Experimental Studies on SAT-based ATPG for Gate Delay Faults

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

Andreas Glowatz Friedrich Hapke Jürgen Schlöffel
NXP Semiconductors GmbH
21147 Hamburg, Germany
{andreas.glowatz,friedrich.hapke,juergen.schloeffel}@nxp.com

Abstract

The clock rate of modern chips is still increasing and at the same time the gate size decreases. As a result, already slight variations during the production process may cause a functional failure. Therefore, dynamic fault models like the gate delay fault model are becoming more important. Meanwhile classical algorithms for test pattern generation reach their limits regarding run time and memory needs.

In this work, a SAT-based approach to calculate test patterns for gate delay faults is presented. The basic transformation is explained in detail. The application to industrial circuits – where multi-valued logic has to be considered – is studied and experimental results are reported.

1 Introduction

Increasing clock rates and decreasing gate sizes in modern chips imply tight constraints on the physical realisation. Slight process variations during production may already cause a violation of these constraints. As a result, a delay in the propagation of signal values may occur. Such a functional defect is often only detectable while the chip is operating at speed. Input stimuli to identify such a failure are typically calculated on a gate level representation of the circuit. Dedicated logical fault models such as the *Path Delay Fault Model* (PDFM) and the *Gate Delay Fault Model* (GDFM) have been developed to model such delay effects.

Delay effects along individual paths from inputs to outputs are considered in the PDFM. Testing all such paths is desirable. But due to the large number of paths already in medium sized circuits, only a small subset can be considered. Therefore the GDFM is very important in practice. This model has been proposed to detect delay effects occurring at gates. Thus, addressing all faults is possible, even for large industrial circuits.

Generating test patterns for such dynamic fault models is computationally very intensive. Even calculating a test pattern for a static *Stuck-At Fault* (SAF) is an NP-complete problem [6]; although it is sufficient to model the circuit in a single time frame. But for dynamic fault models the dynamic behavior within (at least) two consecutive time frames has to be considered. The circuit is initialised by using a first test pattern, then a second test pattern is applied at speed to evaluate the functional correctness of the circuit.

At the same time, the size of the circuits that have to be modelled is steadily increasing and doubles every 18 months. As a result, classical ATPG algorithms reach their limits. On the other hand, tools to solve the *Boolean satisfiability* (SAT) problem have been significantly improved in the recent past [9, 10, 3]. Consequently, they were applied to efficiently solve many practical problems ranging from formal verification [1] and path sensitisation [7] to test pattern generation for SAFs [12].

In this work, we propose a SAT-based approach to calculate test patterns for the GDFM in industrial circuits containing multi-valued logic. For this purpose, the circuit is unrolled for two time frames. By this, the dynamic behavior is modeled adequately without using a special logic. The classification of a *Gate Delay Fault* (GDF) is reduced to injecting a SAF in the second time frame and forcing a transition after the first time frame. Moreover, the approach handles industrial circuits containing multi-valued logic. Therefore, a four-valued logic that includes tri-state values and unknown values to model environment restrictions is applied. As suggested in [4], this multi-valued logic

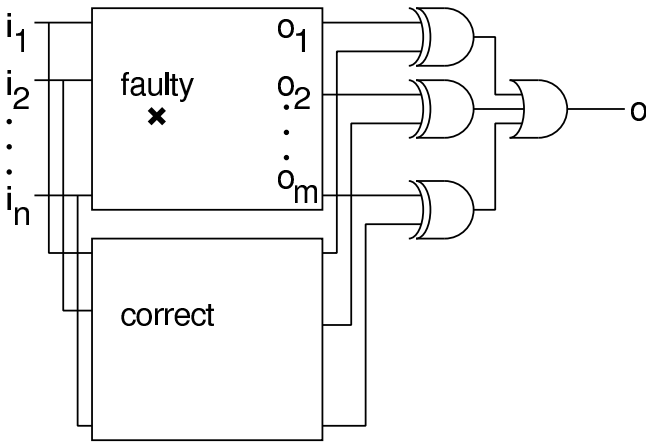

Figure 1. Miter circuit C_m

is mapped to Boolean values to apply a modern Boolean SAT solver. Experimental results for large industrial circuits show the feasibility of the approach.

