Accurate QBF-based Test Pattern Generation in Presence of Unknown Values

Erb, Dominik; Kochte, Michael A.; Reimer, Sven; Sauer, Matthias; Wunderlich, Hans-Joachim; Becker, Bernd

IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)
Vol. 34(12) December 2015

doi: http://dx.doi.org/10.1109/TCAD.2015.2440315

Abstract: Unknown (X) values emerge during the design process as well as during system operation and test application. X-sources are for instance black boxes in design models, clockdomain boundaries, analog-to-digital converters, or uncontrolled or uninitialized sequential elements. To compute a test pattern for a given fault, well-defined logic values are required both for fault activation and propagation to observing outputs. In presence of X-values, conventional test generation algorithms, based on structural algorithms, Boolean satisfiability (SAT), or BDD-based reasoning may fail to generate test patterns or to prove faults untestable. This work proposes the first efficient stuck-at and transitiondelay fault test generation algorithm able to prove testability or untestability of faults in presence of X-values. It overcomes the principal pessimism of conventional algorithms when X-values are considered by mapping the test generation problem to the satisfiability of Quantified Boolean Formulae (QBF). Experiments on ISCAS benchmarks and larger industrial circuits investigate the increase in fault coverage for conventional deterministic and potential detection requirements for both randomized and clustered X-sources.

Preprint

Accurate QBF-based Test Pattern Generation in Presence of Unknown Values

Dominik Erb*, Michael A. Kochte‡, Sven Reimer*, Matthias Sauer*,
Hans-Joachim Wunderlich‡, and Bernd Becker*
*University of Freiburg, Georges-Köhler-Allee 051, 79110 Freiburg, Germany
‡ITI, University of Stuttgart, Pfaffenwaldring 47, 70569 Stuttgart, Germany

***Abstract*—Unknown (X) values emerge during the design process as well as during system operation and test application. X-sources are for instance black boxes in design models, clock-domain boundaries, analog-to-digital converters, or uncontrolled or uninitialized sequential elements.**

To compute a test pattern for a given fault, well-defined logic values are required both for fault activation and propagation to observing outputs. In presence of X-values, conventional test generation algorithms, based on structural algorithms, Boolean satisfiability (SAT), or BDD-based reasoning may fail to generate test patterns or to prove faults untestable.

This work proposes the first efficient stuck-at and transition-delay fault test generation algorithm able to prove testability or untestability of faults in presence of X-values. It overcomes the principal pessimism of conventional algorithms when X-values are considered by mapping the test generation problem to the satisfiability of Quantified Boolean Formulae (QBF).

Experiments on ISCAS benchmarks and larger industrial circuits investigate the increase in fault coverage for conventional deterministic and potential detection requirements for both randomized and clustered X-sources.

***Index Terms*—Unknown values, X-values, ATPG, QBF, SAT, stuck-at fault, transition-delay fault**

I. Introduction

Unknown (X) values are caused by unspecified inputs or black boxes in the design, or during test by uncontrolled sequential elements, at clock domain crossings or A/D boundaries. Since both fault activation and propagation require well-defined values at the fault site and along the propagation path, X-values compromise the testability of faults in the circuit. The propagation of X-values can be controlled by X-blocking design-for-test hardware at additional hardware and performance cost.

Automatic test pattern generation (ATPG) algorithms for stuck-at faults are typically based on structural methods such as the D-algorithm [1], PODEM [2] or FAN algorithm [3], or on Boolean satisfiability (SAT) reasoning [4], [5].

To model signal states in the circuit in presence of X-values, n-valued logics with different accuracy have been introduced. The 5-valued logic for test generation of [1] has been extended to a 9-valued logic [6] to distinguish between X-valued signals in the fault-free and faulty circuit.

Two-valued SAT-based test generation algorithms such as [4], [5], [7], [8] do not model X-values at all. Such algorithms can be extended to process signal states with X-values [9], [10]. In [11], 3-valued signal states are used in a SAT-based test generator for path-delay faults. The minimal encoding of signal states with X- or high-impedance values for SAT-based ATPG is discussed in [12], [13].

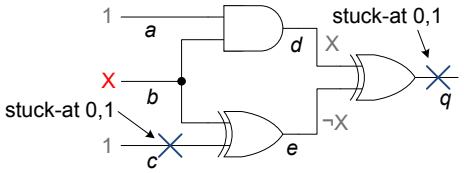

Fig. 1. Circuit with X-source at line b.

Restricted symbolic simulation [14] extends the number of symbols to distinguish *different* X-states and their inversion. This allows to reduce the pessimism of forward implication in test generation [15], unless multiple X-states from different X-sources converge at a gate.

Encoding the states of X-valued signals in the circuit with a *limited* number of symbols, as done in n-valued or restricted symbolic logics, introduces pessimism in the implication process. The limited number of symbols does not allow to reflect all correlations between X-valued signals. At reconvergences, where X-canceling may occur, the accurate output value cannot be computed any more. In consequence, forward and backward implication during test generation based on n-valued logics is pessimistic. In general, test generation algorithms based on n-valued logic cannot prove the untestability of faults in the support of X-valued signals and may not be able to find a detecting pattern for all testable faults.

Figure 1 shows an example of a circuit with one X-source at line b which reconverges at line q. Conventional ATPG algorithms fail to compute a test pattern for any stuck-at fault in this circuit. The reason is that the X-value at line b is propagated to lines e and q. Even if line a is assigned value 1, the reconvergence of the two complementary values at line q is evaluated pessimistically as X. However, the stuck-at-0 (stuck-at-1) faults at c and q are detectable with pattern $(a, c) = (1, 1)$ ($(a, c) = (1, 0)$).

The *accurate computation of signal states* in a circuit in presence of X-values can be achieved by formal reasoning for register-transfer and gate level simulation [16], [17], [18]. These methods employ symbolic computations by Boolean satisfiability (SAT), Quantified Boolean Formula (QBF) reasoning [19], or Binary Decision Diagrams (BDDs).

Increasing the accuracy in fault simulation can result in a significant increase of fault coverage [20]. Accurate fault simulation can be performed even for large circuits by a combination of heuristics and SAT reasoning [21], [22]. Both logic and fault simulation in presence of X-values are NP-complete problems. Yet deterministic test generation for stuck-at faults in presence of X-values is an NP-hard problem ([22] showed that verification of a guessed solution, i.e. fault simulation, is an NP-complete problem).

In contrast to propositional formulae used for SAT, QBF allows a succinct representation for all possible X-values using universal quantification. The recent advances in the performance of QBF solvers, for example conflict driven learning [23], elimination-based algorithms [24], or preprocessing [25] enable exact reasoning about fault testability in presence of Xs even for larger circuits.

In this work we present:

- A mapping of stuck-at fault test generation in presence of multiple X-sources to the QBF domain to accurately determine the testability of a fault [26].
- An extended version of this algorithm for transition-delay faults in presence of X-sources which either generates a test pattern or proves that such a pattern does not exist.
- The first algorithm computing the exact number of potentially detectable faults, i. e. where the fault effect can be measured for at least one X-source assignment.
- An extension of the QBF-based test generation algorithm for further fault detection modes and a discussion of the benefits.

The developed framework combines accurate reasoning with efficient hybrid two- and three-valued SAT-based ATPG, and accurate fault simulation of generated test patterns to keep the runtime as low as possible. QBF-based reasoning is only invoked for faults not classified otherwise. In addition, the impact of different QBF solver techniques on robustness and runtime are discussed.

Thorough experiments show the effectiveness of the framework for stuck-at and transition-delay faults. The pessimism of conventional algorithms is evaluated both for randomized and clustered X-sources.

Section II and Section III give required definitions and a formal problem statement, followed by the presentation of the proposed algorithm in Sections IV, V and the discussion of QBF solving techniques in Section VI. Section VII presents experimental results on academic and industrial circuits, and Section VIII summarizes the paper.

II. Terminology

A. Unknown Values

An unknown (X) value models an unknown *binary* state of a signal. This excludes *undefined* values that are not binary resulting for example from undefined voltage levels. Signals at which unknown values originate are called *X-sources*. The set of X-sources is denoted by S_X. There are $2^{|S_X|}$ possible binary assignments $\mathcal{A}_X = \{0,1\}^{|S_X|}$ to the X-sources. Nodes in the transitive fanout of an X-source are called X-dependent.

In three-valued logic, the three symbols $\{0, 1, \mathrm{X}^P\}$ are used to represent logic value 0, logic value 1, and the unknown binary state X^P of either 0 or 1. The signals with value X^P are computed *pessimistically* in three-valued logic.

Let the functions $\mathrm{val}(s,p) \mapsto \{0,1,\mathrm{X}\}$ and $\mathrm{val}^f(s,p) \mapsto \{0,1,\mathrm{X}\}$ return the value of signal s under pattern p in the fault free circuit and in the circuit affected by fault f, respectively, in presence of X-values. Note that the X-values at signals are computed accurately. Let functions $\mathrm{val}(s,p,a)$ and $\mathrm{val}^f(s,p,a)$ return the binary value $\{0,1\}$ of signal s under input pattern p and X-source assignment $a \in \mathcal{A}_X$ in the fault-free circuit and in the circuit affected by fault f.

B. Boolean and Quantified Boolean Satisfiability

This section provides a brief overview for the satisfiability problem (SAT) and Quantified Boolean Formulae (QBF). The interested reader is referred to [19] for more details.

Deciding the satisfiability of a propositional Boolean formula (SAT) is an NP-complete problem [27]. The formula is typically provided in conjunctive normal form (CNF). A CNF is a conjunction of clauses, and a clause is a disjunction of literals, e. g. $(a \vee \neg b)$ with the Boolean variables a and b.

A quantified Boolean formula (QBF) is a propositional formula in which the variables are quantified (or bounded) by existential ($\exists$) or universal ($\forall$) quantifiers. A QBF can be transformed into the prenex normal form (PCNF) $\psi = \mathcal{Q}X_1\mathcal{Q}X_2\ldots\mathcal{Q}X_n\varphi$, with $\mathcal{Q} \in \{\exists,\forall\}$ and X_i disjoint sets of Boolean variables. In a PCNF all quantifiers are grouped together in a so-called prefix and precede a quantifier-free propositional formula in CNF, called the matrix φ. We define the quantifier level by the number of quantifier alternations (i. e. from $\exists$ to $\forall$ or vise versa), reading the prefix from left to right. Without loss of generality, we assume that level 0 is always existential.

