Test Patterns Compression Technique Based on a Dedicated SAT-based ATPG

Jiří Balcárek
Dept. of Computer Science & Engineering
Czech Technical University in Prague, FEL
Prague, Czech Republic
balcaji2@fel.cvut.cz

Petr Fišer, Jan Schmidt
Faculty of Information Technology
Czech Technical University in Prague
Prague, Czech Republic
fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

Abstract— **In this paper we propose a new method of test patterns compression based on a design of a dedicated SAT-based ATPG (Automatic Test Pattern Generator). This compression method is targeted to systems on chip (SoCs) provided with the P1500 test standard. The RESPIN architecture can be used for test patterns decompression. These are decompressed in the scan chain during the test, no additional hardware is required. The main idea is based on finding the best overlap of test patterns. During the test generation, we are trying to *efficiently generate* patterns as candidates for an overlap, unlike other methods, which are based on efficient overlapping of pre-generated test patterns. The proposed algorithm takes advantage of an implicit test representation as a SAT problem instance. We show results of test patterns compression obtained for standard ISCAS'85 and '89 benchmark circuits. The results are compared with competitive test compression methods.**

Keywords: Test patterns compression, testing, ATPG, SAT

I. INTRODUCTION

Testing of digital circuits is quite a difficult task, for a huge amount test data needed to be delivered to the circuit under test (CUT). With the growing complexity of designs, scan-based test techniques are becoming a standard. The test patterns are shifted into the chain of scan registers (scan chain) through a serial interface and the circuit under test response is shifted out to the response compactor.

Generally, we can use a deterministic or pseudorandom test patterns set for testing. Both have their advantages and drawbacks. Pseudorandom testing may be easily realized on the chip by a linear-feedback shift register (LFSR) [1, 2] or by other automata [3]. Test patterns generated in such a way cover most of easily-detectable faults, but the patterns can be quite inefficient in covering random pattern resistant faults. Also a great number of test patterns are needed to generate. It means that the time consumption may grow up significantly.

On the other hand, we can generate deterministic test patterns for all detectable faults. These patterns, having a considerable data volume, need to be sent into scan chains. This data transfer is realized by a TAM (Test Access Mechanism), which creates an interface between ATE (Automatic Test Equipment) and the on-chip test mechanism.

Design requirements force us to make the TAM as narrow as possible, but sending test patterns through a narrow TAM may cause a considerable growth of the testing time. That is why the compression of the test patterns is needed to decrease the bandwidth between ATE and TAM.

There are several methods used for a test patterns compression. Many of these methods are based on using of some encoding, such as statistical codes [2, 4, 5, 6, 7], run-length codes [4, 5, 6, 8, 9], and Golomb codes [10], others are based on XOR networks [11, 12], hybrid patterns [13], EDT (Embedded Deterministic Test) [14] and reuse of scan chains [15]. Another approaches are based on test patterns compaction [16, 17] and overlapping [18, 19, 20, 21].

In this paper we introduce a novel test patterns compression algorithm *SAT-Compress* based on a design of a dedicated SAT-ATPG [22, 23]. The compression algorithm exploits an overlapping of test patterns. Unlike other compression methods based on this principle [18, 19, 20, 21], we do not use test patterns pre-generated by an ATPG; we generate them on the fly to reach the locally best overlap and maximize the compression. Test patterns compressed this way can be easily decompressed by the RESPIN decompression architecture [24], which is intended to test system on chip (SoC) cores by reusing scan chains.

The paper is organized as follows: first a brief summary of the previous work on test patterns compression is presented in Section II. Then the representation of the test vector set as an instance of the SAT problem is considered and basic principles of SAT-based ATPGs [22] are described (see Section III). The RESPIN decompression architecture is shown in Section IV.A new test patterns compression algorithm based on dedicated SAT-ATPG is introduced in Section V. and finally the experimental results and comparison with other compression methods are shown (see Section VI).

II. PREVIOUS WORK

The proposed test patterns compression method is basically based on finding the best overlap of test patterns pre-generated by an ATPG (Fig. 1). The test patterns are serially shifted into the scan chain. This idea was described in [21] for the first time. This algorithm generally tries to find contiguous and consecutive test patterns having the maximum overlap. Deterministic test patterns are generated by an ATPG and compacted. Patterns in the scan chain are checked whether they match with one or more test patterns which were not employed in the sequence yet. In [20], the pattern overlapping problem is

converted into a Traveling Salesman Problem (TSP), for which different heuristics have been proposed.

