Multiple-Vector Column-Matching BIST Design Method

Petr Fišer, 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* - Extension of a BIST design algorithm is proposed in this paper. The method is based on a synthesis of a combinational block - the decoder, transforming pseudo-random code words into deterministic test patterns pre-computed by an ATPG tool. The column-matching algorithm is used to design the decoder. Using this algorithm, maximum of decoder outputs is tried to be matched with the decoder inputs, yielding the outputs be implemented as wires, thus without any logic.**

The newly proposed enhancement consists in a major generalization of the method. The ATPG possibility of generating more than one test vectors for one fault is exploited, yielding smaller area overhead. The complexity of the resulting BIST logic reduction is evaluated for some of the ISCAS benchmarks.

I. INTRODUCTION

The complexity of present VLSI circuits rapidly grows. Their testing is becoming more and more important, together with the tests complexity and total costs. Using only external test equipment (ATE) is becoming impossible, mainly due to a huge amount of test vectors to be applied, long testing time and very expensive test equipment. Incorporating Built-in Self-Test (BIST) methods becomes inevitable. By now, many BIST methods were developed [1 - 5], all of them trying to find some trade-off between these four mutually antipodal aspects: *the fault coverage*, *test time, the area overhead* and the *BIST design time*. A high fault coverage means either a long test time (exhaustive test), or a high area overhead (deterministic ROM-based BIST). The pseudo-random testing established the simplest trade-off between all these three criteria. With an extremely low area overhead, the circuit can be tested usually up to more than 90% in a relatively small number of clock cycles (thousands).

A combination of a pseudo-random and deterministic BIST is being referred to as a *mixed-mode BIST*. The easy-to-detect faults are tested by pseudo-random test patterns, and the deterministic patterns are generated to test the remaining, undetected faults. The popular bit-fixing [4] and bit-flipping [2] techniques belong to this category.

Our column-matching method is based on a transformation of pseudo-random patterns into deterministic patterns pre-computed by an ATPG (Automatic Test Pattern Generator) tool. This transformation is being done by a combinational block called "*Output decoder*". To reduce the decoder logic, we try to implement as many outputs as possible by wires, without any logic.

To support the mixed-mode testing, the test is divided into two disjoint phases: the *pseudo-random* one and the *deterministic one*. This enables us to significantly reduce the decoder logic, together with the control logic as well.

The BIST area overhead becomes an essential issue now. For ASIC designers the area becomes more important than the design time, since the overall chip design time significantly surpasses the BIST design time. Thus, any improvement of the BIST design methods, in terms of the area overhead, is beneficial. An enhancement of our BIST design method is proposed in this paper. A significant area overhead reduction is involved, for a cost of a longer design time. The improvement consists in a generalization of the basic method, to fully exploit capabilities of ATPGs. The ATPG generates more than one test vectors for each tested fault in our algorithm, thus the algorithm has more freedom in generating the test sequence.

The paper is structured as follows: basic principles of the column-matching method are described in Section II, an overview of the test generation modes is presented in Section III and the newly proposed enhancement principles are described in Section IV. Experimental results are shown in Section V, Section VI concludes the paper.

II. BASIC PRINCIPLES OF OUR METHOD

The method is primarily intended for a test-per-clock BIST, thus the test patterns are applied to the primary inputs of the circuit-under-test (CUT) in parallel. However, the method can be modified for a test-per-scan as well [5].

The column-matching method is based on a transformation of pseudo-random patterns by a combinational block (*Output Decoder*), so that deterministic patterns are generated. Any set of deterministic patterns can be used, thus the fault coverage reached depends on these patterns only. 100% fault coverage is considered in the following text. However, the method can be modified so that smaller fault coverage is reached, with a benefit of smaller area overhead.

The PRPG is mostly constructed as a linear feedback shift register (LFSR) with an appropriate generating polynomial, or as a cellular automaton. The basic structure of such a test-per-clock BIST is shown Fig. 1.