As an example, a QBF in PCNF ψ with three quantifier levels is satisfied if and only if: there *exists* an assignment for all variables on quantifier level 0 such that for *every* assignment for all variables on quantifier level 1, an assignment for all variables on quantifier level 2 *exists*, such that the matrix is satisfied.

Modern QBF solvers are also able to provide a model for free (unbounded) variables of the QBF. Semantically these free variables are similar to variables quantified at level 0. To increase readability, we write in the following that we extract the model for the variables on level 0 instead of using the terminology of free variables.

The complexity of QBF satisfiability is determined by the number of quantifier alternations between existential and universal quantifiers and vice versa in the prenex form. The general problem of QBF satisfiability is a PSPACE-complete problem [28].

III. Fault Detection Requirements

In this work, definite and potential detection requirements of stuck-at and transition-delay faults are distinguished.[1]

[1] Section V-F introduces two further detection modes which may be relevant in certain contexts, e. g. depending on test equipment capabilities or test response post-processing, but they are not the main focus of this work.

A. Definite Detection of Faults

The definite detection (DD) of a fault ensures that the fault effect is measurable at one known circuit output for all possible states of the X-sources.

A *stuck-at fault* f is DD for a pattern p if and only if there is one fixed output o at which the fault effect is observable for every X-source assignment:

$$\begin{aligned}\mathrm{DD}^f(p) := \exists o \in O : \mathrm{val}(o,p), \mathrm{val}^f(o,p) \in \{0,1\} \wedge \\ \mathrm{val}(o,p) \neq \mathrm{val}^f(o,p),\end{aligned} \tag{1}$$

where O is the set of output signals of the circuit.

The definite detection of a *transition-delay fault tf* at signal s requires the consideration of two cycles. In the first cycle, signal s is driven to a defined value ϕ to activate the fault. For a slow-to-rise transition-delay fault, ϕ is 0, for a slow-to-fall fault, ϕ equals 1. In the second cycle, the value of s is inverted and the resulting transition is propagated from s to an observable circuit output. This corresponds to detecting the stuck-at-ϕ fault at signal s in the propagation cycle. Thus, the definite detection of *tf* at signal s under pattern pair (p^{-1}, p) is given as:

$$\begin{aligned}\mathrm{DD}^{tf}(p^{-1},p) := &\mathrm{val}(s,p^{-1}) = \phi \wedge \\ &\mathrm{DD}^{\text{Stuck-at-}\phi \text{ fault at } s}(p).\end{aligned} \tag{2}$$

B. Potential Detection of Faults

A stuck-at fault f is potentially detected (PD) if an output o exists where the fault effect can be deterministically measured for at least one X-source assignment. Furthermore, a known binary value is required for the fault-free circuit. This results in:

$$\begin{aligned}\mathrm{PD}^f(p) := &\neg\mathrm{DD}^f(p) \wedge \exists o \in O : \\ &\mathrm{val}(o,p) \in \{1,0\} \wedge \mathrm{val}^f(o,p) = \mathrm{X}.\end{aligned} \tag{3}$$

Similar to the potential detection requirement for stuck-at faults, potential detection of a transition-delay fault *tf* at signal s requires that the fault is activated and its effect can be deterministically measured in the propagation cycle at at least one output o for at least one logic value assignment to the X-sources:

$$\begin{aligned}\mathrm{PD}^{tf}(p^{-1},p) := &\neg\mathrm{DD}^{tf}(p^{-1},p) \ \wedge \ \mathrm{val}(s,p^{-1}) = \phi \wedge \\ &\mathrm{PD}^{\text{Stuck-at-}\phi \text{ fault at } s}(p).\end{aligned} \tag{4}$$

Notice that for both, stuck-at and transition-delay faults, conventional three-valued modeling and analysis underestimates the number of definitely detected faults since three-valued simulation overestimates the number of signals with an X-value. Consequently, three-valued simulation may also overestimate the number of potentially detected stuck-at or transition-delay faults.

IV. Overview of the Proposed ATPG Framework

Before describing the used algorithms in detail, we give a short overview over the complete ATPG framework. The framework is able to prove the testability of stuck-at and transition-delay faults in presence of X-values. Figure 2 shows the complete flow which combines accurate fault simulation [22], incremental SAT-based test generation with a hybrid two- and three-valued encoding, a fast topological untestability check, and accurate QBF-based reasoning to efficiently analyze the faults.

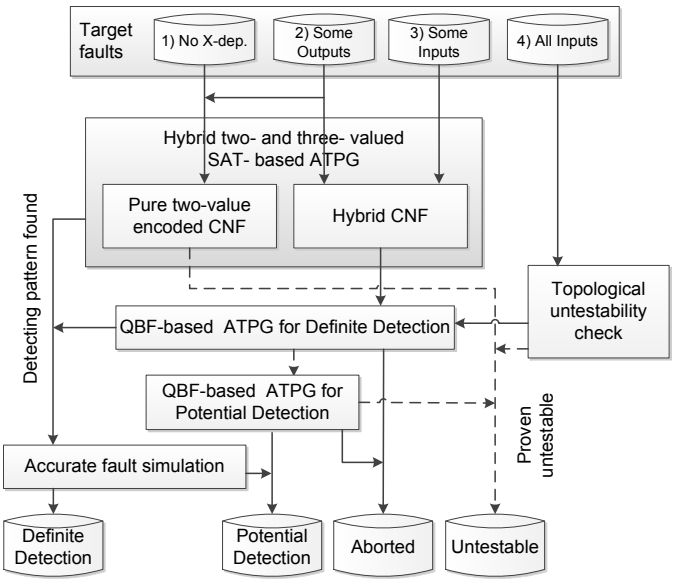

Fig. 2. Overview of the ATPG flow.

Using a topological analysis, the faults under analysis are partitioned into four groups w. r. t. their relation to the X-sources in the circuit (cf. Figures 2, 3):

1) No structural dependence on the X-sources, i. e. neither the justification cone of the fault nor its propagation cone depend on X-sources.
2) A subset of the outputs in the propagation cone depends on X-sources. The justification cone and at least one output in the propagation cone do not depend on X-sources.
3) A subset of the inputs in the justification cone of the fault depends on X-sources. At least one input in its justification cone is a controllable input.
4) The justification cone is driven exclusively by X-sources.

Afterwards the faults of each group are processed using the most suitable algorithms to keep the runtime as low as possible – while guaranteeing an accurate classification. First, all faults without X-dependency are processed by the hybrid SAT-based algorithms based on a pure two-valued signal encoding. In case a constructed formula is satisfiable, a test pattern is extracted and accurately simulated to implement fault dropping and to mark faults as potentially detected (cf. Section III). Otherwise, the fault is untestable.

All faults for which some outputs or some inputs depend on X-sources are subsequently processed by the SAT-based ATPG using a hybrid two- and three-valued encoding. Within this hybrid approach, gates are only encoded using a classical three-valued encoding if *all* inputs are reachable from X-sources. Otherwise, a pure two-valued encoding or a mixture of both is used to reduce the size of the generated CNF significantly. In case a constructed formula is satisfiable, a test

pattern is extracted and simulated. Otherwise, the SAT-based approach only allows to prove the untestability if the fault site itself does not depend on X-sources and fault activation is not possible. For all other faults which may still be detectable, a QBF is constructed and analyzed using a QBF solver for the final classification. Faults for which all inputs depend on X-sources and which have not been classified as untestable by the topological untestability check (cf. Section V-B) are also analyzed using the QBF-based approach.

Finally, each fault classified as untestable is analyzed again for potential detection by the QBF solver (cf. Section V-E).

V. ATPG Algorithms in Detail

This section describes each algorithm of the ATPG framework in detail focusing on stuck-at faults in all subsections but Section V-D.

A. Hybrid Two- and Three-Valued SAT-based ATPG

In SAT-based ATPG, a Boolean formula in CNF is constructed that represents the fault-free and faulty circuits and fault detection requirements. The Tseitin transformation [29] is used to efficiently encode a circuit into CNF. For instance, the clauses representing an AND gate with two inputs a and b and output q in two-valued SAT-based ATPG are: $(\neg q \vee a), (\neg q \vee b), (q \vee \neg a \vee \neg b)$. Additional constraints are added such that the resulting formula is satisfied if and only if there exists an input assignment such that the same output in the fault-free and faulty circuit representations show contrary logic values.

In two-valued SAT-based ATPG, the state of each signal in the fault-free and faulty circuits is modeled by a single binary variable, respectively. Consequently, it does not allow to model X-values. In contrast, three-valued SAT-based ATPG allows to search for test patterns in presence of X-values by modeling the three signal states $\{0,1,X^P\}$ both in the fault-free and faulty circuit. In this three-valued logic, the state of a signal s is encoded by two binary variables (s_1, s_2) such that value 0 corresponds to $(0,1)$, value 1 to $(1,0)$ and the X^P-value to $(0,0)$. In a pure three-valued encoding, the number of clauses is more than doubled compared to two-valued encoding. For example, an AND gate with inputs a and b and output q requires the six clauses: $(\neg q_1 \vee a_1)$, $(\neg q_1 \vee b_1)$, $(q_1 \vee \neg a_1 \vee \neg b_1)$, $(q_2 \vee \neg a_2)$, $(q_2 \vee \neg b_2)$, $(\neg q_2 \vee a_2 \vee b_2)$. Furthermore, for each signal s the state $(s_1, s_2) = (1,1)$ needs to be forbidden, e. g. by adding the clause $(\neg s_1 \vee \neg s_2)$.

To reduce the number of clauses, only the gates that may influence fault activation or propagation need to be modeled. The support (or *cone of influcence*) of a fault is the union of the input cones of outputs reachable from the fault site. Figure 3 shows the justification and propagation cone, as well as the support of a fault.

The search for a test pattern is also sped up by introducing D-chains which explicitly model propagation paths of the fault effect to outputs [4].

