PBO-Based Test Compression

Jiří Balcárek, Petr Fišer, and Jan Schmidt
Dept. of Digital Design
Czech Technical University in Prague
Prague, Czech Republic
{jiri.balcarek, petr.fiser, jan.schmidt}@fit.cvut.cz

***Abstract*—This paper presents a novel ATPG and test compression algorithm based on Pseudo-Boolean (PBO) optimization. Similarly to SAT-based ATPGs, the test for each fault is represented implicitly as a PBO instance. The optimization process solves the problem of maximizing the number of unspecified values in the test. A novel don't care aware circuit-to-PBO conversion procedure is presented. The obtained unspecified values in the test are efficiently exploited in test compression. The produced compressed test sequence is suited for the RESPIN decompression architecture, thus for testing systems-on-chip. The presented experimental results show the efficiency and competitiveness of the proposed method.**

Keywords-ATPG, test compression, Pseudo-Boolean Optimization, SoC testing , RESPIN.

I. INTRODUCTION

Testing of complex systems-on-chip (SoCs) is an everlasting challenge for designers. Due to huge test data volumes, compressed tests must be stored in the automated test equipment (ATE) and submitted to tested chips.

In the state-of-the-art practice, a universal automated test pattern generation tool (ATPG) [1], [2] is used first to *explicitly* choose a non-compacted test for a given circuit, which is then compressed [3], [4]. Such a two-steps process is suboptimal as it loses information.

Recently, *implicit* techniques are becoming popular in test generation [5], [6], [7] and compression. All test vectors for a given fault are represented as solutions of a Boolean satisfiability problem (SAT) instance. This is also the case of SAT-Compress algorithm [8] referenced in this paper. Using implicit test representations, we do not lose any information, when the test is generated and passed to the compression process.

All compressors discussed in this paper target the RESPIN architecture [9], which is aimed for SoC designs compliant with the IEEE P1500 standard [10].

The compression is based on *overlapping* of shifted test vectors [8], [9]. In an optimal case, values of two subsequent test vectors are displaced by one bit, with one bit added. Hence, the presence of test don't cares is essential for the efficiency of the compression.

Unfortunately, common SAT solvers [11], [12] return only one completely specified vector as a solution (satisfiability proof). Thereby, no unspecified bits are present in the test.

There are techniques to obtain incompletely specified SAT solutions [13]-[17], however, they do not distinguish primary inputs from the rest of variables.

In this paper we propose a Pseudo-Boolean Optimization (PBO) [18], [19] technique to generate incompletely specified test vectors. Similarly to SAT, the circuit is converted to a PBO instance, whereas the optimization criterion is set to maximize the number of unspecified values at the primary inputs only.

As our previous results indicated, blindly introduced unspecified values can do more harm than good [20]. Therefore, similarly to [20], we combine the PBO technique with fault simulation. In this process, the PBO generates test vectors with maximum of unspecified bits, which are subsequently assigned a value, so that the fault coverage is maximized.

II. PBO-BASED TEST COMPRESSION ALGORITHM

Principles of the proposed ATPG and test compression algorithm based on Pseudo-Boolean Optimization (let us call it *PBO-Compress*) will be described in this section.

Basically, the compressed bitstream is constructed gradually, bit by bit, by shifting the scan-chain content and adding one bit in each clock cycle. The values of the added bits are determined by solving PBO equations formed by a fault miter (see Subsection II.A) and constraints imposed by the current scan-chain content. Every PBO solution represents an updated scan-chain content.

A. Fault Miter Construction and Its PBO Conversion

In classical SAT-based ATPG algorithms [5], [6], a conceptual hardware (the *fault miter*) is typically constructed for each fault, as a copy of fault-free and faulty circuit, whereas their respective outputs are compared, forming a single-output circuit. The fault is detected under any assignment of primary inputs (PIs) for which the output equals to 1 (fault-free and faulty responses differ for one primary output at least). To find such an assignment, the miter is converted into a SAT instance (CNF). A variable is assigned to each signal (both primary inputs – PIs and internal signals) in the circuit. Each gate is then transcribed into a set of CNF clauses by Tseitin's transformations [21].

The SAT problem is then solved by a conventional SAT solver [11], [12] and a test vector is obtained as a satisfiability proof. If the instance is not satisfiable, the fault is redundant.

