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

Stefan Hillebrecht*, Michael A. Kochte†, Dominik Erb*, 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 may emerge during the design process as well as during system operation and test application. Sources of X-values are for example black boxes, clock-domain boundaries, analog-to-digital converters, or uncontrolled or uninitialized sequential elements.**

**To compute a detecting pattern for a given stuck-at fault, well defined logic values are required both for fault activation as well as for fault effect propagation to observing outputs. In presence of X-values, classical test generation algorithms, based on topological algorithms or formal Boolean satisfiability (SAT) or BDD-based reasoning, may fail to generate testing patterns or to prove faults untestable.**

**This work proposes the first efficient stuck-at fault ATPG algorithm able to prove testability or untestability of faults in presence of X-values. It overcomes the principal inaccuracy and pessimism of classical algorithms when X-values are considered. This accuracy is achieved by mapping the test generation problem to an instance of quantified Boolean formula (QBF) satisfiability. The resulting fault coverage improvement is shown by experimental results on ISCAS benchmark and larger industrial circuits.**

***Index Terms*—Unknown values, test generation, ATPG, QBF**

## I. Introduction

Unknown (X) values emerge during design and test generation, as well as during operation and test application. X-values are caused by unspecified or black boxes in the design, or—during test—by uncontrolled sequential elements, at clock domain crossings or A/D boundaries for example. X-values in the circuit compromise the testability of faults since both, fault activation and propagation, require well defined values at the fault site and along the propagation path, respectively.

Deterministic test generation algorithms (ATPG) for stuck-at faults are typically based on topological search such as the D-algorithm [1], PODEM [2] or FAN algorithm [3], or 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. In the same way, SAT-based test generation algorithms have been extended to process signal states with X-values [7, 8]. Restricted symbolic simulation [9] 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 [10], 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 these $n$-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—and thus 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.

978-3-9815370-0-0/DATE13/©2013 EDAA

Fig. 1 shows an example of a circuit with one X-source at line $b$ which reconverges at line $q$. Classical ATPG algorithms fail to compute a test pattern for any stuck-at fault in this circuit. However, the stuck-at-0 and stuck-at-1 faults at $c$ and $q$ are detectable with the two patterns $(a, c) = (1, 0)$ and $(a, c) = (1, 1)$. For the presented example, restricted symbolic logic based ATPG [10] also fails to generate a test.

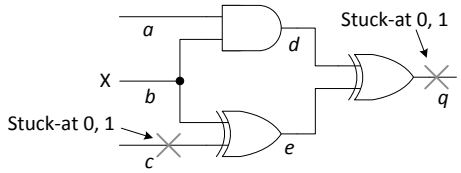


Fig. 1: Circuit with X-source at line $b$

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 [11–13]. These methods employ Boolean satisfiability (SAT), quantified Boolean formula (QBF) reasoning, or symbolic computation by binary decision diagrams (BDDs).

Accurate fault simulation can be performed even for large circuits by a combination of heuristics and SAT reasoning [14]. 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 (Verification of a guessed solution, i.e. fault simulation, is an NP-complete problem).

In general, solving this problem using Boolean satisfiability requires $2^n$ different SAT instances of the circuit to reflect all possible assignments at $n$ X-sources. Quantified Boolean formulae [15], where variables are existentially or universally quantified, allow a succinct representation of these problem instances. The recent advances in the performance of QBF solvers, for example conflict driven learning [16], resolution and expansion based decision algorithms [17] or preprocessing [18], enable exact reasoning about fault testability in presence of Xs even for larger circuits. QBF-based reasoning has already been applied to circuit debugging in [19].

In this work, we propose a mapping of the test generation problem in presence of multiple X-sources to the QBF domain

to exactly determine the testability of a fault. Runtime is kept reasonable by use of efficient two- and three-valued SAT-based ATPG, and accurate fault simulation of generated patterns. QBF-based reasoning is only invoked when absolutely necessary.

