CD-A Based BIST Method

Petr Fišer, Jan Hlavička, 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: The paper introduces a novel test-per-clock BIST design technique for combinational (or full scan) circuits. The method is based on a design of a combinational block that transforms the code words produced by a LFSR into deterministic test patterns pre-generated by some ATPG tool. Our aim is to synthesize this block to be as small as possible. We propose a Coverage-Directed Assignment (CD-A) method to do this. The algorithm firstly finds the rectangle cover of the ones in the test patterns and subsequently tries to derive implicants of the transforming function.

I. INTRODUCTION

The problem of a built-in self-test (BIST) has been studied for more than two decades [1, 2]. Its use is becoming ever more important, since the complexity of the circuits rapidly grows and their internal logic is then hardly accessible from outside. Moreover, BIST enables us to test the circuit during its function (either on-line of off-line), which is of a key importance in a design of highly reliable circuits.

The basic problem of a BIST is the way how to generate the test patterns to reach sufficient fault coverage. In most of cases some pseudo-random pattern generator (PRPG) is used to produce test vectors. The linear feedback shift register (LFSR) is mostly used here. The *exhaustive testing*, in which all the 2^n-1 LFSR code words are fed into the circuit under test (CUT) enables us to reach a complete fault coverage, however, the testing is rather time consuming. When fewer patterns are applied to the CUT, the fault coverage obtained is often not sufficient. A good solution to this problem is to modify the PRPG code words by some additional circuitry. Some methods use an ATPG (automatic test pattern generator) to produce a set of test vectors ensuring the required fault coverage. These patterns are then stored in ROM from where they are applied to the CUT. The area overhead of a ROM is, however, extremely large, thus some compromise solution has to be used. In a *mixed-mode* BIST the PRPG is used to produce several test patterns detecting easy-to-detect faults and then the random pattern resistant faults are detected by patterns stored in ROM. However, the size of a memory is often large, even when using this approach. Here some pattern compression techniques have to be used [3]. Modern methods tend to completely eliminate the ROM. Here the PRPG is used to produce pseudorandom patterns, which are then being modified in order to reach a better fault coverage [4-5]. We propose a BIST method based on a transformation of the PRPG code words into deterministic test patterns pre-computed by some ATPG tool. Thus, the test pattern generator (TPG) consists of a PRPG and the combinational *output decoder*. Our task is to synthesize the output decoder to be as small as possible. The method is designed for a BIST of combinational circuits, thus the order of the test patterns is insignificant. The output decoder is then a combinational block that transforms the PRPG code words into the test patterns.

Earlier we have introduced the *column matching* method [6] to design the output decoder. Here the maximum of the output variables of the decoder is tried to be implemented as mere wires, thus without any logic. This significantly complicates the synthesis of the remaining outputs. In this paper a *Coverage-Directed Assignment* (CD-A) method to design the output decoder is proposed. In its main part, namely the Find Coverage phase, the rectangle cover of all ones in the test set is found. This coverage determines the output parts of the cubes (implicants) of the output decoder. The input parts of these cubes are then derived from the PRPG patterns. The result of the algorithm is a sum-of-products (SOP) form of the output decoder logic.

The paper is organized as follows: the Section II contains the problem statement, the principles of the method are described in Section III and experimental results are discussed in Section IV. Section V concludes the paper.

II. PROBLEM STATEMENT

Let us have an *n*-bit PRPG running for *p* clock cycles. The code words generated by this PRPG can be described by a **C** matrix (code matrix) of dimensions (*p, n*). These code words are to be transformed into the test patterns pre-computed by some ATPG tool. They are described by a **T** matrix (test matrix). For an *r*-input CUT and the test consisting of *s* vectors the **T** matrix will have dimensions (*s, r*). The rows of the matrices will be denoted as *vectors*.

Designing the output decoder means finding a combinational logic that transforms these two matrices. Each of the **T** matrix vectors must be generated from some of the **C** matrix vector. Thus, the transformation consists in finding an *assignment* of all the **T** matrix vectors to some of the **C** matrix vectors. The **C** matrix vectors generating no **T** matrix vector 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.

III. THE CD-A PRINCIPLES

The principles of a Coverage-Directed Assignment (CD-A) 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 [6]. At the beginning of this process a set of implicants that cover all ones in the **T** matrix (output matrix) is found. For each implicant the set of **T** matrix ones it covers is enumerated and the irredundant set (or better the minimum set) of implicants covering all the ones is computed during the covering problem solution. Let us denote the set of the ones covered by an implicant as its *coverage*, the *coverage* of the output matrix will be a set of the coverages of all the implicants in the final solution.