A hybrid two- and three-valued encoding allows to further minimize the size of the generated formula in presence of X-values. The two-valued encoding is used for all signals which

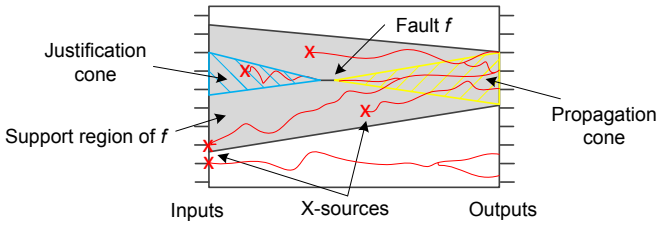

Fig. 3. Support region of stuck-at fault f.

do not depend on X-sources, and three-valued encoding only for signals depending on X-sources. If no (all) inputs of a gate depend on X-sources, the gate is transformed using a pure two-valued (pure three-valued) encoding. If only a subset of the inputs depends on X-sources, the three-valued gate encoding is adjusted to model the inputs without X-dependence with only one variable.

This case is illustrated in Figure 4 for an AND gate at the border between the two- and three-valued encoded area with two-valued input a, X-dependent input b, and X-dependent output q. This gate is represented by the following six clauses: $(\neg q_1 \vee a)$, $(\neg q_1 \vee b_1)$, $(q_1 \vee \neg a \vee \neg b_1)$, $(q_2 \vee a)$, $(q_2 \vee \neg b_2)$, $(\neg q_2 \vee \neg a \vee b_2)$. A similar selective modeling is used in [13].

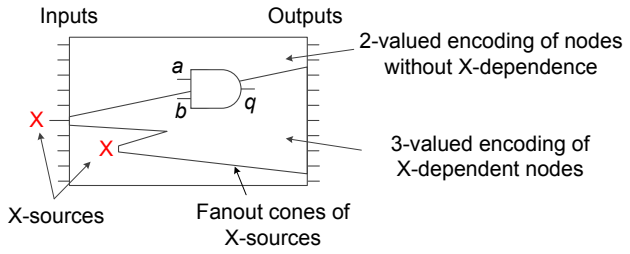

Fig. 4. Hybrid two- and three-valued SAT instance.

This results in a CNF formula which empirically is up to 60% smaller and therefore easier to be analyzed by the SAT solver than a pure three-valued encoding, leading to 20% lower runtimes.

B. Topological Untestability Check

Fault detection requires fault activation at the fault site. For faults which are only reachable from X-sources, justification of a known binary value at the fault site requires X-canceling at a reconvergence of X-valued signals in the input cone.

A simple tracing of branching signals is performed to check for such reconvergences. If none are found for signal s, the faults at s cannot be activated. Consequently, they are untestable w. r. t. the definite detection criterion of Section III. If reconvergences are found, the faults may be testable and are analyzed by the QBF-based approach.

C. QBF-Based Definite Detection of Stuck-at Faults

All faults for which testability or untestability has not been proven yet are subject to QBF-based ATPG. If the QBF is satisfiable, the test pattern is extracted from the model

provided by the QBF solver. The construction of the QBF is split into the generation of the matrix and the quantification of the variables (cf. Section II).

1) Construction of the matrix for ATPG: The matrix of the QBF in CNF is constructed similar to a classical two-valued SAT-based ATPG instance explained in Section V-A. The state of each signal is modeled by a single binary variable. X-values are not explicitly specified in the matrix but modeled by universal variable quantification.

To construct the matrix for a fault f, all necessary gates for the fault-free circuit representation C^G and the propagation cone C_P^f of the fault f in the faulty circuit are modeled as formulae in CNF (cf. Figure 3). Additionally, D-chains are added to encode propagation paths from the fault site to the outputs and to guide the search for a test pattern. For the D-chains, d-variables are added for each signal in the propagation cone of the fault. If the signal s has complementary values in C^G and C_P^f, d_s evaluates to 1. Finally, a single clause $\mathcal{D} := \bigvee_{o \in \mathrm{O}} d_o$ is added to ensure that at least one d-literal of a circuit output is 1. This leads to the following propositional formula in CNF:

$$\text{CUT} = C^G \wedge C_P^f \wedge (\text{D-chain clauses}) \wedge \mathcal{D}.$$

Figure 5 shows an example with two X-sources and a fault, only detectable by accurate modeling. For this fault, C^G includes a two-valued encoded version of the gates G_1, G_2, G_3, G_4 (cf. Section V-A) assuming that no fault is present. C_P^f includes a two-valued encoded version of the gates G_2, G_3, G_4 assuming a logic 0 to be present at signal c and referencing the signals used in C^G otherwise. To model the propagation path, for each gate in C_P^f one *d-variable* is introduced which evaluates to 1 if the output of the corresponding gates in C^G and C_P^f show complementary logic values. Using the signal names as auxiliary variables this leads to the d-literals d_f, d_q and d_r. For the D-chains, further clauses are added ($(d_q \rightarrow d_f)$ and $(d_r \rightarrow d_f)$), and at last the clause $(d_q \vee d_r)$ is added to ensure detection at at least one output.

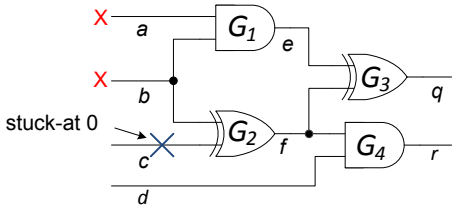

Fig. 5. Circuit with two X-sources at signals a, b and a stuck-at 0 fault at signal c only detectable by accurate modeling.

2) Variable Quantification: All variables used in the matrix need to be properly quantified to guarantee a valid test in case the formula is satisfiable – or otherwise to serve as a proof that a test pattern does not exist. It is important to respect the dependencies of the circuit components and therefore the scope of quantification, i. e. the sequence of quantifier alternations.

For fault detection, we search for *one* test pattern that satisfies the matrix for *all* possible assignments to the X-sources. Thus, the variables representing the circuit inputs are existentially quantified on level 0 and precede the universally quantified variables representing the X-sources on level 1.

The internal signals S and the d-variables used for the D-chains are subsequently existentially quantified at level 2, since they depend on the values of the inputs and X-sources. This results in the following QBF:

$$\underbrace{\exists I}_{\substack{\text{Controllable}\\ \text{inputs}}} \overbrace{\forall X}^{\text{X-sources}} \underbrace{\exists S \exists D}_{\substack{\text{Int. signals,}\\ \text{D-chain variables}}} \text{CUT}.$$

This QBF is satisfiable if and only if there exists an input assignment which excites an observable difference at at least one (not necessarily the same) output for each possible assignment to the X-sources.

Enforcing Definite Detection at Circuit Outputs: To establish definite detection according to Equation (1) of Section III, the solution space is constrained by limiting the detecting outputs to a single fixed one.

This constraint is implemented by additional variables o_i for the outputs in the propagation cone which only evaluate to 1 if the fault effect is observable at output i for all assignments to the X-sources. The clause $(o_1 \vee o_2 \vee \ldots \vee o_n)$ enforces that at least one of the variables o_i evaluates to 1 and thus, the fault is always observable at at least one output. To guarantee that the observable output is fixed for all possible X-values, the variables in $O = \{o_i \mid 1 \leq i \leq n\}$ are existentially quantified at quantifier level 0 preceding the universal quantification of the X-sources on level 1. The relation between o_i and the D-chains are established by adding one implication per output $(o_i \rightarrow d_i)$ to the matrix:

$$\exists \boldsymbol{O} \exists I \forall X \exists S \exists D \left(\text{CUT} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i) \right).$$

This enforces a fixed detecting output over all assignments to X-sources. However, the observable difference, i. e. the signal values in the fault-free and faulty circuit at that output, is still allowed to be one of the three possibilities $(0/1), (1/0), (x_i, \neg x_i)$. The last case corresponds to the situation where an output always shows complementary states in the fault-free and faulty circuit for all assignments to the X-sources, but the values in the fault-free and faulty circuit are not stable for all assignments to the X-sources. In this case, it is not possible to distinguish between a fault-free and a faulty circuit during testing.

Enforcing Known Binary Values at Circuit Outputs: A known binary value at the observing output in the fault-free circuit is enforced by adding two variables v_i^0, v_i^1 per output to represent its stable value in the fault-free case when it detects the fault. This automatically constrains the faulty case as well. If v_i^0 (v_i^1) is true, output i has the stable value 0 (1) in the fault-free circuit. The two implications $(v_i^0 \rightarrow \neg s_i)$ and $(v_i^1 \rightarrow s_i)$ for output i establish that relation, assuming that $s_i \in S$ is the signal variable representing the value of output i in the fault-free circuit. In the formula $\phi_{\text{Stable output}}$, the implication $(o_i \rightarrow (v_i^0 \vee v_i^1))$ ensures that output i has a stable value if o_i

is asserted:

$$\phi_{\text{Stable output}} := \bigwedge_i \big((o_i \rightarrow (v_i^0 \vee v_i^1)) \wedge (v_i^0 \rightarrow \neg s_i) \wedge (v_i^1 \rightarrow s_i)\big).$$

Since the outputs have to be stable independent from the X-sources, the variables $v_i^0, v_i^1 \in V$ are existentially quantified on level 0 resulting in the following QBF:

$$\begin{aligned}\text{DD} := \exists O \, \exists \boldsymbol{V} \, \exists I \, \forall X \, \exists S \, \exists D \\ \Big(\text{CUT} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i) \wedge \phi_{\text{Stable output}}\Big)\end{aligned}$$

As an example, for the circuit shown in Figure 5 with the X-sources a, b, the controllable input c, and the CNF CUT described at the end of Section V-C1 (without specifying all inner variables S), the complete QBF is:

$$\begin{aligned}DD_{CUT} := \exists o_q \, \exists o_r \, \exists v_q^0 \, \exists v_q^1 \, \exists v_r^0 \, \exists v_r^1 \, \exists c \, \exists d \\ \forall a \, \forall b \\ \exists I \, \exists d_f \, \exists d_q \, \exists d_r \\ \Big(\text{CUT} \wedge (o_q \vee o_r) \wedge (o_q \rightarrow d_q) \wedge (o_r \rightarrow d_r) \wedge \\ (o_q \rightarrow (v_q^0 \vee v_q^1)) \wedge (v_q^0 \rightarrow \neg q^G) \wedge (v_q^1 \rightarrow q^G) \wedge \\ (o_r \rightarrow (v_r^0 \vee v_r^1)) \wedge (v_r^0 \rightarrow \neg r^G) \wedge (v_r^1 \rightarrow r^G)\Big)\end{aligned}$$

This QBF is satisfiable if and only if a fault is testable according to the definite detection condition of Section III. If the QBF is not satisfiable, a test pattern for definite detection does not exist.