Section II gives a formal problem statement and required definitions, followed by the presentation of the proposed algorithm in section III. Section IV presents experimental results on ISCAS benchmark circuits and NXP circuits. Section V summarizes the paper.

## II. Terminology

In 3-valued logic, the three symbols $\{0, 1, \mathrm{X}\}$ are used to represent logic value 0 (logic-0), logic value 1 (logic-1) and an unknown state, i. e., either logic-0 or logic-1. Signals at which unknown values originate are called X-sources.

In this work two types of stuck-at fault detection in combinational or full scan circuits are distinguished: Definite detection (DD) and potential detection (PD). A stuck-at fault $f$ is DD if and only if an output $o$ exists where the fault effect is observable independent of the logic value assignment to the X-sources. Let the functions $v^G(p, s)$ and $v^f(p, s)$ return the logic value of signal $s$ under pattern $p$ in the fault free circuit and the circuit under fault $f$, respectively, in presence of X-values. Then, the definite detection of fault $f$ under pattern $p$ is given as

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

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

Stuck-at fault $f$ is potentially detected if an observable output $o$ exists where the fault effect can be deterministically measured for at least one logic value assignment to the X-sources:

$$\begin{aligned}\mathrm{PD}^f(p) := \neg\mathrm{DD}^f(p) \wedge \exists o \in O :\\ v^G(p,o) \in \{1,0\} \wedge v^f(p,o) = \mathrm{X}.\end{aligned} \quad (2)$$

## III. Accurate X-aware Test Pattern Generation

### A. Overview

The proposed ATPG algorithm is able to prove the testability of stuck-at faults in presence of X-values. It combines accurate fault simulation [14], incremental SAT-based test generation with two- and three-valued encoding, and QBF reasoning to efficiently analyze the faults.

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 (c.f. Fig. 2):

1) No structural dependence on the X-sources: Neither the justification cone of the fault, nor its propagation cone have any dependence 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.

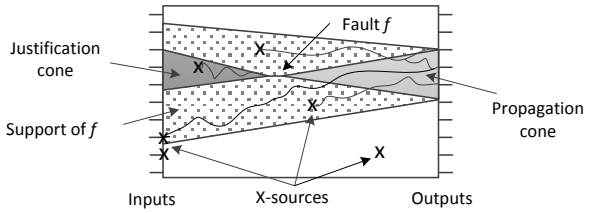


Fig. 2: Fault $f$ and its justification and propagation cone.

4) The justification cone is driven exclusively by X-sources.

The faults are processed as shown in Fig. 3: First, faults in Group 1 and Group 2 are handled by SAT-based ATPG based on a two-value signal encoding where X-values are not considered (①). Untested faults that were not proven untestable in this step, and the faults of Group 3 are then processed by SAT-based ATPG based on pessimistic three-value signal encoding (②). For the faults for which no test pattern is found by three-value signal encoding, a QBF is constructed and analyzed using a QBF solver (④). For the faults in Group 4, a topological untestability check is conducted (③). If this test cannot prove a fault untestable, the fault is also analyzed by the QBF solver (④). For the generated test patterns, accurate fault simulation (⑤) is performed. The following sections explain the steps in detail.

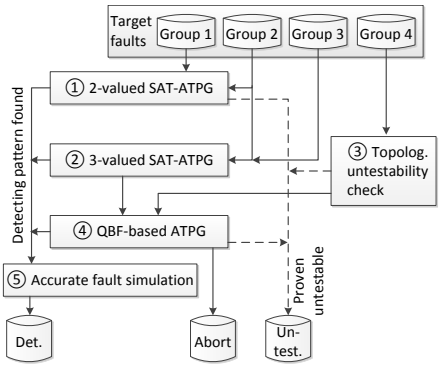


Fig. 3: Overview of the proposed QBF-based ATPG in presence of X-values.

