Boolean Reasoning for Digital Circuits in Presence of Unknown Values

Application to Test Automation

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Michael Andreas Kochte

aus Hoyerswerda, Deutschland

Hauptberichter: Prof. Dr. rer. nat. habil. Hans-Joachim Wunderlich

Mitberichter: Prof. Xiaoqing Wen, Ph.D.

Tag der mündlichen Prüfung: 22. Mai 2014

Institut für Technische Informatik der Universität Stuttgart

2014

Contents

Acknowledgements

It is my pleasure to thank everyone who supported me during the past years. I would like to thank Prof. Hans-Joachim Wunderlich for his professional supervision and for sharing interesting ideas in countless discussions. I would also like to thank Prof. Xiaoqing Wen for his unconditional support during my stay at the Kyushu Institute of Technology in Japan, and for accepting to be the second adviser of my thesis. Both Prof. Wunderlich and Prof. Wen have been extremely supportive during the finalization of the thesis when time was running short.

In addition, I am grateful to Prof. Bernd Becker, Prof. Seiji Kajihara, Prof. Sandip Kundu, and Prof. Kohei Miyase for critical and fruitful discussions all over the world.

I very much enjoyed the work with colleagues and students who were at some time involved in my activities in Stuttgart. Sorry, I cannot list everyone here, you are not forgotten! I'd like to acknowledge those with whom I spend a lot of time in cooperations or personally: Rafał Baranowski, Lars Bauer, Claus Braun, Melanie Elm, Stefan Holst, Michael Imhof, Chang Liu, Eric Schneider, Hongyan Zhang, and Christian Zöllin. A special appreciation for our great cooperation goes to Dominik Erb, Stefan Hillebrecht, and Matthias Sauer from the University of Freiburg. Dominik Erb and Christian Zöllin, thanks for your speedy and thorough proof-reading of this thesis!

A work like this is not possible without administrative and technical assistance: Thank you, Mirjam Breitling, Helmut Häfner, Lothar Hellmeier, and Wolfgang Moser!

A big thank-you also to Kazunari Enokimoto, Kohei Miyase, and Yuta Yamato for making my stay in Japan such an unforgettable experience.

Finally, I thank my parents for their enduring support and encouragement.

Stuttgart, May 2014

Michael Kochte

Abbreviations and Notation

$\wedge$	Logic conjunction
$\vee$	Logic disjunction
$\oplus$	Logic antivalence
$\leftrightarrow$	Logic equivalence
$\rightarrow$	Logic implication
$\neg$	Logic negation
x/y	Value x in the fault-free and value y in the faulty circuit
$\mathcal{A}_X$	Set of possible X-source value assignments
ATE	Automated test equipment
ATPG	Automatic test pattern generation
BDD	Binary decision diagram
BIST	Built-in self test
CNF	Conjunctive normal form
D-value	Value 1 in the fault-free and value 0 in the faulty circuit
DD	Definite detection (of a fault)
EDA	Electronic design automation
func(v)	Function of circuit node v
FFR	Fanout-free region
$G = (V, E)$	Circuit graph G with nodes V and edges E
$G^f = (V^f, E^f)$	Circuit graph G^f under fault f
KTDD	Kleene ternary decision diagram
LFSR	Linear feedback shift register
MATE	Masking aware test set encoding
MISR	Multiple input shift register
NX	Set of nodes with X-value
NX^P	Set of nodes with pessimistic X-value
PD	Potential detection (of a fault)
PPSFP	Pattern-parallel single fault propagation fault simulation

QBF	Quantified Boolean formula
ROBDD	Reduced ordered binary decision diagram
S_X	X-sources in the circuit
SAT	(Boolean) Satisfiability
U-value	Undefined value
val(v, p)	Value of node v under pattern p
val(v, p, a)	Value of node v under pattern p and X-source assignment a
VLSI	Very large scale integration
X-value	Unknown value
X^P-value	Pessimistic X-value
Z-value	High impedance value

Abstract

The exponential growth in digital VLSI design scale and complexity has been enabled by comprehensive adoption of design automation tools. In the digital domain, design automation from design entry over synthesis, validation, verification to test preparation is based on reasoning about logic functions and their manipulation.

Limited knowledge about the circuit behavior may require that nodes in the circuit are modeled as having an unknown value, for instance when using incompletely specified design models. Circuit nodes also need to be modeled as unknown if their values cannot be controlled during operation or test, or if their value during operation is not known at the time of modeling.

To reflect such unknown values in design automation tools, the algorithms typically employ logic algebras with a special symbol 'X' denoting the unknown value. However, the reasoning about functions based on such algebras results in an overestimation of unknown values in the model, and an accurate or optimal solution cannot be found. This pessimism in presence of unknown values causes additional costs at different stages of the design and test process and may even reduce product quality.

This work proposes novel, efficient approximate and accurate algorithms for the analysis of the behavior of digital circuits in presence of unknown values. Heuristics and formal Boolean reasoning techniques are combined to achieve short runtimes. The algorithms allow accurate logic and fault simulation as well as accurate automatic test pattern generation in presence of unknown values. The implications to the overhead and effectiveness of design-for-test structures are studied. The proposed algorithms are the first to completely overcome the pessimism of conventional algorithms found in today's VLSI design automation tools also for larger circuits.

Experiments on benchmark and industrial circuits investigate the pessimism in conventional algorithms and show the increased accuracy achieved by the proposed algorithms. The results demonstrate the benefits of approximate and accurate reasoning in different applications in the VLSI design process, especially in the test automation domain.

Zusammenfassung

Das exponentielle Wachstum der Entwurfskomplexität integrierter Schaltungen wird durch die durchgängige Verwendung von Werkzeugen zur Entwurfsautomatisierung ermöglicht. Für digitale Schaltungsanteile basiert die Entwurfsautomatisierung von der Eingabe über die Synthese, Validierung, Verifikation bis hin zur Testvorbereitung auf der Analyse und Manipulation von Logikfunktionen.

Ist das Schaltungsverhalten nur partiell bekannt, zum Beispiel bei Verwendung unvollständig spezifizierter Modelle, kann über die Werte gewisser Signale keine Aussage getroffen werden. Dies erfordert, dass im Schaltungsmodell entsprechende Signale mit unbekannten Werten modelliert werden. Ebenso müssen Werte von Signalen als unbekannt modelliert werden, wenn ihr Wert während des Betriebs oder der Testdurchführung nicht gesteuert werden kann und zum Zeitpunkt der Modellierung unbekannt ist.

Um diese unbekannten Werte in Entwurfswerkzeugen zu behandeln, verwenden die Algorithmen typischerweise Algebren, die unbekannte Werte mit einem gesonderten Symbol 'X' darstellen. Allerdings führt die Auswertung und Analyse von Funktionen mit solchen Algebren zu einer Überschätzung von Signalen mit unbekannten Werten im Modell, so dass die exakte oder optimale Lösung im Anwendungsbereich nicht gefunden werden kann. Dieser Pessimismus bei Betrachtung von unbekannten Werten verursacht zusätzliche Kosten während des Entwurfs und des Testprozesses. Im schlimmsten Fall kann sogar die Produktqualität darunter leiden.

In der vorliegenden Arbeit werden effiziente approximative und exakte Algorithmen für die Analyse des Verhaltens digitaler Schaltungen in Gegenwart von unbekannten Werten vorgestellt. Diese Algorithmen verbinden Heuristiken und unterschiedliche formale Analysetechniken, um niedrige Laufzeiten zu erreichen. Die entwickelten Algorithmen erlauben die exakte Logik- und Fehlersimulation, als auch die exakte Testmustererzeugung in Gegenwart von unbekannten Werten. Weiterhin werden die Folgen für den prüfgerechten Entwurf untersucht. Die vorgeschlagenen Algorithmen

erlauben erstmals, den Pessimismus in vorherrschenden Algorithmen der Entwurfsautomatisierung vollständig auch für größere Schaltungen zu beseitigen.

Mittels Experimenten anhand von Benchmark- und industriellen Schaltungen wird der Pessimismus in aktuellen Algorithmen untersucht. Die Ergebnisse belegen die Vorteile einer exakten Schaltungsanalyse während des Entwurfs und insbesondere im Bereich der Testautomatisierung.

1 Introduction

The tremendous success of digital integrated circuits is based on the increase of integration scale and density over the past decades, as well as ubiquitous design support by electronic design automation (EDA) tools. EDA tools have been developed to overcome the design complexity gap, the test challenge, and the challenge of reliable design for low power circuits.

The core of most EDA algorithms at gate level is the reasoning about Boolean functions and their manipulation. In the majority of state-of-the-art algorithms, the reasoning about Boolean functions in presence of unknown values is pessimistic and overestimates the number of signals with unknown value. This results in inaccurate and non-optimal solutions with higher costs.

This chapter briefly introduces the reader to the importance of Boolean reasoning in EDA tools. The handling of unknown values in various application fields of EDA tools is also discussed. This motivates the development of novel algorithms to increase accuracy in presence of unknown values. Finally, the organization and contribution of this work are presented.

1.1 Boolean Reasoning in Electronic Design Automation

A large fraction of EDA algorithms requires the representation and manipulation of Boolean functions. Figure 1.1 outlines the design flow of a hardware module comprising the specification and design entry, logic synthesis and optimization, design-for-test integration, as well as physical synthesis. Validation, verification, debug and diagnosis span the whole design process and provide feedback to the single design steps to find and correct design errors as well as manufacturing issues as early as possible. During and after manufacturing, the individual chips are tested for defects. Thorough test-

ing with a high coverage of potential defects is a crucial requirement, especially in safety-critical applications such as the aerospace or automotive industry.

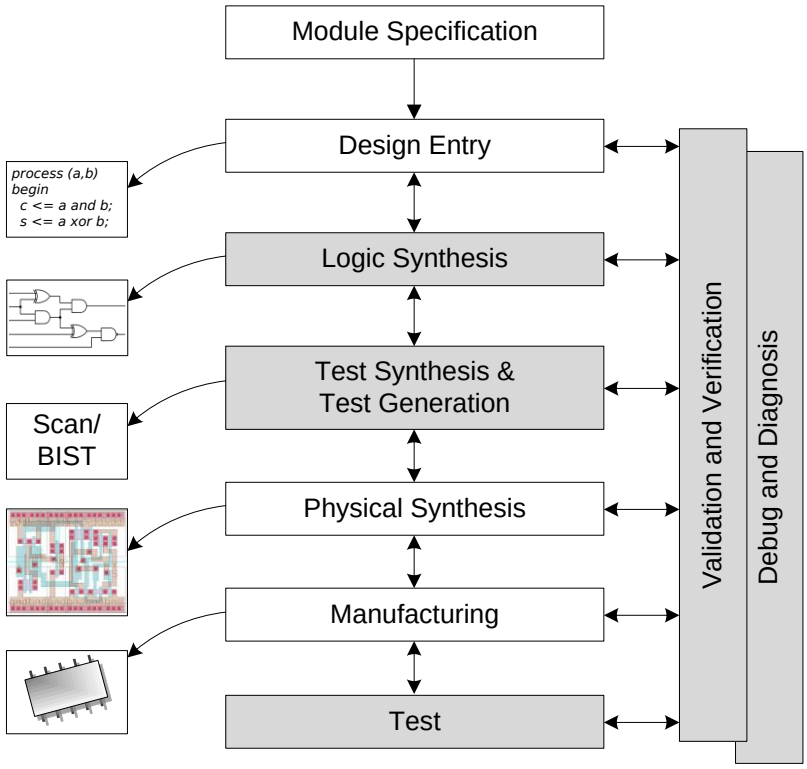


Figure 1.1: Design and manufacturing flow of a hardware module; shaded steps benefit from Boolean reasoning with increased accuracy as proposed in this work.

In Figure 1.1, logic synthesis, validation, verification, debug and diagnosis techniques, as well as test generation fundamentally require algorithms to reason about Boolean functions and to manipulate them. In consequence, the accuracy of the reasoning algorithms critically determines the quality of the solution in the corresponding task. This work focuses on increasing the accuracy of Boolean reasoning in presence of unknown values with immediate benefits for these design steps.

1.2 Unknown Values and Pessimism of Logic Simulation

Unknown values (X-values) describe an unknown state of signals or sequential elements in a circuit or its model. Their value may be undetermined during the design

process, e.g. due to incompletely specified models, or unknown and principally indeterminable during operation.[1] Figure 1.2 depicts the various sources of X-values in the design models and during operation and test.

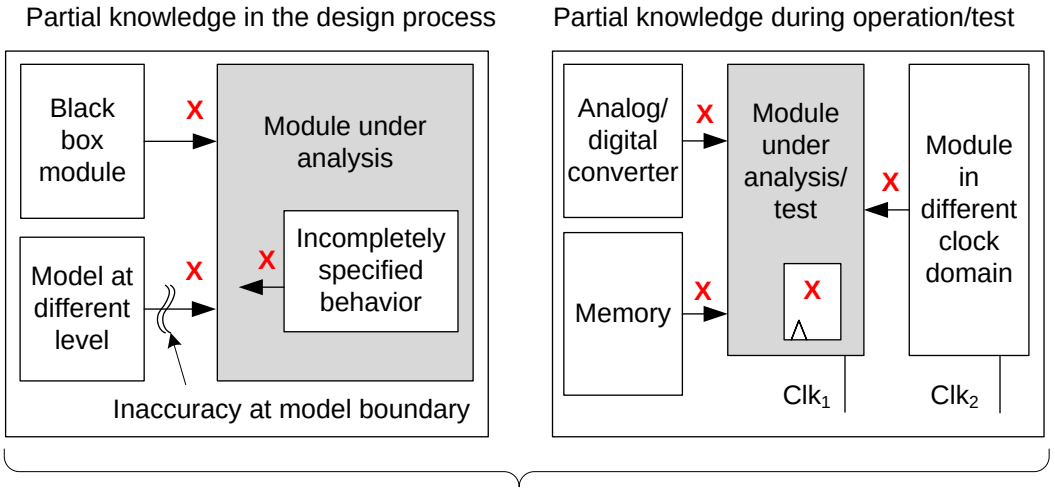


Figure 1.2: Origin of unknown values during the design process, operation and test.

During the design process, partial knowledge of the behavior of the system or of its constituent parts may cause unknown values. In addition, *don't care* values are used in models to express the design intent that a particular state of a signal can be freely chosen by the design tools for optimization.

During operation and test application, the value of certain signals in the circuit may be uncontrollable or not known at modeling time. Such signal values may result from uninitialized memory arrays in the circuit, from uninitialized or uncontrollable flip-flops or latches, or from timing and synchronization issues. They are modeled as X-values.

X-values propagate through combinational logic along sensitized paths. The propagation stops at circuit outputs and scannable elements, at reconvergences of X-valued signals if X-canceling occurs, or is blocked at gates with controlling values.[2] X-values are *canceled out* at a reconvergence if their combination generates a *known* binary value.

[1] However, we require that the state of an X-valued signal is a valid binary value (cf. Section 2.1).
[2] An input with the controlling value of a gate implies a binary output value.

Since the propagation of X-values depends on the state of the circuit and its inputs, it cannot be determined accurately by pattern independent (static) analysis. Given the state of the circuit and its inputs, the *accurate* computation of X-propagation is an NP-complete problem [Chang87]. In practice, the propagation of X-values in a circuit is *estimated* using logic simulation algorithms based on a fixed set of symbols for signal values. The underlying algebraic structures are called n-valued logics. Typically, three, four, or nine-valued logics are used for logic simulation [Breue72, IEEE95, IEEE00].

In addition to 0 (false) and 1 (true), the set of represented values in the commonly used three-valued logic comprises a *single* X-symbol to denote unknown values. If multiple distinct X-values are combined in a circuit, reasoning about the circuit behavior based on the three-valued logic is *pessimistic* and overestimates the propagation of X-values:[3] The values of certain signals are computed as unknown although they actually carry a binary value of either 0 or 1 invariant of the state of the X-sources. Figure 1.3 depicts an example of a circuit structure with one X-source at node b. According to three-valued simulation, the output q produces an X-value under the given input assignment $(a, c) = (1, 1)$. However, an explicit analysis of the two possible assignments to the X-source shows that output q always carries the binary value of 1.

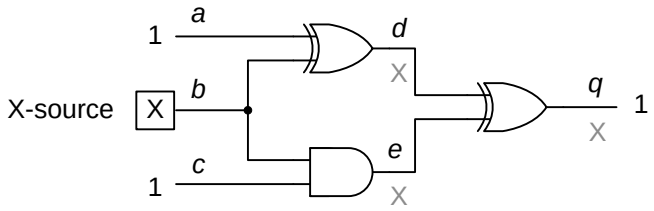


Figure 1.3: Canceling of X-values at signal reconvergence q.

The cause of the overestimation is the limitation of symbols in n-valued logics to denote X-values. This limits the capability to distinguish among different X-values and in consequence, to accurately compute combinations thereof. In the example of Figure 1.3, the values of node d and e are complementary, but this relation cannot be reflected by the single X-symbol in the three-valued logic. Thus, the X-reconvergence at output q can only be pessimistically evaluated.

[3] This pessimistic result is well-defined in the sense that it does not contradict the actual behavior of the circuit for any of the possible assignments to the X-sources.

1.3 VLSI Circuit Testing

The correct operation of a circuit as defined by its specification is threatened by imperfections of the source materials, limitations of the manufacturing process, hostile operation environments, and degradation processes during operation.

Test is a mandatory step in the design process to detect such defects and imperfections in the manufactured devices. During the test application, the behavior of a circuit under test (CUT) is evaluated for a given set of input stimuli or patterns. The comparison of expected responses, given by the specification, and the actual responses of the circuit leads to a pass/fail decision as illustrated in Figure 1.4.

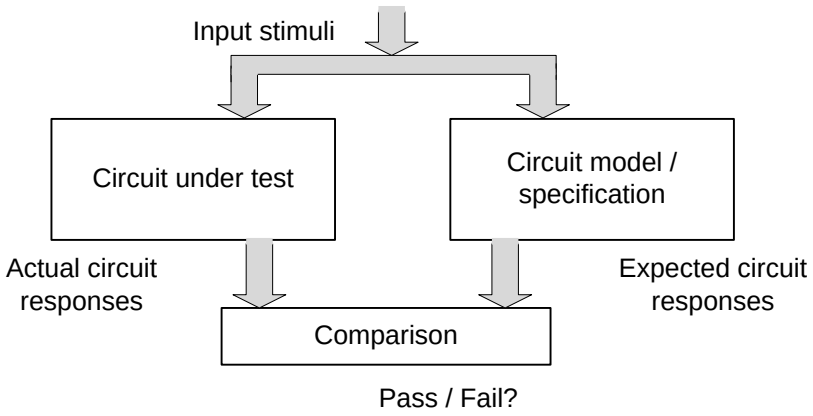


Figure 1.4: Test exercises the circuit under test with a set of input stimuli.

Thorough testing allows to guarantee a specified level of product quality of the devices. Product quality is defined as the percentage of functional shipped chips w.r.t. all shipped chips. Product quality strongly depends on the quality of the test, i.e. the fault or defect coverage.

1.4 Handling of Unknown Values in Testing

During test generation and preparation, X-values in the circuit model have adverse impact on the test quality and need to be reduced as much as possible. X-values reduce the controllability and observability of internal circuit structures. This limits the possibility to thoroughly examine the structures for possible defects by input stimuli. In

addition, design-for-test structures that are used in built-in self-test and embedded deterministic test architectures to reduce the costs of the test process are highly sensitive even to very few X-values.

Figure 1.5 shows a general embedded deterministic test architecture with multiple scan chains, test stimuli decompression and test response compaction [Rajsk04]. X-values may spread from the X-sources to scan chains where they are captured and stored as part of the test response values. During extraction of the test response, the captured values are shifted into response compaction hardware. Due to the nature of response compaction, even slight changes in the input sequence result in a different compacted response. If even a single X-value reaches the compactor, the signature may have an arbitrary value after a few cycles and become completely unusable to reliably distinguish fault-free from faulty test responses.

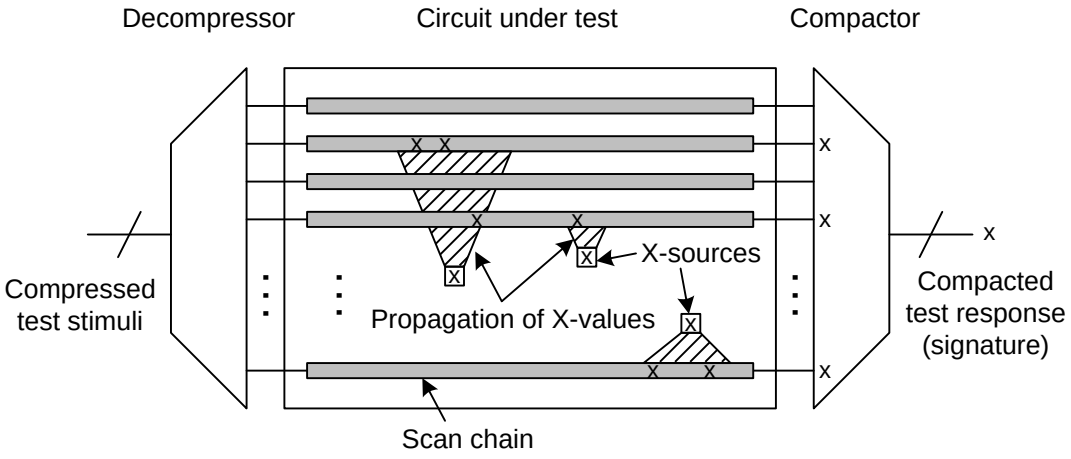


Figure 1.5: Capturing of X-values in an embedded deterministic test architecture.

Both issues, loss of internal controllability and observability, and corruption of signatures in test response compaction, can be remedied by integration of design-for-test structures: The propagation of X-values can be stopped by X-blocking cells, which allow to increase internal controllability and observability at the cost of additional hardware and potentially reduced performance. X-tolerant response compaction and X-masking before compaction are used to obtain valid test signatures if X-values cannot be blocked in the circuit and are captured in the scan chains. In this case, the compaction ratio suffers, or additional compaction control data is required, significantly increasing test data volume. Typically, the incurred hardware, performance or test overhead increases with the number of X-values to be handled.

1.5 Impact of Pessimistic Boolean Reasoning

The inaccurate computation of circuit behavior in presence of X-values has adverse consequences on design, verification and test generation tasks. *Optimistic* computation of X-values in logic simulation may disguise design errors in circuit models if not all possible consequences of X-values in a circuit are considered [Turpi03]. The *pessimistic* analysis, which overestimates the X-values in a circuit, increases the manual verification and debug effort required to investigate potential false positive results [Noppe07]. It may also increase the hardware overhead for required reset circuitry in the design [Chou10].

In test generation, the overestimation of X-values by pessimistic algorithms may increase test-related costs such as test time and test pattern count, and design-for-test hardware overhead. In the worst case, fault coverage and the quality of the final VLSI product are impaired [Kocht11b] resulting in higher costs for customer returns, or even risks in safety critical applications.

The hardware and test data overhead in design-for-test and test compression typically increases with the number of X-values to be handled. An overestimation of X-values is thus undesirable, and may in addition cause loss of response data due to overmasking.

To reduce the cost of pessimistic reasoning in EDA, novel and efficient algorithms are required that can increase the accuracy of reasoning about circuit behavior in presence of X-values.

1.6 Objectives and Contributions of this Work

This work addresses the pessimism of EDA algorithms in presence of X-values. Hybrid algorithms employing heuristic and formal methods are presented, which allow to reason about Boolean functions and their properties in presence of X-values with significantly increased accuracy. Heuristic techniques comprise topological analysis, conventional n-valued logic simulation algorithms, and restricted symbolic simulation. Formal techniques are the symbolic analysis of functions based on decision diagrams as well as the reasoning about satisfiability of Boolean and quantified Boolean formulas, as shown in Figure 1.6.

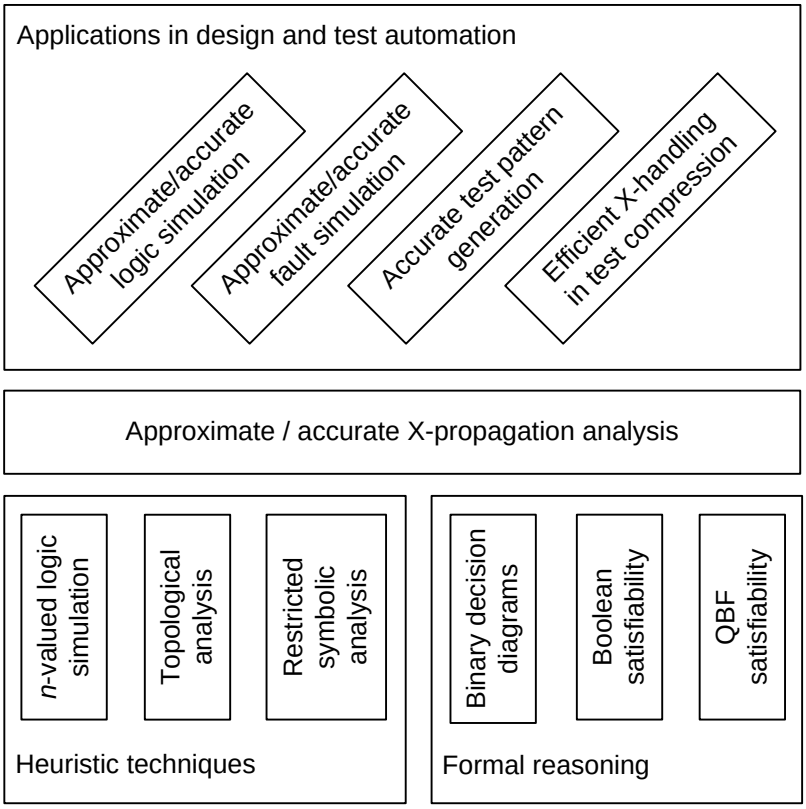


Figure 1.6: Contributions of this work.

By combination of these heuristic and formal techniques, efficient algorithms are constructed which significantly reduce or completely eliminate the pessimism in the analysis of circuits compared to conventional algorithms. The proposed algorithms can be applied and tailored to various application fields in EDA. Here, the focus is on the application to VLSI circuit test. In particular, novel algorithms have been developed for approximate and accurate fault simulation and test pattern generation in presence of X-values. The results of the proposed logic simulation algorithms also enable the optimization of design-for-test structures.

In these application fields, the proposed algorithms result in a significant improvement both over the conventional algorithms—which still form the foundation of commercial tools used in industrial practice—as well as over the state-of-the-art in academic research tools.

Based on the developed algorithms, experimental evaluations are conducted for academic benchmark circuits and industrial circuits. For the first time, the impact of the

inaccuracies in conventional algorithms in logic simulation, fault simulation, and test generation is exactly quantified. These results clearly show that accurate solutions to the tasks in the field of VLSI testing reduce the overhead incurred by pessimistic and approximate solutions, and significantly improve fault coverage and thus product quality.

This work is composed of nine chapters. Chapters 2 to 4 present the fundamentals required in the subsequent chapters. Chapters 5 to 8 discuss the proposed algorithms for Boolean reasoning in presence of unknown values. Chapter 9 summarizes the work and provides a perspective of ongoing and future research directions. The appendices provide basic definitions, additional details of the experimental evaluation, and tabulated results. Appendix C gives a concise statement of the author's contributions to the current state-of-the-art and of the cooperations involved in this work.

2 Sources of Unknown Values

As outlined in the previous chapter, unknown and unspecified values emerge in system models during specification and design. Unknown values also result from modeling signals that are uncontrollable during operation or test of the system, or whose value is not known during modeling. In both cases, the reason is limited knowledge of the system and its components, its environment or operation conditions. This chapter first distinguishes between the concepts of undefined and unknown values, followed by a description of the various sources of unknown values in models and during operation and test.

2.1 Undefined and Unknown Values

In CMOS[1] circuits, two distinct voltage levels, *high* and *low*, represent the logic states *true* and *false*. To operate CMOS circuits reliably, a voltage margin exists in which the state and the behavior of a node is *undefined*. This is shown in Figure 2.1.[2]

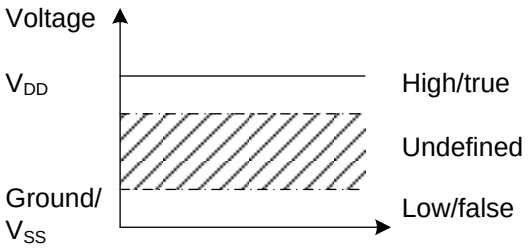


Figure 2.1: Interpretation of voltage levels in a CMOS circuit.

For such undefined voltage levels, the simulation of the circuit behavior requires modeling at electrical level. Even then, minute differences in the operating conditions

[1] Complementary metal oxide semiconductor.
[2] If tristate logic is used, signals may take an additional high impedance state.

or transistor and interconnect parameters, for instance caused by variations, result in unpredictable behavior in different chips.

As a consequence, logic reasoning and simulation at gate or higher abstraction levels produces invalid results. For the circuit in Figure 2.2 a) with an undefined value U at node s, it is possible that both nodes n_1 and n_2 evaluate to 1 depending on how the inverter and the AND gate evaluate the undefined value. Conventional simulation algorithms, discussed in Chapter 5, approximate this behavior pessimistically.

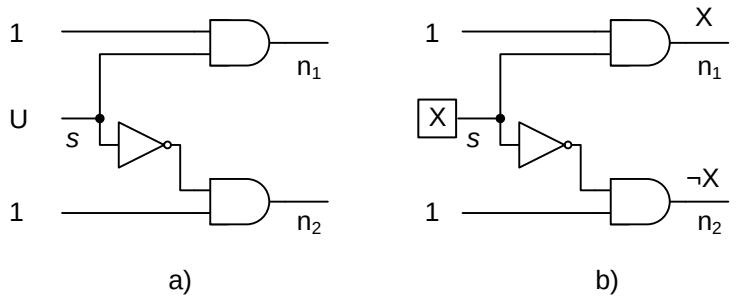


Figure 2.2: Node values in presence of a) an undefined value and b) an unknown value.

An *unknown* value models a valid, well-defined voltage level of either low or high. However, it is unknown which level it is. The resulting behavior differs from the case with undefined values: In the circuit in Figure 2.2 b) with an unknown value X at node s, nodes n_1 and n_2 always assume complementary logic values.

In this work, we target the non-pessimistic, i.e. accurate, evaluation of the circuit behavior in presence of *unknown* values.

2.2 Unknown or Unspecified Values in System and Design Models

In the system specification and design process, partial system models with incomplete specification of structure or behavior often emerge. Incompletely specified models may result from parallel development with multiple developers, unavailable implementation, the combination of incompatible component models or of models at different abstraction levels, or the integration of protected intellectual property.

If a full system model is required, for instance for validation or verification tasks, black box models [Günth00] are used as substitutes to hide or approximate yet unknown implementation details. Depending on the particular design component and model, the black box may produce unknown values at some or all of the outputs of the component.

On the other hand, a model may intentionally not specify the design behavior for all possible cases, i.e. for all system states and input stimuli. For an incompletely specified combinational system, the output behavior is not described for all possible input stimuli either because the outputs are not observed or not relevant during operation, or because certain input stimuli do not occur during operation. In a sequential circuit, a subset of states may be unreachable and not used, or not reached during functional operation. For some states, the output values may be irrelevant or not observed and thus, not specified. In these cases of unspecified behavior, the designer may instruct a synthesis tool to optimize the implementation by assigning a don't care value to the signals or system states.

The representation of these states and their handling in simulation and verification determines the accuracy of the reasoning. In commercial logic simulation tools, unknown and don't care values may cause optimistic as well as pessimistic results. The case of simulation optimism is adverse since it may cause simulation results that conceal design bugs. An example is the evaluation of the Verilog *if* statement during simulation at register transfer level as shown in Algorithm 2.1 below.

Algorithm 2.1 Optimistic evaluation of the *if* statement in Verilog [Turpi03]

```
always @ (CLK or nRESET)
  if (nRESET == 0)
    Count <= 3'b000;
  else if (CountEnable)
    Count <= NxtCount; // no update if CountEnable has an unknown value
```

The Verilog language reference manual specifies that the conditioned statement is executed only if the condition evaluates to *true* [IEEE95]. The statement is not executed if the condition evaluates to *false* or to an unknown or unspecified value. In the example, the second branch is evaluated in simulation only if `CountEnable` equals 1. If `CountEnable` carries an unknown value, i.e. 0 or 1, the simulation optimistically does not evaluate the case when `CountEnable` is 1.

2.3 Unknown Values in System Operation and Test

Different structures in a circuit may require the modeling of unknown values during *functional operation*. The state of uninitialized memory arrays and uninitialized flip-flops or latches is unknown after power-up. When their values are read during operation, unknown values propagate in the circuit. Flip-flops and latches can be set or reset to enforce a defined value. In practice, design constraints may prevent that all sequential elements can be set or reset to a defined value, for instance because of high area and routing overhead of set or reset circuitry and global wiring [Chou10].

During *test application*, uncertain timing, limited control over the circuit or test stimuli, or boundaries to analog blocks may also require modeling of unknown values as follows:

Limited control: Performance or area constraints may pose a restriction on the infrastructure in the system such that not all sequential elements can be set to a defined state. In partial scan design [Agraw88], for example, only a subset of sequential elements is made controllable and observable for the test process, while the remaining elements contain unknown values, as shown in Figure 2.3 a).

Also, asynchronous set or reset signals of sequential elements may produce unknown values during testing if the reset is triggered unintentionally by a test pattern. This can happen for instance during the shift-in of a pattern into a scan chain, or by a pseudo-randomly generated pattern.

Tristate buses are a potential source of undefined values. Bus contention, i.e. multiple gates driving a bus signal at the same time, will cause undefined electrical behavior if complementary values are driven as shown in Figure 2.3 b). On the other hand, if no driver is enabled, the bus signal line has high impedance or floating state. This problem may arise in built-in self-test when pseudo-random patterns are applied. Pull-up or pull-down resistors at the bus signals can remedy the problem at additional hardware and power cost. If the undefined state is sampled in a sequential element or read by a CMOS combinational gate, it is unknown whether it is evaluated as *true* or *false*.[3]

[3]The algorithms presented in this work allow to analyze the propagation of the resulting unknown values in the circuit.

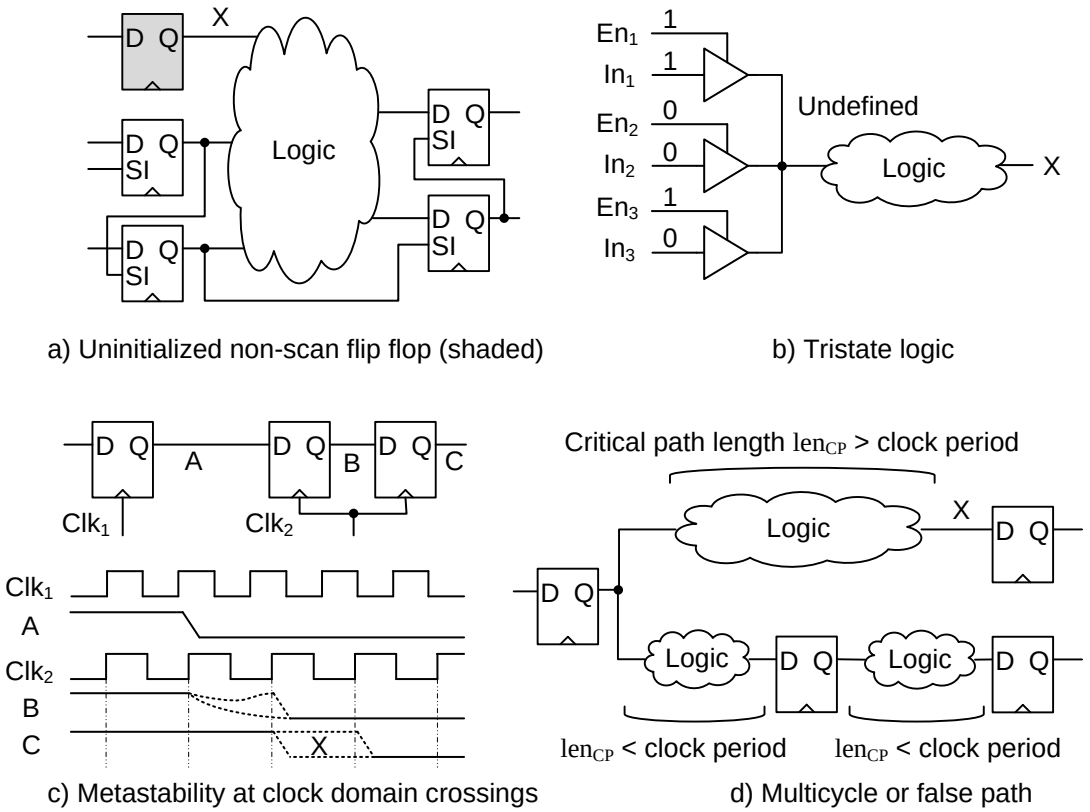


Figure 2.3: Sources of unknown values during operation and test.

Uncertain timing: The timing of a circuit may also be a source of unknown values, especially during test. Combinational feedback loops which oscillate asynchronously w.r.t. the system clock introduce uncertainty if defined states are expected at a certain time. If multiple clocks without a fixed phase relation are used in the system, and data is exchanged between the clock domains, signal metastability [Dike99] at the domain crossing results in unknown values during operation [Feng07] as well as during testing [Wu11]. This can be partially remedied by synchronizing structures as shown in Figure 2.3 c). The two flip-flops driven by Clk_2 form a synchronizing element to ensure stable and well-defined voltage levels at signal C. However, due to subtle differences in gate or interconnect delays, it is unknown in which cycle of Clk_2 signal C assumes the stable target value. If multiple values, for example in a bus, cross a clock domain, this timing uncertainty may cause a divergence of associated values.

Multi-cycle paths with a propagation delay exceeding one clock period can cause unknown behavior during testing, in particular during built-in self-test using pseudo-

random patterns (cf. Figure 2.3 d). In system operation, the end points of such paths are evaluated only after a delay of multiple clock cycles to ensure stable and defined values. In a delay fault test using pseudo-random patterns, this cannot be easily ensured any more. If the output of a multi-cycle path is sampled after just a single clock period, the sampled value is unknown.

False paths in a circuit are not considered during timing optimization and analysis. False paths include paths which cannot be sensitized during functional operation, or whose timing behavior is not relevant for the operation. However, a non-functional test pattern, provided for example by a pseudo-random pattern generator on chip, may sensitize such a path and launch a transition. If the path's propagation delay exceeds the clock period and the output value is sampled, an unknown value may be read, propagate in the circuit and finally compromise the test response.

Power noise and excessive power dissipation during test application can result in a local and temporary drop of power supply voltage, called IR-drop [Saxen03]. Consequently, the transition delay of affected gates or along affected paths increases. If the delay increases beyond the clock period, unknown values may be captured as test responses, which need special handling, for example test response masking [Wen11].

Boundary to analog domains: Analog-to-digital converters generate digital values depending on a sampled physical quantity (current, voltage) at external sensors. It is possible to ensure that the digital output value of such an analog block is within a given bound. However, a particular defined value cannot easily be asserted during operation or test due to noise and limited control of the physical environment.

For the discussed scenarios, design guidelines exist to prevent the occurrence of unknown values at the cost of additional hardware or increased delay. In the field of test, these guidelines are known as scan and built-in self-test design rules [Wang06, Mento08]. Due to their impact on performance and area, the guidelines cannot always be applied. In consequence, sources of unknown values may remain in the circuit.

3 Boolean and Pseudo-Boolean Algebras

This chapter briefly introduces Boolean and pseudo-Boolean algebras. These algebraic structures form the basis of conventional algorithms for the analysis of the behavior of digital (switching) circuits and their manipulation in presence of unknown values. Fundamentals on relations, partially ordered sets, and lattices are provided in Appendix A for completeness. A detailed treatment of the theory of switching circuits can be found in textbooks [Sasao99, Hacht96].

3.1 Boolean Algebras and Functions

An algebra is defined over a set called the underlying or carrier set. It comprises a set of operations over the carrier set and a set of relations.

Boolean Algebras A Boolean algebra $(B, \mathcal{R}_{\leq})$ is a complemented distributive lattice over set B with partial order $\mathcal{R}_{\leq}$ (cf. Appendix A.3). The idempotent, commutative, associative, distributive, and absorption laws hold for the operations $\vee$ (least upper bound) and $\wedge$ (greatest lower bound). Furthermore, each element $b \in B$ has a unique complement.

The cardinality of the carrier set of a finite Boolean algebra is always a power of two. A finite Boolean algebra is isomorphic to the algebra defined over the power set of its carrier set with relation $\mathcal{R}_{\leq}$ as set inclusion, and $\vee$ and $\wedge$ defined as set union and set intersection. The 1 element (unique greatest element) is the carrier set, and the 0 element (unique least element) is the empty set.

Figure 3.1 shows three examples of Boolean algebras as Hasse diagrams. The vertices are labeled with the elements of the power set of the carrier set of one, two and three elements.

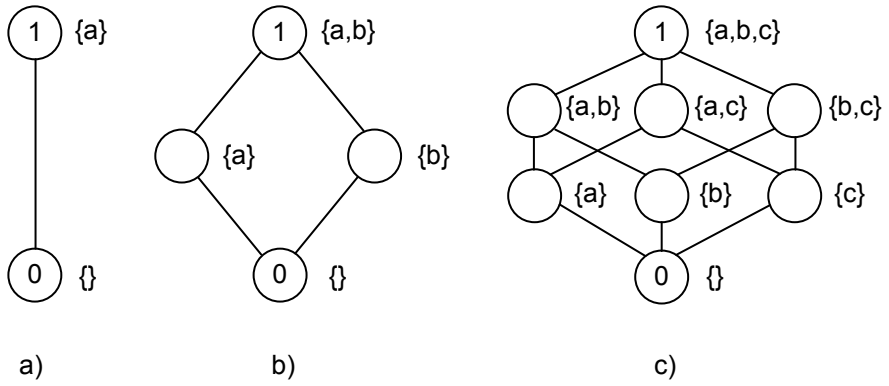


Figure 3.1: Hasse diagrams of two-, four- and eight-valued Boolean algebras over power sets.

If the carrier set of a Boolean algebra has two elements, the algebra is called *two-valued Boolean* or *switching algebra*. In this case the elements are typically denoted 0 and 1, or *false* and *true*. In [Shann38], Shannon shows that the behavior of switching or relay circuits can be expressed and analyzed algebraically using the switching algebra.

Boolean Functions Given a Boolean algebra $(B, \mathcal{R}_{\leq})$, an n-variable Boolean function $f(x_1, \ldots x_n) : B^n \mapsto B$ is a mapping from the n-dimensional Boolean domain B^n to B that is constructed by composition of the unary negation and binary operations $\vee$ and $\wedge$ over elements of B^n in a finite number of steps.

Using Boole's expansion[1] [Boole54], a Boolean function $f : B^n \mapsto B$ can be decomposed w.r.t. variable x_i and represented using its co-factors as:

$$f(x_1, \ldots x_{i-1}, x_i, x_{i+1}, \ldots x_n) = \quad x_1 \wedge f(x_1, \ldots x_{i-1}, 1, x_{i+1}, \ldots x_n) \quad \vee$$
$$\neg x_i \wedge f(x_1, \ldots x_{i-1}, 0, x_{i+1}, \ldots x_n).$$

For the analysis of switching circuits, often the two-valued switching algebra $B = \{0, 1\}$ is employed. The resulting functions are known as switching functions.

[1] Also referred to as Shannon's expansion.

For a switching function f, the set of points of $\{0,1\}^n$ for which f evaluates to 1 is called the *on*-set of f, while the set of points mapped to 0 is called the *off*-set of f. If on- and off-set of f partition $\{0,1\}^n$, then f is called a *completely* specified function. If there are elements in $\{0,1\}^n$ for which a mapping is not specified, the function is called *incompletely* or *partially* specified.

An m-ary Boolean function is defined as the mapping from points in B^n to points in the m-dimensional Boolean space B^m: $f(x_1, \ldots x_n) : B^n \rightarrow B^m$. In the following, functions with range B will be considered unless noted otherwise.

3.2 Multi-Valued Boolean and Pseudo-Boolean Algebras

Many applications in EDA require or benefit from the use of multi-valued algebras with more than two values. Examples are the modeling of incompletely specified functions, logic synthesis, test, or timing analysis. In addition to multi-valued Boolean algebras, this motivated the investigation of multi-valued *pseudo*-Boolean algebras with fewer or relaxed axioms compared to Boolean algebras.[2]

In the following, the application of multi-valued Boolean and pseudo-Boolean algebras to modeling and simulation at gate and switch level, test generation and timing analysis are presented. These applications often require the representation of unknown values. An overview of further applications of such algebras, for example in logic synthesis, can be found in [Mille07].

