Modeling Unknown Values in Test and Verification

Bernd Becker, Matthias Sauer, Christoph Scholl, and Ralf Wimmer

Albert-Ludwigs-Universität Freiburg, Germany
{becker | sauerm | scholl | wimmer}@informatik.uni-freiburg.de

Abstract. With increasing complexities and a component-based design style the handling of unknown values (e. g., at the interface of components) becomes more and more important in electronic design automation (EDA) and production processes. Tools are required that allow an accurate modeling of unknowns in combination with algorithms balancing exactness of representation and efficiency of calculation. In the following, state-of-the-art approaches are described that enable an efficient and successful handling of unknown values using formal techniques in the areas of Test and Verification.

1 Introduction

Unknown (X) values increasingly emerge in different phases of the design and production process, and have to be handled by corresponding electronic design automation (EDA) tools. Examples include unspecified inputs or black boxes in the design, uncontrolled sequential elements, clock domain crossings or A/D boundaries. In all of these cases, the logic value of a signal is not defined and hence, only partial information on the circuit is available.

In the following, we describe current state-of-the-art approaches that enable (in principle) an exact handling of such unknown values using formal techniques in two fundamental areas of the design process, i. e., Test and Verification. In addition, efficient methods to trade off quality and computation times of the analysis are reported.

1.1 Unknown Values in Circuit Test

Logic simulation, fault simulation and test pattern generation are fundamental techniques in electronic design automation with applications, e. g., in validation, test and also product quality estimation.

Automatic test pattern generation (ATPG) algorithms for stuck-at faults either compute a pattern that detects a given fault or prove its untestability. They are typically based on structural methods such as the D-algorithm [1], PODEM [2] or FAN algorithm [3], or on Boolean satisfiability (SAT) reasoning [4–7].

However, depending on the circuit and test method, a very high fraction of signals may have X-values (see e. g., [8], [9]) that have to be taken into account during the test pattern generation process. Such X-sources include non-finalized parts in early design steps. But also during operation and test application, X-values may be caused by uncontrolled sequential elements, at clock domain crossings, or A/D boundaries. Additional X-sources are introduced by specific test methods such as faster-than-at-speed testing [10] or the consideration of complex fault models (e. g., open fault models [11]).

Different extensions of the basic two-valued Boolean circuit logics have been proposed to model signal states in the circuit in presence of X-values (e. g., [12–14]). However, all of them lead to pessimism of forward implication in test generation as reconverging X-values that depend on each other cannot be modeled accurately (cf. Section 3.1). To improve accuracy, restricted symbolic simulation [15] extends the number of symbols to distinguish *different* X-states and their inversion. This allows to reduce the pessimism [16], unless multiple X-states from different X-sources converge at a gate.

In general, the limited number of symbols does not allow to reflect all correlations between X-valued signals and at reconvergencies, where X-canceling may occur, the accurate output value cannot be computed any more. 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.

Therefore, design techniques that remove the impact of X-values on the circuit have been proposed and are specifically employed in the context of so-called build-in-self-test (BIST) schemes. X-canceling [17] or X-masking [18] allows to increase the number of detected faults at the cost of additional hardware structures. The overestimation of X-values in classical algorithms leads to unnecessary effort invested in such X-avoidance techniques.

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 [19–21]. The methods used rely on symbolic computation by Boolean satisfiability (SAT), quantified Boolean formula (QBF) reasoning, or binary decision diagrams (BDDs).

More accurate or even fully accurate fault simulation can be performed even for large circuits by a combination of heuristics and SAT reasoning and allows a significant increase of fault coverage [22–24]. In principle, both logic and fault simulation in presence of X-values are NP-complete problems. And also deterministic test pattern generation for stuck-at faults in presence of X-values is at least an NP-hard problem [24].

In contrast to propositional formulae used for SAT, quantified Boolean formulae [25], where variables are existentially or universally quantified, allow a succinct representation for all possible X-values. The recent advances in the performance of QBF solvers, for example conflict driven learning [26], resolution and expansion based algorithms [27], or preprocessing [28] enable exact reasoning about fault testability in presence of Xs even for larger circuits. Doing this, an efficient stuck-at fault test generation algorithm able to prove testability or untestability of faults in presence of X-values can be realized as outlined in greater detail in Section 3.

1.2 Unknown Values in Verification

Unknown values in circuit verification can occur, for instance, when a circuit is only partially available. Partially available means that for some of the circuit's components only their interface is known, i. e., the signals entering and leaving the components, but neither their internal structure nor the computed function. These missing parts are called *black boxes*. The actual values at their outputs are unknown. Verification has to take this into account.

There are different reasons for considering such partial (or incomplete) circuits: Errors in a circuit design should be detected as early as possible; the later errors are corrected the higher are the incurred costs. Therefore it is desirable to apply verification techniques already in an early stage of the design process when not all parts of a circuit have been implemented yet.

A further reason for considering incomplete circuits is that some modules like multipliers are notoriously hard to verify: If the property to be checked is expected to be independent of such a module, the module can be removed from the circuit, and instead it is checked whether the property under consideration holds for all possible replacements of the missing part. If this is the case, then the property also holds for the complete circuit. Otherwise either the remaining circuit is faulty or the removed module and the property interact in some unexpected way.

Considering incomplete circuits can also be beneficial for error diagnosis during debugging. Assume that an error is contained in one of the circuit's modules, but it is not known in which one. If, after removing one module, verification yields that there is an implementation of the removed part such that the considered property holds, then it is likely that the error is contained in the removed module.

If error diagnosis and error rectification are performed late in the design cycle when already a lot of efforts have been made to perform logic synthesis or even place & route steps for the complete design, then the question will be whether the design can be rectified by changing *locally* confined black boxes only, without introducing new connections to global signals leading to enormous costs for re-synthesis. A similar situation occurs in case of Engineering Change Order (ECO, small changes of specification late in the design cycle) where only locally confined parts (black boxes) should be replaced in order to satisfy the changed specification without sacrificing too much of the design efforts. In this case it is particularly important to preserve the interface of the black boxes.

The synthesis of digital controllers [29, 30] that ensure certain properties of the system at hand can also be considered as a black-box verification problem: The controller to be synthesized is the black box, and one asks whether there is an implementation such that the given property holds.

Depending on the application there are two different problem classes that are of interest: On the one hand, *realizability* asks whether there is an implementation of the black boxes such that the given property holds. On the other hand, *validity* asks whether the property holds for all possible implementations. Since validity and realizability are dual properties—a property φ is valid iff $\neg\varphi$ is not realizable—we concentrate in the following on realizability problems.

The problem whether an incomplete combinational circuit can be completed such that it becomes equivalent to a given specification (*partial equivalence checking*, PEC) was first considered in [31] where several *approximate* and *exact* methods to solve the PEC problem have been presented. If an approximate algorithm reports that there is no implementation for the black boxes such that the specification holds, the desired specification is indeed not realizable. However, if such an algorithm is not able to prove non-realizability, this can be due to the approximate nature of the method, and the desired functionality may nevertheless be *not* realizable. The algorithms in [31] are based on solving SAT or QBF formulations of PEC. The SAT formulations are efficient to solve, but also rather inaccurate due to a coarse approximation. Their accuracy is improved in several steps, leading to a QBF formulation that can solve PEC for a single black box exactly. In [31] additionally an exact characterization of realizability of PEC for multiple black boxes has been proposed (based on the decomposability of a certain Boolean relation). However, no feasible algorithmic method for solving the problem has been given.

Nevertheless, [31] was the first paper to consider an exact solution of the PEC problem taking into account that the *interfaces* of the black boxes in the incomplete circuit have to be preserved. Apart from the approach in [32, 33], the diagnosis and rectification problem respecting local interfaces has not been addressed in the literature so far. In [34, 35], e. g., rectifications are computed, but they are allowed to depend on arbitrary signals in the circuit. (Moreover, in contrast to [35], [34] uses a SAT formulation to compute rectifications for a given set of counterexamples only, without considering correctness for *all* possible inputs.) The approach of [32, 33] solves the PEC problem exactly, but it is restricted to problem instances of moderate sizes, since the black boxes are replaced by function tables using an exponential number of Boolean variables. A more efficient complete approach, based on solving dependency quantified Boolean formulas (DQBFs) was presented in [36].

We have extended the application of realizability checking to sequential circuits which are specified by a set of properties (safety properties or more general properties formulated in Computation Tree Logic (CTL [37]). Here the question is whether an incomplete sequential design may be extended by black box implementations such that a set of given properties is satisfied. Also the problem of deciding validity is considered. We developed various approaches for solving the realizability problem either in an approximate or an exact manner. In the following we discuss some representative approaches: In [38], we provided a series of approximate methods with different precision and costs for deciding the realizability of CTL properties using symbolic methods. The approximations are based on different methods to model the effect of the unknowns at the black box outputs to the overall circuit. Moreover, [38] presents an exact method for deciding realizability for incomplete circuits with several black boxes under the assumption that the black boxes may contain only a bounded amount of memory. This exact method is based on introducing an exponential number of new variables and is therefore only suitable for small problem instances. In [39] similar approximation methods are applied in the context of realizability checking of safety properties based on bounded model checking techniques (BMC—here a sequential circuit is "unrolled" for a number of time frames). This approximate approach leads to SAT or QBF problems. Here, the precision of modeling is not given by the user, but it is adapted automatically based on the difficulty of the problem. The approach is guided by proofs that non-realizability can not be shown using the weaker methods, independently from the number of BMC unrollings, i. e., independently from the length of a counterexample which does not depend on the implementation of the black boxes. [39] has been enhanced later on by [40] which provides proofs based on inductive arguments that non-realizability can not be shown even by our most exact QBF based methods (also independently from the number of BMC unrollings). The approach of [40] provides an exact decision procedure for realizability in the case that the design contains exactly one black box which is allowed to read all input signals (which means that it has "complete information").