D. Extension of the QBF Method for Transition-Delay Faults

To generate test patterns for transition-delay faults, at least two cycles need to be considered. In the first cycle the fault is activated, and in the second cycle a stuck-at fault is assumed to be present at the fault site (cf. Section III). Hence, the QBF-based method presented above is extended by using a time frame expansion such that in the first cycle, the fault site is justified to a predefined value, and in the second cycle, a stuck-at fault is assumed to be present.

Figure 6 shows the support region of transition-delay fault f with two justification cones for the two cycles and the propagation cone within the second cycle. The matrix and the variable quantification are encoded similar to the method for stuck-at faults. The main differences are: (1) the fault-free circuit representation C^G is extended and contains now signals and gates from two cycles, (2) a further constraint is added to adjust the fault-free value in the first cycle (C^G_{tf1}), (3) C^f_P only references signals and gates of the second cycle. Thus, the prefix of the QBF presented in Section V-C2 needs no adjustments, and the new matrix is given by:

$$\text{CUT}_{tdf} = C^G \wedge C^G_{tf1} \wedge C^f_P \wedge (\text{D-chain clauses}) \wedge \mathcal{D}.$$

Such a two-cycle modeling is also used for the hybrid two- and three-valued SAT-based ATPG.

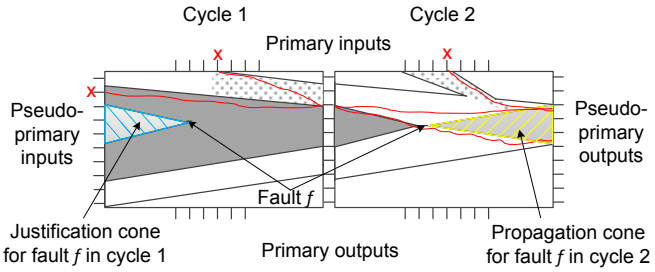

Fig. 6. Support region of transition-delay fault f.

E. Generation of Potentially Detecting Test Patterns

For some faults in the fanout cone of the X-sources, test patterns for definite detection do not exist. For these faults, it may be still worthwhile to analyze whether test patterns for potential detection exist.

Test pattern generation for potential detection is also mapped to QBF satisfiability. According to Equation (3) in Section III, a fault f is potentially detected under a pattern if there exists one fixed output that has a binary value in the fault-free circuit and an X-value in the faulty circuit affected by f. The structure of the QBF instance for potential fault detection is shown in Figure 7.

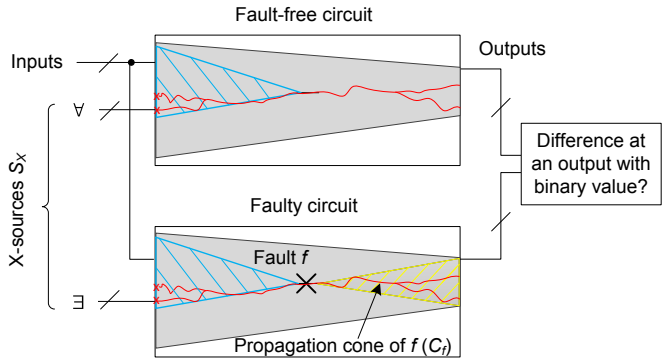

Fig. 7. Principle of the QBF instance for potential detection test generation.

In this instance, the constraints for the fault-free and the faulty circuits are different: For the fault-free circuit, there must be one output $o_i \in O$ with a known binary value v. For the faulty circuit, it is sufficient that output o_i carries the opposite value $\neg v$ just for one arbitrary X-source assignment. Thus, for these two parts different variable quantifications are required. Following the reasoning of the previous sections, this is represented by the following QBF:

$$\begin{aligned}\text{PD} := \; \exists O \, \exists V \, \exists I \, \big[\forall \text{X} \, \exists S \, (C^G \wedge \bigvee_i o_i \wedge \phi_{\text{Stable output}}) \wedge \\ \exists \text{X} \, \exists S \, (C^f_P \wedge \psi)\big],\end{aligned}$$

$$\psi := \bigwedge_i \Big(o_i \rightarrow \big((v_i^0 \rightarrow s_i^f) \vee (v_i^1 \rightarrow \neg s_i^f)\big)\Big).$$

In the fault-free circuit representation, the variables of the X-sources are universally quantified so that an output with a known binary value for all X-source assignments is

searched for. In the faulty circuit representation, it is sufficient to quantify these variables existentially to detect the fault for a single X-source assignment. The variables $o_i \in O$ that specify the detecting output and the variables $v_i^0, v_i^1 \in V$ representing its value in the fault-free circuit are quantified at level 0. This ensures that the value difference at the outputs is present independently of the X-sources in both the fault-free and faulty circuits. The value of output o_i in the faulty circuit is represented by variable s_i^f. If this formula is not satisfiable, then there is no output at which fault f can be potentially detected.

F. ATPG Extensions for Further Fault Detection Modes

The QBF-based test generation framework allows easy incorporation of futher detection modes. Such modes may increase fault coverage and are beneficial in certain test contexts or applications, for example: (1) The detecting output does not need to be fixed and known. (2) Test post-processing is used to determine fault detection when only a *value difference* between the fault-free and faulty circuits at the outputs can be established, but the values are unknown.

For the presented QBF-based algorithms, test patterns for such detection modes can be generated by modifying the variable quantification of the QBF (cf. Section V-C).

To allow multiple outputs for fault detection as in (1), the quantification of all variables enforcing detection at a single fixed circuit outputs (O) is changed from quantifier level 0 to level 2. This leads to the QBF:

$$\text{Det}_{\text{multiple output}} := \exists V \exists I \forall X \exists \boldsymbol{O} \exists S \exists D \left(\text{CUT} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i) \wedge \phi_{\text{Stable output}}\right).$$

If only a *value difference* at an output is required as in (2) above, the quantification of all variables used to enforce a known binary difference at circuit outputs can be changed:

$$\text{Det}_{\text{value difference}} := \exists O \exists I \forall X \exists \boldsymbol{V} \exists S \exists D \left(\text{CUT} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i) \wedge \phi_{\text{Stable output}}\right).$$

However, for this kind of detection, the fault effect cannot be directly measured since the reference value for comparison is not known. Thus, response post-processing is required to find a reference output, or multiple cycles can be used for fault activation and propagation as in [30].

G. Accurate Fault Simulation

Fault simulation is used to find all faults detected by the generated patterns. We employ the fault simulation algorithm of [22] which is able to accurately compute the detected stuck-at faults and transition-delay faults of a pattern in presence of X-values. For completeness this fault simulation algorithm is summarized in this section.

The fault simulation algorithm firstly computes the accurate logic values in the fault-free circuit for a given pattern, and then analyzes all yet undetected faults explicitly. The logic simulation of the fault-free circuit employs pattern-parallel logic simulation of randomized X-source assignments and restricted symbolic simulation to prove signal dependence or independence of X-sources for as many signals as possible. For the subset of signals for which the state is only known pessimistically, a SAT instance is incrementally constructed to compute the accurate logic value or prove dependence on at least one X-source.

Once the signal states in the fault-free circuit are known, the activated faults are processed serially. For each activated fault f, the fanout cone of f is simulated using randomized X-source assignments and restricted symbolic simulation in event-driven manner. If a fault is classified as definite detected by simulation, further analysis is not required. Otherwise, a SAT-based analysis of the outputs of the faulty circuit is conducted to classify the fault as undetected, definitely detected or potentially detected.

VI. Increased Robustness using Different QBF Solver Techniques

The two most successful algorithms for solving QBF are: 1) search-based and 2) elimination-based algorithms. The search-based (and conflict-driven) algorithms [23] are extensions of the pre-dominant technique in SAT solvers. The elimination-based techniques [24], [31] expand the formula "inside-out", i. e. the quantifier prefix is successively eliminated from innermost to outer quantifier level by replacing the variables and expanding the matrix until either a tautology or contradiction is deduced. This method is memory-intensive, and in the worst case the matrix blows-up exponentially. For (small) hard-to-solve instances, elimination techniques are in general better suited than search-based solving. Moreover, many search-based solvers often fail to handle designs with linear/XOR-based logic efficiently [32].

Modern search-based QBF solvers allow to adjust several heuristics and techniques, among others: heuristics for choosing a variable to decide next, the restart heuristic, or the initial assignments for variables. In particular, there exist several decision heuristics [33] in the SAT domain which are adapted for QBF solvers. The heuristics have a huge impact concerning the pruned parts of the search space. These heuristics closely cooperate with restart heuristics preventing the solver to get lost in non-relevant parts of the search space. Therefore, restarts are performed from time to time, resetting the current solver progress (but keeping the learnt information) in order to prune a different part of the search space. For both fields there exist heuristics which are better suited for different kinds of instances, e. g. un/satisfiable instances, or easy- vs. hard-to-solve instances [34].

The ATPG in [26] uses two timeouts of 1 second and 10 seconds for QBF-based test generation with the search-based solver QuBE [35]. The heuristics for both timeouts are identical. The results in [26] show that faults not classified within 1 second are only rarely classified with the larger timeout of 10 seconds. However, QBF solving with the larger timeout consumes most of the runtime of the whole approach. Furthermore, the number of aborted faults increases especially for designs with XOR-based logic.

This motivates to increase the robustness of the test generation approach by two different extensions: firstly, the spe-

cialization of a solver for different timeouts, and secondly, the use of completely different solvers – implementing different QBF solving algorithms. These two approaches are discussed in detail in Section VII-B.

