Journal of Electronic Testing - Theory and Applications manuscript No.
(will be inserted by the editor)

MONSOON: SAT-based ATPG for Path Delay Faults Using Multiple-Valued Logics

Stephan Eggersglüß · Görschwin Fey · Andreas Glowatz · Friedrich Hapke · Juergen Schloeffel · Rolf Drechsler

Received: 28 August 2008 / Accepted: 22 January 2010

Abstract As technology scales down into the nanometer era, delay testing of modern chips has become more and more important. Tests for the path delay fault model are widely used to detect small delay defects and to verify the correct temporal behavior of a circuit.

In this article, MONSOON, an efficient SAT-based approach for generating non-robust and robust test patterns for path delay faults is presented. MONSOON handles tri-state elements and environmental constraints occurring in industrial practice using multiple-valued logics. Structural techniques increase the efficiency of the algorithm. A comparison with a state-of-the-art approach shows a significant speed-up. Experimental results for large industrial circuits demonstrates the feasibility and robustness of MONSOON.

Keywords SAT · Delay testing · ATPG · Path delay faults · Multiple-valued logics

1 Introduction

Due to the increased speed and continuously shrinking feature size of modern circuits, delay testing has become an important issue in the post production test. Several delay fault models have been proposed. Among them, the prevalent fault models are the transition delay fault model and the *Path Delay Fault* (PDF) model [1,2]. The transition delay fault model assumes a large delay defect at one fault site, whereas the PDF model captures small as well as large delay defects distributed along one path in the circuit. Therefore, the PDF model is more accurate. However, the number of paths in the circuit is typically very large. For that reason, usually only tests for critical paths are generated to ensure the correct timing behavior of these paths. Furthermore, tests for the PDFM are used for diagnostic reasons. High-quality tests are required for diagnosis.

Concerning the quality, tests for PDFs can be roughly classified in two categories [3]: non-robust and robust. Both categories differ in the sensitization criteria of the path. Modeling static values for the robust sensitization criterion makes the robust test generation harder. Nonetheless, robust tests provide a higher quality and are therefore more desirable.

Due to the increased complexity of today's designs, structural ATPG algorithms reach their limits, especially when high quality tests are required. The number of faults which cannot be classified grows. On the other hand, powerful *Boolean Satisfiability* (SAT) solvers emerged in the last decade. SAT solvers were shown to be very robust in the field of ATPG [4,5]. Structural ATPG algorithms use static implications to prove many faults untestable [6,7]. In contrast, SAT-based ATPG algorithms use conflict-based learning [8] to dynamically record additional signal correlations or implications, respectively in terms of conflict clauses. The effect of static implications is shown to be subsumed by conflict-based learning and the reuse of learned information [9] in SAT-based ATPG in [10]. However, since SAT-based algorithms work on a Boolean formula instead of the circuit's structure, the ATPG problem has to be converted into a SAT problem.

This is the preliminary version of the article. The final publication is available at www.springerlink.com

S. Eggersglüß · G. Fey · R. Drechsler
University of Bremen,
28359 Bremen, Germany
E-mail: segg@informatik.uni-bremen.de

A. Glowatz · F. Hapke · J. Schloeffel
Mentor Graphics Development GmbH,
21079 Hamburg, Germany

Section 1.1 presents the related work, whereas the contribution of this article is outlined in Section 1.2.

1.1 Related Work

Four different types of test generation algorithms for PDFs can be identified: structural, non-enumerative, algebraic and SAT-based algorithms. Structural algorithms, e.g. [11], work directly on the circuit-structure and benefit from applying efficient implications procedures based on multiple-valued logics. These are typically applied to a subset of critical paths extracted by dedicated algorithms, e.g. [12,13].

Non-enumerative procedures, e.g. [14–16], do not target any specific path, but generate tests for all PDFs in the circuit. By this, enumerating a potentially exponential number of paths in a circuit is avoided. These algorithms are usually based on sensitizing sub-paths or sub-circuits. Algebraic algorithms, e.g. [17], work on canonical data structures like BDDs. These algorithms suffer from their large memory consumption. In [18], a ZBDD-based test generation algorithm for PDF tests for critical paths is presented. However, only non-robust tests for full-scanned circuits are considered.

Similar to the structural algorithms, SAT-based algorithms for PDF testing work on a fault list that contains critical paths. Each path is sensitized separately according to the desired sensitization criterion. The first SAT-based approach was proposed in [19]. There, a seven-valued logic is presented to generate robust tests for PDFs. But the sequential behavior of the circuit, i.e. launch-on-capture, cannot be modeled adequately using this logic. In [20,21], *Incremental SAT* [22,23] is used to speed up test generation for PDFs. However, only non-robust tests are considered.

An efficient circuit-based SAT solver was presented in [24]. This solver which uses ATPG techniques is applied for PDF test generation in [25,26]. The advantage of circuit-based SAT solvers is – simply spoken – that they can apply structural knowledge to speed up the search process. The approach described in [25] focuses on non-robust test generation, whereas [26] targets non-robust as well as robust test generation. However, static values that are necessary for robust tests are not modeled. The underlying circuit-based SAT solver only supports Boolean values. Due to this simplification, generated tests are not guaranteed to be robust.

In the last decade, powerful SAT solvers based on Conjunctive Normal Form (CNF) (see for instance [8, 27–30]) have been developed. Unlike circuit-based SAT solvers, they cannot apply structural knowledge about the circuit-oriented problem natively. Based on the homogeneity of the CNF, powerful conflict analysis and fast Boolean Constraint Propagation (BCP) are utilized. Moreover, CNF-based SAT solvers[1] are more flexible in use. Additional constraints, e.g. modeling of static values, can be incorporated more easily. The efficiency and robustness of CNF-based SAT solvers in the field of ATPG was already shown in [31,4,5,32].

1.2 Contributions

In this article, the problem of generating robust and non-robust tests for PDFs in an industrial environment is targeted. In an industrial environment, modeling only Boolean values is not sufficient. The additional values U (unknown) and Z (high impedance) have to be considered, too. Furthermore, for robust tests, static values have to be guaranteed. The main contributions of this article are:

- Multiple-valued logics – A set of multiple-valued logics (and their Boolean encodings) is presented that allows to model static values and constraints occurring in industrial practice in CNF. As a result, powerful SAT solvers can be applied to industrial circuits.
- Structural analysis – Applying a multiple-valued logic results in a significant overhead for the SAT solver. Techniques that reduce the overhead and, by this, make the application of a SAT solver suitable for PDF test generation are shown. The concept of logic classes is presented to support a unified classification for PDF test generation with different quality. Experiments show that the structural analysis is mandatory for large circuits.
- Incremental SAT formulation – Typically, robust tests are preferred. However, for a robustly untestable fault, a non-robust test is generated. Using the proposed incremental SAT formulation, robust test generation benefits directly from the previous non-robust test generation since parts of the search space have already been traversed.

The techniques presented in this article were implemented as the SAT-based ATPG tool MONSOON. It is shown that MONSOON outperforms a state-of-the-art PDF ATPG tool by several factors on average. Further experiments on industrial circuits with more than three million gates are presented to show the feasibility and the robustness of the approach.

This article[2] is structured as follows. In Section 2, the basic knowledge about PDFs and SAT formulation

[1] In the following, the term *SAT solver* always means a *CNF-based SAT solver*.

[2] Parts of this article, i.e. the SAT formulation for multiple-valued logics, have been published in [33].

is introduced. SAT-based non-robust test generation is described in Section 3. The modeling of static values needed for robust test generation is presented in Section 4, whereas PDF test generation in an industrial environment is treated in Section 5. In Section 6, structural techniques to reduce the size of the SAT instance are studied, whereas in Section 7, the difficulties when applying incremental SAT to speed up test generation for non-robust and robust tests are explained and solved. Experimental results are presented in Section 8 and conclusions are drawn in Section 9.

2 SAT-based ATPG for Path Delay Faults

In this section, the basic concepts of SAT-based ATPG for PDFs are briefly reviewed. The PDF model is introduced in Section 2.1, whereas Section 2.2 explains how to formulate a circuit-oriented problem as a SAT problem.

2.1 Path Delay Fault Model

A PDF models a distributed delay on a path from a (pseudo) primary input to a (pseudo) primary output of a circuit C that exceeds the timing specification. Each structural path in C has two different types of PDFs; one with a rising transition and one with a falling transition. Formally, a PDF is defined by $F = (P, T)$, where $P = (g_1, \ldots, g_n)$ is a path from an input g_1 to an output g_n. The type of transition is given by $T \in \{R, F\}$, where R denotes a rising transition and F a falling transition.

