Improving Test Pattern Compactness in SAT-based ATPG

Stephan Eggersglüß Rolf Drechsler
Institute of Computer Science, University of Bremen
28359 Bremen, Germany
{segg,drechsle}@informatik.uni-bremen.de

Abstract

Automatic Test Pattern Generation (ATPG) is one of the core problems in testing of digital circuits. ATPG algorithms based on Boolean Satisfiability (SAT) turned out to be very powerful, due to recent advances in SAT-based proof engines. SAT-based ATPG clearly outperforms classical approaches especially for hard-to-detect faults. But due to the SAT provers, a major drawback of the resulting test patterns is that a large number of input bits is specified. Thus, the resulting patterns are not well suited for test compaction and compression.

In this paper we present techniques to increase the number of unspecified bits in test patterns generated by SAT-based ATPG tools. We make use of structural properties of the circuit and apply local don't cares. Experimental results on industrial designs show significant reductions of up to 97%.

1 Introduction

While ATPG was considered a solved problem some years ago, due to significantly increasing circuit sizes according to Moore's law, there is a renewed interest in algorithms for efficient test pattern generation. While classical approaches, like PODEM [9] or FAN [8] are based on backtrack search, alternative formulations based on SAT engines have been proposed. The first methods were presented in the early 90s [10, 18]. But at that time no efficient SAT solvers were available.

In the last ten years, there was a tremendous improvement of SAT solvers resulting in speed-ups of several orders of magnitude. This is mainly due to the instance management, efficient learning techniques, non-chronological backtracking and clever decision heuristics (see e.g. [11, 12, 3]). Based on these proof engines, very powerful ATPG tools have been proposed that clearly outperform classical approaches especially on hard-to-test and also on redundant faults. This has been shown in several studies and for varying fault models (see e.g. [19, 17, 4, 6, 5]).

But SAT-based techniques face a major problem, when ATPG is not considered as an isolated problem, but in the complete design flow. The step following ATPG is the compaction of test patterns. But here, due to the algorithmic structure of modern SAT solvers, the patterns are not well suited. For high compaction, the patterns should only specify bits that are needed for showing the faulty behaviour at an output. All other inputs should remain unspecified. While the early DPLL SAT approaches [2, 1] assigned only relevant variables and checked the set of clauses after each iteration, modern solvers assign variables until a contradiction occurs. And if no contradiction occurs, the instance is classified as *satisfiable*. This efficient handling of the instance is a major reason for the performance, but results in over-specified patterns.

A simple alternative would be to check for each input whether the assigned value is needed by running a fault simulation with an assigned don't care value. But this would be far too time consuming for large industrial designs and for this is not feasible. In [13, 16], techniques for eliminating irrelevant variables in counterexamples were presented. However, these approaches have much time overhead and for this are not well suited for SAT-based ATPG. Another possibility is the extension of a SAT solver as done in [15]. But this is not always desirable, because the robustness of modern SAT solvers is likely to be compromised this way.

In this paper we present an approach to SAT-based ATPG that results in very compact test patterns, i.e. patterns with a large number of unspecified bits. We discuss two strategies that make both use of structural information and local don't cares. While one approach describes an integration in the SAT instance, the other is based on a post-processing step. Properties are discussed and experimental results on large industrial designs are given. It is shown that the post-processing approach can reduce the number of specified bits by up to 97%, while the run time is negligible.

The paper is structured as follows: In the next section, a short overview of SAT-based ATPG is given. Different strategies to determine more compact test patterns are presented in Section 3. In Section 4, experimental results are provided and discussed. Conclusions are drawn in Section 5.

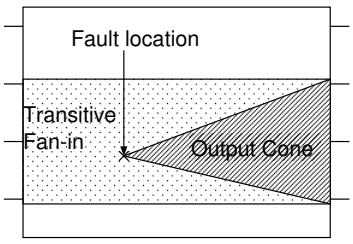

Figure 1. Extraction of the influenced Circuit Parts

2 SAT-based ATPG

In this section, the application of SAT-based ATPG is briefly reviewed. For more details see [10, 18, 17]. In this context, the *Stuck-At Fault Model* (SAFM) is considered[1]. Then, the general transformation of an ATPG instance into a SAT instance is presented. Furthermore, improving the search process of the SAT instance by including problem specific knowledge is briefly described.