### B. Two-valued SAT-based ATPG

Faults in Group 1 have no X-dependencies. Thus, two-valued or binary signal encoding is sufficient to generate a testing pattern or prove fault untestability. For faults in Group 2, both fault activation and propagation to outputs without X-dependence can be analyzed accurately using a binary encoding.

In SAT-based ATPG, the Boolean function of the fault-free and faulty circuit are represented as formula in conjunctive normal form (CNF), or union of clauses. For a netlist, this representation is efficiently obtained using the Tseitin transformation [20]. To reduce the number of clauses, only the propagation cone or cone-of-influence of the faults is modeled and—at the circuit outputs—compared with the outputs of the fault-free circuit model. The search for a test pattern is sped up by introducing D-chains which explicitly model propagation paths of the fault effect [4].

In the two-valued encoding, the state of each signal in the fault-free and faulty circuit is modeled by a single binary variable respectively.

The SAT instances are constructed incrementally during fault analysis. The state-of-the-art incremental SAT solver of [21] is used to solve the instances. This reduces the construction overhead of the SAT instance and allows to exploit knowledge learnt during the analysis in previous steps.

For each fault, first the justification cone is modeled and it is checked whether the fault can be activated. This computation is accurate since for Groups 1 and 2, the fault site does not depend on X-sources. If the fault cannot be activated, it is untestable. If it can be activated, we proceed analyzing the fault propagation to the circuit outputs.

The SAT instance is now incrementally extended with circuit outputs reachable from the fault site. Here, only reachable outputs without X-dependence are considered. The outputs are processed with increasing structural depth. For a reachable output, its justification cone, i.e. all gates in its transitive fanin, is added to the SAT instance of the fault-free and faulty circuit. The justification cones of the reachable outputs are computed efficiently in parallel for up to 64 outputs by a single forward and backward traversal of the circuit. For signals in the propagation cone of the circuit, the clauses for modeling of D-chains are added as well.

Once the output, its justification cone and D-chain have been modeled, it is sufficient to check whether the output can detect a difference between the fault-free and faulty circuit under the considered fault. This is equivalent to checking whether the $d$ variable of the D-chain at the output can be asserted ($d = 1$). This condition is added as assumption, i.e. a temporary condition evaluated only during the current satisfiability analysis in the SAT solver.

If the SAT solver finds a satisfying assignment to the circuit inputs, the fault is detected. If there is no satisfying assignment, the complement $d = 0$ of the assumption, is statically added to the SAT instance to prune the solver's search space in the analysis of the remaining outputs.

A fault of Group 1 without any X-dependencies is marked untestable if the SAT instance is unsatisfiable for all reachable outputs. For faults in Group 2, only outputs without X-dependencies are considered. If the fault is not observable at any of these outputs, the fault is processed using a three-valued analysis as explained in the next section.

### C. Three-valued SAT-based ATPG

Faults of Group 2 which are not proven testable or untestable in the previous step and all faults of Group 3 are now analyzed using three-valued SAT-based ATPG. Three-valued SAT-based ATPG allows to search for test patterns in presence of X-sources by modeling the three signal states $\{0, 1, X\}$ both in the fault-free and faulty circuit. This modeling generates a test pattern if the accurate evaluation of reconvergences of X-valued signals is not required, i.e. fault activation and propagation along a D-chain are achieved by avoiding or blocking X-values in the support of the fault.

In the three-valued encoding, the state of a signal $s$ is expressed by two variables $(s_1, s_2)$ such that logic-0 corresponds to $(0, 1)$, logic-1 to $(1, 0)$ and the X-state to $(0, 0)$. An additional clause $(\neg s_1, \neg s_2)$ is added for each signal $s$ to restrict the states to $\{0, 1, X\}$.

The SAT instance is constructed from the circuit using the Tseitin transformation. Compared to the two-valued encoding, the number of clauses per gate is more than doubled. The instance construction and evaluation is performed incrementally by first considering the activation at the fault site, followed by the analysis of observing outputs. If the fault site cannot be justified, the fault is later processed using the QBF solver. For the propagation path, the D-chain clauses enforce that a propagating signal has well-defined and complementary binary logic values in the fault-free and faulty circuit.