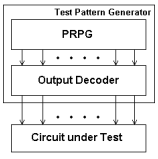

Figure 1: Test-per-clock BIST structure

A. Output Decoder Design

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 test patterns pre-computed by an ATPG tool. The patterns are defined by a **T** matrix (*test matrix*). For an r-input CUT and the test consisting of s vectors the **T** matrix has dimensions (s, r). The rows of the matrices will be denoted as *vectors*. There is no relationship between n and r, since the number of PRPG outputs can be even less than the number of CUT inputs. However, we will consider $n = r$ in this paper, for simplicity.

The output decoder logic modifies the **C** matrix vectors to obtain all the **T** matrix vectors. As the proposed method is restricted to combinational circuits, the order of the test patterns is insignificant. Finding a transformation from the **C** matrix to the **T** matrix means finding a pairing of each of the s rows of the **T** matrix with distinct rows of the **C** matrix –finding a *row assignment* (Fig. 2).

The *Output decoder* is a combinational block converting s n-dimensional vectors of the **C** matrix into s r-dimensional vectors of the **T** matrix. The decoder is represented by a Boolean function having n inputs and r outputs, where only values of s terms are defined; the rest are don't cares implicitly. This Boolean function can be easily described by a truth table, where the output part corresponds to the **T** matrix, while the input part consists of s **C** matrix vectors assigned to the **T** matrix rows. The set of such vectors will be denoted as a *pruned **C** matrix*.

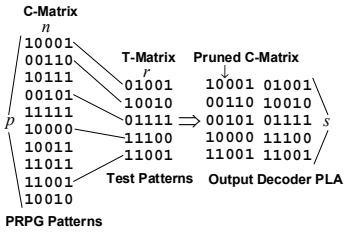

Figure 2: Assignment of the rows

B. The Column-Matching Method

The task is now, how to assign the rows to each other to reach maximum area overhead reduction. The aim of the column-matching method is to assign all the **T** matrix rows to some of the **C** matrix rows so that some columns of the **T** matrix will be *equal* to some of the pruned **C** matrix columns in the result. This involves no logic needed to implement these **T** matrix columns (output variables of the decoder); they are implemented as simple wired connections. This idea can be extended to a *negative matching*. Since most of the LFSR flip-flops are provided with the negated outputs as well, columns with opposite values can be matched too.

An illustrative example is shown in Fig. 3. The matched columns of the pruned **C** matrix and **T** matrix from Fig. 2 are shown here. The **T** matrix column y_1 is matched with the **C** matrix column x_3 (negatively), then y_3 with x_1 (negatively) and y_4 with x_4 (positively).

Thus, the outputs y_1, y_3 and y_4 are implemented without any combinational logic, while the remaining outputs have to be synthesized using some standard two-level Boolean minimization tools, like ESPRESSO [9] or BOOM [10, 11].

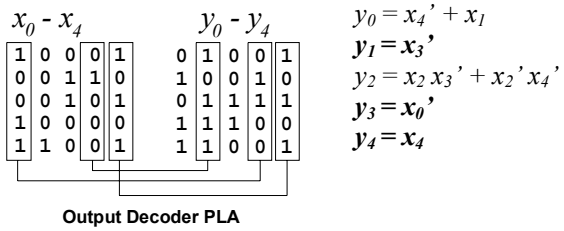

Figure 3: Column matching example

C. Finding the Columns to Be Matched

It is impossible to look for an optimum matching of columns (so that maximum of columns is matched) in practical examples, since the number of possible combinations grows exponentially with the number of columns. Hence, some kind of a heuristic has to be used. In practice, the number of PRPG patterns to be transformed is usually much higher than the number of test vectors ($p >> s$). Then, almost any two columns can be matched together at the beginning. Thus, we select the columns to be matched purely at random, one by one, until there is no possibility for any more matches.

Another difficult task is the very finding out if the column match to be performed is valid, i.e., if it leads to any solution. The only way how to solve this problem is to perform the row assignment. An efficient heuristic based on a *blocking matrix B* has been proposed in [6]. The blocking matrix is a binary matrix (it contains only '0' and '1' values) of dimensions (p, s). Thus, it has as many columns as there are **T** matrix rows and as many rows as there are **C** matrix rows. The value '1' in the cell **B**[k, l] indicates that the k-th **C** matrix row may be assigned to the l-th **T** matrix row, '0' value indicates the contrary. At the beginning of the algorithm all the **B** matrix cells are filled with a '1' value, since there are no restrictions for row assignments. After the i-th **C** matrix column is matched with the j-th **T** matrix column, the **B** matrix cells [k, l] are set to '0' when the k-th input row contains in an i-th column an opposite value to the l-th output row in a j-th column. Thus, rows containing opposite values in the matched columns cannot be assigned to each other.

