Scalable Test Pattern Generator Design Method for BIST

Petr Fišer, Hana Kubátová
Department of Computer Science and Engineering
Czech Technical University
Karlovo nam. 13, 121 35 Prague 2
e-mail: fiserp@fel.cvut.cz, kubatova@fel.cvut.cz

Abstract

A scalable built-in self-test (BIST) equipment design method for combinational or full-scan circuits based on a design of a test pattern generator producing vectors detecting 100% of stuck-at faults is proposed in this paper. Basic principles of the proposed BIST design method are similar to well-known and commonly used methods like bit-fixing, bit-flipping, etc. We introduce a new TPG design algorithm, which offers a good scalability, in terms of the test time, BIST area overhead and the BIST design time. The basis of the test pattern generator is a combinational block - the Decoder, transforming pseudo-random code words into deterministic test patterns pre-computed by an ATPG tool. The Column-Matching algorithm to design the decoder is proposed. Maximum of output variables of the decoder is tried to be matched with the decoder inputs, yielding the outputs be implemented as mere wires, thus without any logic. No memory elements are needed to store the test patterns.

1. Introduction

With the ever-increasing complexity of present VLSI circuits, their testing is becoming more and more important. Using only external test equipment (ATE) to test the chips is becoming impossible, mainly due to a huge amount of test vectors, long test time and very expensive ATE. Incorporating the Built-in Self-Test Equipment (BISTE) becomes inevitable. It requires no external tester to test the circuit, since all the circuitry needed to conduct the test is included in the very circuit. This is paid by an area overhead, long test time and often low fault coverage. Up to now, many BIST design methods have been developed [1-3], all of them trying to find some trade-off between these four aspects that cannot be all satisfied in one time: the *fault coverage*, *test time*, *BIST area overhead* and the *BIST design time*.

To reach high fault coverage, either a long test time or a high area overhead is involved. A pseudo-random testing established the simplest trade-off between all these 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. To improve the fault coverage and to reduce the test time, many enhancements of this pseudo-random principle have been developed. All of them are accompanied by some additional area overhead.

Different ASIC designers integrating BIST logic into their circuits have different requirements. Sometimes there is a requirement to design the BIST logic as soon as possible, regardless the area overhead and the fault coverage (to some extent, of course). For low-power designs, the BIST logic area overhead should be kept as small as possible, whereas the BIST design time is not that important. Or, and this is the most common case in practice, high fault coverage is important, whereas the BIST design time plays a small role.

We propose a flexible way how to design test pattern generators (TPGs) meeting *any* of the above-mentioned restrictions (or, better, quality measures). The designer is able to freely adjust the BIST logic design runtime, BIST logic area overhead and BIST run time, according his preferences. 100% fault coverage (of the non-redundant faults) is considered in the following text. However, the method may be modified so that less fault coverage is reached, with a benefit of less area overhead.

2. Proposed TPG Design Method

The proposed test pattern generator (TPG) consists of two main parts: an LFSR producing pseudorandom patterns and the Decoder, which is a combinational block transforming these patterns into deterministic test vectors computed by an ATPG tool. Generating a fully deterministic test detecting 100% stuck-at faults would involve a huge combinational logic (of the Decoder). Hence, a mixed-mode BIST is used. The BIST run is divided into two phases: the pseudorandom and deterministic one. The difference between our mixed-mode BIST method and the others (like bit-flipping [1], bit-fixing [2]) is that the two phases are disjoint. First, the easy-to-detect faults are covered in the *pseudo-random* phase. Then, a set of deterministic test vectors covering the undetected faults is computed and these tests are then generated by a transformation of the subsequent LFSR patterns. This significantly reduces both the *decoder* and *BIST control logic*. No memory elements are needed to recognize patterns that are to be modified (like in [2]); switching between the two phases is handled by the BIST controller counter.