3.2.1 Three-, Four- and Nine-valued Algebras for Modeling and Simulation

Multi-Valued Algebras for Switch Level Simulation and Signal Resolution

Multi-valued Boolean and pseudo-Boolean algebras have been used for the modeling of circuit behavior at switch or transistor level. The carrier set of these algebras contains symbols to distinguish different signal states and signal driving strengths.

[2] In other contexts, pseudo-Boolean functions refer to a mapping of elements of B^n to integer or real numbers. Here, the term pseudo-Boolean algebra shall refer to a non-Boolean algebra that embeds a switching algebra and is used in the analysis of digital circuits.

Depending on the resolution (number of symbols), different magnitudes of electrical currents in the circuit can be distinguished. Signal resolution is the computation of the signal state resulting from multiple drivers of one signal line as shown in Figure 3.2 a). Signal resolution is required when simulating circuits with pass transistor logic, for instance latches or tristate circuitry [Hayes86a].

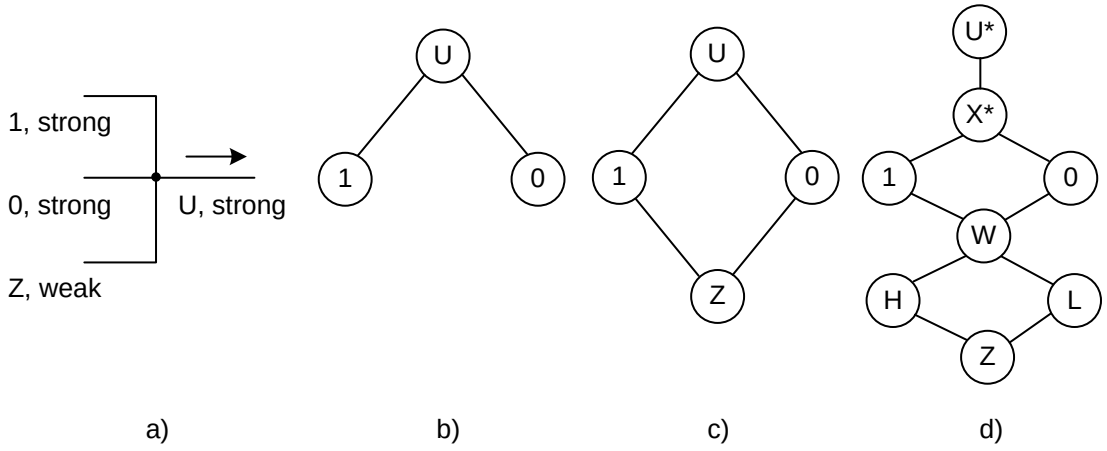


Figure 3.2: Signal resolution: a) Driver contention and resulting U-value, b)-d) Hasse diagrams of three-, four- and eight-valued algebras for signal resolution at switch or logic level.

A three-valued algebra for signal resolution with a single driving strength is given in Figure 3.2 b). If a node has multiple drivers, its state is determined by the least upper bound of the drivers' states. If nodes with different states drive a target node, the resolved state is undefined and represented by a U-symbol.[3] This U-symbol can also be used to specify an unknown or undefined input stimulus [Hayes86a]. As discussed in Section 2.1, this work excludes undefined electrical states.

The three-valued algebra can be extended with a state Z to model a high impedance state with no or low driving strength. This results in a four-valued Boolean algebra over the set $\{0, 1, \mathrm{U}, \mathrm{Z}\}$. Signal resolution is again mapped to the computation of the least upper bound of the driving states. Any of the states $\{0, 1, \mathrm{U}\}$ overrides state Z. The Hasse diagram of this algebra is shown in Figure 3.2 c).

In hardware modeling languages such as Verilog or VHDL, even more symbols are used for signal resolution. For the data types `std_ulogic` and `std_ulogic_vector` in VHDL, nine symbols are used to reflect different signal values and strengths as

[3] In [Hayes86a], the character 'X' is used for this undefined state. In this work, unknown (X) and undefined (U) states are distinguished.

specified in IEEE standard 1164 [IEEE00]. Eight of these symbols are depicted in the Hasse diagram in Figure 3.2 d).[4] The ninth value "-" for *don't care* is not relevant for signal resolution, but only used in modeling and synthesis of incompletely specified functions. The Verilog standard specifies eight different signal strengths in the IEEE standard 1364 [IEEE95]. The extension of the binary values with multiple discrete signal strengths for switch level simulation is generalized in [Hayes86a].

Three-Valued Pseudo-Boolean Algebra for Logic Simulation

The binary switching algebra can be extended by a third symbol X to express an unspecified or unknown value. Such an algebra is often used to specify and reason about incompletely specified functions and the behavior in presence of unknown values.

Different three-valued logic algebras have been proposed by Lukasiewicz, Post and Kleene. They embed the two-valued Boolean algebra, but differ in their interpretation of the X-symbol. Here, only the strong Kleene [Kleen52] logic is considered since its interpretation of the X-symbol is suitable to pessimistically describe the behavior of circuits without contradicting actual circuit behavior in presence of unknown values.

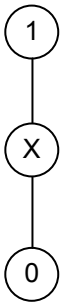
The Kleene logic, in the following referred to as three-valued logic, defines the negation and binary operations $\vee$ and $\wedge$ over the set $\{0, 1, \mathrm{X}\}$ as shown in Figure 3.3. The tables in Figure 3.3 b) also list the behavior of the binary implication and equivalence operations.

This three-valued algebra is used in logic simulation to pessimistically compute the response of a digital circuit in presence of unknown values. Multi-valued algebras with more symbols find application in timing analysis, fault simulation, and test generation as explained in the following.

3.2.2 Multi-Valued Algebras for Hazard Timing Analysis

A hazard in combinational logic is a transient pulse or glitch on a signal node induced by a change of the input values [Unger95]. A hazard may be captured by a sequential element and result in an error. In designs with asynchronous set or reset signals or

[4] X* for undefined value, U* for uninitialized value according to VHDL notation.



(a) Hasse diagram

a	$\bar{a}$
0	1
X	X
1	0

$\vee$	0	X	1
0	0	X	1
X	X	X	1
1	1	1	1

$\wedge$	0	X	1
0	0	0	0
X	0	X	X
1	0	X	1

$\rightarrow$	0	X	1
0	1	1	1
X	X	X	1
1	0	X	1

$\leftrightarrow$	0	X	1
0	1	X	0
X	X	X	X
1	0	X	1

(b) Negation, $\vee$, $\wedge$, implication and equivalence

Figure 3.3: Three-valued Kleene logic algebra.

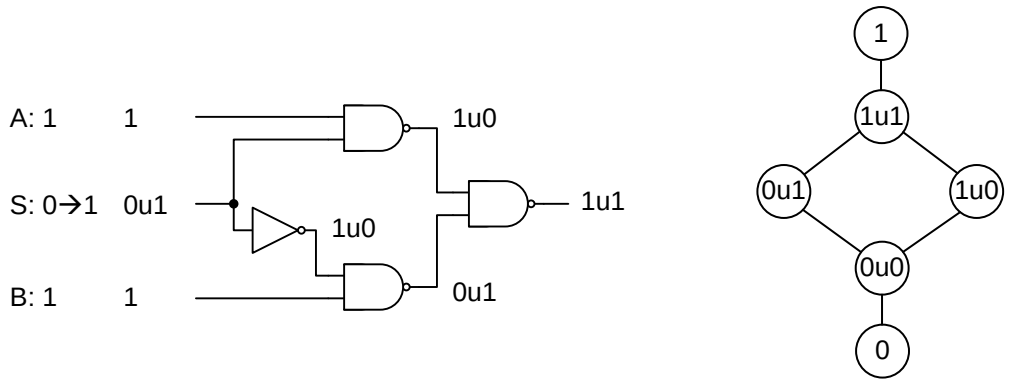
combinational feedback paths, the possibility of hazard generation must be analyzed to avoid for example an unintentional reset.

A static hazard is a transient pulse on a node whose steady state equals the initial state. A dynamic hazard is a transient pulse while the node transitions from one binary value to its complement in the steady state. Hazards may be caused by single or multiple input value changes. In circuits without XOR cells, hazards due to single input changes only emerge at the reconvergences of paths if the branches toggle between controlling and non-controlling values.

A six-valued algebra has been proposed for the analysis of static hazards in combinational circuits in [Yoeli64, Eiche64a, Eiche64b]. The principle idea is to use additional values to express the signal state uncertainty during a value transition. This uncertainty is propagated from the changing input towards the circuit outputs by a breadth-first traversal of the circuit netlist. An example for a multiplexer circuit is given in Figure 3.4. Input s transitions from 0 to 1. This transition is represented by the symbol $0u1$. The Hasse diagram in Figure 3.4 b) defines the operations $\wedge$ and $\vee$ over the six symbols.

Hazard detection based on this six-valued algebra pessimistically computes the possibility of hazard generation and propagation at gates. Potential hazards or u states are blocked at a gate if at least one gate input has the static controlling value of the gate. Otherwise, hazards are propagated.

To distinguish static and dynamic hazards, an eight-valued algebra has been proposed



a) *0u1*-based Simulation of a static-1 hazard

b) Hasse diagram of the six-valued algebra

Figure 3.4: Hazard analysis for a multiplexer circuit using the six-valued algebra.

in [Hayes86b], followed by algebras with even larger carrier sets. In [Brzoz01], n-valued algebras for hazard analysis are generalized, resulting in the *change counting* algebra with potentially infinite carrier set. This algebra explicitly represents all possible value transitions resulting from input value changes by a bit-vector at each node.

3.2.3 Logic Algebras for Fault Simulation and Test Generation

Fault simulation is the computation of the set of faults in a circuit detected by a given test pattern. Automatic test pattern generation is the process of computing test patterns for a given set of target faults.[5] Both in fault simulation and test generation it is required to distinguish potentially different signal values in the fault-free reference circuit and the faulty circuits. Dedicated multi-valued logic algebras have been introduced for this purpose.

The first sound and complete algorithm, called the *D*-algorithm,[6] and the corresponding algebra for test generation was proposed by Roth in 1966 [Roth66]. The D-algorithm employs the five values $\{0, 1, D, \neg D, \mathrm{X}\}$. Symbol 0 (1) denotes that a signal has the same value of 0 (1) in both fault-free and faulty circuits. Symbols D and $\neg D$ represent the effects of a fault. D is an abbreviation of *defect*. The symbol D represents the case that a signal has value 1 in the fault-free, and the value 0 in

[5] Fault simulation and test generation algorithms are discussed in detail in Chapter 6 and 7.
[6] The D-algorithm is complete only in absence of X-sources.

the faulty case (and vice versa for symbol $\neg D$). The X-symbol represents an unknown signal value during test generation or fault simulation.

If X-values do not need to be considered, a four-valued Boolean algebra over the symbols $\{0, 1, D, \neg D\}$ can be defined. The four values are sufficient to represent all possible states in the fault-free and faulty circuit. Figure 3.5 a) shows the Hasse diagram for this Boolean algebra where the least upper bound (greatest lower bound) corresponds to the disjunction (conjunction) of the signal states.

When X-values need to be considered, Roth proposed to use five values. Figure 3.5 b) and c) show the Hasse diagrams for the partial order corresponding to disjunction and conjunction of these five values. Since the two partial orders do not coincide, this structure is not a lattice.

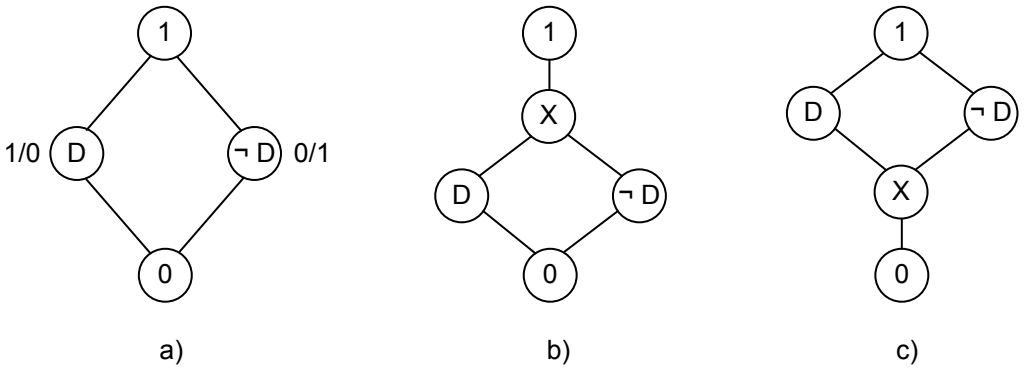


Figure 3.5: Hasse diagrams of a) the four-valued Boolean algebra; b) the partial order of the five-valued D-algebra corresponding to disjunction and c) the partial order of the D-algebra corresponding to conjunction

The X-symbol introduced in the D-algorithm by Roth subsumes multiple states into a single value: The X-symbol is used if the signal state is unknown in the fault-free, in the faulty or both in the fault-free and faulty circuits. This indifference causes a loss of information and reduces accuracy when reasoning about fault effects in in presence of unknown values. Test generation and fault simulation become pessimistic, i.e. for a testable fault a test pattern is not found, or the actual fault coverage is underestimated. This pessimism was first reflected in test generation for sequential circuits in [Muth76].

The pessimism is reduced by two additional symbols to distinguish the three cases where only partial knowledge exists about the signal states. The nine-valued algebra proposed in [Muth76] uses the three values $\{0, 1, \mathrm{X}\}$ both in the fault-free and faulty circuits to reduce the pessimism. The Hasse diagram of the resulting nine-valued

pseudo-Boolean algebra is depicted in Figure 3.6. The tuple x/y denotes value x in the fault-free circuit and value y in the faulty circuit.

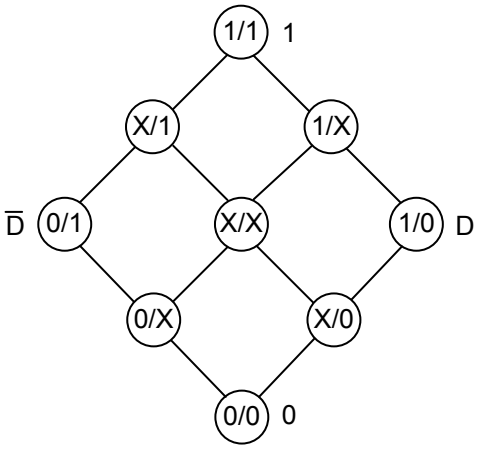


Figure 3.6: Hasse diagram of the nine-valued algebra for test generation.

This nine-valued algebra distinguishes the possible binary values in the fault-free and faulty circuits. Still, the symbol X/X expresses the limited knowledge about one out of four signal states: 0/0, 0/1, 1/0 and 1/1. These four states are distinguished by using knowledge about the signal relation in the fault-free and faulty cases. The ten-valued algebra of [Takam85] uses an additional symbol to express that the value of a signal node is unknown, but it has identical values in the fault-free and faulty circuits. In the eleven-valued algebra of [Cheng88], both an equivalence and antivalence relation between fault-free and faulty circuits are distinguished.

Akers approached the inaccuracy of Roth's five-valued algebra by constructing a Boolean algebra based on the four values $\{0, 1, D, \neg D\}$ and allowing different degrees of uncertainty of signal states [Akers76]. The 16 symbols in the algebra specify which subset of the four values are possible or required at a signal node. Thus, the algebra is defined as the power set algebra over $\{0, 1, D, \neg D\}$, and its partial order relates the symbols w.r.t. their degree of uncertainty. The empty set element $\emptyset$ represents a contradiction, originally denoted as κ. The set of all four values denotes complete lack of knowledge or lack of constraints at a signal node. The remaining 14 values specify all other combinations of values and comprise the values of the five- and nine-valued algebras discussed above. Table 3.1 below lists the values in the discussed algebras for test generation. It illustrates the value correspondences between the algebras, and the indistinguishable values in each algebra.

Table 3.1: Value domains of multi-valued algebras for test generation.

16-valued	11-valued	10-valued	9-valued	5-valued	4-valued
$\emptyset$ (κ)	—	—	—	—	—
{0/0}	0/0	0/0	0/0	0	0
{0/1}	0/1	0/1	0/1	$\neg D$	$\neg D$
{0/0, 0/1}	0/X	0/X	0/X	X	—
{1/0}	1/0	1/0	1/0	D	D
{0/0, 1/0}	X/0	X/0	X/0	X	—
{0/1, 1/0}	Antival.	X/X	X/X	X	—
{0/0, 0/1, 1/0}	X/X	X/X	X/X	X	—
{1/1}	1/1	1/1	1/1	1	1
{0/0, 1/1}	Equival.	Equival.	X/X	X	—
{0/1, 1/1}	X/1	X/1	X/1	X	—
{0/0, 0/1, 1/1}	X/X	X/X	X/X	X	—
{1/0, 1/1}	1/X	1/X	1/X	X	—
{0/0, 1/0, 1/1}	X/X	X/X	X/X	X	—
{0/1, 1/0, 1/1}	X/X	X/X	X/X	X	—
{0/0, 0/1, 1/0, 1/1 }	X/X	X/X	X/X	X	—

The higher accuracy of multi-valued algebras increases the efficiency of test generation algorithms since a larger number of necessary implications can be deduced from the current state of the search for a test pattern [Rajsk90, Cox94].

However, this does not reduce the pessimism inherent to the modeling of unknown values in either the fault-free or faulty circuits by a single symbol. Even the 16-valued algebra is in principle unable to correctly evaluate reconvergences of signals with unknown value since the underlying value domain is limited to $\{0, 1, \mathrm{X}\}$. This makes it impossible to distinguish among different unknown states or to compute their combination accurately. In the general case, the result is a pessimistic computation of signal values. Chapter 5, 6 and 7 present algorithms to reduce or completely eliminate this pessimism in logic and fault simulation, as well as during test generation in presence of unknown values.

4 Representations of Logic Functions and Circuit Modeling

Chapter 4 introduces the representations of Boolean and pseudo-Boolean functions and digital circuits relevant for this work. The reasoning about the behavior and the manipulation of these functions based on their representation is discussed. The discussed algorithms are later employed in the proposed analysis methods in presence of unknown values.

4.1 Representation of Logic Functions

Boolean and pseudo-Boolean functions are used to describe the behavior of switching circuits. The reasoning about their properties and their manipulation requires a representation of the function. In the simplest case, a function can be represented by a formula or a truth table. Since the effort to manipulate a function depends on the properties and size of its representation, sophisticated data structures have been developed and are now in wide use.

4.1.1 Representation as Formula

This section presents Boolean and quantified Boolean formulas, as well as formula based canonical representations.

Boolean Formulas For a Boolean algebra $(B, \mathcal{R}_{\leq})$, a propositional Boolean formula over variables x_i with domain B is a symbol concatenation defined as follows [Hacht96]:

- ▷ A symbol x_i and the constants 0, 1 are Boolean formulas.

- ▷ If g is a Boolean formula, then $\neg g$ is a Boolean formula.
- ▷ With Boolean formulas g, h: $(g) \vee (h)$ and $(g) \wedge (h)$ are Boolean formulas.[1]
- ▷ A Boolean formula is derived by application of the three previous rules in a finite number of steps.

The semantics of a formula map it to a Boolean function, defined for instance as in [Hacht96].

The occurrence of a variable in a formula either in positive or negated form is called a literal. A conjunction of literals is called product term, a disjunction of literals is called a sum. If the conjunction (disjunction) contains all variables of the function, it is called minterm (maxterm). A conjunction of disjunctions is called conjunctive normal form (CNF), a disjunction of conjunctions is called disjunctive normal form (DNF) or sum of products.

Quantified Boolean Formulas A quantified Boolean formula (QBF) is an extension of the expressiveness of Boolean formula in which variables can be quantified by the existential $\exists$ or universal $\forall$ quantifier [Cadol98, Biere09]. In general, the use of quantifiers allows a more succinct representation of Boolean functions. A QBF is defined as follows:

- ▷ A propositional Boolean formula is a QBF.
- ▷ If g is a QBF, then $\exists x\ g$ and $\forall x\ g$ are QBFs.
- ▷ If g is a QBF, then $\neg g$ is a QBF.
- ▷ With QBFs g, h: $(g) \vee (h)$ and $(g) \wedge (h)$ are QBFs.
- ▷ A QBF is derived by application of the above rules in a finite number of steps.

For the formula $\exists x\ g$ ($\forall x\ g$), the occurrence of variable x in $\exists x$ ($\forall x$) is called quantified. Formula g is the scope of x. If variable x occurs in the scope of $\exists x$ or $\forall x$, the occurrence is called bound. A non-quantified occurrence of variable x that is not in the scope of $\exists x$ or $\forall x$ is called free. A variable is called free if it has a free occurrence. A QBF is called closed if it does not contain any free variables.

[1] Parentheses can be dropped assuming precedence of $\wedge$ over $\vee$.

A QBF is in prenex normal form if all quantifiers are grouped together and precede an unquantified Boolean formula, called the matrix or kernel. A QBF can be transformed into a logically equivalent prenex normal form [Biere09].

The semantics of a QBF extend the semantics of a Boolean function by considering variable quantification. Intuitively, formula $\exists x\ g$ is satisfiable if there is some value assignment to x such that g is satisfied, and formula $\forall x\ g$ is satisfiable if g is satisfied for all value assignment to x. A formal definition of the semantics of a QBF is given in [Biere09].

Canonical Forms of Boolean and Quantified Boolean Formulas In general, the representation of a Boolean function by a formula is not unique. Deciding whether two formulas represent the same function is an NP-complete (propositional formula) or NP-hard (quantified formula) problem. A canonical (unique) representation of Boolean functions allows to compare the represented functions by comparing their representations.

A CNF (DNF) formula is canonical if each disjunction (conjunction) is a maxterm (minterm). In that case, two formulas can be syntactically compared to determine the equivalence of the represented functions in polynomial time w.r.t. the size of the formula.

The prenex normal form of a QBF can be transformed into a canonical form by transforming the matrix into canonical CNF (DNF) and using a fixed order of the quantifiers.

4.1.2 Table-Based Representation

Boolean or pseudo-Boolean functions can be explicitly represented by a truth table in which the function values are listed according to variable assignments. With this canonical representation, the equivalence check of two representations requires exponential time in the number of variables.

4.1.3 Cube-Based Representation

The cube-based representation emerged as an efficient data structure used in two-level logic minimization. In the cube-based representation, the on-set of an n-variable

binary Boolean function f is represented as a set of vertices in an n-dimensional space. Function f is represented by a set of product terms, also called cubes. A product of m literals defines an $(n-m)$-dimensional hyper-cube or subspace. A single vertex of the n-dimensional space is denoted by a minterm or product of n literals. A set of cubes is a cover of function f, if and only if all vertices corresponding to the *on*-set of f are contained in the set of cubes, but no vertices of its *off*-set.

The cube calculus is a set of operations defined on a cube or a set of cubes to compute inversion, intersection or union of the represented vertices in the n-dimensional space. Furthermore, complex operations such as Boole's expansion can be computed. The operations can be efficiently implemented as bit-parallel vector operations.

The cube calculus is applied in logic synthesis, e.g. in the two-level heuristic algorithm ESPRESSO [Brayt84] or in the D-algorithm for test generation [Roth66]. It is also used in an early algorithm for accurate logic simulation in presence of unknown values [Chand89] (cf. Chapter 5).

4.1.4 Graph-Based Representation

The idea of a graph-based representation of the function implemented by a switching circuit was introduced by Lee in 1959 [Lee59]. Lee proposed binary decision diagrams, which have later been extended to multi-valued decision diagrams. In particular, ternary decision diagrams have been proposed to represent partially specified functions and unknown values.

Binary Decision Diagrams

Binary decision diagrams (BDDs) are graph-based representations of Boolean functions for use in logic synthesis [Lee59, Akers78]. They did not gain importance until 1986 when Bryant proposed structural restrictions on BDDs that lead to an efficient canonical representation of Boolean functions [Bryan86].

Following [Bryan86], a BDD is a rooted directed and acyclic graph $G = (V, E)$ with two types of vertices: A non-terminal vertex or leaf $v \in V$ has a label $index(v) \in \{1, \ldots, n\}$ and two successors $high(v), low(v) \in V$, also called the *true* and *false* successors, re-

spectively. An $index$ corresponds to a variable of the represented function. A terminal vertex $v \in V$ has a label $value(v) \in \{0, 1\}$.

Furthermore, if $low(v)$ ($high(v)$) of vertex v is not a terminal vertex, then $index(v) < index(low(v))$ ($index(v) < index(high(v))$). This imposes a total order of variable indices on every path from the BDD root to a terminal vertex.

The BDD G with root v represents the Boolean function f_v defined recursively as follows:

- ▷ If v is a terminal vertex:
  If $value(v) = 1$, then $f_v = 1$. If $value(v) = 0$, then $f_v = 0$.
- ▷ If v is a non-terminal vertex and $index(v) = i$, then

$$f_v(x_1, \ldots, x_n) = x_i \wedge f_{high(v)}(x_1, \ldots, x_n) \vee \neg x_i \wedge f_{low(v)}(x_1, \ldots, x_n).$$

Thus, a non-terminal vertex v with $index(v) = i$ corresponds to Boole's expansion of f w.r.t. variable x_i. The $high$ and low successors of v represent the co-factors of f.

Two decision diagrams G, G' are isomorphic if there is a bijective function σ that maps each vertex v of G onto a vertex $\sigma(v) = v'$ of G' such that either both v, v' are terminal vertices and $value(v) = value(v')$, or both v, v' are non-terminal vertices with $index(v) = index(v')$, $\sigma(low(v)) = low(v')$, and $\sigma(high(v)) = high(v')$.

A binary decision diagram is *reduced* if there is no vertex v with $low(v) = high(v)$, and if there are no distinct vertices v, v' with isomorphic sub-graphs in G rooted by v, v'. Reduced ordered binary decision diagrams (ROBDDs) are a canonical graphical representation of Boolean functions [Bryan86]. The equivalence of two ROBDDs with identical variable order is checked by comparison of the ROBDDs in linear time w.r.t. their size.

The size of an ROBDD depends on the variable order. The search for an optimal variable order is an NP-complete problem [Tani93]. Practical approaches use heuristics based on structural dependence analysis of variables, and randomized or dynamic variable reordering [Fujit91, Rudel93, Somen99]. There exist functions which have ROBDDs of exponential size independent of the variable order [Bryan91]. The most prominent examples are multipliers.

In shared ROBDDs [Minat90], isomorphism is exploited not only in ROBDD sub-graphs of one function, but also across ROBDDs of multiple Boolean functions that share variables. One (shared) ROBDD is constructed for all functions assuming the same variable order, and sub-graphs are reused when possible. The equivalence of two functions can be determined by a simple comparison of the root vertices of the shared ROBDDs. If the root vertices are equal, the represented functions are equal as well. This strong canonicity property allows to check equivalence in constant time.

Figure 4.1 below depicts the ROBDD of the carry function of a full adder for the variable order $a < b < c_{\text{in}}$.

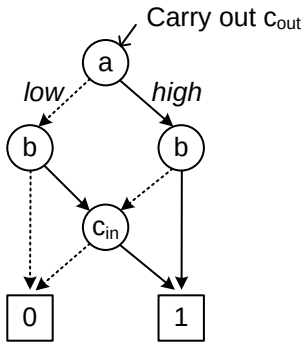


Figure 4.1: Carry function of a full adder $c_{\text{out}} = (a \wedge b) \vee (c_{\text{in}} \wedge (a \oplus b))$ represented as reduced ordered binary decision diagram.

The discussed BDDs are based on Boole's expansion. Other factorizations, for instance based on the Reed-Muller/Davio factorizations [Kebsc92], as well as combinations of factorizations in one BDD have been investigated in [Drech98].

Ternary Decision Diagrams

In multi-valued [Srini90] and multi-terminal decision diagrams [Clark93, Bahar93], more than two distinct values of the terminal nodes are allowed to increase the expressiveness of the representation.

The Kleene ternary decision diagram (KTDD, [Jenni95b, Sasao97]) extends ROBDDs by allowing non-terminal vertices to have an additional X-successors if the variable may carry an unknown value, and by distinguishing three different values {0, 1, X } for terminal vertices. A KTDD can represent partially specified functions. It embeds the

ROBDD of the completely specified function. KTDDs also allow to accurately reason about functions in presence of unknown values.

In a KTDD, the sub-graph reachable from the X-successor of vertex v expresses symbolically the difference between the function $f_{low(v)}$ of the low-successor of v and the function $f_{high(v)}$ of the $high$-successor of v based on Kleene's alignment operation [Kleen52, Mukai86]. The alignment $\Diamond(f_{low(v)}, f_{high(v)})$ evaluates to a binary value only if the function represented by vertex v evaluates to the same binary value for the two possible assignments to v (cf. Figure 4.2). In that case, f_v is independent of the assignment to variable v. Otherwise, the value of f_v evaluates to X [Jenni94, Jenni95b].

$$\text{Alignment } \Diamond(a,b) = \begin{cases} a, & \text{if } a = b \\ \text{X}, & \text{otherwise.} \end{cases}$$

Align. $\Diamond$	0	X	1
0	0	X	X
X	X	X	X
1	X	X	1

Figure 4.2: Kleene's alignment operation.

Figure 4.3 a) illustrates the principal structure of a KTDD. Figure 4.3 b) shows the KTDD of the carry function of a full adder where input b generates an unknown value. The shaded vertices differ from the ROBDD of the function (compare to Figure 4.1).

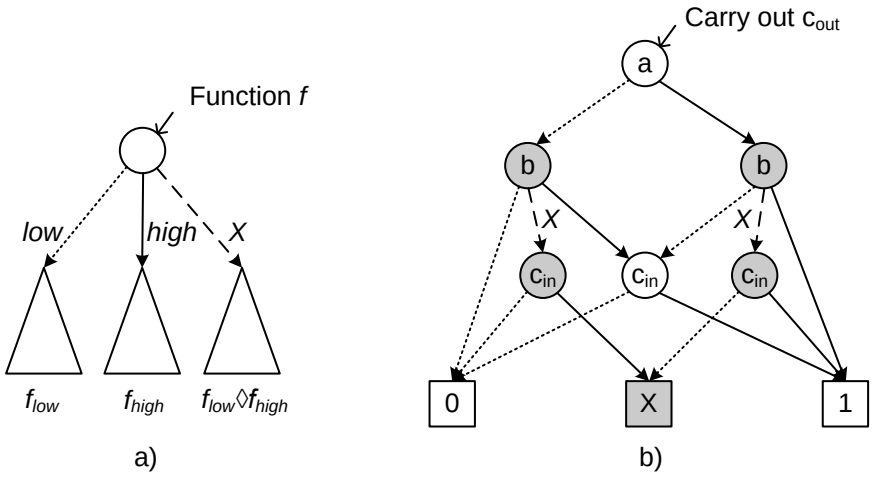


Figure 4.3: Kleene ternary decision diagram: a) Principal structure [Sasao97]; b) Carry function of a full adder $c_{\text{out}} = (a \wedge b) \vee (c_{\text{in}} \wedge (a \oplus b))$ with input b as X-source represented as KTDD.

4.2 Circuit Modeling

The model of a VLSI circuit abstracts its behavioral, structural or physical aspects at different levels. Gajski's Y-chart [Gajsk83] subsumes this understanding. For this work, a structural representation at gate level is required and explained below. For representations at other abstraction levels, the reader may refer to [Gajsk09].

4.2.1 Structural Circuit Modeling

A circuit netlist, i.e. the circuit structure at gate level, is a representation of the circuit constituents (gates, cells, ports) and their interconnection. The netlist can be represented graphically as shown in Figure 4.4 a).

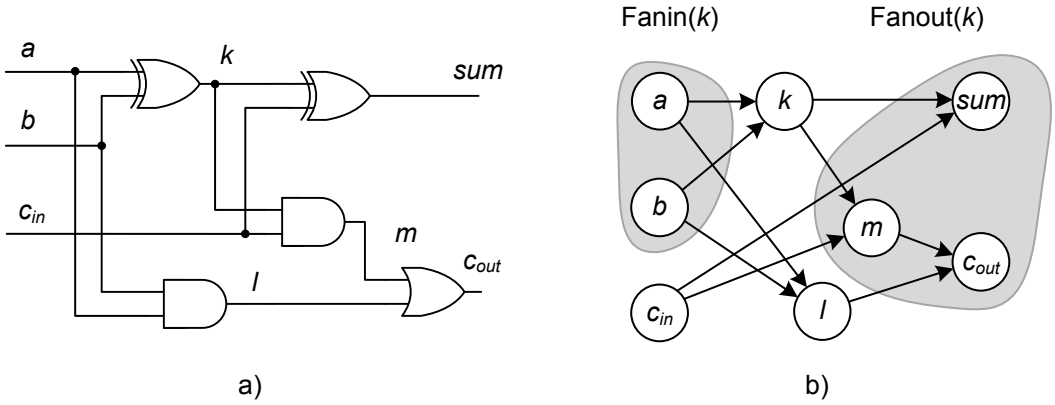


Figure 4.4: Representation of a full adder circuit as a) netlist at gate level and b) circuit graph.

For the automated, algorithmic processing of netlists, a well-defined graph-based model is required. The netlist can be directly transformed into an isomorphic circuit graph which is a directed graph $G = (V, E)$ with vertices V and edges $E \subseteq V \times V$. The vertices represent the elements of the netlist, i.e. input and output ports, gates and complex cells. Each vertex v is assigned a label func(v) that represents the function of the corresponding netlist element, e.g. func$(l) =$ AND. Figure 4.4 b) shows the circuit graph of the full adder netlist in Figure 4.4 a).

Let $succ \subseteq V \times V$ be the binary relation that expresses successorship of two vertices, and let $pred \subseteq V \times V$ be the binary relation that expresses precedence of two vertices.

For the example in Figure 4.4 b), $(a, k) \in succ$ since vertex k is a direct successor of a, and $(k, a) \in pred$ since a is a direct predecessor of k.

The vertices V can be partitioned into two sets I and C. I represents the input ports of the circuit, and C the remaining elements, i.e. gates and cells: $I = \{v \in V \mid |pred(v)| = 0\}$, $C = V - I$. In addition, the circuit output nodes are defined as $O = \{v \in V \mid |succ(v)| = 0\}$.

The transitive fanin or fanin cone of a vertex is defined as the transitive closure of the $pred$ relation of the vertex: Fanin$(v) = \{u \in V \mid pred^*(u, v)\}$. Correspondingly, the transitive fanout or fanout cone of a vertex is defined as the transitive closure of the $succ$ relation: Fanout$(v) = \{u \in V \mid succ^*(v, u)\}$. The cone of influence or support region [Hamza98] of vertex v in G describes the set of vertices which may influence the signal propagation from v to a circuit output. It is the union of the transitive fanin of all vertices in the transitive fanout of v. The shaded areas in Figure 4.4 b) show the transitive fanin and fanout of vertex k in the circuit graph. In this example, the support of k comprises all vertices in the graph.

4.2.2 Structural Modeling of Sequential Circuits

In a combinational circuit, the output response is solely determined by the current input stimuli and the implemented function. In a sequential circuit, the output response also depends on the present state of the circuit. Figure 4.5 shows the principal structure of a sequential circuit including the memory elements, which store the state, and the combinational part to compute state transitions and output values.

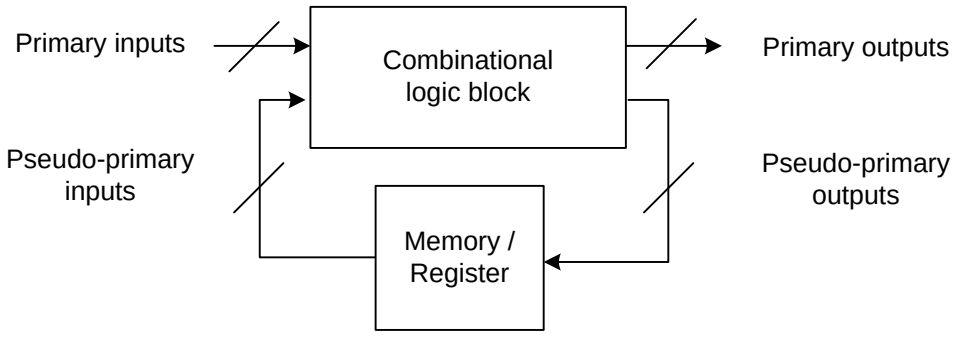


Figure 4.5: Structure of a sequential circuit.

The circuit graph of a sequential circuit may contain cycles. For modeling purposes, the combinational part of the sequential circuit is often considered separately from its memory elements. The sequential nature of a circuit can be expressed by an S-graph which reflects the topological reachability of memory elements [Wunde89].

4.2.3 Behavioral Circuit Modeling

The circuit graph $G = (V, E)$ represents the structure of a circuit at gate level. A forward traversal of this graph from inputs to outputs allows to extract a behavioral model of the circuit, for instance as a Boolean formula. This is achieved by introducing Boolean or pseudo-Boolean variables for inputs, outputs and vertices in the circuit graph. Then, for each vertex $v \in V$, its variable is assigned the corresponding function func(v) over the variables of its predecessors. This mapping from the structure to the behavior yields a formula for each vertex [Wunde91]. Substitution of variables of intermediate vertices by their formula allows to construct a formula for the outputs depending only on input variables.

In an analogous way, the relation based representation of a circuit netlist can be constructed by mapping the function of each vertex to its characteristic equation. The introduction of variables for intermediate vertices as proposed in the Tseitin transformation [Tseit68] allows to generate this relation as formula in CNF in linear time w.r.t. the size of the circuit graph.

4.3 Formal Boolean Reasoning

The reasoning about certain properties of Boolean functions—examples are the tautology check, the satisfiability check, or the equivalence check of two functions—requires algorithms tailored to the representation. Two groups of algorithms are widely used for formal reasoning about Boolean functions: Satisfiability-based analysis, and symbolic algorithms such as the cube calculus or graph-based reasoning.

The decisions about the properties above are NP-complete problems, and the best algorithms known today exhibit exponential runtime or memory requirements in the worst-case (typically w.r.t. the number of input variables). The following two sections

summarize the techniques used in this work to decide these properties efficiently for many practical problem instances. A detailed overview is given in [Biere09].

4.3.1 Satisfiability of Boolean and Quantified Boolean Formulas

Boolean Satisfiability The problem of Boolean satisfiability, i.e. the computation of a variable assignment (model) for a formula of a Boolean function f such that f is satisfied (evaluates to *true*), or to prove that such an assignment does not exist, is an NP-complete problem [Cook71]. The known complete algorithms have exponential time complexity and may require exponential space w.r.t. the number of variables.

The commonly used algorithms process a formula in CNF.[2] The first complete algorithm to decide Boolean satisfiability was the Davis-Putnam (DP) algorithm [Davis60]. The DP algorithm iteratively constructs formulas which are satisfiability-equivalent to the initial formula by resolution of clauses until no new resolvents are found, or until the empty clause has been resolved. In the first case, the represented function is satisfiable. In the latter case, it is unsatisfiable since an empty resolvent clause is the result of two contradicting clauses. The number of resolvents can grow exponentially w.r.t. the number of variables.

The basis of most modern SAT solvers is the Davis-Putnam-Logemann-Loveland (DLL, or DPLL) algorithm [Davis62] which only requires linear space in the number of variables. The DPLL algorithm performs a depth-first search in the possible variable assignments. If a conflicting partial or complete assignment is found, the algorithm backtracks in the search tree to the last decision. At that point, the alternate decision is tried. Efficiency is increased by unit or Boolean constraint propagation, and by pure literal assignment.

To avoid searching similar parts of the search tree multiple times, dynamic conflict analysis, learning and non-chronological backtracking are applied [Zhang01]. If a conflicting assignment is found, the solver searches for the conflicting clauses in an implication graph, which captures all the implications, i.e. consequences, of variable assignments. From these clauses a conflict clause is resolved, which evaluates to *false* if the conflicting condition occurs. The conflict clause is added to the problem instance, effectively pruning an infertile part of the search space.

[2] In the context of Boolean satisfiability, a disjunction of literals in a CNF formula is called clause.

During conflict analysis, the decision that triggered the conflict is also identified. This allows to backtrack directly to the decision causing the conflict without iterating through multiple chronological backtracks. To reduce the effort for unit propagation and backtracking, SAT solvers use optimized data structures for clauses and literals such as *watched literals* [Moske01].

If multiple similar SAT instances are processed, incremental SAT solving avoids the repeated overhead of construction of the whole instance and allows to reuse learnt conflict clauses. Incremental SAT solvers allow to derive the next instance to process from the current one by removal or addition of constraints or clauses [Whitt01, Een03].

Using SAT solvers, the satisfiability or tautology check of a Boolean formula can be directly implemented. The equivalence of two functions f and g is decided by searching for a satisfying assignment to $\neg(f \leftrightarrow g)$, which distinguishes the two functions. This construction is also called a miter [Brand93].

Satisfiability of Quantified Boolean Formulas While the decision of satisfiability of a Boolean formula is an NP-complete problem, the satisfiability of a quantified Boolean formula (QBF-SAT) in general is a PSPACE-complete problem [Garey79]. However, the number of quantifier alternations between existential and universal quantifiers in the prenex form allow to distinguish different complexities, defining the polynomial time hierarchy [Garey79].

A QBF solver processes problem instances that are typically given in prenex normal form with the matrix in conjunctive normal form. The solver searches for a model or assignment to the existentially quantified variables that satisfies the QBF, or proves that a satisfying model does not exists.

The development of algorithms for the QBF-SAT problem started with approaches based on resolution [Bünin95] and depth-first search [Cadol98]. Topological approaches map a QBF-SAT instance to a circuit netlist and then search for satisfying input assignments [Pigor09].

Algorithms based on depth-first search can exploit the available techniques of Boolean satisfiability algorithms. In particular, [Zhang02] demonstrated the application of conflict-driven learning in QBF-SAT instances. Recently, first results on the incremental construction and evaluation of QBF-SAT problems have been reported in [Marin12].

4.3.2 Graph-Based Reasoning and Manipulation of Logic Functions

Canonical representations based on decision diagrams shift the complexity of the reasoning and manipulation of Boolean functions to the construction step of the decision diagrams. While the decision of properties of logic functions and their manipulation can be performed in polynomial time w.r.t. the size of the representation, the construction of the canonical decision diagram may require an exponential amount of time and space w.r.t. the number of variables.

For a given ROBDD, the tautology check, satisfiability check and equivalence check (if shared ROBDDs are used) reduce to constant time comparisons of vertices in the graph.[3] The Boolean difference is computed by checking for variable occurrence in the graph.

The manipulation of functions (inversion, Boolean combination, composition, restriction) is efficient w.r.t. the decision diagram size and requires polynomial time effort [Bryan86].

[3] Similar considerations hold for other types of canonical decision diagrams [Sasao97].

5 Logic Simulation in Presence of Unknown Values

This chapter discusses combinational and sequential logic simulation algorithms at gate-level with focus on the pessimism when X-values need to be considered. It introduces the state-of-the-art in accurate simulation and presents a novel accurate SAT-based combinational logic simulation algorithm [Elm10, Kocht12] in Section 5.2.5 which is applicable to larger circuits. Section 5.3 introduces approximate simulation algorithms and presents an efficient BDD-based algorithm [Kocht11a] that allows to trade-off computational requirements and achievable simulation accuracy. Furthermore, the use of heuristics in hybrid accurate simulation [Hille12] for runtime reduction is discussed in Section 5.4, and an extension for accurate sequential simulation [Erb14a] is presented in Section 5.5.3. Finally, the pessimism in conventional simulation algorithms is quantified in experimental evaluations.

5.1 Introduction

Circuit simulation is the analysis of the behavior of a circuit based on the point-wise evaluation of its model. Simulation can be performed at different modeling levels from electrical level, switch level, logic level, register-transfer level up to behavioral levels such as the transaction level.

The state-of-the-art in logic simulation is based on n-valued logic algebras used for the computation of node values. In presence of unknown values, n-valued logic simulation algorithms are inaccurate and do not compute the signal states accurately. The algorithms compute a *pessimistic* result and overestimate the number of signals with unknown value.

In contrast to a point-wise simulation, *symbolic* simulation of a circuit expresses its behavior functionally, dependent on a specific set of variables.

In the following, we focus on logic simulation algorithms at gate level. Conventional logic simulation can be performed by evaluating the response of every gate to a given input stimulus, also known as plain simulation, or by evaluating only the subset of gates which are affected by value changes w.r.t. a previous value or state. In this latter case of event-based simulation, state changes propagate as events through the circuit model. In both cases of simulation, the gate evaluation is based on an n-valued algebra as discussed in Section 3.2.

These n-valued simulation algorithms are used in design validation and verification, for instance to check logic initializability [Singh00], in hazard timing analysis [Yoeli64, Eiche64a, Brzoz87], or in fault simulation and test generation [Bushn00, Wang06]. Consequently, the results in these applications are pessimistic in presence of unknown values.

5.1.1 Terminology

Signal nodes at which unknown values originate are called X-sources. The set of X-sources is denoted by $S_X \subseteq V$. Each X-source generates a valid binary value of either 0 or 1. However, it is not known which of the two is generated (cf. Section 2.1). There are $2^{|S_X|}$ possible binary assignments $\mathcal{A}_X = \{0,1\}^{|S_X|}$ to the X-sources. Nodes in the transitive fanout of an X-source are called X-dependent.