For faults of Group 2 for which no test pattern was found in the previous step, only the outputs with X-dependencies are processed. If no detecting pattern is found, the fault is analyzed in by the QBF-based ATPG.

### D. Topological Untestability Check

If the fault site of a fault cannot be justified to a known logic value, the fault cannot be definitely detected. For faults of Group 4, the justification cone only depends on X-sources. X-canceling in the justification cone of the fault site can only occur at signal reconvergences. A simple tracing of branching signals is performed to check for reconvergences. If none are found in the justification cone, the fault site of faults in Group 4 cannot be justified. Consequently, they are untestable w.r.t. the definite detection criterion of section II. If reconvergences are found, the fault may be testable and needs to be analyzed using the QBF solver.

### E. QBF-based ATPG

All faults for which testability or untestability has not been proven yet are subject to ATPG based on quantified Boolean satisfiability. A problem instance is modeled as quantified Boolean formula (QBF) and a QBF solver is employed to search for a satisfying model (i.e. a test pattern) or to prove untestability of the fault.

*1) Quantified Boolean Formulae and QBF Satisfiability:* A quantified Boolean formula is a Boolean formula in which the variables are quantified or bound by the existential ($\exists$) or universal ($\forall$) quantifier [15]. In the following we assume that all variables are bound. A QBF can be transformed into the prenex normal form where all quantifiers are grouped together and precede an unquantified Boolean formula, called the matrix or kernel.

A QBF solver is able to process problem instances, typically given as a QBF in prenex normal form where the matrix is in conjunctive normal form (CNF). The QBF solver searches for a model or assignment to the existentially quantified variables that satisfies the QBF, or proves that no such model exists. 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. We will see in the following sections that the particular problem of ATPG in presence of X-values is easier since the number of quantifier alternations is limited. Still, ATPG in presence of X-values is at least NP-hard.

*2) Construction of the CNF Matrix for ATPG:* The matrix of the QBF in CNF is constructed similar to a classical two-valued SAT-based ATPG instance. The state of a signal is modeled by a single binary variable.

The gates of the fault-free circuit $C^G$ and the propagation cone $C_P^f$ of the fault $f$ in the faulty circuit are Tseitin-transformed into CNF. The propagation cone of the fault site is transformed in one step. Fault propagation conditions from the fault site to the outputs are added in form of D-chains. The $d$-variables of the D-chains at the outputs in the propagation cone are added to a single clause (disjunction) and added to the CNF instance. This ensures that the fault effect is observable at at least one output:

$$\text{CUT}_{CNF} = \text{Tseitin}(C^G) \wedge \text{Tseitin}(C_P^f) \wedge (\text{D-chain clauses})$$

The behavior of X-sources is captured by the universal quantification as discussed in the next section.

*3) Variable Quantification:* All controllable inputs $I$ to the circuit are existentially quantified. All X-sources $X$ are universally quantified. Here it is important to respect the scope of quantification, i.e. the sequence of quantifier alternations. In particular, we search for *one* test pattern that satisfies the matrix for *all* possible assignments to the X-sources. Thus, the quantification of circuit inputs precedes the quantification of X-sources.

To bind all variables of the matrix, all internal signals $S$ as well as the variables $D$ of the D-chains are existentially quantified. This results in the following QBF:

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

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 of the fault according to equation (1) of section II, we need to constrain the solution space by limiting the detecting outputs to a single fixed one. That is, for all possible assignments to the X-sources, the fault effect must be observable at one particular output.