VII. Evaluation

A. Experimental Setup

The proposed algorithms are implemented in C. All SAT based approaches use the incremental SAT-solver `antom` [36]. For the QBF-based algorithm, we used an extended QBF version of `antom`, named `quantom` [37], and two timeouts (1 second and 10 seconds). For the first timeout, `quantom` is tuned for easy-to-solve instances. For the 10 second timeout, `quantom` is adjusted to handle hard-to-solve instances more efficiently as discussed in Section VII-B.

We evaluated the algorithms on full-scan circuits of the largest ISCAS'85 and ISCAS'89 benchmarks, as well as larger industrial designs from NXP. All measurements were conducted on a single core of an Intel Xeon CPU at 3.30 Ghz and 16 GB of memory. In all experiments but Section VII-G we assume that a fixed and randomly selected subset of circuit inputs generates X-values. The experiments in Section VII-G investigate the impact of clustered X-sources.

Five different subsets of X-source inputs are generated per circuit. The results show the rounded average over these five experiments per circuit with different percentages of the inputs selected as X-sources.

For each circuit, the collapsed set of stuck-at faults is computed, 1024 random patterns are simulated with a fast three-valued and an accurate fault simulator [22] for each set of X-sources. The remaining random-pattern resistant faults are processed using the proposed test generation algorithms.

B. Increased Robustness using Different QBF Solver Techniques

As described in Section VI, the robustness may be increased by using different QBF solving heuristics for the two timeouts, or even orthogonal QBF solving algorithms. Two different extensions have been evaluated to measure the impact on the ATPG runtime and the number of aborted faults.

1) Problem Specific Solver Settings: In extensive experiments we searched for two proper settings for our used solver `quantom` and for each timeout: The first setting is dedicated to easy-to-solve instances and is used for the 1 second timeout. The second setting is trimmed to solve harder instances and is used for the 10 second timeout. Exemplary, for the easy-to-solve instances we use the DLCS decision heuristic [33] and the glucose restart scheme [34] pruning wide (but not deep) parts of the search space earlier, therefore this is more likely to find short (i. e. easy) solutions/contradictions. In contrast, for hard-to-solve instances we use the decision heuristic of QuBE [35] and the Luby restart heuristic [38] pruning the search space deeper (but not wide), and therefore this is in general better suited for harder benchmarks.

2) Orthogonal Solving Methods: To increase robustness in presence of XOR-based logic in addition to the search-based QBF solver `quantom` (which is still used for the easy-to-solve instances with a timeout of 1 second), the elimination-based QBF solver AIGSolve [39] is also employed using a timeout of 10 seconds. This solver uses Functionally Reduced AND-Inverter-Graphs (FRAIGs) [40] as underlying data-structure for representing the formula. FRAIGs are particularly suitable for circuit designs, since the gates are handled explicitly and efficiently. Unfortunately, this elimination-based method produces much overhead for handling the data-structures and is less suited for easy-to-solve instances. As a consequence, we used AIGSolve only for the instances which cannot be classified using `quantom` within 1 second. Furthermore, the techniques for eliminating a single variable may require a high runtime and cannot be aborted efficiently. Thus, AIGSolve often clearly exceeds the timeout.

The two extension have been evaluated on the ISCAS'85 circuit c7552 with linear/XOR-based error control logic and an industrial circuit p78k to evaluate the extensions for much larger instances (cf. Table I for circuit data).

Figure 8 compares the results of the original solver ('Baseline') and the two extensions described in this section for stuck-at fault test pattern generation. For each approach, the percentage of aborted faults and the runtime in seconds for an X-ratio of 2% and 5% are shown. The results indicate that the error control logic of circuit c7552 causes many faults to be aborted with the timeout for both approaches solely using a search-based solver.

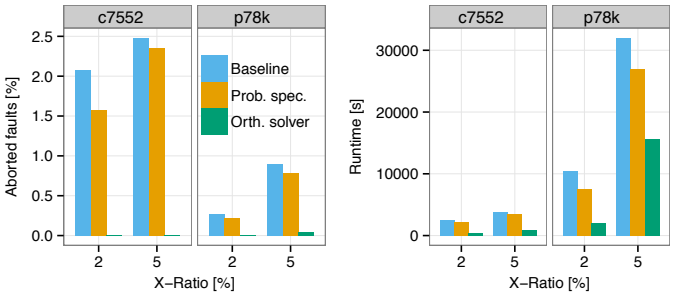

Fig. 8. Comparing the number of aborted faults and the runtime for stuck-at fault test generation of the original solver 'Baseline' and two extensions: Problem specific solver settings ('Prob, spec.') and Orthogonal solving methods ('Orth. solver').

In contrast, if the elimination-based solver AIGSolve is invoked for the hard-to-solve instances, almost all of these faults are classified. Thus, the results clearly show the advantages of using 'orthogonal solving techinques' compared to the baseline algorithm. For circuit c7552, the aborted faults reduce from on average 1.43% to 0.01%. For 108 faults out of these 3046 additionally classified faults a test pattern was found. All others are proven untestable.

Considering the runtime, the use of 'problem specific solver settings' reduces the runtime by up to 28.23% compared to the baseline algorithm, while the use of 'orthogonal solving methods' reduces the runtime by up to 85.49%. For both circuits, we also evaluated the overhead in case only the elimination-based solver AIGSolve is used and hence, AIGSolve also needs to classify all easy-to-solve instances. However, while the number of aborts stays on the level of

'orthogonal solving methods', the runtime is more than twice as high as the runtime of 'baseline'.

The used version of AIGSolve has not been tuned for large netlist-based instances and may require an excessive amout of memory for intermediate results. For certain faults in circuit p89k and p100k for instance, the available memory of 16 GB was insufficient to solve the QBF instance. Thus, in the following experiments, the 'problem specific solver settings' are used.

C. Definite and Potential Stuck-at Fault Detection

The following experiments assess the achievable fault coverage by the proposed algorithm for varying X-ratios and the two circuits c7552 and s13207. Figure 9 shows the increase in the number of definitely detectable faults 'Def. Detect Inc.' over conventional three-valued ATPG for the two circuits and X-ratios from 1 to 99%. The figure also shows the absolute number of faults marked as potentially detected only by accurate fault simulation 'Pot. Det. (Fsim)' or with explicit potentially detecting pattern generation (cf. Section V-E) 'Pot. Det. (ATPG)'. Finally, the number of faults aborted by the QBF solver is given.

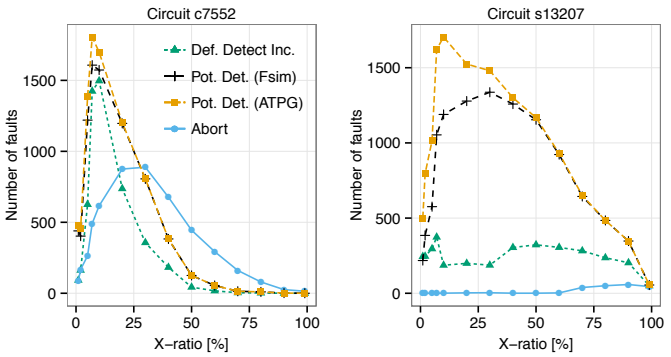

Fig. 9. Increase in definitely detectable faults (Def. Detect Inc.) compared to three-valued ATPG depending on the X-ratio, absolute number of potentially detected and aborted faults.

For all investigated X-ratios, the proposed algorithm generates test patterns for a large number of faults which were classified as untestable or unclassified by the pessimistic conventional three-valued ATPG algorithm.

For circuit c7552, up to 1497 more faults are classified as definitely detected. This correlates to an increase in fault coverage of 15.30% points. Accurate fault simulation is already able to classify up to 1607 untestable faults as potentially detected, which is 16.50% of all faults. Furthermore, if the potentially detecting patterns are explicitly generated, the number of untestable faults marked as potentially detected increases to up to 1806 faults at an X-ratio of 7%. This correlates to 39.76% of the untestable faults. Similar results are also achieved for circuit s13207. Here, fault simulation allows to classify up to 5.95% and potentially detecting pattern generation up to 8.42% of the faults as potentially detected. For higher X-ratios, the number of additionally definitely or potentially detected faults decreases and reaches nearly 0% for an X-ratio of 99%. This is because different X-ratios and input patterns may lead to different numbers of reconvergences in the circuit. As a result, the number of additionally classified faults increases for lower X-ratios (as additional X-sources lead to further reconvergences) and decreases for higher X-ratios (as the high number of X-values prohibits the reconvergences of single X-values).

D. Stuck-at and Transition-Delay Fault Coverage in Larger Circuits

Table I shows the result of the accurate ATPG algorithm targeting stuck-at faults for the largest ISCAS'85 and ISCAS'89 benchmark circuits[2] as well as three larger industrial designs from NXP. For each circuit the table lists the size in number of complex gates and the number of collapsed stuck-at faults. Per circuit, we conduct experiments for the case of 1%, 2%, and 5% of the circuit inputs as X-sources ('X-ratio'). For circuit c6288 with only 32 inputs, the case of 2% is omitted since an X-ratio of 1% and 2% results in a single X-source.

For each of these cases, columns 5 to 7 contain the results for a three-valued SAT-based ATPG depicting the fault coverage 'FC', the fraction of untestable faults 'UT', and the fraction of unclassified faults 'UC'.

Columns 8 to 14 show the results of the proposed ATPG. Column 'ΔFC' lists the increase of fault coverage compared to three-valued ATPG in percent points. Column 'UT' shows the fraction of untestable faults (regarding the definite detection requirements of Section III) and column 'abort' shows the fraction of faults for which no final classification was possible because of the set timeout. Column 'PD' lists the number of untestable faults classified as potentially detected by accurate fault simulation of the generated pattern set. Finally, columns 12 and 13 show the number of faults classified with timeouts of 1 second and 10 seconds, and the last column gives the overall runtime of the accurate ATPG in seconds. On average, fault simulation of 1024 random patterns classifies 88.2% of all detected faults while reducing the runtime of the whole test generation.