In Section 4 we sketch some of the state-of-the-art techniques to solve the realizability problem of incomplete circuits.

1.3 Minimization/Maximization in Test and Verification

We finish this introductory remarks by mentioning an interesting application of unknowns to optimize the quality of patterns. More details can be found in the papers referenced.

Modeling of unknown values can be used to generalize results by forcing a target property to hold while, at the same time, requiring a maximal number of unknown values. Such a solution is helpful as only a minimal set of information needed to guarantee the property is computed and hence the solution is generalized.

A well-known instance of such an optimization problem in the test domain is the problem of finding a test pattern for a given fault requiring only a minimal set of inputs to be defined. In the verification domain, a likewise problem is finding a generalized trace that leads to an (unwanted) error state.

Both problems can be solved (optimally) using maximization techniques such as [41, 42]. They work on top of the encoding techniques presented in this book chapter by requiring a certain primary property (e. g., the detection of a fault) to hold, while at the same time maximizing secondary objectives such as the number of inputs set to X.

2 Basics

In this section, we provide an overview on the underlying formal methods considered in the chapter as well as the handling and encoding of Boolean circuits.

2.1 Boolean Satisfiability and Extensions

The following two subsections provide a brief overview on the satisfiability problem (SAT) and on quantified Boolean formula (QBF). The interested reader is referred to [25] for more details.

Deciding the satisfiability of a propositional Boolean formula (SAT) is an NP-complete problem [43]. 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.

Many SAT-related formalisms have been introduced in recent decades. A prominent extension to the Boolean satisfiability problem is the *Maximum Satisfiability* problem (MaxSAT), an optimization problem, which is used e.g. for the applications referenced in Subsection 1.3. Intuitively, in a MaxSAT problem we try to satisfy *as many clauses as possible* in φ. In this context the clauses are also called *soft clauses.* There are several natural extensions of MaxSAT like *Weighted MaxSAT* and *Partial MaxSAT*. In the former extension the clauses are labeled with non-negative weights and the goal is to maximize the sum of the weights of the satisfied clauses. In the latter extension there are additional so-called *hard clauses*, which *must* be satisfied, whereas the soft clauses are treated as in MaxSAT. Likewise SAT, one obtains a model which indicates the MaxSAT objective: the number of soft clauses (or the sum of the clause weights) which are satisfied simultaneously.

2.2 Quantified Boolean Formulas

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) $\psi = \mathcal{Q}_1 X_1 \mathcal{Q}_2 X_2 \dots \mathcal{Q}_n X_n \varphi$, with $\mathcal{Q}_i \in \{\exists, \forall\}$ 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 [44].

2.3 Dependency Quantified Boolean Formulas

Dependency quantified Boolean formulas (DQBF) are a generalization of QBF. In QBF, each existential variable depends on all universal variables on lower quantification levels. DQBF relaxes this restriction and allows existential variables to depend on arbitrary sets of universal variables.

This section mainly follows the descriptions in [36, 45].

Let φ be a Boolean formula over the Boolean variables $x_1, \ldots, x_n, y_1, \ldots, y_m$, and $D_1, \ldots, D_m \subseteq \{x_1, \ldots, x_n\}$ sets of Boolean variables. A *dependency-quantified Boolean formula (DQBF)* ψ has the form:

$$\psi := \forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D_1) \exists y_2(D_2) \ldots \exists y_m(D_m) : \varphi.$$

The sets D_i are called dependency sets of y_i and the formula φ is ψ's matrix.

We denote $V^\exists = \{y_1, \ldots, y_m\}$ as the set of existential variables and $V^\forall = \{x_1, \ldots, x_n\}$ the set of universal variables. If $y_i \in V_\psi^\exists$ is an existential variable with dependency set D_i, a *Skolem function* for y_i is a function $s_{y_i,D_i} : \mathcal{A}_{D_i} \to \{0,1\}$. In this case, $\varphi[s_{y_i,D_i}/y_i]$ denotes the expression resulting from φ by replacing each occurrence of y_i by a Boolean expression for the Skolem function s_{y_i,D_i}.

For a variable $x \in D_i$ we denote by $s_{y_i,D_i|x=0}$ the Skolem function $s_{y_i,D_i\setminus\{x\}} : \mathcal{A}_{D_i\setminus\{x\}} \to \{0,1\}$ which results from s_{y_i,D_i} by setting the variable x constantly to 0. Accordingly for $s_{y_i,D_i|x=1}$.

Let $\psi := \forall x_1 \forall x_2 \ldots \forall x_n \exists y_1(D_1) \exists y_2(D_2) \ \ldots \exists y_m(D_m) : \varphi$ be a DQBF. ψ is *satisfied* (written $\vDash \psi$) if and only if there are Skolem functions s_{y_i,D_i} for $i = 1, \ldots, m$ such that $\varphi[s_{y_i,D_i}/y_i \ \forall y_i \in V^\exists]$ is a tautology.

First solver implementations for DQBF are already available. We refer the reader to, e. g., [45, 46] for more information on solving DQBFs.

Every QBF can be understood as a DQBF: the QBF $\Psi := \forall X_1 \exists Y_1 \ldots \forall X_n \exists Y_n : \varphi$, where $X_i \subseteq \{x_1, \ldots, x_n\}$ and $Y_i \subseteq \{y_1, \ldots, y_n\}$ are disjoint sets of variables, is equivalent to the DQBF

$$\psi := \forall x_1 \ldots \forall x_n \exists y_1(D_{y_1}) \ldots \exists y_m(D_{y_m}) : \varphi$$

where $D_{y_j} = \bigcup_{\ell=1}^{k} X_\ell$ if Y_k is the unique set with $y_j \in Y_k$.

2.4 From Circuits to Formulas

By using a *Tseitin encoding* [47], a SAT instance (as CNF representation) Φ_C of a circuit C can be generated, whose size is linear in the circuit size. A Tseitin encoding of a circuit defines a Boolean variable for each line. These variables are used to represent the function of each gate based on its inputs using a two-valued logic (01-logic).

For instance, an AND gate with the inputs a and b and the output g is characterized by $g \leftrightarrow (a \wedge b)$. The corresponding encoding Φ_g for this gate g would be

$$\Phi_g := \big\{\{a, \neg g\}, \{b, \neg g\}, \{\neg a, \neg b, g\}\big\}.$$

In extension to the two-valued Tseitin encoding of a circuit, the three-valued $01X$-encoding based on [14] is often used to represent unknown (X) values.

The $01X$-logic consists of three values $\{0, 1, X\}$, which are encoded using two Boolean variables as follows: $0 = (1, 0)$, $1 = (0, 1)$, $X = (0, 0)$. The combination $(1, 1)$ is not allowed.

The $01X$-encoding for the same AND gate g, Φ_g, would be

$$\Phi_g := \big\{\{\neg a_1, g_1\}, \{\neg b_1, g_1\}, \{a_1, b_1, \neg g_1\}, \{a_2, \neg g_2\}, \{b_2, \neg g_2\}, \{\neg a_2, \neg b_2, g_2\}\big\}.$$

In comparison to a standard Tseitin encoding, the support for the X-symbol leads to larger SAT instances and hence usually harder instances but at the same time allows reasoning about unknown values. A drawback of this formulation is its pessimism that may incorrectly predict unspecified values on path reconvergencies.

3 Unknown Values in Circuit Test

As already mentioned 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*. Of course, depending on the circuit structure these unknown values may imply further unknown values within the circuit, and an efficient and effective determination of signals with unknown values turns out to be of major interest with respect to test algorithms.

Logic and fault simulation as well as solving the ATPG problem are essential techniques in electronic design automation. The accuracy and therefore effectiveness of standard algorithms is compromised by unknown or X-values. As demonstrated in the following, using standard three-valued 01X logic this results in a pessimistic overestimation of X-valued signals in the circuit and a pessimistic underestimation of fault coverage.

3.1 Standard X-Logic Simulation

Standard X-logic simulation algorithms are based on n-valued logic systems with a limited number of symbols to denote the signal states in the simulation (i. e., three-valued 01X logic). Not all X-states, and the correlations between them, are represented accurately. The result may either underestimate the number of X-values as in the case of logic simulation using Verilog models [48], or pessimistically overestimate their number.

Example 1. Fig. 1 shows a circuit with three gates and three inputs. The simulation result of pattern $(a, b, c) = (1, \mathrm{X}, 1)$ with a standard 3-valued logic simulator is annotated to the circuit lines. The signals d, e, and f are evaluated to the unknown value X by the simulator. However, exhaustive simulations assuming $b = 0$ and $b = 1$ would show that the output f has the logic value 1 in both cases as the signals d and e always have opposite logical values. Hence, three-valued simulation overestimates the number of signals with an unknown value.

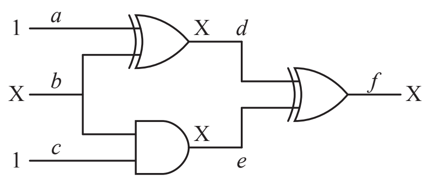


Fig. 1. Pessimistic simulation result with a 3-valued logic simulator.