Logic simulation of a circuit computes for a given input stimulus the state of each circuit node. Let function val(v,p,a) return the binary value $\{0,1\}$ of node $v \in V$ under input pattern p and X-source assignment $a \in \mathcal{A}_X$. The three-valued logic (cf. Section 3.2.1) distinguishes between the binary logic values 0 and 1, and a state with unknown value, in the following denoted as *pessimistic X-value* or X^P. In contrast, accurate simulation determines the nodes with an actual X-value. Thus, for the following discussion, the four values $0, 1, \mathrm{X}^P$, and X of a circuit node v under pattern p are distinguished and defined as follows:

- ▷ Binary value 0: $\forall$ Assignments $a \in \mathcal{A}_X$: val$(v,p,a) = 0$.
- ▷ Binary value 1: $\forall$ Assignments $a \in \mathcal{A}_X$: val$(v,p,a) = 1$.

- ▷ Pessimistic X-value X^P: The value of node v is neither 0 nor 1 according to a pessimistic simulation algorithm. For a node with a pessimistic X-value, it is not known whether input pattern p implies a constant binary value at v independent of the state of the X-sources, or whether v depends on the X-sources.
- ▷ Actual X-value X: Node v depends on the assignments of at least one X-source, i.e.

$$\exists \text{ Assignments } a_1, a_2 \in \mathcal{A}_X \ : \ \mathrm{val}(v, p, a_1) \neq \mathrm{val}(v, p, a_2). \tag{5.1}$$

For brevity, let function $\mathrm{val}(v, p)$ return the value $\{0, 1, \mathrm{X}^P, \mathrm{X}\}$ of node $v \in V$ under pattern p in presence of the X-sources S_X. For a given pattern, the logic simulation result partitions the signal nodes V w.r.t. their state as follows:

- ▷ Set $\mathrm{NX}^P = \{v \in V \mid \mathrm{val}(v, p) = \mathrm{X}^P\}$ comprises all nodes with a pessimistic X-value.
- ▷ Set $\mathrm{NX} = \{v \in V \mid \mathrm{val}(v, p) = \mathrm{X}\}$ comprises all nodes with an X-value.

Note that $\mathrm{NX} = \mathrm{NX}^P$ in a circuit without reconvergences, but in general, $\mathrm{NX} \subseteq \mathrm{NX}^P$. The difference $\mathrm{NX}^P - \mathrm{NX}$ are nodes whose value can be proven to be binary and independent of the assignments of the X-sources by accurate simulation. Figure 5.1 illustrates the partitioning of the circuit nodes w.r.t. their value.

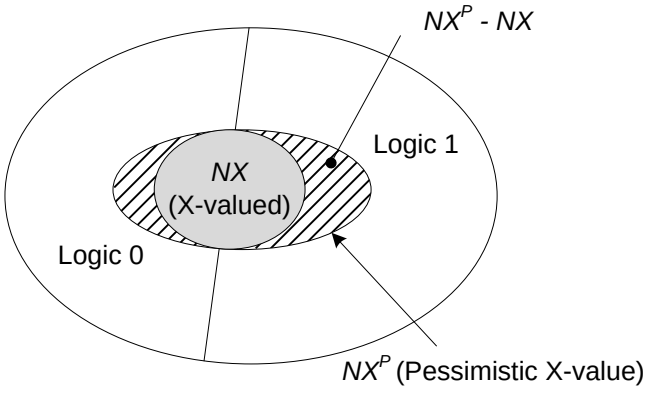


Figure 5.1: Partitioning of circuit nodes w.r.t. their value into 0, 1, NX, and NX^P.

As an example, consider the circuit in Figure 5.2 with one X-source at node b. When this circuit is simulated under input pattern $p : (a, c) = (1, 1)$ using the three-valued pseudo-Boolean algebra of Section 3.2.1, the nodes d, e, f are classified as X^P, i.e. $\mathrm{NX}^P = \{b, d, e, f\}$. Simulating the two possible assignments 0 and 1 to X-source b

explicitly reveals that node f has in both cases value 0 and is in fact independent of b. Thus, NX $= \{b, d, e\}$.

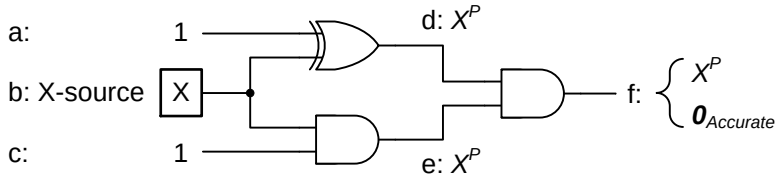


Figure 5.2: Difference between three-valued and accurate simulation at signal f.

5.1.2 Problem Statement and Computational Complexity

The *accurate* logic simulation of a circuit in presence of unknown values computes the node values $\{0, 1, \mathrm{X}\}$ for a given input pattern or stimulus. Input to the simulation is the circuit graph $G = (V, E)$ of the circuit under simulation, the set of X-sources $S_X \subseteq V$ and an input pattern p. Accurate sequential simulation computes the node values for a sequence or a vector of input patterns $\vec{p}$.

Let IS-X-VALUE be the decision problem of computing whether a circuit node v has an X-value, or a binary value independent of the X-sources under a given input pattern p. The computation of IS-X-VALUE can be mapped to the search of a satisfying assignment of the Boolean Formula (5.1). If Formula (5.1) is satisfiable, then there exist at least two different X-source value assignments that imply different values at node v. If Formula (5.1) is not satisfiable, then node v has a binary value invariant of the assignments to the X-sources.

Complexity of Pessimistic Simulation

Pessimistic logic simulation computes the values of circuit nodes for a given input pattern pessimistically as $0, 1$ or X^P. Using a three-valued simulation algorithm, an upper bound of the signals with X-value is computed in linear time w.r.t. the circuit size. This pessimistic solution is well-defined in the sense that the computed binary values do not contradict the accurate values or actual behavior of the circuit. Approximate simulation algorithms with increased accuracy, discussed in Section 5.3, further refine this upper bound and are able to correctly classify a large fraction of node values in the circuit at low computational cost.

NP-Completeness of Accurate Simulation

IS-X-VALUE, i.e. the decision whether the value of a node depends on the X-sources, is an NP-complete problem:

- ▷ IS-X-VALUE lies in NP since a satisfying assignment for Formula (5.1) is verified by computing the value of v using two-valued logic simulation requiring linear runtime w.r.t. $|V|$.
- ▷ IS-X-VALUE is proven to be NP-hard by a polynomial time reduction of the CIRCUIT-SAT problem [Cook71] to an IS-X-VALUE instance [Chang87].

The proof of NP-hardness is as follows: Given a combinational circuit with one output o. CIRCUIT-SAT determines whether there is an assignment to the circuit inputs I such that o evaluates to 1. This is equivalent to deciding IS-X-VALUE for o assuming that *all* circuit inputs are X-sources, i.e. $S_X = I$. This transformation obviously completes in polynomial time.

If IS-X-VALUE is true, then there exists an assignment to S_X (respectively the circuit inputs I) such that $o = 1$. Otherwise, output o has a binary value invariant of the state of the X-sources. The value is computed by two-valued logic simulation of the circuit for an arbitrarily chosen assignment a to S_X with runtime linear in $|V|$. If val$(o, p, a) =$ 1, then $o = 1$ and the CIRCUIT-SAT instance is satisfiable. If val$(o, p, a) = 0$, then $o = 0$ and the CIRCUIT-SAT instance is unsatisfiable.

5.1.3 Three-Valued Logic Simulation

Conventional logic simulation based on the three-valued algebra over $\{0, 1, \mathrm{X}^P\}$ is used to determine a pessimistic superset of propagation paths of X-values and X-valued circuit outputs. In such a simulation, the propagation of X^P-values stops at a gate only if the X^P-value is blocked because of a controlling value of one of the off-path signals. Reconvergent propagation paths of X^P-values are evaluated pessimistically since the limited number of logic values does not allow to distinguish different X-states in the circuit.

Algorithm 5.1 shows the principle of plain three-valued logic simulation. Each signal node in the circuit is processed in topological order. The value of a node is computed

for example by a table lookup in dependence of the value of the corresponding node predecessors.

Algorithm 5.1 Principle of three-valued plain logic simulation.

```
1: ▷ V: set of nodes in the circuit
2: ▷ p: input pattern, p_i: value of input i ∈ I
3: ▷ val_v: value of node v
4: procedure SIMULATECIRCUIT3VLOGIC(p, V)
5:     for v ∈ V in topological order, starting from circuit inputs do
6:         if v ∈ I then                                   ▷ v is an input
7:             val_v := p_v
8:         else                    ▷ Lookup value based on function and input values
9:             val_v := table_lookup(func(v), pred(v))
10:        end if
11:    end for
12: end procedure
```

With respect to the example in Figure 5.2, this computation yields a pessimistic result since the limited symbol set and the operations (cf. Figure 3.3) of the three-valued logic used during the value lookup cannot express the value complementation at node d, and consequently fail to accurately evaluate the reconvergence at node f.

5.2 Accurate Logic Simulation Algorithms

This section discusses complete and sound algorithms to compute the accurate signal node values in logic simulation in presence of unknown values. Section 5.2.5 presents a novel accurate SAT-based algorithm applicable to large circuits.

5.2.1 Exhaustive Enumeration and Simulation

A straightforward way to compute the exact signal states of a circuit is the simulation of all possible assignments to the X-sources for each pattern under simulation. In this case, a two-valued logic simulator can be used for the simulation of the exhaustive pattern set.

In a circuit with k independent X-sources, there exist 2^k possible value assignments $\mathcal{A}_X$ to the X-sources. An exhaustive simulation and analysis of these 2^k assignments

for each pattern under simulation yields the accurate simulation result in presence of X-values [Breue72, Abram90]: If node v has the identical value for all assignments to the X-sources, i.e. $\forall a \in \mathcal{A}_X$: val$(v, p, a) = 0$ or $\forall a \in \mathcal{A}_X$: val$(v, p, a) = 1$, then it does not depend on the X-sources. Otherwise, node v actually carries an X-value.

Exhaustive simulation is applicable to compute the accurate node values if the circuit size is small, and the number of X-sources is low.

5.2.2 Cube-Based Evaluation

The first non-trivial algorithm for the accurate evaluation of a node value in presence of X-values was proposed in [Chand89]. The algorithm requires the representation of the node function and its negation as cubes. The X-sources are considered as circuit inputs with unspecified value. An input pattern is then a partial input assignment and also represented as a cube.

The algorithm checks if the partial input assignment is covered by a cube of the function, and thus the function evaluates to *true*, or if the cube intersection of the function and the input assignment is empty. In the latter case, there are no points in the partial input assignment, i.e. no assignments to the X-sources, that imply a node value of 1. Consequently, the node value is 0 independent of the X-sources. In all other cases, the cube intersection of the input assignment cube and the cubes of the negated function is formed. In case this intersection is empty, i.e. no cubes overlap, the partial input assignment implies that the negated function of the node evaluates to *false* for any X-source assignment and thus, the value of the node equals 1. Otherwise, there are X-source assignments such that the node function as well as its negation evaluate to *true*. In this case, the node value depends on the X-sources. Figure 5.3 illustrates the evaluation of a 2-to-1 multiplexer with one X-source at node s for the input assignment $(a, b) = (1, 1)$, in cube representation ab. It shows the function of output q and its negation, as well as their cube intersection with the input cube ab in a Karnaugh map [Hacht96].

The explicit computation of the two-level cube representation of the function of a circuit node and its inverse may cause exponential memory requirements. As a result, the application to larger circuits is limited.

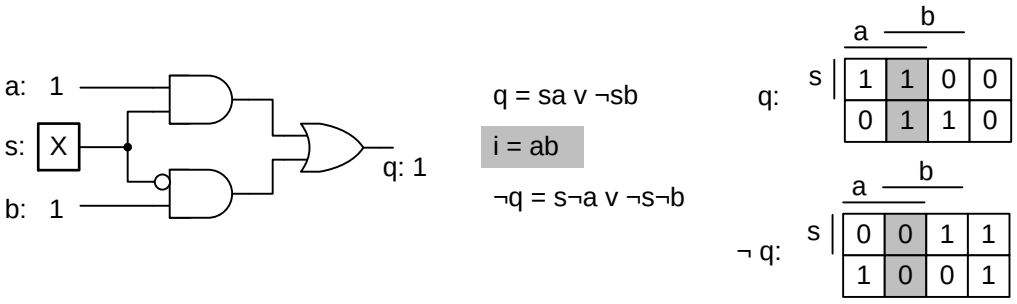


Figure 5.3: Cube-based evaluation of a multiplexer with X-source at node s.

5.2.3 Accurate Evaluation using Decision Diagrams

The symbolic representation of a Boolean function as decision diagram in reduced canonical form allows to extract the Boolean difference or dependence of the function on the used variables. Based on the decision diagram, the node evaluation for different input stimuli can be performed by restricting the BDD.

BDD-Based Symbolic Simulation

In [Bryan87], the analysis of the Boolean behavior of circuits at switch (transistor) level based on ROBDDs was introduced. To reflect the transition process of switches, the nodes assume one of the three values 0, 1 or X'. The value X' "denotes an indeterminate voltage between low and high" [Bryan87]. The value of a node is encoded by two variables as shown in Table 5.1. Functions for the static and dynamic circuit response are then defined over these variable tuples and represented by ROBDDs.

Table 5.1: Value encoding of node y for switch level analysis [Bryan87].

y	$y.1$	$y.0$
1	1	0
0	0	1
X'	1	1

The semantics of the X'-value used in [Bryan87] is less constrained than the semantics assumed in Kleene's strong three-valued logic where the X-value represents a valid binary value, but no intermediate state (cf. Sections 2.1 and 3.2.1). Hence, the symbolic method of [Bryan87] cannot be applied for the exact Boolean simulation targeted in this work.

Still, it is possible to exactly analyze a circuit with X-sources employing its ROBDD representation. Without loss of generality, we consider the X-sources in the circuit as circuit inputs with unknown values. Then, for each input with a known value assignment, the ROBDD of the circuit is restricted [Bryan86] w.r.t. that value. Restricting the BDD variables has a runtime complexity of $\mathcal{O}(n \cdot \log(n))$ in the size of the BDD [Bryan86].

After the restriction step, the X-dependence of a node is easily determined: Due to canonicity of ROBDDs, a comparison of the ROBDD of a node with the tautology function *true* or its inverse *false* reveals whether the node carries a binary value of 0 or 1, or not. The actual comparison with the constant functions, i.e. the two terminal nodes of the ROBDD, is a constant time operation. If the ROBDD of a node is not a constant function, the node value depends on at least one X-source.

This method suffers from the known disadvantages of ROBDD synthesis, i.e. a representation exponential in size w.r.t. the number of input variables for certain arithmetic functions such as multipliers, as well as a strong dependence of the ROBDD size on the variable ordering.

By combining the ROBDD-based analysis with three-valued logic simulation, the number of circuit nodes to be represented by the ROBDD can be significantly reduced. Still, there are circuits such as multipliers, for which this technique cannot prevent the exponential requirement of memory. In Section 5.3.4, a novel approximate ROBDD-based analysis method is presented which trades off accuracy and required computational resources. This proves to be very robust and efficiently applicable to multipliers and large circuits as well.

Accurate Evaluation using Kleene Ternary Decision Diagrams

Kleene ternary decision diagrams (KTDDs, cf. Section 4.1.4) can be used for the representation and manipulation of partially specified functions, as well as for the accurate evaluation in presence of X-sources or partial input assignments [Jenni95b, Jenni95a]. Since KTDDs embed the ROBDD of the represented function, their minimal size is bound by the size of the corresponding ROBDD. Consequently, KTDDs are sensitive to the variable order, and for certain arithmetic functions their size may grow exponentially [Iguch97]. Experimental results show that the KTDD-based representation results in a significantly higher node count and memory requirement than BDDs

[Iguch97]. If a KTDD of a function can be constructed, the accurate evaluation of an input assignment has linear time complexity in the number of variables.

5.2.4 Hardware-Accelerated Evaluation

To accelerate the accurate evaluation of a function in presence of X-values, a hardware-based evaluation based on a mapping to field programmable gate arrays (FPGAs) was proposed in [Iguch99, Iguch04]. The mapped circuit is derived from the BDD of the function and the application of the alignment operation, similar to the construction of KTDDs (cf. Section 4.1.4).

As in the KTDD-based representation, the runtime for the evaluation of an input assignment is linear in the number of variables. In the FPGA-based evaluation, the throughput is increased by pipelining of pattern evaluations. A principal problem lies again in the construction of the decision diagram of the circuit with a possibly exponential number of vertices, limiting the application of the method to small circuits.

5.2.5 Accurate X-Analysis Using Boolean Satisfiability

The computation of the value of a single signal node can also be mapped to an instance of the Boolean satisfiability (SAT) problem. The underlying idea is to search for two assignments to the X-sources for which the considered node takes complementary logic values, or to prove that such assignments do not exist (cf. Formula (5.1)). In the first case, the considered node carries an X-value. In the latter case, it has a constant value invariant of the state of the X-sources.

The following sections explain the basic method of a SAT-based accurate node evaluation recently presented in [Elm10, Kocht12]. Accurate SAT-based simulation proves to be a robust method that does not suffer from the potentially exponential memory requirements of BDD-based accurate simulation. Section 5.4 discusses the integration of techniques to reduce the number of required SAT-based evaluations without compromising the accuracy of the result.

Algorithm Overview

The SAT-based accurate evaluation identifies the set of circuit nodes with an actual X-value for a given input pattern. The method is sound and complete, i.e. firstly, the value of any identified X-valued node depends on at least one X-source, and secondly, all X-valued nodes are identified. The values of the remaining nodes do not depend on the X-sources for the considered pattern.

The evaluation of all nodes in the circuit may require a high computational effort. The evaluation can be restricted to those nodes which carry a pessimistic X-value, as determined for instance by three-valued logic simulation of the input assignment. This step effectively reduces the nodes subject to the SAT-based evaluation to those reachable from the X-sources along sensitized paths. Since X-canceling at a circuit node requires the reconvergence of nodes with *correlated* X-values, the set of nodes to be evaluated by the SAT-mapping is further reduced by a fast topological reconvergence analysis.

Figure 5.4 depicts the flow of the algorithm. The reconvergence analysis is performed for the given input pattern p and the set of nodes with pessimistic X-value (NX^P). Then, a SAT instance is generated that is used to analyze the reconvergences in NX^P. The result of this computation is the set of X-valued nodes for the input assignment. For nodes whose value does not depend on the X-sources, the binary value is computed.

Reduction of SAT-Based Evaluations by Three-Valued Logic Simulation and Reconvergence Analysis

A three-valued logic simulation of the input pattern p is performed to identify an upper bound of nodes with unknown value and avoid the SAT-based evaluation of all nodes in the circuit. At the X-sources, X^P-values are imposed. The result of this simulation are the binary values of a subset of the circuit nodes V, as well as the set of nodes with pessimistic X-value (NX^P). Only the nodes in NX^P need to be considered in the SAT-based evaluation.

The signals in NX^P span a sub-graph in the circuit graph. This sub-graph has the X-sources as roots. The leaves are the circuit outputs with pessimistic X-value, and circuit nodes with pessimistic X-value that have a controlling value at at least one input, as illustrated in Figure 5.5.

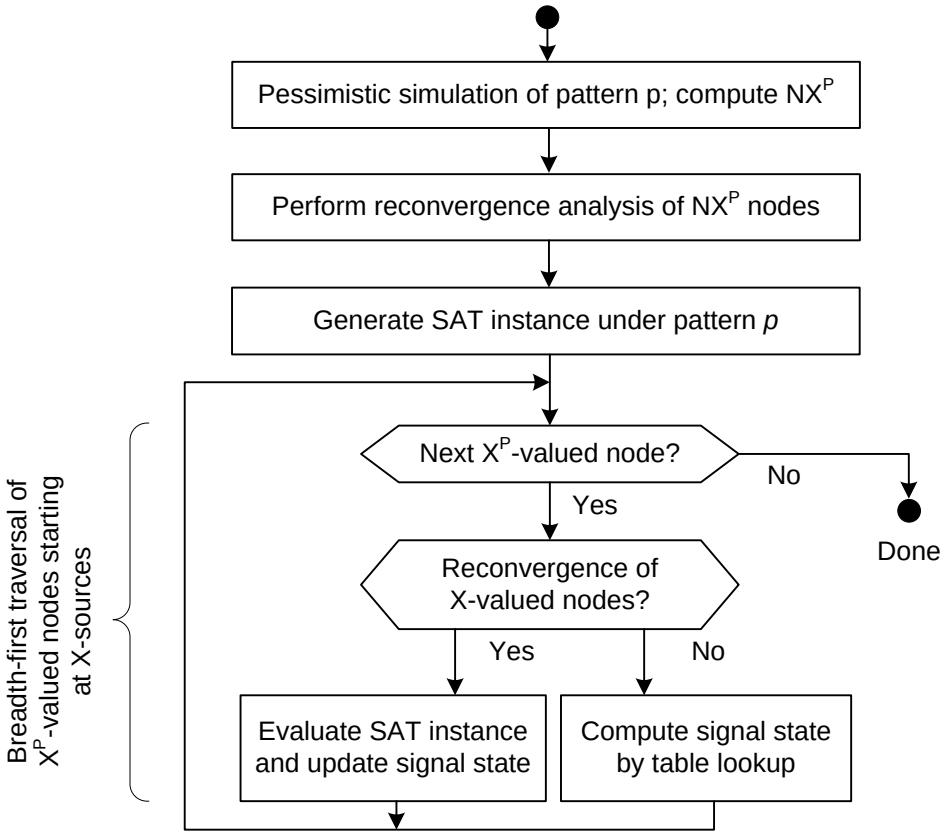


Figure 5.4: Accurate SAT-based evaluation of an input assignment p.

X-canceling can only occur at reconvergences in this graph. The search for reconvergent nodes is implemented as an event-based forward traversal of the nodes in NX^P starting from the X-sources. For each node $v \in \text{NX}^P$ the X-sources in the transitive fanin of v are stored. The input values of gates, obtained from the preceding logic simulation step, are used to determine the sensitized paths along which X^P-values propagate. The nodes at which paths originating at the X-sources reconverge are marked and evaluated in the accurate analysis as explained below.

Generation of the SAT Instance to Prove X-Dependence

The SAT instance to decide about the X-source dependence of a reconvergence v is constructed according to Formula (5.1): Two assignments $a_1, a_2 \in \mathcal{A}_X$ to the X-sources are searched such that the resulting values of v under a_1 and a_2 differ. If such assignments do not exist, then node v has a constant value of either 0 or 1. The search for the

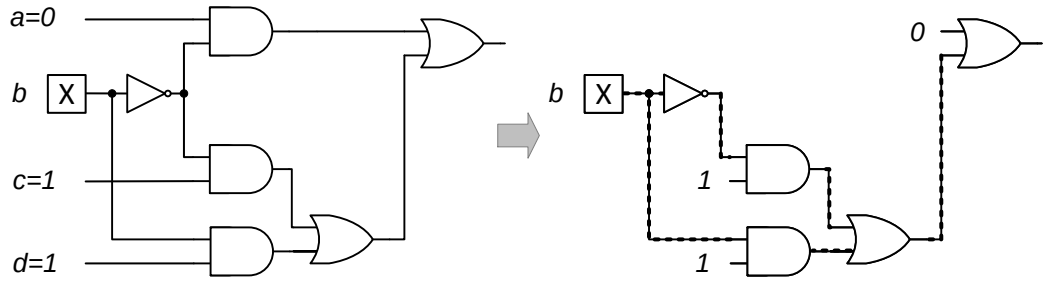


Figure 5.5: Circuit sub-graph showing the reconvergences of nodes with X^P-value.

assignments a_1, a_2 can be conducted serially by computing the satisfiability of v and then $\neg v$, or in one step by computing the satisfiability of an extended instance that considers both cases simultaneously [Elm10]. In the following, the serial approach is discussed.

The SAT instance only needs to consider those nodes in the transitive fanin of node v that belong to NX^P, i.e. nodes whose value potentially depends on the X-sources (Figure 5.6). For all other nodes, a binary value has already been computed by three-valued simulation.

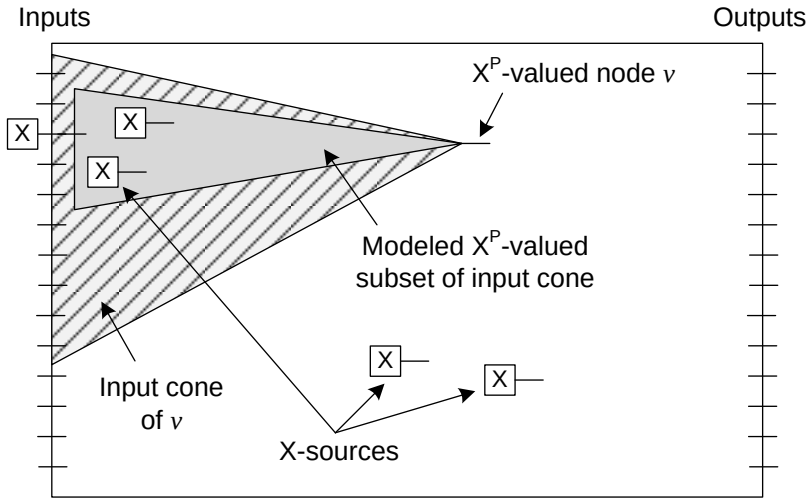


Figure 5.6: Modeled input cone of X^P-valued node v in the SAT instance

The SAT instance P_{v,NX^P} models node v and all X^P-valued nodes in its fanin cone for the

pattern under analysis. The instance is obtained by using the Tseitin transformation [Tseit68]. The only free variables in P_{v,NX^P} are those representing the X-sources in the circuit. If $P_{v,\mathrm{NX}^P} \wedge v$ is not satisfiable, then v carries the value 0. Otherwise, it is computed whether $P_{v,\mathrm{NX}^P} \wedge \neg v$ can be satisfied. If it is satisfiable, then two assignments exist that drive v to complementary values, and $\mathrm{val}(v, p) = \mathrm{X}$. Otherwise, v carries value 1 for the pattern under analysis.

To avoid the overhead of generating separate SAT instances for every node evaluation, a single SAT instance P_p can be generated per input pattern p which computes the accurate value of a node in one step [Elm10]. This exploits the potentially large overlapping of X-valued input cones of the nodes under evaluation. The runtime of the simulation increases with the circuit size and the number of X-sources.

5.3 Approximate Logic Simulation Algorithms

The following section discusses approximate algorithms for logic simulation in presence of X-values. These algorithms have been proposed to increase simulation accuracy over three-valued simulation at low or moderate computational cost. In general, the algorithms cannot compute the exact result or prove signal dependence on X-sources.

The algorithms are classified as follows: Section 5.3.1 introduces algorithms which perform a limited analysis of the possible values of the X-sources. Section 5.3.2 shows the application of implication and learning techniques to increase the accuracy. Sections 5.3.3 and 5.3.4 introduce restricted symbolic approaches which increase the number of symbols in the simulation step by indexed symbols, or by a representation based on decision diagrams.

5.3.1 Limited X-State Expansion and Analysis

The exhaustive simulation of all value assignments to the X-sources in a circuit (Section 5.2.1) is only feasible for small circuits and few X-sources. By considering only a subset of all possible assignments, the effort for analysis is reduced at the cost of the achieved accuracy.

In [Pomer97], the explicit enumeration and simulation of partially defined states in partial scan circuits is conducted. Uncontrolled (non-scan) sequential elements are considered as X-sources and generate X-values in the circuit. To limit the computational complexity of the state expansion, only a fixed number of states, respectively assignments to X-sources, are considered: Multiple subsets of the X-sources $S_X^i \subseteq S_X$ are selected. For each set S_X^i, all $2^{|S_X^i|}$ possible binary values to the included X-sources are assigned, while the remaining X-sources keep the X-value.

A three-valued simulation of these expanded states or assignments is conducted. A node which carries identical binary values for all simulated assignments of one set S_X^i does not depend on the X-sources, even if only a subset of all possible states has been simulated. Nodes with different binary values definitely depend on the X-sources. For the remaining nodes with an X^P-value, or nodes with different values from the set $\{0, 1, X^P\}$ for the simulated states, the dependence on the X-sources is neither proven nor refuted.

A similar principle is applied in [Kang03] where a standard three-valued logic simulation algorithm is extended by exhaustive simulation of small reconvergent regions in the circuit to gain information about X-propagation in this region. This information is then fed back into the three-valued simulation. The circuit partitions for exhaustive analysis are determined after three-valued simulation of the circuit with the pattern under simulation. The method searches for reconvergent regions in which both the fanout stem and the reconvergent node have an X^P-value according to three-valued simulation. The identified regions are simulated with both binary values 0 and 1 at the fanout stem. If the node value at the reconvergence has identical binary values for all X-assignments, this value is used as the accurate output value in the complete circuit simulation. Otherwise, the node actually carries an X-value. If X^P-valued regions overlap, the union of them is formed. For k X-sources in the union, 2^k value assignments for X-valued stems need to be evaluated by simulation to obtain the accurate result. In [Kang03], the size of the considered regions is limited, which leads to pessimistic results.

5.3.2 Implication and Learning Based Algorithms

Structural circuit analysis techniques, termed static [Schul88] or recursive learning [Kunz94], can be applied to increase simulation accuracy in presence of X-values.

These techniques have been initially proposed to speed-up automatic test pattern generation. [Kajih04] shows that the knowledge gained from static circuit analysis allows to prove that certain nodes with pessimistic X-value after three-valued simulation do not depend on X-sources and have defined logic values.

Structural learning algorithms search in the circuit for local combinational invariants, called indirect implications, by logic simulation and by application of the contrapositive law [Schul88]: $(P \rightarrow Q) \leftrightarrow (\neg Q \rightarrow \neg P)$. In static learning, fanout stems in the circuit are subject to two logic simulations with the values 0 and 1. In the simulation result, implications such as $P \rightarrow Q$ are searched, where P, Q are propositions about node values. If the contrapositive of this implication, $\neg Q \rightarrow \neg P$, satisfies certain learning criteria [Schul88, Kajih04], it is added as additional implication to the circuit model. Figure 5.7 shows an example of such an indirect implication from input a to output o, which increases the accuracy of three-valued simulation (in blue). A value of 1 at a implies that o takes value 1 independent of the value of b.

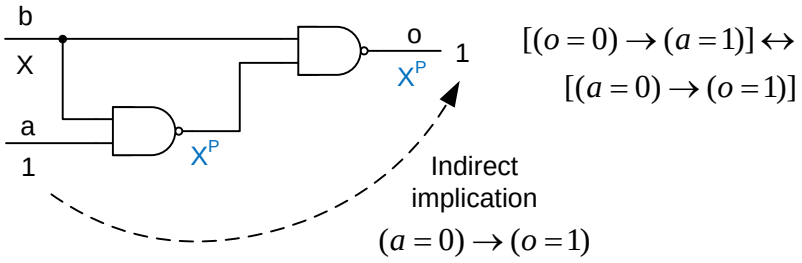


Figure 5.7: Indirect implication found by static learning.

Simulation based on static learning is incomplete and computes only a pessimistic result. In addition, the search for indirect implications requires many circuit simulations, and the number of found implications may be very high. Even with sophisticated learning criteria, the size of the resulting circuit graph with added implications increases significantly.

Recursive learning repeats the learning step until no additional implications are found. This ensures that all implications between signal nodes are discovered and added to the circuit model. In practice, the number of iterations must be limited to prevent an excessive growth of the model.

The experimental evaluation in Section 5.6 shows the high degree of pessimism that still remains after applying static learning techniques in logic simulation.

5.3.3 Restricted Symbolic Simulation

By extending the set of symbols in the algebra, the accuracy of the node evaluation can be increased. A shortcoming of the three-valued logic, as shown in Figure 5.8 a), is the lack of an inverse ¬X of the X-symbol [Breue72] which prevents to compute the reconvergence at line d accurately. A logic with an inverse of the X-value allows the accurate evaluation of reconvergences if X-values from different X-sources never converge, as is the case in Figure 5.8 b). However, the use of such a four-valued logic may produce optimistic results if nodes with uncorrelated X-values converge.

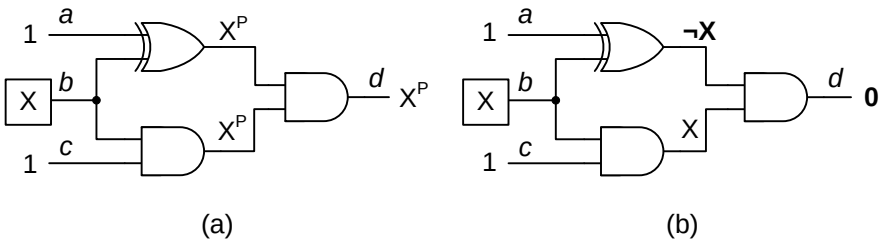


Figure 5.8: Evaluation in (a) three-valued logic and (b) 4-valued logic $\{0, 1, \mathrm{X}, \neg\mathrm{X}\}$.

The introduction of an inverse for the X-symbol is generalized in restricted symbolic or indexed simulation [Ogiha88, Carte89]: Instead of a single X-symbol and its inverse, multiple X-symbols for different, uncorrelated X-values and the corresponding inverses are added to the symbol set:

Two symbols are used to encode the binary states $\{0, 1\}$. Each X-source is assigned a unique symbol, typically encoded as positive integer.[1] Inverted X-values are assigned the negated integer value of the original X-value. This encoding ensures that an X-value does not lose its correlation when propagated through an inverting gate. If two or more X-values from different X-sources converge at a gate and X-canceling does not occur, then the output node is assigned a new symbol which has not been used in the simulation so far. This symbol is then used for the simulation of the fanout of that node. In this case, the correlation between the output X-value and the input X-values is lost. For a circuit with V nodes, the symbol domain is extended by at most $2 \cdot |V|$ symbols during simulation of an input pattern.

Table 5.2 shows the truth table of a two-input AND gate for restricted symbolic simulation. If an X-value X_r and its inversion $\neg\mathrm{X}_r$ reconverge at a gate such that X-canceling

[1] By using a 32-bit machine word per symbol, approximately 2.1 billion different symbols are available.

results, a binary value is computed (marked bold in the Table 5.2). If uncorrelated X-values X_r, X_s are combined, a new unique X-symbol is generated (denoted as $X^\star$) since the result can neither be reduced to a binary value nor expressed symbolically with the restricted symbol set.

Table 5.2: Two-input AND gate truth table for restricted symbolic simulation (adopted from [Kundu91]).

$\wedge$	0	1	X_r	$\neg X_r$	X_s	$\neg X_s$
0	0	0	0	0	0	0
1	0	1	X_r	$\neg X_r$	X_s	$\neg X_s$
X_r	0	X_r	X_r	**0**	$X^\star$	$X^\star$
$\neg X_r$	0	$\neg X_r$	**0**	$\neg X_r$	$X^\star$	$X^\star$
X_s	0	X_s	$X^\star$	$X^\star$	X_s	**0**
$\neg X_s$	0	$\neg X_s$	$X^\star$	$X^\star$	**0**	$\neg X_s$

The node evaluation in restricted symbolic simulation proceeds similar to conventional three-valued logic simulation: Nodes are either evaluated in topological order, or event-based. The evaluation of a single node is performed by a procedural computation instead of a table lookup [Carte89] to determine whether correlated X-values reconverge. The runtime complexity is linear in the circuit size.

Using restricted symbolic simulation, a large fraction of local X-reconvergences are evaluated accurately. X-reconvergences are evaluated pessimistically if X-canceling occurs after combination of different X-values. Figure 5.9 shows a circuit with X-sources at nodes b and c for which restricted symbolic simulation computes a pessimistic result under the input assignment $a = 1$. The value of line f is computed as X^P although it always carries value 0. This is due to the convergence of the two uncorrelated X-values at line e and the resulting introduction of a new X-symbol X_3 that does not encode the correlation with X_1 and X_2 any more. This pessimism increases with the number of X-sources in the circuit: The more X-sources, the higher is the probability of combinations of X-valued nodes and the introduction of uncorrelated new X-symbols.

The accuracy of the result also depends on the netlist structure and node evaluation order since restricted symbolic simulation is not associative [Kundu91]. As an example, Figure 5.10 shows three circuits that implement the conjunction of three input nodes. The evaluation order imposed by the circuit structure results in output values with different accuracy.

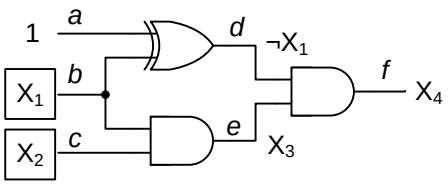


Figure 5.9: Pessimistic node evaluation in restricted symbolic simulation.

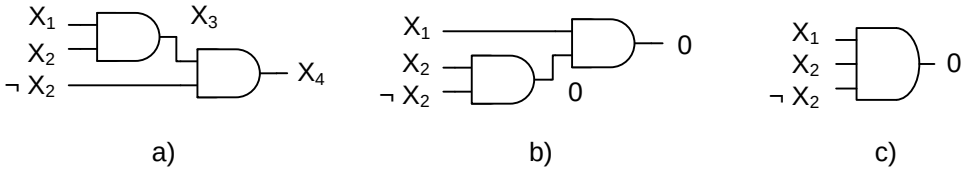


Figure 5.10: Impact of node evaluation order on the result in restricted symbolic simulation: (a) pessimistic; (b), (c) accurate.

Restricted symbolic simulation is used in logic and fault simulation [Carte89], test generation [Ogiha88, Kundu91], and diagnosis [Wen05].

5.3.4 Restricted Symbolic Simulation by Local BDD-based Analysis

The remaining pessimism in restricted symbolic simulation can for instance be overcome by selective application of accurate symbolic node evaluation. However, the memory required to symbolically represent the function of the nodes may still be prohibitively high. This is the case for multipliers where restricted symbolic simulation is not able to reduce the number of pessimistic X-values significantly. In consequence, large parts of a multiplier must be mapped to its symbolic representation which entails high memory requirements. This motivates a *generalized and configurable restricted* symbolic representation of nodes based on ROBDDs as recently published in [Kocht11a].

Assuming that X-canceling occurs locally, the proposed method in [Kocht11a] builds restricted, local ROBDDs which are limited in the size, i.e. in the number of nodes. This yields a scalable approach with configurable accuracy based on a restricted symbolic representation. The method extends ideas where symbolic representations of circuit partitions of limited size are used to bound excessive memory requirements during

equivalence checking and verification at the cost of a potentially pessimistic result [Kuehl97, Wilso00].

The circuit partitions that are represented symbolically with ROBDDs are determined by a fast pessimistic simulation algorithm, for instance by three-valued logic simulation or restricted symbolic simulation (cf. Section 5.3.3). Only for the nodes which carry an X^P-value after this simulation, local ROBDDs are constructed. The reduction of the number of signal nodes for which a BDD representation and analysis is required significantly improves the efficiency and scalability of the overall analysis.

The BDD-based hybrid logic simulation flow is depicted in Figure 5.11 and outlined in Algorithm 5.2: For an input pattern, the set of signals NX^P with a pessimistic X-value is computed using a fast pessimistic simulation algorithm. As a by-product, the binary values of all other signals are known. The nodes with pessimistic X-value are then processed in topological order, starting from the X-sources in the circuit.

For each X-source, a BDD variable is introduced and the trivial BDD for this variable is stored. For all other nodes $v \in NX^P$ to be processed, the Boolean function of the input nodes $pred(v)$ have already been computed since the netlist is processed in topological order. If one of the predecessors of v has the controlling value of the function func(v), or all the predecessors have binary values, then the value of v is derived by a fast table lookup. This step avoids the construction of a BDD at a node where binary input values already imply a defined output value. This case can occur if predecessors of the considered node have been proven to have a binary value. If the value of v cannot be derived by such simple reasoning, an ROBDD is constructed for the Boolean function at v. The method build_bdd in Algorithm 5.2 constructs the ROBDD for v depending on the function of v and the symbolic representation of its inputs.

If the resulting ROBDD of node v is a constant function, its value is set to the corresponding binary value 0 or 1. This value is then used for the simulation of the successors of v. Otherwise, the value is kept as pessimistic X-value. If the size of the BDD of the function of v exceeds a given number of nodes, the symbolic representation of v is simplified as follows: The BDD with excessive size is not used any more and deleted. Instead, a new free variable is introduced and used in the construction of the following BDDs in the output cone of v. In this way, the size of the constructed and stored BDDs are bound by the node threshold and the number of X^P-valued nodes $|NX^P|$ computed by the initial pessimistic simulation of the input stimulus.

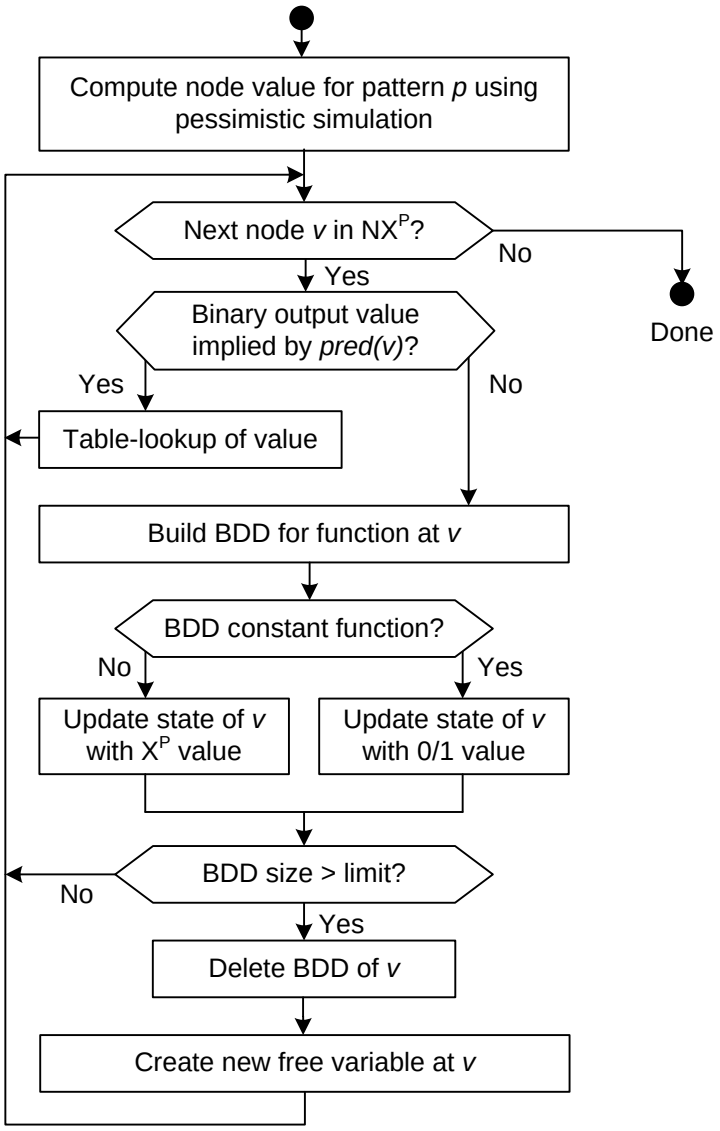


Figure 5.11: Restricted symbolic simulation using local BDDs.

An example of the constructed ROBDDs for a small circuit is given in Figure 5.12. Assume that signal lines b, c, d in the circuit of Figure 5.12 a) are three X-sources, and a, e have the binary value of 1. In Figure 5.12 b), the non-trivial BDDs of the Boolean functions at signals f, g, h, i, k are depicted. bdd_f represents the inversion of signal b and bdd_g represents the conjunction of signals b and c. bdd_h collapses to the constant-0 function. bdd_i^* results from the exclusive disjunction of the functions at d and g. Assuming a node limit of five, this BDD with seven nodes exceeds the limit and is not used in the simulation. It is deleted and a new variable bdd_i is introduced. bdd_k finally reflects the inversion of bdd_i resulting from the negated conjunction at k.

Algorithm 5.2 Restricted symbolic simulation using local BDDs.

```
 1: procedure SIMULATECIRCUITRSSLOCALBDDS(p, V)
 2:     Perform pessimistic logic simulation of p
 3:     NX^P ← {v ∈ V | val(v, p) = X^P}
 4:     for v ∈ NX^P in topological order, starting from X-sources do
 5:         if (∃ n' ∈ pred(v) : val_n' = controlling_value(func(v))) ∨
 6:            (∀ n' ∈ pred(v) : val_n' ∈ {0, 1}) then
 7:             val_v := table_lookup(func(v))
 8:         else
 9:             if v is an X-source) then
10:                 bdd_v := create_bdd_variable(v)
11:             else
12:                 bdd_v := build_bdd(func(v))
13:                 if (bdd_v = 0) then
14:                     val_v := 0
15:                 else if (bdd_v = 1) then
16:                     val_v := 1
17:                 else if (size(bdd_v) > limit) then
18:                     delete_bdd(bdd_v)
19:                     bdd_v := create_bdd_variable(v)
20:                 end if
21:             end if
22:         end if
23:     end for
24: end procedure
```

Using local BDDs with limited size, excessive memory requirements are avoided at a certain loss of simulation accuracy. X-reconvergences may be evaluated pessimistically if the information about a correlation between signal states in the circuit is lost due to the introduction of new variables once a BDD exceeds the node limit. In the evaluation in Section 5.6 this pessimism is investigated and quantified.