The paper is structured as follows: The next section explains the main concepts of SAT-based ATPG, whereas Section 3 deals with the GDFM and the reduction to SAFs. Experimental results are presented in Section 4. In Section 5, conclusions are drawn.

2 SAT-based ATPG

In this section, SAT-based ATPG as introduced in [8, 13, 12] is briefly reviewed. The general transformation of an ATPG problem into a SAT problem is considered in Section 2.1, whereas in Section 2.2 the *stuck-at fault model* (SAFM) is introduced. Section 2.3 shows the use of structural information to reduce the complexity of the SAT instance. In Section 2.4, a multi-valued encoding for circuits with unknown signal values and signals with high impedance is presented.

2.1 SAT Formulation

To find a test for a given circuit C and a fault F, the faulty and the correct circuit are joined to form a miter circuit C_m [2] as shown in Figure 1. In a miter circuit, the faulty and correct circuit share the same inputs $(i_1, ..., i_n)$. The output behaviour of the two circuits is then analysed by comparing the corresponding outputs $(o_1, ..., o_m)$ using XOR gates. The output o of the miter is true, iff at least one output value differs in the faulty circuit and the correct circuit. An assignment to $i_1, ..., i_n$ is a legal test, iff o evaluates to true.

To apply a SAT solver, this miter circuit must be transformed into a Boolean formula represented in *Conjunctive Normal Form* (CNF). A CNF is a conjunction of clauses. A clause is a disjunction of literals, whereas a literal is a Boolean variable or its complement. Each signal line x in C_m is represented by a Boolean variable x and each single gate g in C_m can be converted into a CNF Φ_g by building the characteristic function χ_g of g. Each assignment of the variables attached to g which evaluates χ_g to true is a legal description of g. The following example clarifies the procedure.

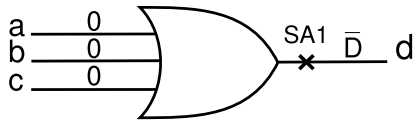

Figure 2. Stuck-at fault

Example 1 *Given an AND gate with the inputs a, b and the output c, the CNF Φ^c_{AND} for this gate can be created by the following Boolean transformations.*

$$\begin{aligned}
\Phi^c_{\text{AND}} &= (a \wedge b \equiv c) \\
&= \overline{((a \wedge b) \oplus c)} \\
&= \overline{(((a \wedge b) \wedge \overline{c}) \vee ((\overline{a \wedge b}) \wedge c))} \\
&= \overline{((a \wedge b \wedge \overline{c}) \vee ((\overline{a} \vee \overline{b}) \wedge c))} \\
&= \overline{((a \wedge b \wedge \overline{c}) \vee (c \wedge \overline{a} \vee c \wedge \overline{b}))} \\
&= (\overline{a} \vee \overline{b} \vee c) \wedge (\overline{c} \vee a) \wedge (\overline{c} \vee b)
\end{aligned}$$

The CNF of other Boolean gates can be created similarly.

The CNF for C_m is therefore the conjunction of the CNFs for each gate.

$$\Phi_{Cm} = \prod_{g \in C_m} \Phi_g$$

To find a test which detects F, an additional contraint must be added. The output o of the miter circuit must be set to 1. As described above, this guarentees that at least one output of C differs in the faulty version and the correct version. This constraint is given by Φ_f.

A legal test for F is derived by applying a SAT solver to $\Phi_{Cm} \wedge \Phi_f$. Every satisfiable assignment can be converted to a test by evaluating the values of the Boolean variables attached to the inputs. If there does not exist a satisfiable solution, the fault is redundant.

2.2 Stuck-at Faults

The SAFM is a static fault model and describes a signal line, which *stucks* permanently at a fixed value 0 or 1. Therefore, there exist two different types of SAFs depending on their fixed value: stuck-at-0 (SA0) and stuck-at-1 (SA1).