This constraint is included by additional variables $o_i$ for the outputs in the propagation cone. Variable $o_i$ shall be *true* 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 $o_i$ is *true* and thus, the fault is always observable at at least one output. The relation between $o_i$ and the modeled D-chains are established by the additional implication per output $(o_i \rightarrow d_i)$. All the variables $O = \bigcup_i o_i$ are existentially quantified preceding the universal quantification of the X-sources.

$$\exists \boldsymbol{O} \exists I \forall X \exists S \exists D : \Big(\text{CUT}_{CNF} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i)\Big)$$

This enforces a fixed detecting output over all assignments to X-sources. However, the observable difference, i.e. the signal state in the fault-free and faulty circuit at that output is still allowed to be one of the four possibilities $(0/1), (1/0), (x_i, \neg x_i), (\neg x_i, x_i)$. The latter two cases correspond to situations where an output always shows complementary states in fault-free and faulty circuit for all assignments to the X-sources, but the value in the fault-free and faulty circuit are not stable for all assignments to X-sources.

The definite detection criterion requires known binary values at the observing output. This is achieved by forcing the QBF solver to search for one stable logic value of the observing output in the fault-free circuit. The two additional variables $v_i^0, v_i^1$ per output represent which stable value a detecting output has. If $v_i^0$ ($v_i^1$) is true, output $i$ has the stable value logic-0 (logic-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 state of output $i$ in the fault-free circuit.

With the implication $(o_i \rightarrow (v_i^0 \vee v_i^1))$ for output $i$, and existential quantification of the variables $v_i^0, v_i^1 \in V$, we obtain the following QBF:

$$\exists O \exists \boldsymbol{V} \exists I \forall X \exists S \exists D : \Big(\text{CUT}_{CNF} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i) \wedge \bigwedge_i \big((o_i \rightarrow (v_i^0 \vee v_i^1)) \wedge (v_i^0 \rightarrow \neg s_i) \wedge (v_i^1 \rightarrow s_i)\big)\Big)$$

This QBF is satisfiable if and only if a fault is testable according to the definite detection condition of section II. If the formula is not satisfiable, no test pattern exists.

In this formula, there are two quantifier alternations. In the polynomial time hierarchy, this corresponds to the complexity class $\sum_3^P$ which is a subset of the PSPACE-complete complexity class [22].

### F. Accurate Fault Simulation

Fault simulation is used to find all faults detected by the generated patterns. We employ the fault simulation algorithm of [14] which is able to accurately compute the detectable stuck-at faults of a pattern in presence of X-values. This fault simulation algorithm is summarized here for completeness.

The fault simulation algorithm firstly computes the exact 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 exact 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 the simulation results already allow to classify the fault as definite detection, 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.

## IV. Evaluation

The proposed algorithm is implemented in C and uses the incremental SAT solver Antom [21] as well as the QBF solver QuBE [23]. We evaluated the algorithm on benchmark and industrial circuits. The experiments are conducted on an Intel Xeon CPU with 2.8 GHz.

### A. Experimental Setup

We consider full-scan circuits of the largest ISCAS'85 and ISCAS'89 circuits as well as three larger industrial designs from NXP (named p*k). We assume that a fixed and randomly selected subset of circuit inputs generates X-values. Three different subsets of X-source inputs are generated per circuit. The reported results are the rounded average over these three experiments per circuit.

For each circuit the collapsed set of stuck-at faults is computed. Then, fault simulation of 1024 random patterns is performed for each set of X-sources to find easily detectable faults. For the remaining random pattern resistant faults, the proposed ATPG algorithm is conducted.

### B. Results

Table I shows the results of the accurate ATPG algorithm. For each circuit the table lists the number of primary and pseudo primary inputs, the size in number of complex gates and the number of collapsed stuck-at faults. Per circuit, we conduct the 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 number of definitely ('DD') and potentially ('PD') detected faults, as well as the number of aborted faults as reported by a commercial X-aware ATPG tool. Please note that due to the inaccuracy of $n$-valued ATPG algorithms, the number of potentially detected faults reported by the commercial tool may over- or underestimate the actual number.

Columns 8 to 14 show the results of the proposed algorithm. Columns '$\Delta$DD' and '$\Delta$FC' show the increase of detected faults and the increase in percent points of fault coverage compared to the commercial tool. Column 'PD' lists the number of potentially detected faults.