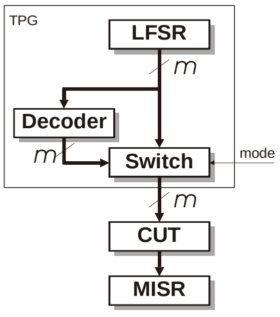

Figure 1. Proposed BIST scheme

3. The Decoder Design

The Decoder is designed by the column-matching algorithm proposed here.

Let us have an *n*-bit LFSR running for *p* clock cycles. The code words generated by this LFSR are described by a **C** matrix (*code matrix*) of dimensions (*p*, *n*). These code words are to be transformed into deterministic test patterns computed by an ATPG tool. The patterns are described 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*. The 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 coupling 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 transforming *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 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*.

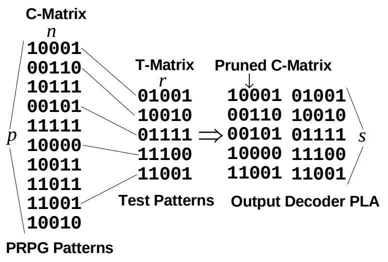

Figure 2: Assignment of the rows

3.1. The Column-Matching Algorithm

Now there is the task to assign the rows to each other, so that the Decoder logic will be as small as possible. The column-matching algorithm has been developed for this purpose. The principle of the algorithm 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 (outputs of the decoder); they are implemented as simple wired connections. This idea can be extended to a *negative matching*, by allowing negated columns to be matched. 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_0 (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 [4] or BOOM [5, 6], which has been developed especially for this purpose.

For more detailed description of the column-matching algorithm see [7, 8].

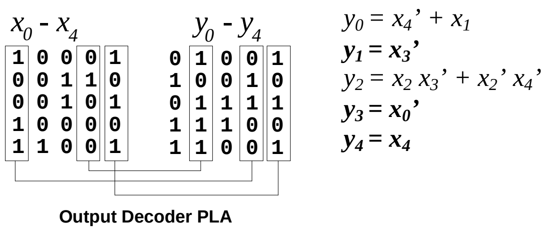

Figure 3: Column-matching example

3.2. Mixed-Mode Column-Matching BIST Example

As it was stated before, the test is divided into two phases – the pseudorandom and deterministic one. An artificial illustrative example is shown in Fig. 4. The BIST logic for a 5-input circuit is to be synthesized here. A 5-bit LFSR is run for 5 cycles first, by which 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. 5.

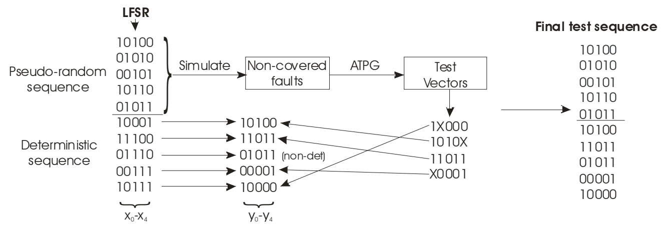

Figure 4: Test sequence generation

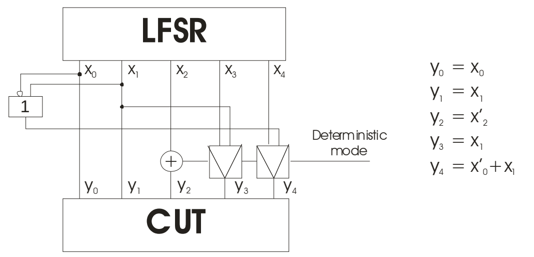

Figure 5: Resulting BIST circuitry

4. The Scaling

The BIST equipment design methodology should cope with the designer's needs, thus be as scalable as possible. The above mentioned four aspects (area overhead, fault coverage, test time, design time) play a big role in the overall design and cannot be optimally satisfied all.

The column-matching based TPG design method is very well scalable in all these aspects. The ways of scaling are described in this section. The fault coverage aspect will not be discussed here, since 100% of detected faults is assumed, see the Introduction.