B[k, l] := '0' when (**C**[k, i] ≠ **T**[l, j] ∧ **T**[l, j] ≠ don't care)

If the negative column match is to be performed, the **B** matrix cells are set to '0' when equal values are present in the respective positions.

The final row assignment involves only a selection of one row from the possible ones for each of the columns.

D. Mixed-Mode Column-Matching BIST

The basic column-matching algorithm was later extended to a mixed-mode BIST [7]. A general structure of our mixed-mode BIST design is shown in Fig. 4. The test is divided into two disjoint phases. In the first, pseudo-random phase the pseudo-random code words are produced by a LFSR and fed to the CUT unmodified. The subsequent patterns are transformed into deterministic vectors by the Output Decoder in the deterministic phase. The switching logic selects the patterns that are to be applied to the CUT. The switching logic then consists of multiplexers, in general. The area overhead caused by the switching logic needs not be too big, since the structure of the BIST controller can be very efficiently exploited here. The circuit's response is evaluated, usually in the multi-input shift register (MISR).

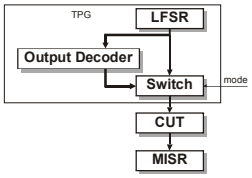

Figure 4: Mixed-mode BIST structure

E. The BIST Design Process

The whole BIST design process is divided into four consecutive phases:

1. Simulation of several (*PR*) pseudo-random patterns for the CUT and determination of undetected faults (by a fault simulation).
2. Computation of deterministic test patterns for these faults by an ATPG tool.
3. Performing the column-matching for the following *Det* pseudo-random PRPG patterns and the deterministic tests.
4. Synthesis of the decoder for the unmatched outputs.

The lengths of the two phases essentially influence the design time and area overhead. For more details see [15].

An artificial illustrative example is shown in Fig. 5. The BIST logic for a 5-input circuit is to be synthesized here. A 5-bit LFSR is run for 5 cycles first where the easily testable faults are detected. Then we run the fault simulation to find the undetected faults, for which the test vectors are generated by an ATPG. At the end the decoder logic is synthesized for these tests and the subsequent LFSR patterns. The resulting circuitry is shown in Fig. 6.

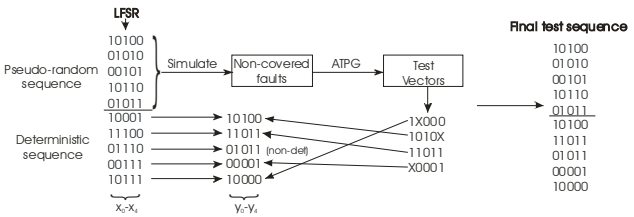

Figure 5: Test sequence generation

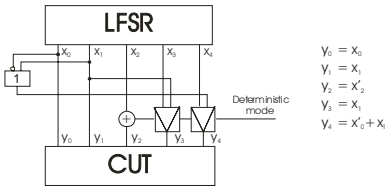

Figure 6: Resulting BIST circuitry

III. Test Pattern Generation

A. Influence of an ATPG

As it was said before, the column-matching method is so universal, that any test vectors set can be used. This implies that the method can be adjusted to any fault model, as long as basic requirements for the test vectors are held. For example, delay faults cannot be tested using simple column-matching, since the test vector pairs have to stay together here. However, after a slight algorithm modification even this could be possible. On the other hand, IDDQ testing is supported without any modification. We will use the stuck-at fault model in this paper, since the ATPG tool used [8] supports this model.

Most of available non-commercial ATPGs can be influenced, so that they produce various sets of test vectors. Our only and necessary requirement for the ATPG tool used is the capability to produce test vectors for a specified set of faults.

