COVERAGE-DIRECTED ASSIGNMENT APPROACH TO BIST

Petr Fiser, Jan Hlavicka[†], Hana Kubátová
Department of Computer Science and Engineering
Czech Technical University
Karlovo nám. 13, 121 35 Prague 2
e-mail: fiserp@fel.cvut.cz, kubatova@fel.cvut.cz

Abstract. *We present a novel test-per-clock BIST method for combinational or full-scan circuits. Our task is to design a combinational circuit, namely the output decoder, transforming the pseudorandom LFSR code words into the required test patterns pre-generated by some ATPG tool. The process is based on finding the coverage of the ones in the test vectors and the subsequent generation of implicants that correspond to this coverage. The implicants are derived from the LFSR code words. The design of the output decoder is taken as a general combinatorial problem and it can be exploited in other areas of logic design too.*

1 Introduction

With the growing complexity of VLSI circuits the use of BIST (built-in self-test) is becoming inevitable. The external testers often cannot reach the internal logic of the chip or the test sequence is unacceptably long. This boosts the memory demands for the tester, as well as the test time. In these cases using the BIST is a good solution especially when the area overhead caused by BIST is not too big. The problem of BIST has been studied for more than twenty years and no satisfactory solution was found yet. The results reached can be found in several survey papers [1, 2].

We propose a test-per-clock BIST method where the test patterns are applied to the primary inputs of the CUT in parallel, thus in each clock cycle one test is being processed. The response is then drawn from the primary outputs and analyzed in the response evaluator, which is mostly a multi-input shift register (MISR).

The basic problem of constructing a BIST is finding a way in which the test patterns are generated. In the simplest approach some pseudo-random pattern generator (PRPG), which is mostly a LFSR (linear feedback shift register) or some kind of a cellular automaton is used to produce test patterns. However, the pseudo-random code words often do not ensure satisfactory fault coverage, and thus they must be modified somehow. Many methods were proposed in this area, see e.g. [3-7]. Most of the methods use some combinational block to transform the PRPG code words into the required test patterns. We propose a BIST method based on this approach too. The test pattern generator (TPG) consists of the PRPG and the combinational Output Decoder. It transforms the pseudorandom patterns into deterministic tests pre-generated by some ATPG tool. The structure of such a BIST is shown in Fig. 1.

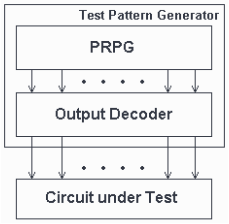


Figure 1: TPG structure

We propose a *Coverage-Directed Assignment* (CD-A) method to construct the output decoder. It is based on finding a rectangle cover of all the ones in the test patterns and the subsequent generation of implicants derived from the PRPG patterns with a help of this cover. The coverage-directed assignment method can be apprehended as a major generalization of the Boolean minimization process where the order of the input vectors is insignificant. Also the column-matching based BIST method presented in [8] can be treated a special case of the CD-A.

The rest of this paper is organized as follows: the Section 2 contains the problem statement, the principles of the method are described in Section 3 and experimental results are discussed in Section 4. Section 5 concludes the paper.

2 Problem Statement

Let us have an *n*-stage (*n*-bit) PRPG running for *p* cycles. Then the code words produced by this PRPG can be described by a ***C** matrix* (code matrix) with the dimensions (*n*, *p*). The parallel outputs of the PRPG are entering the output decoder as input variables x_0 - x_{n-1}. Thus, the columns of a **C** matrix will be sometimes denoted as values of the *input variables* of the output decoder, while the rows of the matrix can be taken as *input vectors* or *input minterms*. The test set pre-computed by an ATPG tool is described by a ***T** matrix* (test matrix). For an *r*-input CUT the output decoder has *r* outputs denoted as y_0-y_{r-1}. For a test set with *s* vectors the **T** matrix will have dimensions (*r*, *s*). The columns of the **T**-matrix will be denoted also as *output variables* of the decoder, the rows as the *output vectors*.

There are some obvious relationships valid for the values mentioned above, like $p \leq 2^n - 1$ (the maximum number of distinct patterns that can be generated by a PRPG) and $p \geq s$, because there must be enough patterns to implement all the test vectors generated by the ATPG. On the other hand, there are no strict requirements regarding the relationship of *n* and *r*, since the number of LFSR stages can be even smaller than the number of CUT inputs. For a larger number of **C** matrix columns the transformations are often easier and the resulting combinational logic is less complex.