To detect a PDF, two test vectors v_1, v_2 are needed to propagate the desired transition along the path P during two consecutive time frames t_1, t_2. The vector v_1 sets the initial value of T in the initial time frame t_1, whereas v_2 launches T in the final time frame t_2 at operating speed. If a delay fault occurs, the final value cannot be observed at g_n. If multiple delay faults are present, a test might not detect the fault because other delay faults may mask the targeted PDF.

Therefore, tests are classified into robust and non-robust tests[3]. A test is called robust iff it detects the fault independently of other delay faults in the circuit. Non-robust tests guarantee the detection of a fault, if there are no other delay faults in the circuit. Robust and non-robust tests differ in the constraints on the off-path inputs of P (sensitization criterion). Off-path inputs of P are inputs of gate $g_i, i \in 1 \ldots n$, which are not on P.

[3] For a detailed discussion about the classification of PDF tests, the reader is referred to [3].

Table 1 Off-path input constraints

	Non-Robust	Robust	
		rising	falling
AND/NAND	$X1$	$X1$	$S1$
OR/NOR	$X0$	$S0$	$X0$

The constraints are shown in Table 1. For a non-robust test, it is sufficient that the off-path inputs have a non-controlling value (ncv) in t_2 (denoted by $X0/X1$). Robust tests are more restrictive. If the on-path transition on g_i is from the ncv to the controlling value (cv) of g_{i+1}, the off-path inputs of g_{i+1} must have a static ncv (denoted by $S0/S1$). Thus, no fault effect can arrive at off-path inputs and the test is robust.

2.2 SAT Formulation

To apply a SAT solver to a circuit-oriented problem, the problem has to be formulated in CNF. A CNF Φ on n Boolean variables is a conjunction of m clauses. Each clause is a disjunction of literals. A literal is a Boolean variable in its positive (x) or negative form ($\overline{x}$). The CNF Φ is said to be *satisfied* iff all clauses are satisfied. A clause c is satisfied iff at least one literal in c is satisfied. The CNF Φ is said to be *unsatisfiable* iff Φ cannot be satisfied. The task of a SAT solver for a given Φ is to generate a solution that satisfies Φ or to prove that no such solution exists.

The CNF Φ_C for a circuit $C = (S, G)$ can be derived as follows, e.g. [34]. A Boolean variable x^s is assigned to each connection $s \in S$. Then, for each gate $g \in G$, the CNF Φ_g is derived from the characteristic function, which can be constructed using the truth table. The conjunction of the CNF of each gate results in the CNF of the circuit:

$$\Phi_C = \prod_{g_i \in G} \Phi_{g_i}$$

3 Non-Robust Test Generation

As described in Section 2.1, two time frames are needed for a non-robust test. Therefore, two Boolean variables x_1^s, x_2^s are assigned to each connection; each describing the value of s in the corresponding time frame. The CNF for each gate is duplicated using the respective variables resulting in the CNF Φ_{C1} for the initial time frame and Φ_{C2} for the final time frame. To guarantee the correct sequential behavior, additional constraints Φ_t describe the functionality of the flip-flops. These constraints guarantee the equivalence of the value of a pseudo primary output in t_1 and the value of the corresponding pseudo primary input in t_2. This results in

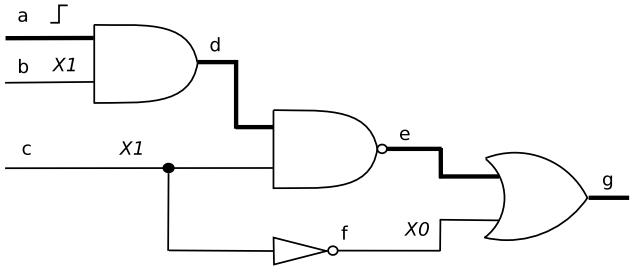


Fig. 1 Example circuit for path a – d – e – g

the following formula:

$$\Phi_{C_{NR}} = \Phi_{C1} \cdot \Phi_{C2} \cdot \Phi_t$$

Finally, the fault specific constraints are added. The fault specific constraints can be considered as fixed assignments to variables and are divided into two parts. The transition must be launched at g_1 (Φ_{tran}) and the off-path inputs of P must be assigned according to the non-robust sensitization criterion as given in Table 1 (denoted by Φ_o).

$$\Phi_P = \Phi_{C_{NR}} \cdot \Phi_{tran} \cdot \Phi_o$$

If Φ_P is satisfiable, P is a non-robustly testable path and the test can be created directly from the calculated solution.

Example 1 Consider the example circuit shown in Figure 1 with $P = (a, d, e, g)$ and $T = R$. The CNF of the circuit is as follows:

$$\Phi_{C1} = \Phi^d_{\mathrm{AND}_{t1}} \cdot \Phi^e_{\mathrm{NAND}_{t1}} \cdot \Phi^f_{\mathrm{NOT}_{t1}} \cdot \Phi^g_{\mathrm{OR}_{t1}}$$

$$\Phi_{C2} = \Phi^d_{\mathrm{AND}_{t2}} \cdot \Phi^e_{\mathrm{NAND}_{t2}} \cdot \Phi^f_{\mathrm{NOT}_{t2}} \cdot \Phi^g_{\mathrm{OR}_{t2}}$$

Because no flip-flops are contained in this circuit, the equation $\Phi_t = 1$ holds. The fault specific constraints for the rising transition are:

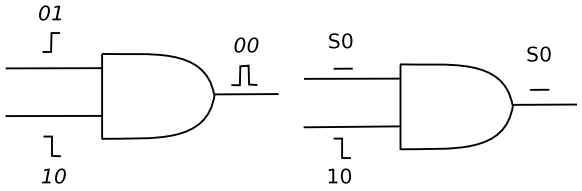
$$\Phi_{tran} = (\overline{a}_{t_1}) \cdot (a_{t_2}), \quad \Phi_o = (b_{t_2}) \cdot (c_{t_2}) \cdot (\overline{f}_{t_2})$$

A corresponding test given by the solution of the SAT solver could be:

$v_1 = \{a_{t_1} = 0, b_{t_1} = 0, c_{t_1} = 0\}$
$v_2 = \{a_{t_2} = 1, b_{t_2} = 1, c_{t_2} = 1\}$

4 Robust Test Generation

According to the robust sensitization criterion, static values have to be modeled. Therefore, Boolean values are not sufficient for robust test generation. Using only Boolean values, two discrete points of time t_1, t_2 are modeled, but no information about the transitions between t_1 and t_2 is given. The following example motivates the use of a multiple-valued logic.



(a) Boolean modeling (b) Explicit static values

Fig. 2 Guaranteeing static values

Table 2 Truth table for an AND gate in L_{6s}

AND	$S0$	00	01	10	11	$S1$
$S0$	$S0$	$S0$	$S0$	$S0$	$S0$	$S0$
00	$S0$	00	00	00	00	00
01	$S0$	00	01	00	01	01
10	$S0$	00	00	10	10	10
11	$S0$	00	01	10	11	11
$S1$	$S0$	00	01	10	11	$S1$

Example 2 Consider the AND gate in Figure 2. If the robust sensitization criterion requires that the output is set to $S0$, setting both output variables corresponding to the two time frames to 0 is insufficient.[4] Then, a rising and a falling transition on the inputs would satisfy the condition, because the controlling value is assumed in t_1, t_2 on different inputs as shown in Figure 2(a). However, if the inputs do not switch simultaneously, which cannot be guaranteed without timing information, a glitch could be produced on the output.

This case has to be excluded by explicitly modeling static values. This ensures that a static value on the output of a gate has its source in one or more static values on the inputs. This is shown at the AND gate in Figure 2(b).

Static values can be handled using the multiple-valued logic

$$L_{6s} = \{S0, 00, 01, 10, 11, S1\}.$$

The name of the value determines the signal's behavior in t_1 and t_2. The first position gives the value of the connection in t_1, whereas the second position describes the value in t_2. The values $S0, S1$ represent the static values. The truth table for an AND gate modeled in L_{6s} is presented in Table 2.

In order to apply a Boolean SAT solver to a problem formulated in multiple-valued logic, each value has to be encoded using Boolean variables. This encoding is not unique. A good encoding has to be found among all possibilities. A detailed discussion how to generate an efficient encoding is given in [35].

[4] According to their description, this simplification is done in [26].