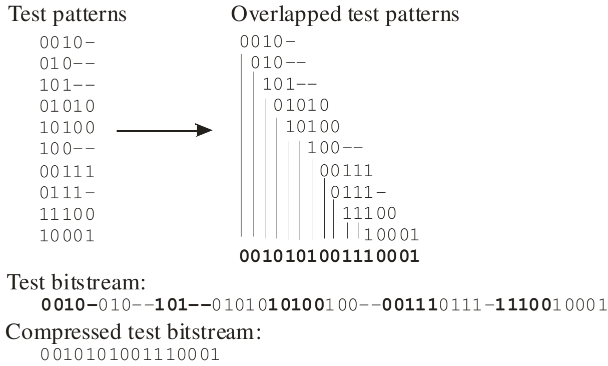

Figure 1. Test patterns overlapping

The COMPAS [18] test patterns compression tool is based on a similar approach, but it does not use compacted test patterns. Test patterns that are to be compressed are pre-generated by an ATPG as well. However, these test patterns should contain as many don't care bits (DC bits, means not specified) as possible. For this purpose, one test pattern for each fault from the fault list is generated Greater number of don't care bits grants the algorithm much more possibilities to combine test patterns and reach better compression. Other improvements are the simulation after every test pattern application and searching for best successors of a given starting pattern (usually an all-zero pattern). These improvements make COMPAS very efficient in comparison with other compression tools, see. Table II. The only weakness of COMPAS is the need for don't care bits. The number of DCs and the fact that the algorithm fully relies on a pre-generated test set can also affect the test compression ratio. When the test patterns are highly specified (they contain only few don't care's), it is much harder for COMPAS to find a good overlap of the test patterns and the efficiency of the test patterns encoding decreases which cause greater memory consumption [25].

Another compression technique based on the RESPIN architecture is presented in [15]. Suitable test patterns are produced by a standard ATPG tool performing dynamic compaction, while constraints to circuit inputs are applied. In contrast to this, we apply these constraints to the SAT instances, in a SAT-based ATPG.

The state-of-the-art compression/decompression architecture used in industry is the Embedded Deterministic Test (EDT) [14]. Here a high test compression is achieved by employing a dedicated but generic test decompressor. The compressed test patterns serve as seeds for a pseudo-random pattern generator ("ring register"), where they are decompressed and further distributed to scan-chains using a XOR-network structure ("phase-shifter"). The compressed test patterns are obtained as a solution of a set of linear equations. Similarly to the previously mentioned test compression methods, the test patterns to be compressed need to be pre-computed by an ATPG. Again, high amount of test don't cares is essential for achieving a good compression ratio [14].

III. SAT-BASED ATPG AND ITS PRINCIPLES

A. *Circuit to SAT Conversion*

Conversion of a digital circuit to a CNF (Conjunctive Normal Form) is an essential task for our SAT-based compression algorithm. The SAT instance in CNF represents in our case the whole set of test patterns detecting a given fault. Each SAT instance is described as a Boolean formula in CNF. A CNF Φ with m Boolean variables is a conjunction of n clauses, where each clause is a disjunction of literals. Each literal is a Boolean variable or its complement. The CNF Φ is satisfiable, if there exists some assignment of variables, for which all clauses are satisfied.

A Boolean variable is assigned to each signal in the circuit C. For each gate, the CNF Φ_g is derived from its characteristic function. The CNF Φ_c representing the circuit's function is constructed as a conjunction of the CNFs of all gates $g_1,\ldots,g_n \in$ C:

$$\Phi_C = \prod_{i=1}^{n} \Phi_{gi}$$

To generate the test for a fault F, the characteristic function Φ_f of the faulty circuit is generated. Only a necessary part of the circuit has to be included in the CNF Φ_f, for reasons of efficiency; circuit parts not propagating the fault may be omitted. Finally the SAT instance for a fault F is obtained as a product of Φ_C, Φ_f and a characteristic function of a XOR of the fault-free and faulty circuit outputs:

$$\Phi_{test}^{F} = \Phi_C \cdot \Phi_F \cdot \Phi_{XOR}$$

The Φ_{test}^{F} solutions represents the whole set of test patterns detecting the fault F. If Φ_{test}^{F} is unsatisfiable, the fault F is undetectable. More information can be found in [22, 23].

Example 1. A circuit having three gates is shown in Fig. 2. To detect a given fault, two copies of the circuit are created: the fault-free circuit and the circuit with a particular fault. In order to detect the fault, output values of these two circuits must differ. This is indicated by XORing their outputs (X and X'). Characteristic functions are derived from logic functions of the gates. For example, let us consider the AND gate D=A^B. For any two functions P and Q, P=Q is equivalent to $(P\Rightarrow Q)\wedge(Q\Rightarrow P)$. In this way the AND gate characteristic function is constructed as $(D\Rightarrow (A\wedge B))\wedge((A\wedge B) \Rightarrow D)$. Next we transform this expression to CNF, obtaining $(\neg D\vee A)\wedge(\neg D\vee B)\wedge(D\vee\neg A\vee\neg B)$.