5.4 Hybrid SAT-based Algorithm for Accurate X-Analysis

Accurate analysis methods as discussed in Section 5.2 can be combined with approximate methods and heuristics to limit the use of formal reasoning as much as possible. This section discusses the tight integration of such methods into an efficient and accurate SAT-based simulation algorithm [Hille12] which significantly reduces runtime

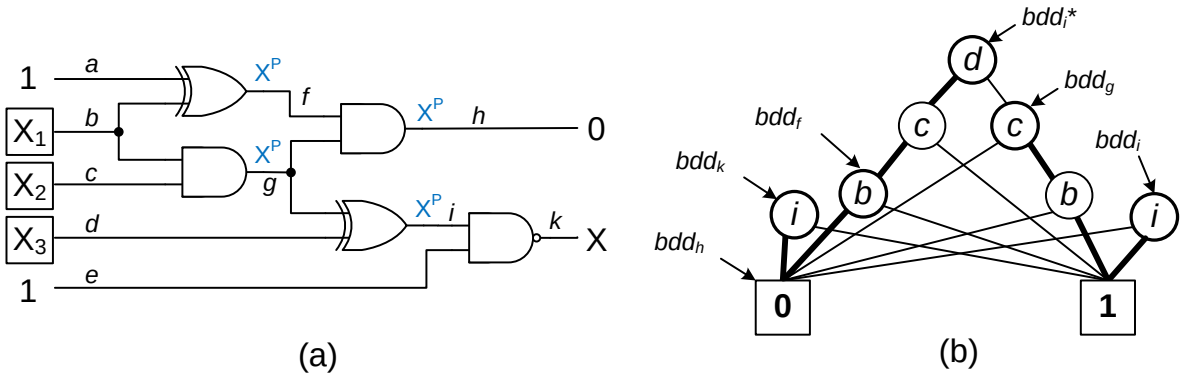


Figure 5.12: BDD construction in the proposed BDD-based simulation.

compared to the plain algorithm of [Elm10, Kocht12].

Figure 5.13 depicts an overview of the proposed algorithm. In the simulation step of an input pattern p, p is simulated using restricted symbolic simulation (RSS) [Carte89] to compute a superset of the X-valued signals. In addition, a limited state expansion and analysis (cf. Section 5.3.1) is conducted to determine as many X-valued signals as possible, and to identify candidate signals for which the dependence on X-sources is yet undetermined. This is implemented by a simulation of randomized assignments to the X-sources. The candidates for pattern p are then exactly classified by proving the X-source dependence using an incremental SAT solver.

5.4.1 Reduction of Nodes with Pessimistic X-Value

The runtime of an accurate analysis increases with the number of nodes to be evaluated. Thus, the number of nodes to be analyzed by formal reasoning must be reduced as much as possible to increase simulation efficiency.

Restricted symbolic simulation, introduced in Section 5.3.3, is very effective in the identification of a subset of nodes with binary value at low computational cost. In the proposed algorithm, the value of node v resulting from restricted symbolic simulation of pattern p is stored in $\texttt{val}(p, v)$.

However, restricted symbolic simulation cannot prove the dependence of nodes on the X-sources. The state-expansion method discussed in Section 5.3.1 is used here as heuristic to efficiently prove the X-source dependency of nodes: A limited number of

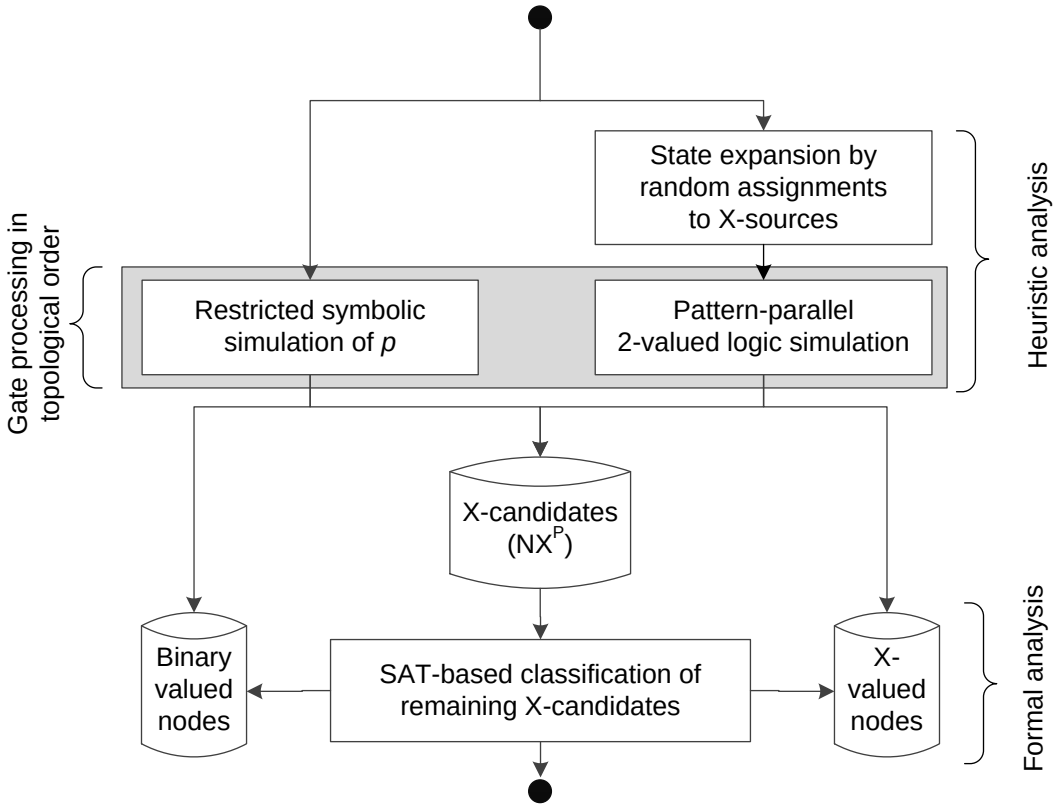


Figure 5.13: X-Analysis using restricted symbolic simulation and limited state expansion by simulation of randomized X-source assignments.

randomized assignments to X-sources is simulated and analyzed. If the node values for at least two assignments differ, the node depends on the X-sources.

For this simulation, a two-valued pattern parallel simulator is used. 64 input stimuli are generated by assigning 64 randomized values to the X-sources for each input pattern under simulation. The 64 stimuli are evaluated in one single simulation. A 64-bit integer $z = [z^0, \ldots, z^{63}]$ is used to represent the values of each node. For input i, z_i is derived from the simulated pattern p and set to $[0, \ldots, 0]$ or $[1, \ldots, 1]$ if i is 0 or 1, respectively. At an X-source q, a randomized 64-bit integer is generated and assigned to $z_q = [z_q^0, \ldots, z_q^{63}], z_q^i \in \{0, 1\}, 0 \leq i \leq 63$. z_q is used for the evaluation of the direct fanout of q.

As shown in Figure 5.14, each node is classified after these two simulations as either (i) having an X-value, (ii) having a binary value, or (iii) as X-*candidate* with a pessimistic X-value. For X-candidates, the dependence of the node value on X-sources needs still to be analyzed using SAT-based reasoning. If restricted symbolic simulation computed

a binary value, the signal is not considered in the subsequent steps. If a pessimistic X-value is calculated for node v, the value $z_v = [z_v^0, \ldots, z_v^{63}]$ of the pattern parallel simulation is taken into account. If at least one pair of values z_v^i, z_v^j, $(0 \leq i, j \leq 63)$ has complementary values, node v carries an X-value. If all z_v^i are equal, the considered 64 randomized assignments do not prove a dependence on the X-sources, and v is marked as X-candidate. The classification of these signals is then conducted with an incremental SAT-solver as explained in the next section.

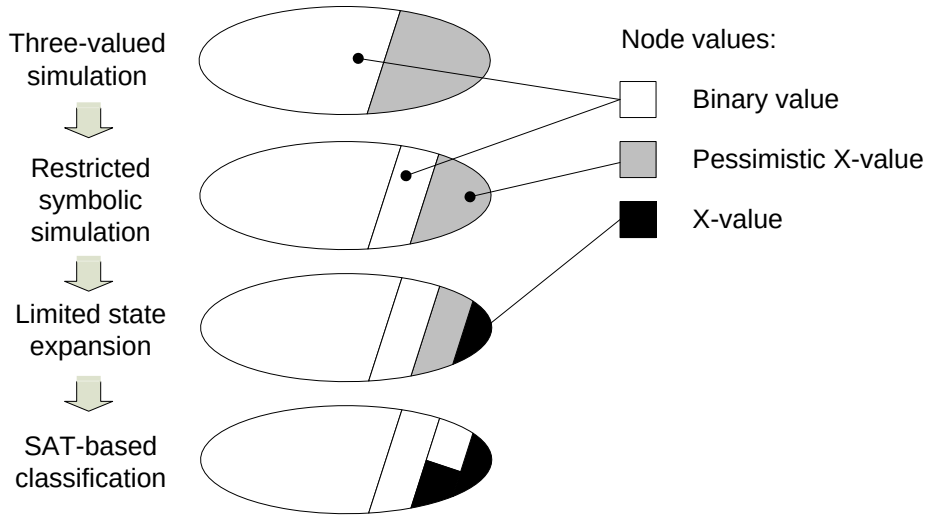


Figure 5.14: Classification of node values after three-valued simulation and subsequent refinement by restricted symbolic simulation, limited state expansion, and accurate SAT-based analysis.

5.4.2 Integration into Incremental SAT-Solving

For the formal proof whether a signal node classified as X-candidate depends on the X-sources or not, a SAT instance is incrementally constructed and processed by the state-of-the-art incremental SAT-solver Antom [Schub10].

For a node v classified as X-candidate it is already known that all 64 random assignments to the X-sources force v to value z_v^i $(0 \leq i \leq 63)$ of either 0 or 1. Node v has a binary value if and only if it can be proven that there is no assignment to the X-sources which implies the complemented value $\neg z_v^i$ at v. Thus, a SAT instance is constructed that is satisfiable if and only if v can be driven to $\neg z_v^i$. If the instance is satisfiable, v

depends on the X-sources and has an X-value. Otherwise, v is independent of the X-sources and is assigned its binary value. This SAT instance is much smaller and faster to analyze than the instance used in Section 5.2.5 since it only searches for a single assignment of to the X-sources to satisfy $v = \neg z_v^i$.

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

To check whether v can be driven to $\neg z_v^i$, the characteristic equations of the gates in the transitive fanin of v are translated into CNF and added to the SAT instance using the Tseitin transformation [Tseit68]. The size of the resulting SAT instance is kept small by only considering the nodes in the fanin of v that carry an X-value under pattern p. The CNF for the transitive fanin of node v is created recursively as shown in Algorithm 5.3, and extended iteratively during the analysis of pattern p.

Algorithm 5.3 CNF creation of the adjustment cone for node v.

```
 1: procedure ADDSIGNALTOCNF(v, CNF)
 2:     if v already Tseitin transformed then
 3:         return;
 4:     end if
 5:     if val(p, v) = 0 then
 6:         CNF := CNF ∪{¬v};
 7:         return;
 8:     end if
 9:     if val(p, v) = 1 then
10:         CNF := CNF ∪{v};
11:         return;
12:     end if
13:     CNF := CNF ∪ GETTSEITINTRANSFORMATION(v);
14:     for all u ∈ pred(v) do
15:         ADDSIGNALTOCNF(u, CNF);
16:     end for
17: end procedure
```

This SAT instance is extended by a temporary unit clause with only one literal (called assumption) for the X-candidate v to constrain the value of the variable for v in the search process of the SAT solver. If the value of v in the pattern parallel simulation was $z_v = [0, \ldots, 0]$, the assumption $\{v\}$ is added to constrain the SAT search to assignments

to the X-sources which imply v to 1. If the instance is satisfiable, v is proven to have an X-value. Otherwise, v does not depend on the X-sources and has value 0. In the latter case, $\texttt{val}(p, v)$ is updated accordingly, and the unit clause $\{\neg v\}$ is added permanently to the SAT instance to reduce runtime for subsequent invocations of the SAT solver. Correspondingly, if the value of v in the pattern parallel simulation was $z_v = [1, \ldots, 1]$, the assumption $\{\neg v\}$ is added to search for an assignment that implies value 0 at node v and proves its X-dependence.

For the classification of the next X-candidate v' in topological order, the CNF instance is extended incrementally to include the transitive fanin of v'. At this point, only the clauses for functions of those nodes are added that are not yet included in the CNF instance (cf. line 2 in Algorithm 5.3).

The algorithm maintains a lookup table which stores for each indexed X-symbol used during the restricted symbolic simulation the current value of the set $\{0, 1, \mathrm{X}^P, \mathrm{X}\}$. Initially, the table is derived from the result of the restricted symbolic simulation. Before analyzing an X-candidate v using the SAT solver, a fast lookup is performed to check whether the indexed X-symbol corresponding to v has already been computed. If the classification for this symbol is already known, the accurate value of v is directly updated. Otherwise, v is classified as described above. This effectively restricts the use of the SAT solver to nodes at which different X-valued nodes converge.

5.5 Sequential Logic Simulation in Presence of Unknown Values

The simulation algorithms discussed in the previous sections can also be used in sequential logic simulation to increase accuracy in presence of X-sources. Pessimistic algorithms may cause a significant increase of X-values in each cycle of the simulation, up to the point where the state of most nodes in the circuit is unknown.

This section first outlines a basic cycle-based simulation algorithm for synchronous sequential circuits based on the three-valued logic algebra. Then, an efficient approximate and a novel accurate simulation algorithm [Erb14a] are presented, which are able to reduce or completely eliminate the simulation pessimism.

5.5.1 Pessimistic Sequential Simulation

Sequential logic simulation of a synchronous circuit can be mapped to cycle-based simulation in which the combinational part of the circuit is evaluated once per cycle and the resulting state information is forwarded to the simulation of the next cycle. In this way, an arbitrary number of cycles can be simulated with low memory requirements as illustrated in Figure 5.15. Algorithm 5.4 shows the principle of synchronous sequential simulation.

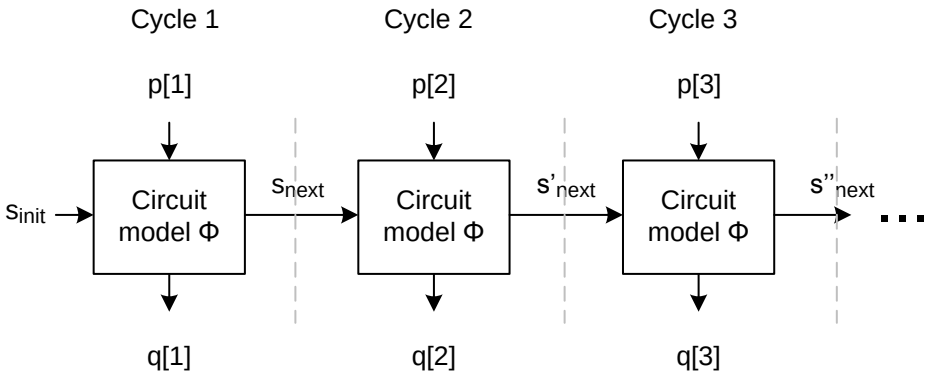


Figure 5.15: Sequential simulation of multiple cycles with initial state s_{init} and input patterns $\vec{p}$.

If the evaluation of the combinational part is based on the three-valued logic, the result of a single cycle is inevitably pessimistic and overestimates the number of nodes with X-value. The simulation pessimism further increases in subsequent cycles. In the worst case, all nodes are assigned a pessimistic X-value in the simulation result, and no useful information is obtained.

Algorithm 5.4 Pessimistic synchronous logic simulation based on three-valued logic.

```
1: procedure SIMULATECYCLES(p⃗, s_init, n)
2:     s ← s_init
3:     for cycle := 1 to n do
4:         s_next ← SIMULATECIRCUIT3VLOGIC(p[cycle], s)        ▷ cf. Section 5.1.3
5:         s ← s_next
6:     end for
7: end procedure
```

Sequential simulation with increased accuracy is required in the verification of synchronizing or reset sequences. A synchronizing (reset) sequence is an input sequence

that drives a sequential circuit starting from a partially or completely unknown state into a known state [Henni64]. Some sequences cannot be verified using pessimistic three-valued sequential simulation [Keim96], but require more accurate simulation techniques.

5.5.2 Approximate Cycle-Based Sequential Simulation

The accumulation of X-values in pessimistic sequential simulation can be confined by reducing the pessimism of the evaluation of each simulated cycle. For instance, the restricted symbolic simulation, the local BDD-based evaluation of Section 5.3, or the accurate SAT-based approach of Section 5.4 can be used to reduce the pessimism during the evaluation of a cycle. For this purpose, the three-valued simulation step in Algorithm 5.4 (line 4) is replaced by one of the more accurate simulation algorithms.

Even if the accurate method of Section 5.4 is applied, the result may still be pessimistic if information about the correlation of X-values at the boundary between two simulation cycles is lost.

5.5.3 Accurate Sequential Simulation

In a separate evaluation of each cycle, inaccuracies are introduced during the propagation of state information between cycles. At the cycle boundary, the correlation of X-valued nodes is lost unless all information is forwarded by a symbolic expression, or the evaluation of node reconvergences v takes into account the transitive fanin of v over all preceding cycles.

Figure 5.16 shows an example where a cycle-based simulation yields a pessimistic result although each cycle is evaluated accurately. The signals a' and b' depend on the X-sources a and b. The two signals are correlated, yet this information is not conveyed as input to the simulation of the second cycle. The fanout branches at node a reconverge in the second cycle and consequently, the simulation result is pessimistic.

At increased computational effort, this problem is resolved by storing and propagating the exact correlations between X-valued nodes in all simulated cycles. If the analysis is conducted using unrestricted BDD-based symbolic simulation, the BDD representa-

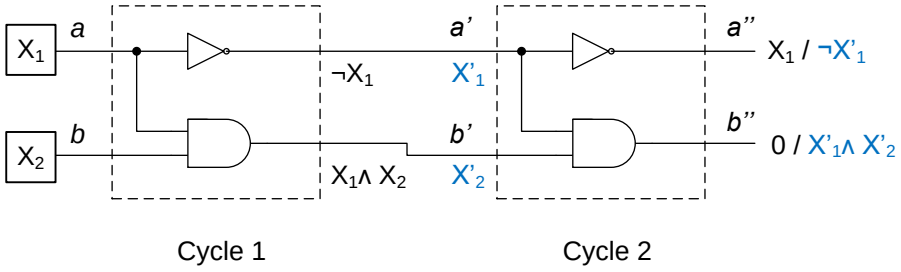


Figure 5.16: Sequential simulation of two cycles with (black) and without (blue) forwarding node correlations at the cycle boundary.

tions of nodes with X-value at the cycle boundary are used in the subsequent cycle. As a consequence, the size of the BDD representations may grow prohibitively large.

If the number of cycles to be simulated is fixed, *time frame expansion* can be applied as proposed in [Erb14a]. Time frame expansion maps the sequential circuit model for k cycles to a combinational circuit model by instantiating the combinational part k times, and by connecting the corresponding pseudo-inputs and outputs of the instances, as shown in Figure 5.17.

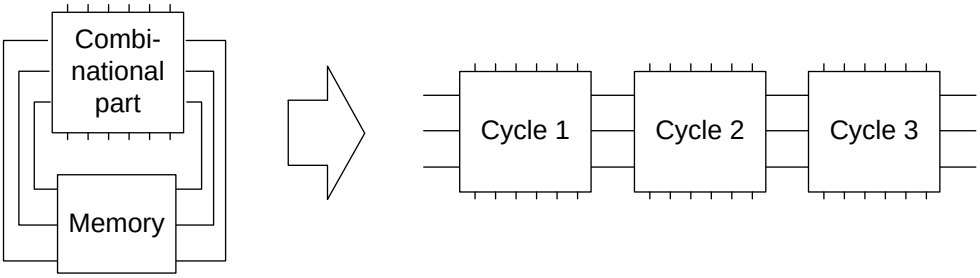


Figure 5.17: Time frame expansion for three cycles.

Correspondingly, the SAT instance required to evaluate whether a node can be driven to complementary values grows with the size of the expanded circuit model and becomes more difficult to solve [Erb14a]. In the worst case, the number of the nodes to be represented grows linearly with the number of simulation cycles. In the example in Figure 5.16, the accurate evaluation of output b'' in the second cycle requires to consider all nodes in both cycles. Experiments in Section 5.6.3 show that a limited number of cycles can be still evaluated accurately with the SAT-based approach even for larger circuits.

5.6 Evaluation on Benchmark and Industrial Circuits

Based on benchmark circuits and industrial circuits provided by NXP, the three-valued simulation (Section 5.1.3), the indirect implication based simulation (Section 5.3.2), the restricted symbolic simulation (Section 5.3.3), the approximate local BDD-based algorithm (Section 5.3.4), and the accurate SAT-based algorithm (Section 5.4) are evaluated and compared. Details on the used circuits are provided in Appendix B.2.

The following sections explain the experimental setup and metrics, followed by the results of combinational and sequential logic simulation.

5.6.1 Setup and Applied Metrics

In the experiments we assume without loss of generality that a subset of primary and pseudo-primary circuit inputs are X-sources and generate X-values. The *X-ratio* denotes the fraction of primary and pseudo-primary circuit inputs that generate X-values. The X-sources are chosen randomly per experiment iteration and are static during the iteration. For sequential simulation, a pseudo-primary input selected as X-source generates an unknown value in the first simulation cycle but can capture a known binary value in subsequent cycles. Since the selection may have a strong impact on the result, in particular if nodes with high fanout are selected as X-source, five iterations are performed per experiment. Reported are the averages of the results of the five iterations.

For the evaluation and comparison of the simulation algorithms, the *simulation pessimism* is quantified as follows: The simulation pessimism of algorithm α, $\alpha \in \{3V, RSS, BDD\}$, is the ratio of X^P-valued nodes computed by algorithm α that are proven to carry a binary value by accurate analysis:

$$p_\alpha = \frac{|\mathrm{NX}_\alpha^P| - |\mathrm{NX}|}{|\mathrm{NX}_\alpha^P|}.$$

If $p_\alpha = 1$, then all X^P-valued nodes computed by algorithm α actually carry a binary value according to accurate simulation.

Appendix B.3 provides the tabulated results for reference.

5.6.2 Pessimism in Combinational Logic Simulation

Simulation Pessimism for Varying X-Ratios In the first experiment, three-valued logic simulation, restricted symbolic simulation, the local BDD-based method, and accurate simulation are compared. According to the results in [Kocht11a], the accuracy of the BDD-based method saturates quickly around a BDD size limit of 50 nodes. This threshold is also used for the following experiments. The X-sources are selected randomly per iterations. Then, 1000 random patterns are simulated with the considered algorithms, and the simulation pessimism is computed per iteration. Figure 5.18 shows the averaged pessimism over the five iterations for circuit c7552 and X-ratios from 1% to 99% of the circuit inputs.

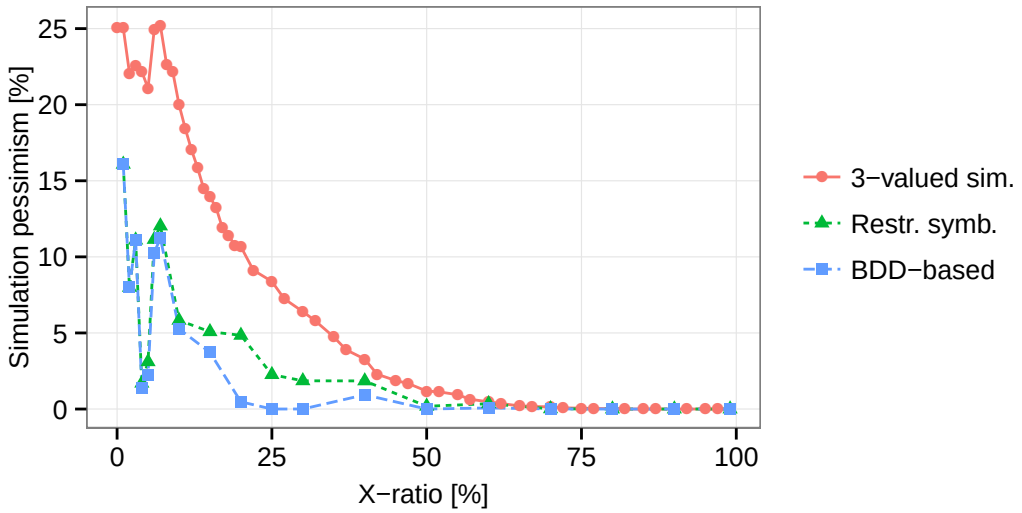


Figure 5.18: Simulation pessimism for circuit c7552.

The pessimism of three-valued simulation exceeds 25% for X-ratios of 1% and 7%, i.e. approximately 25% of the nodes with an X^P-value according to three-valued simulation actually carry a binary value. The pessimism decreases to 0% when the majority of inputs generate X-values. The simulation pessimism does not fall monotonously since different X-ratios and input patterns lead to different numbers of reconvergences within the circuit and therefore also to different numbers of nodes showing an unknown value.

Restricted symbolic simulation and the approximate local BDD-based method are able to reduce the pessimism of three-valued logic simulation and compute tighter lower bounds for X-propagation in the circuit. In general, however, they fail to provide

the exact result. In this particular circuit, both approaches still show a maximum simulation pessimism of over 15%.

Application to Larger Circuits This experiment has been repeated for larger circuits and an assumed X-ratio of 5%. Figure 5.19 shows the simulation pessimism and runtime in seconds for three-valued, restricted symbolic, and the BDD-based approximate simulation algorithm. It also indicates the runtime of the accurate SAT-based simulation as horizontal bar in the lower graphs for comparison.

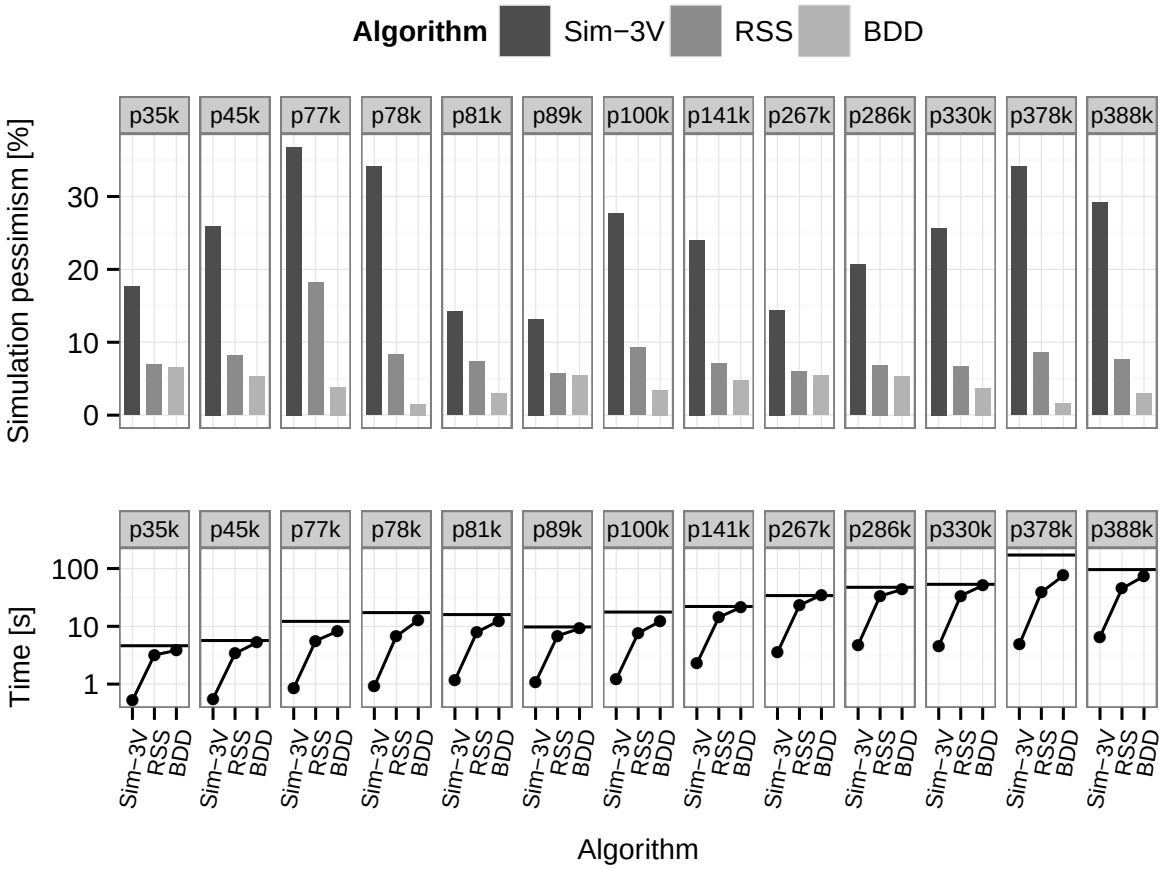


Figure 5.19: Simulation pessimism and runtime of three-valued (Sim-3V), restricted symbolic (RSS), local BDD-based (BDD), and accurate simulation for an X-ratio of 5%.

The pessimism of three-valued simulation ranges from 13.2 to 36.8%, with an average of 24.4%. This means that in average almost one fourth of the circuit nodes classified as X-valued by three-valued simulation actually carry a binary value.

For restricted symbolic simulation, the pessimism still ranges between 5.73 and 18.2%, with an average of 8.21%. Since restricted symbolic simulation simulates a single pattern at a time instead of bit-parallel processing of patterns, runtime increases by 6.99× on average.

The BDD-based algorithm further reduces the pessimism of restricted symbolic simulation. The pessimism ranges between 1.55 and 6.58%, with an average of 4.08%. The more complex BDD-operations result in an increase of runtime by in average 58% compared to restricted symbolic simulation.

The runtime of the accurate SAT-based algorithm exceeds the runtime of the other algorithms for all but one circuit (p267k). On average, the runtime is 15.4× higher than three-valued simulation, or 50.0% higher than the BDD-based simulation.

If X-sources are clustered, the pessimism in conventional simulation algorithms may even be higher since the probability of reconvergences and X-canceling in the transitive fanout of the X-sources rises. In the experiments in [Erb14a] with industrial circuits from NXP, the X-sources are selected as a consecutive part of a scan chain. For example for an X-ratio of 5%, the simulation pessimism of three-valued simulation for circuit p100k increases by 24.5% on average compared to randomly selected X-sources.

Accuracy of Simulation Using Indirect Implications The pessimism of the simulation algorithm using indirect implications proposed in [Kajih04] has been evaluated in [Kocht12]. The experiment computes the number of X^P-valued outputs computed by three-valued simulation that are proven to have a binary value. For the set of ISCAS'85 and ISCAS'89 circuits evaluated in [Kajih04], this number is compared with the result found by the proposed accurate algorithm.

For all circuits, accurate simulation is able to prove a much higher number of X^P-valued signals to have a binary value compared to using indirect implications. For circuit s38417, the average number of identified false X-outputs is more than $20\times$ higher showing that a large fraction of outputs with binary value cannot be discovered by [Kajih04] (cf. Table B.3 in Appendix B.3).

5.6.3 Pessimism in Sequential Logic Simulation

Using the accurate sequential simulation algorithm based on time-frame expansion, the pessimism of three-valued simulation over 100 cycles is assessed for the sequential ISCAS'89 and NXP circuits. For comparison, restricted symbolic simulation has been extended to support multi-cycle simulation as well.

For each circuit, five simulation runs are performed, each considering a set of five input pattern sequences and 100 time frames. The results are averaged over these iterations. Here, 1% or 2% of the pseudo-primary inputs (sequential elements) are assumed to be uninitialized and thus, are selected as X-sources in the first time frame.[2]

Figure 5.20 shows the pessimism per cycle of sequential simulation for circuits s38584 and p141k for an X-ratio of 2%. As seen in the figure, the simulation pessimism of three-valued simulation increases during the first few cycles and saturates at a very high level of approximately 80%. In average over the cycles and circuits, the simulation pessimism is 72% for an X-ratio of 1% and 71% for an X-ratio of 2%.

The figure also shows the pessimism of restricted symbolic simulation, computed accordingly. Restricted symbolic simulation significantly reduces the pessimism compared to three-valued simulation. Yet, the results also demonstrate that depending on circuit structures and input patterns, the prevailing pessimism of restricted symbolic simulation may still be high. The maximum pessimism $p_{RSS} = 71\%$ is reached for circuit p141k in cycle 95.

5.7 Summary

Logic simulation based on n-valued logic algebras pessimistically overestimates the number of X-values in circuits. Various techniques have been proposed in the literature to reduce the pessimism and approximate the accurate result. Of these algorithms, restricted symbolic simulation is able to reduce the pessimism by correctly evaluating simple local reconvergences of X-valued nodes with linear runtime complexity.

The pessimism is further reduced by a novel flexible BDD-based method [Kocht11a] that trades-off accuracy and required computational resources. The method limits

[2] An X-ratio of 5% results in excessive runtimes of the accurate simulation algorithm based on time-frame expansion.

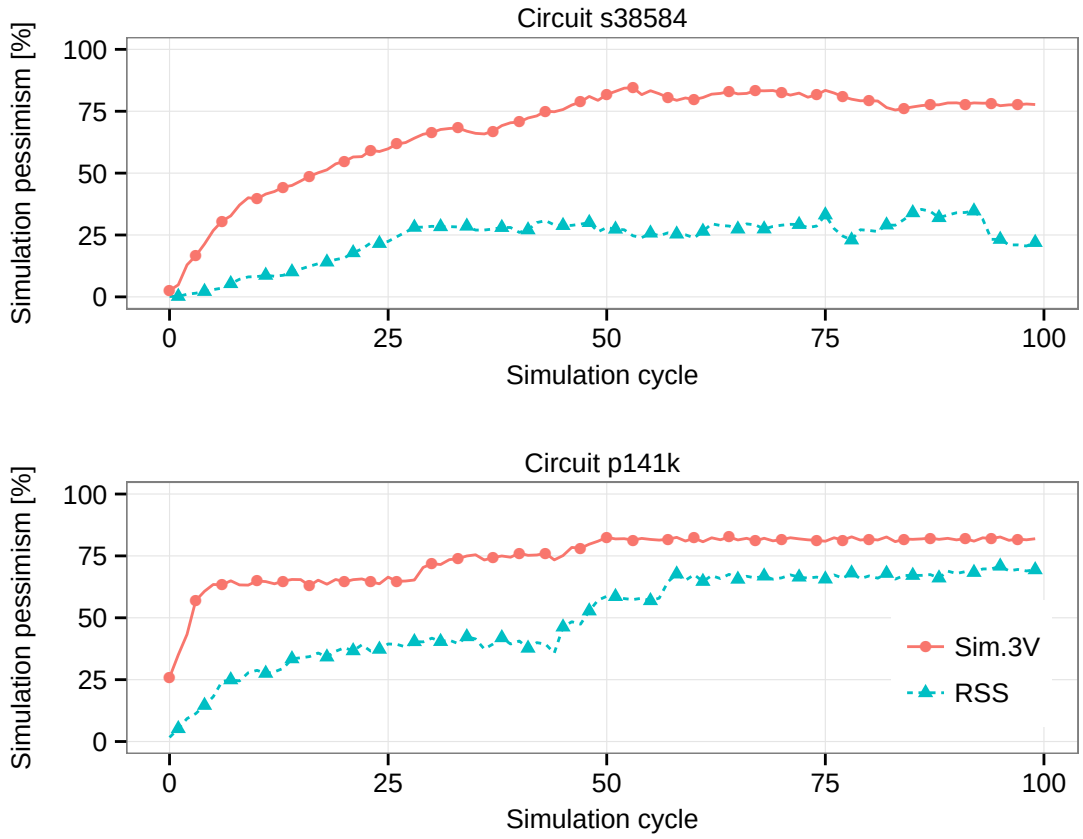


Figure 5.20: Pessimism in three-valued and restricted symbolic sequential simulation for 100 cycles for ISCAS'89 circuit s38584 and NXP circuit p141k for an X-ratio of 2%.

the size of the BDDs which represent the state of circuit nodes symbolically. Since this method prevents an exponential growth of the BDDs, runtime complexity is still polynomial.

The computation of the accurate solution is mapped to Boolean satisfiability [Elm10, Kocht12] and accelerated by use of heuristics [Hille12]. This algorithm has also been extended to perform accurate sequential simulation of a given number of cycles [Erb14a].

The experimental results show the high pessimism of three-valued simulation. Even using restricted symbolic simulation, the remaining pessimism may still exceed 18% in combinational simulation, and 70% in sequential simulation. The proposed approximate BDD-based algorithm achieves a simulation result close to the accurate solution with runtimes only moderately higher than the restricted symbolic simulation. The

average pessimism is only 4.08% for combinational logic simulation.

The runtime of the proposed accurate SAT-based algorithm is kept low by using heuristics to minimize the use of expensive formal reasoning. In the experiments with an X-ratio of 5%, the runtime is on average only 15.4× higher than three-valued simulation, or 50.0% higher than the BDD-based algorithm.

The high pessimism found in three-valued logic simulation by the accurate analysis motivates the investigation of its impact on test quality and the research of accurate algorithms for test generation.

6 Fault Simulation in Presence of Unknown Values

Fault simulation is an essential task during test generation and quality assessment of a VLSI product. Fault simulation is used to analyze the behavior of a circuit model in presence of a given set of faults and input stimuli. It computes the set of faults of a fault model for which the circuit behavior differs from its specified behavior under a given input stimulus. Such faults are called detected. The fault coverage FC of a set of input stimuli is defined as

$$FC = \frac{\mid \text{Detected faults} \mid}{\mid \text{Modeled faults} \mid} \cdot 100\%. \tag{6.1}$$

The fault coverage is an important figure of merit to estimate the product quality of a manufactured chip.

This chapter briefly outlines the fundamentals of fault modeling and conventional fault simulation algorithms based on n-valued logic simulation. These algorithms suffer from pessimism when X-sources are present in a circuit. Section 6.4 describes novel accurate fault simulation algorithms for stuck-at [Hille12] and transition delay fault simulation [Erb14a]. Section 6.5 presents the current state-of-the-art in approximate fault simulation and introduces SAT- and BDD-based algorithms [Kocht11b, Kocht12, Kocht11a] that increase the accuracy of the computed fault coverage in presence of X-values with lower computational requirements by limiting the use of formal reasoning. The chapter concludes with an experimental evaluation in Section 6.6.

6.1 Fault Modeling

Fault modeling is the abstraction of impairments of VLSI circuits to enable algorithmic processing. Following a short introduction, the fault models relevant for this work are

described.

6.1.1 Defects and Fault Modeling

The term defect refers to an unintended structural deviation of the manufactured chip from the specified design. Defects include missing or extra material in the chip, impurities of the materials, or cracks and irregularities in the crystal structure. Causes of defects are impurities of the source materials or of the processing equipment, handling errors, and variations in the manufacturing process [Nassi00].

Fault models abstract from physical defects and their continuous nature. They represent defects as a deviation in a structural model of the system. Faults can be modeled at transaction, register-transfer, gate and electrical level. A fault model should describe the behavior resulting from a defect as accurately as possible, and it should allow for efficient test generation and fault simulation of the comprised faults [Wang06]. Single and multiple fault occurrences, and the temporal behavior of faults (permanent, intermittent or transient) are also distinguished.

During operation, a fault can be activated and wrong information, i.e. wrong output responses or states, may be created in the system and results in an error. During test, such an error may be detected by the test equipment, and the chip can be handled correspondingly, for instance be sorted out.

A large number of different fault models for VLSI circuits have been proposed in the past. Here, only the fault models at gate level relevant for this work are briefly discussed. Extensive overviews of fault models are given in [Bushn00] and [Wunde10].

6.1.2 Classical Fault Models

Stuck-at Faults The predominant fault model in hardware testing is the single stuck-at fault model. The concept of a structural stuck-at fault based hardware test was first published in [Eldre59], and little later, the term *stuck-at fault* was introduced in [Galey61]. A stuck-at fault at a circuit node sets its value to a fixed value, 0 for a stuck-at-0 fault, and 1 for a stuck-at-1 fault. This value is independent of the state of the predecessors of the node.

Since each node may be affected by a stuck-at-0 or a stuck-at-1 fault, the number of single stuck-at faults for a circuit with $|V|$ nodes is $2 \cdot |V|$. Typically, only single stuck-at faults are targeted. The occurrence of multiple stuck-at faults can be represented by single stuck-at faults in a modified circuit graph. The number of fault tuples in a circuit graph grows exponentially with the fault multiplicity. In the remainder of this work, we consider permanent single stuck-at faults and briefly refer to them as stuck-at faults.

Two stuck-at faults in a circuit are equivalent if and only if the Boolean functions of the two corresponding faulty circuits are identical. In this case, there is no input pattern that can distinguish the two faults. Fault collapsing is the process of identifying as few fault equivalence classes as possible for a given circuit. During test generation, only one representative fault of each equivalence class needs to be considered. This reduces the number of faults to be processed significantly [McClu71, Bushn00].

A high stuck-at fault coverage ensures that a high fraction of the nodes in the circuit are excited, and their behavior becomes observable at the circuit outputs during the test. A high stuck-at fault coverage of a test typically implies a high coverage of more complex fault models [Bushn00].

Delay Faults Delay fault models are used to model defective behavior that affects the transition delay of transistors or interconnect wires. Slow-to-rise faults (delayed rising transition) and slow-to-fall faults (delayed falling transition) are distinguished. The three most important delay fault models, the transition delay, the gate delay, and the path delay fault model, are summarized in the following.

In the transition delay fault model, it is assumed that only a single node in the circuit is affected by a defect and exhibits a transition delay higher than its nominal delay. A transition delay fault at a node causes a delayed propagation of a transition through the node such that the transition does not reach the observable outputs at the observation time [Wadsa78]. This implies that a transition delay fault can be detected through a propagation path independent of the path's slack. In a circuit, the number of transition delay faults equals the number of stuck-at faults. At each node, a slow-to-rise and slow-to-fall transition delay fault are considered.

The gate delay fault model also assumes that the delay is lumped at a single node, but takes the delay increase into account [Iyeng90]. Gate delay faults of small size can

only be observed through propagation paths with small slack.

A path delay fault is defined as an increased delay for a rising or falling transition along a particular *path* in the circuit. Thus, the path delay fault model describes defective circuit behavior where multiple nodes exhibit increased transition delay. Such behavior may be caused by multiple defects or parametric changes affecting multiple circuit nodes.

6.2 Fault Detection in Presence of Unknown Values

A fault simulator computes for each fault whether the specified behavior, determined by the fault-free circuit model $G = (V, E)$, and the faulty behavior differs for an input pattern p or pattern sequence $\vec{p}$. The faulty behavior is derived from the faulty circuit model $G^f = (V^f, E^f)$ in which the node at the fault site is substituted according to the particular fault.

In presence of X-values, the effect of a fault may turn an X-value into a binary value of 0 or 1, or vice versa. If an observable output is altered from a constant binary value in the fault-free circuit to an X-value in the faulty circuit, a *potential detection* of the fault can be reported. In this work, definite detection and potential detection of a fault by a given pattern are distinguished:

- ▷ A fault f is definitely detected (DD) if and only if an observable output o exists where the fault effect is present at the observation time independent of the logic value assignment to the X-sources.
- ▷ A fault f is potentially detected (PD) if and only if it is not definitely detected, and there is an observable output o where the fault effect is present for at least one logic value assignment to the X-sources.

For brevity, if a fault f is not definitely detected by a pattern, f will be called undetected in the following.

During the analysis of a pattern, the sets of signal nodes with binary value or X-value differ for the fault-free and the faulty circuit. The superscript f is used to indicate the node value in the faulty circuit under fault f. Let val(v, p) and val$^f(v, p)$ denote the logic value of node v under pattern p in the fault-free and faulty case in presence of X-values (cf. Section 5.1.1).

6.2.1 Stuck-at Fault Detection

A stuck-at fault f is definitely detected (DD) if and only if an output o exists where the fault effect is observable independent of the X-source assignment. Formally, the definite detection of fault f under pattern p is given as

$$\begin{aligned}\text{DD}^f(p) := \exists o \in O :\\ \text{val}(o,p) \in \{1,0\} \wedge \text{val}^f(o,p) \in \{1,0\} \wedge \text{val}(o,p) \neq \text{val}^f(o,p),\end{aligned} \tag{6.2}$$

where O is the set of observable circuit outputs.