For our purposes, it is convenient to obtain a maximally unspecified test vector, that is, a solution with *maximum of unspecified values* at primary inputs. Therefore, solving an optimization version of SAT is needed.

There were several optimization versions of SAT solvers proposed [13], [14], [15], [16], [17], however, they are not suitable for our purpose for one reason, as they maximize the number of *all* don't care (unassigned) variables. We would like to maximize only unassigned values at the circuit primary inputs, i.e., only *some* CNF variables.

Pseudo-Boolean optimization [18] offers a flexible way of defining optimization criteria, moreover the PBO constraints greatly resemble SAT clauses. Therefore, a straightforward conversion of CNF-to-PBO constraints suggests itself:

1) Let $x_1, \dots, x_n$ be variables of the original SAT problem.
2) For each CNF clause $(l_1 + l_2 + \cdots + l_j)$, where l_i are individual literals (variables or their negations) construct an inequality $l_1 + l_2 + \cdots + l_j \geq 1$.
3) If a literal $l_i = x_k$ (variable in its direct form), substitute $l_i = x_k$ in the inequality.
4) If a literal $l_i = \overline{x_k}$ (variable in its negated form), substitute $l_i = (1 - x_k)$.

Still, all the variables are in the Boolean domain, while we need to encode unspecified values. For this purpose, we must use two Boolean variables to encode each literal, for example like this:

TABLE I. LITERAL ENCODING

x_i	x_i^V	x_i^A
0	0	1
1	1	1
U	any	0

The optimization criterion can be then formed as:

$$x_1^A + x_2^A + \cdots + x_{n-1}^A = min. \quad (1)$$

Where $x_1 \dots x_{n-1}$ are primary inputs.

Detecting a fault means to control *defined* values in the circuit, and to observe *defined* values at outputs. Hence, the propagation of undefined values must be observed, and *every* original CNF variable must be doubled in PBO.

B. Characteristic Functions for Tseitin's Transformation

During miter conversion, characteristic functions of all gates in the circuit in CNF form are added to the SAT instance. For a gate with inputs $x_1 \dots x_m$ and output y, the signature of the characteristic function F is $F: \{0,1\}^{m+1} \rightarrow \{0,1\}$. For our problem, we need the function $F: \{0,1,\mathrm{U}\}^{m+1} \rightarrow \{0,1\}$. The strategy is to calculate F in some form, then to encode it by TABLE I. into $F: \{0,1\}^{2m+2} \rightarrow \{0,1\}$ or, alternatively, into two functions $F^V: \{0,1\}^{2m+1} \rightarrow \{0,1\}$, $F^A: \{0,1\}^{2m+1} \rightarrow \{0,1\}$ which have y^V resp. y^A as the last argument, and to convert them to CNF form.

The main task is to find a concise and complete representation of F. By completeness we mean that all possible combinations at input and output are covered, so that the origin, propagation, and termination of undefined values can be calculated.

For this purpose, we adapted D-intersection [22]. Because we represent F as a set of terms in tabular form, TABLE II. includes also the '-' symbol. Notice that incompatibility cannot occur here.

TABLE II. SYMBOL INTERSECTION

	0	1	-	U
0	0	U	0	U
1	U	1	1	U
-	0	1	-	U
U	U	U	U	U

The complete algorithm for generation of a CNF for a given gate is shown in Figure 1. Let us assume that *gate* is a table describing the on-set of a completely specified Boolean function with one output, and that the columns of the table are labeled $x_1, \dots, x_{m-1}, y$. Furthermore, if t is a term, let $t[j]$ be the symbol of t in the column labelled j.

```
CNFlib(gate)
1   // generate the characteristic function
2   minimize gate                                   // by Espresso
3   add the off-set to gate                         // by Espresso
4   move the output column to input columns of gate
5   put all 1s into the output column of gate
6   // calculate intersections
7   do {
8       for each unordered pair (t1, t2) of terms from gate {
9           let s be the intersection of t1[y] and t2[y]
10          if s == U {
11              let t be the intersection of t1 and t2
12              if t is not in gate
13                  insert t into gate with output symbol 1
14          }
15      }
16  } while new terms are added to gate
17  // encode and convert to Boolean domain
18  encode gate using TABLE I. giving F
19  // produce CNF
20  turn F into off-set description                 // by Espresso
21  for each term t in F {
22      start a new clause
23      for each column label j {
24          if t[j] == 0 output j
25          if t[j] == 1 output ¬j
26      }
27  }
```