4.1. Lengths of the Phases

Parameters that most essentially influence the TPG area overhead and BIST design time are the lengths of the two BIST phases. Of course, the BIST execution time is given by the lengths of the phases directly.

The aim of the pseudo-random phase is to detect as many faults as possible, while keeping the test time acceptable. Two aspects play role here: the LFSR polynomial and seed and the test length. Several methods computing the LFSR seed to achieve a good fault coverage have been proposed [9, 10]. However, for simplicity, we just repeatedly select the seed randomly, evaluate the fault coverage reached by using it, and select the best one. This approach allows us to reach a good fault coverage as well, whereas the fault coverage may be almost arbitrarily improved, for a cost of the runtime (by increasing the number of repetitions). Moreover, since the method is not based on any algebraic computations, it is applicable to any type of pseudo-random pattern generators, e.g., cellular automata.

The number of the covered faults as a function of the number of LFSR cycles applied to the CUT follows the saturation curve. First few vectors detect the majority of faults, and then the fault coverage increases only slightly. Thus, the pseudo-random phase should be stopped when the fault coverage does not improve for a given number of clock cycles. This number can be freely adjusted, according to the application specific requirements (the trade-off between the test time and area overhead).

To illustrate the scalability of the method in terms of the length of the pseudo-random phase, the BIST structure was designed for several ISCAS benchmarks [11, 12]. The results are shown in Table 1. The benchmark name is shown in the first column. The *"PR"* column indicates the length of the pseudo-random phase, the *"UD"* column shows the number of faults that were left undetected in this phase. The length of the deterministic phase was set constantly to 1000 clock cycles. The *"GEs"* column shows the total complexity of the column-matching BIST design, in terms of the gate equivalents [13]. The time needed to complete the column-matching procedure is indicated in the last column. The experiment was run on a PC with 1 GHz Athlon CPU, Windows XP.

Table 1: The pseudo-random phase length

bench	*PR*	*UD*	*GEs*	*Time [s]*
s5378	5 K	89	65.5	2259
	10 K	63	31.5	767
	20 K	48	16.5	104
s9234.1	1 K	1674	883	52 300
	50 K	773	333.5	4 400
	200 K	599	212.5	1 600
s13207.1	1 K	1793	699	208 K
	10 K	617	280	3 480
	50 K	182	36	128

A big trade-off between the test length and the area overhead can be seen here. The longer the pseudo-random phase runs, the less area overhead is reached. Consequently, the BIST synthesis time reduces as well.

In the deterministic phase we synthesize deterministic vectors from some of the LFSR patterns that follow the pseudo-random phase. With increasing number of LFSR patterns the chance for finding more column matches increases as well. This is due to having more freedom for selecting the LFSR vectors to be assigned to the deterministic vectors. However, the design runtime rapidly increases with the number of vectors. This is illustrated by Table 2. Its format is retained from Table 1, the "*Det.*" column indicates the length of the deterministic phase. It may be observed that a trade-off between the test time and area overhead can be freely adjusted here too, according to the demands of the BIST designer. The lengths of both the phases significantly influence the BIST design time as well. The design process is being sped up when increasing the length of the pseudo-random phase, since the number of deterministic vectors is being reduced this way. On the other hand, an increasing length of the deterministic phase slows down the Decoder design process.

Table 2: Influence of the the length of the deterministic phase

bench	*inps*	*PR*	*Det.*	*GEs*	*Time [s]*
c3540	50	1000	200	34	0.32
			500	29.5	0.52
			2000	16.5	1.47
			5000	77.5	2.93
s1196	32	5000	200	25.5	0.17
			500	25	0.32
			5000	9	2.16
			10000	4.5	5.83

4.2. Scaling the LFSR Width

As it was said before, the column-matching algorithm is primarily designed for a test-per-clock BIST. The number of the LFSR bits has either be equal to the number of CUT inputs (which involves a large LFSR overhead), or it may be scaled down by using a weighting logic. The effect of such a scaling will be shown here.