Stuck-at fault f is potentially detected (PD) if it is not definitely detected, and an output $o \in O$ exists at which the fault effect can be deterministically measured for at least one X-source assignment:

$$\begin{aligned}\text{PD}^f(p) := \neg\text{DD}^f(p) \wedge \exists o \in O :\\ \text{val}(o,p) \in \{1,0\} \wedge \text{val}^f(o,p) = \text{X}.\end{aligned} \tag{6.3}$$

6.2.2 Transition Delay Fault Detection

The definite detection of a transition delay fault *tf* at node v requires the consideration of at least two cycles. In the first cycle, called *initialization* or launch cycle, node v is driven to a defined value ϕ to activate the fault. For a slow-to-rise transition delay fault, ϕ is 0; for a slow-to-fall fault, ϕ equals 1. In the second cycle, called *propagation* or capture cycle, the value of v is inverted, and the resulting transition is propagated from v to an observable circuit output. In the faulty circuit, the detection of the delayed transition corresponds to the detection of the stuck-at-ϕ fault at node v in the propagation cycle. Thus, the definite detection of *tf* at node v under pattern pair $\vec{p} = (p_{\text{Init}}, p_{\text{Prop}})$ is:

$$\begin{aligned}\text{DD}^{tf}(p_{\text{Init}}, p_{\text{Prop}}) := \text{val}(v, p_{\text{Init}}) = \phi \ \wedge \ \text{val}(v, p_{\text{Prop}}) = \neg\phi \ \wedge \ \exists o \in O :\\ \text{val}(o, p_{\text{Prop}}), \text{val}^{tf}(o, p_{\text{Prop}}) \in \{1,0\} \wedge \text{val}(o, p_{\text{Prop}}) \neq \text{val}^{tf}(o, p_{\text{Prop}}),\end{aligned} \tag{6.4}$$

with O the set of observable circuit outputs in the propagation cycle.

Similar to the potential detection requirement for stuck-at faults, the potential detection of a transition delay fault tf at node v requires that the fault is activated, and its effect can be deterministically measured in the propagation cycle at an output o for at least one X-source assignment:

$$\begin{aligned}\text{PD}^{tf}(p_{\text{Init}}, p_{\text{Prop}}) := \neg\text{DD}^{tf}(p_{\text{Init}}, p_{\text{Prop}}) \ \wedge \ \text{val}(v, p_{\text{Init}}) = \phi \ \wedge \ \text{val}(v, p_{\text{Prop}}) = \neg\phi \ \wedge \\ \exists o \in O : \text{val}(o, p_{\text{Prop}}) \in \{1, 0\} \wedge \text{val}^{tf}(o, p_{\text{Prop}}) = \text{X}. \end{aligned} \tag{6.5}$$

6.2.3 Computational Complexity of Fault Simulation in Presence of Unknown Values

This section elaborates on the computational complexity of stuck-at fault simulation.[1] Pessimistic n-valued fault simulation has polynomial runtime complexity. Accurate combinational and sequential fault simulation based on accurate logic simulation are NP-hard problems.

Complexity of n-valued Fault Simulation

The problem of pessimistic fault simulation in a combinational circuit based on an n-valued logic algebra has been proven to be at least as hard as matrix multiplication in [Harel87]. Experimental results on benchmark circuits show that the runtime complexity lies indeed between $\mathcal{O}(n^2)$ and $\mathcal{O}(n^3)$ w.r.t. the size of the circuit [Goel80]. This observation is also intuitive since with increasing circuit size, the evaluation effort for the fault-free and faulty circuits grows linearly. Also, the number of faults in the stuck-at and transition delay fault models increases linearly. Finally, experience shows that larger circuits require a higher number of test patterns to reach high fault coverage [Goel80].

Complexity of Accurate Fault Simulation

The decision problem SAF-NOT-DD computes whether a stuck-at fault in a combinational circuit in presence of unknown values is *not* definitely detected at output o by

[1] The findings can be extrapolated to transition delay fault simulation which can be mapped to logic simulation and stuck-at fault simulation [Waicu87].

a given pattern p. SAF-NOT-DD can be expressed as a Boolean satisfiability problem derived from Equation (6.2) by restriction to output o and negation:

$$\text{val}(o,p) = \text{X} \ \vee \ \text{val}^f(o,p) = \text{X} \ \vee \ \text{val}(o,p) = \text{val}^f(o,p),$$

which can also be stated as:

$$\begin{aligned} &\exists \text{ Assignments } a_1, a_2 \in \mathcal{A}_X \ : \text{val}(o,p,a_1) \neq \text{val}(o,p,a_2) \vee \\ &\text{val}^f(o,p,a_1) \neq \text{val}^f(o,p,a_2) \vee \text{val}(o,p,a_1) = \text{val}^f(o,p,a_1).^2 \end{aligned} \tag{6.6}$$

SAF-NOT-DD is an NP-complete problem:[3]

- ▷ SAF-NOT-DD is an NP problem: A model or satisfying assignment for Equation (6.6) is verified by computing and comparing the values of output o using two-valued logic simulation of the given pattern p and the two X-source assignments a_1, a_2. This requires linear runtime w.r.t. the circuit size $|V|$.
- ▷ SAF-NOT-DD is NP-hard: There is a polynomial time reduction of an instance of CIRCUIT-SAT (cf. Section 5.1.2) to an instance of SAF-NOT-DD.

To prove NP-hardness by reduction of CIRCUIT-SAT to SAF-NOT-DD, we consider a combinational circuit with one output o. To determine whether there is an assignment to the circuit inputs I such that o evaluates to *true*, we assume that *all* circuit inputs are X-sources, i.e. $S_X = I$, and the input pattern is $p = \emptyset$. This transformation does not modify the circuit and completes in polynomial time.

The stuck-at-1 fault at output o is definitely detected if and only if $\text{val}(o,p) = 0$. Thus, if the stuck-at-1 fault at o is not definitely detected (as decided by SAF-NOT-DD), there is at least one assignment $a \in \mathcal{A}_X$ to the X-sources such that $o = 1$ in the fault-free circuit. In that case, assignment a satisfies the CIRCUIT-SAT instance.

The accurate computation whether a fault is not definitely detected in a sequential circuit in a *fixed* number of cycles is also an NP-complete problem. In [Becke99], the problem is shown to be NP-hard by a polynomial time reduction of the non-tautology problem [Garey79]. The problem is NP-complete since for a found assignment sequence to the X-sources (bound in the number of cycles), a two-valued logic simula-

[3] This can be extended to multiple or all circuit outputs without changing the result since the number of outputs is bound by the circuit size $|V|$.

tion can be performed in linear runtime in the circuit size and number of cycles to verify that the fault is not detected for the assignment sequence.

6.3 Conventional Fault Simulation Algorithms and Their Limitations

The following sections summarize the predominant fault simulation algorithms based on n-valued algebras for combinational and sequential circuits. In presence of X-values, the number of definitely detected faults is underestimated due to the inherent pessimism of node evaluation based on n-valued logic algebras. In addition, state-of-the-art algorithms classify faults as potentially detected if there is an output with a binary value in the fault-free circuit that carries an X^P-value in the faulty circuit. Note that such a classification is inaccurate and may even overestimate the potentially detected faults. Figure 6.1 a) shows a case where the classification of a stuck-at-1 fault as potential detection is optimistic, i.e. the fault is marked as potentially detected but cannot be detected for any possible assignment to the X-sources. Figure 6.1 b) shows a case where the result is pessimistic, i.e. a fault marked as potentially detected is definitely detected.

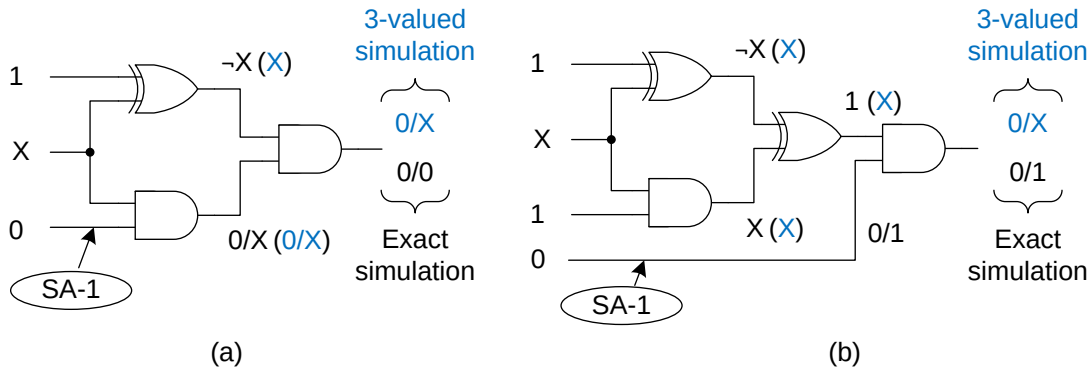


Figure 6.1: Inaccuracy in the classification of potentially detected faults in fault simulation based on three-valued logic: a) optimistic and b) pessimistic result.

6.3.1 Combinational Fault Simulation

In combinational fault simulation, a combinational circuit or the combinational part of a sequential circuit is analyzed. In conventional algorithms, the node evaluation in the fault-free and faulty circuits is based on pessimistic n-valued logic algebras.

Serial Fault Simulation

Serial fault simulation evaluates the circuit nodes in G and G^f for each fault f and input pattern serially and separately. The resulting computational effort is high since similarities between fault evaluations are not exploited. If fault dropping is applied, i.e. faults detected by one pattern are excluded from the simulation of subsequent patterns, the runtime is significantly reduced. The principle of serial fault simulation is shown in Algorithm 6.1. The algorithms discussed next increase efficiency by reducing the number of node evaluations, and by pattern- or fault-parallel processing.

Algorithm 6.1 Serial fault simulation of pattern p and a set of faults F.

```
 1: procedure SERIALFAULTSIMULATION(V, p, F)
 2:     for all f ∈ F do
 3:         if f is yet undetected then                    ▷ Apply fault dropping.
 4:             o ← SIMULATECIRCUIT3VLOGIC(p, V)           ▷ cf. Section 5.1.3
 5:             o^f ← SIMULATECIRCUIT3VLOGIC(p, V^f)
 6:             if o ≠ o^f then   ▷ Responses of fault-free and faulty circuit differ.
 7:                 Mark f as detected.
 8:             end if
 9:         end if
10:     end for
11: end procedure
```

Concurrent Fault Simulation

The concurrent fault simulation algorithm processes patterns serially, but considers a set of faults in parallel in event-based manner. It exploits that subsequent patterns typically impose only few state changes in the circuit. The algorithm computes for the considered faults local differences of fault-free and faulty behaviors. A pattern causes an event for a particular fault, i.e. a value difference compared to the fault-free case, if a fault is activated or propagated. This difference is propagated until it either vanishes

or becomes observable at a circuit output [Ulric73]. Compared to serial simulation, the concurrent algorithm achieves a speed-up since the evaluation of G and G^f stops as soon as all the differences and events imposed by the pattern have been processed.

The local evaluation of the node states is based on n-valued logic algebras. Typically, two, three or four values are used to represent the state in G or G^f, allowing to model unknown and high impedance states.

Deductive Fault Simulation

In deductive fault simulation, an explicit logic simulation of the fault-free circuit is performed for a pattern, and detected faults are directly derived from the resulting values of circuit nodes [Armst72]. Each circuit node is assigned a list which stores faults observable at the node under pattern p. The circuit graph is traversed in topological order. At each node the fault list is updated such that it contains (i) the activated faults of the node and (ii) the faults that have been activated by the pattern in the transitive fanin of the node and propagated to the node along sensitized paths. In contrast to the concurrent fault simulation algorithm, deductive fault simulation updates the fault list of a node by efficient set operations (set intersection, union and difference) in a single step per pattern.

The algorithm presented in [Armst72] proposes the inaccurate evaluation of fault lists in presence of unknown values. If a fault is propagated to an input of a node that has at least one input with unknown value, but a binary output value in the fault-free case, the fault is marked as potentially detected. Such a classification is inaccurate (cf. Section 6.3).

Pattern-Parallel Single Fault Propagation Fault Simulation

In pattern-parallel single fault propagation based fault simulation (PPSFP), the circuit behavior is analyzed for multiple input patterns in parallel. The input patterns and the resulting node values are grouped and encoded into one or multiple words of the machine architecture. Bit-parallel operations are used for node evaluations which are based on n-valued algebras.

For each group of patterns, the fault-free behavior of the circuit is computed once. Then, it is checked explicitly for each fault whether the faulty behavior differs from the faulty-free behavior [Waicu85].

Analysis of Signal Dominators and Fanout-Free Regions The computational effort can be reduced by exploiting node dominators[4] of the fault sites. For a fault f with a node dominator dom(f) observable at a circuit output under pattern p, f is detected if and only if it is activated, and the fault effect is propagated to its dominator. This allows to divide the explicit analysis for every fault f into the computation of the observability of fault dominators, and the computation of fault activation and propagation to the dominators [Hong78, Lee91].

The observability of dominator dom(f) is given by the Boolean difference of the function Ψ implemented by the transitive fanout of dom(f) w.r.t. dom(f) under pattern p. It can be computed using the co-factors of Ψ:

$$\frac{d\Psi(p)}{d\text{dom}(f)} = \Psi_{\text{dom}(f)=0}(p) \oplus \Psi_{\text{dom}(f)=1}(p). \tag{6.7}$$

To compute the Boolean difference efficiently, the transitive fanout of dom(f) is evaluated by an event-based simulator with the complemented value of the fault-free case. The simulation stops when the difference to the fault-free values in the transitive fanout either vanishes, or an output is reached. Only in the latter case, dom(f) is observable under p.

Within a fanout-free region (FFR), the information obtained from fault-free analysis is sufficient to determine the activation of a fault and its local propagation to the leaf of the fanout-free region, i.e. to a fanout stem or circuit output. Fault detection of the activated and propagated faults in the fanout-free region then only depends on the observability of the leaf, which dominates all nodes in the region. The region analysis is conducted for multiple patterns in parallel to achieve high speed-ups [Antre87, Schul88, Lee91].

Apart from vector based processing, pattern parallel fault simulation can further be accelerated by using data-parallel machine architectures. The execution on the pro-

[4]Node d dominates node v if all paths starting from v and ending in a circuit output contain d.

cessing units of graphics processors, for example, allows an acceleration by one to two orders of magnitude [Kocht10a, Li11].

The node evaluation in the fault-free circuit, the evaluation of the dominator observability, and the reasoning in fanout-free regions is based on n-valued logic algebras. Consequently, the computed fault coverage underestimates the actual coverage of a test set if X-values are present.

6.3.2 Fault Simulation in Sequential Circuits

Multi-cycle fault simulation is essential for test validation and the computation of fault coverage in partial scan circuits and circuits with memory blocks that cannot be directly controlled. In such circuits, a synchronizing or reset sequence [Cho93, Lee96] is used to set the memory elements to a defined state. Multi-cycle fault simulation is also required in complex test applications with multiple capture cycles [Zoell10] and in software-based self-tests for processors [Psara10].

Multi-cycle fault simulation must consider the effects of faults activated in previous simulation cycles and their propagation. Both fault activation and propagation to an observable node may require multiple cycles, as illustrated in Figure 6.2.

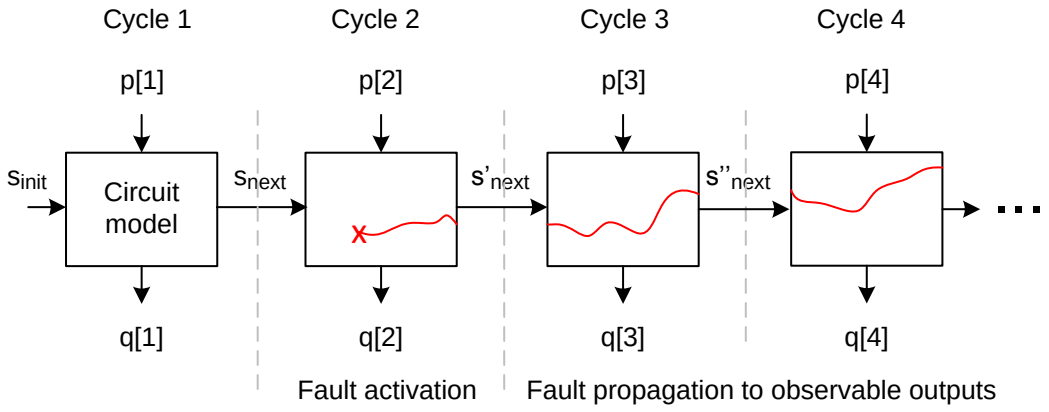


Figure 6.2: Multi-cycle fault simulation.

State-of-the-art algorithms for multi-cycle fault simulation comprise the event-based concurrent algorithm [Ulric73], the differential simulation of faulty machines [Cheng90, Nierm92], and the application of pattern-parallel single fault propagation techniques [Goude91].

In these algorithms, three- and four-valued logic algebras are used for node evaluation in the fault-free and faulty circuit. Consequently, the simulation result in presence of X-values is inaccurate and may underestimate the actual fault coverage of a test set. Furthermore, as discussed in Section 5.5, the simulation of multiple cycles using n-valued algorithms may increase the simulation pessimism from cycle to cycle.

6.3.3 Transition Delay Fault Simulation

Delay faults in synchronous sequential circuits are detected if a signal transition excited by an input transition reaches an observed circuit node later than the measurement time. To compute the fault coverage, two cycles, the *initialization* and the *propagation* cycle, need to be considered. The initialization cycle imposes an initial value at the fault site. During the propagation cycle, a transition of an input propagates through the fault site to an observable circuit node. Consequently, at least two patterns p_{Init} and p_{Prop} need to be simulated as shown in Figure 6.3.

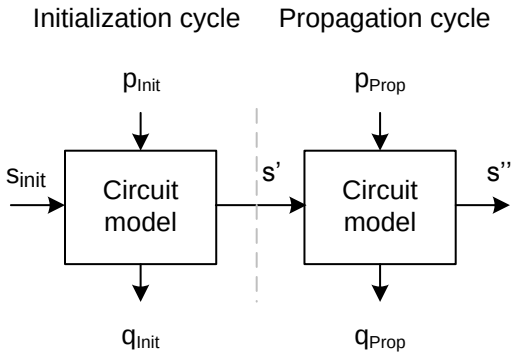


Figure 6.3: Delay fault simulation with initialization and propagation cycle.

Transition and gate delay fault simulation exploits the correspondence between a slow-to-rise fault in the propagation cycle and a stuck-at-0 fault at that fault site [Waicu87]. A slow-to-rise fault is detected if p_{Init} implies the initial value of 0 at the fault site, and if p_{Prop} detects the corresponding stuck-at-0 fault in the propagation cycle.[5] In consequence, stuck-at fault simulation algorithms can be extended for efficient transition and gate delay fault simulation [Schul87].

[5] A similar correspondence exists between a slow-to-fall fault and a stuck-at-1 fault at the fault site.

In addition to the discussion of multi-cycle fault simulation above, the accuracy of delay fault simulation is impaired because fault detection requires binary values at the fault site both in the initialization and the propagation cycle.

6.4 Accurate Fault Simulation Algorithms

Accurate fault simulation classifies a set of target faults as definite detect, potential detect or undetected for a test set in presence of X-values (cf. Section 6.2). In principle, it is possible to decide about *definite detection* of a fault by exhaustive simulation of all possible assignments to the X-sources for each pattern. However, this computation has to be performed for each fault separately and causes excessive computational effort. The following sections show how formal reasoning and heuristics are used in the recently published algorithms [Hille12, Erb14a] to conduct efficient accurate stuck-at fault simulation and transition delay fault simulation even for larger circuits.

6.4.1 Overview of Accurate Fault Simulation

The proposed accurate fault simulation algorithm is based on the heuristics and formal SAT reasoning presented in Sections 5.2.5 and 5.4 to accurately compute the signal node values in the fault-free and faulty circuits. An overview of the fault simulation of a pattern p is given in Figure 6.4: Three-valued fault simulation is used to mark as many target faults as possible as definitely detected. For the remaining faults, an exact analysis is conducted.

The analysis starts with the accurate logic simulation according to Section 5.4 of the fault-free circuit for pattern p to derive the set of activated faults. These faults are then analyzed serially. For the faulty simulation of an activated fault f, f is injected into the circuit model. The algorithm then proceeds in two phases, similar to the evaluation in the fault-free case: A heuristic simulation and an accurate calculation step. During the heuristic simulation, the behavior of the faulty circuit is simulated in event-driven manner using restricted symbolic simulation and two-valued parallel-pattern logic simulation. The two-valued pattern-parallel simulation evaluates a limited number of randomized assignments to the X-sources. If the results of the restricted symbolic simulation allows fault classification as definite detection, or if the simulation results of

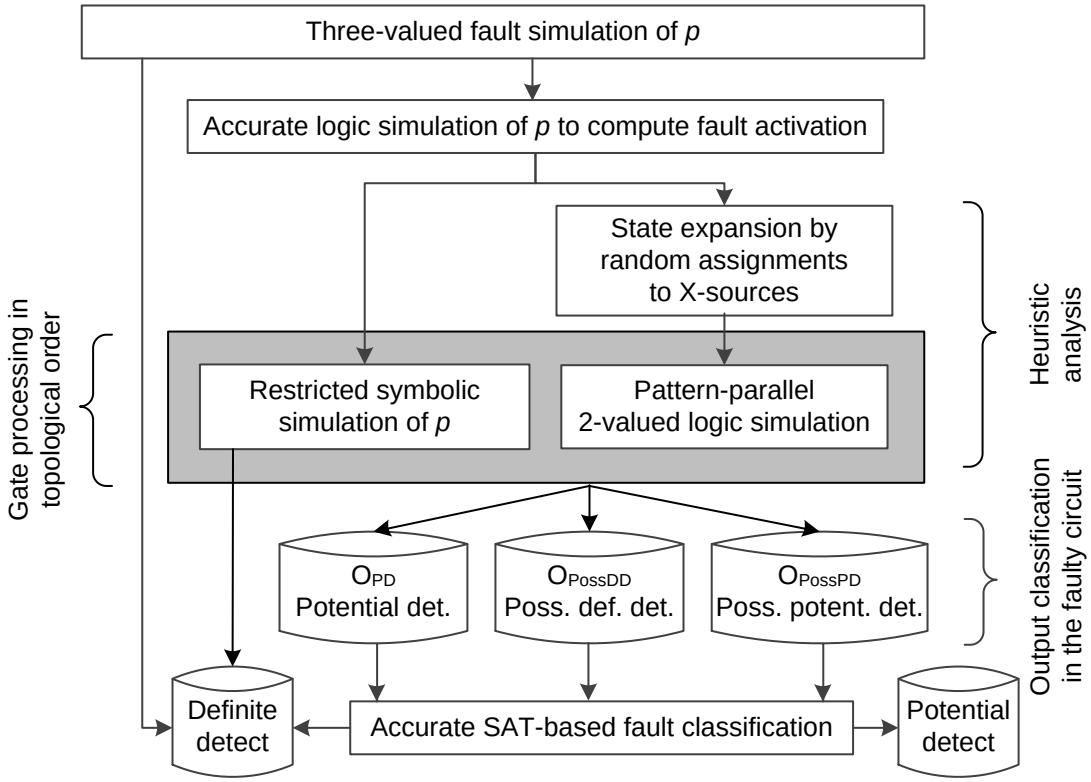


Figure 6.4: Accurate fault simulation for a pattern p and classification as definite detect (DD) or potential detect (PD).

randomized X-source assignments classifies a fault as undetected, further analysis is not required. Otherwise, the SAT solver is invoked to analyze the outputs reachable from the fault site in the faulty circuit. Internal signals within the faulty circuit do not need to be considered because the values at observable outputs are sufficient to reason about fault detection.

6.4.2 Fault Analysis by Restricted Symbolic Simulation and Pattern-Parallel Simulation

For each activated fault f, the circuit outputs $o_1, \ldots, o_k$ in the transitive fanout of f are classified using the results of the faulty circuit simulations. According to the detection conditions stated in Section 6.2, only those outputs o_i are considered which carry a binary value in the fault-free circuit: $\text{val}(o_i, p) \in \{0, 1\}$.

If there is one output o_i with a binary value in the faulty case $\text{val}^f(o_i, p) \in \{0, 1\}$ ac-

cording to restricted symbolic simulation, and $\text{val}^f(o_i, p) \neq \text{val}(o_i, p)$, then f is marked as definitely detected, and the algorithm proceeds with the next undetected fault. If all outputs in the transitive fanout have binary values equal to the fault-free case, i.e. $\text{val}^f(o_i, p) \in \{0, 1\}$ and $\text{val}^f(o_i, p) = \text{val}(o_i, p)$ for $1 \leq i \leq |O|$, then f is undetected by the pattern.

Otherwise, there is at least one output with a value different from the fault-free case. The outputs are divided into three sets, as shown in Figure 6.5: Potential detect outputs O_{PD}, possibly definitive detect outputs O_{PDD}, and possibly potential detect outputs O_{PPD}. The sets O_{PDD} and O_{PPD} contain outputs for which the analysis result so far does not allow to decide about definite or potential detection of the fault, and further analysis is required.

The set O_{PD} contains all outputs at which the fault can be potentially detected: An output o_i is added to the set O_{PD} if the faulty value z_{o_i} in the pattern-parallel simulation of randomized X-source assignments does not equal $[0, \ldots, 0]$ nor $[1, \ldots, 1]$.[6] In this case, there are at least two different assignments to the X-sources that cause different values at o_i in the faulty circuit and consequently, o_i depends on the X-sources.

For an output o_i for which restricted symbolic simulation could not compute a binary state, and z_{o_i} equals either $[0, \ldots, 0]$ or $[1, \ldots, 1]$ after state expansion, it is not known whether it depends on the X-sources. A formal analysis will later compute its accurate state. If all $z^j_{o_i}$ $(0 \leq j \leq 63)$ are equal to $\neg\text{val}(o_i, p)$, o_i is added to O_{PDD} since it may be an output at which the fault can be definitely detected. If the formal analysis later proves that o_i carries a binary value, then f is definitely detected. Otherwise, f is potentially detected.

On the other hand, if all $z^j_{o_i}$ $(0 \leq j \leq 63)$ are equal to $\text{val}(o_i, p)$, o_i is added to O_{PPD} since it may be an output at which the fault can be potentially detected. If the formal analysis reveals that o_i has an X-value, then f is potentially detected. Otherwise, f cannot be detected at o_i under pattern p at all.

6.4.3 Fault Classification by SAT-based Reasoning

If the set O_{PDD} is not empty, the output values in the faulty circuit are computed using the incremental SAT solver. This is similar to the fault-free case discussed in

[6] z_{o_i} is the 64-bit simulation value of node o_i after state expansion (cf. Section 5.4.1).

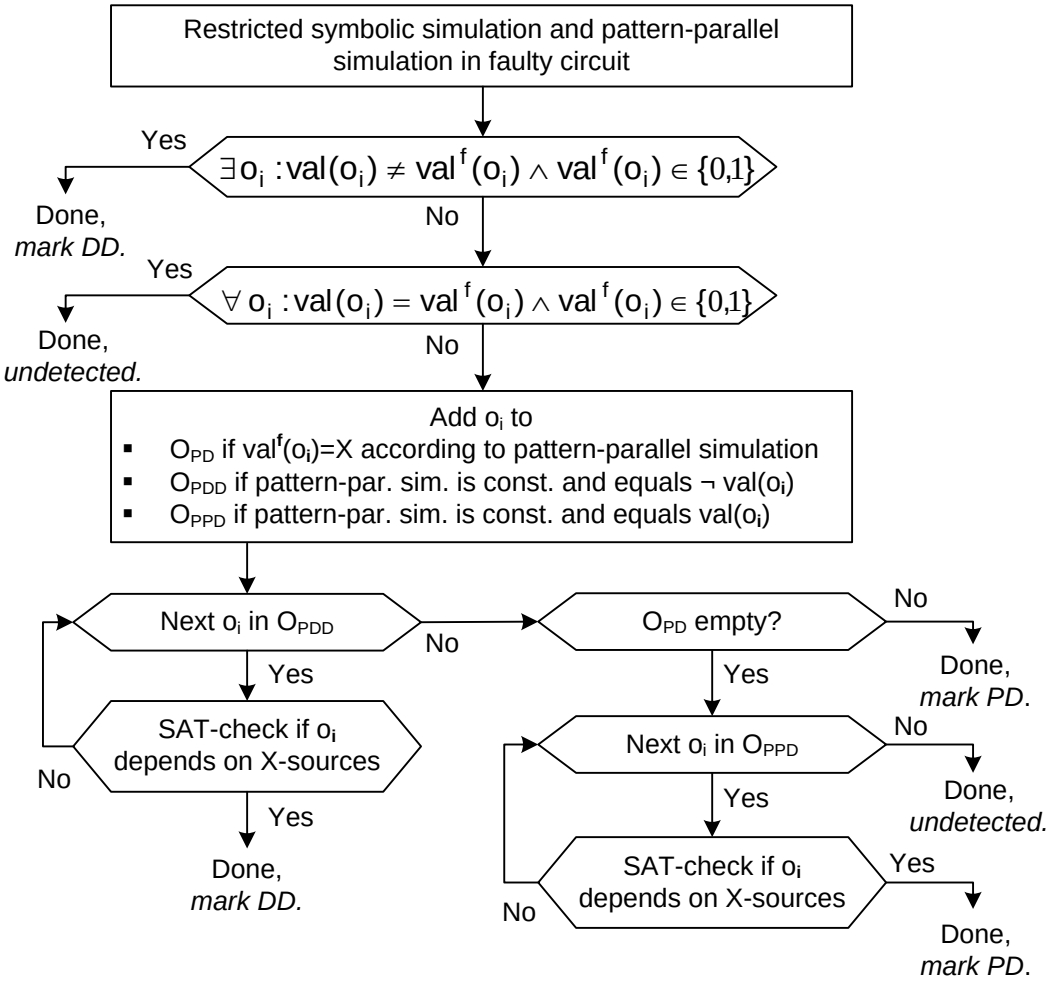


Figure 6.5: Output classification in the faulty circuit to determine definite detection (DD) or potential detection (PD) of fault f.

Section 5.2.5: A SAT instance is constructed which is satisfiable if and only if the considered output depends on the X-sources and has an X-value. If output o_i has an X-value in the faulty circuit, o_i is removed from O_{PDD} and added to O_{PD}. In the other case, the fault is marked as definitely detected because the formula

$$\mathrm{val}(o_i, p) \in \{0,1\} \wedge \mathrm{val}^f(o_i, p) \in \{0,1\} \wedge \mathrm{val}(o_i, p) = \neg \mathrm{val}^f(o_i, p) \tag{6.8}$$

evaluates to true. Thus, the fault is detected for all possible assignments to the X-sources. Then the next undetected fault is analyzed.

If O_{PDD} is empty and O_{PD} is not empty, the stuck-at fault is marked as potential detection, and the algorithm proceeds with the next undetected fault.

If the current fault is neither definitely nor potentially detected and O_{PPD} is not empty, the SAT solver is used to determine if one of the outputs in O_{PPD} has an X-value in the faulty circuit. In that case, the fault is marked as potentially detected. If all outputs in O_{PPD} have a binary value in the faulty circuit equal to their value in the fault-free circuit, the fault remains unmarked and undetected by pattern p.

6.4.4 Extension to Accurate Transition Delay Fault Simulation

Accurate transition delay fault simulation [Erb14a] requires time frame expansion of the circuit model as discussed in Section 5.5.3, and a modification of the fault simulation algorithm introduced above. According to the definition in Section 6.2.2, a transition delay fault is definitely detected if

- ▷ the fault site carries the required initialization value ϕ in the initialization cycle (for a slow-to-rise (slow-to-fall) fault, ϕ is 0 (1)), and
- ▷ the stuck-at-ϕ fault describing the behavior of the transition delay fault in the propagation cycle is definitely detected.

The accurate analysis of a transition delay fault is conducted incrementally: After the accurate simulation of the initialization cycle, it is checked whether the fault site carries the value ϕ in the fault-free circuit.

If the fault site has value $\neg\phi$ in the initialization cycle, the fault is marked as undetected for the considered pattern pair. If the fault site has the initialization value ϕ, the stuck-at-ϕ fault in the propagation cycle is analyzed using the algorithm of Section 6.4.1. If the stuck-at fault is definitely detected, the transition delay fault is also marked as definitely detected. If the stuck-at fault is potentially detected in the propagation cycle, the transition delay fault is marked as potentially detected. It is marked undetected if the stuck-at fault is not detected.

6.5 Approximate Fault Simulation Algorithms

For very large circuits or a very large number of patterns under analysis, the accurate fault simulation may be computationally too expensive. Approximate fault simulation algorithms aim at a reduction of the simulation pessimism of n-valued algorithms

at lower computational costs. In general, these algorithms cannot provide the exact result.

This section discusses approximate fault simulation algorithms based on static learning, restricted fault simulation, as well as novel hybrid techniques that combine limited formal reasoning and pessimistic algorithms.

6.5.1 Fault Simulation Based on Static Learning

The static learning based algorithm for increased accuracy in logic simulation (Section 5.3.2) has also been applied to fault simulation [Kajih04]. An indirect implication between two nodes $(a \rightarrow b)$ resulting from static learning in the fault-free circuit is used during evaluation of the faulty circuits if the fault cannot affect the implication. However, if either node a or node b is reachable from the fault site, any learnt implication between nodes a and b is neglected during the analysis of the faulty circuit. This trades-off the resulting accuracy with a more expensive, separate learning step for each faulty circuit.

The method is further limited by the incompleteness of static learning, which detects only a subset of invariants in a circuit (cf. Section 5.3.2). Static learning reduces the pessimism in conventional fault simulation algorithms only slightly, as shown in the experimental evaluation in Section 6.6.

6.5.2 Fault Simulation Based on Restricted Symbolic Logic

Restricted symbolic simulation (Section 5.3.3) can also be applied in fault simulation to reduce the pessimism due to X-values. For this purpose, the three-valued node evaluation the fault simulation algorithm is substituted by restricted symbolic simulation for the fault-free and faulty circuit evaluation.

Figure 6.6 shows a circuit with one X-source at node b and input pattern $(a, c) = (0, 1)$. In this circuit, restricted symbolic simulation is able to correctly evaluate the X-reconvergence at node q in the fault-free and faulty circuit under the stuck-at-1 fault at node a. Three-valued fault-simulation fails to compute the detection of the stuck-at-1 fault at a since the reconvergence at q is evaluated pessimistically and results in an X^P-value at q in both the fault-free and faulty circuit.

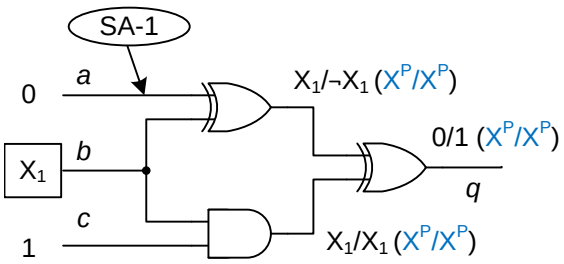


Figure 6.6: Fault-free/faulty node values according to restricted symbolic simulation and three-valued simulation (in blue).

The complexity of fault simulation based on restricted symbolic simulation is still polynomial in the circuit size since the complexity of node evaluation is linear in the circuit size.

6.5.3 BDD-based Hybrid Fault Simulation

The hybrid fault simulation approach of [Becke99] allows sequential fault simulation with increased accuracy. It dynamically selects between three fault simulation algorithms with different levels of accuracy, depending on the available memory resources. The three simulation algorithms comprise

- ▷ an accurate BDD-based fault simulation,
- ▷ an approximate fault simulation using accurate BDD-based logic simulation in the fault-free circuit and three-valued logic simulation in the faulty circuit, and
- ▷ pessimistic three-valued fault simulation for both the fault-free and faulty circuits.

When the accurate fault simulation is applied, the symbolic state of sequential elements resulting from symbolic simulation of cycle t must be stored for the simulation of the subsequent cycle $t + 1$. In the same way, the symbolic state in each of the faulty circuits must be stored to obtain the accurate result. This may easily exceed the available memory resources, in which case the algorithm changes the fault simulation algorithm and re-simulates the cycle with reduced accuracy. The algorithm is less suitable for circuits with many X-sources.

6.5.4 SAT-Based Approximate Fault Simulation

The following simulation algorithm limits the application of formal reasoning in space. It conducts a SAT-based analysis of the fault-free circuit to obtain exact fault activation information, but computes the fault propagation in the faulty circuit pessimistically using three-valued simulation [Kocht11b, Kocht12].

Figure 6.7 shows the overall flow of the algorithm: First, each pattern is accurately simulated for the fault-free circuit using the SAT-based method of Section 5.2 or 5.4 (step ① and ② in the figure). The fanout-free regions (FFR) in the circuit are then processed one at a time. The activated faults in the fanout-free regions are extracted using the accurate logic values of the fault-free case. To determine the observability of the activated faults, the corresponding fanout stems are evaluated by explicit fault injection and simulation (step ④).

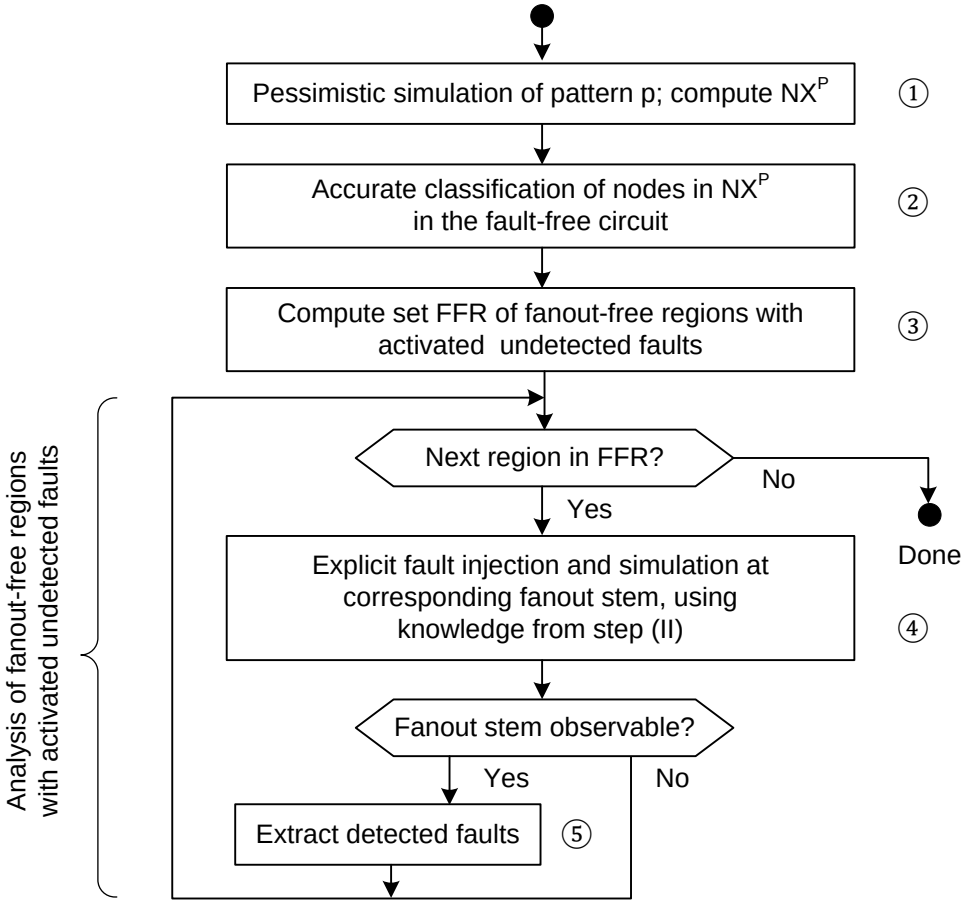


Figure 6.7: Approximate SAT-based fault coverage computation.

Fault Classification For each fanout-free region, the activation of each fault in the region is determined using the node values from the accurate logic simulation. For the local propagation of the fault effects to the corresponding fanout stem, the accurate values are used as well: A fault effect only propagates to the stem if there are non-controlling values at all off-path nodes along the path. Within the fanout-free region, the fault effect can only propagate along a single path towards the next fanout stem. Off-path nodes of this propagation path are not affected by the fault. However, a faulty value at a node may influence a reconvergence of X-values in the output cone of the fault as shown in Figure 6.8. While fault activation can be accurately derived based on the values of the accurate logic simulation, the local propagation to the corresponding fanout stem is only pessimistically computed by path tracing in the fanout-free region.

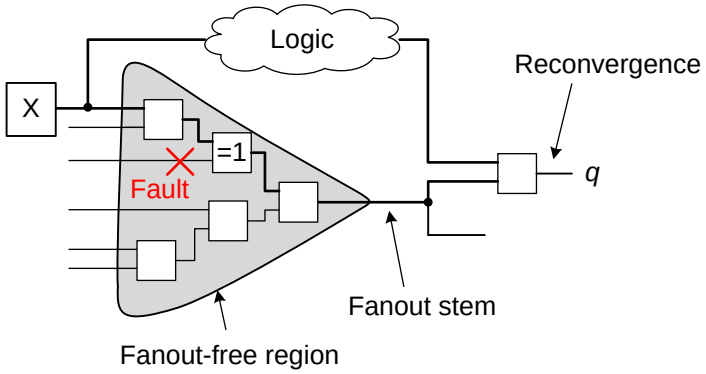


Figure 6.8: Fault impacting the X-reconvergence through the fanout-free region at node q.

If faults are activated and propagated to a stem, explicit fault injection and simulation starting at the stem is conducted to determine stem observability at the outputs or at an intermediate node dominator [Lee91]. Instead of an accurate observability analysis of the fanout stem, a pessimistic three-valued logic simulation is performed in step ④ to reduce the simulation effort for the observability computation. In the faulty circuit, the accurate input values of the fanout cone of a stem are obtained from step ② as illustrated in Figure 6.9.

Remaining Pessimism The algorithm cannot provide the accurate fault coverage of a test pattern because of the pessimistic evaluation of nodes in the transitive fanout of faults:

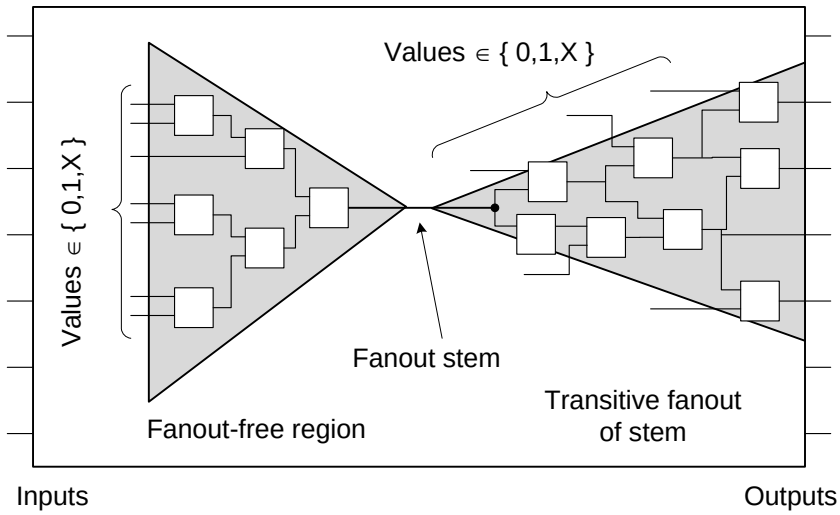


Figure 6.9: Computation of fanout stem observability and fault detection in fanout-free regions.

- ▷ A reconvergence of X-valued nodes lies in a fanout-free-region and is evaluated pessimistically during local propagation analysis.

- ▷ An X-reconvergence in the transitive fanout of a fanout stem is evaluated pessimistically during the stem observability computation.

An example is given in Figure 6.10 where the stuck-at-1 fault at node a influences the reconvergence of the X-source b at node q. This definitely detected fault is pessimistically classified as potentially detected.

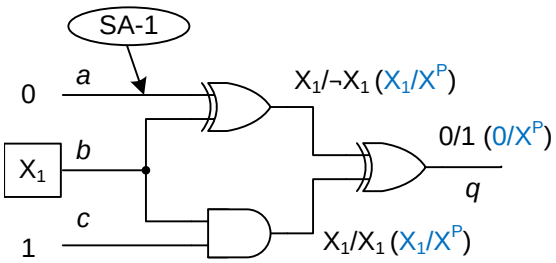


Figure 6.10: Accurate (black) and approximate SAT-based (blue) fault simulation result (Fault-free/faulty circuit state).

6.5.5 Approximate Fault Simulation using Restricted Local BDDs

The restricted symbolic simulation using BDDs (Section 5.3.4) can be applied in fault simulation to trade-off simulation accuracy and required computational resources by setting different BDD node limits. In the proposed algorithm [Kocht11a], the BDD-based restricted symbolic simulation with increased accuracy is applied in those simulation steps where the incurred computational effort is still low.

