How to do weighted random testing for BIST?

Joachim Hartmann
Fachbereich Informatik
Universität des Saarlandes
66041 Saarbrücken

Günter Kemnitz
Institut für Technische Informatik
Technische Universität Dresden
01062 Dresden

Germany

Abstract

In this paper, a strategy is proposed which takes into account all *aspects of weighted random testing for BIST. Our approach arises from results concerning the impact of weight rounding and a new combination of known techniques like coupling unweighted and weighted pattern generation, basing weight calculation on a precomputed test [2, 6], numerical maximization of pattern coverage [4], GURT-like hardware implementation [10], and avoiding auto-correlations.*

As an empirical evaluation, we examined the only random pattern resistant ISCAS 85 benchmarks c2670 and c7552. For these circuits, 100% *fault coverage was achieved after a total of* 16,000 *and* 256,000 *patterns, respectively. The hardware overhead compared to a pure random test is less than* 2.5%.

1 Introduction

Due to their low hardware costs built-in self-tests (BISTs) based on random patterns are very attractive. However, some circuits contain random pattern resistant faults which cannot be detected after a reasonable amount of time. To overcome this problem, the following strategies have been proposed: test point insertion [3], addition of deterministically computed patterns or of a random-like sequence containing such patterns [1, 7], and *weighted* random testing, which means that the primary inputs (PIs) are given individual probabilities (*weights*) of being 1 [8, 9]. When choosing one of these strategies one has to take into account the design effort (including calculations for test point location, test vector computation, weight optimization, ...), hardware overhead, test length, and fault coverage. The decision which solution is to be adopted depends on the circuit and on how priorities are given to the above factors. Nevertheless, the weighted random approach, especially if it couples unweighted and weighted pattern generation, is promising in many cases.

Previous papers dealing with weighted random testing mainly focussed on single aspects of the issues mentioned above: Fast heuristic optimization procedures were suggested in [6, 2]. But, compared to numerical methods, they are inferior with respect to fault coverage and test length (cf. [4]). A numerical optimization achieving excellent fault coverages for one weight set was given in [9]. However, this approach consumes much CPU time since it requires computation (or approximation) of detection probabilities. Numerical optimizations can be accelerated if they are based on precomputed tests. In this case a combination of unweighted and weighted pattern generation should be adopted as it has been pointed out in [4]. Nevertheless, there remains the task of producing weighted random patterns on-chip. Efficient hardware generators have been developed in [10, 6], but, with a reasonable amount of hardware, they are not able to realize precisely those input probabilities calculated by the weight optimization. Therefore, usually, optimal weights with respect to unconstrained optimizations have to be rounded to values that can be made feasible. Of course, this may result in solutions that are suboptimal for the constrained problem.

In this paper, we present a strategy which takes into account all these aspects. Our approach arises from results concerning the impact of weight rounding and combines elements of known techniques for weight optimization and implementing weighted random pattern generators. As an empirical evaluation of our method, we examined c2670 and c7552, the only random pattern resistant ISCAS 85 benchmark circuits. For both circuits, 100% fault coverage was achieved. The additional circuitry required an overhead of less than 2.5% compared to the implementation of a pure random test.

2 Weight optimization

Following [6, 2, 4], we will apply a two-phased test. In the first phase, unweighted random patterns are used to detect well random testable faults. From test vectors, which are calculated by ATPG for the remaining faults, input probabilities for the second phase are derived. As it has been proposed in [4], this can be done by a numerical optimization whose objective is to maximize the expected coverage of these patterns.