When the weighted pattern testing is used [14, 15], a new block has to be introduced into the BIST design – the *weighting logic* block, see Fig. 6. The LFSR width (*r*) may be then less than the number of CUT inputs (*m*), the number of TPG outputs is increased by applying the weighting logic block.

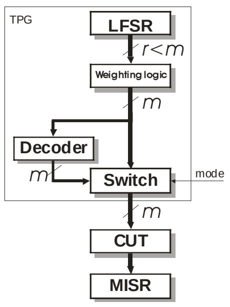

Figure 6. Weighted column-matching BIST scheme

The effects of the LFSR scaling are shown in Table 3 for the ISCAS s13207.1 benchmark having 700 inputs and compared with a standard approach (when no weights are used). The "*LFSR*" column indicates the width of the LFSR used (*r*). In a case of weighted pattern testing the number of gates needed for the weighting logic is shown in the next column. The total TPG area overhead (including the LFSR) is computed in terms of gate equivalents [13]. An *n*-input NAND gate counts for 0.5*n* GEs, the two-input XOR gate (used in LFSR) is 2.5 GEs. The size of a D flip-flop is considered to be 4 GEs. We have tried to scale down the originally 700-bit LFSR to 30 bits. It can be seen that optimum results are obtained for the 50-bit LFSR, the TPG area reduction is more than 70% with respect to the original (700-bit unweighted) case. When the LFSR width is scaled down more, the TPG area overhead rapidly increases, since the weighted LFSR is not able to cover enough faults, due to reduced randomness of the patterns. The column-matching results for the 30-bit LFSR are not present, due to very high column-matching algorithm runtimes.

Table 3: LFSR scaling

LFSR (r)	*w. gates*	*time [s]*	*GEs*
700 (no weights)	-	4720	2975
700 (3 weights)	518	245	3361
200 (3 weights)	365	653	1231
50 (3 weights)	365	2835	671
45 (3 weights)	365	3423	693
40 (3 weights)	365	29434	1288
30 (3 weights)	365	-	-

4.3. Multiple-Vector Column-Matching

The more "freedom" has the column-matching algorithm in selection of the matches, the better it performs. Some ATPG tools [16] are able to produce more than one test pattern per one fault. This can be efficiently exploited by the column-matching algorithm. The test set is then much larger, yielding the column-matching process be slower. However, due to more freedom for a column match selection, the area of the Decoder is less. This is documented by Table 4. The "*vct/flt*" column indicates the number of test vectors per one fault. The total number of generated test vectors is shown in the next column. The TPG design time and its area overhear (wrt. the original circuit) is shown next. The improvement with respect to the original, one-vector, method is indicated in the last column.

Table 4: Multiple-vector column-matching

bench	*vct/flt*	*vcts.*	*time [s]*	*overhead*	*impr.*
c1908	1	36	6.7	5.7 %	
	10	340	55.9	3.0 %	48 %
c3540	1	31	3.9	2.2 %	

bench	*vct/flt*	*vcts.*	*time [s]*	*overhead*	*impr.*
c3540	10	101	19.1	1.6 %	27 %
	100	555	90.0	1.3 %	42 %
c7552	1	106	1104.8	17.0 %	
	10	1206	16124.7	14.8 %	13 %
s1196	1	55	5.5	11.1 %	
	10	259	109.0	7.8 %	30 %
s1238	1	33	2.9	6.7 %	
	100	95	16.7	4.6 %	31 %
s5378	1	19	7.7	1.5 %	
	100	258	181.5	0.9 %	40 %
s9241.1	1	52	160.7	5.3 %	
	10	564	3508.6	4.9 %	10 %

5. Experimental Results

5.1. Comparison with Other State-of-the-Art Methods

The proposed column-catching method is compared with the bit-fixing accompanied by a "bit-correlating" ATPG [2], the "3-Weight Weighted Random BIST" proposed in [15] and the row matching method [3]. The comparison is shown in Table 5. The *"TL"* columns indicate the total length of the test, the *"GEs"* columns give the number of gate equivalents of the BIST combinational circuits and the "*lit.*" columns indicate the number of literals in the sum-of-product (SOP) form of the decoding logic. Test lengths have been set approximately equal. The empty cells indicate that the data for the respective circuit was not available.