In the most general case, possible obtained test sets can be divided as follows:

1. *Non-compacted test without don't cares*. Such a test set is usually obtained by a random-pattern simulation and a subsequent deterministic test generation. The test is usually long and far from optimum.
2. *Compacted test set without don't care values*. Here the test comprises of minimum of test vectors (in the optimum case), obtained after deterministic test set generation and compaction, followed by the don't care substitution.
3. *Compacted test set with don't cares*. The test comprises of minimum of test vectors (in the optimum case), obtained after deterministic test set generation followed by a compaction. The don't care values are retained. However, their number is usually negligible.
4. *Non-compacted test with don't cares*. The test set is produced by a deterministic test set generator. No test compaction is executed.
5. *One test vector for each fault*. The test pattern generation is usually accompanied by a fault simulation. Thus, after one test vector is produced during the test generation process, fault simulation is executed for this vector and faults detected by it are removed from the processed fault list. This was the case assumed in the preceding cases. The ATPG may proceed in the simplest way, by generating one deterministic vector for each fault. No fault simulation or test compaction is involved. Test

vectors with many don't care values are usually obtained. The test set is often large, comparing to previous cases, however many don't care values are present in the test, which is usually beneficial.

6. *More than one test vector for each fault.* As a generalization of the previous item, more test vectors for each fault can be produced, if possible. The test is then even longer, but offers much greater flexibility.
7. *All the possible test vectors for each fault.* This is the most general case. Some ATPGs are able to produce all the possible test vectors for each fault. However, the test set size is then prohibitively large, thus such a case usually cannot be used in practice.

We use the Atalanta ATPG tool [8] to generate test vectors. All the above mentioned test sets can be obtained by it.

According the paper proposed in [6], don't cares present in the test set (**T** matrix) are beneficial, since they bring more freedom to the column matches choosing, and consequently reduce the BIST area overhead. Thus, is more advantageous to choose the test generation alternatives 3-7. We have found that the test set compaction method, which is used in the Atalanta tool, does not perform well for our purposes, thus we have introduced our static compaction method based on joining test vectors. Then, we have enhanced the column-matching algorithm to be able to handle test sets having more than one test vectors for each fault, to improve the results.

B. Simple Test Set Compaction

Due to the fact that the test set compaction performed by an ATPG is often lacking in quality and is insufficient for our method, we had to introduce a static compaction method to our algorithm. Maximum of the don't care values in the test set should be retained after the compaction. An exact test compaction algorithm is usually not applicable, since its time complexity is prohibitively large (it is an NP-hard problem). Thus, a heuristic method has to be used.

The algorithm we use is simple but effective. It is based on joining pairs of test vectors. Two test vectors can be joined, when they have a non-empty intersection. The result of their joining will be that intersection. Considering that a test vector t_1 detects a fault set F_1 and a test vector t_2 detects a fault set F_2, their intersection $t_1 \cap t_2$ detects faults $F_1 \cup F_2$.

Let us have a test set comprising of v vectors. Each vector is compared with each other and the size (dimensionality) of their intersection is computed. Two vectors having the "biggest" intersection will be joined. In other words, two vectors differing in at least one bit cannot be joined (since they have an empty intersection); two vectors having minimum collisions with a don't care on one side and a '0' or '1' value on the other side are joined. Test vectors loosing a minimum number of don't cares by their joining are joined. This is being repetitively performed until there is no chance to join any more vectors. The complexity of such an algorithm is $O(v^3)$, which sometimes means a significant computational time increase. The number of test vectors can be significantly reduced by this method, see Table 1.

IV. Multiple-Vector Column-Matching

A. Basic Principles

The more "freedom" has the column-matching algorithm in selection of the matches, the better it performs. Particularly, more don't care values in the test set induce more column matches and thus less area overhead [6]. Let us consider an example where two test vector sets are to be mapped onto PRPG patterns, one set generated by the 3-rd APTG mode (compacted test set with don't cares), the second one as the 5-th one (one test vector for each fault, with don't cares). The second test set will be much larger than the first one. On the other hand, more don't care values will be present in the second one (in general). Practical examples have shown that even when there are more test vectors to be generated by the Output decoder, the BIST area overhead is less if the test vectors have many don't care values. Thus, the second case will perform better, in terms of the area overhead.