A well-known approach for generating test patterns for the SAFM is the D-algorithm [11]. Here, the Boolean logic is extended by the values D and $\overline{D}$. The value D ($\overline{D}$) is applied to a signal line if the line is assigned to 1 (0) in the correct and 0 (1) in the faulty circuit. The following example clarifies the procedure:

Example 2 *Consider the OR gate shown in Figure 2. A SA1 fault occurs at line d. To find a corresponding test for the given fault, the values of the inputs must be fixed to the non-controlling value of the gate. This implies $\overline{D}$ on the output.*

2.3 Using Structural Information

Solving the CNF of a full miter circuit is very complex. Therefore, some optimisations are done to improve the solving process. First, only a part of the miter circuit is included in the SAT instance. Including the fanout-cone of F and the fanin-cone of all influenced outputs is sufficient. The remaining part of the circuit is not relevant for generating a test. Moreover, it is sufficient to duplicate the fanout-cone of the fault, because the fault effect can only be propagated therein. In this way, the size of the SAT instance, i.e. the number of clauses and literals, can be reduced and the run time and memory needs of the solving process decrease.

Another important improvement is the inclusion of structural information in the SAT instance as shown in [13]. There, the concept of D-chains – originally proposed for the D-algorithm [11] – was applied in the ATPG algorithm TEGUS.

For each gate g on a potential D-chain, an additional variable g_D is introduced. A gate g is on a potential D-chain if it is on a path from the error location to an output. The variable g_D equals 1 if the fault effect is propagated to an output via g or, in other words, if the value g_c in the correct circuit and the value g_f in the faulty circuit differ:

$$g_D \rightarrow g_c \oplus g_f$$

In the following, the formulation of this implication in SAT is represented by Φ_D^{gD}.

If a gate g propagates the fault effect to an output, at least one successor $h_1, ..., h_n$ must also propagate the fault effect. Therefore, the following implication is included in the SAT instance:

$$g_D \rightarrow \sum_{i=1}^{n} h_i$$

Example 3 *Figure 3 shows an example circuit including a SA1 on line e. The continuous lines distinguish the original circuit, while the broken lines mark the duplicated fanout-cone of the fault. For line e and line g, additional variables e_D and g_D are added, respectively.*

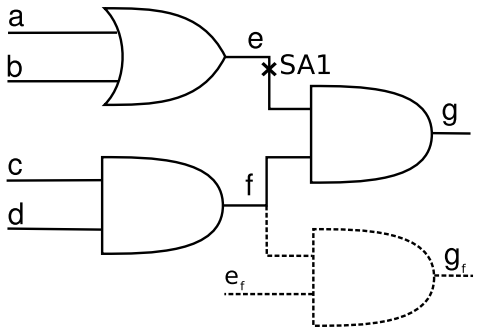

Figure 3. Example circuit with duplicated fanout-cone

Table 1. Encoding of L_4

x	c_x	c_x^*	Interpretation
0	0	0	x is 0
1	1	0	x is 1
U	1	1	x is unknown
Z	0	1	x is at high impedance

To calculate a test, the constraints described above are added and the faulty line e_f must be fixed to 1. This results in the following CNF Φ_{SA1}:

$$\begin{aligned}\Phi_{SA1} &= \Phi^{e}_{\text{OR}} \wedge \Phi^{f}_{\text{AND}} \wedge \Phi^{g}_{\text{AND}} \\ &\quad \wedge \Phi^{gf}_{\text{AND}} \wedge \Phi^{gD}_{\text{D}} \wedge (e_f) \wedge (g_D)\end{aligned}$$

The corresponding test is: $\{a = 0, b = 0, c = 1, d = 1\}$

2.4 Multi-Valued Logic

As described above, each signal line in a circuit can be encoded by one Boolean variable. However, industrial circuits contain multi-valued elements, e.g. tri-state elements, and restrictions which cannot be modelled by Boolean logic. Therefore, two additional values U (unknown) and Z (high impedance) are needed. An unknown value is often applied to inputs, which cannot be controlled, e.g. due to the integration in a larger circuit. A Z value can be produced e.g. by a busdriver. This results in a four-valued logic $L_4 = \{0, 1, U, Z\}$.

Because SAT is usually defined for formulas in Boolean logic, each value of L_4 is encoded by two Boolean variables c, c^*. There are 24 possible encodings for L_4. A detailed discussion is given in [4]. The encoding for a signal x used in our approach is shown in Table 1.