Table 5: Comparison results

	Bit-fixing [2]		Weighted BIST [15]		Row matching [3]		Column Matching	
Bench	*TL*	*lit.*	*TL*	*lit.*	*TL*	*GEs*	*TL*	*GEs*
c880	-	-	-	-	640	21	1 K	15
c1355	-	-	-	-	1.8 K	0	1.5 K	15
c1908	-	-	-	-	4.7 K	8	3 K	10.5
c2670	10 K	385	8 K	269	6 K	119	5 K	113
c3540	-	-	-	-	4.8 K	4	5.5 K	1.5
s420	10 K	59	1.4 K	67	-	-	1 K	24.5
s641	10 K	98	768	45	7.7 K	6	4 K	15
s713	-	-	-	-	4.8 K	4	5 K	16.5
s838	10 K	183	3.1 K	108	-	-	6 K	130
s1196	10 K	97	16.8 K	67	10 K	36	10 K	6
s1238	-	-	17 K	33	-	-	4 K	26.5
s5378	10 K	332	18.4 K	68	-	-	11 K	19.0

5.2. Results for Standard Benchmarks

Since the comparison shown in Table 5 describes results for a few small benchmark circuits only, we will present a more exhaustive result table (Table 6), for some of the "bigger" and hard to test ISCAS [11, 12] and ITC'99 [17] benchmarks. The BIST circuitry was synthesized in two modes for each benchmark – first, the test length was set to be relatively small (the white rows). In the second mode a big effort has been put to obtain low area overhead in a reasonable time. The test is prolonged and some improvement techniques are sometimes used. This is indicated in the *"method"* column. The legend to the values is below the table. The *"inps"* column indicates the number of the benchmark inputs, the *"TL"* column gives the lengths of the two phases. The next column shows the total number of column matches (*M*) reached. The complexity of the switching logic is shown in the *"SW GEs*" column, the complexity of the output decoder in *"OD GEs"*. These numbers are summed together in the *"Total GEs*" column and the percentage of the area overhead of the Output Decoder and Switch, with respect to the CUT GEs is shown in the "*BIST Overhead*" column. The runtime needed to complete the column matching process is indicated in the last column.

6. Conclusions

A scalable mixed-mode BIST equipment design method based on a *column-matching* principle has been proposed. Pseudorandom LFSR code words are being transformed into deterministic test patterns. The transformation is being done by a purely combinational block; no additional registers are required.

The pseudo-random and deterministic phases are separated, which enables to reach less area overhead of the control logic. The lengths of both phases may be freely adjusted to find a trade-off between the test time and area overhead. It has been shown that the length of the pseudo-random phase has a crucial impact on the result. The length of the deterministic phase influences the result as well, though not that significantly.

A big scalability of the method, in terms of the area overhead, test time and design time is shown. A weighted pattern testing principle is used to reduce the LFSR width. Next, multiple-vector column-matching method reducing the area overhead is proposed.

The algorithm should serve as a basic guideline how to design more complex BIST designs, i.e., the multiple-scan chain based BIST, the STUMPS architecture, etc. The method should be as general, as the other state-of-the-art methods are (e.g., bit-flipping, bit-fixing). The obtained results, in terms of the area overhead, are comparable to the other methods, sometimes they are significantly better. The method has been tested on standard benchmarks and the results were compared with other state-of-the-art methods.

Acknowledgement

This research has been supported by MSMT under research program MSM6840770014.