The SAFM is a static fault model. One line in the circuit is considered to be stuck at the constant value 0 or 1 and is therefore independent from the primary inputs. The modeling of a *Stuck-At Fault* (SAF) in a given circuit is described in the following. A particular SAF is called *testable*, iff a test pattern exists which produces different output values in the faulty and the fault-free circuit at least at one output. Otherwise the fault is *untestable*.

Modern SAT solvers work on the problem represented as a *Conjunctive Normal Form* (CNF). A CNF is a set of clauses, a clause is a set of literals and a literal is a variable in its positive or negative form. A CNF is satisfied iff an assignment exists, which satisfies all clauses. A clause is satisfied iff at least one literal is satisfied.

Due to reasons of efficiency, modern SAT solver detect *satisfiability*, iff all variables are assigned and no contradiction occurs instead of checking whether all clauses are satisfied. As a result, there exists no unspecified variable in the computed solution.

The classical transformation of an ATPG instance into a SAT instance as described in [10] is explained in the following. To each line in the circuit, a Boolean variable is assigned. The functionality of each gate in the circuit is represented by a set of clauses. The complete CNF of the circuit is the conjunction of the clauses of all gates.

Consider the circuit in Figure 1. To model the SAF, all outputs at which the fault effect may be observed have to be determined. The transitive fan-in cone of these outputs influences the detection of the fault and for this must be added to the SAT instance.

To detect the fault, a faulty and a fault-free version of the circuit have to be modeled. Since different values can only occur in the output cone of the fault location, the transitive fan-in cone can be shared between both versions. Only the output cone has to be duplicated. This results in a reduction of the SAT instance.

To guide the search process and to guarentee that the fault effect can be observed at the outputs, structural information is added to the SAT instance as suggested in [18] and originally proposed for the D-algorithm [14].

Beside the variables for the faulty and the fault-free version, a third variable g_D is therefore introduced for each gate g in the output cone. This variable denotes whether the gate is on a D-chain[2] ($g_D = 1$) or not ($g_D = 0$).

Further details about the modeling of the necessary additional implications can be found in [17].

However, in industrial circuits it is insufficient to model only Boolean values. Due to page limitation, the modeling of multi-valued logic is not explained here. A detailed description can be found in [7].

In summary, the SAT instance is only satisfiable, iff a test pattern is found that yields at least one wrong output value if the fault is present. In every case, all included inputs are assigned with specified values.

3 Improving Compactness

SAT-based ATPG algorithms have been shown to be effective even for large industrial circuits. But a weakness of this method is the large number of specified bits in the computed test pattern.

During their search for a solution of the problem, state-of-the-art SAT solvers, like e.g. Zchaff [12] or MiniSat [3], prove either the unsatisfiability by showing that no solution for the given formula exists or the satisfiability by computing a Boolean assignment of the formula. In state-of-the-art SAT solvers, the stopping criterion of the latter case is the complete assignment of all variables.

More formally, a solution of a Boolean formula $f(x_1, ..., x_n)$ is found, iff

$$\forall x_i \mid 1 \leq i \leq n : \; x_i \in \{0, 1\}$$

and no contradiction exists.

From this solution the test pattern is directly determined by the assignment of the input variables. Due to the complete Boolean assignment, all bits of the test pattern of the considered part of the circuit have a specified value. That means they are either 0 or 1, but not X (*don't care*).

[1] The techniques in the following can also be applied to more complex fault models, like gate delay or path delay, if the ATPG engine is based on SAT.

[2] A D-chain denotes a path from the fault location to a primary output, where on each gate along the path the values of the faulty and fault-free versions differ.

```
 1 testpattern t = X;
 2 set<input> s;
 3 list l;
 4 foreach(output o) do
 5 {
 6    if (observable(o))
 7    {
 8       l.push(o);
 9       while (!l.empty())
10       {
11          gate g = l.first_element();
12          if (g == INPUT) s.add(g);
13          else l.add(g.all_predecessors())
14          l.remove(g);
15       }
16       foreach (input i in s) do
17       {
18          t.set_computed_bit(i);
19       }
20       break;
21    }
22 }
```

Figure 2. Pseudo-code of the Post-Processor

In contrast, classical ATPG algorithms such as FAN [8] assign X-values to signals during their search process and, as a result, immediately generate test patterns with a small number of specified bits.

