A SAT-Based Encoding for Finding a Minimal Automated Test Pattern Generation Test-Suite

Alexander Feldman[1] and **Mikoláš Janota**[2] and **Alexandre Perez**[1] and **Johan de Kleer**[1]
[1]PARC Inc.
Palo Alto, CA 94304, USA
e-mail: afeldman,aperez,dekleer@parc.com
[2]Czech Technical University in Prague
Prague, Czech Republic
e-mail: mikolas.janota@gmail.org

Abstract

Existing methods for Automated Test Pattern Generation (ATPG) for digital Integrated Circuits (ICs) typically have two phases: (1) computation of test-vectors for all faults and (2) compaction of the test-vectors into a test-suite. The compaction is needed, because, the main cost of testing is the time necessary to apply all test-vectors one after another. This paper proposes a novel encoding that can calculate the whole ATPG test-suite in one go, thus solving together both (1) and (2). To the best of our knowledge, our method and encoding are the first to provide theoretical lower bounds on the size of the test-suite. We experiment extensively on the ISCAS-85 benchmark. Our method solves circuits of non-trivial size. For example, we find a test-suite of only 13 vectors that tests all 942 stuck-at-0 and stuck-at-1 possible faults for the c880 arithmetic-logic unit and control circuit. Similarly, all 2000 stuck-at-1 faults in the c6288 16×16-bit multiplier can be covered by only 3 vectors. Our method is capable of generating proofs showing that the generated test-suites are of minimal size.

1 Introduction

Automated Test Pattern Generation (ATPG) for digital circuit is a well studied subject with an extensive application to semiconductor manufacturing. More than 20 years ago, the cost of testing of complex ICs was approaching 40-50% of the total manufacturing cost [5, p. 42]. Semiconductor testing costs are increasing as due to shrinking feature size [18] defects are more likely and there are more transistors.

The end goal of the ATPG process is to generate a full suite of test-vectors. A test-suite should cover every possible single-fault in the circuit model of the IC. Typically, all stuck-at-0 and stuck-at-1 faults are tested because there is statistical evidence that if there are some impurities in the IC substrate or dust or other defects, they will manifest themselves as stuck-at-0 or stuck-at-1 faults. The biggest cost of IC testing is not generating the test-suite but applying the test-suite to the newly made IC at test-time. That is because each test vector is applied sequentially. As a result the most important objective function in ATPG is to minimize the number of test-vectors.

Existing approaches to practical ATPG first use random inputs to cover for about 90% of all possible faults in an IC [2]. The rest of the faults are considered "difficult" from the viewpoint of combinatorics and they are generated with SAT-based methods [3]. A difficult fault can be, for example, on the carry path in a Carry-Lookahead Adder (CLA). All regular input lines, in this case, must be fed ones, without which it is impossible to test for stuck-at-0. Finally, once all possible faults are covered by test-vectors, the test-suite is **compacted**.

A shortcoming of most existing approaches is that they solve separately the test-vector generation and the test-suite compaction. Furthermore, random test-vectors are not very good candidates for compaction. Consider, for example, a 16-bit multiplier. Significant percentage of the stuck-at-1 faults can be tested with a single test consisting of mostly zeroes. A random input, on the other hand, would have approximately half of its input bits set to zero, hence it will not compact easily with other test-vectors.

The reason of using random test-inputs is that ATPG was developed [17] long before SAT solvers became powerful [1]. Nowadays, when SAT solvers can solve formulas with thousands of variables, it is possible to combine both ATPG generation and ATPG compaction into a single optimization problem. This is what we achieve in this paper: we provide a method that can generate a whole test-suite with one call to a SAT solver.

2 Preliminaries

Digital ICs without flip-flops are modeled as circuits, and that is our input representation. Definition 1 is directly adopted from Vollmer [20].