The results show that the accurate ATPG algorithm is able to generate detecting patterns for a higher number of faults than classical algorithms. With higher number of X-sources, up to 24% higher fault coverage can be achieved due to the accurate analysis. For the larger industrial circuits, up to 7% higher fault coverage is achieved.

Column 12 ('Class. 1s') and 13 ('Class. 10s') list the number of faults which are classified by the QBF solver using a timeout of 1 second respectively 10 seconds. Most faults can be quickly classified as testable or untestable. Only for a small fraction, in average 7.4%, the QBF solver must be restarted with higher timeout. When processing the aborted faults with a timeout of 10 seconds, only few additional faults can be classified, and hardly ever is a fault then proven testable. Depending on circuit structure and number of X-sources, some faults cannot be classified even with a timeout of 10 seconds (shown in column 'Abort').

The last column of the table ('Time') lists the runtime in seconds of the test pattern algorithm including accurate fault simulation, 2-valued and 3-valued SAT-based ATPG and QBF-based ATPG. The runtime is dominated by the QBF processing effort for hard faults with the timeout of 10 seconds.

We investigate the pessimism of classical ATPG for higher X-ratios on two circuits, c7552 and s13207. Fig. 4 shows the increase in the number of definitely detectable faults ($\Delta$DD) for the two circuits for X-ratios from 1 to 99%. The figure also shows the absolute number of potentially detected faults (PD) and the number of aborted faults with a timeout of 10 seconds.

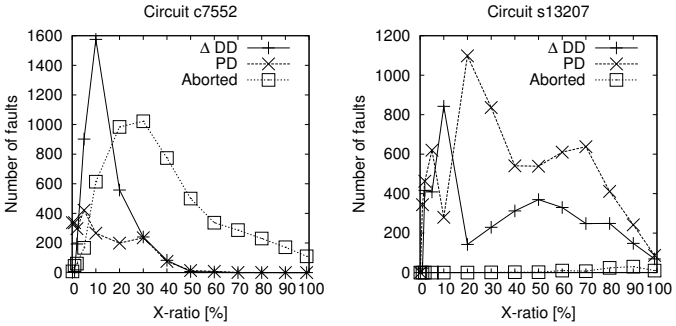


Fig. 4: Increase in definitely detectable faults ($\Delta$ DD) compared to classical ATPG depending on the X-ratio.

For all investigated X-ratios, the proposed algorithm generates testing patterns for a large number of faults which were classified as untestable or aborted by the pessimistic classical ATPG algorithm.

The results also show a relatively high number of aborted faults for circuit c7552 which results from the depth-first search strategy of the QBF solver. This strategy seems to cause high runtimes for circuits with XOR logic, such as c7552 with parity check structures.

The pessimism in the classical ATPG algorithm is highest for circuits with many reconverging paths such as the multiplier c6288, and circuits with XOR logic like c7552. Here fault coverage increases by up to 24 (c6288) respectively 9 percent points (c7552) for an X-ratio of 5%. For the industrial circuits, the accurate QBF-based ATPG algorithm increases fault coverage by approximately 2 percent points averaged over X-ratios of 1, 2 and 5%. For circuit p78k and an X-ratio of 1%, the number of undetected faults is reduced by more than 50% and fault coverage increases from 97.9% to 98.9%.

## V. Conclusions

This paper proposed the first ATPG algorithm able to prove testability or untestability of stuck-at faults in presence of unknown values. The algorithm combines incremental 2- and 3-valued SAT-based test pattern generation, accurate fault simulation in presence of unknown values, and QBF-based test generation.

The algorithm is accurate and overcomes the pessimism of classical ATPG algorithms when unknown or unspecified values need to be considered. The experiments assessed the existing pessimism in a state-of-the-art commercial ATPG.

The results show that depending on circuit structure and X-sources, the fault coverage can be significantly increased by the proposed accurate analysis.