Table 3 Boolean encoding $\eta_{L_{6s}}$ for L_{6s}

var	$S0$	00	01	10	11	$S1$
x_1	0	1	0	1	0	0
x_2	0	1	1	1	1	0
x_3	0	0	0	1	1	1

As encoding, a logarithmic encoding is used. The minimal number of Boolean variables n needed to encode a value depends on the number of values of a multiple-valued logic L_m and is calculated as follows: $n = \lceil \log_2 |L_m| \rceil$. As a result, three variables are needed to encode each value of L_{6s}. The Boolean encoding $\eta_{L_{6s}}$ for L_{6s} used in this article is shown in Table 3. For example, the connection c has three variables x_1^c, x_2^c, x_3^c. Hence, an assignment $\{x_1^c = 0, x_2^c = 0, x_3^c = 1\}$ is interpreted as the value $S1$ of L_{6s}.

The resulting CNF for a 2-input AND gate with inputs a, b and output c using $\eta_{L_{6s}}$ is presented in Table 4. The CNF template for each gate type can be created using a truth table and a logic minimizer, e.g. ESPRESSO [36]. This has to be done only once and the generated CNF templates can be reused for each circuit under test. The SAT formulation of the circuit using L_{6s} is similar to the SAT formulation described in Section 3. However, instead of two variables, three variables are assigned to each connection and the circuit CNF is derived from the CNF of each gate using $\eta_{L_{6s}}$. The robust sensitization criterion is modeled by fixing the corresponding assignments. Note, that there is no need to build the CNF for the complete circuit. For a specific PDF, only the fan-in cone of gates on the path has to be transformed into CNF using L_{6s}. If flip-flops are contained in this fan-in cone, the fan-in cone of these flip-flops has to be considered, too. By this, the sequential behavior is modeled adequately.

Let FF be the set of flip-flops contained in the fan-in cone of path P. For all gates located in the fan-in cone of at least one flip-flop in FF but not in the fan-in cone of P, only the value during t_1 is relevant. Therefore, these gates can be modeled in Boolean logic (one time frame) as described in Section 2.2. Consequently, only one variable is needed. Typically, the number of gates which can be modeled in Boolean logic is far higher than the number of gates which have to be modeled in a multiple-valued logic. If a predecessor f of such a gate g is modeled in L_{6s}, only x_3^f is used for Φ_g. As Table 3 shows, $\eta_{L_{6s}}$ was chosen such that the assignment of x_3 always determines the value of the connection in t_1. This guarantees the consistency of the circuit's function.

In contrast to the pure Boolean modeling of the circuit, using L_{6s} causes a significant overhead for the CNF size of the circuit. On the other hand, tests with a higher quality can be obtained and, as the experimental results in Section 8 below show, test generation for robust tests can be executed in reasonable time.

Table 4 CNF for an AND gate using $\eta_{L_{6s}}$

$(\overline{x}_1^a + x_1^c + \overline{x}_2^c)$	$(x_2^b + x_3^b + \overline{x}_2^c)$	$(\overline{x}_1^a + x_2^a)$
$(\overline{x}_1^b + x_1^c + \overline{x}_2^c)$	$(\overline{x}_2^a + \overline{x}_3^b + x_2^c)$	$(\overline{x}_1^b + x_2^b)$
$(\overline{x}_3^a + \overline{x}_3^b + x_3^c)$	$(\overline{x}_3^a + \overline{x}_2^b + x_2^c)$	$(x_3^b + \overline{x}_3^c)$
$(x_1^a + x_1^b + \overline{x}_1^c)$	$(\overline{x}_2^a + \overline{x}_2^b + x_2^c)$	$(x_3^a + \overline{x}_3^c)$
$(x_2^a + x_3^a + \overline{x}_2^c)$	$(x_2^a + x_2^b + \overline{x}_2^c)$	$(\overline{x}_1^c + x_2^c)$

5 Industrial Application

In this section, the problem of PDF test generation in industrial practice is considered. Additional constraints that have to be handled in industrial circuits are introduced and structural techniques to reduce the size of the SAT instance are presented.

5.1 Additional Values

For industrial applications, more requirements have to be met for PDF test generation. Besides the Boolean values, two additional values have to be considered. The value Z describes the state of high impedance and occurs for example in modeling busses. Gates that are able to assume Z are named tri-state gates. If a connection has a fixed value which is not known, the value U (unknown) is assumed. The unknown value can for instance occur if some flip-flops cannot be controlled. Then, the output of the flip-flop has always the value U, i.e. it is fixed to U.

A test generation algorithm has to be able to handle these additional values when it is applied in industrial practice. In [4], a four-valued logic $L = \{0, 1, U, Z\}$ used for modeling stuck-at faults in industrial circuits is presented. For PDF test generation, two time frames have to be considered; L is not sufficient. Therefore, the Cartesian product of all values in L is needed to represent all possible value combinations on a connection. For non-robust PDF test generation, this results in the 16-valued logic L_{16}:

$$L_{16} = \{00, 01, 10, 11, 0U, 1U, U0, U1, UU, 0Z, 1Z, Z0, Z1, UZ, ZU, ZZ\}$$

As described in Section 4, additional static values are needed for robust PDF test generation. Therefore, the 19-valued logic L_{19s} is required:

$$L_{19s} = \{S0, 00, 01, 10, 11, S1, 0U, 1U, U0, U1, UU, 0Z, 1Z, Z0, Z1, UZ, ZU, ZZ, SZ\}$$

Table 5 Derived logics of L_{19s} and L_{16}

L_{11s}	=	$\{S0, 00, 01, 10, 11, S1, 0U, 1U, U0, U1, UU\}$
L_{8s}	=	$\{S0, 00, 01, 10, 11, S1, 0U, 1U\}$
L_{6s}	=	$\{S0, 00, 01, 10, 11, S1\}$
L_9	=	$\{00, 01, 10, 11, 0U, 1U, U0, U1, UU\}$
L_6	=	$\{00, 01, 10, 11, 0U, 1U\}$
L_4	=	$\{00, 01, 10, 11\}$

Table 6 Mapping between logic class and applied logic

logic class	robust	non-robust
LC_Z	L_{19s}	L_{16}
LC_{U1}	L_{11s}	L_9
LC_{U2}	L_{8s}	L_6
LC_B	L_{6s}	L_4

The logic L_{19s} contains three additional static values: $S0, S1, SZ$. A static U value is meaningless, because the behavior of the signal is unknown.

In principle, L_{19s} can be used to model the circuit for robust PDF test generation. However, logics with less values are generally more compact in their CNF representation than logics with more values. The exclusive use of L_{19s} would result in excessively large SAT instances and typically in longer run time. This also holds for non-robust test generation using L_{16} exclusively. Fortunately, typically only a few connections in a circuit can assume all values contained in L_{19s} or in L_{16}. For example, there are only very few gates in a circuit that are able to assume the value Z.

Therefore, it is proposed to use not only one multiple-valued logic (e.g. L_{19s}) but a set of multiple-valued logics which are derived from L_{19s} (robust test generation) or L_{16} (non-robust test generation), respectively. These derived logics contain a smaller number of values, i.e. a subset of values. The idea behind this approach is that each gate is modeled using a logic that contains only those values which can be assumed by the input and output connections of the gate. Using this approach, the size of the CNF is reduced.

5.2 Logic Classes

In order to use a unified structural classification for non-robust and robust test generation, logic classes are introduced. Using logic classes allows for grouping gates independently from the desired quality of the test and independently from the multiple-valued logics used, respectively. As a result, the structural analysis can be applied for non-robust as well as for robust test generation once for a circuit as a pre-processing step.

All gates that can always assume the same set of values are grouped into one logic class and are modeled in the same logic. Four different logic classes are identified for the classification of gates:

$$LC_Z, LC_{U1}, LC_{U2}, LC_B$$

In the following, the properties of each logic class are described as well as the dedicated logics that are used. Note that for each logic class, two different logics can be used according to the desired quality of the test, i.e. non-robust or robust. The derived logics are presented in Table 5.

- LC_Z – A gate g belongs to LC_Z if all values of L can be assumed in t_1, t_2. Obviously, only tri-state gates belong to this class. As described above, for robust test generation, L_{19s} is used, whereas for non-robust test generation L_{16} is applied.
- LC_{U1} – A gate g belongs to LC_{U1} if the values $0, 1, U$ can be assumed in t_1, t_2, but not Z. These gates are modeled using the derivative logics L_{11s} (robust) and L_9 (non-robust):
- LC_{U2} – A gate g belongs to LC_{U2} if the values $0, 1$ can be assumed in t_1, t_2, whereas U can be assumed only in t_2. In other words, a gate is in LC_{U2} if a flip-flop in the fan-in cone of the gate is no source of unknowns but can be fed by a unknown from t_1. As a result, the value U can be propagated only in t_2. The corresponding logics are L_{8s} for robust test generation and L_6 for non-robust test generation.
- LC_B – A gate g belongs to LC_B if only $0, 1$ can be assumed in t_1, t_2. Then, L_{6s} (robust) and L_4 (non-robust) are applied. Note that these gates are modeled as described in Section 3 and Section 4.