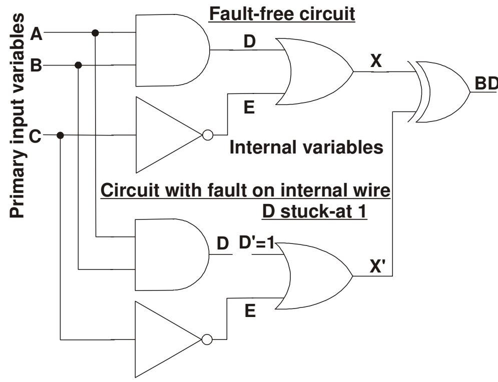

Figure 2. Faulty and fault-free circuit [23]

All gates in Fig. 2 are processed in this way, to obtain the final CNF for detecting the stuck-at-1 fault on D:

$$(\neg D \vee A)\wedge(\neg D \vee B)\wedge(D \vee \neg A \vee \neg B)\wedge(C \vee E)\wedge(\neg C \vee \neg E)\wedge(X \vee D) \wedge(X \vee \neg E)\wedge(\neg X \vee D \vee E)\wedge(D')\wedge(X' \vee \neg D')\wedge(X' \vee \neg E) \wedge(\neg X' \vee D' \vee E)\wedge(C \vee E)\wedge(\neg C \vee \neg E)\wedge(\neg X \vee X' \vee BD) \wedge(X \vee \neg X' \vee BD)\wedge(X \vee X' \vee \neg BD) \wedge(\neg X \vee \neg X' \vee \neg BD) \quad (1)$$

The assignment of input variables corresponds to the test pattern detecting this fault. In particular, the example CNF (1) is satisfied by assignment A=1, B=0, C=1, D=0, D'=1, E=0, X=0, X'=1, BD=1. Therefore, the stuck-at-1 at D fault is detected by the pattern (A, B, C) = (101).

B. *SAT-based ATPG*

A SAT-based ATPG algorithm [23] can be described in four steps:

1) Generate a fault list for a given circuit.

2) Pick one fault from the fault list and generate its CNF (CNF implicitly represents the whole set of the test patterns detecting a given fault).

3) Solve the CNF of the fault and obtain its particular solution which represents one test pattern detecting the respective fault. If the CNF is unsatisfiable, the fault is removed from the fault list as an undetectable fault.

4) Simulate the test pattern obtained in step 3 and remove all detected faults from the fault list. Repeat steps 2-4 until the fault list is empty.

For details on the SAT-based ATPGs, see [22, 23].

IV. TEST DECOMPRESSION

The decompression of compressed test patterns is conducted by the RESPIN (REusing Scan chains for test Pattern decompression) [15, 24] architecture. This architecture is intended for testing systems on chip (SoCs). Basically, scan chains of different cores are *reused* for updating of the content of the tested core scan chain (SC). RESPIN is based on the IEEE 1500 proposed standard for testing of embedded cores [26]. Each core has a serial test access, which is a mandatory element of IEEE 1500 and a parallel access. A core test wrapper, which is the interface between the embedded core (core terminals) and its system on chip environment (rest of the integrated circuit and TAM), is defined. By definition, the core test wrapper is implemented on-chip and should have three mandatory modes:

1) Normal operation – the core is connected to its system IC (integrated circuit) environment and the wrapper is transparent.

2) Core-internal test – the TAM is connected to the core such that test patterns can be applied at the cores inputs and responses are observed at cores outputs.

3) Core-external test - the TAM is connected to the interconnect wiring and logic such that a test patterns can be applied at cores outputs and responses are observed at the cores inputs.

An example of RESPIN architecture for one SC is shown in Fig. 3.

This architecture consists of ETC (Embedded Test Core) and CUT. During the test, both are isolated from the rest of the SoC by a test wrapper. The decompression of test patterns is performed in the ETC, which is supposed to be a reused scan chain of another core (ETC SC). This ETC SC is extended only by a single multiplexer (controlled by signal *test* - t_c) and a feedback wire. Reusing of scan chains grants a minimal decompression hardware overhead.

The CUT scan chain is the scan chain of the tested core. It is filled by the decompressed test patterns from the ETC SC and, after application of the functional clock, the responses are shifted out to the signature analyzer (SA). SA compares responses with the expected signature and the circuit is marked as fault-free or faulty.

The control signal t_c (*test*) of the ETC SC is also connected to the scan enable signal of the CUT SC. This control signal selects between the serial mode (t_c = 0) and the circular mode (t_c = 1) of the ETC SC and between capture mode (t_c = 0) and scan mode (t_c = 1) of the CUT SC.