For a formal description, we assume that n is the number of PIs and m is the number of calculated test patterns. Let $t^i = (t^i_1, \ldots, t^i_n) \in \{0, 1, *\}^n$ ($*$ means don't care) be the ith calculated test pattern and p_i be the probability that a weighted random input combination matches[1] t^i. If q_j is the probability that the jth PI has the logical value one, it holds the following identity:

$$p_i = \prod_{j, t^i_j \neq *} (q_j \cdot t^i_j) + (1 - q_j) \cdot (1 - t^i_j).$$

Let N be the number of weighted random patterns to be used in phase 2. Then the *expected pattern uncoverage for N* is defined by

$$\overline{U^N} := \frac{1}{m} \cdot \sum_{i=1}^{m} (1 - p_i)^N.$$

$\overline{U^N}$ equals the expected percentage of t^i's not matched by N random patterns. $\overline{U^N}$ is a convex function of each single q_j. Thus, it is well suited for a downhill climbing which takes descents along coordinate axes since for these directions the line search problem can be solved numerically (e.g. by golden section search) (cf. [4]).

As a matter of fact, q_j does not require any optimization, if for all pattern indices i, it is $t^i_j = *$ ($\Rightarrow q_j \leftarrow 0.5$), or $t^i_j = * \vee t^i_j = 1$ ($\Rightarrow q_j \leftarrow 1$), or $t^i_j = * \vee t^i_j = 0$ ($\Rightarrow q_j \leftarrow 0$). It is crucial to our approach that such situations occur often since they allow a cheap hardware implementation. For c2670 (c7552), where $n = 157$ (206), we calculated $m = 184$ (185) different patterns which allowed 50 (25) 0.5-, 2 (22) 1-, and 43 (50) 0-assignment for the q_j's.

In our experiments, we observed that the optimization procedure described above calculated many weights which are very close to 0 and 1. Such weights, especially if they have an irregular structure, require high hardware implementation costs. For that reason, we restricted the line search of the descending procedure to computing only weights from $[2^{-k}, 1 - 2^{-k}]$, where k is a small non-negative integer. Input probabilities lying outside this interval after an unconstrained optimization ("$k = \infty$") then accumulate to the values 2^{-k} and $1 - 2^{-k}$. As a consequence the number of different weights is reduced. Moreover, random bits with these biases can easily be taken from the output and the inverted output of an k-input AND gate which is fed by uniformly distributed random bits.

	c2670		*c7552*	
	PC	*FC*	*PC*	*FC*
$k = \infty$	65.52%	99.61% (10)	58.68%	99.83% (13)
$k = 5$	65.19%	99.96% (1)	58.59%	99.99% (1)
$k = 4$	64.17%	100% (0)	58.01%	99.97% (2)
$k = 3$	55.66%	99.69% (8)	53.37%	99.99% (1)

Table 1: Optimization restricted to $[2^{-k}, 1 - 2^{-k}]$

		continuous	3-valued
c2670	*PC*	64.17%	60.49%
	FC	100% (0)	99.85% (4)
c7552	*PC*	53.37%	52.17%
	FC	99.99% (1)	99.99% (1)

Table 2: Continuous vs. 3-valued optimization

Results obtained in this way are shown in Table 1. For c2670 and c7552, it gives pattern coverages (PC), overall fault coverages[2] (FC), and in brackets the total number of undetected testable faults. They were achieved by applying 9000 (c2670) and 64000 (c7552) pure random patterns and the same number of weighted input vectors. As it can be seen in Table 1, pattern coverage monotonically increases in k, while fault coverage becomes maximal for $k = 4$ (c2670) and $k = 3, 5$ (c7552).

As a further step to reducing the number of different weights, we forbade all input probabilities different from 2^{-k}, 0.5, and $1 - 2^{-k}$. To do that, we replaced the golden section search for one-dimensional optimization by a procedure taking that value $\{2^{-k}, 0.5, 1 - 2^{-k}\}$ which minimizes the expected pattern uncoverage. In Table 2, results for this 3-valued optimization are given. The same pattern numbers as in Table 1 were applied. k was chosen as 4 for c2670 and as 3 for c7552. It follows that losses in fault coverage are only slight.

[1] A pattern $t = (t_1, \ldots, t_n)$ is said to match t^i if for all j it holds: $t^i_j \neq * \Rightarrow t_j = t^i_j$.