A summary of the mapping between logic class and applied logic is given in Table 6. How to classify each gate is discussed in detail in Section 6.

5.3 Boolean Encoding

For the Boolean encoding of L_{19s}, five Boolean variables are needed to encode each value. The used encoding $\eta_{L_{19s}}$ which yields the best results in [35] is shown in Table 7. The encoding of the derived logics is implied by this encoding. For example, the encoding of the value $S1$ is $\{\overline{x}_1\overline{x}_2x_3x_4\overline{x}_5\}$. For L_{11s}, which needs only four variables, $S1$ is encoded by the first four variables, i.e. by $\{\overline{x}_1\overline{x}_2x_3x_4\}$, whereas for L_{6s} and L_{8s}, the value is encoded by $\{\overline{x}_1\overline{x}_2x_3\}$. For this reason, these encodings are said to be *compatible* with each other. The procedure for the encoding of L_{16} and derived logics is similar.

In Table 8, the impact of the different logics on the size of the CNF for two example gate types (AND and busdriver) is presented. The truth table of a busdriver

Table 7 Boolean encoding $\eta_{L_{19s}}$ for L_{19s}

var	$S0$	00	01	10	11	$S1$	0U	1U	U0	U1	UU	0Z	1Z	Z0	Z1	UZ	ZU	ZZ	SZ
x_1	0	1	0	1	0	0	1	1	1	0	1	0	1	0	1	1	0	0	1
x_2	0	1	1	1	1	0	0	0	1	0	0	1	0	0	0	1	0	0	1
x_3	0	0	0	1	1	1	0	1	1	1	1	1	1	1	0	1	0	0	0
x_4	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	1	0	0
x_5	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Table 8 Size of CNF for different logics

		AND		bus driver	
logic	#var	#cls	#lit	#cls	#lit
L_{19s}	5	-	-	114	561
L_{11s}	4	30	97	-	-
L_{8s}	3	21	71	-	-
L_{6s}	3	15	40	-	-
L_{16}	4	-	-	86	426
L_9	4	20	50	-	-
L_6	3	14	35	-	-
L_4	2	6	14	-	-

Table 9 Truth table of busdriver

control		0	1	Z	U
data	0	Z	0	U	U
	1	Z	1	U	U
	Z	Z	Z	Z	Z
	U	Z	U	U	U

for one time frame is shown in Table 9. In column *logic*, the logic used is named, whereas in column *#var*, the number of variables of the encoding is given. In columns entitled *#cls* the number of clauses for an AND gate and a busdriver is shown, respectively. Columns named *#lit* report the number of literals. The table entries are empty when the logic is not applicable for the gate type, for example, a Z is interpreted as U for an AND gate, while a busdriver can always assume the value Z so only L_{19s} and L_{16} apply.

The overhead of using a higher-valued logic is significant. Modeling as many gates as possible in a logic with fewer values is therefore desirable.

6 Structural Classification

In this section, the algorithm that determines the logic class of each gate is given. The classification is only executed once for each circuit in a pre-processing step. Therefore, the overhead is negligible. The overall goal of the classification is to classify as many gates as possible into logic class LC_B to allow for a compact circuit representation. The pseudo-code of the structural classification is described in Algorithm 1.

To begin, the tri-state gates are identified. Typically, the number of these gates is small compared to the number of Boolean gates. Because all of them may assume any value in $\{0, 1, U, Z\}$ in t_1, t_2, they are inserted into logic class LC_Z (line 4). All inputs that are fixed to an unknown state are also identified and inserted into logic class LC_{U1} because they can assume the value U in t_1, t_2 but not the value Z (line 5).

The next step is to determine the output cone of both, the tri-state gates and the fixed inputs. All gates which have been inserted into a logic class so far can be considered as sources of unknown values in the circuit. Note that a Z-value is interpreted as U in a Boolean gate. Consequently, each gate in each output cone of these gates can itself assume an unknown state in t_1, t_2. Therefore, they have to be inserted into logic class LC_{U1}. This is done by the while-loop in lines 9-21.

If an unknown value reaches a flip-flop in the initial time frame, it is propagated again in the final time frame (due to launch-on-capture). Therefore, these flip-flops that are inserted in LC_{U1} are temporary stored (lines 16-17). Once all elements of LC_{U1} are determined, the stored flip-flops are processed again by the while loop in lines 22-31, where the output cone of each flip-flop is determined. If a gate is not in LC_{U1} and LC_Z, the value U can only be assumed in t_2 but not in t_1. For that reason, it is inserted into LC_{U2}. The remaining gates cannot assume a non-Boolean value in t_1 or t_2. Therefore, they are inserted into LC_B (lines 32-26).

When, creating the SAT instance for a PDF, each gate has to be transformed into CNF. Depending on the desired quality, the logic class is mapped to a specific logic which have to be used for this gate to provide a CNF which is as small as possible. The mapping is summarized in Table 6.

One problem arising from the use of different logics in modeling a circuit is the handling of logic transitions. A logic transition occurs if at least one direct predecessor of gate g is modeled in a different logic than g. Due to the different Boolean encodings for the logics, inconsistencies would occur at g. Therefore, inputs of g modeled in a logic with fewer values have to be converted to the higher-valued logic of g. Inconsistencies are avoided by additional constraints. The following example demonstrates the procedure.

Example 3 Consider a busdriver b that is modeled in L_{19s}. The control input of b, named c, is modeled in L_{19s}, too. The corresponding variables are $x_1^b, x_2^b, x_3^b, x_4^b$,

Algorithm 1 Structural classification in logic classes

```
1: LogicClass LC_Z, LC_U1, LC_U2, LC_B = ∅
2: GateList l = ∅
3: GateList f = ∅
4: LC_Z.insert(all_tri_state_gates())
5: LC_U1.insert(all_inputs_fixed_to_unknown())
6: l.push(all_tri_state_gates())
7: l.push(all_inputs_fixed_to_unknown())
8: mark_as_seen(l.all())
9: while !l.empty() do
10:    Gate g = l.pop_first_element()
11:    for all succ ∈ g.all_successors() do
12:       if not_seen(succ) then
13:          mark_as_seen(succ)
14:          l.push(succ)
15:          LC_U1.insert(succ)
16:          if is_FlipFlop(succ) then
17:             f.push(succ)
18:          end if
19:       end if
20:    end for
21: end while
22: while !f.empty() do
23:    Gate g = f.pop_first_element()
24:    for all succ ∈ g.all_successors() do
25:       if not_seen(succ) then
26:          mark_as_seen(succ)
27:          f.push(succ)
28:          LC_U2.insert(succ)
29:       end if
30:    end for
31: end while
32: for all gate ∈ all_gates() do
33:    if not_seen(gate) then
34:       LC_B.insert(gate)
35:    end if
36: end for
```

x_5^b and $x_1^c, x_2^c, x_3^c, x_4^c, x_5^c$, respectively. For the data input of b, named d, L_{8s} is applied. The three corresponding variables are x_1^d, x_2^d, x_3^d. In order to obtain a consistent CNF, d is converted to L_{19s} and two additional variables x_4^d, x_5^d are assigned. Due to the compatible encoding of L_{19s} and L_{8s}, it is straight forward to restrict d to the values of L_{8s}. Table 7 shows that fixing x_4^d to 1 and fixing x_5^d to 0 is sufficient.

The above structural analysis significantly reduces the complexity of SAT-based PDF test generation for industrial circuits.

7 Incremental SAT

Generating robust tests for PDFs is desirable. Unfortunately, typically only few paths in a circuit are robustly testable. For those paths which are not robustly testable, a non-robust test is generated (if one exists).[5]

[5] This procedure can be directly extended to the functional sensitization criterion. However, this article is restricted to the non-robust and robust sensitization criterion.

Table 10 CNF sizes of incremental SAT formulation

logic class	$\eta_{L_{16}}$		Φ_{Static}		$\eta_{L_{16}} + \Phi_{Static}$			$\eta_{L_{19s}}$		
	#cls	#lit	#cls	#lit	#v	#cls	#lit	#v	#cls	#lit
AND										
LC_{U1}	20	50	18	58	5	38	108	4	30	97
LC_{U2}	14	35	16	52	4	30	87	3	21	71
LC_B	6	14	9	31	3	15	45	3	15	40
busdriver										
LC_Z	86	426	25	81	5	111	507	5	114	561

The approach considered so far in Section 5 required two independent SAT instances for both types of tests. Both instances are optimized either for non-robust or for robust test generation. A SAT instance built for non-robust test generation is not suitable for robust test generation, because static values are not modeled. On the other hand, a SAT instance built for robust test generation can generally be used for non-robust test generation but causes too much overhead for non-robust test generation.