Figure 1. An algorithm generating the CNF characteristic function of a library gate for Tseitin's transformation with encoded undefined values

The algorithm has four main phases. The first one (lines 1 to 5) derive the characteristic function. Lines 6 to 16 are the main phase, which adds terms describing the behavior of undefined values to the function. Finally, the third phase (line 18) encodes the table and phase four (19-27) outputs the resulting CNF, using CNF and DNF duality.

The obtained CNF is then rewritten into a PBO instance in the straightforward way described in Subsection II.A.

The algorithm is as feasible as Espresso minimization [23] and off-set generation are.

For purposes of the test compression algorithm, a library of PBO instances for every supported gate is created using the procedure from Figure 1. Thus, the conversion is run only once.

C. The PBO-Compress Algorithm

The principles of the PBO-Compress algorithm are the same as those of SAT-Compress [8], except of the test patterns generation, where instead of generating and solving SAT for each fault, a PBO instance is generated and solved. For the complete algorithm see [8] and [20].

Test vectors having *maximum* of unspecified values are produced by the PBO solver. Specifying additional values generally decreases the chance for overlapping, but increases the number of covered faults. We have shown in [20] that this issue is crucial – when the local fault coverage is lost, the compression process is prolonged and the resulting bitstream is bigger too.

However, specifying a bit *needs not* always increase the fault coverage. This is the principle of the care bits injection procedure used in PBO-Compress. Maximum "meaningful" number of care bits are injected into a test pattern; injecting yet more care bits wouldn't increase fault coverage.

Similarly to [24] (where don't care bits are injected), all test pattern bits that are unassigned are tried for *care bit injection* by fault simulation. Both '0' and '1' values are tried and the case maximizing the number of detected faults is chosen. If no care value injected yields a fault coverage improvement, the unspecified value is retained.

III. Experimental Results

We have tested the algorithm on a subset of ISCAS [25], [26], IWLS [27] and ITC'99 [28] benchmarks. Only smaller circuits were chosen for a thorough testing of the algorithms properties. Since the results greatly resemble results from [20], we refer to this paper for results of bigger circuits.

Since all the algorithms are greedy, they are sensitive to many aspects, like the initial pattern choice, the order in which the faults are processed, the order of care-bits injection, and the structure of the source file. For this reason, a single measurement for one benchmark circuit and one algorithm configuration cannot bring reliable results [29], [30]. Therefore, except of the final experiments, we have conducted 1,000 measurements for each tested circuit and configuration, with the above-mentioned aspects set randomly.

In SAT-Compress, MiniSAT [11] was used as a SAT solver, in PBO-Compress MiniSAT+ [19] was used as PBO solver.

Recently we have published the SAT-Compress algorithm, where unspecified values were injected into completely specified test patterns, the Coverage Preserving Don't Care Injection (CPDCI) [20]. The approach presented in this paper treats the unspecified values in the opposite way; care bits are injected into maximally unspecified patterns.

The results for some selected benchmarks and the average values are shown in TABLE III. Average percentages of unassigned test bits obtained by the four techniques: SAT-Compress maximizing the numbers of don't cares regardless the fault coverage, SAT-Compress retaining the fault coverage (CPDCI), and PBO-Compress without and with the care bits injection are shown in the "*DCs*" columns. In the last case, two DC values are given: before ("*DCs*") and after ("*CBI*") care bits injection. The average bitstream lengths are provided in "*Bits*" columns.

Even though very few care bits are typically injected (1.5% on average), they greatly increase the number of covered faults. This is shown in the "*FD"* column. The values indicate the percentage of total faults covered due to the injected care bits.

We can see that approaches blindly maximizing the numbers of unspecified values are inferior to approaches retaining the fault coverage, in sense of final bitstream lengths. It can be observed that results of similar quality are obtained by SAT-Compress with unspecified values injection and PBO-Compress with care bits injection. This just confirms the theory that the compression algorithm behaves indifferently of the way of producing of unspecified values in the test cubes. Even though a real maximum of unspecified values is produced by PBO, the fault coverage aspect prevails. Still, PBO-Compress is capable of producing more don't cares which can be possibly used in some further processing.