The above-mentioned notion can be extended, so that there will be many test vectors available to choose from. The aim of the column-matching algorithm wouldn't be to synthesize all the test vectors then; the aim would be to synthesize vectors that cover all faults (from the given fault set), regardless by what vectors. Thus, even more freedom will be given to the matching algorithm, which yields better results. This is the main idea of the *multiple-vector column-matching*.

In order to adapt basic column-matching principles to be able to exploit more test vectors for each fault, several modifications had to be done. First of all, each test vector has to be accompanied by a *fault mask*. The fault mask is a binary vector identifying faults that are detected by the test vector. First, the fault list for the tested circuit is determined. The size of the fault mask is then equal to the number of faults, each position in the fault list corresponding to one fault. A '1' value indicates that the respective fault is detected by the test vector, '0' the contrary. The fault mask is obtained by a fault simulation of the respective test vector. After the fault masks are generated, all the test vectors are put together; there is no need to distinguish between them. Information on what vector was generated for what fault may be lost.

B. Multiple-Vector Test Set Compaction

Since the number of test vectors generated by an ATPG is often large, static test compaction should be performed. The algorithm described in Subsection 3.2 is used. The only modification is that after joining the test vectors their fault masks have to be joined too. The resulting fault mask is obtained by OR-ing the two fault masks, since faults detected by both joined vectors are detected by the resulting vector.

Even when the test set compaction reduces the freedom given to the column-matching, we have found experimentally that it is advantageous to perform it. When the test set compaction is not performed, the column-matching runtime is prohibitively long and usually there is no improvement in the quality of the result.

Example

Let us consider a 5-input CUT having 10 faults. The two example test vectors together with their fault masks will be joined as follows:

```
10-0-  1100101001
1-10-  0100100100
1010-  1100101101
```

C. Modified Row Assignment

The basic column-matching algorithm remains unchanged but the row assignment. Since there are many test vectors for each fault, the test set is redundant. Thus, not all test vectors have to be synthesized by the Output decoder; the aim is to detect all faults now, regardless by what vectors. Not all the **B** matrix columns have to be assigned then.

Making a row assignment is an NP-hard problem, similar to a binate covering problem. Thus, solving it exactly becomes impossible. We use a heuristic, where the heuristic function for a selection of a **B** matrix column (test vector) is the number of yet undetected faults it detects. At the beginning of the algorithm, the **B** matrix column detecting most of faults (i.e., test vector having most '1's in the fault mask) is selected. For this column a row having a '1' value in the respective position in the **B** matrix is found, so that this row has a minimum number of '1's in other positions. It is the row (**C** matrix vector), that may be transformed into the required test vector and simultaneously may be transformed into a minimum of others. The selected column and row are assigned to each other, removed from the **B** matrix and the detected faults are removed from the fault list. The column selection is repeated, until the fault list is empty or an undetectable fault is encountered (which means an invalid assignment). When an invalid assignment is returned, the last column match is taken back and another column matches are tried.

Basic principles of the row assignment are outlined by the following pseudo-code:

```
Assign {
   set(fl); // create a complete fault list
   do {
       c = FindBestColumn(B, faultmasks, fl);
         // find column detecting most faults from fl
       r = FindBestRow(B, c);
         // find a row, so that B[r, c] = 1 and has a
            minimum of 1's
       if (r != NULL) { // a row B[r, c]=1 was found
            MakeMatch(c, r);
            RemoveFaults(fl, c);
               // remove faults detected by c from fl
            RemoveColumn(B, c);
               // remove c from B matrix
       } else return(FAIL);
   } while(!empty(fl));
   return(SUCCEED);
}
```

Algorithm 1: Row Assignment

D. Modified BIST Design Process

Summarizing all the modifications needed to be done to extend the BIST design method to support multiple-vector column-matching, the whole process consists of these phases:

1. Simulation of several (*PR*) pseudo-random patterns for the CUT and determination of undetected faults.
2. Computation of the deterministic test patterns for these faults by an ATPG tool, generating more than one test pattern for each fault.
3. Fault simulation for each of the test vectors, i.e., computing fault masks.
4. Test set compaction.
5. Performing the multiple-vector column-matching.
6. Synthesis of the decoder for the unmatched outputs.