For fault simulation like the parallel pattern single fault (PPSFP) or the concurrent algorithm [49], [50], [51], [52] this leads to the fact that they either pessimistically underestimate the number of detected faults or vice versa, the number of potentially detected faults is overestimated and count a fraction of potentially detected faults as detected [53]. Both inaccuracies impact product quality and may increase test overhead and cost.

3.2 Accurate Logic Simulation

Removing the pessimism of classical n-valued fault simulation requires to solve the problem of distinguishing reconverging X-values (as present in Fig. 1) that depend on each other.

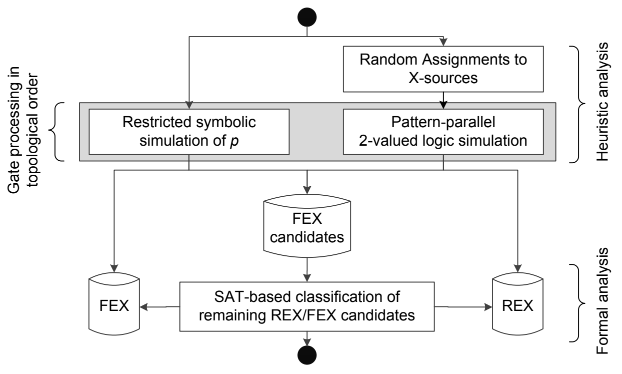

Random Assignments to X-sources
Gate processing in topological order
Restricted symbolic simulation of p
Pattern-parallel 2-valued logic simulation
Heuristic analysis
FEX candidates
FEX
SAT-based classification of remaining REX/FEX candidates
REX
Formal analysis

Fig. 2. Exact fault free simulation for a pattern p [24].

In [24] an accurate logic simulation algorithm based on solving a sequence of SAT-instances is proposed.

It consists of two consecutive steps as depicted in Fig. 2. In the first *heuristic analysis* step a restricted symbolic simulator and a 2-valued logic simulator are used as heuristics to classify a high number of REXs (Real X), FEXs (False X) and FEX candidates at low computational cost. In the second *formal analysis* step, the set of FEX candidates is formally analyzed. For the formal proof whether a FEX candidate is a REX or not, the state-of-the-art incremental SAT solver antom [54] is utilized. The details of the steps are described in the folowing.

Heuristic Analysis In the heuristic analysis the pattern p is simulated using restricted symbolic simulation (RSS, [15]) and 2-valued pattern-parallel simulation of randomized assignments to the X-sources to classify as many signals as REX, FEX and FEX candidates as possible. The gates of the circuit are processed in topological order and for each gate, RSS and 2-valued simulation are performed. The identified FEX candidates are later classified using SAT reasoning.

In RSS, for each X-value at the X-sources a unique symbol X_i is introduced in addition to the two symbols for logic-0 and logic-1. Hence, X-values from different X-sources are distinguishable. Furthermore, each X-symbol can be negated. This allows the correct evaluation of simple local reconvergences of X-valued signals and increases accuracy compared to 3-valued simulators. For the example in Fig. 1, RSS correctly computes the output value at f as logic-1, since the symbol X_b introduced at X-source b is correctly tracked at d as $\neg\mathrm{X}_b$ and at signal e as X_b. Hence, the reconvergence is exactly evaluated to logic-1. Thus, RSS identifies a subset of $\mathrm{FEX}^G(p)$. In the proposed algorithm, the resulting value of RSS of signal s and pattern p is stored in $v^G(p, s)$.

A subset of $\mathrm{REX}^G(p)$ is efficiently found by a 2-valued pattern-parallel logic simulation. 64 random patterns are generated by assigning randomized values to the X-sources. The signal values are computed in one simulation run. One 64-bit integer $v = [v^0, \dots, v^{63}]$ is used to represent the values of each signal. For input i, v_i is derived from the simulated pattern p and set to $v_i = [0, \dots, 0]$ or $v_i = [1, \dots, 1]$ if i is logic-0 or logic-1, respectively. At X-source q, a randomized 64-bit integer is generated and assigned to $v_q = [v_q^0, \dots, v_q^{63}], v_q^k \in \{0, 1\}, 0 \leq k \leq 63$. v_q is used for the evaluation of the direct fanout of q.

After finishing both simulations, each signal is classified as logic-0, logic-1 or REX, FEX or FEX candidate. If RSS derived a logic value, the signal does not need to be considered in the subsequent steps. If an unknown value is calculated for s, the values of $v_s = [v_s^0, \dots, v_s^{63}]$ of the pattern-parallel simulation are taken into account. If at least one pair of values $v_s^k, v_s^l (0 \leq k, l \leq 63)$ has complementary values, the signal s belongs to $\mathrm{REX}^G(p)$. If all v_s^k bit are equal, s is marked as FEX candidate. The classification of these signals is done with an incremental SAT-solver as explained in the next section.

Classification of Remaining FEX Candidates The FEX candidates are exactly classified by use of an incremental SAT solver. Input to the SAT solver is a Boolean formula in conjunctive normal form (CNF) which maps the classification of a signal to a Boolean satisfiability problem.

For each FEX candidate s it is already known that all 64 random assignments to the X-sources force s to value $v_s^k (0 \leq k \leq 63)$ of either logic-0 or logic-1. Signal s is a FEX, if and only if it can be proven that s cannot have the complementary value $\neg v_s^k$ for any assignment to the X-sources. Thus, the Boolean formula is constructed such that it is satisfiable, if and only if s can be driven to

$\neg v_s^k$. If the formula is satisfiable, s depends on the X-sources and is classified as REX. Otherwise s is independent of the X-sources and classified as FEX.

The FEX candidates are evaluated starting from the X-sources in topological order. To increase efficiency, the SAT instance is extended incrementally for each FEX candidate exploiting the result from the simulation step as well as learnt knowledge from analysis of previous FEX candidates.

To check whether s can be driven to $\neg v_s^k$, the characteristic equations of the gates in the adjustment cone, resp. transitive fanin, of s are translated into CNF and added to the SAT instance using the Tseitin transformation (c.f. 2.4. The size of the resulting SAT instance is reduced by only considering the gates which have been classified as REX or FEX candidate for pattern p.

This SAT instance is extended by a temporary unit clause with only one literal (called assumption) for FEX candidate s which constrains the value of s in the search process of the SAT solver. If the value of s in the pattern parallel simulation was $v_s = [0, \ldots, 0]$, the assumption $\{s\}$ is added to constrain the SAT search to assignments to the X-sources which imply s to logic-1. If the instance is satisfiable, s belongs to the set REX. Otherwise s is a FEX with value logic-0 and $v^G(p, s)$ is updated. In the latter case, the unit clause $\{\neg s\}$ is added permanently to the SAT instance to reduce runtime for subsequent calculations of the SAT solver. Correspondingly, if the value of s in the pattern parallel simulation was $v_s = [1, \ldots, 1]$, the assumption $\{\neg s\}$ is added.

For the classification of the next FEX candidate s' in topological order, the CNF instance is extended incrementally to include the adjustment cone of s', i. e.,, only the clauses for gates which are not yet Tseitin transformed are added.

During exact simulation, the algorithm maintains a lookup table derived from the result of the RSS step. The table contains the information if a symbol for an X-state assigned to signals during RSS is a logic-0, a logic-1 or a REX. Before analyzing a FEX candidate s using the SAT technique, a fast lookup is performed to check whether the corresponding symbol X_s has already been computed. If the classification for X_s is already known, s is set to the corresponding state. Otherwise, s is classified as described above. This effectively restricts the use of the SAT solver to signals at which REX values converge.

3.3 Accurate Fault Simulation

We distinguish definite detection (DD) and potential detection (PD) of a fault. A fault f is definitely detected (DD) if an observable output o exists where the fault effect is visible 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 a pattern p in the fault free and faulty case in presence of unknown values.

The definite detection of a stuck-at-ϕ fault f ($\phi \in \{0, 1\}$) at line l under a pattern p is given as

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

where O is the set of output signals of the circuit. If f is not definitely detected, f is potentially detected (PD) if the fault is activated and an observable output o

exists where the fault effect can be deterministically measured for at least one logic value assignment to the X-sources:

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

Note that 3-valued fault simulation underapproximates the number of definitely detected faults since three-valued simulation overestimates the number of signals with X-values. Consequently, the number of potentially detected faults provides an overapproximation.

The exact simulation classifies a set of target faults as definitely detected (DD), potentially detected (PD) or undetected for a test set in presence of unknowns. An overview of the fault simulation of a pattern p is given in Fig. 3. 3-valued fault simulation is used to mark as many target faults as possible as DD. For the remaining faults, an exact analysis is conducted.

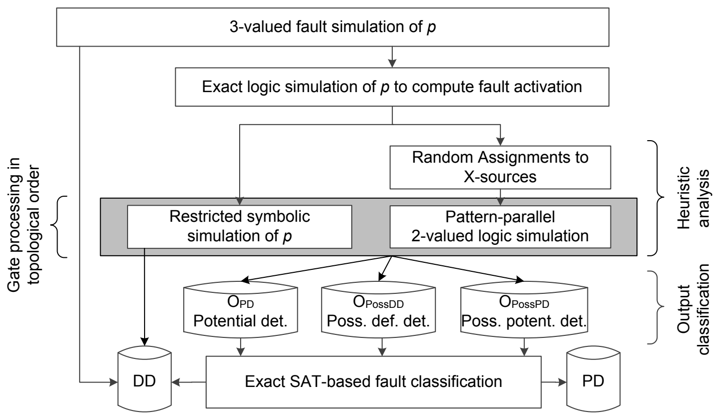

3-valued fault simulation of p
Exact logic simulation of p to compute fault activation
Random Assignments to X-sources
Gate processing in topological order
Restricted symbolic simulation of p
Pattern-parallel 2-valued logic simulation
Heuristic analysis
O_PD Potential det.
O_PossDD Poss. def. det.
O_PossPD Poss. potent. det.
Output classification
DD
Exact SAT-based fault classification
PD

Fig. 3. Exact fault simulation for a pattern p and classification as definitely detected (DD) or potentially detected (PD) [24].