3 Gate Delay Faults

In contrast to the SAFM, the GDFM is a dynamic fault model. It describes a delayed signal at a gate. Due to this,

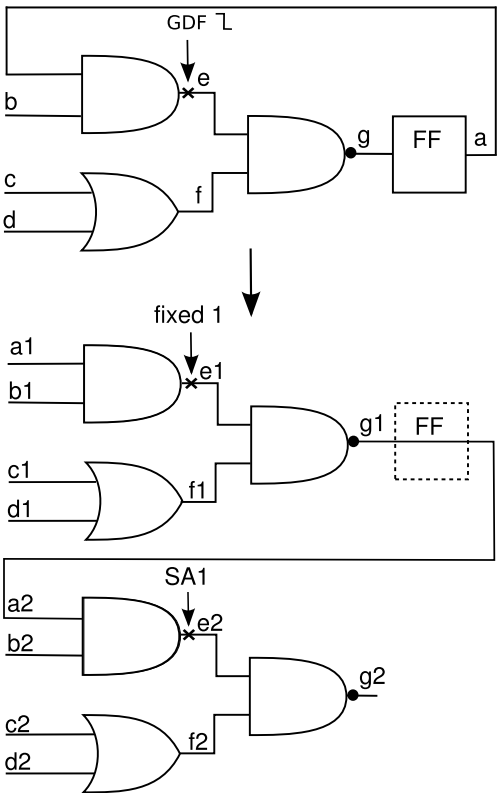

Figure 4. Unrolled example circuit

the faulty value is propagated. There exist two different types of GDFs: *rising* and *falling*. A rising GDF describes a delayed transition from logic 0 in the *initial* time frame t_1 to logic 1 in the *final* time frame t_2, whereas the falling GDF describes a delayed transition from logic 1 in t_1 to logic 0 in t_2.

To detect a GDF, two vectors v_1, v_2 are needed. The vector v_1 is the test vector for t_1 and the vector v_2 the test vector for t_2. The initial vector sets the initial value of the transition, whereas the final vector causes the transition at speed. To detect a fault, the transition must be propagated to an output.

As shown in [5], GDFs can be modelled by injecting SAFs. Therefore, the initial value of the faulty line must be fixed to the initial value of the GDF. Then, a SA0 (SA1) fault is injected at the faulty line in t_2 for a rising (falling) GDF to guarentee a detection of a faulty value, i.e. a delay, in t_2 and its propagation to an output.

For this reason, the circuit C is unrolled by duplicating it. The original circuit C_1 represents t_1 and the duplicated circuit C_2 represents t_2. The sequential behaviour, i.e. the correct behaviour of latches, is modelled by connections between the original and the duplicated circuit.

To apply a SAT solver to the problem, the unrolled circuit C_t must be transformed into a CNF derived from the following equation:

$$\Phi_{Ct} = \Phi_{C1} \wedge \Phi_{C2} \wedge \Phi_{seq}$$

The CNF for C_1 is represented by Φ_{C1}, whereas Φ_{C2} is the CNF for C_2. The term Φ_{seq} describes the sequential behaviour of C_t by connecting the pseudo primary outputs of C_1 with the pseudo primary inputs of C_2. Note, that the CNF of the circuit is not derived directly from Boolean logic, but from the Boolean encoding of the multi-valued logic presented in Section 2.4.

Considering the constraints for modelling a GDF represented by Φ_{fm}, the CNF Φ_{test} for a GDF test is given by:

$$\Phi_{test} = \Phi_{Ct} \wedge \Phi_{fm}$$

Example 4 *Figure 4 shows a circuit in original form (upper part) and in unrolled form (lower part) with a falling GDF at line e. After duplicating the circuit, the pseudo primary output g is the input of the corresponding pseudo primary input $a2$ in the duplicated alle elements of the circuit. To initialise the test in the initial time frame, line $e1$ is fixed to 1, whereas a SA1 fault is injected at $e2$ to propagate the delayed signal in the final time frame to an output.*

The CNF Φ_{Ct} for the unrolled circuit is given by the conjunction of the following CNFs Φ_{C1}, Φ_{C2} and Φ_{seq}:

$$\begin{aligned}
\Phi_{C1} &= \Phi^{e1}_{\text{AND}} \wedge \Phi^{f1}_{\text{OR}} \wedge \Phi^{g1}_{\text{NAND}} \\
\Phi_{C2} &= \Phi^{e2}_{\text{AND}} \wedge \Phi^{f2}_{\text{OR}} \wedge \Phi^{g2}_{\text{NAND}} \\
\Phi_{seq} &= (g1 \vee \overline{a2}) \wedge (\overline{g1} \vee a2)
\end{aligned}$$