The results show that in presence of X-sources, three-valued ATPG is incapable of generating test patterns for all testable faults or to prove untestability in case no pattern was found. This causes a huge number of unclassified faults. In contrast, the proposed ATPG framework not only increases the stuck-at fault coverage by up to 32.25% points for the ISCAS benchmarks (by up to 7.99% points for the industrial circuits from NXP) but also proves the untestability for almost all other faults. On average over all circuits, 4.05% of the faults could be marked as additionally detectable, 9.10% are marked as untestable, and only for 0.54% of the faults no final classification was found. For on average 6.18% of the untestable faults, at least a potentially detecting pattern was found by accurate fault simulation.

Most of the faults classified by the QBF-based ATPG are found within the timeout of 1 second. Over 99% of the faults solved with the 10 secondes timeout are classified as

[2]The easily testable circuit s35932 has been omitted since a few random patterns already detect all testable faults for the considered X-ratios.

untestable. Hence, the overall runtime is dominated by faults for which untestability was proven.

Table II additionally reports the results for transition-delay faults for ISCAS'89 and NXP circuits. As for Table I, the columns show the fault coverage, the fraction of faults classified as untestable, and the fraction of unclassified faults for a three-valued SAT-based ATPG and the accurate ATPG. Furthermore, the number of potentially detected faults, faults classified within a timeout of 1 or 10 seconds, and the overall runtime is given.

For transition-delay faults, the proposed ATPG increases the fault coverage significantly by up to 7.91% points. On average over all tested circuits and 5% of the inputs selected as X-sources, 3.06% more faults could be marked as detectable. Furthermore, most of the faults for which no test pattern could be generated are proven to be untestable. Out of these 16.36% untestable faults, for 10.22% a potentially detecting pattern was found.

The runtime for generating transition-delay faults considering two cycles is moderate in most cases. For the NXP circuit p78k, the runtime is even lower compared to the accurate stuck-at fault classification considering one cycle.

E. Comparison to a Commercial ATPG Tool for Stuck-at Faults

We additionally compared the proposed accurate ATPG to a state-of-the-art commercial X-aware ATPG tool using an abort-limit of 1 000 backtracks per fault. Table III shows the results for circuit c6288, for which the fault coverage increase of the accurate analysis is very high, and the three largest circuits from Table I. For the three larger circuits, the results of the commercial ATPG are in line with the three-valued ATPG used in all other experiments (cf. [26]). For circuit c6288, a 16 bit multiplier with many reconverging paths, the fault coverage of the commercial tool is even worse compared to the three-valued ATPG (cf. Table I). Without accurate analysis, such a result requires the use of X-blocking design-for-test to achieve a sufficient fault coverage.

Considering all circuits listed in Table III, and 5% of the inputs selected as X-sources, the proposed accurate QBF-based ATPG increases the fault coverage on average by 11.51% points compared to the commercial tool. Hence, the applied commercial tool seems to be incapable of evaluating reconvergences of X-values accurately.

However, the total runtime of the accurate ATPG is on average one order of magnitude higher than for the commercial tool. Most of the runtime is spent for solving QBFs using a timeout of 10 seconds. As most of these faults are classified as untestable, the total runtime can be more than halved by skipping the analysis with a 10 second timeout. This reduces the fault coverage only slightly. On average over all circuits and X-values in Table III, the fault coverage would then be less than 0.01% smaller while the runtime decreases by 68.05%. Also, a heuristic approach such as [41] can be used if the runtime is too high for large circuit instances.

Figure 10 depicts the range of the additionally achieved

TABLE I
RESULTS OF THE PROPOSED ACCURATE ATPG IN CONTRAST TO A THREE-VALUED ATPG FOR STUCK-AT FAULTS.

Circuit	Gates	Faults	X-Ratio	**3-val. SAT-based ATPG**			**Proposed QBF-based ATPG**						
			[%]	FC [%]	UT [%]	UC [%]	**ΔFC [%pt.]**	**UT [%]**	Abort [%]	PD	Class. 1s	Class. 10s	Time [s]
c6288	2 416	8 704	1.0	83.23	0.76	12.81	**14.12**	**2.51**	0.10	34	151	2	110
			5.0	62.42	1.34	28.19	**32.25**	**4.67**	0.28	40	292	6	297
c7552	4 043	10 816	1.0	91.80	0.21	8.00	**0.98**	**6.38**	0.85	199	654	18	1 090
			2.0	88.51	0.28	11.21	**1.72**	**8.20**	1.57	215	793	67	2 168
			5.0	68.02	0.62	31.37	**6.48**	**23.15**	2.35	219	2 299	182	3 478
s13207	8 027	21 5281	1.0	89.50	1.37	9.13	**1.41**	**9.09**	0.00	193	1 768	0	18
			2.0	84.26	2.02	13.72	**1.42**	**14.33**	0.00	377	2 724	1	27
			5.0	78.84	3.80	17.37	**1.65**	**19.52**	0.00	595	3 493	1	29
s15850	10 211	25 421	1.0	94.19	1.35	4.46	**0.74**	**5.06**	0.01	118	974	1	37
			2.0	92.59	1.53	5.88	**0.76**	**6.63**	0.01	158	1 338	0	46
			5.0	81.51	2.69	15.80	**1.85**	**16.57**	0.06	119	3 648	3	241
s38584	21 462	58 263	1.0	93.50	4.85	1.65	**0.08**	**6.42**	0.00	196	918	0	18
			2.0	90.48	4.74	4.78	**0.34**	**9.18**	0.00	356	2 657	0	29
			5.0	82.64	4.93	12.43	**0.55**	**16.81**	0.00	960	7 016	14	93
s38417	23 537	59 041	1.0	95.54	0.97	3.49	**0.17**	**4.23**	0.05	405	1 902	22	440
			2.0	93.58	1.30	5.12	**0.25**	**6.07**	0.10	689	2 786	34	800
			5.0	86.54	2.75	10.71	**0.45**	**12.87**	0.15	1 109	5 960	14	1 085
p78k	74 243	225 476	1.0	97.30	0.41	2.29	**1.40**	**1.20**	0.11	329	1 607	244	3 656
			2.0	93.60	0.88	5.53	**3.43**	**2.74**	0.22	578	3 976	467	7 507
			5.0	84.25	2.21	13.55	**7.99**	**6.98**	0.78	1 667	10 219	1 399	26 989
p89k	88 726	239 090	1.0	91.11	0.73	8.16	**1.00**	**7.32**	0.57	771	12 896	2 894	22 694
			2.0	85.49	1.00	13.51	**1.10**	**12.82**	0.60	890	23 716	4 554	27 993
			5.0	70.90	2.40	26.70	**1.86**	**26.51**	0.72	1 180	53 957	3 851	32 943
p100k	96 685	259 322	1.0	95.31	0.57	4.11	**0.85**	**3.46**	0.38	922	6 721	985	15 474
			2.0	91.48	0.90	7.61	**2.06**	**6.01**	0.45	922	13 443	1 315	20 405
			5.0	80.54	1.92	17.54	**3.45**	**14.52**	1.49	2 871	33 219	2 665	65 154

fault coverage of the proposed accurate ATPG compared to the commercial tool for the five iterations per circuit and X-ratio (cf. Section VII-A). For circuit p78k with an average increase in fault coverage of 7.99% points (Table III), the increase ranges from 6.53% to 9.29% points. For an X-ratio of 5%, circuit c6288 has an average increase in fault coverage of 34.25% points compared to the commercial tool. For this scenario, the minimum increase in fault coverage is 8.05% points and boosts coverage from 87.34% to 95.40%. The maximum increase in coverage is 56.50% points, lifting coverage from 38.97% to 95.47%.

We also evaluated the influence of the abort-limit used for each fault in the commercial tool. If the abort-limit is increased to 10 000 backtracks per fault, the fault coverage increases only slightly. For an X-ratio of 5%, the fault coverage is on average 0.17% points higher than for the lower abort-limit. However, the proposed accurate ATPG increases fault coverage by another 11.34% points on average. For the higher abort-limit, the runtime of the commercial tool increases by on average seven times compared to the lower abort-limit.

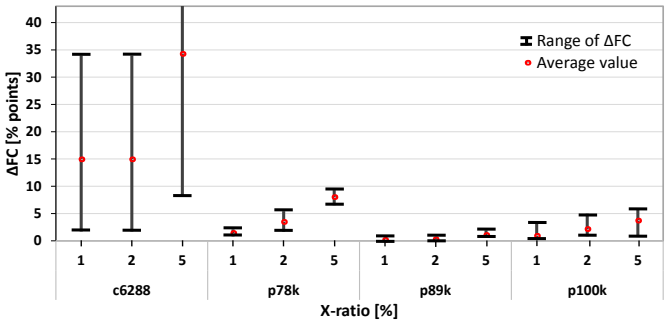

Fig. 10. Range of fault coverage increase of the proposed accurate ATPG compared to a commercial ATPG tool.

TABLE II
RESULTS OF THE PROPOSED ACCURATE ATPG IN CONTRAST TO A THREE-VALUED ATPG FOR TRANSITION-DELAY FAULTS.