The main idea of the method it is based on the following important fact: during the testing process for combinational circuits, the order of test patterns generated by an ATPG tool is insignificant and thus the patterns can be reordered in any way. In other words, any vector (row) from a **T** matrix can be assigned to any vector of a **C** matrix. Moreover, the rows in the **C** matrix need not form a compact block. The excessive patterns that are not transformed into test vectors just represent idle cycles of the PRPG. They do not disturb the testing, but only extend its length. If a low-power testing is required, we may use some pattern inhibition techniques - see [9]. Finding a transformation from **C** matrix to **T** matrix means finding a *matching* of all *s* rows of **T** matrix with any distinct *s* rows of **C** matrix.

3 Principles of the Method

The principles of a Coverage-Directed Assignment method are based on a simple notion: after assigning all the rows of the **C** matrix to the **T** matrix rows some kind Boolean minimization has to be performed. During this process implicants that cover all the ones in the output matrix (**T** matrix) are looked for. For each implicant the set of **T** matrix ones it covers is evaluated. Let us denote this set as a *coverage* of an implicant, the *coverage* of the output matrix will be a set of the coverages of all the implicants in the solution.

In the Coverage-Directed Assignment method we proceed backwards – first, we find a rectangle cover [10] of the output matrix and then we find the implicants that meet this coverage. Then the Boolean minimization process can be completely omitted and even performing the row assignment is not necessary – the implicants of a final function are being produced by this method. The final set of SOP forms (PLA matrix) is produced by joining the implicants with their coverage.

3.1 Find Coverage Phase

The coverage of the **T** matrix is produced in this phase. The problem fully corresponds to solving the rectangle covering problem that appear in many areas of logic design [10], however our algorithm slightly differs from the others. Let us state several Definitions:

Definition 1

Let t_i be an implicant. The *coverage set* $C(t_i)$ of the implicant t_i is a set of vectors (rows) of the **T** matrix, in which at least one “1” value is covered by this implicant. In other words, the coverage set is a set of vectors of the output matrix for which t_i is an implicant for at least one output variable.

■

Definition 2

The *coverage mask* $M(t_i)$ of the implicant t_i is the set of columns of the **T** matrix, in which all vectors included in $C(t_i)$ have one or more “1” value.

The coverage mask $M(t_i)$ can also be expressed as a vector in the output matrix corresponding to the term t_i. In the following text we will use both representations of the coverage mask.

■

Definition 3

The *coverage* of an implicant t_i is a pair of the sets $C(t_i)$ and $M(t_i)$ for which the following equation holds:

$$\forall a \in C(t_i), \forall b \in M(t_i) : \mathbf{T}[a,b] \neq "0"$$

The “1” values covered by t_i are identified by the Cartesian product $C(t_i) \times M(t_i)$.

■

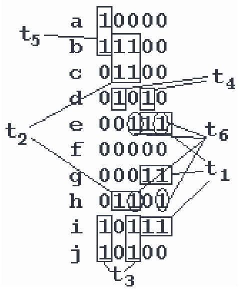
Definition 4

The *coverage of the matrix* ***T*** is a set of coverages $\{C(t_i), M(t_i)\}$ so that

$$\forall a < s, \forall b < r, \mathbf{T}[a,b] = "1" : \{a,b\} \in \bigcup_{\forall i} C(t_i) \times M(t_i)$$

■

The principles of the Find Coverage phase are illustrated by Fig. 2. Here the coverage of an example **T** matrix consisting of 6 implicants t_1-t_6 is shown. All the ones are covered, while no zero is covered, thus this coverage is complete and valid. The coverage sets and coverage masks are shown in Table 1. Note, that the rows of the matrix are labeled *a-j* instead of numbering, as later the numbers could be confusing.



a 10000
b 11100
c 01100
d 01010
e 00111
f 00000
g 00011
h 01101
i 10111
j 10100

Figure 2: The coverage of the T matrix

Table 1: The coverage sets and masks

Implicant	$C(t_i)$	$M(t_i)$
t_1	$\{e, g, i\}$	{0, 0, 0, 1, 1}
t_2	$\{b, c, h\}$	{0, 1, 1, 0, 0}
t_3	$\{i, j\}$	{1, 0, 1, 0, 0}
t_4	$\{d\}$	{0, 1, 0, 1, 0}
t_5	$\{a, b\}$	{1, 1, 0, 0, 0}
t_6	$\{e, h\}$	{0, 0, 1, 0, 1}

Obviously, there exist many possible covers of a particular function. Finding the *minimum cover*, i.e. the minimum number of implicants is a NP hard problem, thus some heuristic must be used to generate the coverage sets. We use a heuristic that sequentially tries to find the coverage sets that cover the maximum yet uncovered ones in the **T** matrix.