[2] The fault coverage was taken from all non-redundant stuck-at faults.

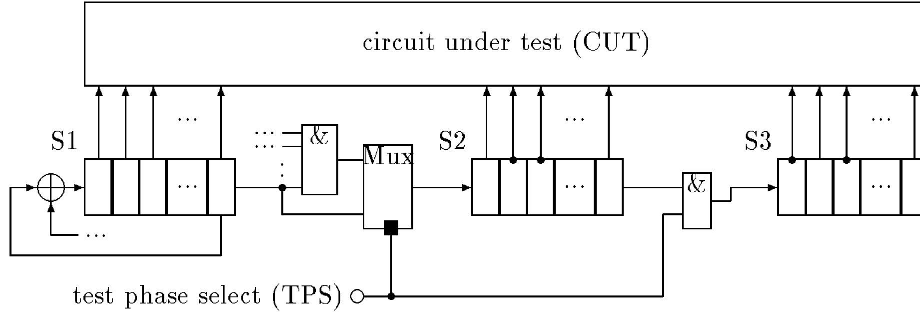

Figure 1: Proposed test pattern generator structure

3 Realization of the Test Pattern Generator

The special structure of weights calculated by the procedure described in the previous section allows an efficient implementation of the pattern generator which is similar to, but conceptually simpler than Wunderlich's GURT [10]. The generator is shown in Figure 1. It consists of three shift registers S1, S2, S3 with lengths l_1, l_2, and l_3. S1 has a linear feedback defined by a primitive polynomial in order to produce a maximum length sequence. As a consequence S1's outputs are one with probability $\frac{1}{2}$. The control signal *test phase select* (TPS) is set to one during phase 1 and set to zero during phase 2. For TPS$=1$, S1, S2 and S3 are connected in series. Thus, the uniformly distributed bits from S1 are propagated to S2 and S3, and we have $q_j = 0.5$ for $j = 1, \ldots, n$. If TPS$=0$ (phase 2), the multiplexer between S1 and S2 passes to S2 the logical AND of k random sequences derived from S1. Thus after l_2 steps, S2's outputs and inverted outputs have probability 2^{-k} and $1-2^{-k}$, respectively, to be one. The circuitry separating S2 from S3 is a single AND-gate setting S3's input to zero if TPS$=0$. So after at most l_3 steps, outputs and inverted outputs of S3 will carry constant zeros ($q_j = 0$) and ones ($q_j = 1$), respectively.

A problem arises by auto-correlations within the pattern sequence produced in phase 2. To avoid them, we took the logical EXOR of two stages of S1 for each input of the AND gate between S1 and S2. A detailed description of possible dependencies and an explanation of how they are avoided by the above strategy is given in [5].

To get repeatable results, the pattern generator has to be reset at the beginning. The reset value can be chosen as a hard to match t^i in order to increase fault coverage. Resetting to a specific value causes no overhead since a flipflop's reset can be done to 1 and to 0 at the same costs.

To finish this section, let us sum up the additional hardware required for our pattern generator. It needs a shift register cell for each input of the CUT, some EXOR cells for the feedbacks of S1, a k-input and a 2-input AND gate, a multiplexer, and k EXOR gates (for calculating the input signals of the AND gate between S1 and S2). A flipflop for storing the TPS-signal also has to be added.

4 Experimental Results

The proposed design technique for pattern generation has been applied to c2670 and c7552. Table 3 shows the fault coverages obtained for different test lengths. (A test length given as $2 \times x$ means that the whole test consisted of x unweighted and x weighted patterns.) For c2670, all faults can be detected after only $16,000$ patterns, for c7552 $256,000$ patterns are necessary. Note that the later number is not too high: If the time necessary for contacting the CUT is taken into account, test lengths merely become critical if they are in the order of seconds. However, assuming a delay of 100 nanoseconds for c7552, our test would take only 0.0256 seconds.