In industrial practice, it is important, that computed test patterns have a high number of unspecified bits. This is required such that techniques like test compaction and test compression can be applied. In the following, strategies are presented that reduce the number of specified bits in test patterns computed by SAT-based ATPG.

In this section, two post-processing strategies that result in more compact test patterns are introduced. In Section 3.1, the exploitation of structural properties about the observability of the fault effect is presented, while Section 3.2 applies local don't cares. In Section 3.3, an alternative strategy which includes the relevant information in the SAT instance is briefly considered.

3.1 Observability at Outputs

As described in Section 2, for a given SAF, the fault must be justified at the faulty line and is then propagated towards the outputs, so that the fault effect is observable at least at one output.

During the creation of the SAT instance, it is not known along which paths the fault effect is propagated to the outputs. Therefore, all possible paths and their transitive fan-in cone have to be included. As a result, the test pattern is over-specified.

To reduce the number of specified bits in the test pattern t, a post-processor is applied after calculating the solution. The pseudo-code of the post-processor is shown in Figure 2.

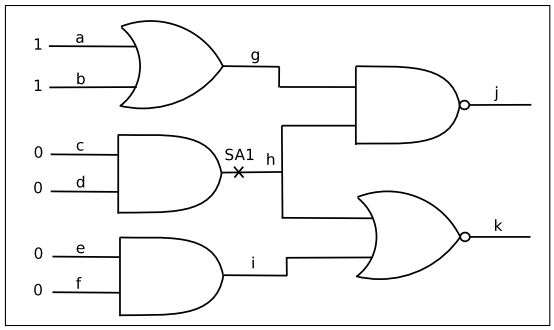

Figure 3. Example Circuit

First, all bits of the test pattern are set to X (line 1). Then, the inputs s of the transitive fan-in cone of output o at which the fault effect is observable are identified by backtracing (line 9-15). The assignments of all inputs in s are extracted and the corresponding bits in the test pattern are set to the computed value (line 18). Because it is sufficient that the fault effect can be observed at only one output, this must be done only once. Therefore, the complexity of this post-processing step is $\mathcal{O}(n)$, where n denotes the number of elements in the circuit for a randomly chosen o.

Because it is likely that the fault effect can be observed at more than one output, the number of specified bits in the test pattern depends on the chosen output. To find the output with the smallest number of specified bits for the calculated solution, the procedure must be executed for each output o at which the fault effect is observable. In this case, the complexity of the procedure is $\mathcal{O}(n \cdot k)$, where k denotes the number of outputs in the output cone.

Example 1. *Consider the circuit given in Figure 3. A SA1 fault is to be tested at line h. A corresponding test pattern of the classical SAT approach can be found at the left side of the inputs.*

The fault effect can be observed at both outputs. Choosing output j, the post-processor backtraces to the inputs a, b, c, d and sets the corresponding bits in the test pattern to the specified value. The bits for the inputs e, f remain X. Choosing output k however results in specified bits for inputs c, d, e, f and don't care bits for a, b.

3.2 Applying Local Don't Cares

In this section, a procedure which exploits the knowledge about local don't cares is introduced. This procedure can be combined with the technique presented in Section 3.1.

In the procedure described above, the inputs are identified by backtracing without considering internal assignments. Consequently, all inputs from which the output o is structural dependent are represented by

```
 1 testpattern t = X;
 2 set<input> s;
 3 list l;
 4 foreach(output o) do
 5 {
 6    if (observable(o))
 7    {
 8       l.push(o);
 9       while (!l.empty())
10       {
11          gate g = l.first_element();
12          if (g == INPUT) s.add(g);
13          else if (on_d_chain(g))
14               l.add(g.all_predecessors());
15          else if (contr_in_val(g))
16               l.add(g.pred_with_contr_val());
17          else
18               l.add(g.all_predecessors());
19          l.remove(g);
20       }
21       foreach (input i in s) do
22       {
23          t.set_computed_bit(i);
24       }
25       break;
26    }
27 }
```

Figure 4. Pseudo-code of the Post-Processor applying local Don't Cares

specified bits in the test pattern. But not all considered internal signals in the transitive fan-in cone of o are necessary for detecting the fault.