Definition 1 (Boolean Circuit)**.** *Given a basis B, a Boolean circuit $\mathcal{G}$ over B is defined as $\mathcal{G} = \langle V, E, \alpha, \beta, \chi, \omega \rangle$, where $\langle V, E \rangle$ is a finite directed acyclic graph, $\alpha : E \to \mathbb{N}$ is an injective function, $\beta : V \to B \cup \{\star\}$, $\chi : V \to \{x_1, x_2, \ldots, x_n\} \cup \{\star\}$, and $\omega : V \to \{y_1, y_2, \ldots, y_m\} \cup \{\star\}$. The following conditions must hold:*

1. *If $v \in V$ has an in-degree 0, then $\chi(v) \in \{x_1, x_2, \ldots, x_n\}$ or $\beta(v)$ is a 0-ary Boolean function (i.e., a Boolean constant) in B;*
2. *If $v \in V$ has an in-degree $k > 0$, then $\beta(v)$ is a k-ary Boolean function from B;*
3. *For every $i, 1 \leq i \leq n$, there is exactly one node $v \in V$ such that $\chi(v) = x_i$;*
4. *For every $i, 1 \leq i \leq m$, there is exactly one node $v \in V$ such that $\omega(v) = y_i$.*

In this paper we use the standard basis: multi-input AND, OR, NAND, and NOR gates; XOR and XNOR gates; buffers an inverters. The encoding we present, however, works with any other basis.

The function α determines the ordering of the edges that go into a Boolean function when the ordering matters (such as in implication). The function α is not necessary if B consists of symmetric functions only.

The function β determines the type of each node in the circuit: a function in the basis B. The function χ specifies the set of input nodes $\{x_1, x_2, \ldots, x_n\}$. The function ω specifies the set of output nodes $\{y_1, y_2, \ldots, y_n\}$. A node v is non-output, or computational, if $\chi(v) = \star$ and $\omega(v) = \star$.

Figure 1 shows a simple and frequently used circuit that is used for adding the two binary numbers i_1 and i_2 and a carry input bit c_i. The output is found in the sum bit Σ and in the carry output c_o. The full-adder consists of two identical half-adders. Both half-adders are made of an AND-gate and an XOR-gate. The AND-gates are marked as a_1 and a_2, respectively. The XOR-gates are denoted as x_1 and x_2, respectively. Finally the two most-significant bits from the half-adders (z_1 and z_3) are combined by the OR-gate o into the carry output c_o.

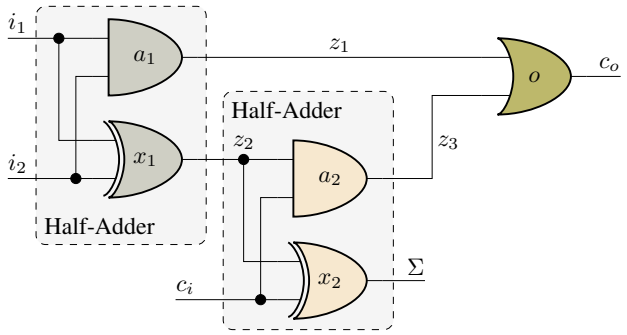

Figure 1: A full adder

We use the standard ATPG definitions [5] of fault-location, fault, stuck-at-0, and stuck-at-1. We convert circuits to Boolean formulas and, ultimately, to Conjunctive Normal Forms (CNFs). This is achieved by adding a Tseitin variable at the output of each gate. In an abuse of notation we sometime use interchangeably the terms circuit and Boolean formula.

Definition 2 (Test-Vector)**.** *Given a circuit $\mathcal{G}$ with primary inputs X, fault location set F, and a fault location $f \in F$ a test-vector that tests f is an assignment $X = \chi$, such that $\mathcal{G}(X = \chi, f = 0) \neq \mathcal{G}(X = \chi, f = 1)$.*

Notice that a test-vector for an assumable f may not exist. This does not change any of the algorithms we propose and for the rest of the paper we assume that there always exists at least one test vector for each assumable input.

Definition 3 (Test-Suite)**.** *Given a circuit $\mathcal{G}$ with a set of fault locations F, a test-suite Λ is a set of test-vectors $\{\chi_1, \chi_2, \ldots, \chi_n\}$ such that for any $f \in F$ there is at-least one test-vector $\chi \in \Lambda$ that tests f.*

3 Encoding

Throughout this section we assume a fixed circuit $\mathcal{G}$, which is being tested. We develop an encoding of the compaction problem for $\mathcal{G}$ and some fixed number of testing vectors k.