Table I. shows that while the CUT SC captures the system response (t_c = 0, capture mode) and shift it out to the signature analyzer (t_c = 1, scan mode). The ETC SC is in serial mode (t_c = 0) and loads one new bit from the encoded test data (compressed test bitstream). Thus only one bit per test pattern is scanned into the ETC SC from the ATE. Whenever the CUT SC scans in test data (t_c = 1, scan mode) from the ETC SC, the ETC SC is in circular mode and the feedback is used to shift valid part of the test pattern back to the ETC SC and reuse it in new test pattern. For using cores as ETC SC both their shift and capture clocks must be compatible with the CUT SC clocks and ETC SC must have at least as many scan cells as the CUT SC.

More specific details of the RESPIN architecture can be found in [15, 24].

The RESPIN architecture is suitable for decompression of the test patterns compressed by overlapping, because while the ETC SC preserves the useful bits of the test patterns by looping through the feedback, the CUT SC obtains responses on previous test pattern and shifts them to the

signature analyzer (SA) to be evaluated. At the same time a new test pattern is shifted from ETC SC into the CUT SC. Moreover, the hardware overhead is minimal since the decompression is made by reusing of the SCs of non-tested cores and no additional decompression hardware is required.

TABLE I. OPERATION MODES OF ETC SC AND CUT SC [15, 24]

t_c = scan enable	0	1
CUT	capture mode	scan mode
ETC	serial mode	circular mode

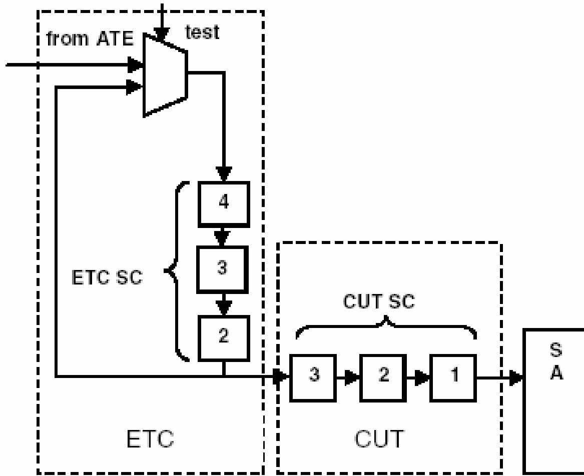

Figure 3. RESPIN testing architecture [15, 24]

V. THE SAT-COMPRESS ALGORITHM PRINCIPLES

A. Basic Idea

In our new approach we try to eliminate weaknesses of COMPAS. The main idea is not to overlap test patterns pre-generated by an ATPG, but to generate most suitable test patterns on the fly, to reach the (locally) best overlap. The basic question is how to find these test patterns. Each fault has its set of test patterns by which it is detected. If we were able to pick the right pattern for each fault in the right order, we could have reached the best possible compression of the test patterns.

Because explicit computation and storing of all these test patterns is inefficient (and mostly even infeasible), we were forced to find another, more efficient way of test patterns set representation.

We have researched possibilities of implicit representations of test patterns. We have found that we can take advantage of principles of SAT-based ATPGs and efficiently represent all test patterns for one fault *implicitly*, by one SAT problem instance in a CNF. The CNF test set representation is much less memory consuming than a standard tabular test set representation. Our proposed test set compression algorithm sequentially generates the compressed test bitstream. A SAT instance in CNF is generated for each fault. This SAT instance contains both variables representing the circuit's primary inputs and variables representing internal signals. The test pattern is determined by values of the primary input variables in the SAT solution; values of internal variables are of no significance. The variables not present in the generated CNF or in the SAT solution are set as don't cares (DCs). A SAT instance is solved with constraints given by the test pattern generated in the preceding algorithm step. A satisfiable solution of a SAT instance forms the next suitable test pattern for an overlap. Higher time consumption can be expected, because this method requires a repeated SAT solving. Our aim, however, is to find the best overlap and maximize the test patterns compression, as testing time, not test generation time, forms currently a bottleneck.

B. The SAT-Compress Algorithm Description

We try to find the best overlap by gradually building the compressed test patterns bitstream for the scan chain. The basic algorithm is shown in Fig. 5.

1) First, we generate a complete fault list (FL).

2) A zero state (all-zero test pattern) or the test pattern covering any fault from the fault list is used as the initial test pattern.

3) The pattern is simulated and all detected faults are deleted from the fault list. The leftmost bit of the pattern goes to the resulting bitstream (Fig. 4).

4) The pattern is shifted left and a DC bit is put to its rightmost position. This is the mask for the next pattern (Fig. 4, next pattern mask). The mask constraints the values of primary inputs (PI) in subsequent SAT instances. Only care bits generate the PI constraints.