The exact analysis starts with the exact logic simulation of the fault free circuit for pattern p to compute the set of activated faults. These faults are then analyzed serially. For the fault simulation of an activated fault f, f is injected into the circuit model. The algorithm then proceeds in two phases similar to the fault free approach: A heuristic simulation and an exact calculation step. During the simulation step the behavior of the faulty circuit is simulated in event-driven manner by RSS and 2-valued pattern-parallel logic simulation which evaluates random assignments to the X-sources. If the results of the simulations allow the fault classification as DD or undetected, a further analysis is not required. Otherwise, the SAT solver is invoked for analysis of the outputs of the faulty circuit. Internal signals in the faulty circuit do not need to be considered since the values at observable outputs are sufficient to reason about fault detection.

3.4 Accurate Test Pattern Generation (X-ATPG)

The ATPG framework from [55] is able to prove the testability of stuck-at faults in presence of X-values. Fig. 4 shows the complete flow which combines accurate fault simulation (c.f. Section 3.3), incremental SAT-based test generation with a classical three-valued encoding and accurate QBF-based reasoning to efficiently analyze the faults.

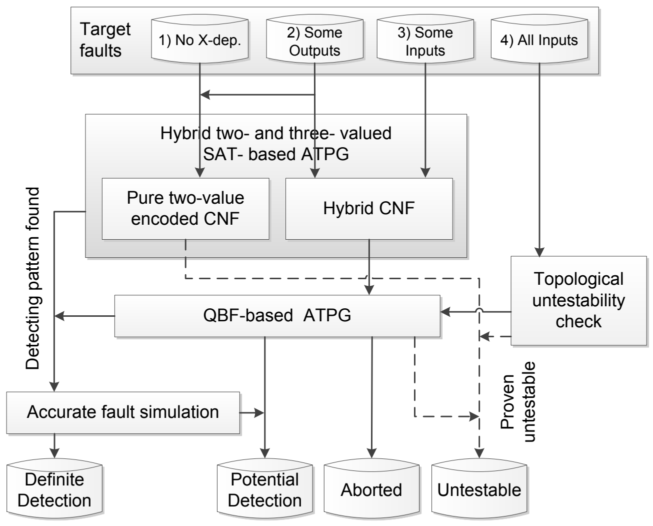


Fig. 4. Overview of the ATPG flow.

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. Fig. 4):

1. No structural dependence on the X-sources: 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 3.2). 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. 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 a topological untestability check 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 3.3).

QBF-based Detection of Stuck-at Faults The construction of the QBF is split into the generation of the matrix and the quantification of the variables.

Construction of the Matrix The matrix of the QBF in CNF is constructed similar to a classical two-valued SAT-based ATPG instance. 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. 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 logically 1. This leads to the following propositional formula in CNF:

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

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 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. 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), the solution space is constrained 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 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 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 the 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. In these cases, 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:

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

With the existential quantification of the variables $v_i^0, v_i^1 \in V$ on level 0 we obtain the following QBF:

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

This QBF is satisfiable if and only if a fault is testable according to the definite detection condition of Section 3.3. If the formula is not satisfiable, it is proven that no test pattern exists for definite detection.

Taken together, we depicted a complete ATPG flow 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.

4 Unknown Values in Verification

In the following we turn to the formal verification digital circuits in the presence of unkowns. We first define partial circuits and validity and realizability of a property regarding a partial circuit. Then we present approaches how partial circuits can be analyzed.

4.1 Incomplete Circuits

An *incomplete (or partial) circuit* is a combinational or sequential circuit containing so-called *black boxes* (BBs). A black box is a module of a circuit whose interface is known but not its internal structure. A partial sequential circuit is sketched in Fig. 5. The circuit contains m black boxes $\mathrm{BB}_1, \ldots, \mathrm{BB}_m$, shown as black rectangles. Their input signals are denoted by $\boldsymbol{I}_1, \ldots, \boldsymbol{I}_m$ and their output signals by $\boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_m$. The primary inputs of the circuit are $\boldsymbol{x} = (x_0, \ldots, x_n)$, the current state is given by the signals $\boldsymbol{s} = (s_0, \ldots, s_r)$. The input cones of the black boxes compute the functions $\boldsymbol{I}_i = \boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{i-1})$. We thereby assume that there are no cyclic dependencies between the black boxes and that they are topologically ordered, i. e., BB_i only depends on the values computed by $\mathrm{BB}_1, \ldots, \mathrm{BB}_{i-1}$. To simplify notation, we assume w. l. o. g. that no black box output is directly connected to a black box input, i. e., $\boldsymbol{Z}_i$ is disjoint from $\boldsymbol{I}_j$ for all i, j. If this is not the case, we insert a buffer between the corresponding black boxes, which does not modify the functionality of the circuit. Finally the output $\boldsymbol{y}$ and the next state $\boldsymbol{s}'$ of the circuit are given by the Boolean functions $(\boldsymbol{y}, \boldsymbol{s}') = \boldsymbol{R}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_m)$.

We assume that the contents of the black boxes are combinational circuits. If we allow the black boxes to contain an arbitrary amount of memory, the interesting decision problems (see below) become undecidable [56]. The case of black boxes with a bounded amount of memory can be reduced to the case

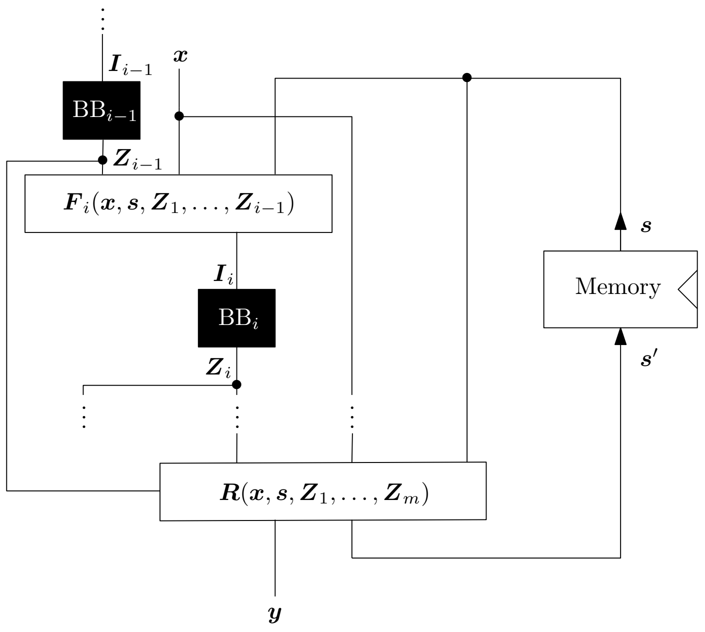

$\boldsymbol{I}_{i-1}$
$\boldsymbol{x}$
BB$_{i-1}$
$\boldsymbol{Z}_{i-1}$
$\boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{i-1})$
$\boldsymbol{s}$
$\boldsymbol{I}_i$
Memory
BB$_i$
$\boldsymbol{s}'$
$\boldsymbol{Z}_i$
$\boldsymbol{R}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_m)$
$\boldsymbol{y}$

Fig. 5. Notations for an incomplete sequential circuit

of combinational black boxes by adding the memory of the black boxes to the surrounding circuit such that these memory cells are read and written only by the corresponding black box.

For a given property φ two questions regarding a partial circuit are of interest: On the one hand, *realizability* asks whether there is an implementation of the black boxes such that the complete circuit satisfies φ. On the other hand, *validity* asks whether φ is satisfied for all possible implementations. Since validity of φ is given iff $\neg\varphi$ is not realizable, we restrict ourselves in the following to realizability problems.

For partial combinational circuits we assume that the property φ is given as a circuit. The resulting realizability problem is known as the *partial equivalence checking problem* (PEC) [31]. We combine the property circuit and the partial circuit into a single miter circuit: corresponding inputs are connected, corresponding output are combined via an XOR gate; in case of several outputs, the outputs of the XOR gates are combined via an OR gate. The resulting circuit has the property that its single output is 1 for an assignment of the primary inputs and an implementation of the black boxes iff the specification φ and the implementation compute the same output values. Realizability means then: Are there implementations of the black boxes such that the output of the miter circuit is constantly 1?

For partial sequential circuits we consider invariant properties: Given a Boolean formula $\text{inv}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{y})$, which describes the states of the circuit that satisfy the invariant, are there implementations of the black boxes such that $\text{inv}(\boldsymbol{x}, \boldsymbol{s}, \boldsymbol{y})$ is satisfied in each step of the circuit? For more general classes of properties like arbitrary CTL properties, we refer the reader to [57, 38, 58].

4.2 Incomplete Combinational Circuits

We will first show how the PEC problem can be solved for combinational circuits. We extend the methods known from the previous sections, starting with SAT-based symbolic 01X simulation, which were already used in the previous sections. Finally, we extend these methods to DQBF-based formulations which constitute a complete decision method for PEC.