IV. Conclusions

We have presented a test generation and compression method based on Pseudo-Boolean Optimization. A novel and original method to convert a conceptual hardware (miter) to a PBO instance was proposed. Similarly to SAT-based ATPGs, the PBO instance implicitly represents test vectors for a given fault. Additionally, the PBO-based approach offers a possibility of introducing an optimization criterion. In our case, it is the amount of unspecified values in the solution, which is maximized.

The experimental results show that the PBO-based algorithm unfortunately brings no qualitative breakthrough, compared to the previously published CPDCI technique. Even the scalability suffers here, since the PBO solving is much more time-expensive that SAT solving. However, it necessarily fits into an exploration of possibilities of generating and efficiently exploiting test don't cares and also more unspecified values are obtained, which could help in further refining the compression algorithm.

ACKNOWLEDGEMENT

This research has been supported by the grant of the Czech Technical University in Prague, SGS14/105/OHK3/1T/18.

REFERENCES

[1] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," in IEEE Trans. Comput. Vol. 32, No. 12, December 1983, pp. 1137-1144.

[2] H.K. Lee and D.S. Ha, "Atalanta: an Efficient ATPG for Combinational Circuits," Techn. Report, 93-12, Dep't of Electrical Eng., Virginia Polytechnic Institute and State University, 1999.

[3] J. Rajski, "Embedded Deterministic Test," in IEEE Trans. on CAD, vol. 23, No. 5, 2004, pp. 776-792.

[4] J. Jeníček and O. Novák, "COMPAS Advanced test compressor," in Proc. of IEEE East-West Design and Test Symposium 2010, pp. 543-548.

[5] T. Larrabee, "Test pattern generation using boolean satisfiability," in IEEE Trans. on Computer-Aided Design, vol. 11, 1992, pp. 4-15.

[6] R. Drechsler, S. Eggersglüß, G. Fey, and D. Tille, "Test Pattern Generation using Boolean Proof Engines". Springer Netherlands, 2009, XII, p. 192.

[7] R. Dobai, M. Baláž, "SAT-based generation of compressed skewed-load tests for transition delay faults", Microprocessors and Microsystems (MICPRO), Vol. 37, Issue 2, March 2013, pp. 196-205.

[8] J. Balcárek, P. Fišer, and J. Schmidt, „Techniques for SAT-based Constrained Test Pattern Generation," in Microprocessors and Microsystems, Vol. 37, Issue 2, March 2013, Elsevier, pp. 185-195.

[9] R. Dorsch and H.-J. Wunderlich, "Reusing Scan Chains for Test Pattern Decompression," in Journal of Electronic Testing: Theory and Applications (JETTA), Vol. 18, Issue 2, April 2002, pp. 231–240.

[10] E.J. Marinissen, et al., "On IEEE P1500's Standard for Embedded Core Test," in Journal of Electronic Testing, August 2002, Vol.18, Issue 4-5, pp. 365-383.

[11] N. Éen, N. Sorensson, "An extensible SAT-solver," in Lecture Notes in Computer Science 2919 - Theory and Applications of Satisability Testing, Springer Verlag, 2004 pp. 333-336.

[12] N. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an effcient SAT solver," in Proc. of 39th Design Automation Conference (DAC 2001), pp. 530-535.

[13] R. Ben-Eliyahu, R. Dechter, "On Computing Minimal Models," Annals of Mathematics and Artif. Intelligence, vol. 18, 1993, pp. 2-8.

[14] C. Pizzuti, "Computing Prime Implicants by Integer Programming," in 8th International Conference on Tools with Artificial Intelligence, 1996, pp. 332.

[15] V. Manquinho et al. "Prime implicant computation using satisfiability algorithms," in 9th International Conference on Tools with Artificial Intelligence, Newport Beach, CA, 1997, pp. 232-239.

[16] K. Ravi and F. Somenzi, "Minimal assignments for bounded model checking," in TACAS'04, Barcelona, Spain, 2004, pp. 31-45.