5) To generate the next test pattern having the highest overlap with the previously generated one, a CNF for each fault in the fault list is generated, while the primary input variables are set according the mask. The orders of faults are of no significance. If a CNF for a fault is satisfiable for a given assignment of primary variables, a new test pattern is obtained. If none of these CNFs is satisfiable, the pattern is shifted one more bit left, which generates a new mask of SAT constraints.

6) These operations (3-6) are performed while the fault list is not empty or until all care bits from the mask of primary inputs setting are shifted out while there is still no satisfiable CNF (rest of the fault in FL is undetectable).

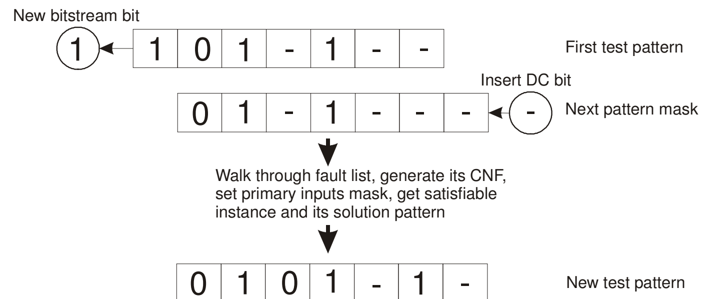

Figure 4. Next pattern generation example

```
1. Generate FL                                  (FL = fault list)
2. TP = 0                  (TP = test pattern = all-zero seed)
   mask = DC                                  (DC = don't care)
3. FL = FL - detected_by_simulation(TP)
   compressed_bitstream += TP[0]
4. mask = TP[1 .. n-1] + DC
5. do {
      for each fault in FL {
         Create CNF
         Apply mask to PIs in CNF
         if CNF is SAT {
            break the for loop
         }
      }
      TP = CNF_Solution
      FL = FL - detected_by_simulation(TP)
      compressed_bitstream += TP[0]
      mask = TP[1 .. n-1] + DC
   }
6. while (FL!=0 or (mask==DC and All_CNF==UNSAT))
```

Figure 5. The SAT-Compress algorithm

VI. EXPERIMENTAL RESULTS

The experimental results and a comparison of our algorithm with similar compression methods are presented in the following two subsections. Unfortunately, comparison with [15] could not be performed, since only results for 20 parallel scan-chains are presented in [15].

The measurement was performed on Intel Xenon CPU - 2GHz with 4GB RAM.

A. Comparison of Different Compression Techniques

A comparison of SAT-Compress with other state-of-the-art test compression techniques is presented in Table II. The first column "*Bench.*" represents the benchmark name. The compressed test lengths in bits, for seven different competitive methods, are shown then. A comparison of only seven biggest ISCAS'89 circuits is shown, since no more relevant data was available to us. The last column shows the compressed test data size for our proposed compression tool SAT-Compress. An all-zero initial test pattern for both COMPAS and SAT-Compress is used, thus the results are not influenced by different initial states.

It can be concluded from Table II., that our proposed algorithm can reach better compression of test patterns than most of the presented techniques; we are able to obtain results such as the state-of-the-art compression methods EDT or COMPAS and in some cases even better. The time consumptions of SAT-Compress are shown in Table III. The comparison between presented tools was not possible to measure because of the unavailability of source codes.

B. Comparison COMPAS and SAT-Compress Algorithms

In this Subsection we will present a more detailed comparison of SAT-Compress and COMPAS, as a representative of test compression algorithms based on overlapping of patterns.

Results for ISCAS'85 and '89 benchmarks [27, 28] are presented in Table III.