V. Experimental Results

A. Test Set Compaction Results

The test set compaction results are shown here. The number of test vectors processed is essential to the column-matching phase. Its runtime rapidly grows with an increasing number of test vectors [15]. On the other hand, the number of don't care values in the test significantly increases the performance of the column-matching algorithm. The compaction algorithm should thus reduce the number of test vectors, while keeping a sufficient number of don't cares. The algorithm described in Subsection III.B fully satisfies these requirements.

Experimental results performed on some of the ISCAS benchmarks [12, 13] are shown in Table 1. First, "*PR*" pseudo-random patterns were simulated using HOPE fault simulator [14]. Test sets for the undetected faults were computed by Atalanta ATPG [8]. The ATPG was set to generate "*vct/vlt*" vectors for each fault. The total number of test vectors is shown in the "*ATPG*" column. After the compaction, their number was reduced to "*compact*". The amount of don't care values in the final compacted test set is shown in the last column. It can be well observed that with an increasing number of test vectors the number of don't cares decreases. This is due to the fact that the compaction algorithm preferably selects vectors having many don't cares to be joined. However, this "disadvantage" is compensated by the freedom offered by the number of vectors more than enough.

TABLE 1.
TEST COMPACTION RESULTS

bench	*PR*	*vct/flt*	*ATPG*	*compact*	*DC*
c1908	1000	1	42	36	50 %
		10	382	340	25 %
c2670	10 K	1	201	74	83 %
		10	1824	825	77 %
c3540	1000	1	31	25	72 %
		10	117	101	65 %
		100	663	555	56 %
c7552	10 K	1	215	106	69 %
		10	2141	1206	68 %
s1196	1000	1	93	55	59 %
		100	392	259	56 %
s1238	3000	1	45	33	57 %
		100	140	95	52 %
s5378	10 K	1	23	19	92 %
		100	289	258	92 %

bench	*PR*	*vct/flt*	*ATPG*	*compact*	*DC*
s9234.1	50 K	1	321	99	82 %
		10	2899	1003	81 %
s13207.1	10 K	1	466	74	96 %
		10	1538	362	96 %

B. Comparison with Unmodified Column-Matching

Reduction of the BIST area overhead with respect to the original column-matching algorithm is demonstrated in this Subsection. We have run the whole BIST synthesis process using the original column-matching algorithm (using one test vector per one fault) and the column-matching algorithm exploiting multiple vectors, for each of the presented benchmark circuit. The results are shown in Table 2. Some of the table column labels are retained from Table 1. The "*PR/det*" column indicates the length of the pseudo-random and deterministic phase, respectively. The "*M*" column indicates the number of column matches obtained, from the maximum possible (number of CUT inputs). The column-matching algorithm runtime in seconds is indicated in the next column. Then the area overhead of the synthesized BIST (i.e., the output decoder and switch), with respect to the size of the original circuit is shown. The improvement reached by the proposed method, with respect to the simple method, is shown in the last column. It can be seen that the improvement is quite significant in many cases, sometimes almost halving the area overhead (c1908).

VI. CONCLUSIONS

An extension of the column-matching mixed-mode BIST method has been proposed. Pseudorandom PRPG code words are being transformed into deterministic test patterns computed by some ATPG tool. The influence of the ATPG test pattern generation mode on the column-matching is studied in this paper. A new method, where more than one vectors per one fault are exploited, is proposed. This gives the algorithm more freedom in the column matches choice, which yields smaller BIST area overhead. This area reduction sometimes reaches 50% of the original circuit.

A heuristic static test set compaction method is proposed. It is based on joining test vectors, together with their fault masks. A significant reduction in the number of test vectors is obtained, whereas many test don't cares are retained.

Our BIST method can be used for any fault model, if a proper fault simulator and ATPG tool are provided. The fault coverage reached depends only on the ATPG tool as well.

ACKNOWLEDGEMENT

This research was supported by a grant GA 102/04/2137 and MSM6840770014.