The top-down point of view of the encoding is as follows. First encode the problem as a single circuit $\mathcal{T}$, whose inputs represent the unknown testing vectors. This means that the circuit $\mathcal{T}$ has $k \times m$ inputs if the $\mathcal{G}$ has m inputs and it evaluates to true if and only if it is fed k testing vectors sensitizing all possible single-stuck-at-faults of $\mathcal{G}$.

Once the circuit $\mathcal{T}$ is constructed, decide its satisfiability by a single call to a SAT solver through standard means, which involves introducing fresh (Tseitin) variables in order to avoid exponential blowup when producing CNF input [15, 19]. If $\mathcal{T}$ is satisfiable, we have solved the original problem—the satisfying inputs of $\mathcal{T}$ form the desired collection of testing vectors. If the circuit $\mathcal{T}$ is unsatisfiable, testing of $\mathcal{G}$ cannot be compacted into k vectors and a larger value of k has to be considered.

We remark that it would be possible to directly encode the problem into SAT but constructing a circuit first enables us to avail of common structures and easily apply standards simplifications, such as constant propagation.
We proceed by developing the different parts of the encoding.

3.1 Single Fault

The first ingredient of the construction is the calculation of a testing vector for a single fault. This means that we are looking for a vector of input values under which the faulty circuit differs from the original.

Conceptually, this means constructing two copies of the circuit, the original circuit and the faulty one, and asserting a disequality between them. Effectively, this is an equivalence check between the original and the faulty circuit.

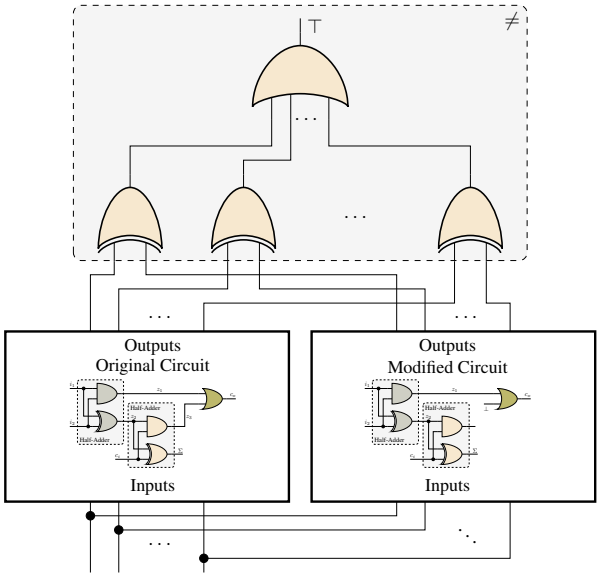

Figure 2: A classical single-fault miter

Figure 3 shows a single-fault miter based on the full-adder running example from Sec. 2. There are two copies of the full-adder: the original circuit made of gates a_1, a_2, x_1, x_2, and o and the circuit with the fault that is made of a_1', a_2', x_1', x_2', and o'. The fault that is being tested is on the upper-input of x_2, which is on a fanout branch. The three PIs i_1, i_2, and c_i are pair-wise joined together and the two pairs of outputs Σ, Σ', c_o, and c_o' are compared with the help of a comparator. The latter consists of the two XOR-gates

c_1 and c_2 and the OR-gate c_3. One can see that an input assignment to the PIs such that the output of c_3 is $\top$ is a test-vector that differentiates the fault of interest.

Let $\mathcal{G}_{C=v}$ denote the circuit $\mathcal{G}$ where a component C is substituted by a fixed value $v \in \{0, 1\}$. Let $\mathcal{D}_{C=v}(t)$ represent the circuit that evaluates to true if and only if the circuits $\mathcal{G}$ and $\mathcal{G}_{C=v}$ differ, given the inputs t. The circuit $\mathcal{D}_{C=v}(t)$ is constructed as follows (see also 2).

1. Construct $\mathcal{G}$ connecting it to the inputs t.
2. Construct $\mathcal{G}_{C=v}$ connecting it to the inputs t, while reusing common components constructed in step 1.
3. Connect the outputs of the circuits constructed in step 1 and step 2 into a miter.