Circuit	Gates	Faults	X-Ratio	3-val. SAT-based ATPG			Proposed QBF-based ATPG						
			[%]	FC [%]	UT [%]	UC [%]	**ΔFC [%pt.]**	**UT [%]**	Abort [%]	PD	Class. 1s	Class. 10s	Time [s]
s13207	7 951	23 972	1.0	81.32	8.30	10.38	**0.34**	**18.34**	0.00	151	2 446	12	51
			2.0	77.69	7.38	14.92	**0.34**	**21.97**	0.00	199	3 535	11	49
			5.0	64.13	7.25	28.63	**0.10**	**35.77**	0.00	356	6 839	1	54
s15850	10 211	28 686	1.0	83.33	7.41	9.25	**0.50**	**16.15**	0.02	28	2 571	4	70
			2.0	80.65	6.65	12.70	**0.76**	**18.57**	0.02	62	3 533	4	79
			5.0	67.98	5.63	26.39	**2.33**	**29.49**	0.21	402	7 110	68	736
s38584	21 462	66 504	1.0	89.47	5.16	5.37	**0.06**	**10.46**	0.00	325	3 523	12	88
			2.0	85.59	3.29	11.11	**0.13**	**14.25**	0.02	792	7 292	13	269
			5.0	77.54	2.25	20.22	**0.36**	**22.08**	0.03	1 383	13 230	18	351
s38417	23 537	68 744	1.0	94.59	1.39	4.03	**0.40**	**4.97**	0.05	219	2528	1	318
			2.0	87.67	1.85	10.48	**0.70**	**11.29**	0.34	889	6 651	1	21 24
			5.0	75.59	3.14	21.27	**2.03**	**21.71**	0.67	1 559	13 065	79	4 359
p78k	74 243	308 208	1.0	96.65	0.42	2.93	**1.58**	**1.73**	0.04	708	4 369	74	1 347
			2.0	93.80	0.41	5.80	**3.13**	**2.94**	0.14	1 637	8 394	246	4 632
			5.0	84.66	0.52	14.82	**7.91**	**6.82**	0.61	4 293	21 116	853	19 781
p89k	88 726	299 028	1.0	85.49	1.72	12.79	**0.20**	**13.99**	0.32	977	36 278	671	14 769
			2.0	73.69	1.39	24.92	**0.71**	**23.67**	1.93	3 973	65 568	1 471	65 151
			5.0	50.75	1.25	48.00	**4.25**	**39.74**	5.25	9 618	111 866	3 737	167 968
p100k	96 685	324 586	1.0	87.69	0.97	11.34	**0.62**	**11.09**	0.60	3 832	33 138	726	24 557
			2.0	82.97	0.72	16.31	**1.17**	**14.95**	0.91	5 479	46 315	1 274	34 856
			5.0	65.06	0.65	34.29	**1.45**	**30.59**	2.89	17 585	93 608	5 313	102 448

TABLE III
RESULTS OF THE PROPOSED ACCURATE ATPG COMPARED TO A STATE-OF-THE-ART COMMERCIAL ATPG TOOL.

Circuit	X-Ratio	Commercial ATPG				Proposed QBF-based ATPG				
	[%]	FC [%]	UT [%]	UC [%]	Time [s]	**ΔFC [%pt.]**	**UT [%]**	Time QBF 1s [s]	Time QBF 10s [s]	Total Runtime [s]
c6288	1.0	82.81	0.39	17.15	119	**14.57**	**2.51**	12	90	110
	5.0	61.66	0.39	38.81	229	**33.38**	**4.67**	30	240	297
p78k	1.0	97.30	0.00	2.71	10	**1.39**	**1.20**	531	2 739	3 656
	2.0	93.59	0.00	6.41	21	**3.43**	**2.74**	1 063	5 612	7 507
	5.0	84.24	0.00	15.76	43	**7.99**	**6.98**	3 624	19 067	26 989
p89k	1.0	91.94	0.82	7.23	183	**0.17**	**7.32**	5 079	13 766	22 694
	2.0	86.32	0.81	12.87	219	**0.26**	**12.82**	7 193	15 882	27 993
	5.0	71.74	0.78	27.48	79	**1.03**	**26.51**	7 497	18 142	32 943
p100k	1.0	95.28	0.21	4.51	198	**0.88**	**3.46**	2 102	10 094	15 474
	2.0	91.42	0.21	8.39	430	**2.12**	**6.01**	2 821	12 241	20 405
	5.0	80.35	0.19	19.51	1 098	**3.64**	**14.52**	7 875	38 833	65 154

F. Multiple Output and Value Difference Detection Modes

As described in Section V-F, the QBF-based test generation is extended to support the two detection modes $\text{Det}_{\text{multiple output}}$ and $\text{Det}_{\text{value difference}}$ to increase coverage in certain test applications. We evaluate the circuits c7552 and s13207 for up to 99% X-sources at the inputs for these two dedection modes. Figure 11 shows the increase of detected faults w. r. t. these two detection modes over definite detection (using a logarithmic scale for the y-axis).

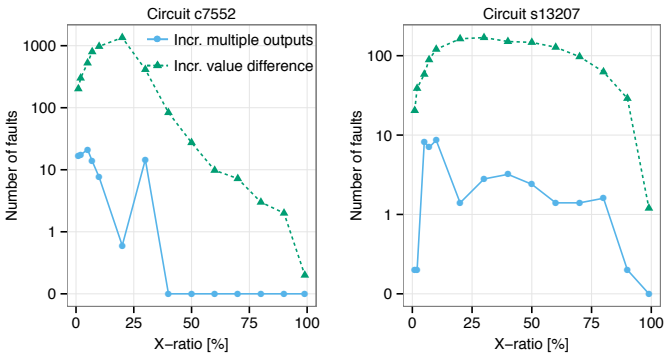

Fig. 11. Increase in detectable faults for detection modes allowing multiple outputs or value differences, compared to definite detection for different X-ratios.

For both circuits the relaxation of one fixed detecting output to 'multiple outputs' increases the fault coverage only slightly by at most 22 faults for the considered X-ratios. This corresponds to 0.88% of the untestable faults. The runtime increases by on average 50% compared to the runtime of the proposed ATPG restricting fault detection to only a single output. The benefit of allowing multiple outputs for fault detection during ATPG thus is rather small.

For the detection mode 'value difference' and circuit c7552, up to 1200 faults, i. e. up to 27.56% of the untestable faults, showed unknown but complementary values at at least one output providing good candidates for further post-processing.

G. Clustered X-Sources

In order to evaluate the impact of clustered X-sources, we performed additional experiments for the two largest circuits from Table I. In a first experiment (cf. Table IV), 5% of the pseudo-primary inputs out of one, and 10% of the pseudo-primary inputs out of two scan-chains are selected consecutively as X-sources. In a second experiment (cf. Table V), the X-sources are clustered by the input signal name: From the inputs sorted by name, a consecutive subset of 5% and 10% of the inputs is selected as X-sources. In the third experiment (cf. Table VI), we evenly distributed the X-sources over all inputs for 5% and 10%. Hence, for circuit p89k with 4683 inputs, 5% of the inputs are selected as X-sources and in iteration 1, we selected input $1, 21, 41, \ldots$ as X-sources.

The results in the Tables IV-VI show the average over 5 runs per circuit and X-ratio. Also for clustered X-sources, the results show the high pessimism in conventional test generation algorithms. For all experiments, the fault coverage increases significantly by accurate analysis compared to three-valued ATPG (column '3-v. SAT'). For experiment one, the fault coverage increases by up to 5.31% points, for experiment two by up to 8.00% points, and for experiment three by up to 8.84% points.

TABLE IV
DETECTED STUCK-AT FAULTS FOR DIFFERENT SCAN-CHAIN CONFIGURATIONS AS CLUSTERED X-SOURCES.

Circuit	Num. faults	X-Sources Chain	X-Sources [%]	3-v SAT FC [%]	QBF ΔFC [%pt.]	Pot. det. (PD)
p89k	186 645	1	5.0	63.98	1.82	22 576
		2	10.0	63.77	3.11	17 261
p100k	247 375	1	5.0	84.41	2.56	12 096
		2	10.0	68.60	5.31	26 190

TABLE V
DETECTED STUCK-AT FAULTS IN CASE X-SOURCES ARE CLUSTERED BY INPUT SIGNAL NAME.

Circuit	Num. faults	X-Sources [%]	3-v SAT FC [%]	QBF ΔFC [%pt.]	Pot. det. (PD)
p89k	186 645	5.0	81.52	0.82	3 390
		10.0	61.64	1.00	19 945
p100k	247 375	5.0	83.84	5.02	1 249
		10.0	75.01	8.00	9 532

TABLE VI
DETECTED STUCK-AT FAULTS IN CASE X-SOURCES ARE EVENLY DISTRIBUTED OVER THE INPUTS.

Circuit	Num. faults	X-Sources [%]	3-v SAT FC [%]	QBF ΔFC [%pt.]	Pot. det. (PD)
p89k	186 645	5.0	77.10	0.47	11 309
		10.0	46.80	2.62	39 875
p100k	247 375	5.0	80.87	1.37	20 887
		10.0	51.84	8.84	43 669

Similar to randomized X-sources, the proposed ATPG classifies a high fraction of undetectable faults as potentially detected. In experiment one, for circuit p100k and 5% of one scan-chain selected as X-sources, for 37.50% of the undetectable faults, a potentially detecting pattern is computed. Similar results are achieved in experiment two for circuit p89k and 10% of the inputs selected as X-sources. Thus, for 23.25% of the undetectable faults a potentially detecting pattern was found. For experiment three, up to 48.49% of the undetectable faults could be classified as potentially detected (p100k, 5% X-sources).

VIII. Conclusions

This paper proposed an accurate ATPG algorithm able to prove the testability or untestability of stuck-at and transition-delay faults in presence of unknown values. The algorithm combines hybrid two- and three-valued SAT-based test pattern generation, accurate fault simulation in presence of unknown

values, and QBF-based reasoning. The pessimism of conventional test generation algorithms in presence of unknown values is overcome by mapping the search for a test pattern to the satisfiability of a QBF.

In addition, different fault detection modes and increased robustness by using different QBF solver techniques are discussed. Finally, the first algorithm accurately computing patterns for all potentially detectable faults is presented.

The experimental results show that depending on circuit structure and X-sources, the fault coverage can be significantly increased by the proposed accurate analysis. For circuit p78k, the number of untested faults is more than halved by the proposed method.

ACKNOWLEDGEMENTS

The authors thank Linus Feiten and Karsten Scheibler (University of Freiburg) for their support, Jürgen Schlöffel (Mentor Graphics, formerly NXP) for supplying industrial benchmarks, and the reviewers for valuable comments. This work was partially supported by the German Research Foundation (DFG) under grants BE 1176/14-2 and WU 245/17-1 (Project ACCESS).

REFERENCES

[1] J. P. Roth, "Diagnosis of automata failures: A calculus and a method," *IBM J. Res. Dev.*, vol. 10, no. 4, pp. 278–291, july 1966.

[2] P. Goel, "An implicit enumeration algorithm to generate tests for combinational logic circuits," in *Proc. Fault Tolerant Computing Symposium*, 1980, pp. 145–151.

[3] H. Fujiwara and T. Shimono, "On the acceleration of test generation algorithms," *IEEE Transactions on Computers*, vol. C-32, no. 12, pp. 1137 –1144, 1983.

[4] T. Larrabee, "Test pattern generation using Boolean satisfiability," *IEEE Trans. CAD*, vol. 11, no. 1, pp. 4–15, jan 1992.