REFERENCES

[1] V.K. Agrawal, C.R. Kime and 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] H.J. Wunderlich and G. Keifer. *Bit-Flipping BIST*, Proc. ACM/IEEE International Conference on CAD-96 (ICCAD96), San Jose, California, November 1996, pp. 337-343

[3] N.A. Touba and E.J. McCluskey. *Altering a Pseudo-Random Bit Sequence for Scan-Based BIST*, Proc. of ITC'96, pp. 167-175

[4] N.A. Touba and E.J. McCluskey. *Bit-Fixing in Pseudorandom Sequences for Scan BIST*, IEEE TCAD, Vol. 20, No. 4, April 2001, pp. 545-555

[5] M. Chatterjee and D.K. Pradhan. *A BIST Pattern Generator Design for Near-Perfect Fault Coverage*, IEEE Transactions on Computers, vol. 52, no. 12, December 2003, pp. 1543-1558

[6] P. Fišer, J. Hlavička and H. Kubátová. *Column-Matching BIST Exploiting Test Don't-Cares*. Proc. 8th IEEE Europian Test Workshop (ETW'03), Maastricht (The Netherlands), 25.-28.5.2003, pp. 215-216

[7] P. Fišer and H. Kubátová, „*An Efficient Mixed-Mode BIST Technique*", Proc. 7th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop 2004 (DDECS'04), SK, 18.-21.4.2004, pp. 227-230

[8] H.K. Lee and D.S. Ha. *Atalanta: an Efficient ATPG for Combinational Circuits*. Technical Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, Blacksburg, Virginia, 1993

[9] R. K. Brayton, et al, „Logic Minimization Algorithms for VLSI Synthesis", Boston, MA, Kluwer Academic Publishers, 1984

[10] J. Hlavička and P. Fišer, „BOOM - a Heuristic Boolean Minimizer". Proc. International Conference on Computer-Aided Design ICCAD 2001, San Jose, California (USA), 4.-8.11.2001, pp. 439-442

[11] P. Fišer and J. Hlavička, „BOOM - A Heuristic Boolean Minimizer", Computers and Informatics, Vol. 22, 2003, No. 1, pp. 19-51

[12] F. Brglez and 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

[13] F. Brglez, D. Bryan and K. Kozminski. *Combinational Profiles of Sequential Benchmark Circuits*, Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989

[14] H.K. Lee and D.S. Ha. *An Efficient Forward Fault Simulation Algorithm Based on the Paralel Pattern Single Fault Propagation*, Proc. of the 1991 International Test Conference, pp. 946-955, Oct. 1991

[15] P. Fišer and H. Kubátová, „*Influence of the Test Lengths on Area Overhead in Mixed-Mode BIST",* Proc. 9th Biennial Baltic Electronics Conference (BEC'04), Tallinn (Estonia), 3.-6.10.2004, pp. 201-204

TABLE 2.
COMPARISON WITH SIMPLE COLUMN-MATCHING

bench	*PR/det*	*vct/flt*	*compact*	*M*	*time [s]*	*overhead*	*improvement*
c1908	1 K / 1 K	1	36	28/33	6.7	5.7 %	
		10	340	30/33	55.9	3.0 %	48 %
c3540	1 K / 1K	1	31	48/50	3.9	2.2 %	
		10	101	48/50	19.1	1.6 %	27 %
		100	555	49/50	90.0	1.3 %	42 %
c7552	10 K / 1K	1	106	152/207	1104.8	17.0 %	
		10	1206	159/207	16124.7	14.8 %	13 %
s1196	1 K / 1 K	1	55	27/32	5.5	11.1 %	
		10	259	28/32	109.0	7.8 %	30 %
s1238	3 K / 1 K	1	33	27/32	2.9	6.7 %	
		100	95	28/32	16.7	4.6 %	31 %
s5378	10 K / 1 K	1	19	214/214	7.7	1.5 %	
		100	258	213/214	181.5	0.9 %	40 %
s9241.1	200 K / 1 K	1	52	224/247	160.7	5.3 %	
		10	564	225/247	3508.6	4.9 %	10 %