In the CD-A algorithm we proceed backwards: an irredundant cover of the output matrix is found and after that the implicants that correspond to the cover are derived. The process thus consists of two *totally independent* phases: the Find Coverage phase computes the coverage of the **T** matrix (independently on the **C** matrix) and in the subsequent Find Implicants phase the proper implicants are computed from the **C** matrix with a help of the coverages.

A. 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 [7], 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 number of coverages of the **T** matrix determines the number of implicants in the final SOP form. Obviously, there exist many possible coverages of a particular matrix. Finding the *minimum rectangle cover* is a NP hard problem, thus some heuristic must be used. Moreover, the minimum rectangle cover does not ensure the minimal complexity of the output decoder. This problem will be further explained in Subsection IV.A. We use a heuristic that sequentially tries to find the coverage sets that cover the maximum yet uncovered ones in the **T** matrix. At the beginning, the **T** matrix vector with the maximum number of ones is selected and included into the first coverage set, while the coverage mask is set equal to this vector. Next, we systematically try to add such vectors to this set, so the number of ones covered by them is increasing. After each such an addition the decision has to be made whether to continue adding vectors (if possible) or terminate the set generation. Too large coverage sets may cause problems when generating the implicants (see Subsection III.C). In praxis, the decision is made at random with a given probability. The probability of the further increase of the number of vectors covered by a processed coverage is determined by a *depth factor* (*DF*). The influence of this factor on the result will be studied in Subsection IV.A. After completing one coverage, another one is constructed in the same way, until all the "1" entries are covered.

B. Example of Find Implicant Phase

The principles of the Find Coverage phase are illustrated by Fig. 1. The ten **T** matrix vectors are labeled a-j. A minimum rectangle cover of this matrix consisting of six coverages has been found. The coverage sets and coverage masks are shown in Table 1.

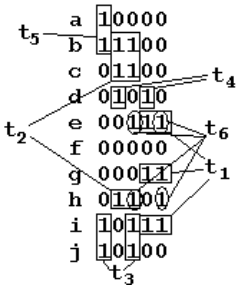

Fig. 1: The coverage of the T matrix

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}

C. Find Implicant Phase

When the coverage of the **T** matrix is found the implicants corresponding to the coverage sets should be derived from the vectors (minterms) of the **C** matrix. Such implicants will be denoted as implicants that *fulfil* the respective coverage. Their properties will be studied in this section.

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. ■

Now we will state the necessary conditions for a term t_i to fulfil the given coverage $\{C(t_i), M(t_i)\}$:

1. Obviously, all the **C** matrix minterms included in t_i imply a "1" output value for t_i. The cardinality of the coverage set $|C(t_i)|$ determines the number of **T** matrix vectors, in which the outputs included in $M(t_i)$ have "1" value. A term that fulfils a coverage $C(t_i)$ must then generate at least $|C(t_i)|$ ones from the **C** matrix vectors, i.e. at least $|C(t_i)|$ **C** matrix vectors must be included in t_i:

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

2. 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 greater or equal to the cardinality of the intersection of the respective coverages. Thus, the previous condition applies too all the intersections of the coverages too.
3. It may happen that the candidate term contains so many **C** matrix minterms, that there is will be enough "free" minterms left for the not yet processed coverage sets. So the additional condition must be applied:

$$p - \sum_{j=0}^{p} \varphi(t_i, \mathbf{C}[j]) \geq \left| \bigcup_{j \notin \mathbf{P}} C(t_j) - \bigcup_{j \in \mathbf{P}} C(t_j) \right|$$

where **P** is a set of the already processed implicants.

The candidates for implicants fulfilling a particular coverage are being constructed by expansion of the **C** matrix terms. The principles of the method will not be described in detail here, since it exceeds the scope of this paper.

D. Illustrative Example

Let us have the **C** matrix consisting of 20 vectors as shown in Fig. 2 (the matrix is divided into four columns to save space). The vectors are labeled A-T. We will try to derive the six implicants that fulfil the coverage from the previous example.

```
     A 01111   E 00001   I 01000    M 00100   Q 00000
C =  B 01101   F 10001   J 11000    N 10011   R 01100
     C 00011   G 11001   K 10111    O 00010   S 00101
     D 01011   H 01110   L 11010    P 01001   T 10100
```

Fig. 2: The example **C** matrix

In Table 2 the intersections of the coverage sets and their cardinalities are computed, together with the cardinalities of the sets. The coverage sets intersections not present in Table 2 are empty. Table 3 shows the final results. The left side of this table shows the structure of the terms t_1 - t_6 that form the final solution (the input variables of the decoder are labeled $x_0 - x_4$). The process of generating the implicants is not described in this example, however it is apparent from the example that the solution is valid. The inclusion relation of the **C** matrix vectors to the terms is shown in the right-hand side of the Table, together with the final assignment of the **C** and **T** matrix vectors.