SAT-based Approximations SAT-based methods use symbolic $\{0, 1, X\}$ simulation: The outputs of the black boxes carry unknown values and are therefore assigned the value X, while the primary inputs are forced to be either 0 or 1. Realizability is refuted if an input pattern can be found which leads to value 0 at the primary output of the miter circuit. Realizability is proven if all input patterns lead to output value 1.[1] However, a third case is possible, namely that the unknown value X propagates to the primary output for some input patterns. In this case, no statement can be made regarding realizability.

To decide whether there exists an input pattern that refutes realizability, a SAT-formulation can be used. To encode the three-valued logic we use the encoding introduced in Section 2.4 and force the output y of the miter circuit to be zero by adding appropriate unit clauses. The result is a Boolean formula in CNF whose satisfiability proves that the design is not realizable.

While this method is efficient in practice, it has the drawback that it constitutes a rather coarse approximation: If the unknown value X propagates to the output, no statement about the realizability can be made.

QBF-based Approximations The quality of the approximation can be improved by universal quantification over the possible input values: For all possible values at the inputs, there have to be values of the black box outputs such that the desired property is satisfied (i. e., the output of the miter circuit is 1). This yields QBF formulations for deciding PEC. We first show how to derive the matrix of the QBF formula and then define an appropriate quantifier prefix.

In contrast to circuit test applications where unknown values typically appear at the primary or secondary inputs of the circuit, we have to take into account here that black boxes are not necessarily directly connected to the primary inputs, but to internal signals. In this case not all possible combinations of values may arrive at the inputs of the black boxes. Since we use universal quantification for the black box inputs we have to ensure that the matrix of our formula is satisfied if the value of the black box inputs $\boldsymbol{I}_i$ deviates from the values obtained as a

[1] Note that in this case also validity holds.

function $\boldsymbol{F}_i(\boldsymbol{x}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{i-1})$. This leads to the following formula:

$$\varphi := \big(\boldsymbol{I}_1 \not\equiv \boldsymbol{F}_1(\boldsymbol{x})\big) \vee \cdots \vee \big(\boldsymbol{I}_m \not\equiv \boldsymbol{F}_m(\boldsymbol{x}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_{m-1})\big) \vee R(\boldsymbol{x}, \boldsymbol{Z}_1, \ldots, \boldsymbol{Z}_m).$$

By applying Tseitin transformation [47], which introduces auxiliary variables $\boldsymbol{H} = (h_1, \ldots, h_p)$ for the internal signals of the circuit, one can obtain a CNF φ' that is satisfiability equivalent to φ and whose size is linear in the size of φ. The variables in $\boldsymbol{H}$ are existentially quantified in the quantifier prefix.

As we will see later when we consider complete decision procedures for PEC, QBF is—like the SAT-based method described above—only an approximation in case that the design contains more than one black box. However, we can give both under- and over-approximations: If an over-approximating QBF is unsatisfied, we can conclude the unrealizability of the PEC. If an under-approximating QBF is satisfied, this implies the realizability of the PEC. The other outcomes do not allow a statement regarding the realizability of the PEC. Over- and under-approximations only differ in their quantifier prefix.

For a QBF prefix $\mathcal{Q}_1 V_1\, \mathcal{Q}_2 V_2\, \ldots\, \mathcal{Q}_k V_k$ with variables $V = V_1 \cup \cdots \cup V_k$ and quantifiers $\mathcal{Q}_i \in \{\exists, \forall\}$ such that $Q_i \neq Q_{i+1}$ for all $i = 1, \ldots, k-1$, we say that a variable $u \in V$ is in the *scope* of variable $w \in V$ if V_i, V_j are the unique sets with $u \in V_i$, $w \in V_j$ and $j < i$ holds. We write $w \prec u$ if u is in the scope of w. We extend this to vectors $\boldsymbol{U}, \boldsymbol{W}$ of variables such that $\boldsymbol{U} \prec \boldsymbol{W}$ iff $u \prec w$ for all $u \in \boldsymbol{U}$ and $w \in \boldsymbol{W}$.

For an over-approximating quantifier prefix we have to take care that each black box output is (at least) in the scope of the (primary and black box) inputs which are directly or indirectly read by the black box. For an under-approximating prefix, each black box output is allowed to be at most in the scope of these variables.

Formally spoken, the requirement on the quantifier order can be translated as follows: Each over-approximating QBF prefix has to satisfy the constraint

$$\forall i = 1, \ldots, m : \boldsymbol{I}_i \subseteq \{v \in V \mid v \prec \boldsymbol{Z}_i\}, \tag{3}$$

while for each under-approximation we have

$$\forall i = 1, \ldots, m : \boldsymbol{I}_i \supseteq \{v \in V \mid v \prec \boldsymbol{Z}_i\}. \tag{4}$$

Since the Tseitin variables $\boldsymbol{H}$ are implied by the gate's inputs, they are added as the right-most existential quantifier block.

Example 2. An over-approximation is given by $\forall \boldsymbol{I}_1 \ldots \forall \boldsymbol{I}_m \exists \boldsymbol{Z}_1 \ldots \exists \boldsymbol{Z}_m \forall \boldsymbol{x} \exists \boldsymbol{H} : \varphi'$, an under-approximation by $\exists \boldsymbol{Z}_1 \ldots \exists \boldsymbol{Z}_m \forall \boldsymbol{x} \forall \boldsymbol{I}_1 \ldots \forall \boldsymbol{I}_m \exists \boldsymbol{H} : \varphi'$.

Over- and under-approximating prefixes are not unique, and the choice of the prefix can influence the truth value of the formula. It is therefore desirable to make the approximation as strong as possible, i. e., having the black box outputs in the scope of as few (many) universal variables as possible. An approximation is exact if it is both an over- and an under-approximation, or equivalently, if (3) and (4) are satisfied with equality. In general, such an exact QBF formulation does not need to exist if the circuit contains more than one black box.

Example 3. Consider an incomplete circuit with a single black box, i. e., $m = 1$. Then

$$\forall \boldsymbol{I}_1 \exists \boldsymbol{Z}_1 \forall \boldsymbol{x} \exists \boldsymbol{H} : \varphi'$$

is an exact QBF formulation, i. e., it is satisfiable if and only if the PEC is realizable.

Complete Methods The QBF method provides more accuracy than the 01X-approximation. However, in case the design contains more than one black box, QBF is still not powerful enough to express realizability exactly, because there is typically no QBF prefix that expresses the dependencies of the black box outputs from the corresponding inputs exactly.

Example 4. Consider a design with two black boxes BB_1 and BB_2. The input of BB_1 is x_1, its output y_1, the input of BB_2 is x_2 and its output y_2. There are three admissible QBF prefixes: $\forall x_1 \forall x_2 \exists y_1 \exists y_1$, $\forall x_1 \exists y_1 \forall x_2 \exists y_2$, and $\forall x_2 \exists y_2 \forall x_1 \exists y_1$. None of these is exact: in the first, both black box outputs depend on both inputs, in the second prefix, BB_2 depends on both inputs, and in the third this holds for BB_1.

In order to be able to express the dependencies correctly, one has to resort to DQBF ([36], see also Section 2.3). It is able to express arbitrary dependencies and therefore constitutes a complete decision method for arbitrary incomplete combinational circuits.

We specify the quantifier prefix of the DQBF; the matrix is the same as for the QBF approximations. The primary inputs $\boldsymbol{x}$ and the black box inputs $\boldsymbol{I}_1, \ldots, \boldsymbol{I}_m$ are again universally quantified, all other variables are existentially quantified. The dependency set of black box output $z_{i,j}$ contains exactly the inputs $\boldsymbol{I}_i$ of BB_i. Hence, the resulting DQBF is:

$$\psi := \forall \boldsymbol{x} \forall \boldsymbol{I}_1 \ldots \forall \boldsymbol{I}_m \exists \boldsymbol{Z}_1(\boldsymbol{I}_1) \ldots \exists \boldsymbol{Z}_m(\boldsymbol{I}_m) \exists \boldsymbol{H}(\boldsymbol{x}, \boldsymbol{I}_1, \ldots, \boldsymbol{I}_m) : \varphi'.$$

The formula ψ is satisfied if and only if we can replace all $\boldsymbol{Z}_i$ with Skolem functions $\boldsymbol{f}_{\boldsymbol{Z}_i, \boldsymbol{I}_i}$ such that φ' becomes a tautology. The Skolem functions $\boldsymbol{f}_{\boldsymbol{Z}_i, \boldsymbol{I}_i}$ exist if and only if there are implementations for the black boxes BB_i of the PEC, such that the specification is realized. Therefore the Skolem functions constitute implementations of the black boxes that satisfy the specification.