The fact that robust as well as non-robust test generation is executed sequentially can be exploited by using incremental SAT. The use of incremental SAT is not new in the field of ATPG for PDFs. However, in previous works, e.g [20], path segments are added incrementally to speed up test generation for non-robust tests considering all paths or a large number of overlapping paths. In this work, a new incremental SAT formulation which is based on complete and also independent paths is proposed which is more typical in practice. Here, the encoding of static values is added incrementally to the SAT instance. Consequently, robust test generation directly benefits from the previous non-robust test generation.

The application of this incremental formulation is as follows. At first, a SAT instance Φ_{NR-p} for non-robust test generation is built for path p. If it is unsatisfiable, p is non-robustly untestable and, consequently, robustly untestable. If p is non-robustly testable, a SAT instance Φ_{R-p} for robust test generation is built. The SAT instance Φ_{R-p} is composed according to the following equation:

$$\Phi_{R-p} = \Phi_{NR-p} \cdot \Phi_{static}$$

The CNF Φ_{static} describes the static value justification of p. That means the separate modeling of static values in contrast to the logic modeling given by Φ_{NR-p}. Incrementally adding Φ_{static} to Φ_{NR-p} results in a SAT instance suitable for robust test generation and provides the following advantages.

- Build time – Instead of building a completely new SAT instance for robust tests, execution time is saved by reusing the existing SAT instance Φ_{NR-p} .

Table 11 Boolean encoding $\eta_{L_{16}}$ for L_{16}

var	00	01	10	11	0U	1U	U0	U1	UU	0Z	1Z	Z0	Z1	UZ	ZU	ZZ
x_1	0	0	1	1	0	1	0	0	1	1	0	0	0	1	1	1
x_2	0	1	0	1	0	0	0	1	1	1	0	1	1	0	0	1
x_3	0	0	0	0	1	1	0	1	1	1	1	0	1	1	0	0
x_4	0	0	0	0	1	1	1	1	1	0	0	1	0	0	1	1

– Learned information – Conflict clauses created during non-robust test generation can be reused during robust test generation. This prunes large parts of the search space.
– Structural information – According to the robust sensitization criterion, not all off-path inputs of p have to be guaranteed to be static. Therefore, some parts of the circuit do not have to be included in Φ_{static}.

In the following, a description how to derive Φ_{static} is given. At first, an additional variable x_S is assigned to each connection. This variable determines whether the signal on the connection is static ($x_S = 1$). If a static signal has to be forced on an off-path input g, x_S^g is fixed to 1. In order to justify this value, additional implications are added for each gate in the fan-in cone of g. For gate g with direct predecessors $h_1, ..., h_n$, these are as follows. If the non-controlling value ncv is on the output of g, all direct predecessors $h_1, ..., h_n$ have to be statically non-controlling between both time frames, too:

$$(x_S^g = 1 \wedge g = ncv) \rightarrow \prod_{i=1}^{n} x_S^{h_i} = 1 \quad (1)$$

If the controlling value cv is on the output of g, at least one predecessor h_i has to be statically controlling:

$$(x_S^g = 1 \wedge g = cv) \rightarrow \sum_{i=1}^{n} (x_S^{h_i} = 1 \cdot h_i = cv) \quad (2)$$

Thus, it is guaranteed, that a static value on an off-path input is justified. These additional implications are transformed into CNF according to the logic used and corresponding encoding of g. However, the minimal size of the CNF is given up for robust test generation by directly encoding the implications into CNF. The CNF size using incremental SAT is generally larger than using $\eta_{L_{19_s}}$ and more variables are needed.

Table 10 shows the effect on the CNF size for an AND gate and a busdriver. Column $\eta_{L_{16}}$ gives the CNF size for non-robust test generation (using the Boolean encoding shown in Table 11) and column Φ_{Static} presents the CNF size for the additional implications for static value justification. In column $\eta_{L_{16}} + \Phi_{Static}$, the total CNF sizes for robust test generation with incremental SAT is presented. For comparison, the CNF sizes for robust test generation using $\eta_{L_{19_s}}$ is repeated in the last column.

Table 12 CNF for an AND gate using η_{L_4}

$(\overline{x}_1^a + \overline{x}_1^b + x_1^c)$	$(x_1^a + \overline{x}_1^c)$	$(x_1^b + \overline{x}_1^c)$
$(\overline{x}_2^a + \overline{x}_2^b + x_2^c)$	$(x_2^a + \overline{x}_2^c)$	$(x_2^b + \overline{x}_2^c)$

Table 13 CNF description for static value justification for an AND gate using η_{L_4}

$(\overline{x}_2^a + x_1^b + \overline{x}_2^b + \overline{x}_S^c)$	$(x_1^a + \overline{x}_2^a + \overline{x}_1^b + \overline{x}_S^c)$	$(x_S^a + \overline{x}_2^b + \overline{x}_S^c)$
$(\overline{x}_1^a + \overline{x}_1^b + x_2^b + \overline{x}_S^c)$	$(x_S^a + \overline{x}_1^b + \overline{x}_S^c)$	$(\overline{x}_2^a + x_S^b + \overline{x}_S^c)$
$(\overline{x}_1^a + x_2^a + \overline{x}_2^b + \overline{x}_S^c)$	$(\overline{x}_1^a + x_S^b + \overline{x}_S^c)$	$(x_S^a + x_S^b + \overline{x}_S^c)$

As mentioned above, not all gates have to be included in Φ_{static}. For example, if a rising transition occurs at the on-path input of an AND gate under the robust sensitization criterion, the off-path inputs – and consequently their fan-in cone – do not have to be considered. This is because according to the sensitization criterion, no static value has to be guaranteed here. Let G_S^F be the set of gates on which static values have to be guaranteed for the path delay fault F. Then, only those gates that are located in the fan-in cone of at least one gate $g \in G_S^F$ are included in Φ_{static}. As a result, the size of the CNF for robust test generation can be reduced.

Example 4 Consider an AND gate $c = a \cdot b$ with inputs a, b. The CNF that models an AND gate under L_4 is shown in Table 12. Given the additional variables x_S^a, x_S^b, x_S^c, the implications Φ_{Static} described in Equation 1 and Equation 2 are presented in CNF in Table 13. Each clause represents an illegal assignment of the variables. For example, the clause $(\overline{x}_1^a + x_S^b + \overline{x}_S^c)$ implies that if a static value has to be justified at line c ($x_S^c = 1$) and the signal on line b is not guaranteed to be static ($x_S^b = 0$), line a has to assume the value 1 in the first time frame ($x_1^a = 1$); see the Boolean encoding given in Table 11.

Note, that each of these additional clauses contains the literal $\overline{x}_S^c$. This means, that these clauses are only to be considered if $x_S^c = 1$, i.e. a static value has to be justified at this line. Otherwise, if $x_S^c = 0$, all clauses are satisfied by this assignment and no implication is derived.

This incremental approach requires some overhead when modeling a circuit, as the most compact encoding cannot always be chosen. Instead, structural information about static values is added to the SAT instance for

Table 14 Run time comparison with competitive approach – non-robust

circ	KF-ATPG	MONSOON	impr
s5378	0:22m	0:03m	7.3
s9234	0:10m	0:04m	2.5
s13207	0:15m	0:10m	1.5
s15850	0:20m	0:08m	2.5
s35932	0:05m	0:01m	5.0
s38417	0:33m	0:06m	5.5
s38584	0:21m	0:08m	2.6
b14	1:39m	0:26m	3.8
b15	2:11m	0:30m	4.4
b17	2:18m	0:27m	5.1
b18	3:29m	1:50m	1.9
b20	2:34m	1:23m	1.9
b21	2:42m	1:30m	1.8
b22	3:47m	1:57m	1.9
average			3.4

non-robust generation. Additionally, as an advantage, learned information can be reused.

8 Experimental Results

In this section, experimental results are presented. The algorithms were implemented in C++ as the tool MONSOON. The tool MONSOON was integrated into the ATPG framework of NXP Semiconductors as a prototype. Experiments were performed on an AMD Dual Opteron 2222 (3.0 GHz, 65,536 MB, GNU/Linux). As SAT solver, MiniSat v1.14 [28] is used. The 40,000 longest paths of each circuit were chosen for PDF test generation. The fixed delay model was applied for path extraction. For sake of simplicity, each gate has a fixed delay of 1. For each targeted PDF, a broadside test where the first test vector can be fully scanned is generated.