The faulty and the original circuits act on the same set of primary inputs and the faulty one is obtained by altering a single component. This enables us to reuse the common sub-circuits. In our implementation, in the faulty circuit $\mathcal{G}_{C=v}$ we only construct the part of the circuit that is affected by the replacement of C by the value v; the rest is copied from the original circuit $\mathcal{G}$. Constant propagation is used to propagate the value v in the faulty circuit.

The idea of component reuse is illustrated in Fig. 5 which is a continuation of the full-adder running-example. The fact that the fault only affects x_2 allows the elimination of the rest of the gates in the faulty circuit. As a result only one PO (c_o) has to be compared. This allows the good circuit to also lose all gates that do not drive the other PO (Σ). This is a substantial saving of 7 gates. The actual savings depends on how close the fault is to a primary fault and on the topology of the circuit.

3.2 Multiple Faults, Single Test Vector

The next step in the construction is to test for all the faults on a single input vector. Let $\mathcal{F}$ be a set of all considered fault pairs (C, v). We do not know upfront which faults will be tested by a given input vector and therefore we construct $|\mathcal{F}|$ disequality tests while initially not imposing any restrictions on them.

The crucial observation here is that the original circuit does not have to be constructed anew for each disequality. This allows for further sub-circuit reuse by constructing $|\mathcal{F}|$ miters in parallel, while reusing the original circuit $\mathcal{G}$; see 4.

Conceptually, this part of the construction produces a sequence of circuits $\mathcal{D}_{C=v}(t)$ for each $(C, v) \in \mathcal{F}$, for a single unknown vector of inputs t. The next step is to consider all k unknown input vectors.

3.3 Multiple Faults, Multiple Test Vectors

Finally, by putting it all together, we test for all the faults across all the unknown test vectors. For each test vector t_i and a fault $f \in \mathcal{F}$ consider the circuit $\mathcal{D}_f(t_i)$. This results in a matrix of circuits, where each row covers all the possible faults across the same unknown test vector, i.e., a single column tells us which test vectors are testing the given fault; see 6. We call this matrix the *testing matrix*.

Having constructed this matrix, enables us to express the final condition of the problem, i.e., that each fault is covered by at least one test vector. Semantically this means that at least one circuit $\mathcal{D}_f(t_i)$ must be set to true in any given column.

Observe that there is no sharing of sub-circuits between different rows of the matrix because these act on different sets of inputs However, each row shares the parts of the original non-faulty circuit.

3.4 Symmetries

The presented encoding exhibits several symmetries. First, we observe that the constructed testing matrix of circuits (6) translates to a matrix of 0's and 1's once it is given the right set of testing vectors. Hence, structurally it is similar to graph incidence matrix, for which symmetries were heavily studied [12].

A natural symmetry breaker is to impose lexicographic order on the contents of the test vectors, or, on the contents of the testing matrix. However, it is incorrect to break both symmetries at the same time. We show that focusing on the contents of the testing matrix is advantageous since it naturally strengthens the encoding. This is done as follows.

Assume that we are already given the contents of the testing input vectors, i.e., the testing matrix is filled in with 0/1 so that each column has at least one occurrence of 1. Since there must be at least one testing vector testing the first fault, we can reorder the rows of the matrix so that this vector is on the first line. In another words, the matrix has 1 in the top-left corner.

This idea may be generalized to other faults (columns). The second fault has already been tested by the first vector, or the vector that it is testing it can be swapped into the second position. In other words, we add the symmetry breaker that the second column must have at least one occurrence of 1 in the first two rows. More generally, in the i-th column there must be at least one occurrence of 1 in the first i rows; see 7. This reasoning affects the first $k \times k$ rectangle of the matrix—the restriction in the k-th column results in the original because there only k rows in the matrix.

Since these restrictions are eventually encoded into CNF, this symmetry effectively *shortens* the corresponding clauses. Indeed, the constrained that at least one test vector tests any given fault f translates to a disjunction of the literals representing the respective encoding of the circuits $\mathcal{D}_f(t_i), i \in 1, 2, \ldots, k$. Due to the symmetry breaking, this clause is shortened to the length j in the j-th column.

4 Experimental Evaluation

Table 1 shows the 14 combinational circuits from the 74xxx and ISCAS-85 benchmarks [4, 11].