4.3 Incomplete Sequential Circuits

We have extended the application of realizability checking to sequential circuits which are specified by a set of properties (safety properties or more general properties formulated in Computation Tree Logic (CTL [37]). In [38], e. g., we provided a series of approximate methods with different precision and cost for deciding the realizability of CTL properties using symbolic methods. The approximations were based on different methods to model the effect of the unknowns at the black box outputs to the overall circuit and also an exact

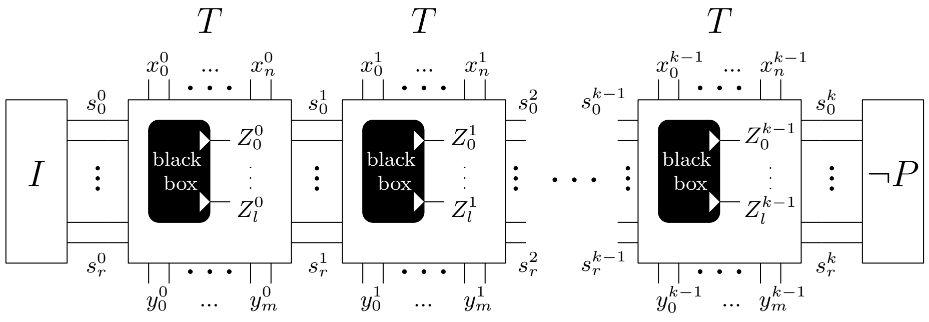


Fig. 6. Encoding of the BMC Problem for Incomplete Designs [40].

method was presented for deciding realizability for incomplete circuits with several black boxes under the assumption that the black boxes may contain only a bounded amount of memory. In this overview however, we restrict our attention to realizability checking of safety properties based on bounded model checking techniques (BMC) [39, 40].

BMC for Incomplete Designs BMC for incomplete designs aims to refute the realizability of a property, that is, it tells the designer, no matter how the unknown parts of the system will be implemented, the property will always fail. To put it in other words, the error is already in the implemented system. If this is the case, then we call the property P *unrealizable*. Here we restrict the properties to *invariants*. In a first formulation we make use of QBF modeling where the variables representing the black box outputs are universally quantified. We allow black box replacements to have arbitrary sequential behavior, that is, the black box can produce different output values for the same input values at different time steps. To encode the BMC problem of incomplete designs we are naming the variables as shown in Fig. 6. We use an upper index to specify the time instance of a variable. s_i^j denotes the i-th state bit in the j-th unfolding (let $\boldsymbol{s}^j = s_0^j, \ldots, s_r^j$). The same holds for the primary inputs $\boldsymbol{x}^j = x_0^j, \ldots, x_n^j$, the primary outputs $\boldsymbol{y}^j = y_0^j, \ldots, y_m^j$, and the black box outputs $\boldsymbol{Z}^j = Z_0^j, \ldots, Z_l^j$. The next state variables $\boldsymbol{s}^{j+1}$ depend on the current state, the primary inputs and the black box outputs. The whole circuit is transformed according to [47] using additional auxiliary variables $\boldsymbol{H}^j$ for each unfolding depth j. The predicate describing the initial states is given by $I(\boldsymbol{s}^0)$. Since we assume a single initial state in this paper, the initial state $I(\boldsymbol{s}^0)$ is encoded by unit clauses, setting the respective state bits to their initial value. The transition relation of time frame i is given by $T(\boldsymbol{s}^{i-1}, \boldsymbol{x}^{i-1}, \boldsymbol{Z}^{i-1}, \boldsymbol{s}^i)$. The invariant $P(\boldsymbol{s}^k)$ is a Boolean expression over the state variables[2] of the k-th unfolding. Using this information, the quantifier prefix (and the matrix) for the unrealizability problem results in the

[2] In general the property can also check the primary outputs, but for sake of convenience, we omit details here.

QBF formula (5). For the sake of simplicity we include the variables representing the primary outputs of unfolding depth j into $\boldsymbol{H}^j$.

$$\begin{array}{rcllll} BMC(k) & := & \exists\ \boldsymbol{s}^0\ \boldsymbol{x}^0 & \forall\ \boldsymbol{Z}^0 & \exists\ \boldsymbol{H}^0 \\ & & \exists\ \boldsymbol{s}^1\ \boldsymbol{x}^1 & \forall\ \boldsymbol{Z}^1 & \exists\ \boldsymbol{H}^1 \\ & & & \vdots & \\ & & \exists\ \boldsymbol{s}^{k-1}\ \boldsymbol{x}^{k-1} & \forall\ \boldsymbol{Z}^{k-1} & \exists\ \boldsymbol{H}^{k-1} \\ & & \exists\ \boldsymbol{s}^k & & \end{array}$$

$$I(\boldsymbol{s}^0) \wedge \bigwedge_{i=1}^{k} T(\boldsymbol{s}^{i-1}, \boldsymbol{x}^{i-1}, \boldsymbol{Z}^{i-1}, \boldsymbol{s}^i) \wedge \neg P(\boldsymbol{s}^k) \qquad (5)$$

The semantics following from the prefix corresponds to the following question:

> Does there exist a state $\boldsymbol{s}^0 = s_0^0, \ldots, s_r^0$ and an input vector $\boldsymbol{x}^0 = x_0^0, \ldots, x_n^0$ at depth 0 such that for all possible values of the black box outputs $\boldsymbol{Z}^0 = Z_0^0, \ldots, Z_m^0$ there exists an assignment to all auxiliary variables $\boldsymbol{H}^0$ (resulting in a next state $\boldsymbol{s}^1 = s_0^1, \ldots, s_r^1$) and an input vector $\boldsymbol{x}^1 = x_0^1, \ldots, x_n^1$ at depth 1, etc. such that the property is violated?

The BMC procedure iteratively unfolds the incomplete circuit for $k = 0, \ldots, K$ until a predefined maximal unfolding depth K is reached. If a QBF solver finds $BMC(k)$ satisfiable, the unrealizability of the property P has been proven. In that case the resulting system can reach a “bad state” after k steps, no matter how the black box is implemented.

We can prove that, whenever $BMC(k)$ is *unsatisfiable*, there is an implementation of the black box which is able to avoid error paths of length k *as long as the black box is allowed to read all primary inputs*. However, if the black box in the design at hand is not directly connected to all primary inputs, (i. e., if the black box does not have “complete information”), such an implementation does not need to exist. Thus, for black boxes having “incomplete information” the property may be unrealizable although BMC with QBF modeling is not able to prove this. In this case, DQBF is necessary to express the actual dependencies of the black boxes on their inputs.

In the following we give an example illustrating the approach:

Example 5. Consider the incomplete circuit shown in Fig. 7. The state bits s_0 and s_1 depend on the current state, the primary input x, and the black box outputs Z_0 and Z_1, respectively, and are computed by the transition functions $s_0' = x + Z_0$ and $s_1' = x \cdot Z_1 + s_0 \cdot \neg Z_1$. Let the invariant property $P = \neg(s_0 \wedge s_1)$ state that s_0 and s_1 must never be 1 at the same time. Let the initial state of the system be defined as $s_0^0 = s_1^0 = 0$. After checking for an initial violation of the property, the BMC procedure unfolds the system once, and tries to find an assignment to x^0 such that for all possible assignments to Z_0^0 and Z_1^0 the state $(1, 1)$ can be reached. Indeed, $x^0 = 1$ implies $s_0 = 1$ for all assignments to the black box outputs, however, for $Z_1^0 = 0$ $s_1 = 1$ can not be obtained (neither by

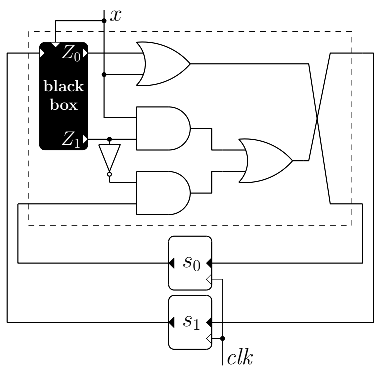


Fig. 7. Example Incomplete Design [40].

setting $x^0 = 0$ nor $x^0 = 1$). Thus, $BMC(1)$ is unsatisfiable and BMC continues by adding a second copy of the transition relation to the problem. If $x^0 = 1$, the current state bit s_0^1 at the second unfolding evaluates to 1 as well. Furthermore, if x^1 is set to 1, the next state bits s_0^2 and s_1^2 evaluate to 1 for all values of Z_0^1 and Z_1^1. Hence, when applying the input pattern $x^0 = x^1 = 1$, a state violating P can be reached after two steps for all actions of the black box and thus, $BMC(2)$ is satisfiable and P is unrealizable.

SAT-based Approximations and 01X-Hardness As already shown in previous sections, QBF can be approximated using 01X-logic. In the previous example, e. g., it is not necessary to use QBF encoding for Z_0. When applying $Z_0 = X$, unrealizability still can be proven by applying $x^0 = x^1 = 1$.

Since the problem instances using 01X-modeling are typically easier to solve, we are using the following verification flow:

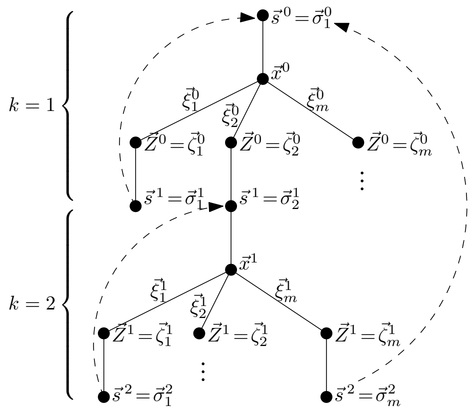
Given an incomplete design and an invariant, we start the BMC process with a pure 01X-modeling, that is we extend Boolean logic by a third value 'X' which then is applied to all black box outputs. Using the two-valued encoding proposed by Jain (cf. Section 2.4), the BMC unfoldings still yield SAT problems which can be solved by a state-of-the-art SAT solver. However, 01X-modeling may be too coarse to prove unrealizability leading to unsatisfiable BMC instances for every unfolding depth (we call such verification problems *01X-hard*). In [39] we presented a method based on Craig interpolation to classify 01X-hard problems on-the-fly along the BMC process, thus preventing the solver running into unsatisfiable instances forever. Additionally, the computed Craig interpolants provide information about the origin of the 01X-hardness and a subset of the black box outputs which have to be modeled more precisely using QBF is heuristically determined. Now a QBF-based BMC tool processes the information gathered from the Craig interpolants and uses one universally quantified variable for each black box output which needs a more precise modeling. Using a combined 01X/QBF-modeling (or a pure QBF-modeling) the BMC unfoldings yield QBF formulas. In that way, the precision of modeling is not given by the user, but it is adapted automatically based on the difficulty of the problem.