The experiments are divided in three parts. Section 8.1 shows a run time comparison with a competitive approach on Boolean circuits. In Section 8.2, the industrial circuits are introduced and the benefits of the structural analysis are evaluated. Finally, experimental results for non-robust and robust test generation are shown in Section 8.3. Here, also the advantages of the incremental SAT formulation are presented.

8.1 Comparison with Competitive Approach

First of all, the efficiency of our CNF-based approach for non-robust PDF test generation is shown. Table 14 shows a run time comparison with KF-ATPG [25] – a state-of-the-art circuit-SAT based Path-Delay-Fault ATPG – which uses structural ATPG techniques. Furthermore, it maintains a path-status graph and such works in an incremental manner. Since KF-ATPG is only able to generate non-robust tests for Boolean circuits, the run time comparison is performed only for this type of tests.[6] As benchmarks, larger circuits from the ISCAS'89 as well as from the ITC'99 benchmark suite are taken. Column *circ* shows the circuit's name. Run time is given in CPU minutes. Results for KF-ATPG are presented in column *KF-ATPG*, whereas results for our approach are given in column *MONSOON*. The factor of improvement of MONSOON is shown in column *impr*. MONSOON clearly outperforms KF-ATPG by a factor of up to 7.3. The average factor of improvement is 3.4.

Table 15 Information about the industrial circuits

circ	#PIs	#POs	#FFs	LC_Z	LC_{U1}	LC_{U2}	LC_B
p44k	739	56	2,175	0	0	0	100
p57k	8	19	2,291	<0.1	0.2	25.7	74.0
p80k	152	75	3,878	0	0	0	100
p88k	403	183	4,302	0.6	7.3	21.4	70.7
p99k	167	82	5,747	0	1.5	5.6	92.9
p177k	768	1	10,507	0.8	27.4	54.5	17.3
p456k	1,723	72	14,900	3.3	20.4	75.7	0.6
p462k	1,815	604	29,205	0.2	13.6	34.7	51.5
p565k	996	201	32,409	6.5	7.3	59.0	27.2
p1330k	617	90	104,630	<0.1	9.0	15.0	75.9
p2787k	46,015	274	58,835	0.3	39.4	25.5	34.8
p3327k	4,093	274	148,184	3.5	16.1	80.2	0.2
p3852k	6,052	303	173,738	1.5	15.6	82.5	0.4

8.2 Structural Analysis

Industrial circuits provided by NXP Semiconductors GmbH, Hamburg, Germany are considered for the further experiments. All of these circuits are considered as hard-to-test. Therefore, a timeout is used for each path to limit the overall run time. If a test for a path could not be solved within 10 MiniSat restarts[7], the path is listed as aborted or unclassified. The general timeout for one circuit is set to 72 CPU hours.

Statistical information about the industrial circuits is provided in Table 15. The first column gives the circuit's name. The name denotes roughly the size of the circuit, e.g. p3852k contains over 3.8 million elements. Besides Boolean gates, the industrial circuits contain primitives such as MUX gates. In column #PIs and #POs the number of primary inputs and primary outputs are given, respectively, whereas column #FFs presents the number of flip-flops. The last four columns

[6] To the best of our knowledge, no free PDF test generator that can generate robust tests and/or can handle static values and industrial constraints is available for comparison.

[7] A restart is defined as a certain number of conflicts (100 at the beginning). After each restart, the number of conflicts is increased (by 50%).

Table 16 Run time comparison with and without structural analysis – non-robust

	MONSOON – Non-Robust										
	w/o analysis					with analysis					
circ	#vars	#cls	ab.	time	bld	#vars	#cls	ab.	time	bld	impr
p44k	131,494	479,817	12,799	$> 72h$	1:23h	62,429	155,466	0	1:35h	41:38m	45.5
p57k	71,069	259,606	1,259	3:52h	42:28m	34,908	93,828	7	46:00m	19:06m	5.0
p80k	44,556	167,459	13,844	20:33h	39:18m	17,870	49,526	16	49:25m	15:13m	25.2
p88k	22,950	81,055	0	40:08m	9:16m	11,099	27,618	0	6:39m	4:02m	6.0
p99k	11,699	43,150	5	22:59m	5:23m	5,675	14,380	0	5:30m	2:14m	4.2
p177k	120,702	432,912	9,338	39:46h	2:45h	81,972	251,580	8,274	39:22h	1:49h	1.0
p456k	34,644	136,304	1,107	3:34h	55:52m	24,872	86,077	652	2:09h	39:04m	1.7
p462k	22,428	78,172	0	1:00h	33:53m	14,983	45,590	0	33:06m	22:39m	1.8
p565k	8,338	33,277	293	47:40m	14:40m	5,497	17,955	1	25:03m	8:41m	1.9
p1330k	31,448	101,800	0	1:05h	42:26m	27,727	85,049	0	58:14m	37:19m	1.1
p2787k	91,873	328,164	613	5:07h	2:20h	76,700	250,263	473	4:08h	1:53h	1.2
p3327k	57,154	187,002	665	8:11h	1:26h	33,089	91,337	94	4:31h	52:51m	1.8
p3852k	101,761	334,016	2,855	19:14h	2:25h	59,009	162,237	865	9:10h	1:25h	2.1
			42,778		14:22h			10,392		9:10h	7.6

show the results of the structural pre-processing of the circuit. In each of these columns, the percentage of gates contained in the identified logic class is presented. Note that the number of gates in logic class LC_{U2} is larger than the number of gates in LC_{U1}, because unknown values reaching a flip-flop after the first time frame are propagated again in the second time frame. The large circuits especially contain a large number of gates which are not in the Boolean logic class LC_B. For example, only 0.2% of all gates in p3327k can be modeled in Boolean logic. On the other hand, the percentage of gates in LC_Z is also very small.

In order to show the benefit of the structural analysis, a run time comparison of non-robust generation with and without structural analysis is given in the following. We also made experiments for robust test generation. However, the effect was similar. Therefore, the results are not reported here. Table 16 presents the run time results for non-robust PDF test generation. The approach without incorporating the analysis (column *w/o analysis*) handles all gates in the highest-valued logic L_9/L_{16}. Columns named *#vars* give the average number of variables contained in the SAT instance. In columns *#cls*, the average number of clauses is presented. The number of aborts, i.e. PDFs which could not be classified in the given interval, is shown in column *ab.* and the run time in CPU minutes (m) or CPU hours (h) is listed in columns named *time*. Columns *bld* give the total run time spent for building the SAT instances.

The results clearly show that the structural analysis is mandatory for efficient SAT-based PDF test generation. The highest speed-up factor is greater than 45x for non-robust test generation (p44k) and the average speed-up factor is 7.6x. Parts of the improvement stems from run time savings during CNF generation. The CNF is larger without structural analysis which influences the run time needed to build the SAT instances. Without structural analysis all gates are conservatively modeled as belonging to LC_Z/LC_{U1} – even gates belonging to LC_B. The structural analysis unveils this overhead and gates are modeled more compactly. This is directly reflected by the overall size of the CNFs. With structural analysis, gates in LC_B are modeled by less clauses than gates in LC_Z/LC_{U1}. The SAT instances are therefore generally more easy to solve. The number of unclassified faults also significantly decreases by using the structural analysis. The portion of unclassified PDFs is only about 24%.

8.3 Different Configurations

Next, five different PDF test generation configurations are evaluated:

- Non-Robust – Only non-robust test generation using L_{16} and derived logics is performed.
- Robust – Only robust test generation using L_{19s} and derived logics is performed.
- Sequential – At first, non-robust test generation using L_{16} and derived logics is performed. If the PDF is non-robustly testable, robust test generation is executed. Here, a completely new SAT instance is built using L_{19s} and derived logics. This configuration is the typical flow used to generate a test with highest quality if no incremental approach is available.
- Incremental – Similar to the sequential configuration. However, if the PDF is non-robustly testable, robust test generation is performed using the incremental SAT formulation. All learned information is kept. By this, robust test generation benefits from the previous non-robust test generation.