Notice that there are more faults than components. In ATPG for digital ICs, faults are placed both on input and output wires of gates, as this delivers better statistical coverage of physical faults [5]. Table 2 gives the number of collapsed faults [5] for each circuit.

Before experimenting with the proposed encoding we had to run single-fault ATPG as many circuits presented a small number of untestable faults. This was done with the single-fault miter shown in Figure 2 and it did not give difficulty to the SAT solver.

The results in Table 3 show exact solutions and lower and upper bounds for the size of the test-suite. The CPU time spent in SAT-solving has been limited to 24 h. The UNSAT calls become more difficult as the number of test-vectors approaches the one necessary for having a SAT solution.

To generate the upper bounds shown in Table 3, we have started the search from a guess for the number of test-vector, $k = 30$. After that, we have checked for decreasing values

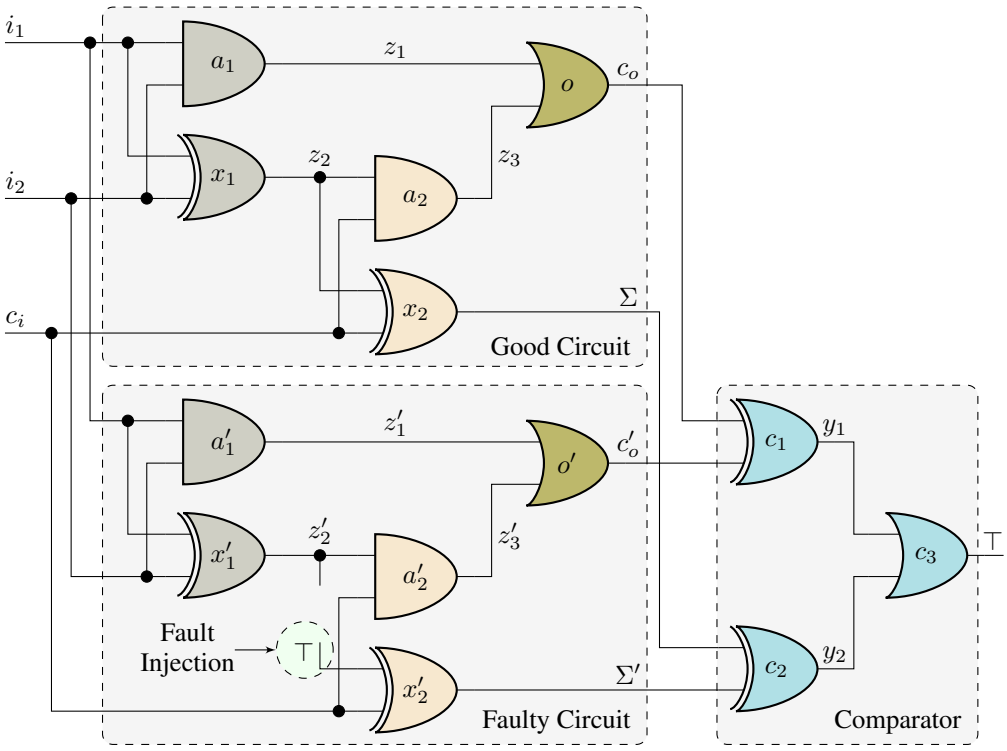

Figure 3: A single-fault miter from the fault adder

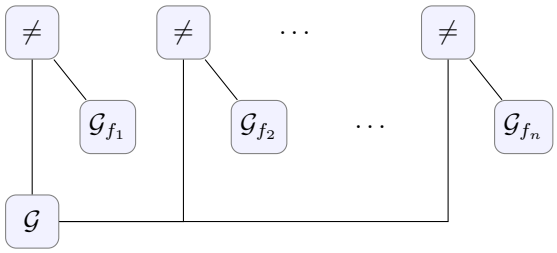

Figure 4: A parallel miter that tests n single faults

Name	Description	PIs	POs	Gates
74182	4-bit CLA	9	5	19
74L85	4-bit comparator	11	3	33
74283	4-bit adder	9	5	36
74181	4-bit ALU	14	8	65
c432	27-channel interrupt controller	36	7	160
c499	32-bit SEC circuit	41	32	202
c880	8-bit ALU	60	26	383
c1355	32-bit SEC circuit	41	32	546
c1908	16-bit SEC/DEC	33	25	880
c2670	12-bit ALU	233	140	1193
c3540	8-bit ALU	50	22	1669
c5315	9-bit ALU	178	123	2307
c6288	32-bit multiplier	32	32	2416
c7552	32-bit adder	207	108	3512

Table 1: 74xxx and ISCAS-85 digital circuits

of k, until the SAT solver returned UNSAT or until the $24\,\mathrm{h}$ timeout has been reached.