The overall flow of the proposed fault simulation algorithm is shown in Figure 6.11. For each pattern, the algorithm starts with BDD-based restricted symbolic simulation of the fault-free circuit. The computation of fault activation and fault propagation is performed separately: The fanout free regions of the circuit are processed one at a time. The algorithm determines fault activation and local fault propagation in the fanout-free region using the node values computed by the BDD-based simulation.

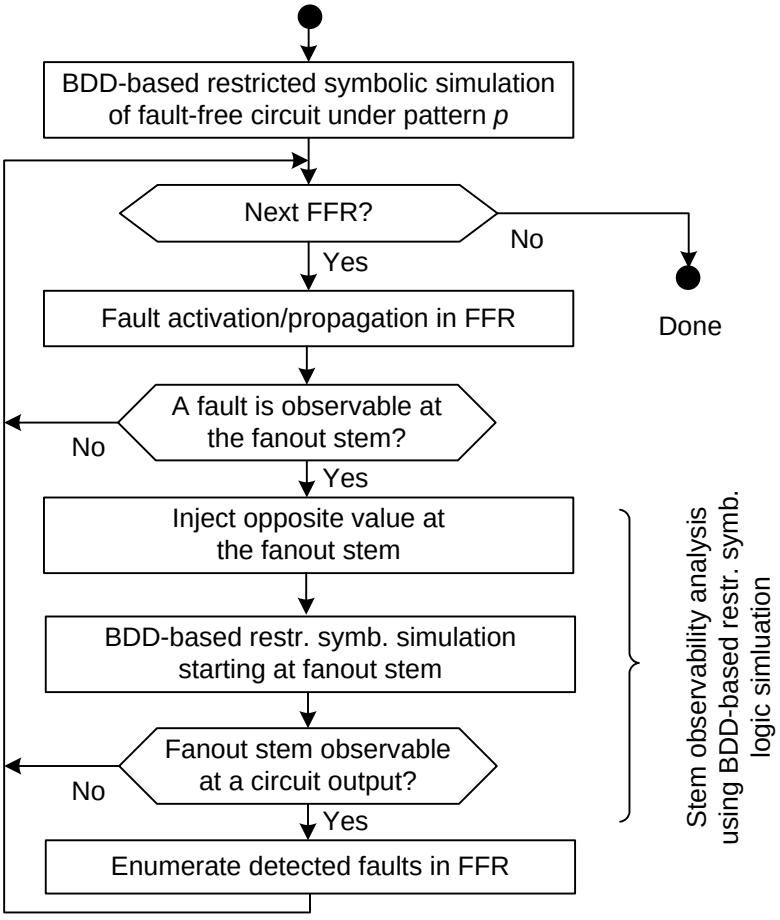


Figure 6.11: Fault simulation flow based on local BDDs.

In the fanout-free region of the faulty circuit, the fault effect propagates only along a single path to the fanout stem. As shown in Figure 6.8, this path may be part of a branch of an X-valued fanout and reconvergence. The value of the reconvergence may be impacted by the fault. As in the approximate SAT-based method above, such cases are evaluated pessimistically during local propagation analysis with three-valued reasoning.

The algorithm checks for each fanout-free region if it contains undetected faults that are activated by the test pattern and observable at the corresponding fanout stem. In that case, the algorithm proceeds with the stem observability analysis. In contrast to [Kocht11b, Kocht12], where the stem observability analysis is conducted using a pessimistic three-valued logic simulation, this fault simulation algorithm increases the accuracy of this step as well. The simulation starts at the fanout with the opposite value of the fault-free circuit and stops at outputs. The resulting logic values are compared with the values of the fault-free simulation. If the binary values differ, the fanout stem is observable and therefore, the activated and locally propagated faults in the fanout-free region are detected by pattern p.

6.6 Evaluation on Benchmark and Industrial Circuits

For stuck-at fault simulation, the increase in fault coverage by fault simulation based on restricted symbolic logic (Section 6.5.2), by the approximate SAT-based algorithm (Section 6.5.4), by the BDD-based algorithm (Section 6.5.5), and by the accurate SAT-based algorithm (Section 6.4) is evaluated. For transition delay fault simulation, the accurate algorithm of Section 6.4.4 is compared to three-valued simulation.

The conducted experiments are similar to the evaluation of logic simulation algorithms (cf. Section 5.6.1): 1000 randomized input patterns are analyzed for different sets of X-sources. The number of detected faults is averaged over the different sets and compared. The results are tabulated in Appendix B.4.

6.6.1 Pessimism in Stuck-at Fault Simulation

Analysis for Varying X-Ratios For circuit c7552, Figure 6.12 shows the increase in fault coverage of the investigated accurate and approximate algorithms w.r.t. three-

valued fault simulation for different X-ratios.

The results of accurate fault simulation shows that a significant number of faults are actually detectable with the simulated test set. The fault coverage increases by up to 14.2 percent points. This highest increase is achieved when 10% of the inputs are X-sources. The figure also shows the result for three approximate fault simulation algorithms: The hybrid SAT-based method of [Kocht12], the BDD-based algorithm of [Kocht11a] (with a node limit of 50), and fault simulation based on restricted symbolic simulation (RSS). These algorithms are able to increase fault coverage remarkably, but in general, they fail to compute the accurate result.

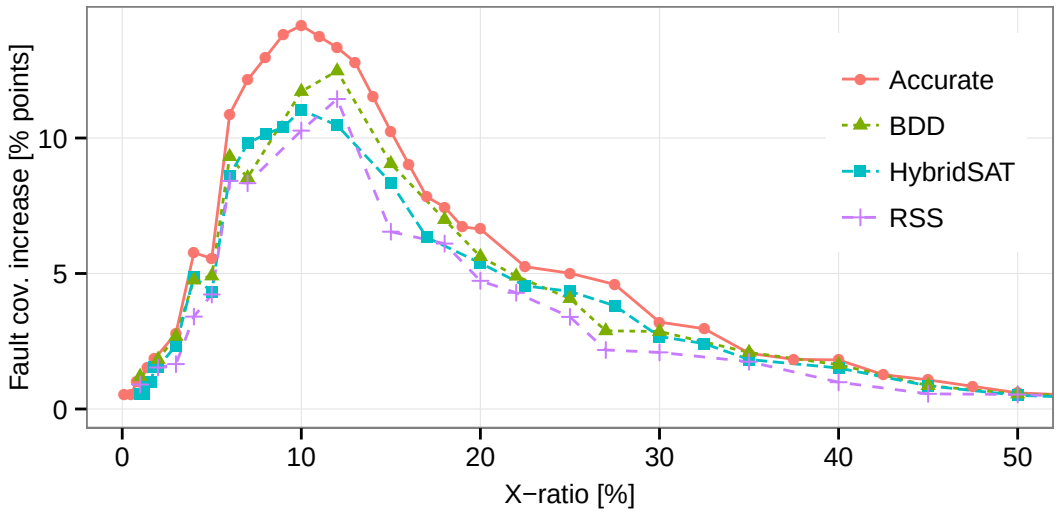


Figure 6.12: Increase in stuck-at fault coverage by accurate, approximate BDD-based [Kocht11a], hybrid SAT-based [Kocht12], and restricted symbolic (RSS) fault simulation for circuit c7552.

The runtime of the accurate algorithm reaches the maximum of 45s for an X-ratio of about 35%. Compared to [Kocht12] with a runtime of 2358s, the proposed algorithm is 52.4× faster because of its use of heuristics: For small X-ratios, the runtime is low since restricted symbolic simulation uncovers many nodes with binary value at simple X-reconvergences. If the SAT solver is required, the size of the constructed SAT instance is small. For high X-ratios, the pattern parallel simulation of randomized X-source assignments determines most of the X-valued nodes.

Application to Larger Circuits For larger circuits, the increase of fault coverage of the discussed algorithms is compared for an X-ratio of 5%. Fault simulation is

performed using restricted symbolic simulation (RSS), the approximate BDD-based method [Kocht11a] (with a node limit of 50), and the accurate SAT-based algorithm [Erb14a]. Since the approximate algorithm of [Kocht12] exhibits much higher runtime than the accurate algorithm, it is excluded in the following. Results in [Kocht12] show that its accuracy varies widely depending on the circuit.

Figure 6.13 shows the increase in fault coverage over three-valued fault simulation in percent points. For restricted symbolic simulation, the increase in fault coverage is between 0.31 and 7.46 percent points, with an average of 2.32 percent points. The BDD-based approach is able to classify slightly more faults as detected. In average, fault coverage rises by 2.64 percent points over three-valued simulation, but the BDD-based approach does not reach the accurate fault coverage.

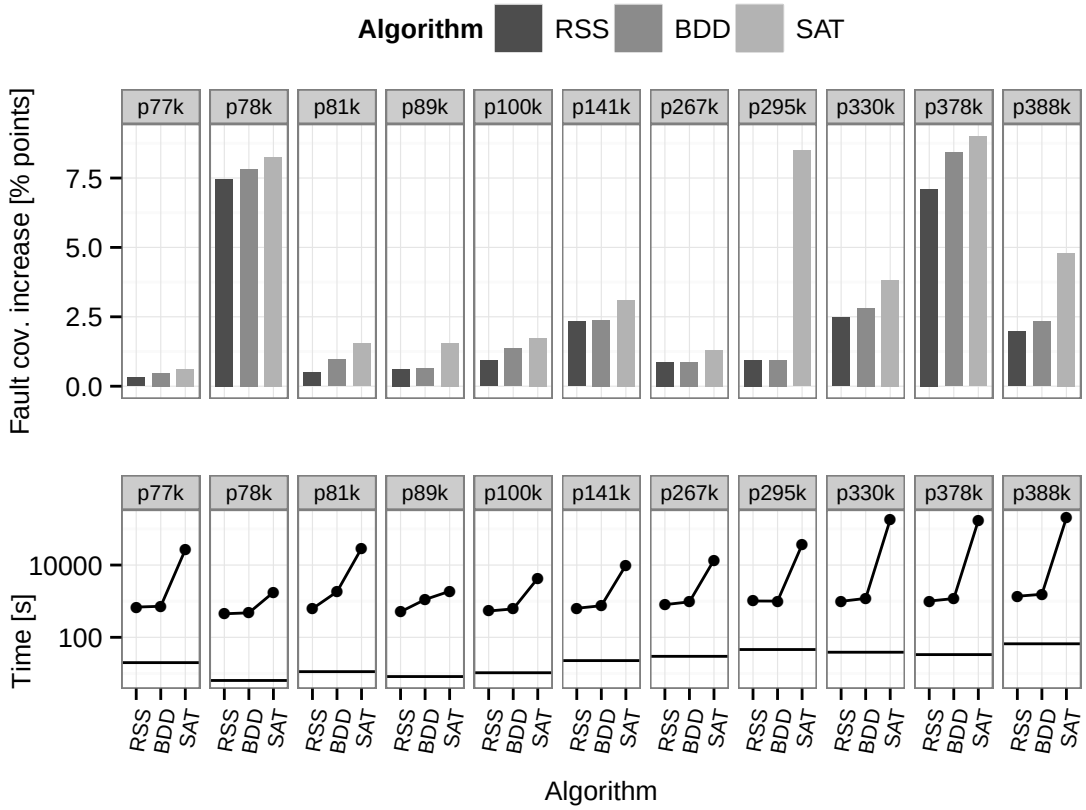


Figure 6.13: Increase in stuck-at fault coverage by accurate SAT-based, approximate BDD-based [Kocht11a], and restricted symbolic (RSS) fault simulation for an X-ratio of 5%.

The fault coverage increase of the accurate algorithm ranges from 0.62 to 9.00 percent

points compared to three-valued fault simulation. On average, the increase is 4.02 percent points.

The lower part of the figure shows the runtime of the algorithms in seconds. As reference, the runtime of three-valued fault simulation using pattern-parallel single fault propagation is shown as horizontal bar. Since the approximate and accurate algorithms do not process patterns in parallel, runtime is significantly higher: In restricted symbolic simulation, the runtime increases by 29.5× on average. The runtime of the BDD-based algorithm is on average 32.1% higher than the runtime of restricted symbolic simulation. The runtime of the accurate algorithm is on average 59.5× higher than the BDD-based approximate simulation. For larger circuits, the runtime of the exceeds 40 hours, dominated by proving that faults are neither definitely nor potentially detected by a pattern.

If X-sources are clustered, the fault coverage increase may even be higher than for randomly selected X-sources. The experiments in [Erb14a] select the X-sources as consecutive part of a scan chain. For circuit p100k and an X-ratio of 5%, fault coverage increases by 2.11 percent points over three-valued simulation on average. This is an increase of 21.3% compared to randomly selected X-sources.

Pessimism of Fault Simulation using Indirect Implications The pessimism of the enhanced fault simulation algorithm of [Kajih04] has been evaluated in [Kocht12]. For a set of random input patterns, the experiment computes the number of faults additionally classified as detected over three-valued fault simulation. For the ISCAS'85 and ISCAS'89 circuits evaluated in [Kajih04], this number is compared with the result of the accurate algorithm.

For all circuits, accurate fault simulation detects a much higher number of faults compared to the algorithm using indirect implications (cf. Table B.5 in the Appendix). For circuit s38417, the average number of additionally detected faults is $7.9\times$ higher.

6.6.2 Pessimism in Transition Delay Fault Simulation

Based on time frame expansion, an accurate transition delay fault simulation has been implemented and used to assess the pessimism in conventional transition delay fault simulation. The achieved increase in fault coverage over three-valued fault simulation

for an X-ratio of 5% is shown in Figure 6.14. On average, an additional 6.48% of the faults are found to be definitely detected by accurate fault simulation. The maximum increase in coverage of 12.8 percent points is computed for circuit p378k. For this circuit, coverage rises from 77.2% (three-valued) to 90.0%.

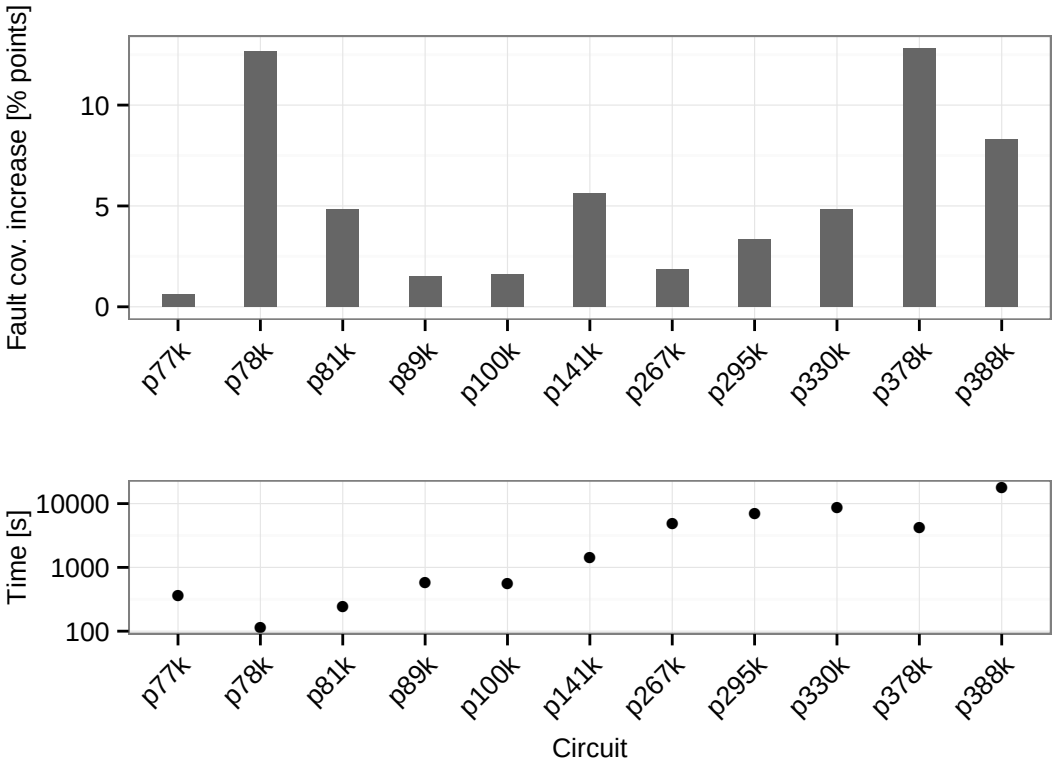


Figure 6.14: Increase in transition delay fault coverage by accurate fault simulation and runtime of accurate fault simulation for an X-ratio of 5%.

The lower part of the figure shows the runtime of accurate fault simulation in seconds. Although two time frames are considered in the expanded circuit model, the runtime is notably smaller than for accurate stuck-at fault simulation. This is because for many faults the complex output classification (cf. Figure 6.4) is skipped if the fault is not activated in the first time frame. On average, the runtime is only 7× higher than three-valued fault simulation.

6.7 Summary

The pessimism in fault simulation based on n-valued logic algebras can be reduced by extending the approximate and accurate reasoning techniques discussed for logic

simulation in Chapter 5. Here, restricted symbolic simulation, SAT-based hybrid simulation, and the local BDD-based approximate simulation technique are applied to stuck-at fault simulation [Kocht11b, Kocht11a, Kocht12]. The results show that a large fraction of faults can be classified as detected by a given test set if the simulation accuracy increases. The observed maximum increase of the BDD-based method exceeds 8.4% of the faults for circuit p378k.

The accurate solution is computed by a mapping to Boolean satisfiability [Hille12]. The maximum increase in fault coverage over the BDD-based algorithm is more than 8× (2×) for circuit p295k (p388k). Although the accurate fault simulation has considerably higher runtime than the approximate methods, it can be still applied to larger industrial circuits and offers a significantly higher coverage at no additional hardware cost or test time.

Similar results have been obtained for accurate transition delay fault simulation [Erb14a] with a maximum increase in fault coverage of over 12 percent points. Here, runtime is much smaller than for stuck-at fault simulation since for not activated faults, the expensive propagation analysis to outputs is not necessary.

7 Test Pattern Generation in Presence of Unknown Values

Automatic Test Pattern Generation (ATPG) is the algorithmic search for input stimuli (test patterns) that test for the presence of a targeted fault in a circuit, or proving that such stimuli do not exist. In the latter case, the fault is untestable.

Algorithms and effective heuristics developed for ATPG have proven to be very useful in other application fields like circuit optimization and redundancy identification, equivalence checking and verification, diagnosis or robustness analysis. Despite their importance, state-of-the-art test generation algorithms are pessimistic in presence of X-values since they are based on n-valued logic algebras. As far as test generation is concerned, the algorithms may fail to generate a test pattern for a testable fault if fault activation or propagation is affected by X-values.

This chapter starts with the discussion of the algorithmic problem of ATPG and a brief overview of conventional algorithms and their limitations. Then, a BDD-based accurate ATPG algorithm (Section 7.3) and a novel QBF-based accurate ATPG algorithm [Hille13] (Section 7.4) targeting stuck-at faults in circuits with X-values are presented. The extension to accurate multi-cycle test generation [Erb13] increases fault coverage even further (Section 7.4.7). In addition, approximate ATPG algorithms based on limited symbolic reasoning are discussed in Section 7.5. However, these algorithms cannot provide the accurate result or prove fault untestability. The chapter closes with results from experimental evaluations of accurate and pessimistic n-valued reasoning in test pattern generation, and a brief summary.

7.1 Problem Statement

ATPG searches for an input pattern that satisfies one or multiple fault activation and fault propagation conditions. Fault activation causes a fault effect locally at the fault site. Fault propagation ensures that the evoked fault effect is propagated to an observable circuit node for fault detection.

7.1.1 Fault Detection

The conditions for definite (DD) and potential detection (PD) of a fault under a *given* test pattern in presence of X-values are stated in Section 6.2. A fault f is called testable or detectable if a test pattern p (or vector $\vec{p}$) exists that detects f: $\exists\, p\, :\, \text{DD}^f(p)$. Fault f is potentially detectable if there is a pattern p which potentially detects f: $\exists\, p\, :\, \text{PD}^f(p)$. Finally, a fault is untestable with regard to the detection conditions in Section 6.2 if there is no pattern that definitely detects the fault: $\forall\, p\, :\, \neg\text{DD}^f(p)$.

7.1.2 Computational Complexity

In circuits without X-sources, both fault activation and fault propagation are Boolean constraints for stuck-at or transition delay faults.[1] The search for an input stimulus satisfying these constraints is an NP-complete problem for combinational circuits and stuck-at or transition delay faults. This has been proven for instance in [Ibarr75] and [Fujiw82].

Deterministic test pattern generation for sequential circuits is an NP-hard problem. Test pattern generation for combinational circuits can be directly mapped to sequential ATPG, constraining the solution to a single cycle. Yet, the verification of a found solution, i.e. test pattern sequence or vector, is in general not possible in polynomial time. Consider an n-bit counter with overflow as example of a sequential circuit. To test for the stuck-at-0 fault at the overflow signal starting from the 0-state, 2^n cycles are required. Fault simulation of the 2^n cycles requires exponential runtime in the number of sequential elements.

[1] Complex fault models may require different types of constraints, e.g. pseudo-Boolean constraints for open interconnect faults or power-droop test generation [Erb14b, Czutr12].

In presence of X-values, accurate ATPG for combinational and sequential circuits are both NP-hard problems. The NP-complete problem of combinational ATPG without consideration of X-values is a special case of ATPG in presence of X-values and can be directly reduced to it. Combinational ATPG in presence of X-values is not NP-complete since the verification of a computed test pattern needs to consider all 2^k different assignments to the k X-sources in the circuit. In Section 6.2.3 it was already shown that accurate fault simulation in presence of X-values is an NP-complete problem. Section 7.4.5 shows that the problem is a PSPACE problem belonging to the $\sum_3^P$ complexity class in the polynomial time hierarchy [Garey79].

Accurate sequential ATPG is an NP-hard problem for the same reason. In addition, the generated test pattern sequence may be of exponential size in the number of memory elements.

7.2 Conventional ATPG Algorithms and Their Limitations

ATPG systems typically combine ATPG algorithms with fault simulation of generated patterns. The ATPG algorithm is then invoked only for faults that are hard-to-test or untestable. Figure 7.1 shows the iterative application of an ATPG algorithm and its interaction with fault simulation.

Conventional deterministic ATPG algorithms map the search for a test pattern for a given fault to a search in a circuit graph (termed topological ATPG), to symbolic computation, or to Boolean satisfiability (SAT). Especially for sequential test generation, non-deterministic test generation based on genetic algorithms have been developed to cope with the higher problem complexity.

As discussed in Section 3.2.3, at least four different values are required to distinguish the possible signal states in the fault-free and faulty circuits. Typically they are denoted as $\{0, 1, D, \neg D\}$ or $\{0/0, 1/1, 1/0, 0/1\}$.[2] A four-valued Boolean algebra is defined over this carrier set to reason about the circuit behavior in presence of a fault and to derive a test pattern.

To model X-values in the fault-free and faulty circuits, n-valued logics with different accuracy have been introduced. The five-valued logic for test generation of [Roth66]

[2] Explicitly showing the value in the fault-free/faulty circuit.

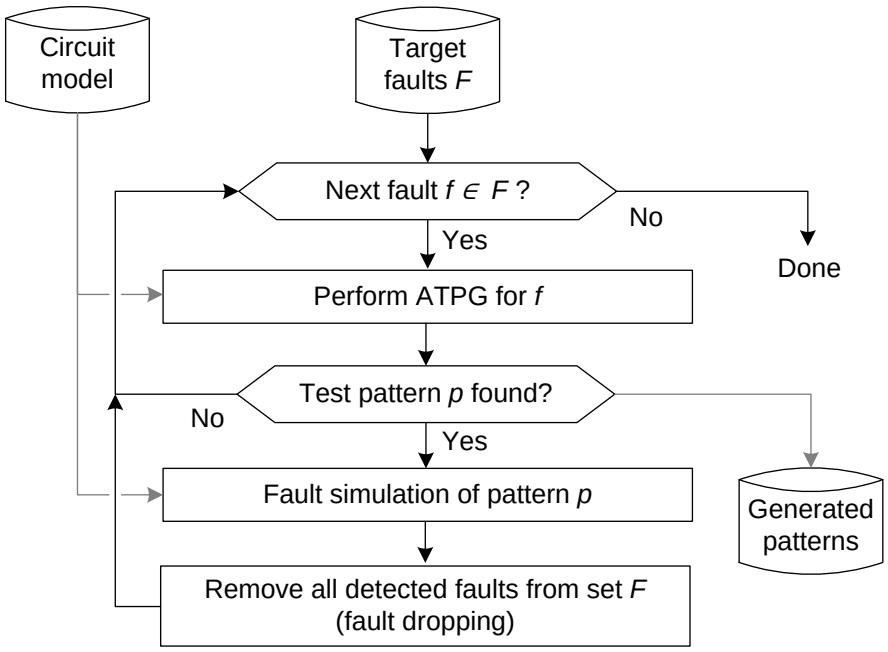


Figure 7.1: General ATPG flow.

adds a single X-symbol to the four values $\{0, 1, D, \neg D\}$ to denote the state of a node with an unknown value in the fault-free circuit, the faulty circuit, or both. It has been extended to a nine-valued logic [Muth76] to distinguish among these three cases. This increases the accuracy of test generation in sequential circuits. To generate tests for circuits with tristate logic, the symbol set can be extended by a symbol to represent the high impedance (floating) state. This additional symbol does not alter the accuracy of test generation in presence of X-values and will not be considered it in the following. Multi-valued signal modeling has also been introduced to SAT-based verification and test generation algorithms [Flore98, Jain00, Tille10].

The logic algebras defined over these symbol sets are typically pseudo-Boolean algebras. ATPG algorithms based on these algebras are pessimistic and incomplete in presence of X-values, i.e. in general they cannot guarantee to find a test pattern for a testable fault, or prove fault untestability if X-sources impact fault activation or propagation.

7.2.1 Combinational ATPG

The research on deterministic test generation algorithms for combinational circuits started with the graph-based or topological search for a test pattern, guided by the netlist. While test pattern generation based on binary decision diagrams was also investigated, the tremendous improvement in solving strategies for Boolean satisfiability problems shifted the focus to the mapping of test generation to the Boolean satisfiability problem. Today, Boolean satisfiability-based test generation offers high flexibility and robustness, i.e. with only few aborted faults.

Topological Test Pattern Generation

Topological test pattern generation is the deterministic search for an input stimulus based on the circuit graph or netlist so that a targeted fault is detected. The search starts at the fault site and traverses the circuit graph along internal nodes in forward and backward directions to the inputs and outputs, propagating the fault effect and deciding on required node assignments.

The first systematic and complete topological test generation algorithm *in absence of X-sources* is the D-algorithm [Roth66] based on Roth's five-valued D-calculus (cf. Section 3.2.3). During the search for a test pattern, the D-algorithm maintains and dynamically updates two sets of nodes which store the state of the search:

- The *D-frontier* contains the circuit nodes to which the fault effect D or $\neg D$ has been propagated to, and from which propagation can be continued towards the outputs by additional (sensitizing) internal node assignments. These are gates which have a fault effect at an input, but the output value is not yet implied by the current input values.
- The *J-frontier* contains all internal nodes that have been assigned a value during the search which is not yet justified, i.e. the assigned value is not yet implied by the values of the circuit inputs.

The D-algorithm aims to push the D-frontier towards a circuit output by selecting a node from the D-frontier and continuing the propagation so that the fault effect becomes observable. Secondly, the D-algorithm aims to push the J-frontier backwards

to the circuit inputs so that all internal node assignments are justified, i.e. implied by assignments to the circuit inputs (cf. Figure 7.2).

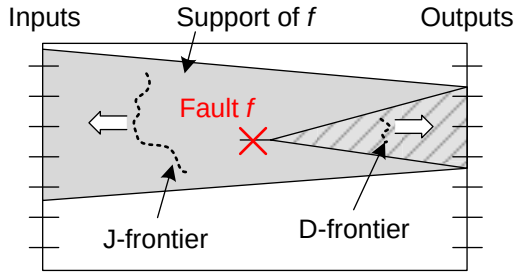


Figure 7.2: Justification (J) and propagation (D) frontier during test generation.

During the search, the D-algorithm determines how to propagate the fault effect and how to justify internal node assignments. The consequences of assignments to a node v are propagated in the circuit via forward implications to successors of v, and via backward implications to predecessors of v. Also, the J-frontier is updated according to the resulting state.

All nodes in the J-frontier must be justified. For each node, the D-algorithm decides how the node value is justified by locally selecting an assignment to the node inputs such that the node value is implied.

Conflicting node values may result from propagation or justification decisions. If either in the fault-free or faulty circuit complementary binary values are required at a node, the conflict is handled by backtracking: The last decision is revoked and an untried alternative is taken. If all alternatives result in conflicts, the fault is untestable.

The path-oriented decision making (PODEM) algorithm for test generation [Goel81] and the FAN algorithm (fanout-oriented test generation, [Fujiw83]) increase the efficiency of the D-algorithm by restricting the explored search space to the circuit inputs and so-called headlines, i.e. the leaves or output nodes of fanout-free regions in the circuit. Furthermore, controllability and observability metrics are used to guide the test pattern search at decision points.

The algorithms above only exploit direct forward or backward implications during the search process. This limits the consequences derived from node or input assignments and may defer the detection of resulting conflicts. Indirect implications, introduced in Section 5.3.2, are Boolean invariants in the circuit, which are not found by direct implications. Circuit analysis by static or recursive learning allows to discover indirect

implications [Schul88, Kunz97]. The circuit graph can then be extended with these implications to detect conflicts earlier.

BDD-Based Test Pattern Generation

Binary decision diagrams (BDD) have also been used in test generation. The BDD-based test generation algorithm CATAPULT [Gaede88] computes the controllability of each node and the observability of each node as BDDs. A test for a fault exists if the conjunction of the two BDDs, restricted by the required fault activation condition, is satisfiable. This algorithm proved efficient for faults for which topological test generation is difficult. TSUNAMI [Stani91] searches for the set of input stimuli that propagates the fault effect by a path-oriented traversal of the fanout cone of the fault location. If the fault effect cannot be propagated along a node v on a path to the circuit outputs, i.e. complementary values cannot be justified at v by input assignments, another path must be chosen. The computation of Boolean differences is mapped to algebraic computations using BDDs.

BDD-based test generation suffers from the limitations of BDDs: For certain sensitive structures with a high fanout, such as multipliers, an exponential amount of memory is required to store the BDD. Also, in many practical applications, it is not required to symbolically compute the complete test set for one targeted fault. A single test pattern is typically sufficient. This task is more robustly solved by mapping test generation to Boolean satisfiability.

Boolean Satisfiability-Based Test Pattern Generation

The principal idea of test generation based on Boolean satisfiability (SAT) is to map the search for a test pattern to the decision of satisfiability of a Boolean formula. The transformation of the test generation problem into a SAT instance, i.e. a Boolean formula in conjunctive normal form (CNF), is summarized below. The satisfiability of the formula can be decided using a SAT solver (cf. Section 4.3.1).

Instance Generation In SAT-based test generation, the Boolean functions of the fault-free circuit Ψ and the circuit with fault f, Ψ^f, are represented as characteristic functions in conjunctive normal form (CNF), here referred to as $C_{\text{fault-free}}$ and C_f, respec-

tively. The CNF formula of the nodes is obtained from the netlist or circuit graph by applying the Tseitin transformation [Tseit68].

It is sufficient to model only the support of the fault site in the fault-free circuit and the fanout cone of the fault site in the faulty circuit, as shown in Figure 7.3. The state of the other circuit nodes does not affect fault detection.

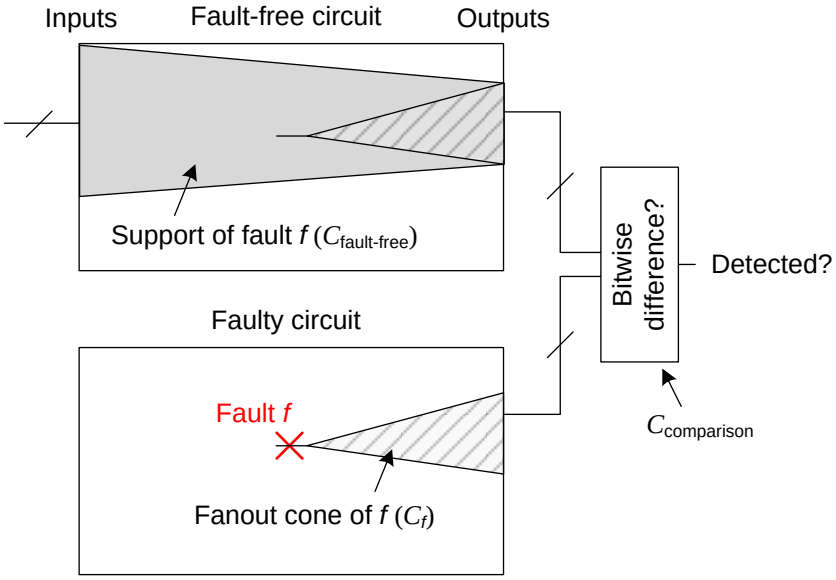


Figure 7.3: SAT instance generation with clauses for the fault-free circuit $C_{\text{fault-free}}$, faulty circuit C_f and output comparison $C_{\text{comparison}}$.

In a two-valued encoding, the state of each node is modeled by a single binary variable. For the nodes in the fanout cone of f in the faulty circuit, different literals are used so that different node values in the fault-free and faulty circuits can be expressed. At the boundary of the fanout cone of f, the literals of the nodes that are not reachable from the fault site are shared with the fault-free circuit because their values in the fault-free and faulty circuits are identical.

The test generation instance T_f is extended by additional clauses to compare the values of the circuit outputs reachable from the fault site: $C_{\text{comparison}} = \bigvee_{o \in O}(v_o \neq v_o^f)$. The resulting instance T_f enforces that there is at least one output where complementary values are observed:

$$T_f = C_{\text{fault-free}} \wedge C_f \wedge C_{\text{comparison}}.$$

If T_f is satisfiable, a test for fault f exists, and the test pattern can be extracted from the satisfying variable assignment provided by the SAT solver.

Modeling of Propagation Paths The search for a test pattern is sped up by extending T_f with topological information to guide the SAT solver in the search for a propagation path. This extension is termed D-chain [Larra92] since it directs the solver to establish a contiguous sensitized propagation path from the fault site to an observable node. For all nodes on the path, different values in the fault-free and faulty circuits, i.e. D or $\neg D$, are enforced by additional clauses.

The D-chain is constructed with help of difference literals d for nodes in the fanout cone of the fault. For node v, the clauses to express $d_v \rightarrow (v \neq v^f)$ are added so that node v is forced to have complementary binary values in the fault-free and faulty circuits if d_v is asserted. The D-chain now forces the solver to propagate a value difference at a node to at least one of its successors. For a node v with one successor $succ(v)$, $d_v \rightarrow d_{succ(v)}$. For a node v with multiple successors, $d_v \rightarrow \bigvee_{u \in succ(v)} d_u$. Furthermore, at the fault site f, d_f is asserted.

Incremental Construction To reduce the average construction time of the SAT instance and the average time required to analyze a fault [Tille08], the SAT instance is constructed and analyzed incrementally. Typically, the incremental analysis starts with fault activation and continues with fault effect propagation to outputs only if the fault can be activated.

During the incremental analysis of fault propagation, the outputs reachable from the fault site are added to the instance and analyzed one at a time. Thus, the D-chains in the fanout cone of the fault must also be extended incrementally. In contrast to the forward directed D-chain used in [Larra92], a *backward* direction of the D-chain is more suitable here. This backward D-chain starts from a target output and leads to the fault site. For output o, the difference literal is set to $d_o \leftrightarrow \neg(o \leftrightarrow o^f)$. For a node v with a difference, the D-chain forces a difference at at least one of its predecessors $pred(v)$, i.e. $d_v \rightarrow \bigvee_{u \in pred(v)} d_u$. After adding an output o and the backward D-chain originating at o to the SAT instance, d_o is asserted, and the difference literals of previously added outputs are deasserted.

Hybrid Multi-Valued Node Encoding To process circuits with X-sources or tristate logic, the two-valued node encoding in the fault-free and faulty circuits is extended to a three- or four-valued encoding for both the fault-free and faulty circuits [Jain00].

Optimized instance generation for such multi-valued logics was investigated in [Shi05, Fey06].

The three-valued encoding of nodes typically requires more than twice the amount of clauses than two-valued encoding. To minimize the additionally required variables and clauses, the three-valued modeling should only be applied for X-dependent nodes, i.e. nodes to which X-values can propagate to. This is achieved by a hybrid SAT instance, which combines the two-valued and three-valued parts of the circuit into a single SAT instance [Erb13], shown in Figure 7.4.

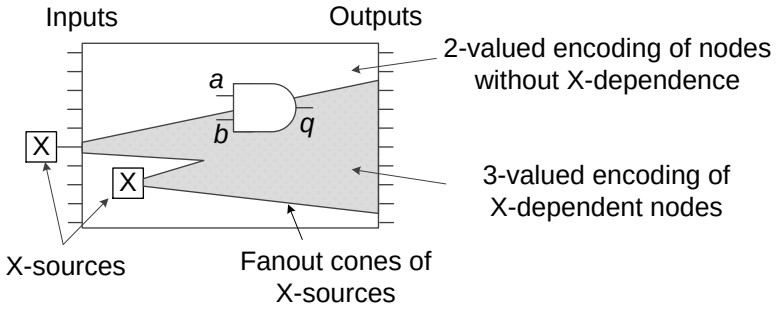


Figure 7.4: Hybrid two- and three-valued SAT instance.

For a two-input gate at the boundary between the two-valued and three-valued domain (Figure 7.4), one input (a) can only carry a binary value, while the other input (b) as well as the output (q) can carry both binary and X^P-values. Table 7.1 shows the resulting truth table for a two-input AND gate at the boundary. For input b and output q, the three possible values $\{0, 1, \mathrm{X}^P\}$ are encoded by two variables b_v, b_x and q_v, q_x.

Table 7.1: Truth table of a two-input AND gate at the boundary of the two- and three-valued region.

val(a) / a_v	val(b)	b_v	b_x	val(q)	q_v	q_x
0	0	0	0	0	0	0
	1	1	0	0	0	0
	X^P	0	1	0	0	0
1	0	0	0	0	0	0
	1	1	0	1	1	0
	X^P	0	1	X^P	0	1

The clauses of the resulting characteristic function of this AND gate are:

$$\underbrace{(a_v \vee \neg q_v) \wedge (a_v \vee \neg q_x)}_{\text{Output } q = 0 \text{ if input } a = 0.} \wedge$$

$$\underbrace{(\neg a_v \vee b_v \vee \neg q_v) \wedge (\neg a_v \vee \neg b_v \vee q_v) \wedge (\neg a_v \vee b_x \vee \neg q_x) \wedge (\neg a_v \vee \neg b_x \vee q_x)}_{\text{Output } q \text{ equals input } b \text{ if input } a = 1.}.$$

7.2.2 Sequential ATPG

A test for a fault in a sequential circuit consists of a sequence of test patterns applied in consecutive cycles such that the target fault is activated and propagated to an observable output. Test generation for sequential circuits is required if the sequential elements in a circuit are not directly controllable or observable.[3]

For sequential circuits without feedback between the sequential elements, such as pipelines, the maximum number of cycles of a test is bounded by the maximum sequential depth of the sequential elements. This bound is the depth of the acyclic S-graph of the circuit. For cyclic circuits, the worst-case upper bound for the test length is exponential in the number of sequential elements [Bushn00].

Two principal methods are used for sequential test generation: Deterministic test pattern generation using time frame expansion and (fault) simulation-based algorithms:

Time Frame Expansion transforms the sequential circuit model into a combinational equivalent that considers k consecutive cycles of operation (cf. Figure 5.17). A test generation algorithm for combinational logic then searches for a test for a target fault in the expanded model. Typically, test generation starts with fault activation in the initial cycle and propagation to an observable output. Then, it proceeds backwards to justify internal node values until an allowed initial state is reached [Cheng89].

Simulation-based Test Generation is a non-deterministic search for a test pattern sequence. Using a set of given or pseudo-randomly generated test patterns, sequential (fault) simulation is used to select those patterns or pattern (sub-)sequences which are most effective in detecting targeted faults in the sequential circuit. The most successful approaches employ genetic algorithms [Goldb89] using fitness-based individual selection, mutation, and crossover operations on the candidates. Compared to deterministic

[3] Sequential circuits can be instrumented with dedicated design-for-test hardware to increase controllability and observability of sequential elements (cf. Section 8.1).

algorithms, test generation based on genetic algorithms allows to process much larger circuits [Corno96], [Hsiao97]. Simulation-based test generation may fail for hard-to-test faults since the search space for test patterns is not systematically explored. In principle, fault untestability cannot be proven.

7.3 Accurate Test Pattern Generation Based on BDDs

In [Keim03], an accurate BDD-based test pattern generation method has been proposed. It constructs the fault detection function $D(\Psi, f)$ of a circuit for a target fault f as the difference of the fault-free circuit behavior Ψ and the faulty behavior Ψ^f:

$$D(\Psi, f) := \Psi(\vec{p}) \oplus \Psi^f(\vec{p}).$$

If $D(\Psi, f)$ is satisfiable, then there is a test pattern vector that tests for the presence of fault f. The fault detection function can be represented as an ROBDD. If the ROBDD contains the 1-terminal, f is testable and a pattern can be easily extracted from the ROBDD.

For accurate test pattern generation in presence of X-values, the X-sources S_X are represented as pseudo-input variables in the ROBDD of the fault detection function. This results in an ROBDD over $|I| + |S_X|$ variables. Once this ROBDD is constructed, all variables representing an X-source are universally quantified. For variable s_X, this corresponds to the following operation (using the BDD restrict and apply operation [Bryan86]):

$$\forall\, s_X \;:\; D(\Psi, f) := D(\Psi, f)[0/s_X] \wedge D(\Psi, f)[1/s_X].$$

While the restrict operation has a runtime complexity of $\mathcal{O}(n \log n)$, the apply operation required for the conjunction of the two restricted ROBDDs has quadratic runtime complexity in the size of its input ROBDDs: $\mathcal{O}(n^2)$. To obtain the fault detection ROBDD representing the test pattern set, all $|S_X|$ variables need to be universally expanded. This restricts the usage of this technique to small circuits with only few X-sources.

7.4 Accurate Test Pattern Generation Based on Formal QBF Reasoning

The first mapping of the test pattern generation problem in presence of X-values to the satisfiability of quantified Boolean formulas (QBF) has been proposed in [Hille13]. It overcomes the principal inaccuracy and pessimism of conventional algorithms when X-values are considered and can prove the untestability of all untestable faults. In contrast to the BDD-based approach of [Keim03], the method does not suffer from the principal shortcomings of BDDs and is applicable to large circuits under test.

7.4.1 Overview of the Accurate Combinational ATPG Framework

The proposed ATPG framework [Hille13, Erb13] combines accurate fault simulation (cf. Section 6.4), incremental SAT-based test generation with two- and three-valued node encoding, and QBF reasoning to efficiently analyze the faults. Using a topological analysis, the faults under analysis are partitioned into four groups depending on their relation to the X-sources in the circuit (cf. Figure 7.5):

Group 1: No structural dependence on the X-sources. Neither the fanin cone of the fault f, nor its fanout cone have any dependence on X-sources, i.e. the support of f does not contain any X-sources.

Group 2: A subset of the outputs in the fanout cone of the fault depends on X-sources. The fanin cone does not depend on X-sources.

Group 3: A subset of the inputs in the fanin cone of the fault depends on X-sources. In other words, at least one input in the fanin cone is a controllable input.

Group 4: The fanin cone is driven exclusively by X-sources.

The faults are processed as shown in Figure 7.6: First, faults in Group 1 are analyzed by the SAT-based ATPG based on a two-valued node encoding without consideration of X-values (①). Then, for faults of Group 2, a hybrid two- and three-valued node encoding (②) is used in the SAT-based ATPG: The fault activation is checked using two-valued encoding. If the fault cannot be activated, it is untestable. Otherwise, the SAT instance is incrementally extended to analyze fault propagation. Three-valued encoding is used for all X-dependent nodes. The handling of faults in Group 3 is

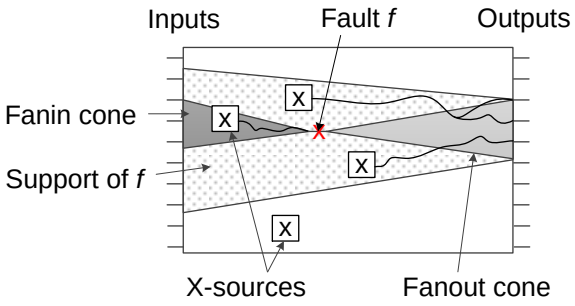


Figure 7.5: Fault f, its fanin and fanout cone, and its support.

similar. However, since the fault site is X-dependent, failed fault activation in the hybrid SAT instance does not imply fault untestability. For the faults of Group 2 and 3 for which no test pattern is found, 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.

