minimization of objective functions $\varphi_r(\vec{c})$ and $\varphi_s(\vec{c})$. In the following, the *fitness* of an individual $\vec{c}$ denotes the value computed by $\varphi_r(\vec{c})$ or $\varphi_s(\vec{c})$.

During the first stage, individuals are optimized with respect to the expected test length $\bar{L}(\vec{c})$. We approximate $\bar{L}(\vec{c})$ by φ_r [9], with $r = 1$:

$$\bar{L}(\vec{c}) \approx \varphi_r(\vec{c}) := \sum_{f \in F} \left(\frac{1}{\delta(f)}\right)^r.$$

With that formula, only the weight vector $\vec{q}$ is affected by the optimization.

During the second stage, individuals are optimized with respect to the actual fault coverage yielded by weighted random pattern sequence of length L. We set a maximum number of pattern to $\tilde{L} = 2^{14} = 16384$ ($\tilde{L} = 5000$ for the combinational circuits). $\tilde{L}$ is doubled if no individual yielding complete fault coverage is created after a certain number of iterations. When $\tilde{F}$ denotes the set of detected faults, the fitness of each individual is computed by the objective function:

$$\varphi_s(\vec{c}) = L \cdot (2 - \frac{|\tilde{F}|}{|F|})$$

If no faults remain undetected, $\varphi_s(\vec{c})$ equals the test length L. Otherwise $L = \tilde{L}$ is weighted with a (penalty) factor. In particular, φ_s guarantees a further improvement even if all individuals yield a complete fault coverage. In this case optimization continues with respect to the test length.

4.2. Genetic Operations

Genetic operations may be divided into *Mutation* and *Crossover*. Both types of operations may vary depending on the problem domain. Each genetic operation is performed with a given probability and may produce individuals which have a higher fitness than their ancestors. The opposite case, though, is also possible. For selection we use the $(N + \lambda)$-method in combination with (N, μ)-selection [1]. The λ best individuals of the preceding population are added to the current population. From the resulting set of $N + \lambda$

	[13]			$W_{2,b}$			$W_{3,b}$			$W_{4,b}$			$W_{5,b}$		
Circ.	*TL*	*UF*	*WS*	*TL*	*UF*	*WS*	*TL*	*UF*	*WS*	*TL*	*UF*	*WS*	*TL*	*UF*	*WS*
c2670	4703	0	-	9344	0	$W_{2,2}$	6624	0	$W_{2,2}$	4960	0	$W_{2,0}$	6688	0	$W_{2,0}$
c7552	3422	0	-	12.6k	0	$W_{2,2}$	4992	0	$W_{2,1}$	3200	0	$W_{2,1}$	3200	0	$W_{2,1}$
s9234.1	20096	3	-	36.1k	15	$W_{2,0}$	36.1k	2	$W_{2,0}$	36.1k	1	$W_{2,0}$	36.1k	1	$W_{2,0}$
s13207.1	10.7k	0	-	14.7k	0	$W_{2,1}$	14.7k	0	$W_{2,1}$	13.4k	0	$W_{2,1}$	13.4k	0	$W_{2,1}$
s15850.1	17.3k	0	-	32.6k	0	$W_{2,1}$	32.3k	0	$W_{2,0}$	18.5k	0	$W_{2,0}$	18.5k	0	$W_{2,0}$
s38584.1	21.1k	0	-	36.1k	5	$W_{2,0}$	36.1k	1	$W_{2,0}$	36.1k	1	$W_{2,0}$	28.4k	0	$W_{2,0}$

Table 1. Impact of rounding exact weights.

individuals only the best $\mu \leq N$ constitute a set of privileged individuals on which genetic operations may be performed. From this set pairs of individuals (parents) are chosen at random to be duplicated. With given probabilities the corresponding genetic operations are iteratively applied to the duplicates (clones) converting them into new individuals (children). An iteration step ends after N children have been created. The children then represent the new population. This selection method guarantees that the best individuals will never be lost to the reproduction process.

During the first stage of the GA the operations are applied only on the weight vector. In the second stage the *mutation* operator may be applied to all vectors an individual consists of, whereas the *crossover* operator is restricted to the weight vector only.

1-time-crossover: A cut position i, $1 \leq i \leq n$ which divides the weight vector of both clones into two parts is determined at random. Two children are created by swapping the second parts of the weight vectors.