QBF-based Approximations and QBF-hardness However, even when using the more precise QBF modeling technique to model the unknown behavior of the black box, no result is guaranteed. At this point an extension given in [40] is introduced into the workflow. Similar to 01X-hardness for 01X-modeled incomplete designs, a QBF modeled BMC problem can now be classified as *QBF-hard*, if BMC would continuously run into unsatisfiable unfoldings.

As already discussed above, under certain conditions (black boxes having "incomplete information") the BMC procedure using a QBF formulation is not able to prove unrealizability even if the property is indeed unrealizable. In this sense the QBF formulation is a sound but incomplete approximation (just as 01X-modeling which is also an approximation, but is strictly coarser). If unrealizability can not be proven due to the approximative nature of the method or if the property is really realizable, then the BMC procedure described above would produce unsatisfiable QBF formulas for all unfoldings and would never return a result.

The idea of proving QBF-hardness is as follows: The QBF-based BMC procedure classifies a property as unrealizable, iff there exist input sequences of some length k such that independently from the black box actions the property will be violated after k steps. Conversely, the QBF-based BMC procedure is not able to prove unrealizability with an unfolding of length k or smaller, if for each input valuation in each time frame there is an action of the black box such that the property is fulfilled after k steps, and additionally all states on these paths also fulfill the property. Furthermore, if we can prove for this scenario that after at most k steps every state has already been visited before, we can be sure that the QBF-based BMC procedure will *never* produce a satisfiable instance, since for every input pattern it is possible to determine at least one realization of the black box leading to a state which does not violate the property, independently from the length of the unfolding.

This concept is illustrated in Fig. 8. Let $\boldsymbol{s}^0 = \boldsymbol{\sigma}_1^0$ be the initial state which fulfills P. Next, the graph branches for all possible assignments $\boldsymbol{\xi}_1^0, \ldots, \boldsymbol{\xi}_m^0$ to the primary inputs $\boldsymbol{x}^0$. For each of these values $\boldsymbol{\xi}_i^0$ there exists an action of the black box outputs $\boldsymbol{Z}^0 = \boldsymbol{\zeta}_i^0$ leading to next states $\boldsymbol{s}^1 = \boldsymbol{\sigma}_i^1$ which all fulfill P. Once a state is equivalent to a state which was visited before (which is indicated by a dashed backward arrow in Fig. 8 stating that $\boldsymbol{\sigma}_1^1 = \boldsymbol{\sigma}_1^0$, $\boldsymbol{\sigma}_1^2 = \boldsymbol{\sigma}_2^1$, $\boldsymbol{\sigma}_m^2 = \boldsymbol{\sigma}_1^0$, respectively), this branch does not need to be further explored. If at some depth all so far explored states point back to already visited states, then the black box outputs are set in a way that the system remains in "good states" forever, i. e., we are in the situation sketched above and we can be sure that the QBF-based BMC procedure will never produce a satisfiable instance, independently from the length of the unfolding. Thus, determining whether a graph fulfilling the aforementioned properties exists answers the question of whether a design is QBF-hard.



$k=1$
$k=2$
$\vec{s}^0=\vec{\sigma}^0_1$
$\vec{x}^0$
$\vec{\xi}^0_1$
$\vec{\xi}^0_2$
$\vec{\xi}^0_m$
$\vec{Z}^0=\vec{\zeta}^0_1$
$\vec{Z}^0=\vec{\zeta}^0_2$
$\vec{Z}^0=\vec{\zeta}^0_m$
$\vec{s}^1=\vec{\sigma}^1_1$
$\vec{s}^1=\vec{\sigma}^1_2$
$\vec{x}^1$
$\vec{\xi}^1_1$
$\vec{\xi}^1_2$
$\vec{\xi}^1_m$
$\vec{Z}^1=\vec{\zeta}^1_1$
$\vec{Z}^1=\vec{\zeta}^1_2$
$\vec{Z}^1=\vec{\zeta}^1_m$
$\vec{s}^2=\vec{\sigma}^2_1$
$\vec{s}^2=\vec{\sigma}^2_m$

Fig. 8. QBF-Hardness Graph [40].

In [40] it has been shown that the existence of a QBF-hardness graph can be checked using a series of QBF formulas. Once the QBF-hardness of the design under verification is proven, two options are considered. In case of a combined 01X/QBF-modeling an abstraction refinement procedure will identify more black box outputs for QBF-modeling (in an extreme case yielding a pure QBF-modeled problem) and repeat the QBF-based BMC procedure. If all black box outputs are already QBF-modeled, the design is passed to a BMC tool supporting DQBF with Henkin quantifiers providing the next level of accuracy of black box modeling. Fig. 9 illustrates the complete verification flow.

5 Conclusion

In this paper, we have seen that unknown values appear at several points in the design process of a digital circuit. We have presented different methods for modeling such unknown values: (1) 01X-logic, which is efficient, but pessimistic and over-estimates the set of signals which carry an unknown value, (2) quantified Boolean formulas, which constitute an accurate formalism for modeling many problems arising in the test of digital circuits, and (3) dependency-quantified Boolean formulas, which are accurate if partial knowledge has to be taken into account, e. g., when black boxes in a circuit design have only access to a subset of the circuit's signals.

References

1. Roth, J.P.: Diagnosis of automata failures: A calculus and a method. IBM J. Res. Dev. **10**(4) (1966) 278–291
2. Goel, P.: An implicit enumeration algorithm to generate tests for combinational logic circuits. In: Proc. Fault Tolerant Computing Symposium. (1980) 145–151

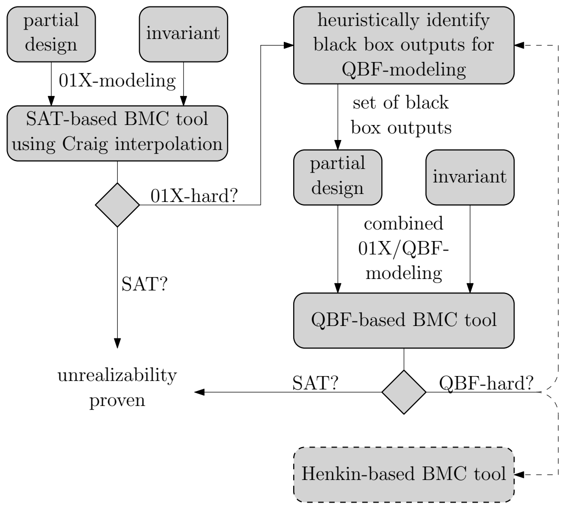


Fig. 9. Workflow.

3. Fujiwara, H., Shimono, T.: On the acceleration of test generation algorithms. IEEE Trans. on Computers **C-32**(12) (1983) 1137 –1144
4. Larrabee, T.: Test pattern generation using Boolean satisfiability. IEEE Trans. on Computer-Aided Design **11**(1) (1992) 4–15
5. Stephan, P., Brayton, R., Sangiovanni-Vincentelli, A.: Combinational test generation using satisfiability. IEEE Trans. on CAD of Integrated Circuits and Systems **15**(9) (1996) 1167–1176
6. Czutro, A., Polian, I., Lewis, M., Engelke, P., Reddy, S.M., Becker, B.: Thread-parallel integrated test pattern generator utilizing satisfiability analysis. International Journal of Parallel Programming **38**(3-4) (2010) 185–202
7. Eggersglüß, S., Drechsler, R.: Atpg based on Boolean satisfiability. In: High Quality Test Pattern Generation and Boolean Satisfiability. Springer (2012) 59–70
8. Wohl, P., Waicukauski, J., Neuveux, F.: Increasing scan compression by using X-chains. In: Int'l Test Conference (ITC). (2008) 1–10
9. Ramdas, A., Sinanoglu, O.: Toggle-masking scheme for X-filtering. In: European Test Symposium (ETS). (2012) 1–6
10. Ahmed, N., Tehranipoor, M.: A novel faster-than-at-speed transition-delay test method considering IR-drop effects. IEEE Trans. on Computer-Aided Design of Integrated Circuits and Systems **28**(10) (2009) 1573–1582
11. Hillebrecht, S., Polian, I., Engelke, P., Becker, B., Keim, M., Cheng, W.T.: Extraction, simulation and test generation for interconnect open defects based on enhanced aggressor-victim model. In: Int'l Test Conference (ITC). (2008) 1–10
12. Muth, P.: A nine-valued circuit model for test generation. IEEE Trans. on Computers **C-25**(6) (1976) 630–636
13. Flores, P., Neto, H., Marques Silva, J.: An exact solution to the minimum size test pattern problem. In: IEEE Int'l Conf. on Computer Design (ICCD). (1998) 510–515