3.2 Find Implicant Phase

When the coverage of the **T** matrix is found, the implicants that have the required coverage sets should be generated from the **C** matrix. Such implicants will be denoted as implicants that *fulfil* this coverage. Their properties are studied in this section.

The implicant generation is based on the following Definition and Theorem.

Definition 5

Let us introduce an *inclusion function* $\varphi(t_1, t_2)$ for two terms t_1 and t_2 of the same dimension:

$\varphi(t_1, t_2) = 1$ if $t_2 \subseteq t_1$, thus t_2 is included in t_1.

$\varphi(t_1, t_2) = 0$ otherwise.

■

Theorem 1

The implicant t_i fulfils the coverage $C(t_i)$ if the number of minterms in the **C** matrix that are included in t_i is equal to the size of the $C(t_i)$ set, i.e.:

$$\sum_{j=0}^{p} \varphi(t_i, \mathbf{C}[j]) = |C(t_i)|$$

■

Proof

If a minterm of the **C** matrix is included in a term t_i, the term will have a value 1 for the values of the input variables corresponding to this minterm. If exactly j minterms of the **C** matrix are included in t_i, the term will have a value 1 for j rows of the **C** matrix. The size of the $C(t_i)$ set determines the number of rows of the **T** matrix, for which the term t_i has at least one value 1. Thus, if all the rows of the **C** matrix need to be assigned to the rows of the **T** matrix, the relation stated above must be valid.

■

This condition for selecting the implicants is still not sufficient. The terms that fulfil the coverages intersecting in one or more **T** matrix vectors must have a non-empty intersection; the number of the **C** matrix minterms included in this intersection must be equal to the size of the intersection of the respective coverages. Thus, all the intersections of the coverages must be computed and the previously stated condition must be applied to them too.

The implicant generation phase will be shown in our continuing example. First, we compute the intersections of the coverage sets:

$C(t_1) \cap C(t_3) = \{i\}$, $C(t_1) \cap C(t_6) = \{e\}$, $C(t_2) \cap C(t_5) = \{b\}$, $C(t_2) \cap C(t_6) = \{h\}$

Other set intersections than those listed above are empty.

We start, e.g. with the term t_1. We try to find a term that includes exactly 3 minterms from the **C** matrix (because $|C(t_1)| = 3$). The possible term is `t1 = (-01--)`.

Now the minterms that are contained in this term are assigned to it:

```
          A 10000
          B 11100
          C 00001
          D 10101 -> t_1
    C =   E 01111
          F 01001
          G 01110
          H 10110 -> t_1
          I 00110 -> t_1
          J 11010
```

Figure 5.

The term t_2 has to contain 3 minterms, while no vector assigned to t_1 must be assigned to it, as $C(t_1) \cap C(t_2) = \varnothing$. The possibility is `t2 = (--00-)`. The size of $C(t_3)$ is equal to 2 and $|C(t_1) \cap C(t_3)| = 1$, thus exactly one minterm assigned to t_3 has to be assigned to t_1 as well, while the second one need not be assigned yet. We will select `t3 = (--10-)`. We continue this way, until all implicants are found. Finally, the output vectors are assigned to the implicants (see Table 1) to generate the final PLA matrix. The final result will be as follows:

```
                             A 10000 -> t_2, t_5      PLA Matrix:
t_1 = (-01--)                B 11100 -> t_3
t_2 = (--00-)                C 00001 -> t_2, t_6      -01-- 00011
t_3 = (--10-)                D 10101 -> t_1, t_3      --00- 01100
t_4 = (---11)        C =     E 01111 -> t_4           --10- 10100
t_5 = (1-0--)                F 01001 -> t_2           ---11 01010
t_6 = (00---)                G 01110                  1-0-- 01010
                             H 10110 -> t_1           00--- 00101
                             I 00110 -> t_1, t_6
                             J 11010 -> t_5
```

SOP Forms:

$y_0 = x_2x_3'$

$y_1 = x_2'x_3'+ x_3x_4+ x_0x_2'$

$y_2 = x_2'x_3'+ x_2x_3'+ x_0'x_1'$

$y_3 = x_1'x_2+ x_3x_4+ x_0x_2'$

$y_4 = x_1'x_2+ x_0'x_1'$

Figure 6: The resulting terms

3.4 Generalized Coverage-Directed Assignment

The CD-A approach can be easily modified for a longer PRPG run, where there are more **C** matrix rows than the **T** matrix rows ($p > s$). Only the suitable **C** matrix rows are assigned to all the **T** matrix rows, while the remaining rows in **C** are ignored.