2-time-crossover: Two cut positions $i, j, i \neq j$ dividing both weight vectors into three parts are chosen at random. Two children are created by swapping the middle parts of the weight vectors.

Mutation: With probabilities $\frac{P_{mut}(\vec{q})}{n}$, $\frac{P_{mut}(\vec{t})}{n}$ and $\frac{P_{mut}(\vec{v})}{2+b}$ each component of the corresponding vector is changed to a value chosen at random from $W_{2,b}$, $I\!B$ and $\{1, ..., l_1\}$, respectively. Hence, $P_{mut}(\vec{q})$, $P_{mut}(\vec{t})$ and $P_{mut}(\vec{v})$ denote the expected number of components to be changed in the corresponding vectors of each chosen individual in each iteration step. We denote *mutation* with regard to $\vec{q}$, $\vec{t}$, $\vec{v}$ and $\vec{r}$ as $mut(\vec{q})$, $mut(\vec{t})$, $mut(\vec{v})$ and $mut(\vec{r})$ respectively. To avoid the generation of unvalid individuals, with respect to the target hardware, we guarantee that $v_1, ..., v_{4+2b}, r_1, r_2 \leq l_1$.

Using the listing of primitive polynomials presented in [2], we were able to perform two different kinds of heuristics concerning $mut(\vec{r})$.

H1: Restrict the selection of polynomials to those, which define a primitive polynomial $f(x) = 1 + x^{r_1} + x^{r_2}$ of degree r_2. If no such polynomial exists, select the next smaller existing. Here, $P_{mut}(\vec{r})$ gives the chance, that a given polynomial is exchanged by another one (or by chance, the same one).

	GA, first stage		
Circ.	*TL*	*UF*	*WS*
c2670	5984	0	$W_{2,2}$
c7552	4736	0	$W_{2,2}$
s9234.1	36.1k	7	$W_{2,1}$
s13207.1	9184	0	$W_{2,0}$
s15850.1	17.8k	0	$W_{2,2}$
s38584.1	16.8k	0	$W_{2,0}$

Table 2. Improvement by the GA's first phase.

H2: Set $r_2 := l_1$ and select r_1 freely, only considering the length of R_1, as mentioned above. Here, $P_{mut}(\vec{r})$ gives the chance, that r_1 is modified. Note, that $mut(\vec{q})$ changes r_2 indirectly, by varying the length of R_1.

Heuristic H1 promotes the use of random sequences of high quality and therefore speeds up the convergence of the GA. In contradiction to H1, heuristic H2 does not force the exclusive use of m-sequences but considers a greater number of polynomials to be applied on the individuals. Here, the posibility of finding non-primitive polynomials yielding good results is given.

5. Experimental Results

The implementation was done in C++ using the BDD-package of Brace et al. [4]. For the experiments we considered some of the ISCAS benchmark circuits [5, 6] that contain a lot of random pattern resistant faults (hard faults). In case of the sequential circuits only their combinational parts were considered. These hard faults are identified heuristically by a fault simulation with equiprobable random pattern sequences of the lengths 1024 for the combinational circuits and 8196 the sequential circuits, respectively. Additionally, we excluded redundant faults from the fault list, identified by our tool.

In our GA, we use a very small population of $N = 4$ individuals, $(4+4)$-concept and $(4, 4)$-selection. This combination produces a high selection pressure and aims at a fast convergence. To prevent the *bottleneck-effect*, i.e. the occurance of most similar or even identical individuals within a population, we use a very high mutation rate concerning the weight vector ($P_{mut}(\vec{q}) = 1.0$). In doing so, we promote the creation of new weight combinations within the weight vectors. Of course, this also leads to an increased deterioration of some individuals in each iteration step. However, practice has shown that this effect does not affect the procedure in a negative way. The crucial point in this connection is that (N, μ)-selection will eliminate low-fitness individuals while the best individuals are protected through the $(N + \lambda)$-concept.

At the beginning of a GA-run the initial population is generated randomly. To get starting points that are not too bad we exclude the constant values 0 and 1 from being assigned to components of $\vec{q}$. The initial population is further improved by applying numerical *hill-climbing* [13] with respect to a subset $I_j \subset I$ of inputs to the weight vector of each of the μ fittest individuals, whereby $|I_1| = ... = |I_\mu| = \lceil \frac{n}{\mu} \rceil$ and $I_1 \cup ... \cup I_\mu = I$. The resulting continuous values are rounded with respect to the chosen $(2, b)$-valued weight set afterwards.