The constraints for modelling the falling GDF on line e are shown in the following equation:

$$\Phi_{fm} = (e1) \wedge \Phi^{g2f}_{\text{NAND}} \wedge \Phi^{g2D}_{\text{D}} \wedge (e2_f) \wedge (g2_D)$$

A corresponding test for the falling GDF on line e is:

$$\begin{aligned}
v_1 &= \{a1 = 1, b1 = 1, c1 = 1, d1 = X\} \\
v_2 &= \{b2 = 1, c2 = 1, d2 = X\}
\end{aligned}$$

Note, that in practice an additional constraint must be added. The values of a primary input must be equivalent in both time frames. This is due to the test equipment, where it is hard to change the test value on the primary inputs at speed during the test procedure. This constraint is considered in our approach and given by the following CNF Φ_{eq}:

$$\begin{aligned}
\Phi_{eq} &= (\overline{b1} \vee b2) \wedge (b1 \vee \overline{b2}) \wedge (\overline{c1} \vee c2) \wedge (c1 \vee \overline{c2}) \\
&\quad \wedge (\overline{d1} \vee d2) \wedge (d1 \vee \overline{d2})
\end{aligned}$$

The test presented above is therefore invalid, because d has no valid assignment in practice. A valid test is:

$$\begin{aligned}
v_1 &= \{a1 = 1, b1 = 1, c1 = 1, d1 = 0\} \\
v_2 &= \{b2 = 1, c2 = 1, d2 = 0\}
\end{aligned}$$

Table 2. Memory usage and run time for ATPG

					Stuck-at		*Gate delay*	
circuit	*in*	*out*	*la*	*tri*	*time*	*mem*	*time*	*mem*
p44k	739	56	2175	0	3:13h	89.6MB	>20h	111.6MB
p77k	171	507	2977	0	0:34m	133.1MB	4:49h	153.3MB
p80k	152	75	3878	0	59:15m	180.0MB	5:40h	284.9MB
p88k	331	256	4309	412	16:12m	128.3MB	2:22h	183.3MB
p99k	167	94	5747	0	13:25m	124.6MB	1:03h	178.8MB
p462k	1815	1193	29205	597	3:41h	748.9MB	13:41h	901.4MB
p565k	964	210	32409	169	2:41h	948.8MB	5:35h	1.246GB

Table 3. Instance sizes

	Stuck-at				*Gate delay*			
	Max		*Mean*		*Max*		*Mean*	
circuit	*Vars*	*Cls*	*Vars*	*Cls*	*Vars*	*Cls*	*Vars*	*Cls*
p44k	101,491	328,732	60,446	209,001	201,692	667,582	86,661	299,822
p77k	240,912	827,614	848	2,765	42,507	154,039	6,689	22,462
p80k	356,452	1,293,556	7,483	22,697	611,070	2,218,834	20,768	67,777
p88k	93,133	298,652	5,047	15,676	222,865	760,032	17,976	55,152
p99k	34,640	129,746	5,301	16,139	268,238	886,489	10,288	31,919
p462k	391,113	1,361,732	7,336	22,399	764,654	2,525,819	31,159	92,039
p565k	1,705,996	5,536,393	4,663	15,638	259,513	916,592	11,920	35,727

4 Experimental Results

The techniques presented in the previous sections have been implemented in C++. Experimental results for this implementation on some industrial benchmark circuits from NXP Semiconductors GmbH, Hamburg, Germany are provided in this section. The experiments were carried out on an AMD Athlon 3500+ (2.2 GHz, 4096MByte, Linux). As the ATPG core engine, an improved version of the tool PASSAT [12] was used. Note, that a fault simulator is used for identifying other faults, that are detected by a generated test as it is industrial practice. As a result, the set of tested faults depends on the ATPG algorithm.

Table 2 presents the run times and the needed memory for ATPG. In the first column the name of the circuit is presented. Here, the number within the name shows the number of gates the circuit contains, i.e. p565k means, that there are more than half a million gates in the circuit. Hereafter, further details of the circuit are given, e.g. column *in* gives the number of inputs, column *out* the number of outputs, *la* the number of latches and *tri* the number of tri-state elements.