For determining the value of a basic gate g like AND, NAND, OR or NOR, it is not always necessary to know all values of the predecessors. If the controlling value c (0 for AND, NAND, 1 for OR, NOR) is applied at at least one incoming connection, then the value of the gate is determined. Consequently all other incoming connections can be substituted by X-values. Only one incoming connection with a controlling value has to be backtraced to guarantee the correct value of the gate.

This property can be exploited when calculating the transitive fan-in cone of output o under a specific assignment. The pseudo-code of the extended algorithm is shown in Figure 4. Instead of directly backtracing all predecessors of the considered gate of the circuit, the element is analyzed with respect to the assignment.

If the gate is located on a D-chain, i.e. the fault effect is propagated along this gate, all predecessors must be backtraced (line 13-14). This is due to the restriction, that all side-inputs of the D-chain must be set to non-controlling value to propagate the fault effect.

If the assignment of at least one incoming connection of the considered gate is the controlling value (line 15), then only the corresponding predecessor has to be backtraced. The other predecessors are not addressed anymore and therefore can be treated as X. Note, that this is not an exact method. In case of having more than one controlling value, the choice is based on heuristics.

In all other cases, all predecessors must be backtraced to ensure the correct value. Finally, the bits of all inputs which have been considered during backtracing are set to the computed value in the test pattern.

Equal to the procedure presented in Section 3.1, the number of specified bits depends on the chosen output. To determine the output with the smallest number of specified bits for the given assignment, the same procedure as above can be applied.

Example 2. *Consider again the circuit in Figure 3. Choosing k as observed output results in considering all predecessors of k, because k is on a D-chain, i.e. h and i have to be backtraced. Because both gates are controlled by at least one incoming connection, only one predecessor of h and i, respectively, have to be considered. This results in only two specified bits (one of $\{c, d\}$ and one of $\{e, f\}$) which are needed to detect the fault instead of all six.*

3.3 SAT Encoding

As an alternative to the post-processor, a strategy is presented which encodes the relevant information directly into the SAT instance. As a result, the information which bits of the test pattern can be set to unspecified value is derived directly from the solution.

Therefore, for each gate g in a circuit an additional variable g_C is introduced. This variable indicates whether a value can be substituted by a don't care ($g_C = 0$) or not ($g_C = 1$).

Let f be the gate with the outgoing fault connection, then f_C must be set to 1. Furthermore, it has to be guarenteed, that the C-variable of exactly one output of which the D-variable (see Section 2) is assigned to 1 is also assigned to 1. The values have to be propagated towards the inputs with respect to local don't cares by including additional implications into the SAT instance.

This encoding was implemented and experiments were carried out. Although the results were comparable to those of the post-processing in terms of quality, the run time and number of not classified faults increased significantly due to the increased size of the SAT instance. Due to page limitation, the results are therefore left out.

4 Experimental Results

In this section, experimental results of the presented approaches are shown. The industrial circuits are provided by NXP Semiconductors GmbH, Germany. All experiments were carried out on an AMD64 3500+ (2200 MHz, 4096 MByte, GNU/Linux). As SAT solver, MiniSat [3] was used.

The general test procedure is as follows. For a given fault, a test pattern is computed. Afterwards, the post-processor is started to reduce the number of specified

Table 1. Results – Run Time

	default		post		post min		post ext.		post min ext.	
circ	time	ab.	time	ab.	time	ab.	time	ab.	time	ab.
p44k	48:13m	0	48:24m	0	47:58m	0	50:55m	0	51:09m	0
p49k	2:14h	102	2:13h	103	2:13h	103	2:13h	103	2:18h	108
p77k	0:30m	0	0:29m	0	0:29m	0	0:29m	0	0:29m	0
p80k	11:54m	0	29:24m	0	29:23m	0	30:08m	0	30:10m	0
p88k	12:36m	0	12:59m	0	13:05m	0	12:51m	0	12:56m	0
p99k	9:35m	0	9:30m	0	9:33m	0	9:40m	0	9:45m	0
p177k	1:40h	0	1:39h	0	1:39h	0	1:51h	0	1:53h	0
p462k	2:58h	10	2:57h	10	2:56h	10	2:59h	10	2:59h	10
p565k	2:38h	0	2:39h	0	2:38h	0	2:39h	0	2:39h	0
p1330k	6:01h	0	6:07h	0	6:10h	0	5:56h	0	5:56h	0

bits. Additionally, a fault simulator is started to check whether the test pattern finds additional faults.