[17] I. Dillig, T. Dillig, K.L. McMillan, and A. Aiken, "Minimum satisfying assignments for SMT," in Proceedings of the 24th international conference on Computer Aided Verification, 2012, pp. 394-409.

[18] E. Boros and P. L. Hammer, "Pseudo-Boolean optimization," in Discrete Applied Mathematics, Vol. 123, Issues 1–3, 15 Nov. 2002, pp. 155-225.

[19] N. Éen, N. Sorensson, "Translating Pseudo-Boolean Constraints into SAT," in Journal on Satisfiability, Boolean Modeling and Computation, vol. 2 2006, pp. 1-25.

[20] J. Balcárek, P. Fišer, and J. Schmidt, "Simulation and SAT Based ATPG for Compressed Test Generation," in Proc. of 16th Euromicro Conference on Digital Systems Design, Sep. 4-6, 2013, pp. 445-452.

[21] G.S. Tseitin, "On the complexity of derivation in propositional calculus," in Studies in Constructive Mathematics and Mathematical Logic, Steklov Mathematical Institute, 1968, pp. 115–125.

[22] J.P. Roth, "Diagnosis of automata failures: A calculus and a method," IBM J. Res. Develop., vol. 10,1966, p. 278.

[23] R.K. Brayton et al., *Logic Minimization Algorithms for VLSI Synthesis*, Boston, MA, Kluwer Academic Publishers, 1984, 192 p.

[24] S. Kajihara and K. Miyase, "On Identifying Don't Care Inputs of Test Patterns for Combinational Circuits," Proc. Int'l Conf. on Computer Aided Design, Nov. 2001, pp. 364-369.

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

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

[27] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide," Technical Report 1991-IWLS-UG-Saeyang, MCNC.

[28] F. Corno, M.S. Reorda, G. Squillero, "RT-level ITC'99 benchmarks and first ATPG results," in: Proc. of the IEEE Design and Test of Computers (2000), pp. 44-53.

[29] A. Puggelli, T. Welp, A. Kuehlmann, and A. Sangiovanni-Vincentelli, "Are Logic Synthesis Tools Robust?," in Proc. of the 48th Design Automation Conference (DAC), 5-9 June 2011, pp. 633-638.

[30] P. Fišer, J. Schmidt, and J. Balcárek, "Sources of Bias in EDA Tools and Its Influence," in Proc. of 17th IEEE Symposium on Design and Diagnostics of Electronic Systems (DDECS), Warsaw (Poland), April 23-25, 2014, pp. 258-261.

TABLE III. EXPERIMENTAL RESULTS

Circuit	*SAT-Compress, max. DCs*		*SAT-Compress, CPDCI*		*PBO-Compress*		*PBO-Compress + CB injection*			
	Bits	*DCs*[%]	*Bits*	*DCs* [%]	*Bits*	*DCs* [%]	*Bits*	*DCs* [%]	*CBI* [%]	*FD* [%]
5xp1	155.58	8.56	102.32	0.52	106.82	1.82	96.77	1.43	0.43	8.09
b03_C	139.29	12.7	113.12	2.31	111.71	5.80	115.5	4.91	4.13	10.27
b9	271.23	9.11	215.10	4.14	217.76	9.94	228.72	8.61	7.53	20.32
c1355	292.18	1.80	265.57	0.29	264.09	0.79	265.32	0.70	0.59	5.27
c432	245.26	9.50	185.93	5.11	193.58	13.70	178.39	10.20	9.02	18.63
c499	222.66	2.18	198.28	0.29	200.74	0.90	204.46	0.74	0.52	4.22
c8	363.39	24.05	276.09	15.00	281.74	36.32	295.27	33.10	30.74	23.98
dc2	133.97	14.44	91.24	8.70	98.20	16.65	91.43	16.51	13.17	13.27
f51m	273.02	12.44	163.93	3.35	167.55	5.93	142.79	6.64	3.26	14.75
cht	156.09	6.76	133.98	4.94	138.03	6.77	126.61	6.57	5.98	6.26
i1	189.25	8.74	152.75	6.10	164.25	21.36	161.43	19.79	17.78	26.68
i3	392.22	8.55	349.81	4.83	361.34	13.15	391.51	10.65	9.99	64.31
Avg.	221.07	11.51	175.65	6.37	180.25	13.29	179.72	12.17	10.70	17.33