Table 17 Average CNF sizes for each configuration

	Non-Robust		Robust		Sequential		Incremental		Incr. (Rob.)	
circ	#vars	#cls	#vars	#cls	#vars	#cls	#vars	#cls	#vars	#cls
p44k	62,429	155,466	83,751	304,889	79,898	277,887	85,212	257,278	90,229	279,676
p57k	34,908	93,828	38,241	130,065	35,835	104,666	36,468	102,749	40,157	122,184
p80k	17,870	49,562	20,724	73,356	19,831	65,962	20,129	61,122	21,257	66,704
p88k	11,099	27,618	12,765	41,134	11,903	34,222	12,191	32,379	13,245	36,936
p99k	5,675	14,380	6,875	24,078	6,410	20,161	6,693	19,000	7,330	22,092
p177k	81,972	251,580	82,623	307,409	81,991	253,060	82,261	253,383	93,782	326,946
p456k	24,782	86,077	24,815	118,519	24,789	91,165	25,899	93,719	31,081	132,433
p462k	14,983	45,590	15,671	55,857	15,176	47,030	15,262	46,747	17,379	57,856
p565k	5,497	17,955	5,649	25,602	5,510	19,573	5,855	20,341	7,211	28,747
p1330k	27,727	85,049	28,002	110,745	27,771	85,560	27,819	85,422	31,698	106,243
p2787k	76,700	250,263	77,475	291,814	76,709	250,717	76,807	250,888	82,085	288,757
p3327k	33,089	91,337	33,187	105,472	33,124	94,769	34,425	96,951	38,283	113,297
p3852k	59,009	162,237	59,049	189,187	59,021	165,461	60,248	167,442	68,837	207,156

Table 18 Experimental results for PDF test generation with MONSOON

		Non-Robust		Robust			Sequential			Incremental		
circ	#nr	ab.	time	#rob	ab.	time	#rob	ab.(r)	time	#rob	ab.(r)	time
p44k	19,152	0	1:35h	7,400	0	2:02h	+0	0	3:15h	+0	0	4:48h
p57k	4,149	7	46:00m	1,509	898	1:40h	+0	896	2:04h	+884	0	59:51m
p80k	15,175	16	49:25m	401	901	1:01h	+0	885	1:32h	+750	2	59:04m
p88k	5,851	0	6:39m	1,606	0	8:30m	+0	0	11:09m	+0	0	8:32m
p99k	7,095	0	5:30m	702	1	6:12m	+0	0	8:53m	+0	0	6:50m
p177k	1,195	8,274	39:22h	846	3,042	10:17h	-588	30	39:47h	-558	0	39:37h
p456k	7,330	652	2:09h	1,214	164	1:32h	-30	50	2:38h	-22	0	2:16h
p462k	8,855	0	33:06m	2,602	0	39:49m	+0	0	35:52m	+0	0	36:26m
p565k	10,033	1	25:03m	2,340	78	27:53m	+0	59	34:49m	+33	0	28:11m
p1330k	8,473	0	58:14m	6,740	0	1:09h	+0	0	56:51m	+0	0	59:29m
p2787k	1,949	473	4:08h	411	55	3:13h	-1	4	4:13h	+3	0	4:10h
p3327k	18,284	94	4:31h	6,251	23	1:34h	-2	23	5:03h	+17	0	4:37h
p3852k	11,413	865	9:10h	3,411	72	4:14h	-131	3	10:21h	-132	4	9:41h
total		10,382			5,234		-752	1,950		+869	6	

– Incr. (Rob.) – Here, robust test generation only is performed with the incremental SAT formulation. In contrast to the incremental configuration, only one SAT instance is solved.

Table 17 shows the average CNF sizes for each configuration. As expected, the number of variables and clauses is higher for the robust configuration. More variables and clauses are needed for the additional constraints. Again, as expected, the average number of variables and clauses of the sequential configuration lies between the non-robust and robust configuration, because non-robustly untestable PDFs are also robustly untestable and no SAT instance for a robust test has to be built. The incremental configuration needs constantly more variables than the sequential configuration but in most cases less clauses. The same effect is observed when comparing the average CNF size of the incremental (robust only) and the robust configuration.

Table 18 presents the experimental results for PDF test generation. The first column *circ* gives the name of the circuit. Column *#nr* (*#rob*) shows the number of non-robustly (robustly) testable paths. The results for each configuration are shown in the respective column. In the last two configurations (sequential and incremental), non-robust test generation is performed first (as presented in column *Non-robust*) followed by robust test generation if needed. Here, column *#rob* shows the number of robust test pattern compared to the robust configuration. The presented number of aborts in column *ab.(r)* gives the number of unclassified robust PDFs. For example, p3852k has 865 unclassified non-robust PDFs. For these aborted faults, robust test generation is not performed.

Although being typically more complex, in some cases robust test generation is much faster – up to a factor of 2.9 (p3327k) – than non-robust test generation. This can be explained by the decreased fault coverage. As expected, the sequential configuration needs more run time than the non-robust configuration, since non-robust test generation is performed for each fault at first. However, the sequential configuration generates fewer robust tests than the robust generation when the number of non-robust aborts is high – especially for p177k, p456k and p3852k. This is due to the large num-

Table 19 Comparison between robust and incremental configuration

	Robust			Incr. (Rob.)			impr
circ	ab.	time	%bld	ab.	time	%bld	
p44k	0	2:02h	73%	0	2:39h	56%	1.30
p57k	898	1:40h	26%	2	53:28m	54%	0.53
p80k	901	1:01h	41%	10	36:10m	78%	0.59
p88k	0	8:30m	82%	0	7:55m	91%	0.93
p99k	1	6:12m	78%	0	5:44m	91%	0.92
p177k	3,042	10:17h	23%	2,959	17:39h	14%	1.72
p456k	164	1:32h	67%	61	1:27h	88%	0.95
p462k	0	39:49m	96%	0	40:54m	98%	1.03
p565k	78	27:53m	64%	2	26:37m	82%	0.95
p1330k	0	1:09h	95%	0	1:12h	94%	1.04
p2787k	55	3:13h	85%	1	3:04h	98%	0.95
p3327k	23	1:34h	77%	12	1:52h	78%	1.19
p3852k	72	4:14h	42%	223	4:31h	46%	1.07
total	5,234		65%	3,270		74%	1.01

ber of aborts for which robust test generation is not performed.

The incremental configuration is faster than the sequential configuration (except for p44k) and nearly as fast as the non-robust configuration. As a result, the overhead of generating a robust test on top of a non-robust test is very small. At the same time, due to the learned information and structural knowledge, very few PDFs could not be classified robustly. As a result, the total number of robust tests significantly increases compared to the sequential configuration and even compared to the robust configuration. Therefore, if non-robust and robust test generation are to be executed, the incremental configuration is the best choice.

Table 19 shows a direct comparison between robust and incremental configuration for generating robust tests only. The SAT solver is called for the incremental configuration (column *Incr. (Rob.)*) only once and no learned clauses from a previous non-robust test generation are available. The improvement of the robust configuration over the incremental (robust only) configuration is shown in column *impr*.

Although the SAT instances are larger using the incremental configuration (see Table 10), the run time is comparable on average. However, the incremental configuration produces less aborts than the robust configuration even without the information learned from non-robust test generation. The reason for this is that the explicit encoding of static values supports the reasoning engine. Therefore, the incremental configuration is preferable.

9 Conclusions

In this article, MONSOON, a SAT-based approach for generating non-robust and robust tests for PDFs in an industrial environment is presented. For modeling static values as well as industrial constraints, a set of multiple-valued logics and the transformation to Boolean SAT was described. Furthermore, to reduce the complexity of PDF test generation in industrial circuits, a structural analysis is shown to be mandatory. The concept of logic classes to support a unified structural classification for test generation with different quality is introduced. In addition, to exploit the similarity of non-robust and robust PDF test generation, an incremental SAT formulation was presented which further reduces the number of aborts. A comparison with a state-of-the-art PDF ATPG tool yields a significant speed-up. Further experiments on large industrial circuits with over 3.8 million elements show the scalability and the robustness of MONSOON.

Acknowledgements This 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 by the German Research Foundation (DFG) under contract number DR 287/15-1.