In the following, we first discuss the run time of the algorithms followed by an analysis of the quality.

In Table 1, results for the approaches presented in Section 3 are shown. The name of the circuit (column *circ*) roughly denotes the size of the circuit, e.g. p1330k contains over 1.3 millions of gates. For comparison, results of the default approach without the usage of a post-processor are presented in column *default*. In column *post*, results of the approach presented in Section 3.1 are given for a randomly chosen output, while column *post min* provides results for the output with the smallest number of specified bits. The results for the approaches presented in Section 3.2 are given in column *post ext.* and *post min ext.*, respectively.

Column *time* provides the run time. In column *ab.*, the number of faults which could not be classified due to time limit is presented.

Studying the results of Table 1, it can be observed, that the additional use of the post-processor only results in small run time overhead. This is due to the linear complexity of the algorithm. Only in case of p80k, the run time is higher by more than a factor of two. This can be explained by the likewise increased number of test generator calls, i.e. the fault simulator detects a smaller number of additional faults detected by the test patterns. In all other cases, this increase of the calls cannot be observed.

Comparing the configurations *post* and *post min*, it can be noticed, that using *post min* results more often in (minimal) smaller run time, although more calculations have to be done. This is due to the usage of a fault simulator, i.e. different test patterns cause a different set of targeted faults. This cannot be observed using *post ext.* and *post min ext.* Here, the run time of *post ext.* remains always smaller (or equal) compared to *post min ext.*

Compared to the default configuration, the run times using the post-processor are in most – but not all – cases only slightly larger (except p80k). The number of not classified faults remains almost stable.

In Table 2 and Table 3, results concerning the average number of specified bits are presented for the approach with and without local don't cares, respectively. Column *#Input* lists the total number of inputs (both primary and pseudo-primary) of the circuit. In column *%bits*, the average percentage of specified bits of the presented approaches is provided (in *default*, this is the number of inputs included in the SAT instance). Column *%def* gives the percentage of specified bits in relation to the default configuration. Finally, column *#PAT* denotes the number of generated test patterns.

The usage of a post-processor without applying local don't cares reduces the specified bits significantly. The results show a reduction of up to 69%. It can be noticed, that there are only slight differences between the configurations *post* and *post min*. Although in *post min*, the output with the smallest number of specified bits is considered, the total number of specified bits is not always smaller. This is again due to the usage of a fault simulator.

Applying the post-processor considering local don't cares results in a even more reduced percentage of specified bits in test patterns. In the worst case, the reduction is still over 60%, while in the best case it is up to 97% (p177k).

The experiments show, that the presented post-processor is able to reduce the number of care bits drastically. With nearly no overhead in run time, SAT-based ATPG algorithms are able to generate compact test patterns which are well suited for techniques like test compaction and compression.

5 Conclusions

In this paper, we presented techniques which make use of structural properties of the circuit and apply local don't cares in form of a post-processor. As a result, the number of specified bits in test patterns generated from SAT-based ATPG tools could be reduced significantly of up to 97% as experimental results have shown, while run time is negligible in most cases. The resulting test patterns are well suited for techniques like test compaction and compression.

Table 2. Specified Bits – Post-Processing

		default			post			post min		
circ	#Input	%bits	%def	#Pat	%bits	%def	#Pat	%bits	%def	#Pat
p44k	2914	70.01	100	5946	59.31	76.56	5542	59.31	76.56	5542
p49k	637	47.38	100	379	17.86	37.82	373	17.85	37.80	376
p77k	3148	1.94	100	123	0.59	31.27	121	0.59	31.83	125
p80k	4030	10.05	100	4025	4.99	49.71	10694	4.99	49.71	10694
p88k	4712	4.38	100	5757	2.95	67.05	5890	2.95	67.05	5890
p99k	5914	5.48	100	3300	4.23	77.15	3285	4.23	77.15	3285
p177k	11275	21.24	100	3755	7.97	38.16	3890	7.93	37.98	3846
p462k	31020	0.85	100	9316	0.30	35.31	9223	0.30	35.42	9198
p565k	33405	0.24	100	8638	0.16	66.80	8664	0.16	67.26	8715
p1330k	105247	0.26	100	12151	0.17	69.81	12477	0.17	69.81	12477