TABLE II. COMPAS AND SAT-COMPRESS TEST LENGTHS

Bench.	COMPAS			SAT-Compress			
	bits	*avg.*	*var.*	*bits*	*avg.*	*var.*	*time [s]*
c17	9	9	1.5	11	12.3	1	0
c432	195	218	30.6	189	198.2	21.8	3
c499	260	303.7	47.6	187	198.2	8.7	6
c880	540	412.7	44.3	410	561.5	60.99	10
c1355	1040	1126	68.6	349	-	-	74
c1908	1009	989.8	49.6	624	-	-	229
c2670	6553	5940	269.7	2223	-	-	2305
c3540	747	743.7	32.9	1622	-	-	3569
c5315	1255	1159.5	64.8	881	-	-	156
c6288	82	95.2	36.6	97	-	-	165
c7552	6005	6430.7	345.2	4840	-	-	7406
s27	16	13.2	2.3	23	21.5	2.1	0
s208	130	124.1	9.1	202	209	15.9	0
s298	101	79.6	5.5	137	139.7	8.2	0
s344	85	80.9	6.7	116	114.7	10.2	0
s349	85	80.1	6.8	116	114.4	10	0
s382	123	111.3	6.6	191	179.2	12.8	0
s386	264	255	10.8	304	319	12.5	4
s400	121	109	7.2	173	170.9	9.7	0
s420	352	315.9	29.1	624	574.8	46.4	11
s444	116	107	7.5	160	150.6	10.5	0
s510	160	156	8.9	210	211.8	10.3	2
s526	344	349.8	18.4	521	482.8	22.7	4
s526n	344	350.2	18.1	493	-	-	3
s641	397	393.3	24.2	710	670.7	28.4	15
s713	428	403.8	26.1	642	672	38.7	32
s820	460	504.6	21.9	697	697.7	26.3	28
s832	494	498.8	19.5	729	690.7	24.1	36
s838	920	762.3	79.6	1800	1638.3	103.5	136
s953	723	700.4	24.7	825	-	-	51
s1196	740	738.5	23.5	1211	1202.2	41.9	173
s1238	741	769.2	24.3	1300	1268.7	41.7	365
s1423	596	621.5	38.6	794	827.6	43.9	72
s1488	488	461.5	15.1	546	-	-	20
s1494	431	451.3	16.6	573	-	-	25
s5378	2148	1995.6	73.8	2407	-	-	631
s9234	11594	11309.6	310.7	9928	-	-	68119
s13207	4163	-	-	10457	-	-	67751
s15850	8234	-	-	12987	-	-	114932
s38417	24198	24926.3	1717.7	19291	-	-	91713
s38584	7291	-	-	14271	-	-	122143

The first column "*Bench.*" presents the name of the benchmark. The compressed test lengths generated by COMPAS and SAT-Compress, both starting with an all-zero test pattern, are shown in the column "*bits*". Then we have tried to repeatedly run the algorithms starting with different

initial test patterns. The average compressed test patterns lengths are shown in the columns "*avg.*" and average variations of compressed test patterns lengths are shown in the columns "*var.*". The time consumptions of test patterns compression for SAT-Compress algorithm in seconds are shown in the "time" column. The results of this measurement show that the bitstream length for both tools significantly depends on the initial test pattern and starting with an all-zero seed (which is the default setting for COMPAS) can produce outstandingly poor results, out of the range of the variation (see, e.g., s298 for COMPAS). In some cases, our tool reaches much better compression than COMPAS, but it may also fail. For an unknown reason the efficiency of proposed algorithm is much better for ISCAS'85 benchmarks than for ISCAS'89 ones. As can be seen from Table III., the time consumption may be considerable for larger circuits, but we suppose, that scalability of proposed algorithm may grow up by using of more scan chains and division of the circuit into smaller parts.

Figs. 6, 7, and 8 show examples of distributions of bitstream lengths using different starting patterns for COMPAS and SAT-Compress. In both cases we always start with a test pattern covering one particular fault. The number of restarts is equal to the number of faults.

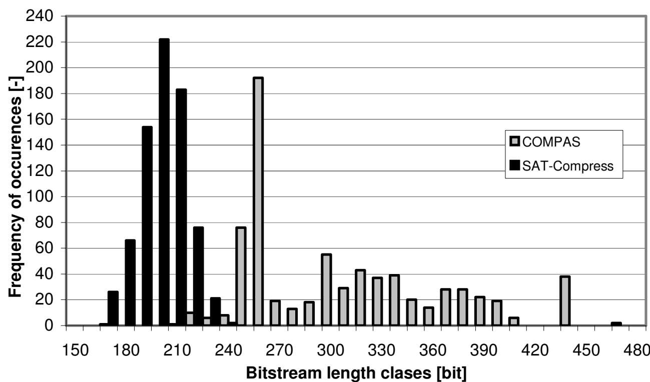

Figure 6. Frequency of bitstream length distribution (c499)

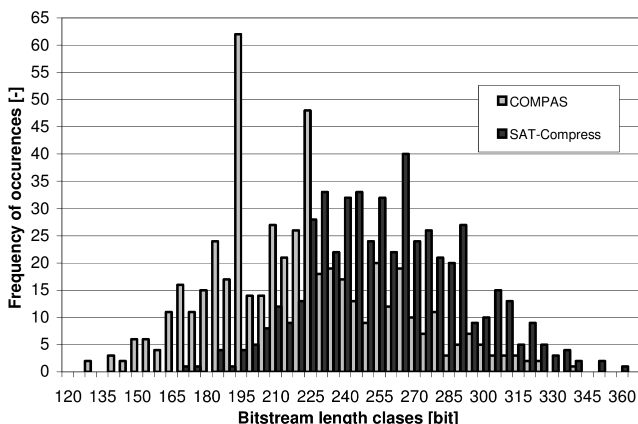

Figure 7. Frequency of bitstream length distribution (c432)