The succeeding columns are divided into two parts. The first part presents results of ATPG for the SAFM and the second part shows results of ATPG for the GDFM. For both fault models, the run time (column *time*) and the used memory (column *mem*) are given. The run time for GDFs is higher than the run time for SAFs by a factor of 2-13, whereas the memory usage only increases approximately by a factor of 1.5. Although considering a more complex fault model, the run times and memory needs for the GDFM are still comparable to the results for the SAFM.

The longer run times of ATPG for GDFs are caused by several aspects. First, because of unrolling the circuit, the depth of the circuit is larger than for SAFs. Therefore the SAT instance contains usually more clauses and variables as it is shown in Table 3. This table is again divided into two parts. The left part lists the data for SAFs and the right part shows the data for GDFs. The columns *Max* and *Mean* give the largest and the average size, respectively, where the columns (*Cls*) and (*Vars*) give the number of variables and clauses, respectively.

As expected, – due to unrolling the circuit – the SAT instances for GDFs are generally much bigger (approximately by a factor of 2-7) than the SAT instances for SAFs. Only for p77k and p565k the maximum size of the variables and clauses is larger while testing SAFs. But the average size of the instances shows, that the large numbers are exceptions. This is explained by the use of a fault simulator, because the set of SAFs, which are injected for modelling GDFs, is different to the set of SAFs.

Table 4 shows the number of targets (column *targets*), the number of redundant faults (column *redundant*) and the number of aborted faults (20 sec per fault) (*aborted*) for both fault models. The large difference of the run times of *p77k* in both fault models can be explained by compar-

Table 4. Fault Coverage

	Stuck-at			*Gate delay*		
circuit	*targets*	*redundant*	*aborted*	*targets*	*redundant*	*aborted*
p44k	64,105	2,418	0	109,806	39,169	0
p77k	163,310	9,181	0	282,728	199,008	0
p80k	197,834	124	0	311,426	14,289	62
p88k	147,742	2,756	0	156,744	22,699	0
p99k	162,019	2,141	0	273,376	28,015	2
p462k	673,949	132,112	10	1,135,408	485,848	37
p565k	1,026,851	26,917	0	1,525,586	88,536	0

ing the number of redundant faults for the SAFM and the GDFM. While for the SAFM the fault coverage is about 94%, the fault coverage for the GDFM is about 30%.

Although the GDFM is a more powerful (dynamic) fault model than the SAFM, the number of aborted GDFs remains very small in comparison to the number of the targets in the circuit. Almost all faults can be classified.

5 Conclusions

In this paper we presented a SAT-based approach to calculate tests for the GDFM. We showed the modelling of GDFs by injecting SAFs and studied the formulation for a modern Boolean SAT solver in detail. Experimental results for industrial circuits containing multi-valued logic were provided, which show, that a SAT-based approach is well suited for the GDFM as well.

6 Acknowledgements

This research work was supported in part by the German Federal Ministry of Education and Research (BMBF) in the Project MAYA under the contract number 01M3172B and in part by DFG grant DR 287/15-1.

References

[1] A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1579 of *LNCS*, pages 193–207. Springer Verlag, 1999.

[2] D. Brand. Verification of large synthesized designs. In *Int'l Conf. on CAD*, pages 534–537, 1993.

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

[4] G. Fey, J. Shi, and R. Drechsler. Efficiency of multi-valued encoding in SAT-based ATPG. In *ISMVL '06: Proceedings of the 36th International Symposium on Multiple-Valued Logic (ISMVL'06)*, pages 25–30, 2006.

[5] M. Gharaybeh, M. Bushnell, and V. Agrawal. Classification and test generation for path-delay faults using single stuck-fault tests. In *Int'l Test Conf.*, pages 139–147, 1995.

[6] O. Ibarra and S. Sahni. Polynomially complete fault detection problems. *IEEE Trans. on Comp.*, 24:242–249, 1975.

[7] J. Kim, J. Whittemore, J. P. Marques-Silva, and K. Sakallah. On applying incremental satisfiability to delay fault testing. In *DATE '00: Proceedings of the conference on Design, automation and test in Europe*, pages 380–384, 2000.

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

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

[10] 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.

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

[12] 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.

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