For the first stage of the GA, it shows that a number of $M_1 = 6400$ iterations is sufficient to obtain discrete weight vectors of high quality for all circuits considered. The most recent population of the first stage will form the initial population of the second stage of the GA. If the weight vectors started with are sufficiently good, the second stage may already be terminated after a number of $M_2 = 100$.

Notation of the tables: *WV* denotes the number of weight vectors, *WS* is the number of weight sets, the test length is labeled with *TL*, and number of undetected faults is *UF*

5.1. Rounding of Exact Weights

Table 1 shows the results obtained after rounding the optimized exact input probabilities of [13] with respect to weight sets $W_{a,b}$, $a \in \{2, 3, 4, 5\}$, $b \in \{0, 1, 2\}$ and the hardware of [19]. The second column gives the results obtained using exact weights [13]. From all weight sets of identical cardinality, those yielding the best result were considered. Obviously, the results become worse the fewer weights are used. We can improve the results concerning the weight set $W_{2,b}$ of Table 1 by using the first stage of the described genetic algorithm. In Table 2 the results of these runs are given. Note, that TL includes the number of equiprobable random patterns. The results confirm the advantage of using GAs. For instance, in the case of circuit s38584.1 the rounding of exact weights leads to a significant deterioration of the random testability[1]. However, applying the GA to circuit s38584.1 yields a result which is even better than the result obtained using exact weights. In Table 3, our best results are compared with those obtained in [10, 11, 17]. The high quality of our results is evident, especially considering that the other approaches use a large number of weight vectors, in particular for circuit c7552.

Note that the approaches of [11, 17] can generally not be used in combination with a $(2, b)$-valued WRPG, as the latter restricts the range of applicable weight vectors to the equiprobable weight vector and exactly one additional $(2, b)$-valued weight vector, $b \geq 0$.

[1] This circuit originally was not considered in [13]

5.2. Evolution of $(2, b)$-valued WRPGs

Hartmann's approach requires a considerably smaller hardware-overhead than all other hardware approaches described so far. So our goal is to verify our results optimized according to φ_r and applied to [19] with an $(2, b)$-valued WRPG. Thereby the search space of the GA increases considerably. Experiments show that it is necessary to have a good population as a starting point.

The results of the GA's second step are given in Table 4. TLW denotes number of weighted random patterns, whereas TLU denotes the number of unweighted random patterns. The maximal number of unweighted pattern is set to five times the number of weighted pattern. The notation in column UF means "number of undetected hard faults / number of undetected non hard faults".

As can be seen, for all circuits considered, we receive a good result either with heuristic H1 or with heuristic H2: For each circuit we detect all target faults (hard faults). Considering all non redundant faults 100% or almost 100% are detected by the generated BIST-structure.

A comparison of the results in Table 4 with those of others, Table 3, is inadequate, since there are more than two weight vectors used. Therefore their results do not match the hardware-requirements. On the other hand, we easily outperform the results of [10], given in Table 3.

6. Conclusions

We have considered the problem of optimizing weighted random pattern test in a BIST environment. Combining a genetic approach together with a OBDD-based methods, we were able to obtain a highly efficient BIST-scheme with almost no area penalty.

In a first stage of the Genetic Algorithm high quality discrete input probabilities are computed based on *exact* fault detection probabilities.

To efficiently realize the discrete input probabilities in hardware, a BIST-scheme for the realization of (only) two weight sets is optimized in the second phase of the GA. Our experiments demonstrate, that patterns derived from this hardware result in fault efficiencies of (nearly) 100%.

References

[1] T. Bäck. *Evolutionary Algorithms in Theory and Practice*. Oxford University Press, 1996.

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

[3] M. Blum, A.K. Chandra, and M.N. Wegman. Equivalence of free boolean graphs can be decided probabilistically in polynomial time. *Information Processing Letters*, 10:80–82, 1980.

[4] K.S. Brace, R.L. Rudell, and R.E. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[5] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Int'l Symp. Circ. and Systems*, pages 1929–1934, 1989.