## ACKNOWLEDGMENT

The authors thank Paolo Marin, Matthias Sauer and Tobias Schubert for support with the SAT and QBF solvers Antom and QuBE. This work was partially supported by the German Research Foundation (DFG) under grants BE 1176/14-2, WU 245/5-2 and WU 245/11-1.

## 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 Transactions on Computers*, vol. C-25, no. 6, pp. 630–636, june 1976.

[7] 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.

[8] A. Jain, V. Boppana *et al.*, "Testing, verification, and diagnosis in the presence of unknowns," in *Proc. IEEE VLSI Test Symposium (VTS)*, 2000, pp. 263–268.

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

[10] S. Kundu, I. Nair *et al.*, "Symbolic implication in test generation," in *Proc. Conference on European Design Automation*, 1991, pp. 492–496.

[11] 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.

[12] 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.

[13] 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.

[14] S. Hillebrecht, M. A. Kochte *et al.*, "Exact stuck-at fault classification in presence of unknowns," in *Proc. IEEE European Test Symposium (ETS)*, 2012, pp. 1–6.

[15] H. K. Büning and U. Bubeck, *Handbook of Satisfiability*, ser. Frontiers in Artificial Intelligence and Applications 185. IOS Press, 2009, ch. Theory of quantified Boolean formulas.

[16] 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.

[17] 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.

[18] 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. Springer, 2010, vol. 6175, pp. 85–98.

[19] A. Sülflow, G. Fey, and R. Drechsler, "Using QBF to increase accuracy of SAT-based debugging," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS)*, 2010, pp. 641–644.

[20] G. Tseitin, "On the complexity of derivation in propositional calculus," *Studies in constructive mathematics and mathematical logic*, vol. 2, no. 115-125, pp. 10–13, 1968.

[21] T. Schubert, M. Lewis, and B. Becker, "Antom—solver description," *SAT Race*, 2010.

[22] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA, 1979.

[23] E. Giunchiglia, P. Marin, and M. Narizzano, "QuBE7.0 System Description," *JSAT*, vol. 7, no. 2-3, pp. 83–88, 2010.