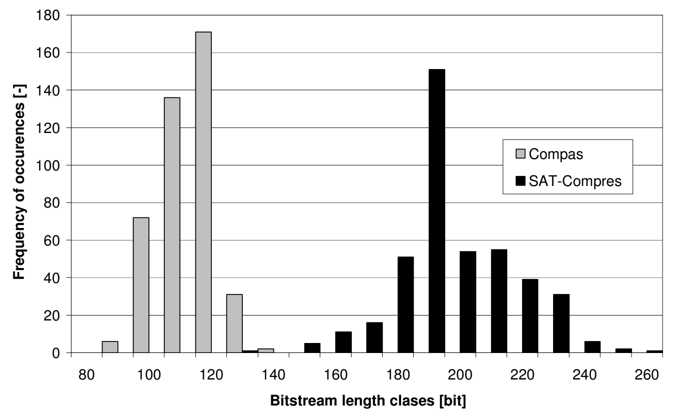

Figure 8. Frequency of bitstream length distribution (s400)

As can be seen, the compressed bitstream length seems to have Gaussian-like distribution and the only difference between SAT-Compress and COMPAS characteristics is their displacement. This Gaussian distribution of the compressed bitstream lengths for different starting seeds is a common characteristic of these two tools for all tested benchmarks. It can be seen that the selection of the initial test pattern has a crucial impact on the resulting compressed test length, for both algorithms. The way of its proper choice will be a topic of our further investigation.

VII. CONCLUSION

New test patterns compression algorithm (SAT-Compress) based on a modification of SAT-based ATPG is presented. This algorithm utilizes a CNF (Conjunctive Normal Form) implicit representation of test patterns and tries to compress the test patterns by overlapping. In contrast to competitive state-of-the-art test compression techniques, the proposed algorithm does not rely on a pre-generated test set; most suitable test patterns are being generated on the fly.

The test decompression is based on a generic RESPIN architecture, where the patterns are being decompressed in the tested circuit scan-chain. Thus, the circuit needs not be modified to apply the test.

The proposed method was compared with seven state-of-the-art test compression algorithms and a detailed comparison with COMPAS has been made. The comparison results seem to be promising – SAT-Compress achieved the best test compression ratio for many benchmark circuits. Moreover, there are yet many ways of possible improvements. These will be a topic of our further research. Possibilities of application of the method to multiple scan-chain designs will be also studied in the future.

ACKNOWLEDGMENT

This research has been supported by MSMT under research program MSM6840770014 and by the grant of the Czech Grant Agency GA102/09/1668.

References

[1] B. Koenemann, “LFSR-Coded Test Patterns for Scan Designs,” in Proc. of European Test Conf., IEEE CS Press, 1991, pp. 581-590.

[2] C.V. Krishna, N.A. Touba, “Reducing Test Data Volume Using LFSR Reseeding with Seed Compression,” in Proc. of the International Test Conference, 2002, pp. 321-330.

[3] F. Fummi, D. Sciuto, “Implicit test pattern generation constrained to cellular automata embedding,” 15th IEEE VLSI Test Symposium (VTS'97), 1997, pp.54-60.

[4] P. T. Gonciari, B. M. Al-Hashimi, N. Nicolici, “Variable length input Huffman coding for system-on-a-chip test,” IEEE Trans. Computer-Aided Design, vol. 22, 2003, pp. 783–796.

[5] H. Ichihara, A. Ogawa, T. Inoue, and A. Tamura, “Dynamic test compression using statistical coding,” in Proc. ATS, 2001, pp. 143–148.

[6] H. Ichihara, K. Kinoshita, I. Pomeranz, and S. Reddy, “Test transformation to improve compaction by statistical encoding,” in Proc. Int. Conf. VLSI Design, 2000, pp. 294–299.

[7] A. Jas, J. Ghosh-Dastidar, and N. A. Touba, “Scan vector compression/decompression using statistical coding,” in Proc. VLSI Test Symp., 1999, pp. 114–120.

[8] A. Chandra, K. Chakrabarty, “Frequency-Directed Run-Length (FDR) Codes with Application to System-on-a-Chip Test Data compression,” in Proc. of VLSI Test Symposium, 2001, pp. 42–47.

[9] A. Jas and N. A. Touba, “Test vector decompression via cyclical scan chains and its application to testing core-based designs,” in Proc. ITC, 1998, pp. 458–464.

[10] A. Chandra, K. Chakrabarty, "Test Data Compression for System-on-a-Chip Using Golomb Codes," 18th IEEE VLSI Test Symposium (VTS'00), 2000, pp.113-121.

[11] I. Bayraktaroglu and A. Orailoglu, “Test volume and application time reduction through scan chain concealment,” in Proc. Design Automation Conf., 2001, p. 151–155.

[12] S. M. Reddy, K. Miyase, S. Kalihara, and I. Pomeranz, “On test data volume reduction for multiple scan chain design,” in Proc. VLSI Test Symp., 2002, pp. 103–108.