To describe the overhead for our pattern generators, we counted shift register cells connected to the CUT, extra flipflops (FFs) and gates separately. In this way, we can avoid assumptions about the number of gate equivalents (GEs) for FFs (which depend on the given technology) and about the existence of input registers which might be used for the generator. For the gate count of the additional gates, we took the following values (static CMOS): $0.5 \cdot n$ GEs for an n-input NAND or NOR, $2.5 \cdot (n-1)$ GEs for an n-input EXOR, and 1.5 GEs for a $2:1$ multiplexer (realized as transmission gate). The results are given in Table 4.

c2670		c7552	
test length	fault coverage (undetected faults)	test length	fault coverage (undetected faults)
2×5000	99.58% (11)	2×16000	99.88% (9)
2×6000	99.85% (4)	2×32000	99.96% (3)
2×7000	99.88% (3)	2×64000	99.99% (1)
2×8000	100.00% (0)	2×128000	100.00% (0)

Table 3: Fault coverages obtained by pattern generator

	FFs on inputs	extra FFs	additional gates
c2670	157	1	14.5 GEs
c7552	206	1	12.0 GEs

Table 4: Overhead for pattern generator

It becomes clear that the overhead is mainly caused by the shift register cells which are also necessary for a LFSR based generation of pure random patterns. Assuming ≥ 4 GEs per flipflop, less than 2.5% of the overall overhead are due to the additional gates and the control flipflop.

A comparison to other approaches (cf. [5]) shows that our solution achieves highest fault coverages while requiring minimum hardware overhead. Design effort, which here means calculation of weights, is only lower in [6, 2]. However, the difference is negligible as the most expensive part (ATPG) is the same as in our method. Moreover, in [6, 2] time savings during weight optimization have to be paid by considerable losses in fault coverage (cf. [4]).

5 Conclusion

In this paper, a method to implement self-tests which takes into account design effort, hardware overhead, fault coverage, and test length has been described. For the most intractable ISCAS 85 benchmark circuits, we succeeded in detecting all non-redundant stuck-at faults after an acceptable number of patterns. An urgent and interesting question is how similar results can be obtained for sequential circuits and fault models. It should be possible to attack this problem by a generalization of the presented scheme.

Acknowledgements

The authors would like to thank Marita Hero and Wolfgang Vogelgesang for reading the manuscript.

References

[1] S. Akers and W. Jansz. Test set embedding in a built-in self-test environment. In *Proc. of the Int. Test Conf.*, 1989.

[2] F. Brglez, C. Gloster, and G. Kedem. Built-in self-test with weighted random pattern hardware. In *Proc. of the Int. Conf. on Comp. Design*, 1990.

[3] E. B. Eichelberger and E. Lindbloom. Random-pattern coverage enhancement and diagnosis for LSSD logic self-test. *IBM Journal Research and Development*, 27(3), May 1983.

[4] J. Hartmann. On numerical weight optimization for random testing. In *Proc. of the joint EDAC-EUROASIC Conf.*, 1993.

[5] J. Hartmann and G. Kemnitz. How to do weighted random testing for BIST? TR-13/1993, SFB 124, University of Saarbrücken, 1993.

[6] F. Muradali, V. K. Agarwal, and B. Nadeau-Dostie. A new procedure for weighted random built-in self-test. In *Proc. of the Int. Test Conf.*, 1990.

[7] I. Pomeranz and S. Reddy. 3-weight pseudo-random test generation based on a deterministic test set. In *Proc. of the VLSI Design Conf.*, 1992.

[8] H. Schnurmann, E. Lindbloom, and R. G. Carpenter. The weighted random test-pattern generator. *IEEE Trans. on Computers*, C-24, 1975.

[9] H.-J. Wunderlich. On computing optimized probabilities for random tests. In *Proc. of the Design Automation Conf.*, 1987.

[10] H.-J. Wunderlich. Self test using unequiprobable random patterns. In *Proc. of the Int. Symp. on Fault-Tolerant Computing*, 1987.