[5] P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli, "Combinational test generation using satisfiability," *IEEE Trans. CAD of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1167–1176, sep 1996.

[6] P. Muth, "A nine-valued circuit model for test generation," *IEEE Trans. on Computers*, vol. C-25, no. 6, pp. 630–636, june 1976.

[7] J. Marques-Silva and K. Sakallah, "Robust search algorithms for test pattern generation," in *Proc. International Symposium on Fault-Tolerant Computing (FTCS)*, June 1997, pp. 152–161.

[8] H. Chen and J. Marques-Silva, "TG-PRO: A new model for SAT-based ATPG," in *Proc. IEEE International High Level Design Validation and Test Workshop (HLDVT)*, Nov 2009, pp. 76–81.

[9] P. Flores, H. Neto, and J. Marques Silva, "An exact solution to the minimum size test pattern problem," in *Proc. International Conference on Computer Design (ICCD)*, 1998, pp. 510–515.

[10] A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and M. Hsiao, "Testing, verification, and diagnosis in the presence of unknowns," in *Proc. IEEE VLSI Test Symposium (VTS)*, 2000, pp. 263–268.

[11] C.-A. Chen and S. K. Gupta, "A satisfiability-based test generator for path delay faults in combinational circuits," in *Proc. ACM/IEEE Design Automation Conference (DAC)*, 1996, pp. 209–214.

[12] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloffel, "PASSAT: Efficient SAT-based test pattern generation for industrial circuits," in *Proc. IEEE Computer Society Annual Symposium on VLSI*. IEEE, 2005, pp. 212–217.

[13] R. Drechsler, S. Eggersglüss, G. Fey, A. Glowatz, F. Hapke, J. Schlöffel, and D. Tille, "On acceleration of SAT-based ATPG for industrial designs," *IEEE Trans. CAD*, vol. 27, no. 7, pp. 1329–1333, Jul. 2008.

[14] J. Carter, B. Rosen, G. Smith, and V. Pitchumani, "Restricted symbolic evaluation is fast and useful," in *Proc. IEEE International Conference on Computer-Aided Design (ICCAD)*, 1989, pp. 38 –41.

[15] S. Kundu, I. Nair, L. Huisman, and V. Iyengar, "Symbolic implication in test generation," in *Proc. Conference on European Design Automation*, 1991, pp. 492–496.

[16] M. Elm, M. A. Kochte, and H.-J. Wunderlich, "On determining the real output Xs by SAT-based reasoning," in *Proc. IEEE Asian Test Symposium*, 2010, pp. 39–44.

[17] H.-Z. Chou, K.-H. Chang, and S.-Y. Kuo, "Accurately handle don't-care conditions in high-level designs and application for reducing initialized registers," *IEEE Trans. CAD*, vol. 29, no. 4, pp. 646–651, 2010.

[18] C. Wilson, D. Dill, and R. Bryant, "Symbolic simulation with approximate values," in *Formal Methods in Computer-Aided Design*, ser. LNCS, W. Hunt and S. Johnson, Eds. Springer, 2000, vol. 1954, pp. 507–522.

[19] A. Biere, M. Heule, H. van Maaren, and T. Walsh, Eds., *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications 185. IOS Press, 2009.

[20] M. A. Kochte, M. Elm, and H.-J. Wunderlich, "Accurate X-propagation for test applications by SAT-based reasoning," *IEEE Trans. CAD*, vol. 31, no. 12, pp. 1908–1919, 2012.

[21] S. Hillebrecht, M. A. Kochte, H.-J. Wunderlich, and B. Becker, "Exact stuck-at fault classification in presence of unknowns," in *Proc. IEEE European Test Symposium (ETS)*, 2012, pp. 1–6.

[22] D. Erb, M. A. Kochte, M. Sauer, S. Hillebrecht, T. Schubert, H.-J. Wunderlich, and B. Becker, "Exact logic and fault simulation in presence of unknowns," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 19, no. 3, Jun. 2014.

[23] L. Zhang and S. Malik, "Conflict driven learning in a quantified Boolean satisfiability solver," in *Proc. IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2002, pp. 442–449.

[24] A. Biere, "Resolve and expand," in *Proc. 7th Int'l Conf. on Theory and Applications of Satisfiability Testing (Selected Papers)*, ser. Lecture Notes in Computer Science, vol. 3542. Springer, 2005, pp. 59–70.

[25] E. Giunchiglia, P. Marin, and M. Narizzano, "sQueezeBF: An effective preprocessor for QBFs based on equivalence reasoning," in *Theory and Applications of Satisfiability Testing*, ser. LNCS, vol. 6175. Springer, 2010, pp. 85–98.

[26] S. Hillebrecht, M. A. Kochte, D. Erb, H.-J. Wunderlich, and B. Becker, "Accurate QBF-based test pattern generation in presence of unknown values," in *Proc. Conf. on Design, Automation and Test in Europe (DATE)*, 2013, pp. 436–441.

[27] S. A. Cook, "The complexity of theorem-proving procedures," in *STOC*. ACM, 1971, pp. 151–158.

[28] L. J. Stockmeyer and A. R. Meyer, "Word problems requiring exponential time (preliminary report)," in *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, ser. STOC '73. New York, NY, USA: ACM, 1973, pp. 1–9.

[29] G. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in constructive mathematics and mathematical logic*, vol. 2, 1968.

[30] D. Erb, M. A. Kochte, M. Sauer, H.-J. Wunderlich, and B. Becker, "Accurate multi-cycle ATPG in presence of X-values," in *Proc. IEEE Asian Test Symposium (ATS)*, Nov 2013, pp. 245–250.

[31] F. Pigorsch and C. Scholl, "An AIG-based QBF-solver using SAT for preprocessing," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, June 2010, pp. 170–175.

[32] M. Soos, K. Nohl, and C. Castelluccia, "Extending SAT solvers to cryptographic problems," in *Theory and Applications of Satisfiability Testing-SAT 2009*. Springer, 2009, pp. 244–257.

[33] J. Marques-Silva, "The impact of branching heuristics in propositional satisfiability algorithms," in *Progress in Artificial Intelligence*. Springer, 1999, pp. 62–74.

[34] G. Audemard and L. Simon, "Refining restarts strategies for SAT and UNSAT," in *Principles and Practice of Constraint Programming*. Springer, 2012, pp. 118–126.

[35] E. Giunchiglia, M. Narizzano, and A. Tacchella, "QUBE: A system for deciding quantified Boolean formulas satisfiability," in *Proc. of IJCAR*, 2001.

[36] T. Schubert and S. Reimer, "antom," 2015, https://projects.informatik.uni-freiburg.de/projects/antom.

[37] S. Reimer, F. Pigorsch, C. Scholl, and B. Becker, "Enhanced integration of QBF solving techniques." in *MBMV*, 2012, pp. 133–143.

[38] M. Luby, A. Sinclair, and D. Zuckerman, "Optimal speedup of Las Vegas algorithms," *Information Processing Letters*, vol. 47, no. 4, pp. 173–180, 1993.

[39] F. Pigorsch and C. Scholl, "Exploiting structure in an AIG based QBF solver," in *Conf. on Design, Automation and Test in Europe*, April 2009.

[40] A. Mishchenko, S. Chatterjee, R. Jiang, and R. K. Brayton, "FRAIGs: A unifying representation for logic synthesis and verification," EECS Dept., UC Berkeley, Tech. Rep., 03 2005.

[41] D. Erb, K. Scheibler, M. A. Kochte, M. Sauer, H.-J. Wunderlich, and B. Becker, "Test Pattern Generation in Presence of Unknown Values Based on Restricted Symbolic Logic," in *Proc. IEEE International Test Conference (ITC)*, 2014, pp. 1–10.

Dominik Erb received his master degree in computer science at University of Freiburg in January 2013. Currently he is a PhD Student at the group of computer architecture of Prof. Bernd Becker in Freiburg. His current research interests include automatic test pattern generation in presence of unknown values as well as interconnect open defects, defect-based testing and fault-diagnosis.

Michael A. Kochte received a Diploma in Computer Science in 2005 and a Dr. rer. nat. (Ph.D.) degree from the University of Stuttgart in 2014. Since 2007 he's been working at the Institute for Computer Architecture and Computer Engineering of the University of Stuttgart on hardware infrastructure and security, test generation, and VLSI dependability.

Sven Reimer received a Diploma in computer science from University of Freiburg in 2009. Since 2009 he is a PhD student at the group of computer architecture of Prof. Bernd Becker in Freiburg. His research interests include the development and utilization of formal methods such as SAT, MaxSAT, and QBF in various applications.

Matthias Sauer studied Computer Science at the University of Freiburg and received his master degree in July 2010 and his PhD in December 2013. He is currently working in the computer architecture group of Prof. Bernd Becker at the University or Freiburg on the application of formal methods to hardware security and the timing analysis of digital circuits.

Hans-Joachim Wunderlich received a Diploma in Mathematics from the University of Freiburg in 1981 and the Dr. rer. nat. (Ph.D.) from the University of Karlsruhe in 1986. Since 1991 he has been a full Professor and since 2002 he has been the director of the Institute of Computer Architecture and Computer Engineering (ITI) at the University of Stuttgart. He is editor of various international journals and program committee member of a variety of IEEE conferences on design and test of electronic systems. Hans-Joachim Wunderlich has published more than 250 reviewed scientific papers in journals, book and conference proceedings. His research interests include test, reliability and fault tolerance of microelectronic systems. Hans-Joachim Wunderlich is a fellow of IEEE.

Bernd Becker is a Full Professor (Chair of Computer Architecture) at the Faculty of Engineering, University of Freiburg, Germany. Prior to joining the University of Freiburg in 1995, he was with J. W. Goethe-University Frankfurt as an associate professor for complexity theory and efficient algorithms. His research activities include design, test and verification methods for embedded systems and nano-electronic circuitry. He is a Co-Speaker of the DFG Transregional Collaborative Research Center "Automatic Analysis and Verification of Complex Systems (AVACS)" and a Director of the Centre for Security and Society, University of Freiburg. He is a fellow of IEEE and Member of Academia Europaea.