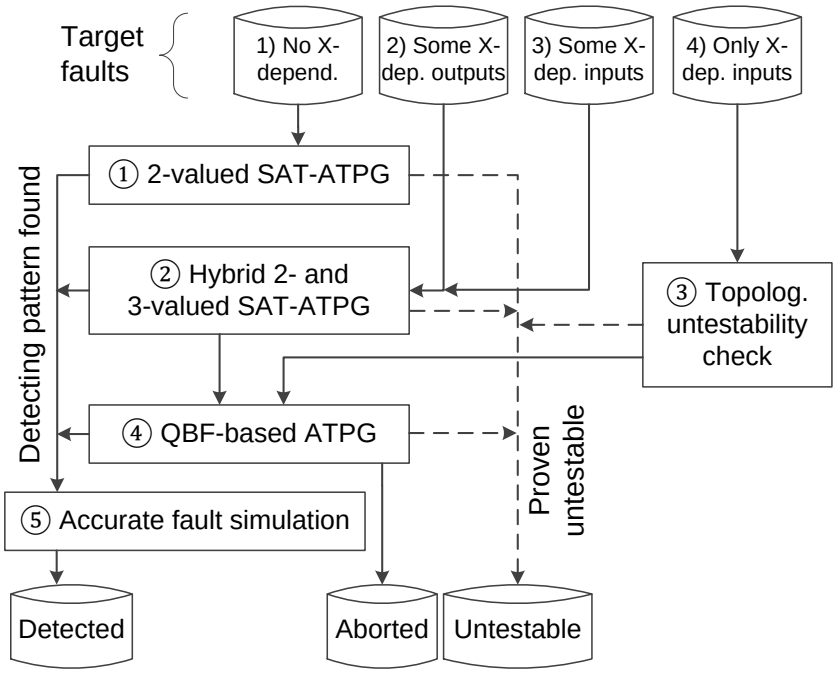


Figure 7.6: Overview of the proposed QBF-based ATPG framework.

7.4.2 Two-valued SAT-based ATPG

Faults in Group 1 have no X-dependency. Thus, two-valued or binary signal encoding is sufficient to generate a testing pattern or prove fault untestability. The SAT instance is constructed incrementally as described above in Section 7.2.1:

For each fault, first the justification cone is modeled and it is checked whether the fault can be activated. If the fault cannot be activated, it is untestable. If it can be activated, the fault propagation to the circuit outputs is analyzed.

The SAT instance is now incrementally extended to analyze the fault propagation to the outputs. Here, only outputs in the fanout cone of the fault site are considered. The outputs are processed with increasing structural depth. For an output, all gates in its fanin cone are added to the SAT instance of the fault-free and faulty circuits. For nodes in the fanout cone of the fault, the clauses for the D-chains are added as well.

Once the output, its fanin cone and the D-chain have been modeled, it is sufficient to check whether the output can detect a difference between the fault-free and faulty circuits under the considered fault. This is achieved by setting the difference literal d_o of the D-chain at output o to 1 and checking for satisfiability of the SAT instance. This condition is added as an 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 $d_o = 0$ 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 is marked untestable if the SAT instance is unsatisfiable for all outputs reachable from the fault site.

7.4.3 Hybrid Two- and Three-valued SAT-based ATPG

The testability of faults in Group 2 and Group 3 partially depends on the X-sources: For faults in Group 2, both fault activation and propagation to outputs without X-dependence can be analyzed accurately using a binary encoding. For faults in Group 3, only a subset of the nodes in the fanin cone of the fault site are X-dependent. To avoid the separate construction of SAT instances with two- and three-valued node encoding, a hybrid instance is constructed in which nodes without X-dependence are encoded

with two values. All other nodes are encoded using three values (cf. Hybrid Multi-Valued Node Encoding in Section 7.2.1).

This hybrid modeling is able to generate a test pattern for a testable fault if and only if fault activation and propagation do not require X-canceling at a reconvergence of X-valued nodes, i.e. blocking of X-values in the support of the fault is sufficient for definite detection.

Figure 7.7 shows the analysis of faults of Group 2 and 3. The instance construction and evaluation is performed incrementally by first considering the activation at the fault site (①), followed by the analysis of reachable outputs (②). If fault activation fails for a fault of Group 2, the fault is untestable. If fault propagation fails, untestability is not decided yet, and the fault needs to be analyzed by the accurate QBF-based ATPG (③). For faults in Group 3, untestability cannot be decided at all by this hybrid node encoding. If test generation for these faults fails, they are also processed by the accurate QBF-based ATPG (③).

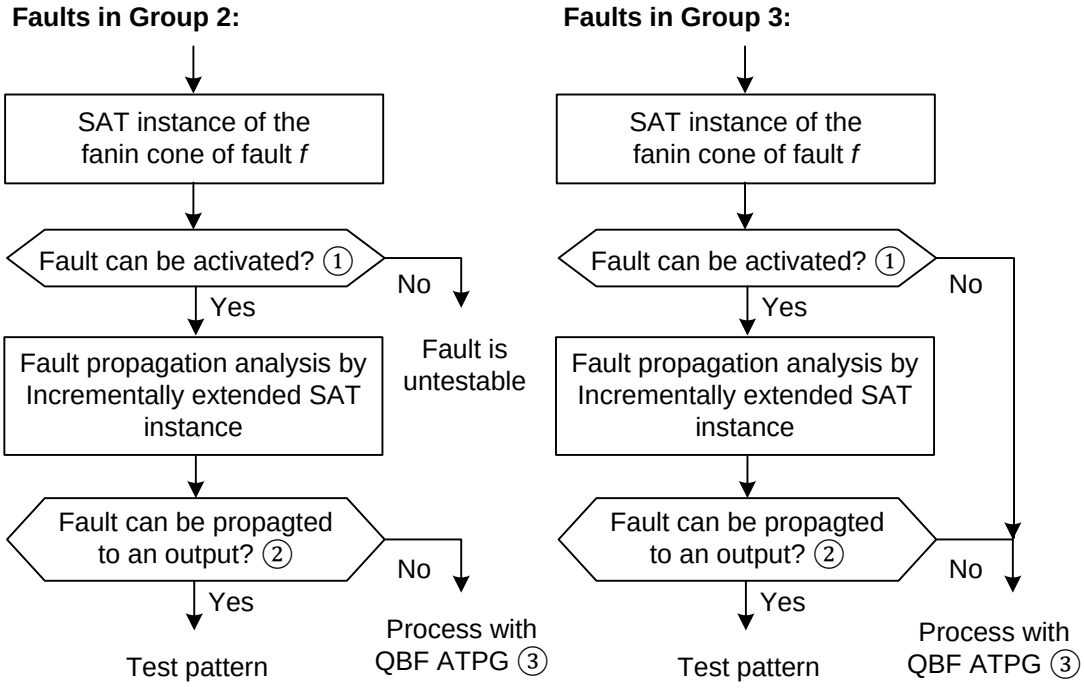


Figure 7.7: Processing of faults in Group 2 and 3 using the hybrid SAT instance.

7.4.4 Topological Untestability Check

If the fault site of a fault cannot be justified to a binary value, the fault cannot be definitely detected. For faults of Group 4, the fanin cone of the fault site only depends on X-sources. Thus, a binary value at the fault site can only result from X-canceling at a reconverge of X-valued nodes in the fanin cone. To check for reconvergences, a tracing of branching nodes in the fanin cone is performed. If no reconvergences are found, the fault site cannot carry a binary value, and the fault cannot be activated. Consequently, the fault is untestable w.r.t. the definite detection criterion of Sections 6.2 and 7.1.1. If reconvergences are found, the fault may be testable and is analyzed using the accurate QBF-based ATPG described below.

7.4.5 QBF-based ATPG

All faults for which testability or untestability has not been proven yet are subject to an accurate analysis based on quantified Boolean satisfiability (cf. Section 4.3.1). For each target fault, a problem instance is generated as quantified Boolean formula (QBF) by constructing the matrix and then quantifying the variables. A QBF solver is employed to search for a satisfying model (i.e. a test pattern) or to prove the untestability of the fault.

Construction of the CNF Matrix for ATPG

The matrix of the QBF for fault f is constructed similar to a conventional two-valued SAT-based ATPG instance. Two-valued node encoding is used, i.e. the value of a node is represented by a single binary variable.

The clauses of the gates in the fault-free circuit $C_{\text{fault-free}}$ and the faulty circuit C_f (respectively the fanout cone of f) are obtained by the Tseitin transformation. Fault propagation conditions from the fault site to the outputs are added in form of D-chains. The disjunction of the d-variables of outputs reachable from the fault site $C_{\text{comparison}} = \bigvee_{o \in O} d_o$ is added to the CNF instance to ensure that the fault effect is observable at at least one output:

$$\mathcal{C}^f_{CUT} := C_{\text{fault-free}} \wedge C_f \wedge C_{\text{D-chain}} \wedge C_{\text{comparison}}.$$

Variable Quantification

All controllable inputs I to the circuit are existentially quantified. All X-sources S_X are universally quantified. Here it is important to respect the scope of quantification, i.e. the order of quantifier alternations. In particular, we search for *one* test pattern that satisfies the formula $\mathcal{C}^f_{CUT}$ for *all* possible assignments to the X-sources. Thus, the universal quantification of the X-sources is in the scope of the existential quantification of the circuit inputs.

To bind all variables of the matrix, the internal nodes and outputs V, as well as the d-variables D of the D-chains are existentially quantified, too. This results in the following QBF:

$$\underbrace{\exists_{i \in I}\, i}_{\substack{\text{Controllable}\\ \text{inputs}}} \overbrace{\forall_{x \in S_X}\, x}^{\text{X-sources}} \underbrace{\exists_{v \in V}\, v\, \exists_{d \in D}\, d}_{\substack{\text{Int. nodes,}\\ \text{D-chain var.}}} \left(\mathcal{C}^f_{CUT}\right), \text{ or in short}$$

$$\exists\, I\, \forall\, S_X\, \exists\, V\, \exists\, D\, \left(\mathcal{C}^f_{CUT}\right).$$

This QBF is satisfiable if and only if there exists an input assignment which excites a 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 (6.2) of Section 6.2, the solution space is constrained by restricting the detecting outputs to a 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.

$$\underbrace{\exists \boldsymbol{O}}_{\substack{\text{Selection of one}\\ \text{detecting output}}} \exists I \forall S_X \exists V \exists D : \Big(\mathcal{C}^f_{CUT} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i)\Big)$$

This enforces a fixed detecting output over all assignments to X-sources. However, the observable difference, i.e. the node values in the fault-free and faulty circuits at that output, are still allowed to be one of the four possibilities $(0/1)$, $(1/0)$, $(\mathrm{X}_i, \neg\mathrm{X}_i)$, $(\neg\mathrm{X}_i, \mathrm{X}_i)$. The latter two cases correspond to situations where an output always shows complementary binary values in the fault-free and faulty circuits for all assignments to the X-sources, but the values in the fault-free and faulty circuits 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 val^0_i, val^1_i per output are used for a unary encoding of the stable value of a detecting output. If val^0_i (val^1_i) is true, output i has the stable value 0 (1) in the fault-free circuit. The two implications $(val^0_i \rightarrow \neg v_i)$ and $(val^1_i \rightarrow v_i)$ for output i establish that relation, assuming that v_i is the node variable representing the state of output i in the fault-free circuit.

With the implication $(o_i \rightarrow (val^0_i \vee val^1_i))$ for output i, and existential quantification of the variables $val^0_i, val^1_i \in V_{FF}$, we obtain the following QBF:

$$T^{DD}_f := \exists O \overbrace{\exists \boldsymbol{V_{FF}}}^{\substack{\text{Selection of stable}\\ \text{value in fault-free circuit}}} \exists I \forall S_X \exists V \exists D : \Big(\mathcal{C}^f_{CUT} \wedge \bigvee_i o_i \wedge \bigwedge_i (o_i \rightarrow d_i) \wedge \bigwedge_i \big((o_i \rightarrow (val^0_i \vee val^1_i)) \wedge (val^0_i \rightarrow \neg v_i) \wedge (val^1_i \rightarrow v_i)\big)\Big). \quad (7.1)$$

T^{DD}_f is satisfiable if and only if fault f is definitely detectable. The satisfying model provided by the QBF solver specifies the values of the inputs I and also points to the output at which the fault is definitely detected. If the formula is not satisfiable, there is no test pattern for definite detection of f.

The polynomial time hierarchy classifies NP-hard problems. A representative complete problem of each class in the polynomial hierarchy is given by QBF-SAT with different numbers of quantifier alterations in the prenex form [Garey79]. In the formula T^{DD}_f

above, the number of quantifier alternations is two. In the polynomial time hierarchy, this corresponds to the complexity class $\sum_3^P$.

Test Pattern Compaction

To limit the number of test patterns, it is possible to perform static compaction of the generated patterns by merging compatible patterns, since only the inputs in the support of the fault site have specified values. For faults with overlapping support regions, dynamic compaction [Goel79] is more effective as it considers multiple target faults *during* the search for a test pattern. In [Czutr09], dynamic compaction has been applied to conventional SAT-based ATPG by incrementally extending the SAT instance with additional target faults. The recent support for incremental construction and evaluation of QBFs [Marin12] may allow a similar method for accurate ATPG in presence of X-values.

7.4.6 Potential Detection of Faults

For some faults in the fanout cone of the X-sources, test patterns for definite detection do not exist. For these faults, it may be still worthwhile to analyze whether test patterns for potential detection exist. Test pattern generation for potential detection is also mapped to QBF satisfiability. According to Equation 6.3 in Section 6.2, fault f is potentially detected under a pattern if there is one fixed output which has a binary value in the fault-free circuit, and an X-value in the faulty circuit under f. The structure of the QBF instance for potential fault detection is shown in Figure 7.8.

In this instance, the constraints for the fault-free and the faulty circuits are different: For the fault-free circuit, there must be a fixed output $o^{PD} \in O$ with a binary value val (similar to T_f^{DD} above). But for the faulty circuit, it is sufficient that output o^{PD} carries the opposite value $\neg val$ just for one arbitrary X-source assignment. Thus, for these two parts different variable quantifications are required. Following the reasoning

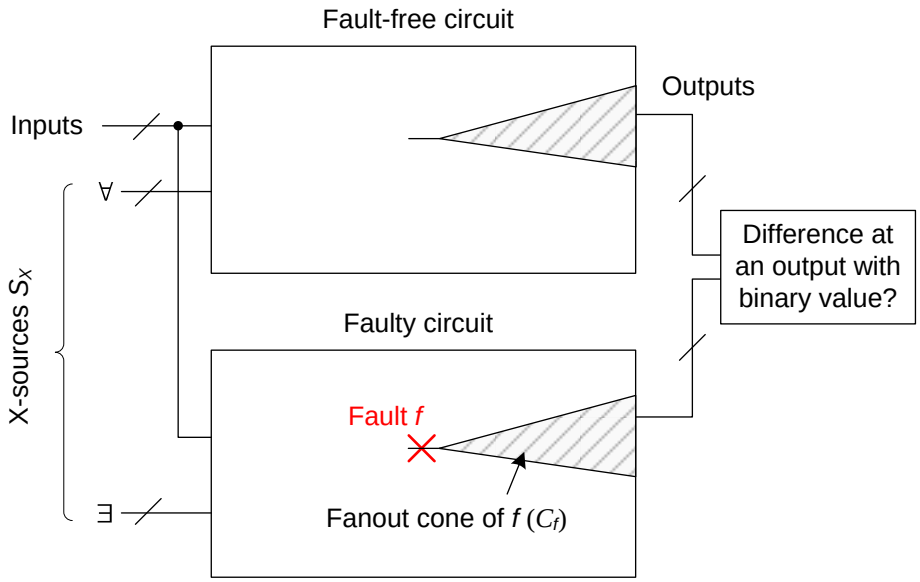


Figure 7.8: Principle of the QBF instance for potential detection test generation.

of the previous sections, this is represented by the following QBF:

$$T_f^{PD} := \exists O \exists V_{FF} \exists I \Big[\forall S_X \exists V : \Big(C_{\text{fault-free}} \wedge \bigvee_i o_i \wedge \bigwedge_i \big((o_i \rightarrow (val_i^0 \vee val_i^1)) \wedge (val_i^0 \rightarrow \neg v_i) \wedge (val_i^1 \rightarrow v_i) \big) \Big) \wedge \exists S_X \exists V : \Big(C_f \wedge \bigwedge_i \Big(o_i \rightarrow \big((val_i^0 \rightarrow v_i^f) \vee (val_i^1 \rightarrow \neg v_i^f) \big) \Big) \Big) \Big].$$

In the fault-free circuit representation, the variables of the X-sources are universally quantified so that an output with a binary value for all X-source assignments is searched for. In the faulty circuit representation, it is sufficient to quantify these variables existentially to detect the fault for a single X-source assignment. The variables $o_i \in O$ that specify the detecting output o^{PD} and the variables $val_i^0, val_i^1 \in V_{FF}$ that specify its value in the fault-free circuit are quantified at the highest scope level. This ensures that they are bound both in the clauses for the fault-free and faulty circuit to compute the value difference at the outputs. The value of output i in the faulty circuit is represented by variable v_i^f.

If T_f^{PD} is not satisfiable, then there is no output with binary value in the fault-free circuit at which fault f can be potentially detected. It may still be possible that the fault effect is propagated to outputs that carry an X-value in the fault-free circuit. Since

in a practical test application, at least the reference value in the fault-free circuit must be known, this is not investigated here.

7.4.7 QBF-based Accurate Sequential ATPG

In circuits where a subset or all of the sequential elements have an unknown state, a fault may require multiple test cycles to be detected [Pomer97, Pomer03] even though a single test cycle is sufficient for its detection if the inputs to the combinational part are controllable. The generation of such multi-cycle tests was proposed in [Pomer11] based on random patterns and fault simulation. This technique is limited because it generates unnecessarily long test sequences and fails to generate patterns for hard-to-test faults. Also, it cannot prove fault untestability.

Based on time frame expansion, three-valued sequential ATPG is able to generate a test pattern sequence for a fault if fault activation and propagation at most requires blocking of X-values to justify binary values in the considered cycles. In [Erb13], circuit structures are discussed that require *accurate reasoning* to find a test pattern sequence for certain faults. One example is given in Figure 7.9 where the controlled reconvergence in the second time frame and accurate reasoning is required to test the stuck-at-0 fault at node b.[4]

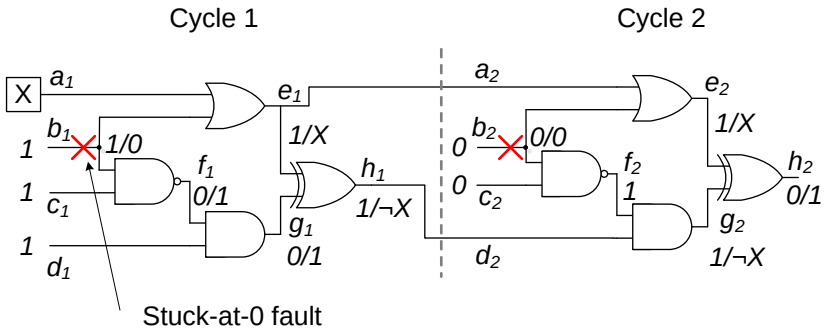


Figure 7.9: Detection of stuck-at-0 fault at node b requires the reconvergence of X-valued node a in the second cycle.

To deterministically generate tests for such cases, accurate reasoning must be employed in the expanded circuit model. Here, the accurate QBF-based test generation approach of Section 7.4.5 is extended to support accurate sequential test generation

[4]The sequential circuit has primary inputs b, c, and flip-flops connecting nodes e to a, and h to d.

for a fixed number k of considered cycles. Thus, the fault-free circuit is represented as the union of clauses of the characteristic functions of the nodes in the expanded circuit model:

$$ {}_kC_{\text{fault-free}} := \bigcup_{1 \leq i \leq k} \underbrace{\Big(\text{Node } v \in V_i\Big)}_{\substack{\text{Tseitin transformation} \\ \text{of the nodes in cycle } i}}. $$

Accurate multi-cycle test generation must consider the effects of the fault from previous cycles in the fault-free circuit model. Thus, the matrix of the constructed QBF contains the union of the fanout cones of the fault site in each considered frame since the fault effects in one time frame must be propagated properly to all subsequent time frames. This results in the following representation of the faulty circuit ${}_kC_f$:

$$ {}_kC_f := \bigcup_{1 \leq i \leq k} \underbrace{\Big\{\text{Node } v \in \text{Fanout}(f_i)\Big\}}_{\substack{\text{Tseitin transformation of the nodes} \\ \text{in the fanout cone of fault } f \text{ in cycle } i}}. $$

Note that Fanout(f_i) includes all nodes reachable from fault f_i in cycle i in the expanded model up to cycle k. Since it is possible that the effect of an activated fault in one time frame propagates through a faulty gate in a later time frame, the faulty gate instances in the k cycles are modeled such that fault effects of earlier time frames are considered. Furthermore, fault activation shall be allowed in any of the k cycles, and even multiple fault activation may be required to test a fault [Erb13]. This is reflected by a disjunction of the difference literals of the fault sites in the k cycles $\bigvee_{1 \leq i \leq k} d_{fi}$, which is part of the D-chain clauses. Note that this constraint does not require that f be activated in the same cycle for all assignments to the X-sources. It is possible that the fault is activated in different cycles depending on the X-source assignments.

Let $C_{\text{Output/value selection}}$ denote the clauses to enforce a fixed detecting output in the k-th cycle and its binary value in the fault-free circuit, as introduced in Equation (7.1) above. With this, the matrix of the QBF instance for fault f and k considered cycles is:

$$ {}_k\mathcal{C}^f_{CUT} := {}_kC_{\text{fault-free}} \wedge {}_kC_f \wedge C_{\text{D-chain}} \wedge C_{\text{Output/value selection}}. $$

The QBF instance ${}_kT_f$ below is satisfiable if and only if there is a test pattern sequence of length k that activates fault f and propagates its effect to at least one fixed output

$o \in O_k$ in the k-th cycle:

$$_kT_f := \exists\, O_k\, \exists\, V_{FF}\, \exists\, I\, \forall\, S_X\, \exists\, V\, \exists\, D : \left({}_k\mathcal{C}^f_{CUT} \right).$$

For each targeted fault, the QBF instances can be constructed with increasing number of considered cycles until a user-defined bound is reached. Each instance $_kT_f$ is evaluated by the QBF solver. Analogous to the case of combinational ATPG, $_kT_f$ can be modified to generate test patterns for potential detection.

7.5 Approximate Test Pattern Generation Algorithms

In the literature, approximate ATPG algorithms in presence of X-values have been proposed based on *limited* symbolic reasoning: Encoding the state of nodes with unknown values using an n-valued logic algebra, as done in conventional algorithms, introduces pessimism during forward and backward implication in test generation. To increase the accuracy, the application of restricted symbolic simulation and hybrid approaches based on genetic algorithms have been proposed.

Test Generation Based on Restricted Symbolic Logic

In [Kundu91], restricted symbolic logic (cf. Section 5.3.3) is applied in topological test generation to reduce the pessimism of forward implication. The result of this algorithm is still pessimistic for the following cases:

- ▷ The algorithm maintains a D-frontier to guide the fault effect propagation. However, in the D-frontier only nodes with a defined value $\in \{0, 1, D, \neg D\}$ are considered for propagation. Figure 7.10 a) shows an example where *un*restricted symbolic X-values are required to propagate the fault and to find a test pattern.
- ▷ Backward implication does not consider symbolic X-values. In consequence, node justification that requires X-canceling is not explored by the algorithm. Figure 7.10 b) shows an example where the fault effect cannot be propagated to node q since the required justification of value 1 at node c fails.

▷ If two or more different X-valued nodes converge, forward implication computes the resulting value as a new X-symbol without correlation to the input values (according to restricted symbolic logic, cf. Section 5.3.3).

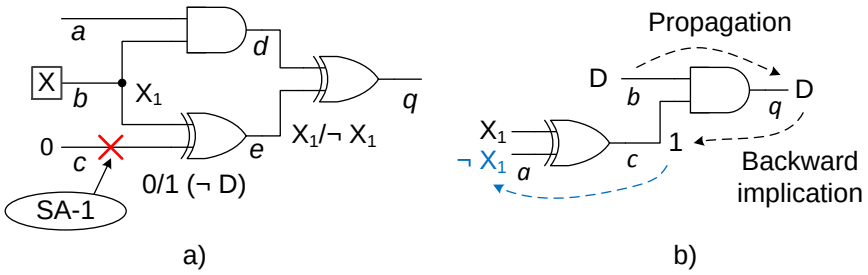


Figure 7.10: Limitations of test generation using restricted symbolic logic: a) Propagation stops prematurely at node e; b) Backward implication from node c to a with value $\neg X_1$ fails.

With these limitations, this test generation algorithm is incomplete: It cannot prove the untestability of faults in the support of X-valued nodes, and it is not able to find a test pattern for all testable faults.

Approximate Test Generation Based on Genetic Algorithms

The test pattern generation algorithm for sequential circuits proposed in [Keim03] combines BDD-based symbolic simulation and test generation techniques. It searches for short test pattern sequences with high fault coverage using a genetic algorithm [Goldb89]. In this algorithm, the pattern sequences are the individuals. Their fitness is derived from the number of faults detected by the sequence, and the sequence length. New test sequences are generated by a crossover operation that mixes the input assignments of two parent sequences that are selected by their fitness.

To compute the faults detected by a sequence, the hybrid sequential fault simulator of [Becke99] is used (Section 6.5.3). Once the fault coverage saturates, test sequences can be deterministically generated for hard-to-test faults, starting from the current state of the circuit. These sequences are computed using BDDs (cf. Section 7.3), but only for a small number of cycles and only when most sequential elements have already been initialized [Keim03].

7.6 Evaluation on Benchmark and Industrial Circuits

The experiments investigate the pessimism of conventional ATPG algorithms based on n-valued logic algebras and the increase in fault coverage achievable by accurate reasoning.

7.6.1 Experimental Setup

In the experiments, the larger ISCAS'85 and ISCAS'89 benchmark circuits, as well as industrial designs from NXP are used. It is assumed that a fixed and randomly selected subset of inputs generates X-values. Five different subsets of X-source inputs are generated per experiment. The reported results are the rounded average of the results of these five X-configurations per circuit.

For each circuit the collapsed set of stuck-at faults is computed. 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, three-valued ATPG and the proposed accurate combinational or sequential ATPG algorithms are performed.

Appendix B.5 provides the tabulated results for reference.

7.6.2 Accurate Combinational ATPG

For combinational ATPG, the combinational parts of the circuits are considered. X-sources are selected from the inputs of this combinational part.

Fault Coverage Increase for Varying X-Ratios At first, the increase in fault coverage compared to three-valued ATPG is evaluated for varying X-ratios based on two circuits, c7552 and s13207. Figure 7.11 shows the increase in the number of definitely detectable faults (Def. Detect Inc.) for the two circuits and X-ratios from 1 to 99%. The figure also shows the absolute number of potentially detected faults found by accurate fault simulation (Pot. Det. Fsim), and the number of potentially detectable faults identified by QBF-based ATPG (Pot. Det. ATPG). Finally, the figure shows the number of faults aborted by the accurate ATPG after a timeout of 10 seconds.

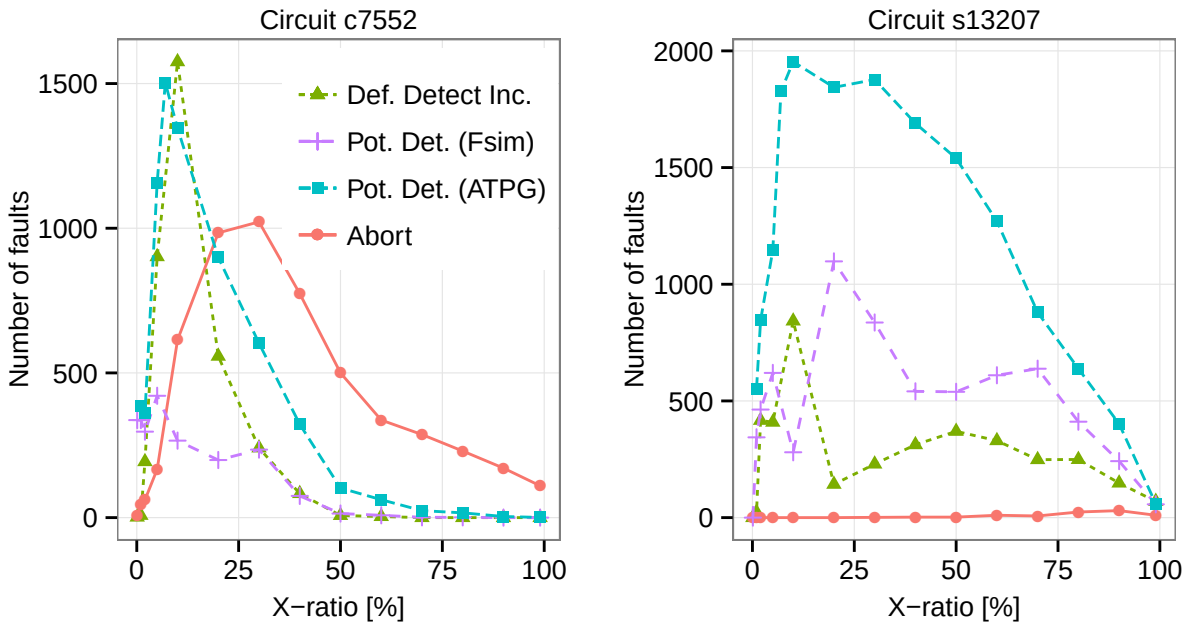


Figure 7.11: Increase in definitely detectable faults (Def. Detect Inc.) compared to three-valued ATPG depending on the X-ratio [Hille13], absolute number of potentially detected and aborted faults.

For all investigated X-ratios, accurate ATPG generates test patterns for a large number of faults that were classified as untestable or aborted by pessimistic three-valued ATPG. A considerable fraction of faults is proven to be potentially detected by fault simulation. However, the QBF-based analysis of potential detectability is able to generate potentially detecting test patterns for many more faults that are not definitely detectable.

The results also show a relatively large number of aborted faults for circuit c7552, which includes parity check structures. This causes high runtimes due to the depth-first search strategy of the used QBF solver.

Application to Larger Circuits For X-ratios of 1%, 2%, and 5%, the increase of fault coverage of accurate ATPG w.r.t. three-valued ATPG is shown in Figure 7.12. Accurate ATPG is able to generate a detecting pattern for a large number of faults which have been aborted or classified as untestable by three-valued ATPG. With increasing X-ratio, up to 24 percent points (c6288) higher fault coverage is achieved by the accurate

analysis. For the larger industrial circuits, up to 7 percent points higher fault coverage is achieved (p78k).

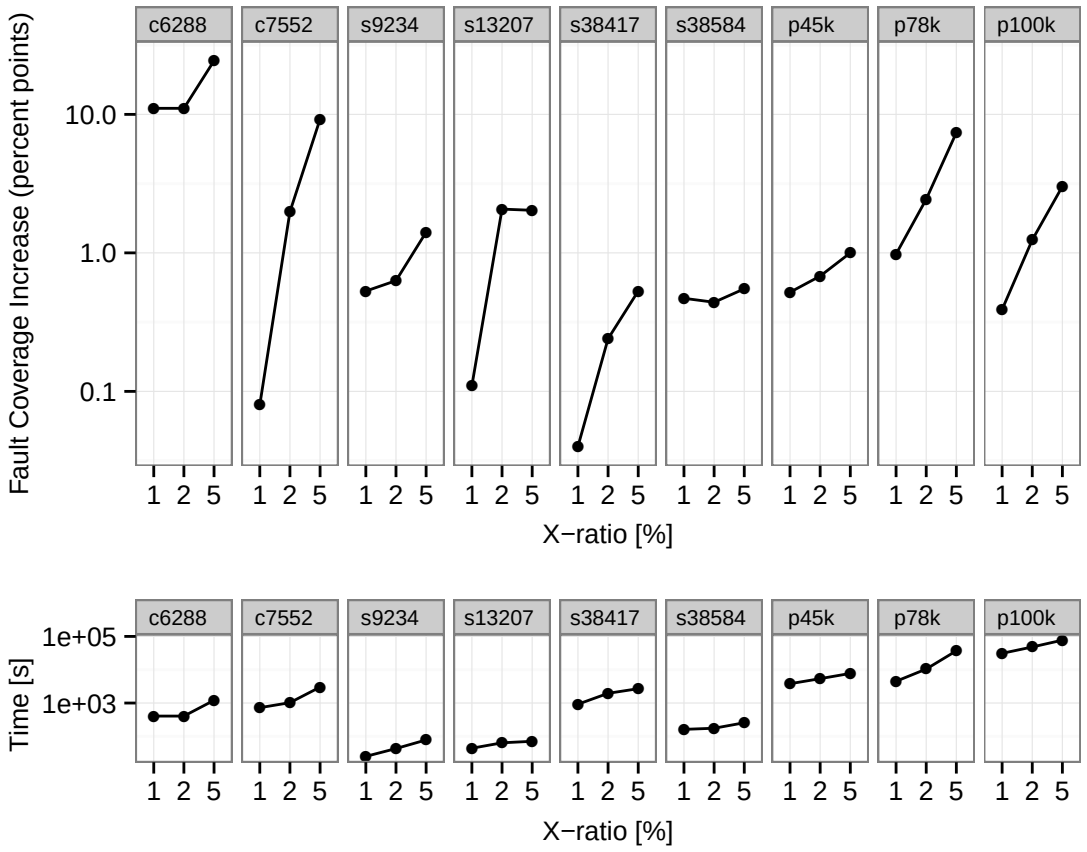


Figure 7.12: Fault coverage increase of accurate ATPG compared to three-valued ATPG and runtime in seconds of accurate ATPG.

The pessimism in conventional ATPG algorithms is highest for circuits with many reconverging paths such as the multiplier c6288, and circuits with parity 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, accurate ATPG 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%.

The figure also shows the runtime in seconds of accurate ATPG including three-valued ATPG and accurate fault simulation (cf. Figure 7.6). The runtime is dominated by the

runtime of the QBF solver for hard faults with a timeout of 10 seconds.

7.6.3 Accurate Sequential ATPG

In the selection of X-sources for sequential ATPG, primary and pseudo-primary inputs are distinguished: Selected *primary* inputs generate a unique X-value in each cycle, selected *pseudo-primary* (e.g. uninitializable non-scan flip-flops) generate an X-value in the first cycle, but can be initialized for subsequent cycles by capturing a binary value. The other inputs are controllable.

Fault Coverage Increase over Multiple Time Frames The first experiment compares the fault coverage of a SAT-based three-valued ATPG implementation with accurate QBF-based ATPG for different numbers of test cycles. Circuit s5378 is analyzed for up to nine time frames and an X-ratio of 5%. Figure 7.13 shows the fault coverage of both approaches.

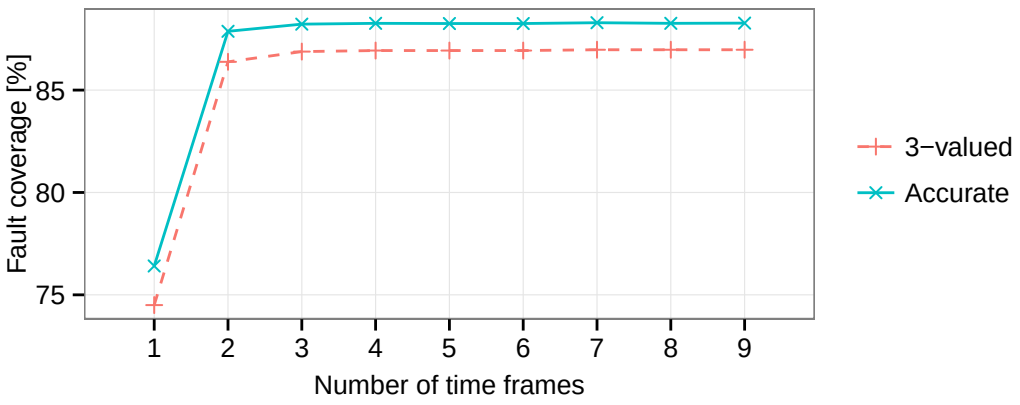


Figure 7.13: Fault coverage in multi-cycle three-valued and accurate ATPG for circuit s5378 and an X-ratio of 5%.

For the three-valued ATPG, the consideration of a second time frame increases fault coverage by 11.9 percent points since a large fraction of uninitialized sequential elements can be set to a defined value in the first time frame. Using the accurate ATPG, the fault coverage increases even further by up to 1.33 percent points and always exceeds the coverage of three-valued ATPG when the same number of time frames are used. Compared to single-cycle ATPG, the maximum increase is reached at seven time frames with 13.8 percent points higher coverage.

The figure also shows that the fault coverage increase quickly saturates after three time frames. A larger number of time frames leads to much higher runtime for the untestability proofs of yet undetected faults.

Application to Larger Circuits Based on the results of Figure 7.13, the maximum number of considered time frames for larger circuits in the following experiments is limited to three. Figure 7.14 shows the increase in fault coverage of three-valued and accurate multi-cycle ATPG compared to single-cycle three-valued ATPG for up to three time frames and an X-ratio of 5%.

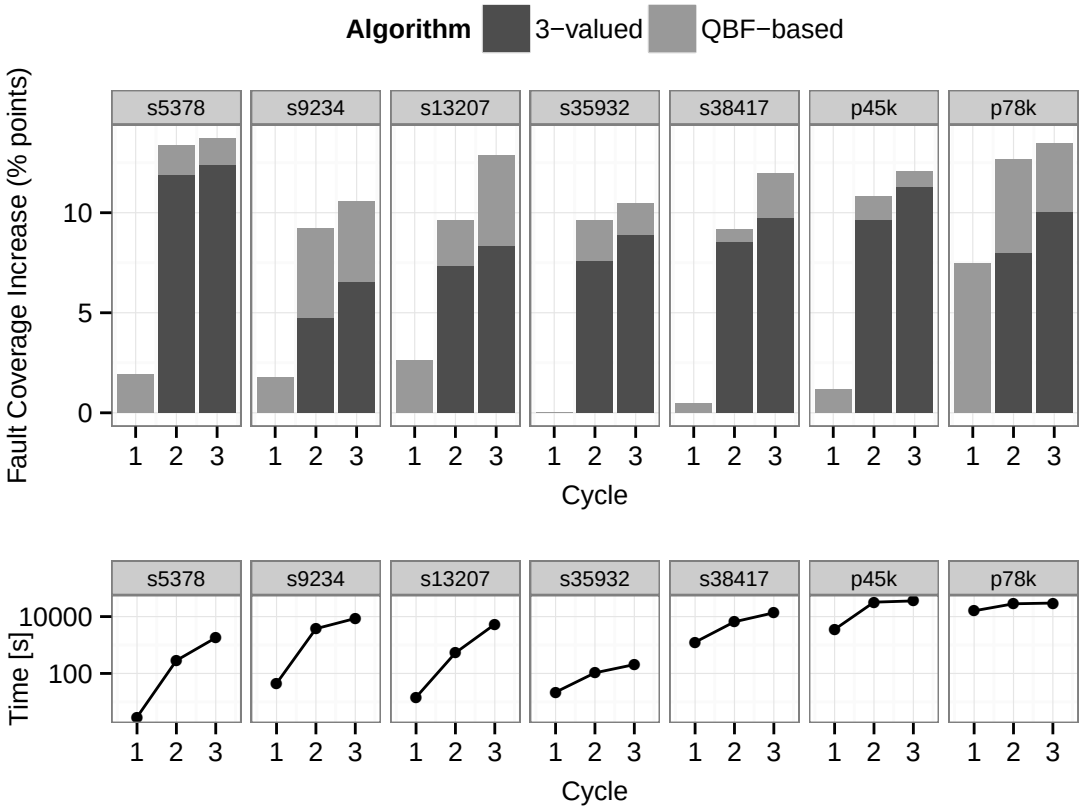


Figure 7.14: Fault coverage increase over single-cycle three-valued ATPG for up to three cycles and an X-ratio of 5%, and runtime of accurate QBF-based ATPG.

The results show that three-valued multi-cycle ATPG already increases fault coverage considerably: The maximum increase is 11.9 percent points for two time frames (s5378) and 12.38 percent points for three time frames (s5378). However, accurate

ATPG further increases the coverage since situations as discussed in Figure 7.9 are correctly considered. For an X-ratio of 5%, fault coverage further increases by up to 4.53 percent points for the ISCAS'89 circuits, and by up to 3.41 percent points for the industrial circuits over to the three-valued multi-cycle ATPG. For circuit p78k, 13.4% more faults are classified as detectable in comparison to single-cycle three-valued ATPG. This boosts fault coverage from 85.2% to 98.7% without the need for any X-blocking or X-masking hardware. In consequence, the number of untested faults is reduced by more than 90%.

On average for an X-ratio of 5% and a maximum of three time frames, three-valued multi-cycle ATPG classifies 9.86% more faults, and the accurate multi-cycle ATPG 12.3% more faults as detectable compared to single-cycle three-valued ATPG.

Figure 7.14 also shows the runtime of the accurate QBF-based ATPG in seconds. For most circuits, the runtimes increase super-linearly with the number of considered time frames showing the high complexity of the underlying problem. Still, accurate ATPG can be applied to industrial circuits for a limited number of cycles.

7.7 Summary

State-of-the-art test generation algorithms are based on pessimistic n-valued logic algebras. Consequently, these algorithms may fail to generate test patterns for all testable faults in the support of X-sources. To overcome this pessimism, accurate test generation in presence of X-values has been mapped to the satisfiability of quantified Boolean formulas [Hille13]. A complete ATPG framework integrating hybrid n-valued test generation and accurate fault simulation has been developed and applied to benchmark and industrial circuits. The framework has been extended to support multi-cycle test generation [Erb13].

The experimental results show that accurate test generation can increase fault coverage over conventional test generation algorithms by more than 24 percent points (circuit c6288). The proposed accurate algorithms are applicable to larger benchmark and industrial circuits.

8 Application to Test Data Compression and Compaction

The cost for test generation and application constitutes a significant fraction of the overall cost of a VLSI product and may even exceed the cost for actual manufacturing [Bushn00, ITRS12]. The test cost depends on test equipment (Automated Test Equipment, ATE) required to conduct electrical tests on chips, required test time, design and hardware overhead of test, and test generation costs. ATE costs increase with test data volume, i.e. the number of test stimuli and responses, the number of ATE interface channels or pins, and the channel frequency or data bandwidth. With increasing test coverage, the requirements on test equipment, test data volume, test time and test generation typically rise as well [Nag02].

Design-for-test targets the increase of fault coverage and reduction of test costs by easing the effort for test generation and application. This is achieved by dedicated hardware structures that increase observability and controllability of circuit nodes, and structures to reduce the bandwidth requirements on ATE.

This chapter firstly introduces design for test structures and a novel test data compression algorithm [Kocht09, Kocht08] that enables a cost-efficient low bandwidth test. Section 8.2 quantifies the adverse impact of pessimistic X-propagation analysis of a test set on test response compaction, which results in overmasking of valuable response data [Kocht12], or even masking of erroneous response bits.

8.1 Design for Testability

8.1.1 Scan Design for High Controllability and Observability

In sequential circuits, a long input pattern sequence may be required to reach a particular state, even if the resetting of the sequential elements is possible. Certain states may not be reachable at all. This affects the test time, test data volume and reachable fault coverage. Scan design increases the testability of sequential circuits: The n sequential elements (flip-flops or latches) are substituted by scan cells and connected to one or multiple scan chains, also illustrated in Figure 1.5 in Section 1.4. This allows to set any state in at most n cycles, and any of the n cells can be observed after at most n cycles [Willi73]. If the n scan cells are grouped into k chains, the k bits shifted out in one cycle are called a scan vector.

In full-scan design, all of the n sequential elements are included in the scan chains and become directly controllable and observable. This reduces the problem of test generation for a sequential circuit to that of test generation for its combinational part. In partial-scan design, only a subset of the n sequential elements is included in scan chains. In this case, test generation has to consider the state of the non-scan sequential elements. If the non-scan elements are resetable, they can be set into a defined state before test application. Otherwise, they must be considered as X-sources representing an unknown state.

8.1.2 Compression of Test Stimuli

During test application, a large amount of test stimuli is applied to the circuit under test. In a built-in self test (BIST) architecture, this data is generated on chip by a pseudo-random test pattern generator or by embedded deterministic patterns [Kiefe97]. In the latter case, the hardware overhead critically depends on the amount of data to be stored. In external test application, the data is delivered by an ATE. Test time and ATE cost increase with the amount of required data.

Test data compression reduces the amount of data to be stored or transmitted. Compression can be achieved in the space domain or time and space domain (sequential decompression) [Wang08a]. Decompression in the space domain include linear decompression and broadcast schemes. Sequential decompression is based on linear

feedback shift registers (LFSRs) [Koene91], or cellular automata [Rajsk04]. The seed bits for these finite state machines represent the compressed test patterns.