References

[1] H.J. Wunderlich and G. Keifer. Bit-Flipping BIST, Proc. International Conference on CAD-96 (ICCAD96), San Jose, California, November 1996, pp. 337-343

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

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

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

[5] 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

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

[7] P. Fišer, H. Kubátová and J. Hlavička: Column-Matching BIST Exploiting Test Don't-Cares", Proc. 8th IEEE European Test Workshop, Maastricht, 2003, pp. 215-216

[8] P. Fišer and H. Kubátová: An Efficient Mixed-Mode BIST Technique, Proc. 7th IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop 2004, Tatranská Lomnica, SK, 18.-21.4.2004, pp. 227-230

[9] C. Fagot, O. Gascuel, P. Girard, C. Landrault: On calculating efficient LFSR seeds for built-in self test; Proc. IEEE European Test Workshop 1999 (ETW'99), Constance, Germany, 1999, pp. 7-14

[10] S. Hellebrand, H.-J. Wunderlich, A. Hertwig: Mixed-Mode BIST Using Embedded Processors; Journal of Electronic Testing Theory and Applications (JETTA), Vol. 12, Nos. 1/2, February/April 1998, pp. 127-138

[11] F. Brglez and H. Fujiwara, „A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan", Proc. of ISCAS 1985, pp. 663-698

[12] F. Brglez, D. Bryan and K. Kozminski, „Combinational Profiles of Sequential Benchmark Circuits", Proc. of ISCAS, pp. 1929-1934, 1989

[13] G. De Micheli, "Synthesis and Optimization of Digital Circuits", McGraw-Hill, 1994

[14] H.J. Wunderlich, „Self Test Using Unequiprobable Random Patterns", International Symposium on Fault-Tolerant Computing, 1987

[15] S. Wang. Low Hardware Overhead Scan Based 3-Weight Weighted Random BIST. In Proceedings of the 2001 IEEE international Test Conference (October 30 - November 01, 2001). ITC. IEEE Computer Society, Washington, DC, 868.

[16] 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, 199

[17] F. Corno, M. Sonza Reorda and G. Squillero, "*RT-Level ITC 99 Benchmarks and First ATPG Results*". IEEE Design & Test of Computers, July-August 2000, pp. 44-53

Table 6: Results for standard benchmarks

Bench	*inps*	*TL (PR + Det.)*	*method*	*M*	*SW GEs*	*OD GEs*	*Total GEs*	*BIST Overhead*	*Time [s]*
c7552	207	7 K + 1 K		131	261	325	586	19 %	500
		10 K + 1 K	2)	155	100	168.5	456.5	15%	887
s713	54	500 + 500		52	24	3	27	8 %	0.56
		3 K + 1 K		**54**	18	**0**	18	5 %	0.32
s1196	32	2 K + 1 K		28	13.5	23.5	37	7 %	1.20
		9 K + 1 K		**32**	6	**0**	6	1 %	0.04
s9234	247	50 K + 1 K		208	163.5	156	319.5	8 %	350
		200 K + 1 K	1)	225	127.5	66	193.5	5 %	3500
s13207.1	700	1 K + 1 K		638	456	294	750	13 %	4000
		50 K + 1 K		**700**	36	**0**	36	< 1 %	13
s15850.1	611	10 K + 1 K		478	397.5	187	584.5	9 %	812
		100 K + 2 K		553	306	66.5	372.5	5 %	1244
s38417	1664	10 K + 1 K		1240	1365	1389.5	1389.5	17 %	24 K
		100 K + 2 K	2)	1503	1245	489	1734	11 %	17 K
s38584.1	1464	10 K + 1 K		1435	379.5	57.5	437	3 %	650
		100 K + 1 K		**1464**	165	**0**	165	1 %	34
b12	126	1 K + 1 K		117	37.5	45	82.5	9 %	40
		10 K + 1 K	1)	118	33	34	67	7 %	1080
b14	277	1 M / 2 K		84	318	8017	8335	141 %	170 K
		100 M / 1 K		90	328.5	2663.5	3319.5	56 %	100 K

1) 10 vectors per fault
2) 3-weights