14. Jain, A., Boppana, V., Mukherjee, R., Jain, J., Fujita, M., Hsiao, M.: Testing, verification, and diagnosis in the presence of unknowns. In: IEEE VLSI Test Symposium (VTS). (2000) 263–268
15. Carter, J., Rosen, B., Smith, G., Pitchumani, V.: Restricted symbolic evaluation is fast and useful. In: IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD). (1989) 38 –41
16. Kundu, S., Nair, I., Huisman, L., Iyengar, V.: Symbolic implication in test generation. In: Proc. Conference on European Design Automation. (1991) 492–496
17. Touba, N.: X-canceling MISR : An X-tolerant methodology for compacting output responses with unknowns using a MISR. In: Int'l Test Conference (ITC). (2007) 1–10
18. Tang, Y., Wunderlich, H., Engelke, P., Polian, I., Becker, B., Schloffel, J., Hapke, F., Wittke, M.: X-masking during logic BIST and its impact on defect coverage. IEEE Trans. on Very Large Scale Integration (VLSI) Systems **14**(2) (2006) 193–202
19. Elm, M., Kochte, M.A., Wunderlich, H.J.: On determining the real output Xs by SAT-based reasoning. In: IEEE Asian Test Symposium (ATS). (2010) 39–44
20. Chou, H.Z., Chang, K.H., Kuo, S.Y.: Accurately handle don't-care conditions in high-level designs and application for reducing initialized registers. IEEE Trans. on Computer-Aided Design **29**(4) (2010) 646–651
21. Wilson, C., Dill, D., Bryant, R.: Symbolic simulation with approximate values. In Hunt, W., Johnson, S., eds.: Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD). Vol. 1954 of LNCS. Springer (2000) 507–522
22. Kochte, M.A., Elm, M., Wunderlich, H.J.: Accurate X-propagation for test applications by SAT-based reasoning. IEEE Trans. on Computer-Aided Design **31**(12) (2012) 1908–1919
23. Hillebrecht, S., Kochte, M.A., Wunderlich, H.J., Becker, B.: Exact stuck-at fault classification in presence of unknowns. In: European Test Symposium (ETS). (2012) 1–6
24. Erb, D., Kochte, M.A., Sauer, M., Hillebrecht, S., Schubert, T., Wunderlich, H.J., Becker, B.: Exact logic and fault simulation in presence of unknowns. ACM Trans. on Design Automation of Electronic Systems (TODAES) **19**(3) (2014)
25. Biere, A., Heule, M., van Maaren, H., Walsh, T., eds. Frontiers in Artificial Intelligence and Applications 185. In: Handbook of Satisfiability. IOS Press (2009)
26. Zhang, L., Malik, S.: Conflict driven learning in a quantified Boolean satisfiability solver. In: IEEE/ACM Int'l Conf. on Computer Aided Design (ICCAD). (2002) 442–449
27. Biere, A.: Resolve and expand. In: Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT). Vol. 3542 of LNCS, Springer (2005) 59–70
28. Giunchiglia, E., Marin, P., Narizzano, M.: sQueezeBF: An effective preprocessor for QBFs based on equivalence reasoning. In: Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT). Vol. 6175 of LNCS. Springer (2010) 85–98
29. Bloem, R., Könighofer, R., Seidl, M.: SAT-based synthesis methods for safety specs. In: Int'l Conf. on Verification, Model Checking, and Abstract Interpretation (VMCAI). Vol. 8318 of LNCS, Springer (2014) 1–20
30. Bloem, R., Egly, U., Klampfl, P., Könighofer, R., Lonsing, F.: SAT-based methods for circuit synthesis. In: Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD), IEEE (2014) 31–34
31. Scholl, C., Becker, B.: Checking equivalence for partial implementations. In: ACM/IEEE Design Automation Conference (DAC), ACM Press (2001) 238–243

32. Jo, S., Matsumoto, T., Fujita, M.: SAT-based automatic rectification and debugging of combinational circuits with LUT insertions. In: IEEE Asian Test Symposium (ATS), Niigata, Japan, IEEE Computer Society (2012) 19–24
33. Jo, S., Gharehbaghi, A.M., Matsumoto, T., Fujita, M.: Debugging processors with advanced features by reprogramming LUTs on FPGA. In: Int'l Conf. on Field-Programmable Technology (FPT), Kyoto, Japan, IEEE (2013) 50–57
34. Smith, A., Veneris, A.G., Ali, M.F., Viglas, A.: Fault diagnosis and logic debugging using boolean satisfiability. IEEE Trans. on CAD of Integrated Circuits and Systems **24**(10) (2005) 1606–1621
35. Sülflow, A., Fey, G., Drechsler, R.: Using QBF to increase accuracy of SAT-based debugging. In: Int'l Symposium on Circuits and Systems (ISCAS), Paris, France, IEEE (2010) 641–644
36. Gitina, K., Reimer, S., Sauer, M., Wimmer, R., Scholl, C., Becker, B.: Equivalence checking of partial designs using dependency quantified Boolean formulae. In: IEEE Int'l Conf. on Computer Design (ICCD), Asheville, NC, USA, IEEE Computer Society (2013) 396–403
37. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic Verification of Finite–State Concurrent Systems Using Temporal Logic Specifications. ACM Trans. on Programming Languages and Systems **8**(2) (1986) 244–263
38. Nopper, T., Scholl, C.: Symbolic model checking for incomplete designs with flexible modeling of unknowns. IEEE Trans. on Computers **62**(6) (2013) 1234–1254
39. Miller, C., Kupferschmid, S., Lewis, M.D.T., Becker, B.: Encoding techniques, craig interpolants and bounded model checking for incomplete designs. In: Int'l Conf. on Theory and Applications of Satisfiability Testing (SAT). Vol. 6175 of LNCS, Springer (2010) 194–208
40. Miller, C., Scholl, C., Becker, B.: Proving QBF-hardness in bounded model checking for incomplete designs. In: Int'l Workshop on Microprocessor Test and Verification (MTV), IEEE Computer Society (2013)
41. Sauer, M., Reimer, S., Polian, I., Schubert, T., Becker, B.: Provably Optimal Test Cube Generation Using Quantified Boolean Formula Solving. In: Asia and South Pacific Design Automation Conference (ASPDAC). (2013) 533–539
42. Reimer, S., Sauer, M., Schubert, T., Becker, B.: Using maxbmc for pareto-optimal circuit initialization. In: Int'l Conf. on Design, Automation & Test in Europe (DATE), IEEE (2014) 1–6
43. Cook, S.A.: The complexity of theorem-proving procedures. In: Annual ACM Symposium on Theory of Computing (STOC), ACM (1971) 151–158
44. Stockmeyer, L.J., Meyer, A.R.: Word problems requiring exponential time (preliminary report). In: Annual ACM Symposium on Theory of Computing (STOC), New York, NY, USA, ACM (1973) 1–9
45. Gitina, K., Wimmer, R., Reimer, S., Sauer, M., Scholl, C., Becker, B.: Solving DQBF through quantifier elimination. In: Int'l Conf. on Design, Automation & Test in Europe (DATE), Grenoble, France, IEEE (2015)
46. Fröhlich, A., Kovásznai, G., Biere, A., Veith, H.: iDQ: Instantiation-based DQBF solving. In: Intl. Workshop on Pragmatics of SAT (POS), Vienna, Austria (2014)
47. Tseitin, G.S.: On the complexity of derivation in propositional calculus. Studies in Constructive Mathematics and Mathematical Logic **Part 2** (1970) 115–125
48. Turpin, M.: The dangers of living with an X (bugs hidden in your Verilog). In: Boston Synopsys Users Group Meeting. (2003) 1–34
49. Ulrich, E.G., Baker, T.: The concurrent simulation of nearly identical digital networks. In: Papers on Twenty-five years of electronic design automation. 25 years of DAC (1988) 318–323

50. Waicukauski, J., Eichelberger, E., Forlenza, D., Lindbloom, E., McCarthy, T.: Fault simulation for structured VLSI. VLSI Systems Design **6**(12) (1985) 20–32
51. Antreich, K., Schulz, M.: Accelerated fault simulation and fault grading in combinational circuits. IEEE Trans. on Computer-Aided Design **6**(5) (1987) 704 – 712
52. Lee, H., Ha, D.: An efficient, forward fault simulation algorithm based on the parallel pattern single fault propagation. In: Int'l Test Conference (ITC). (1991) 946–955
53. Rudnick, E., Patel, J., Pomeranz, I.: On potential fault detection in sequential circuits. In: Int'l Test Conference (ITC). (1996) 142–149
54. Schubert, T., Lewis, M., Becker, B.: Antom—solver description. SAT Race (2010)
55. Hillebrecht, S., Kochte, M.A., Erb, D., Wunderlich, H.J., Becker, B.: Accurate QBF-based test pattern generation in presence of unknown values. In: Int'l Conf. on Design, Automation & Test in Europe (DATE). (2013) 436–441
56. Pnueli, A., Rosner, R.: Distributed reactive systems are hard to synthesize. In: Annual Symposium on Foundations of Computer Science, IEEE Computer Society (1990) 746–757
57. Miller, C., Nopper, T., Scholl, C.: Symbolic CTL model checking for incomplete designs by selecting property-specific subsets of local component assumptions. In: Workshop "Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen" (MBMV), Universitätsbibliothek Berlin, Germany (2009) 87–96
58. Nopper, T., Scholl, C.: Approximate symbolic model checking for incomplete designs. In: Int'l Conf. on Formal Methods in Computer Aided Design (FMCAD). Vol. 3312 of LNCS, Springer (2004) 290–305