The most important factor for the SAT performance is the number of faults. It is visible in Table 1 that most circuits have unequal number of S-A-0 and S-A-1 faults. This is due the equivalent fault collapsing and if the circuits are made predominantly of OR-gates or AND-gates. Table 3 shows that test-vectors that test for S-A-0 faults do not cover many S-A-1 faults and vice versa (the bounds for all S-A-0 + S-A-1 faults are close to the sum of S-A-0 and S-A-1).

It is visible that our methods "prefers" circuits of small depth with multiple PIs and POs.

5 Related Work

SAT-based encodings have been used to generate test sets covering multiple faults – such as [9,10], in which the ATPG problem is formulated as a quantified boolean formula and solved using incremental SAT. Our encoding is distinct in that it does enforce minimality of generated test sets.

Several approaches to generate reduced test sets have been proposed. *Static compaction* methods are typically applied as a post-processing step to reduce the size of test sets obtained with ATPG algorithms [6, 8, 14]. Since these static compaction approaches reduce the size of a given test set by eliminating redundant test patterns, decoupled from the process of generating the test set, minimality is not guaranteed.

Dynamic compaction methods, on the other hand, are integrated into the ATPG process. They use different sets of heuristics to increase fault detection coverage while the test set is being constructed. In [16], *close-to-minimal* test sets are generated by first grouping sets of faults based on shared *necessary assignments* and then running the ATPG procedure on each set of faults. In [7], the authors propose a similar SAT-based approach to ours – insofar as to leverage the fact that the correct circuit can be shared for multiple faults

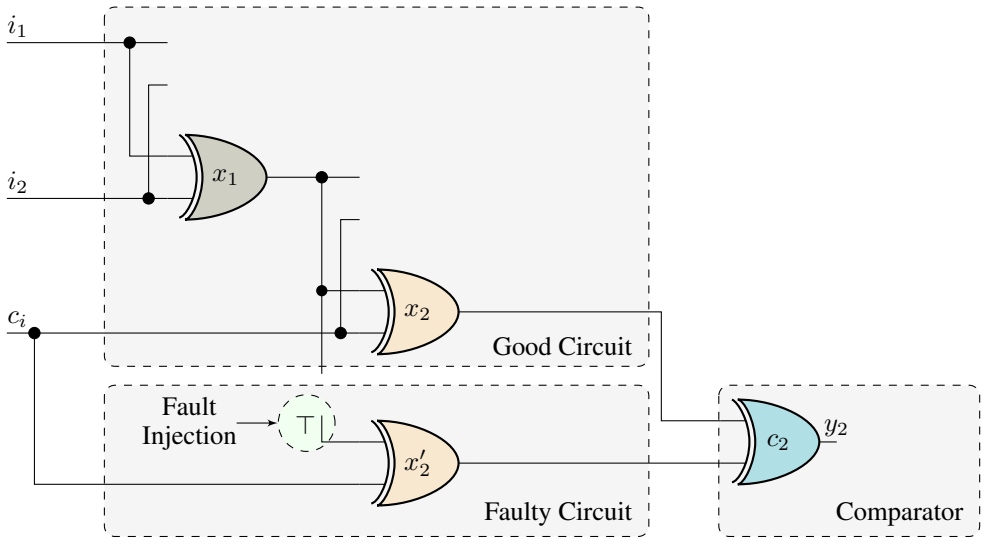

Figure 5: A compacted single-fault miter from the fault adder

$$\begin{array}{ccc} \mathcal{D}_{f_1}(t_1) & \ldots & \mathcal{D}_{f_n}(t_1) \\ \ldots & \ldots & \ldots \\ \mathcal{D}_{f_1}(t_k) & \ldots & \mathcal{D}_{f_n}(t_k) \\ \hline \sum \geq 1 & \ldots & \sum \geq 1 \end{array}$$