[13] D. Das and N. A. Touba, “Reducing test data volume using external/ LBIST hybrid test patterns,” in Proc. ITC, 2000, pp. 115–122.

[14] Rajski, J., “Embedded Deterministic Test” IEEE Trans. on CAD, vol. 23, No. 5, 2004, pp. 776-792.

[15] R. Dorsch and H.-J. Wunderlich, “Tailoring ATPG for embedded testing,” in Proc. ITC, 2001, pp. 530–537.

[16] Seiji Kajihara, Kewal K. Saluja, "On Test Pattern Compaction Using Random Pattern Fault Simulation," vlsid, Eleventh International Conference on VLSI Design: VLSI for Signal Processing, 1998, pp.464-470.

[17] I. Pomeranz, S. M. Reddy, ”A New Approach to Test Generation and Test Compaction for Scan Circuits,“ Proc. of the conference on Design, Automation and Test in Europe (DATE’03), 2003, pp. 11000-11006.

[18] O. Novák, J. Zahrádka, “COMPAS – Compressed Test Pattern Sequencer for Scan Based Circuits,” in Proc. of EDCC, 2005, pp. 403-414.

[19] C. Dufaza, H. Viallon, C. Chevalier, "BIST hardware generator for mixed test scheme," edtc, European Design and Test Conference (ED&TC '95), 1995, pp. 424-431.

[20] C. Su, K. Hwang, “A Serial Scan Test Vector Compression Methodology,” in Proc. ITC, 1993, pp. 981-988.

[21] Daehn, W., Mucha, J,: Hardware Test Pattern Generation for Built-in Testing. Proc. of IEEE Test Conference, 1981, pp. 110-113.

[22] T. Larrabee, “Test Pattern Generation Using Boolean Satisfiability,” IEEE Transactions on Computer-Aided Design, 1992, pp. 4-15.

[23] Drechsler, R., Eggersglüß, S., Fey, G., Tille, D., “Test Pattern Generation using Boolean Proof Engines,” Publisher Springer Netherlands, ISBN 978-90-481-2360-5, 2009, XII, p. 192.

[24] Schafer L., Dorsch R., Wunderlich H.J., “RESPIN++- Deterministic Embedded Test,“ Proc. of the European Test Workshop, 2002, pp.37-42.

[25] Jeníček J., Novák O.:A Test Pattern Compression Based on Pattern Overlapping, Proc. of DDECS 2007, Apr. 2007, Krakow, Poland, pp.29 - 34, ISBSN: 1-4244-1161-0.

[26] E. J. Marinissen, Y. Zorian, R. Kapur, T. Taylor, and L. Whetsel, “Towards a Standard for Embedded Core Test: An Example,” in Proceedings of the IEEE International Test Conference (ITC), pp. 616–627, IEEE, 1999.

[27] 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.

[28] 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.

[29] Janak H. Patel, Ilker Hamzaoglu, "Test Set Compaction Algorithms for Combinational Circuits," iccad, International Conference on Computer-Aided Design (ICCAD '98), 1998, pp.283-289.

[30] Irion, A.; Kiefer, G.; Vranken, H.; Wunderlich, H.-J., “CircuitPartitioning for Efficient Logic BIST Synthesis,” Proc. DATE, 2001, pp.88-93.

[31] Hamzaoglu, I., and J.H. Patel, “Reducing Test Application Time for Full Scan Embedded Cores,” Proc. Of Int. Symp. on Fault Tolerant Computing, 1999, pp. 260-267.

[32] A. Chandra and K. Chakrabarty, “Test Data Compression and Test Resource Partitioning for System-on-a-Chip Using Frequency-Directed Run-Length (FDR) Codes,” IEEE Transactions on Computers, vol. 52, May 2003, to appear.

TABLE II. COMPARISON OF THE TEST DATA AMOUNT FOR DIFFERENT COMPRESSION TECHNIQUES

Bench.	MinTest [29]	Stat. Coding [7]	LFSR Reseeding [2]	Illinois Scan [30, 31]	FDR Codes [8, 32]	EDT [14]	RESPIN++ [24]	COMPAS [18]	SAT-Compress
s5378	20,758	15,417	6,180	14,572	12,346	-	17,332	2,148	2,407
s9234	25,935	19,912	12,112	27,111	22,152	-	17,198	11,594	9,928
s13207	163,100	52,741	11,285	109,772	30,880	10,585	26,004	4,163	10,457
s15850	58,656	49,163	12,438	32,758	26,000	9,805	32,226	8,234	12,987
s35932	21,156	-	-	-	22,744	-	-	1,860	5,096
s38417	113,152	172,216	34,767	96,269	93,466	31,458	89,132	24,198	19,291
s38584	161,040	128,046	29,397	96,056	77,812	18,568	63,232	7,291	14,271