TABLE I: RESULTS OF THE QBF-BASED ATPG COMPARED TO A STATE-OF-THE-ART COMMERCIAL ATPG

| Circuit | Inputs | Gates | Faults | X ratio [%] | Commercial ATPG DD | PD | Abort | Proposed QBF-based ATPG ΔDD | **ΔFC [%]** | PD | Abort | Class. 1s | Class. 10s | Time [s] |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| c6288 | 32 | 2 416 | 8 704 | 1.0 | 7 523 | 287 | 336 | 963 | **11.06** | 23 | 25 | 143 | 20 | 406 |
| | | | | 5.0 | 6 134 | 350 | 935 | 2 123 | **24.39** | 57 | 74 | 566 | 49 | 1 207 |
| c7552 | 207 | 3 513 | 9 756 | 1.0 | 9 048 | 386 | 42 | 8 | **0.08** | 337 | 45 | 595 | 42 | 726 |
| | | | | 2.0 | 8 684 | 351 | 154 | 193 | **1.98** | 298 | 60 | 737 | 53 | 1 026 |
| | | | | 5.0 | 5 941 | 1 116 | 557 | 902 | **9.24** | 422 | 163 | 2 629 | 203 | 3 011 |
| s09234 | 247 | 5 597 | 13 892 | 1.0 | 12 178 | 23 | 0 | 73 | **0.53** | 139 | 0 | 1 349 | 0 | 25 |
| | | | | 2.0 | 11 049 | 51 | 0 | 88 | **0.63** | 225 | 0 | 2 546 | 0 | 43 |
| | | | | 5.0 | 9 435 | 159 | 0 | 196 | **1.41** | 321 | 0 | 3 726 | 0 | 78 |
| s13207 | 650 | 7 951 | 20 094 | 1.0 | 18 308 | 8 | 0 | 23 | **0.11** | 344 | 0 | 1 480 | 0 | 44 |
| | | | | 2.0 | 16 902 | 17 | 0 | 417 | **2.07** | 464 | 0 | 2 542 | 1 | 64 |
| | | | | 5.0 | 15 714 | 30 | 0 | 409 | **2.03** | 620 | 0 | 3 322 | 0 | 71 |
| s15850 | 600 | 9 772 | 24 543 | 1.0 | 23 077 | 5 | 0 | 118 | **0.48** | 67 | 7 | 1 082 | 15 | 269 |
| | | | | 2.0 | 22 336 | 20 | 0 | 55 | **0.22** | 465 | 53 | 1 670 | 15 | 872 |
| | | | | 5.0 | 20 466 | 98 | 0 | 91 | **0.37** | 532 | 52 | 3 223 | 22 | 887 |
| s35932 | 1 763 | 16 065 | 51 649 | 1.0 | 45 552 | 0 | 0 | 0 | **0.00** | 35 | 0 | 1 904 | 0 | 31 |
| | | | | 2.0 | 44 456 | 0 | 0 | 0 | **0.00** | 66 | 0 | 3 366 | 0 | 41 |
| | | | | 5.0 | 41 369 | 0 | 0 | 0 | **0.00** | 160 | 0 | 7 247 | 0 | 68 |
| s38417 | 1 524 | 22 179 | 56 325 | 1.0 | 54 296 | 57 | 0 | 24 | **0.04** | 268 | 40 | 1 660 | 43 | 923 |
| | | | | 2.0 | 52 307 | 173 | 0 | 136 | **0.24** | 936 | 99 | 3 122 | 83 | 1 957 |
| | | | | 5.0 | 48 758 | 462 | 0 | 297 | **0.53** | 1 732 | 109 | 5 458 | 182 | 2 698 |
| s38584 | 1 462 | 19 253 | 53 845 | 1.0 | 48 995 | 33 | 0 | 253 | **0.47** | 686 | 0 | 2 655 | 0 | 162 |
| | | | | 2.0 | 47 911 | 68 | 0 | 238 | **0.44** | 461 | 0 | 3 468 | 2 | 175 |
| | | | | 5.0 | 44 387 | 205 | 0 | 296 | **0.55** | 312 | 0 | 6 481 | 7 | 256 |
| p45k | 3 739 | 39 786 | 107 814 | 1.0 | 103 523 | 301 | 78 | 557 | **0.52** | 63 | 75 | 3 103 | 56 | 3 801 |
| | | | | 2.0 | 101 729 | 636 | 106 | 736 | **0.68** | 167 | 168 | 4 199 | 97 | 5 399 |
| | | | | 5.0 | 92 530 | 1 946 | 111 | 1 083 | **1.00** | 220 | 233 | 11 723 | 202 | 7 750 |
| p78k | 3 148 | 74 243 | 225 476 | 1.0 | 220 773 | 412 | 2 | 2 204 | **0.98** | 415 | 237 | 1 226 | 308 | 4 494 |
| | | | | 2.0 | 214 351 | 1 053 | 13 | 5 495 | **2.44** | 848 | 608 | 2 940 | 564 | 10 480 |
| | | | | 5.0 | 192 281 | 2 524 | 58 | 16 644 | **7.38** | 2 091 | 2 075 | 9 163 | 1 480 | 36 928 |
| p100k | 5 902 | 90 712 | 247 376 | 1.0 | 241 460 | 952 | 355 | 958 | **0.39** | 203 | 369 | 3 097 | 187 | 31 041 |
| | | | | 2.0 | 232 415 | 2 172 | 572 | 3 069 | **1.24** | 341 | 620 | 9 102 | 286 | 48 143 |
| | | | | 5.0 | 201 284 | 5 641 | 1 500 | 7 462 | **3.02** | 988 | 1 376 | 32 790 | 675 | 77 170 |