Figure 6: The encoding for generating an ATPG test-suite is a matrix of circuits.

$$\begin{array}{ccccc} \mathcal{D}_{f_1}(t_1) & \mathcal{D}_{f_2}(t_1) & \mathcal{D}_{f_2}(t_1) & \ldots & \mathcal{D}_{f_n}(t_1) \\ \hline \sum \geq 1 & \mathcal{D}_{f_2}(t_1) & \mathcal{D}_{f_2}(t_1) & \ldots & \mathcal{D}_{f_n}(t_1) \\ & \hline \sum \geq 1 & \mathcal{D}_{f_2}(t_1) & \ldots & \mathcal{D}_{f_n}(t_1) \\ & & \hline \sum \geq 1 & \ldots & \end{array}$$

Figure 7: Symmetry breaking in the encoding.

– to be run after pruning the portion of faults that have good random testability.

SAT has been used to decide equivalents of circuits with a number of ingenious techniques [13]. In our encoding, effectively, we are looking for counterexamples to equivalents between the faulty and the original circuit.

6 Conclusions

This paper proposes a method for computing minimal size test-suites for digital ICs. This is achieved by combining the two optimization problems: fault-test generation and test-suite compaction into a single optimization problem. Early empirical evaluation on ISCAS-85 circuits shows that problems of non-trivial size can be solved with modern SAT-solvers.

We plan to extend our experiments to ISCAS-89 and to invent encoding elements that can improve the performance of the SAT-solving. Such elements are tuples of faults that cannot be tested simultaneously and various extra constraints enforcing lower bounds.

Name	Unused PIs	S-A-0	S-A-1	S-A-0 + S-A-1
74182	0	31	52	83
74L85	0	41	82	123
74283	0	52	76	128
74181	0	85	142	227
c432	0	170	354	524
c499	0	307	451	758
c880	0	331	611	942
c1355	0	299	1275	1574
c1908	0	412	1467	1879
c2670	76	748	1847	2595
c3540	0	1102	2326	3428
c5315	0	1617	3733	5350
c6288	0	5744	2000	7744
c7552	1	2163	5385	7548

Table 2: Number of faults in the ISCAS-85 circuits

References

[1] Tomáš Balyo, Nils Froleyks, Marijn JH Heule, Markus Iser, Matti Järvisalo, and Martin Suda. Proceedings of SAT competition 2020: Solver and benchmark descriptions. 2020.

[2] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann. OPMISR: The foundation for compressed ATPG vectors. In *Proceedings International Test Conference 2001*, pages 748–757, 2001.

[3] Armin Biere and Wolfgang Kunz. Sat and atpg: Boolean engines for formal hardware verification. In *Proceedings of the 2002 IEEE/ACM International Conference on Computer-Aided Design*, ICCAD '02, page 782–785, New York, NY, USA, 2002. Association for Computing Machinery.

[4] Franc Brglez and Hideo Fujiwara. A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran. In *Proc. ISCAS'85*, pages 695–698, 1985.

Name	S-A-0		S-A-1		S-A-0 + S-A-1	
	LB	UB	LB	UB	LB	UB
74182	5	5	5	5	10	10
74L85	9	9	> 17	≤ 23	> 18	≤ 23
74283	6	6	6	6	10	10
74181	7	7	8	8	12	12
c432	11	11	> 14	≤ 27	> 15	≤ 27
c499	> 16	-	> 15	-	> 16	-
c880	9	9	12	12	13	13
c1355	> 16	-	> 19	-	> 16	-
c1908	> 16	≤ 20	> 14	-	> 13	-
c2670	15	15	> 15	-	> 16	-
c3540	> 14	-	> 16	-	> 14	-
c5315	> 15	≤ 17	> 14	≤ 30	> 15	-
c6288	> 5	≤ 30	3	≤ 8	> 5	-
c7552	> 15	≤ 22	> 15	-	> 14	-

Table 3: Exact solutions, Lower Bounds (LBs), and Upper Bounds (UBs) for the number of test-vectors in a test-suite for the ISCAS-85 circuits

[5] Michael L. Bushnell and Vishwani D. Agrawal. *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Kluwer Academic Publishers, Boston, 2000.