References

1. Smith, G.L.: Model for delay faults based upon paths. In: Proceedings of the International Test Conference. (1985) 342–349
2. Lin, C.J., Reddy, S.M.: On delay fault testing in logic circuits. IEEE Transactions on Computer-Aided Design for Circuits and Systems **6** (1987) 694–703
3. Krstić, A., Cheng, K.T.: Delay Fault Testing for VLSI Circuits. Kluwer Academic Publishers, Boston, MA (1998)
4. Drechsler, R., Eggersglüß, S., Fey, G., Glowatz, A., Hapke, F., Schloeffel, J., Tille, D.: On acceleration of SAT-based ATPG for industrial designs. IEEE Transactions on Computer-Aided Design for Circuits and Systems **27** (2008) 1329–1333
5. Drechsler, R., Eggersglüß, S., Fey, G., Tille, D.: Test Pattern Generation using Boolean Proof Engines. Springer (2009)
6. Schulz, M.H., Auth, E.: Improved deterministic test pattern generation with applications to redundancy identification. IEEE Transactions on Computer-Aided Design for Circuits and Systems **8** (1989) 811–816
7. Qiu, W., Walker, D.M.H.: An efficient algorithm for finding the K longest testable paths through each gate in a combinational circuit. In: Proceedings of the International Test Conference. (2003) 592–601
8. Marques-Silva, J.P., Sakallah, K.A.: GRASP: A search algorithm for propositional satisfiability. IEEE Transactions on Computers **48** (1999) 506–521
9. Marques-Silva, J.P., Sakallah, K.A.: Robust search algorithms for test pattern generation. In: Proceedings of the International Symposium on Fault-Tolerant Computing. (1997) 152–157
10. Fey, G., Warode, T., Drechsler, R.: Reusing learned information in SAT-based ATPG. In: Proceedings of the International Conference on VLSI Design. (2007) 69–76
11. Fuchs, K., Fink, F., Schulz, M.H.: DYNAMITE: an efficient automatic test pattern generation system for path delay faults. IEEE Transactions on Computer-Aided Design for Circuits and Systems **10** (1991) 1323–1335

12. Wang, L.C., Liou, J.J., Cheng, K.T.: Critical path selection for delay fault testing based upon a statistical timing model. IEEE Transactions on Computer-Aided Design for Circuits and Systems **23** (2004) 1550–1565
13. Huang, I.D., Gupta, S.K.: Selection of paths for delay testing. In: Proceedings of the IEEE Asian Test Symposium. (2005) 208–215
14. Fuchs, K., Pabst, M., Rössel, T.: RESIST: a recursive test pattern generation algorithm for path delay faults considering various test classes. IEEE Transactions on Computer-Aided Design for Circuits and Systems **13** (1994) 1550–1562
15. Pomeranz, I., Reddy, S.M., Uppaluri, P.: NEST: a nonenumerative test generation method for path delay faults in combinational circuits. IEEE Transactions on Computer-Aided Design for Circuits and Systems **14** (1995) 1505–1515
16. Tragoudas, S., Karayiannis, D.: A fast nonenumerative automatic test pattern generator for path delay faults. IEEE Transactions on Computer-Aided Design for Circuits and Systems **18** (1999) 1050–1057
17. Drechsler, R.: BiTeS: a BDD based test pattern generator for strong robust path delay faults. In: Proceedings of the European Conference on Design Automation. (1994) 322–327
18. Christou, K., Michael, M.K., Tragoudas, S.: On the use of ZBDDs for implicit and compact critical path delay fault test generation. Journal of Electronic Testing: Theory and Applications **24** (2008) 203–222
19. Chen, C., Gupta, S.K.: A satisfiability-based test generator for path delay faults in combinational circuits. In: Proceedings of the Design Automation Conference. (1996) 209–214
20. Kim, J., Whittemore, J., Marques-Silva, J.P., Sakallah, K.A.: On applying incremental satisfiability to delay fault testing. In: Proceedings of Design, Automation and Test in Europe. (2000) 380–384
21. Chandrasekar, K., Hsiao, M.S.: Integration of learning techniques into incremental satisfiability for efficient path-delay fault test generation. In: Proceedings of Design, Automation and Test in Europe. (2005) 1002–1007
22. Hooker, J.N.: Solving the incremental satisfiability problem. Journal of Logic Programming **15** (1993) 177–186
23. Shtrichman, O.: Pruning techniques for the SAT-based bounded model checking problem. In: Proceedings of the Correct Hardware Design and Verification Methods. Volume 2144 of Lecture Notes in Computer Science. (2001) 58–70
24. Lu, F., Wang, L.C., Cheng, K.T., Huang, R.: A circuit SAT solver with signal correlation guided learning. In: Proceedings of Design, Automation and Test in Europe. (2003) 892–897
25. Yang, K., Cheng, K.T., Wang, L.C.: Trangen: a SAT-based ATPG for path-oriented transition faults. In: Proceedings of the ASP Design Automation Conference. (2004) 92–97
26. Lu, S.Y., Hsieh, M.T., Liou, J.J.: An efficient SAT-based path delay fault ATPG with an unified sensitization model. In: Proceedings of the International Test Conference. (2007)
27. Moskewicz, M.W., Madigan, C.F., Zhao, Y., Zhang, L., Malik, S.: Chaff: Engineering an efficient SAT solver. In: Proceedings of the Design Automation Conference. (2001) 530–535
28. Eén, N., Sörensson, N.: An extensible SAT solver. In: Proceedings of the International Conference on Theory and Applications of Satisfiability Testing. Volume 2919 of Lecture Notes in Computer Science. (2004) 502–518
29. Goldberg, E., Novikov, Y.: BerkMin: A fast and robust satsolver. Discrete Applied Mathematics **155** (2007) 1549–1561
30. Biere, A.: PicoSAT essentials. Journal of Satisfiability, Boolean Modeling and Computation **4** (2008) 75–97
31. Czutro, A., Polian, I., Lewis, M., Engelke, P., Reddy, S.M., Becker, B.: TIGUAN: Thread-parallel integrated test pattern generator utilizing satisfiability analysis. In: Proceedings of the International Conference on VLSI Design. (2009) 227–232
32. Eggersglüß, S., Drechsler, R.: Increasing robustness of SAT-based delay test generation using efficient dynamic learning techniques. In: Proceedings of the IEEE European Test Symposium. (2009)
33. Eggersglüß, S., Fey, G., Drechsler, R., Glowatz, A., Hapke, F., Schloeffel, J.: Combining multi-valued logics in SAT-based ATPG for path delay faults. In: Proceedings of the ACM & IEEE International Conference on Formal Methods and Models for Codesign. (2007) 181–187
34. Larrabee, T.: Test pattern generation using Boolean satisfiability. IEEE Transactions on Computer-Aided Design for Circuits and Systems **11** (1992) 4–15
35. Eggersglüß, S., Drechsler, R.: On the influence of Boolean encodings in SAT-based ATPG for path delay faults. In: Proceedings of the International Symposium on Multiple-Valued Logic. (2008) 94–99
36. Sentovich, E.M., Singh, K.J., Lavagno, L., Moon, C., Murgai, R., Saldanha, A., Savoj, H., Stephan, P.R., Brayton, R.K., Sangiovanni-Vincentelli, A.L.: SIS: A system for sequential circuit synthesis. Technical report, University of Berkeley (1992)

Stephan Eggersglüß received his Diploma in Computer Science from the University of Bremen, Germany, in 2006. In 2006, he spent eight months with the Design for Test group of Philips Semiconductors, Hamburg, Germany. Since then he has been with the research group of Computer Architecture at the University of Bremen. He is currently working towards his Dr. degree. His research interests include Boolean Satisfiability and delay testing.

Görschwin Fey received his Diploma in Computer Science from the Martin-Luther Universität, Halle-Wittenberg, Germany in 2001. Since then he has been with the research group of computer architecture at the University of Bremen, where he received the Dr. degree in 2006. He was a guest associate professor at the University of Tokyo from Dec. 2007 to Jun 2008.

Andreas Glowatz received his diploma degree in computer science from the Fachhochschule in Hamburg, Germany, in 1988. Glowatz is project manager and senior principal at Mentor Graphics Development in Hamburg, Germany. He worked for many years in the development for DfT tools and his current interests are ATPG, Test Compression, and Logic BIST.

Friedrich Hapke received his diploma degree in electronics from the Fachhochschule in Hamburg, Germany. He is the Design For Test manager of the Mentor Graphics Development in Hamburg, Germany. His interest includes advanced testing techniques, automatic test pattern generation, failure diagnosis and design automation in general for DSM technologies. Hapke is the ho-

der of over 10 patents in the area of Design For Test inventions.

Juergen Schloeffel received his diploma degree in physics from Georg-August-University in Goettingen, Germany, in 1986. Schloeffel is project manager and principal at Mentor Graphics Development Hamburg in Germany. His current interest includes advanced testing techniques, failure diagnosis, Yield improvements and design automation for DSM technologies.

Rolf Drechsler received the Diploma and Dr. Phil. Nat. degrees in computer science from the J.W. Goethe-University, Frankfurt am Main, Germany, in 1992 and 1995, respectively. He was with the Institute of Computer Science at the Albert-Ludwigs-University, Freiburg im Breisgau, Germany, from 1995 to 2000. He joined the Corporate Technology Department of Siemens AG, Munich, in 2000, where he worked as a Senior Engineer in the Formal Verification Group. Since October 2001, he has been with the University of Bremen, Bremen, Germany, where he is now a Full Professor for Computer Architecture. His research interests include verification, testing, logic synthesis, and system design.