Let us remark, that this CD-A modification consists in modifying the Implicant Generation phase only, as the Find Coverage phase is completely independent on the **C** matrix. We must modify Theorem 1, above all. As not all the **C** matrix minterms will be included in the solution, the condition of equality of the size of the coverage set and the number of minterms covered by the searched term is not required. The term may cover more minterms, while the excessive ones are omitted in the final solution. Thus, the equation from Theorem 1 is modified to:

$$\sum_{j=0}^{p}\varphi(t_i,\mathbf{C}[j]) \geq |C(t_i)|$$

Similarly, this modification applies also to the intersections of the coverage sets.

4 Experimental Results - ISCAS Benchmarks

In order to test the algorithm on some practical examples we have chosen a subset of the ISCAS [10] benchmarks. The test patterns for all benchmark files were generated by the TurboTester [11, 12]. As a pseudorandom pattern generator a LFSR of the width equal to the number of primary inputs of the CUT was used, the number of patterns generated was fixed to 5000. The results are shown in Table 2. For each particular benchmark the number of its primary inputs (*r*) is given, together with the test length (*s*). The CD-A results are indicated by the number of terms obtained in the Find Coverage phase together with the total number of literals obtained in the second phase.

Table 2. ISCAS benchmarks

benchmark	inputs (*r*)	test length (*s*)	terms/literals
c432	36	40	49/141
c499	41	40	47/132
c880	60	36	41/106
c1355	41	84	105/358
c1908	33	107	139/564
c2670	233	84	104/309

5 Conclusions

We have introduced a novel test-per-clock BIST method for combinational circuits. It is based on a transformation of the PRPG code words into the required test patterns generated by some ATPG tool. The proposed algorithm firstly finds a rectangle cover of the “1”s in the test patterns and then the implicants that fulfil this coverage are generated from the PRPG code matrix.

The method was tested on the ISCAS benchmarks and satisfactory results were reached.

Acknowledgement

This research was in part supported by grant 102/01/1531 of the Czech Grant Agency (GACR) "Formal Approaches in Digital Circuit Diagnostics - Testable Design Verification"

References

[1] Agarwal, V., K., Kime, C., R., Saluja, K., K.: A tutorial on BIST, part 1: Principles, IEEE Design & Test of Computers, vol. 10, No.1 March 1993, pp.73-83, part 2: Applications, No.2 June 1993, pp.69-77

[2] McCluskey, E., J.: BIST techniques, IEEE Design & Test of Computers, vol. 2 No.2 Apr. 1985. pp.21-28, BIST structures. vol. 2 No.2 Apr. 1985. pp. 29-36

[3] Chatterjee, M., Pradhan, D., J.: A novel pattern generator for near-perfect fault coverage, Proc. of VLSI Test Symposium 1995, pp. 417-425

[4] Touba, N., A., McCluskey, E., J.: Synthesis Techniques for Pseudo-Random Built-In Self-Test, Technical Report, (CSL TR # 96-704), Departments of Electrical Engineering and Computer Science Stanford University, August 1996

[5] Bardell, P., H., McAnney, W., H., Savir, J.: Buit-In Test for VLSI: Pseudorandom Techniques, New York: Wiley, 1987.

[6] Hartmann, J., Kemnitz, G.: How to Do Weighted Random Testing for BIST, Proc. of International Conference on Computer-Aided Design (ICCAD), pp. 568-571, 1993

[7] Kiefer, G., Vranken, H., Marinissen, E.J., Wunderlich, H.J.: Application of deterministic logic BIST on industrial circuits, Proc. Int. Test Conf. (ITC’00), Atlantic City, NJ, Oct. 2000, pp. 105-114.

[8] Fišer, P., Hlavička, J.: Column-Matching Based BIST Design Method. Proc. 7th IEEE Europian Test Workshop (ETW'02), Corfu (Greece), 26.-29.5.2002, pp. 15-16

[9] Girard, P. et al.: A test vector inhibiting technique for low energy BIST design. IEEE VLSI Test Symposium, May 1999, pp. 407-412.

[10] Hassoun, S. - Sasao, T.: Logic Synthesis and Verification, Boston, MA, Kluwer Academic Publishers, 2002, 454 pp.

[11] Brglez, F., Fujiwara, H.: A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan, Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985

[12] Jervan, G., Markus, A., Paomets, P., Raik J., Ubar, R.: A CAD System for Teaching Digital Test, Proc. of the 2nd European Workshop on Microelectronics Education, Kluwer Academic Publishers, pp. 287-290, Noordwijkerhout, the Netherlands, May 14-15, 1998

[13] http://www.pld.ttu.ee/tt/