TABLE 2. Coverage sets

Cov. set	Contains	Card
$C(t_1)$	{e, g, i}	3
$C(t_2)$	{b, c, h}	3
$C(t_3)$	{i, j}	2
$C(t_4)$	{d}	1
$C(t_5)$	{a, b}	2
$C(t_6)$	{e, h}	2
$C(t_1) \cap C(t_3)$	{i}	1
$C(t_1) \cap C(t_6)$	{e}	1
$C(t_2) \cap C(t_5)$	{b}	1
$C(t_2) \cap C(t_6)$	{h}	1

TABLE 3. The resulting terms and assignments

t_1 = (---1-)
t_2 = (--00-)
t_3 = (1-1--)
t_4 = (-1---)
t_5 = (0---1)
t_6 = (1-0--)

$t_1 = x_3$
$t_2= x_2'x_3'$
$t_3 = x_0x_2$
$t_4 = x_1$
$t_5 = x_0'x_4$
$t_6 = x_0x_2'$

```
     A 01111 ⊆ t1, t4, t5
     B 01101 ⊆ t4, t5
     C 00011 ⊆ t1, t5
     D 01011 ⊆ t1, t4, t5
     E 00001 ⊆ t2, t5      -> b
     F 10001 ⊆ t2, t6      -> h
     G 11001 ⊆ t2, t4, t6
     H 01110 ⊆ t1, t4
     I 01000 ⊆ t2, t4
C =  J 11000 ⊆ t2, t4, t6
     K 10111 ⊆ t1, t3      -> i
     L 11010 ⊆ t4, t6
     M 00100               -> f
     N 10011 ⊆ t1, t6      -> e
     O 00010 ⊆ t1          -> g
     P 01001 ⊆ t2, t4, t5
     Q 00000 ⊆ t2          -> c
     R 01100 ⊆ t4          -> d
     S 00101 ⊆ t5          -> a
     T 10100 ⊆ t3          -> j
```

IV. EXPERIMENTAL RESULTS

A. Influence of the DF on the Result

As an example we have chosen the c432 ISCAS benchmark circuit [8], whose test vectors were generated by an ATPG tool TurboTester [9]. The c432 circuit had 36 inputs and the test set contained 40 vectors. The **C** matrix vectors were produced by a 36-stage LFSR running for 300 cycles. We have varied the DF from 1:100 to 2:1 to determine its influence on the complexity of the output decoder. The results of the experiment are shown in Table 4. The resulting decoder is described by the number of group product terms in the SOP form, the output cost (OC), which is the number of wires entering the OR gates (i.e. the total number of terms) and the number of literals in the SOP form.

TABLE 4. Influence of DF on the result

DF	terms/OC/lit
1:100	40/734/114
1:10	43/696/121
1:5	47/644/135

DF	terms/OC/lit
1:2	59/555/174
1:1	56/574/169
2:1	72/468/216

There is the lowest number of terms for very low values of DF. In fact, the Find Coverage phase generates only one-vector coverages in this case, and thus it does not influence the result. Notice the number of terms equal to the number of the **T** matrix vectors. Even if the number of terms is low, the output cost is rather high. This is due to the fact that the coverage masks contain a lot of "1" values. We can significantly reduce the output cost by increasing the value of DF. However, the number of terms then grows, thus the trade-off must be found.

B. ISCAS Benchmarks

In order to test the algorithm on some practical examples we have chosen a subset of the ISCAS benchmarks. The test patterns for all benchmark files were generated by TurboTester. 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, DF was fixed to 1:3. The results are shown in Table 5. 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 5. 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

V. CONCLUSIONS

We have proposed a novel test-per-clock BIST method for combinational circuits. It is based on a synthesis of a combinational block transforming the PRPG code words into arbitrary tests. The algorithm firstly finds a coverage of the ones in the test patterns and then generates implicants that fulfil this coverage. The method was tested on the ISCAS benchmarks and satisfactory results were reached. General principles of the method can be exploited also in other areas of logic design, e.g. in Boolean minimization.

ACKNOWLEDGEMENT

This research has been in part supported by the GA102/03/0672 grant (Research of methods and tools for verification of embedded computer system fault tolerance) and MSM 212300014 , 1999 – 2003.

REFERENCES

[1] V. K. Agarwal, C. R. Kime, K. K. Saluja: "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] E. J. McCluskey: "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] V. K. Agarwal, E. Cerny: „Store and Generate Built-In Testing Approach", Proc. of FTCS-11, pp. 35-40, 1981

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

[5] N. A. Touba, E.J. McCluskey: "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

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

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

[8] F. Brglez, H. Fujiwara: „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

[9] G. Jervan, A. Markus, P. Paomets, J. Raik, R. Ubar: "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