Circ.	ours			[11]			[17]			[10]		
	WV	*TL*	*UF*	*WV*	*TL*	*UF*	*WV*	*TL*	*UF*	*WV*	*TL*	*UF*
c2670	2	4960	0	5	3840	0	6	7552	0	2	16k	0
c7552	2	3200	0	7	6400	0	21	18.8k	0	2	256k	0
s9234.1	2	36.1k	1	-	-	-	9	23.1k	0	-	-	-
s13207.1	2	9184	0	-	-	-	3	7532	0	-	-	-
s15850.1	2	17.8k	0	-	-	-	5	14.5k	0	-		-
s38584.1	2	16.8k	0	14	18.9k	0	3	14.5k	0	-	-	-

Table 3. Comparing our best results with the literature.

Circ.	GA, second stage											
	H1						H2					
	WV	*WS*	*UF*	*TLW*	*TLU*	*FE*	*WV*	*WS*	*UF*	*TLW*	*TLU*	*FE*
c1908	2	$W_{2,1}$	0/5	1005	1379	98.72	2	$W_{2,1}$	0/0	967	5416	**100.00**
c2670	2	$W_{2,2}$	0/0	2261	897	**100.00**	2	$W_{2,2}$	0/0	2496	944	100.00
c3540	2	$W_{2,0}$	0/10	757	1642	99.70	2	$W_{2,0}$	0/0	1102	2728	**100.00**
c7552	2	$W_{2,2}$	9/0	10000	7110	99.88	2	$W_{2,2}$	0/1	3828	1140	**99.99**
s9234.1	2	$W_{2,1}$	0/3	28919	29436	**99.95**	2	$W_{2,0}$	7/0	32768	43176	99.89
s15850.1	2	$W_{2,2}$	0/1	13974	59715	99.99	2	$W_{2,2}$	0/0	12711	25556	**100.00**
s38584.1	2	$W_{2,0}$	0/13	3783	17387	99.96	2	$W_{2,0}$	0/2	8334	40660	**99.99**

Table 4. Determine the BIST-structure by the GA's second phase.

[6] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational circuits and a target translator in fortran. In *Int'l Symp. Circ. and Systems, Special Sess. on ATPG and Fault Simulation*, pages 663–698, 1985.

[7] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[8] D.E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning*. Addision-Wesley Publisher Company, Inc., 1989.

[9] J. Hartmann. On numerical weight optimization for random test. In *European Conf. on Design Automation*, pages 223–230, 1993.

[10] J. Hartmann and G. Kemnitz. How to do weighted random testing for BIST? In *Int'l Conf. on CAD*, pages 568–571, 1993.

[11] R. Kapur, S. Patil, T.J. Snethen, and T.W. Williams. Design of an efficient weighted random pattern generation system. In *Int'l Test Conf.*, pages 491–500, 1994.

[12] R. Krieger. PLATO: A tool for computation of exact signal probabilities. In *VLSI Design Conf.*, pages 65–68, 1993.

[13] R. Krieger, B. Becker, and C. Ökmen. OBDD-based optimization of input probabilities for weighted random test. In *Int'l Symp. on Fault-Tolerant Comp.*, pages 120–129, 1995.

[14] R. Krieger, B. Becker, and R. Sinković. A BDD-based algorithm for computation of exact fault detection probabilities. In *Int'l Symp. on Fault-Tolerant Comp.*, pages 186–195, 1993.

[15] B. Krishnamurthy and I.G. Tollis. Improved techniques for estimating signal probabilities. *IEEE Trans. on Comp.*, 38(7):1041–1045, 1989.

[16] I. Pomeranz and S.M. Reddy. 3-weight pseudo-random test generation based on a deterministic test set. *IEEE Trans. on Comp.*, pages 1040–1049, 1993.

[17] B. Reeb and H.-J. Wunderlich. Deterministic pattern generation for weighted random pattern testing. In *European Design & Test Conf.*, pages 30–36, 1996.

[18] J. Savir, G.S. Ditlow, and P.H. Bardell. Random pattern testability. *IEEE Trans. on Comp.*, 33:79–95, 1984.

[19] J.A. Waicukauski, E. Linbloom, E.B. Eichelberger, and O.P. Forlenza. A method for generating weighted random test patterns. *IBM J. Res. Develop.*, 33(2):149–161, 1989.

[20] H.-J. Wunderlich. PROTEST: A tool for probabilistic testability analysis. In *Design Automation Conf.*, 1985.

[21] H.-J. Wunderlich. On computing optimized input probabilities for random tests. In *Design Automation Conf.*, pages 392–398, 1987.