[6] Fulvio Corno, Paolo Prinetto, Maurizio Rebaudengo, and Matteo Sonza Reorda. New static compaction techniques of test sequences for sequential circuits. In *European Design and Test Conference, ED&TC '97, Paris, France, 17-20 March 1997*, pages 37–43. IEEE Computer Society, 1997.

[7] Stephan Eggersglüß, Rene Krenz-Baath, Andreas Glowatz, Friedrich Hapke, and Rolf Drechsler. A new sat-based ATPG for generating highly compacted test sets. In Jaan Raik, Viera Stopjaková, Heinrich Theodor Vierhaus, Witold A. Pleskacz, Raimund Ubar, Helena Kruus, and Maksim Jenihhin, editors, *IEEE 15th International Symposium on Design and Diagnostics of Electronic Circuits & Systems, DDECS 2012, Tallinn, Estonia, April 18-20, 2012*, pages 230–235. IEEE, 2012.

[8] Paulo F. Flores, Horácio C. Neto, and João P. Marques Silva. On applying set covering models to test set compaction. In *9th Great Lakes Symposium on VLSI (GLS-VLSI '99), 4-6 March 1999, Ann Arbor, MI, USA*, pages 8–11. IEEE Computer Society, 1999.

[9] Masahiro Fujita and Alan Mishchenko. Efficient sat-based ATPG techniques for all multiple stuck-at faults. In *2014 International Test Conference, ITC 2014, Seattle, WA, USA, October 20-23, 2014*, pages 1–10. IEEE Computer Society, 2014.

[10] Masahiro Fujita, Naoki Taguchi, Kentaro Iwata, and Alan Mishchenko. Incremental ATPG methods for multiple faults under multiple fault models. In *Sixteenth International Symposium on Quality Electronic Design, ISQED 2015, Santa Clara, CA, USA, March 2-4, 2015*, pages 177–180. IEEE, 2015.

[11] Mark Hansen, Hakan Yalcin, and John Hayes. Unveiling the ISCAS-85 benchmarks: A case study in reverse engineering. *IEEE Design & Test*, 16(3):72–80, 1999.

[12] Marijn J. H. Heule. Optimal symmetry breaking for graph problems. *Mathematics in Computer Science*, 13(4):533–548, 2019.

[13] Andreas Kuehlmann, Viresh Paruthi, Florian Krohm, and Malay K. Ganai. Robust boolean reasoning for equivalence checking and functional property verification. *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.*, 21(12):1377–1394, 2002.

[14] Xijiang Lin, Janusz Rajski, Irith Pomeranz, and Sudhakar M. Reddy. On static test compaction and test pattern ordering for scan designs. In *Proceedings IEEE International Test Conference 2001, Baltimore, MD, USA, 30 October - 1 November 2001*, pages 1088–1097. IEEE Computer Society, 2001.

[15] David A. Plaisted and Steven Greenbaum. A structure-preserving clause form translation. *J. Symb. Comput.*, 2(3):293–304, 1986.

[16] Santiago Remersaro, Janusz Rajski, Sudhakar M. Reddy, and Irith Pomeranz. A scalable method for the generation of small test sets. In Luca Benini, Giovanni De Micheli, Bashir M. Al-Hashimi, and Wolfgang Müller, editors, *Design, Automation and Test in Europe, DATE 2009, Nice, France, April 20-24, 2009*, pages 1136–1141. IEEE, 2009.

[17] J. Paul Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of R & D*, 10:278–291, 1966.

[18] Ed Sperling. Test costs spiking. Semiconductor Engineering, 2020. `https://semiengineering.com/test-costs-spiking/`.

[19] G. S. Tseitin. On the complexity of derivations in the propositional calculus. *Studies in Constructive Mathematics and Mathematical Logic*, Part II, ed. A.O. Slisenko, 1968.

[20] Heribert Vollmer. *Introduction to Circuit Complexity: A Uniform Approach*. Springer Science & Business Media, 2013.