Table 3. Specified bits – Post-Processing applying local Don't Cares

		default			post ext.			post min ext.		
circ	#Input	%bits	%def	#Pat	%bits	%def	#Pat	%bits	%def	#Pat
p44k	2914	70.01	100	5946	7.59	9.72	6149	7.59	9.72	6149
p49k	637	47.38	100	379	16.76	35.48	368	16.68	35.32	377
p77k	3148	1.94	100	123	0.49	26.51	118	0.49	26.51	118
p80k	4030	10.05	100	4025	3.17	31.64	12915	3.17	31.64	10985
p88k	4712	4.38	100	5757	1.15	26.23	5752	1.15	26.23	5752
p99k	5914	5.48	100	3300	1.52	27.86	3354	1.52	27.86	3354
p177k	11275	23.24	100	3755	0.69	2.99	4086	0.70	3.00	4113
p462k	31020	0.85	100	9316	0.13	15.92	9254	0.14	15.96	9235
p565k	33405	0.24	100	8638	0.09	39.79	8695	0.09	39.83	8729
p1330k	105247	0.26	100	12151	0.04	16.59	11967	0.04	16.59	11967

References

[1] M. Davis, G. Logeman, and D. Loveland. A machine program for theorem proving. *Comm. of the ACM*, 5:394–397, 1962.

[2] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:506–521, 1960.

[3] N. Eén and N. Sörensson. An extensible SAT solver. In *SAT 2003*, volume 2919 of *LNCS*, pages 502–518, 2004.

[4] S. Eggersglüß, G. Fey, and R. Drechsler. SAT-based ATPG for path delay faults in sequential circuits. In *IEEE Int'l Symp. on Circuits and Systems*, 2007.

[5] S. Eggersglüß, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloeffel. Combining multi-valued logics in SAT-based ATPG for path delay faults. In *ACM & IEEE Int'l Conf. on Formal Methods and Models for Codesign*, 2007.

[6] S. Eggersglüß, D. Tille, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloeffel. Experimental studies on SAT-based ATPG for gate delay faults. In *Int'l Symp. on Multi-Valued Logic*, 2007.

[7] G. Fey, J. Shi, and R. Drechsler. Efficiency of multi-valued encoding in SAT-based ATPG. In *Int'l Symp. on Multi-Valued Logic*, pages 25–30, 2006.

[8] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Trans. on Comp.*, 32:1137–1144, 1983.

[9] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic. *IEEE Trans. on Comp.*, 30:215–222, 1981.

[10] T. Larrabee. Test pattern generation using Boolean satisfiability. *IEEE Trans. on CAD*, 11:4–15, 1992.

[11] J. Marques-Silva and K. Sakallah. GRASP: A search algorithm for propositional satisfiability. *IEEE Trans. on Comp.*, 48(5):506–521, 1999.

[12] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conf.*, pages 530–535, 2001.

[13] K. Ravi and F. Somenzi. Minimal assignments for bounded model checking. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 2988 of *LNCS*, pages 31–45, 2004.

[14] J. Roth. Diagnosis of automata failures: A calculus and a method. *IBM J. Res. Dev.*, 10:278–281, 1966.

[15] S. Safarpour, A. Veneris, R. Drechsler, and J. Hang. Managing don't cares in Boolean satisfiability. In *Design, Automation and Test in Europe*, pages 260–265, 2004.

[16] S. Shen, Y. Qin, and S. Li. A faster counterexample minimization algorithm based on refutation analysis. In *Design, Automation and Test in Europe*, pages 672–677, 2005.

[17] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schlöffel. PASSAT: Efficient SAT-based test pattern generation for industrial circuits. In *IEEE Annual Symposium on VLSI*, pages 212–217, 2005.

[18] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. Combinational test generation using satisfiability. *IEEE Trans. on CAD*, 15:1167–1176, 1996.

[19] P. Tafertshofer, A. Ganz, and K. Antreich. Igraine - an implication graph based engine for fast implication, justification, and propagation. *IEEE Trans. on CAD*, 19(8):907–927, 2000.