Both types of compression exploit that typically only a small subset of bits in a test pattern needs to be specified to test the target faults [Miyas04], while the remaining bits may have an arbitrary value (*don't care* bits).

Compressed test patterns can be generated directly during ATPG by adding a combinational representation of the decompressor structure to the circuit model [Wang08b]. For maximum fault coverage, the compactor at the output side has to be added to the circuit model as well. The efficiency of this approach strongly depends on the type of compression and compaction.

The alternative is a two-step process, which first generates an uncompressed test set with maximum coverage and a small number of specified bits, and then encodes the patterns in an efficient way. This also adds flexibility in the choice of the encoding technique. The two-step process is applicable to encoding techniques that work on a pattern-by-pattern basis, as well as continuous reseeding-based compression techniques such as [Volke03].

8.1.3 Compaction of Test Response Data

At the output side, a large amount of test response data must be evaluated, i.e. compared with the expected responses. Since it is not known in advance where the effect of a defect can be observed, each response bit must be evaluated. In addition, in multi-site test, many devices are tested in parallel, and the individual responses must be evaluated in parallel. This increases the output (response data) bandwidth requirements even further, which may exceed the capabilities of the ATE.

Output compaction can be applied in the space domain, time domain, or both [Saluj83, Barde82, Barnh01]. Space compaction benefits from simple construction without timing constraints, while complex compaction in space and time domain offers higher compaction ratios up to multiple orders of magnitude. The compacted responses, called signatures, are highly sensitive to the response data to ensure that even a single bit difference alters the resulting signature and allows for fault detection.

8.1.4 Masking-Aware Test Set Encoding (MATE) for Extreme Response Compaction

Test time in scan-based designs is significantly reduced by splitting long scan chains into multiple short ones. However, input and output bandwidth requirements increase. Multi-site test in volume production requires *low and constant bandwidth* at the input and output side. The maximum bandwidth requirement during test determines the required number of tester channels (pins) and their operation frequency, with direct implications for ATE cost.

At the output side, the bandwidth increase due to scan chain splitting can be handled by aggressive compaction of the scan vector into a single parity bit. If the compactor structure is included during ATPG to overcome fault masking in the compactor, this approach does not impair fault coverage nor diagnostic resolution [Holst09].

However, the bandwidth management at the input side becomes a challenge when using compression methods that exploit don't care bits in test patterns: Including the compactor during ATPG results in overspecified patterns since ATPG explicitly defines the propagation paths of faults through the compactor. In Masking-Aware Test Set Encoding (MATE, [Kocht09]), this problem is solved by an iterative test pattern encoding method for reseeding-based test compression. MATE employs a heuristic algorithm to identify don't care bits in the patterns and limit the number of specified bits per pattern before computing the seed bits for the encoded patterns.

Identification of Don't Care Values in Test Patterns

For a given test set and target faults, the bits in each pattern that are not required for fault detection can be approximated by heuristic algorithms based on simulation and critical path tracing, e.g. [Miyas04], or by employing the implication engine of a three-valued ATPG algorithm as proposed in [Kocht08]. The *optimal* solution, however, requires accurate reasoning as discussed in Chapter 7 since the don't care bits in a test pattern behave like X-sources.

In [Sauer13], the problem is solved optimally by a mapping to a QBF instance. In this instance, the QBF solver may choose to assign either a binary value or a don't care value to a circuit input. Let two additional variables $i_\exists$ and $i_\forall$ per input i represent the

binary or don't care value of input i. In principle, the choice of a binary or don't care value for an input is mapped to a multiplexer as shown in Figure 8.1.

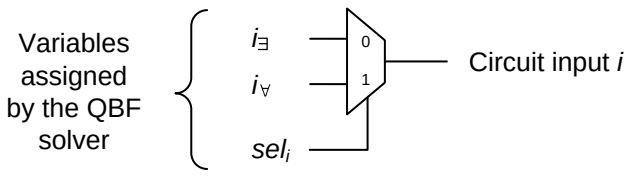


Figure 8.1: Principle of QBF-based accurate identification of don't care bits.

The clauses $C_{\text{Dont-Care-Select}}$ represent this choice depending on variable sel_i per input:

$$C_{\text{Dont-Care-Select}} := (sel_i \wedge (i \leftrightarrow i_\exists)) \wedge (\neg sel_i \wedge (i \leftrightarrow i_\forall)).$$

Further, let the clauses $C_{\text{Detection}}$ represent the fault detection constraints. Then, the QBF instance considering one input is $\exists sel_i \, \exists i_\exists \, \forall i_\forall : \big(C_{\text{Circuit}} \wedge C_{\text{Dont-Care-Select}} \wedge C_{\text{Detection}}\big)$.

The results in [Sauer13] show that heuristic approaches identify fewer don't care bits than the QBF-based approach. However, heuristic approaches approximate the optimal solution very well, with a difference in don't care bits of only approximately 1 percent point for larger circuits at much better scalability.

The heuristic method of [Kocht08] is able to limit the maximum number of specified bits in a test pattern and can be combined with test pattern duplication [Pomer06] to further reduce the limit. This is relevant for the generation of efficient concurrent online test architectures with encoded patterns [Kocht10b] and test compression [Kocht09] as discussed next.

Test Set Encoding Exploiting Don't Care Values

The proposed masking aware test set encoding (MATE) uses an iterative flow as shown in Figure 8.2: An initial test set T is generated using ATPG for the circuit including the compactor for the targeted faults.

Test set stripping [Kocht08] is applied to the patterns with a given limit l_{avg} of specified bits per pattern. For some faults there exists no pattern in test set T that detects it with at most l_{avg} specified bits. MATE allows that the number of specified bits in a single pattern exceeds l_{avg} as long as the limit is kept on average over multiple patterns. The

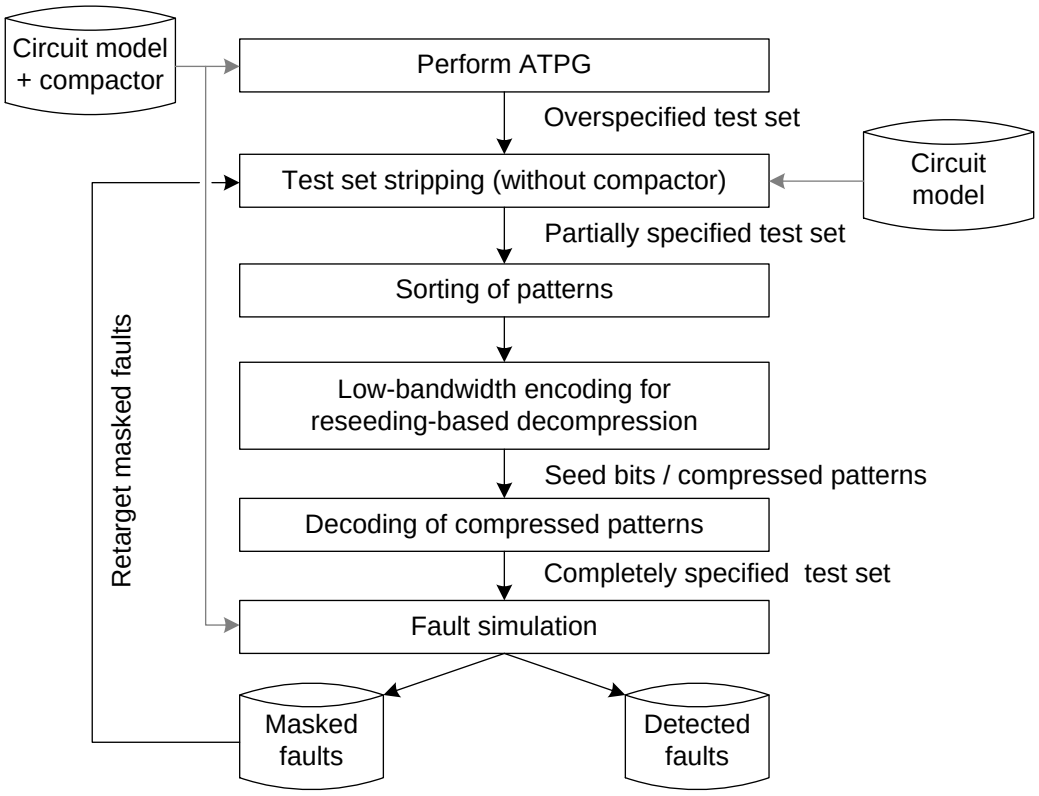


Figure 8.2: Masking-aware test set encoding (MATE) for extreme response compaction.

set of partially specified patterns T_{strip} is sorted such that the pattern sequence shows a balanced distribution of specified bits. This increases the probability that the patterns in T_{strip} can be encoded for the reseeding-based decompressor.

In the next step, the seed bits for the partial reseeding scheme are computed for the sorted patterns. An LFSR is used as a decompressor. The seeds are computed considering a window of multiple patterns (or scan vectors) to balance fluctuations in the density of specified bits in the vectors and patterns. The seed bits are then decoded with the targeted decompressor LFSR. The resulting fully specified patterns T_{decoded} are subject to fault simulation using the circuit model including the compactor.

Some of the targeted faults may have been missed because of inadvertent masking or the rare case of a failed encoding. These faults are retargeted in the next iteration of the flow. Once all target faults have been detected or saturation is reached, i.e. no additional faults have been detected in one iteration, the process stops.

The average number of specified bits l_{avg} in a test pattern that can be encoded by the

partial reseeding scheme depends on

- ▷ the coding efficiency of the reseeding scheme,
- ▷ the number of injected seed bits n to the decompressor per cycle (this typically corresponds to the number of used ATE channels), and
- ▷ the maximum length t of the scan chains in the circuit.

Assuming a coding efficiency of 90%, l_{avg} evaluates to $0.9 \cdot n \cdot t$. n is set to the minimum value that results in no or only a negligible number of failed encodings.

Still, even if the number of specified bits theoretically allows encoding, some parts of the vectors or sequences of vectors may exhibit a conflicting combination of specified bits, and the encoding might fail in rare cases. The accurate method of [Sauer13] is able to compute and enumerate test patterns with the smallest amount of specified bits, which may alleviate the problem. However, that method is not scalable enough to be applied to larger circuits. Experimental results show that the heuristic method of [Kocht08] provides sufficiently good results for successful test set encoding.

Achieved bandwidth reduction: The MATE approach has been applied to industrial circuits provided by NXP. For the decompressor, an 128-bit LFSR is used. The results are compared to the original scan and test architecture, extended by the space compaction of [Mitra04].

Using MATE, the number of test patterns increases by $2.85\times$ on average when the compactor is included during ATPG. Still, the test time is reduced by $2.4\times$ on average due to test application using very short scan chains in MATE.

For all but two circuits, only two or fewer seed bits have to be injected into the decompressor per shift cycle. On average over all circuits, 2.1 bits have to be injected per cycle. At the output side, one compacted bit is generated per cycle. Table 8.3 shows the reduction of the required input and output bandwidth of MATE compared to the original test architecture and space compaction. The weighted average of the bandwidth reduction by MATE is $31.2\times$.

The original scan architecture of circuit p469k contains just one scan chain. Thus, the bandwidth is already minimal. However, after splitting the chain and with a small increase in bandwidth, MATE reduces the test time by a factor of $7.5\times$. For circuit p378k, the original scan architecture already has very short scan chains and is not

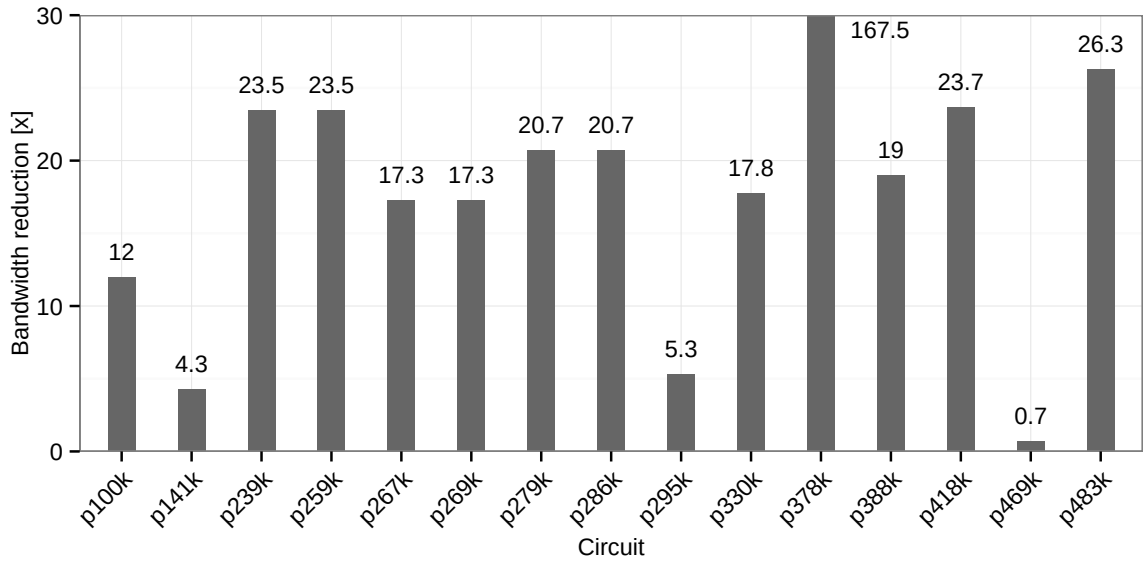


Figure 8.3: Bandwidth reduction of MATE compared to original test architecture and space compaction of [Mitra04].

subject to splitting. Consequently, test time is not reduced by MATE but increases by a factor of $3.3\times$ because of the higher number of generated patterns. At the same time, the required bandwidth reduces by a factor of $167.5\times$.

On average, the fault coverage increases slightly since identified don't care bits in the initial test set T are filled with different pseudo-random values during decompression in T_{decoded}. The diagnostic success rate, evaluated using the algorithm of [Holst09], slightly increases by 1.6% on average compared to the original test architecture. In line with the results of [Holst09], a larger number of patterns typically increases diagnostic resolution.

8.2 Handling Unknown Values by Design-for-Test

8.2.1 Blocking of Unknown Values

X-values in the circuit reduce node controllability and observability. X-sources can be blocked by test control points or X-blocking cells as shown in Figure 8.4. This limits the propagation of X-values in the circuit.

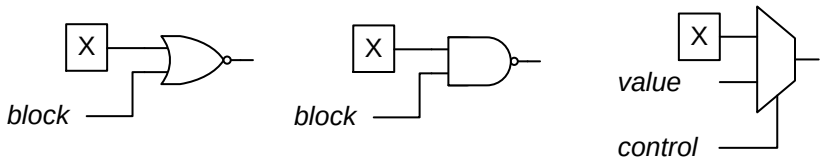


Figure 8.4: Examples of X-blocking cells and a test control point.

X-blocking cells introduce additional delay, which may prevent their placement at X-sources that drive long paths. Also, the additional wiring overhead for global control signals may prohibit blocking all of the X-sources in the circuit.

8.2.2 Test Response Compaction in Presence of Unknown Values

If an unknown value propagates to the response compactor and is included in the signature, the signature may be corrupted and useless for fault detection. The vulnerability to X-values depends on the compaction scheme. In consequence, different techniques have been developed to prevent X-values from corrupting the response data or signature. All of them benefit if the number of X-values to be handled can be reduced by accurate analysis as proposed in Chapter 5.

X-tolerant space compactors are combinational circuits that compact one scan vector into a signature per shift-out cycle. They are constructed such that a limited number of X-values may enter the compactor but the signature is still able to distinguish fault-free responses from faulty responses. Prominent examples are discussed below, an extensive overview can be found in [Wang08a].

In [Patel03], the application of space compactors based on error detecting codes is investigated. In presence of k X-values in a response scan vector, up to 2^k signatures correspond to the fault-free circuit response. The authors suggest to perform the comparison on the ATE. Obviously, if the number of X-values can be reduced by accurate analysis, the number of comparisons and the required storage on the ATE can be significantly reduced.

In X-compact [Mitra04], the applied error detecting code is a linear block code that allows a low fixed number of X-values per scan vector to enter the compactor. The authors propose to determine the signature bits with dependence on X-values by clas-

sical (pessimistic) logic simulation. These bits must then be masked by the ATE and excluded from comparison with the fault-free signature.

The post-processing overhead imposed on the ATE by such schemes is addressed by an on-chip X-filter in [Sharm05]. The X-filter is controlled by additional external inputs and computes a deterministic signature on chip for a low number of X-values per vector.

In [Wohl07], a selector structure allows to select the response bits that enter the compactor. The selectors are controlled by masks and additional ATE channels. This allows to change the scan chains that feed the compactor per pattern to a certain degree per scan vector. In the case of a large number of X-values, a lot of response data is still lost since the selection must be enabled for many scan vectors.

X-tolerant time compactors compute a signature over the response bits of multiple scan vectors, multiple patterns, or even the whole test pattern set. Two types of time compactors and related X-handling approaches are distinguished: finite impulse response (FIR) compactors such as convolutional compactors [Rajsk05], and infinite impulse response (IIR) compactors like MISR (Multiple Input Shift Registers) based compaction.

X-values fed into finite impulse response compactors corrupt only a subset of bits of the signature. If the number of X-values entering the compactor is small, the corrupted bits can be tolerated and extracted on the ATE [Mruga05] without a significant loss of response data. X-values fed into infinite impulse response compactors may affect all signature bits generated in future compaction cycles (until the next reset of the compactor). To a certain degree, they can be tolerated by X-canceling schemes [Touba07] where response data without X-dependency is extracted during post-processing of response data by algebraic computations.

A pessimistic analysis of X-propagation in the circuit increases the number of X-values included in the signature. This reduces the amount of useful response data without X-dependency, the compaction ratio, or increases the hardware overhead. In order to maximize the compaction ratio, it is crucial to accurately identify the response bits with an X-value.

X-masking logic converts predetermined X-values in response bits during shift-out to a specified value by using X-blocking cells (cf. Figure 8.4). X-masking logic is synthesized between the scan chain outputs and the inputs of the compaction logic to control which scan chains or scan vectors feed the compactor, for example [Chick04, Chao05].

Most approaches mask a superset of the scan cells that capture X-values. This overmasking is undesirable since any response data—even correct response bits—contains valuable information for diagnosis. A trade-off between the control data for X-blocking cells and overmasked response bits has to be found. With fewer X-values to be masked, better solutions for the trade-off between overhead and overmasking can be found. In masking approaches for BIST [Narus03, Tang06], this also reduces the hardware overhead required to store the masking control function.

For all the aforementioned X-handling schemes it is beneficial to identify the response bits carrying X-values as accurately as possible. An overapproximation of the response bits with X-values, as provided by pessimistic n-valued simulation algorithms, increases the overhead for X-handling, results in overmasking and in loss of valuable response and failure data.

Overmasking in Scan Chain- and Vector-Based X-Masking

The following experiment investigates the impact of accurate logic simulation on simple X-masking architectures [Kocht12]. In scan vector and scan chain masking, either a complete scan vector or complete chain is masked per pattern. More complex schemes may benefit even more from an accurate analysis. The experiment is performed on industrial circuits provided by NXP.

The used scan architecture of the circuits is identical to the one used in the MATE experiments in Section 8.1.4, i.e. many very short scan chains to reduce test time. Different subsets of primary and pseudo-primary circuit inputs are randomly selected as X-sources per iteration of the experiment. Here, 1% of the inputs are assumed to generate X-values. From these distributions of X-sources, three are selected for the evaluation: The distribution with the minimum number of X-valued outputs, an average one and the one with the maximum number of X-valued outputs. For each X-source distribution, test patterns are generated with a commercial X-aware ATPG tool. On average, the X-density, i.e. the fraction of primary and pseudo-primary outputs

with an X-value, is 2.4%. This is in line with results reported from industry [Wohl07] where X-densities of even up to 10% have been observed.

Figure 8.5 shows the degree of overmasking in scan vector and chain masking for the average X-distribution.[1] Let $\#\text{XChains}_{\text{3-valued}}$ denote the number of chains that contain at least one X^P-valued response bit according to three-valued analysis. And let $\#\text{XChains}_{\text{Accurate}}$ be the number of chains that contain at least one X-valued response bit according to the accurate analysis. Then, the overmasking ratio quantifies the fraction of chains (vectors) which are needlessly masked due to the pessimism of three-valued analysis:

$$\text{Overmasking ratio} = 100\% \cdot \left(1 - \frac{\#\text{XChains}_{\text{Accurate}}}{\#\text{XChains}_{\text{3-valued}}}\right).$$

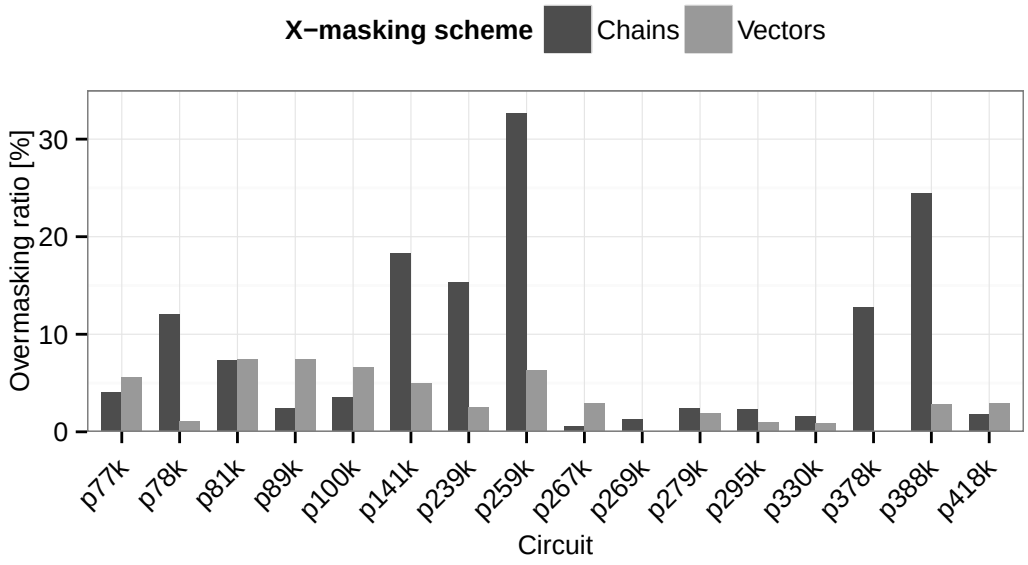


Figure 8.5: Overmasking of scan chains (vectors) for the average X-source distribution with 1.0% X-ratio.

For circuit p259k, for instance, 32.6% of the masked scan chains (6.3% of the vectors) after three-valued analysis are overmasked. Thus, a pessimistic three-valued analysis results in a significant loss of response and potential failure data.

[1] The results for the X-configurations with minimum and maximum number of X-valued outputs are given in Table B.11 in the Appendix.

Error Masking Probability in X-Compact Space Compaction

The X-compact space compactor [Mitra04, Mitra05] compacts n response bits of a scan vector into an m bit signature and is able to tolerate a low and fixed number of X-values in the vectors. It can be constructed deterministically or stochastically. The stochastic construction assumes that the scan chain outputs that are compacted into one signature bit are chosen randomly with a given probability p. This permits the stochastic analysis of error masking probabilities [Mitra05], which is applied in the following to assess the impact of an accurate analysis of X-propagation in the circuit compared to pessimistic three-valued analysis. The analysis is performed for the industrial circuits and test sets of Section 8.1.4.

Following the argument of [Mitra05], the probability that a *single* erroneous response bit in a scan vector is masked in the compactor due to k X-values is:

$$p_{\text{masked}} = \left(1 - p \cdot (1-p)^k\right)^m . \tag{8.1}$$

This masking probability p_{masked} is minimal when $p = \frac{1}{k+1}$ [Mitra05]. Equation (8.1) can be generalized to estimate the masking probability of t erroneous bits in the vector:

$$p^t_{\text{masked}} = \left(1 - \sum_{1 \le i \le t, i \text{ is odd}} \binom{t}{i} \cdot p \cdot (1-p)^{t-i+k}\right)^m . \tag{8.2}$$

Practical values for t are reported to range from three to five [Mitra05].

For the experiments, the number of signature bits m is conservatively chosen to be 22 (according to [Mitra04] suitable for designs with up to 2048 scan chains). As in the preceding experiment (Section Overmasking in Scan Chain- and Vector-Based X-Masking), we assume that 1% of the inputs generate X-values and compute the average X-density in the scan vectors per circuit by three-valued simulation and accurate analysis. This determines k, the number of X-values per vector, and in consequence p. With these values, p^t_{masked} is computed for $t = 1, 3, 5$ according to Equation (8.2) and plotted in Figure 8.6.[2]

In general, the masking probability is rather high, indicating that the X-compact scheme is suitable only for very low X-densities. In all cases, the accurate analysis reduces the

[2]Circuit p77k has been omitted in the graph for clarity since its error masking probability is two orders of magnitude smaller compared to the other circuits.

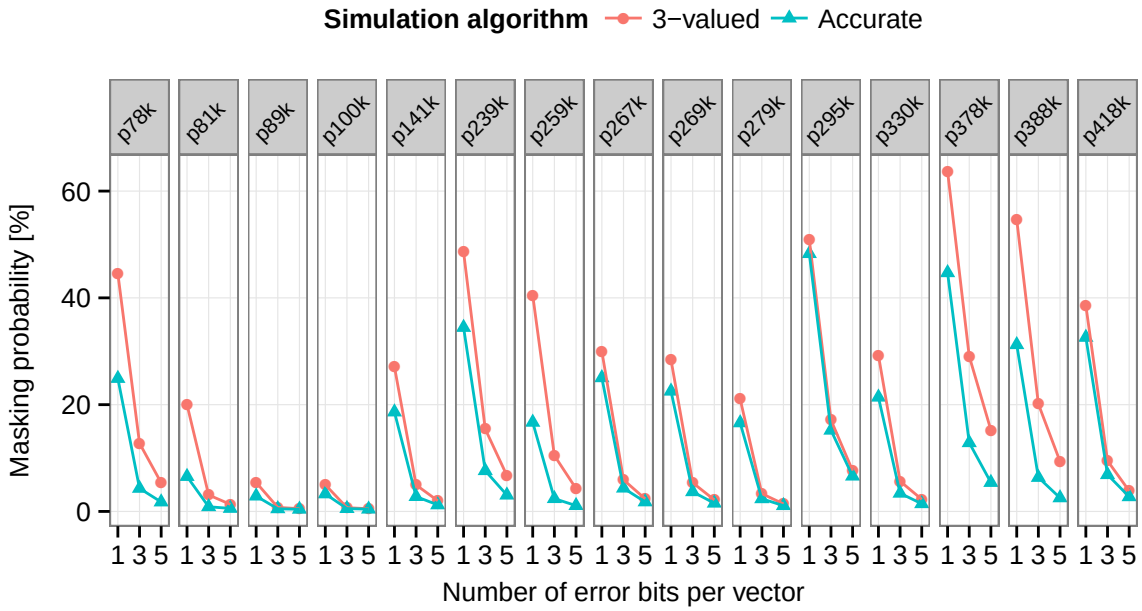


Figure 8.6: Masking probability of error bits in a scan vector for three-valued and accurate analysis of the test set.

masking probability by a factor of up to $4.4\times$.

For different X-densities, Figure 8.7 shows the masking probability according to Equation (8.2) and different numbers of erroneous bits t per vector. The accurate analysis reduces the X-values in the circuit response on average by 26.6% compared to three-valued simulation for the circuits and scenario considered above. Based on this value, the masking probability p^t_{masked} reduces by up to 29.5%.

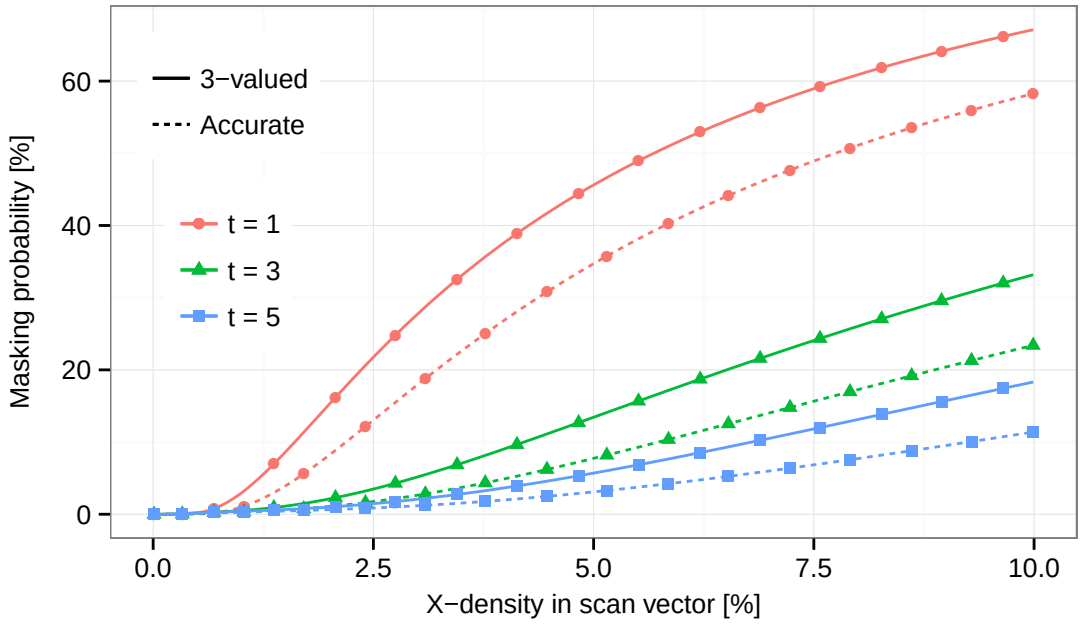


Figure 8.7: Masking probability of error bits in a scan vector depending on X-density in the response for three-valued and accurate X-analysis.

8.3 Summary

To reduce test time and response bandwidth requirements in multi-site testing, extreme output compaction has been proposed recently. This scheme, however, significantly increases the input bandwidth requirements on the ATE. The masking aware test set encoding (MATE, [Kocht09]) approach proposed in this work overcomes this challenge to enable low pin count test application. It uses a reseeding-based decompressor with low constant input bandwidth and exploits don't care bits in a partially specified test set. This test set is generated by post-processing of a fully specified test set using an effective test set stripping algorithm [Kocht08] to identify the input assignments not required for fault detection. Such don't care bits behave like X-sources in the circuit. Thus, the identification of the maximum number of don't care input assignments is an optimization problem of the NP-hard decision problem of accurate ATPG. For larger circuits, this complexity demands for a heuristic algorithm, as the proposed one.

Test response compaction requires special handling of X-values since they may corrupt

the compacted response. Different methods have been proposed to tolerate, cancel, or mask X-values in compaction architectures. All of the these methods benefit in terms of hardware overhead, test data overhead, or overmasking of response data if the number of X-values to be handled can be reduced. In consequence, a pessimistic n-valued analysis of X-propagation in the circuit needlessly increases test cost and reduces test quality. For scan chain and vector based X-masking schemes [Kocht12] and the X-compact space compactor, the impact of an accurate analysis has been quantified. The accurate analysis of X-propagation in the circuit allows to significantly reduce overmasking of response data in the compactor.

9 Conclusions

Unknown (X) values emerge in different steps of the design of digital circuits. Sources of X-values during modeling are partially specified blocks or inaccuracies at model boundaries. During operation and test, the state of nodes may be unknown at modeling time due to uninitialized or uncontrolled sequential elements, at analog/digital or clock domain boundaries. The resulting X-values have adverse impact on the quality of design validation, verification, test and diagnosis tasks.

Reasoning about circuit behavior in presence of X-values and computing their propagation paths are NP-hard or NP-complete problems. In industrial practice, the problem is pessimistically approximated by use of modeling and algorithms based on n-valued logic algebras. However, these n-valued algorithms cannot accurately evaluate combinations of different X-values and consequently overestimate the number of X-values in the circuit.

The consequences of this pessimism in test generation and test-related design steps are a potential loss of fault coverage and product quality, as well as increased test overhead. This overhead includes design effort and additional hardware for design-for-test structures, increased test data volume, and higher test time.

This pessimism and its consequences in the field of testing have been addressed in this work. Algorithms for accurate logic and fault simulation, and accurate combinational and multi-cycle test generation for circuits with X-sources have been presented. The combination of heuristics and formal reasoning, based on binary decision diagrams and Boolean and quantified Boolean satisfiability, allow for the first time accurate reasoning even for larger circuits.

Accurate logic simulation in presence of X-values is an NP-complete problem. In this work, a flexible approximate algorithm using local binary decision diagrams is developed to trade-off computational effort and accuracy. Furthermore, a complete algorithm is proposed that computes the accurate propagation of X-values by using heuristics and Boolean satisfiability. The increased accuracy is beneficial for design valida-

tion (e.g. of initialization sequences), as well as to reduce overmasking and costs in test data compression.

These algorithms have been extended and applied to fault simulation. Accurate fault simulation showed that a considerable fraction of faults that are classified as undetected by widely used n-valued algorithms are actually detected by a given test set. This increase in fault coverage and product quality does not incur any additional hardware or test overhead.

Accurate test generation is an NP-hard problem. Here, it has been handled by a mapping to the satisfiability of quantified Boolean formulas. The results from benchmark and industrial circuits demonstrate that a remarkable increase in fault coverage compared to state-of-the-art test generation algorithms is possible without additional design-for-test hardware structures.

In summary, accurate Boolean reasoning in digital circuits with X-sources is practically applicable with direct benefits for test quality in terms of coverage and design-for-test costs, and eventually the final product.

Ongoing and Future Research

The algorithms developed in this work for accurate reasoning about circuit behavior in presence of X-values have already been extended and applied in related fields:

Circuit nodes affected by interconnect open faults depend on aggressor nodes closeby located in the layout. This may introduce sequential behavior and unknown node states during operation. In the recent work of [Erb14b], the accuracy of interconnect open fault simulation has been increased by adapting the accurate fault simulation algorithm of [Erb14a]. This step helps to improve the overall efficiency of ATPG for interconnect open faults.

The mapping of the accurate test generation problem to QBF satisfiability allowed handling industrial circuits of moderate size. For much larger circuits, a different, domain specific modeling is currently investigated: Restricted symbolic logic (cf. Section 5.3.3) is a powerful heuristic to increase the accuracy of logic simulation in presence of X-values. The semantics of this logic can also be used to increase the accuracy of test generation by explicitly encoding indices of X-symbols, extending the characteristic formulas of the modeled gates correspondingly, and the introduction of a novel

D-chain. While such a modeling fails to provide the accurate result, first results indicate a negligible difference to the accurate solution at a remarkable speed-up of the test generation process.

The proposed algorithms can also form the basis for other fields where X-values emerge. With an increasing number of X-values, the importance of accurate reasoning grows as well. Defect mechanisms in nanoscale technology prompt for thorough testing of small gate delay faults. For certain faults, this mandates testing at increased, faster than nominal frequency. In launch-of-capture delay testing, this implies a drastic increase of X-values in the capture cycle. Accurate test pattern generation and fault simulation can help to increase fault coverage and product quality, and to improve effectiveness of response compaction in this type of test application.

Recently, reconfigurable scan networks emerge as a scalable means of access to the increasing amount of on-chip instrumentation. Such scan networks possess high sequential depth, and access pattern generation requires the consideration of complex combinational and sequential dependencies [Baran12]. Verification of safety and security properties [Baran13], test generation and diagnosis of these scan networks can benefit from accurate reasoning if a subset of elements cannot be reset or is defective.

Bibliography

[Abram90] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design—Revised printing*. IEEE Press, 1990. ISBN 0-7803-1062-4.

[Agraw88] V. D. Agrawal, K.-T. Cheng, D. D. Johnson, and T. Sheng Lin. Designing circuits with partial scan. *IEEE Design & Test*, 5(2):8–15, March 1988.

[Akers76] S. B. Akers. A logic system for fault test generation. *IEEE Transactions on Computers*, C-25(6):620–630, June 1976.

[Akers78] S. B. Akers. Binary decision diagrams. *IEEE Transactions on Computers*, C-27(6):509–516, 1978.

[Antre87] K. Antreich and M. H. Schulz. Accelerated fault simulation and fault grading in combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(5):704–712, 1987.

[Armst72] D. Armstrong. A deductive method for simulating faults in logic circuits. *IEEE Transactions on Computers*, 100(5):464–471, 1972.

[Bahar93] R. Bahar, E. Frohm, C. Gaona, G. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *Digest of Technical Papers of 1993 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 188–191. November 1993.

[Baran12] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Modeling, verification and pattern generation for reconfigurable scan networks. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1–9. IEEE Computer Society, 2012.

[Baran13] R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Securing access to reconfigurable scan networks. In *Proceedings of the 22nd IEEE Asian Test Symposium (ATS)*, pages 295–300. 2013.

[Barde82] P. Bardell and W. McAnney. Self-testing of multichip logic modules. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 200–204. 1982.

[Barnh01] C. Barnhart, V. Brunkhorst, F. Distler, O. Farnsworth, B. Keller, and B. Koenemann. OPMISR: The foundation for compressed ATPG vectors. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 748–757. 2001.

[Becke99] B. Becker, M. Keim, and R. Krieger. Hybrid fault simulation for synchronous sequential circuits. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 15(3):219–238, Springer, 1999.

[Biere09] A. Biere, M. Heule, H. van Maaren, and T. Walsh, editors. *Handbook of Satisfiability*. Frontiers in Artificial Intelligence and Applications 185. IOS Press, 2009. ISBN 978-1-58603-929-5.

[Boole54] G. Boole. *An Investigation of the Laws of Thought*. Walton and Maberly, 1854.

[Brand93] D. Brand. Verification of large synthesized designs. In *Proceedings of the 1993 IEEE/ACM International Conference on Computer-aided Design (ICCAD)*, pages 534–537. IEEE Computer Society Press, 1993.

[Brayt84] R. K. Brayton, G. D. Hachtel, C. T. McMullen, and A. L. Sangiovanni-Vincentelli. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers, 1984. ISBN 978-0-89838-164-1.

[Breue72] M. A. Breuer. A note on three-valued logic simulation. *IEEE Transactions on Computers*, 21(4):399–402, April 1972.

[Bryan86] R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Bryan87] R. Bryant. Boolean analysis of MOS circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(4):634–649, 1987.

[Bryan91] R. E. Bryant. On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication. *IEEE Transactions on Computers*, 40(2):205–213, February 1991.

[Brzoz87] J. A. Brzozowski and C.-J. H. Seger. A characterization of ternary simulation of gate networks. *IEEE Transactions on Computers*, 36(11):1318–1327, 1987.

[Brzoz01] J. Brzozowski, Z. Esik, and Y. Iland. Algebras for hazard detection. In *Proceedings 31st IEEE International Symposium on Multiple-Valued Logic*, pages 3–12. 2001.

[Bushn00] M. L. Bushnell and V. D. Agarwal. *Essentials of electronic testing for digital, memory and mixed-signal VLSI circuits*. Kluwer Academic Publishers, 2000. ISBN 978-0792379911.

[Bünin95] H. K. Büning, M. Karpinski, and A. Flögel. Resolution for quantified Boolean formulas. *Journal Information and Computation*, 117(1):12–18, February 1995.

[Cadol98] M. Cadoli, A. Giovanardi, and M. Schaerf. An algorithm to evaluate quantified Boolean formulae. In *Proceedings of the Fifteenth National/Tenth Conference on Artificial Intelligence/Innovative applications of artificial intelligence (AAAI/IAAI)*, pages 262–267. American Association for Artificial Intelligence, 1998.

[Carte89] J. L. Carter, B. K. Rosen, G. L. Smith, and V. Pitchumani. Restricted symbolic evaluation is fast and useful. In *Digest of Technical Papers: IEEE International Conference on Computer-Aided Design (ICCAD-89)*, pages 38–41. 1989.

[Chand89] S. Chandra and J. Patel. Accurate logic simulation in the presence of unknowns. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 34–37. 1989.

[Chang87] H. P. Chang and J. A. Abraham. The complexity of accurate logic simulation. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 404–407. 1987.

[Chao05] M.-T. Chao, S. Wang, S. Chakradhar, and K.-T. Cheng. Response shaper: a novel technique to enhance unknown tolerance for output response compaction. In *Proceedings of the IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 80–87. 2005.

[Cheng88] W.-T. Cheng. Split circuit model for test generation. In *Proceedings of the 25th ACM/IEEE Design Automation Conference (DAC)*, pages 96–101. June 1988.

[Cheng89] W.-T. Cheng and T. J. Chakraborty. Gentest: an automatic test-generation system for sequential circuits. *IEEE Computer*, 22(4):43–49, 1989.

[Cheng90] W.-T. Cheng and M.-L. Yu. Differential fault simulation for sequential circuits. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 1(1):7–13, Springer, 1990.

[Chick04] V. Chickermane, B. Foutz, and B. Keller. Channel masking synthesis for efficient on-chip test compression. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 452–461. 2004.

[Cho93] H. Cho, S.-W. Jeong, F. Somenzi, and C. Pixley. Synchronizing sequences and symbolic traversal techniques in test generation. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 4(1):19–31, Springer, 1993.

[Chou10] 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 Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(4):646–651, 2010.

[Clark93] E. M. Clarke, M. Fujita, P. C. McGeer, K. McMillan, J. C.-Y. Yang, and X. Zhao. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation, 1993. Carnegie Mellon University Computer Science Department, Paper 453.

[Cook71] S. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on Theory of Computing*, pages 151–158. 1971.

[Corno96] F. Corno, P. Prinetto, M. Rebaudengo, and M. Sonza Reorda. GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(8):991–1000, 1996.

[Cox94] H. Cox and J. Rajski. On necessary and nonconflicting assignments in algorithmic test pattern generation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):515–530, 1994.

[Czutr09] A. Czutro, I. Polian, P. Engelke, S. M. Reddy, and B. Becker. Dynamic compaction in SAT-based ATPG. In *Proceedings of the IEEE Asian Test Symposium (ATS)*, pages 187–190. November 2009.

[Czutr12] A. Czutro, M. Sauer, T. Schubert, I. Polian, and B. Becker. SAT-ATPG using preferences for improved detection of complex defect mechanisms. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 170–175. 2012.

[Davis60] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7(3):201–215, July 1960.

[Davis62] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, July 1962.

[Dike99] C. Dike and E. T. Burton. Miller and noise effects in a synchronizing flip-flop. *IEEE Journal of solid-state circuits*, 34(6):849–855, June 1999.

[Drech98] R. Drechsler and B. Becker. Ordered Kronecker functional decision diagrams—a data structure for representation and manipulation of boolean functions. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 17(10):965–973, October 1998.

[Een03] N. Eén and N. Sörensson. Temporal induction by incremental SAT solving. *Electronic Notes in Theoretical Computer Science*, 89(4):543–560, 2003.

[Eiche64a] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. In *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 111–120. November 1964.

[Eiche64b] E. B. Eichelberger. Hazard detection in combinational and sequential switching circuits. *IBM Journal of Research and Development*, 9(2):90–99, March 1964.

[Eldre59] R. D. Eldred. Test routines based on symbolic logical statements. *Journal of the ACM*, 6(1):33–37, January 1959.

[Elm10] M. Elm, M. A. Kochte, and H.-J. Wunderlich. On determining the real output Xs by SAT-based reasoning. In *Proceedings of the IEEE Asian Test Symposium (ATS)*, pages 39–44. 2010.

[Erb13] D. Erb, M. A. Kochte, M. Sauer, H.-J. Wunderlich, and B. Becker. Accurate multi-cycle ATPG in presence of X-values. In *Proceedings of the IEEE Asian Test Symposium (ATS)*, pages 245–250. 2013.

[Erb14a] D. Erb, M. A. Kochte, M. Sauer, S. Hillebrecht, T. Schubert, H.-J. Wunderlich, and B. Becker. Exact logic and fault simulation in presence of unknowns. *Accepted for publication in ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2014.

[Erb14b] D. Erb, K. Scheibler, M. Sauer, and B. Becker. Efficient SMT-based ATPG for interconnect open defects. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*. 2014. Paper 5.7 (1).

[Feng07] Y. Feng, Z. Zhou, D. Tong, and X. Cheng. Clock domain crossing fault model and coverage metric for validation of SoC design. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1385–1390. EDA Consortium, 2007.

[Fey06] G. Fey, J. Shi, and R. Drechsler. Efficiency of multi-valued encoding in SAT-based ATPG. In *Proceedings of the 36th International Symposium on Multiple-Valued Logic (ISMVL)*, pages 25–25. May 2006.

[Flore98] P. Flores, H. Neto, and J. Marques Silva. An exact solution to the minimum size test pattern problem. In *Proceedings of the International Conference on Computer Design (ICCD)*, pages 510–515. 1998.

[Fujit91] M. Fujita, Y. Matsunaga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level logic synthesis. In *Proceedings of the Conference on European Design Automation (EURO-DAC)*, pages 50–54. IEEE Computer Society Press, 1991.

[Fujiw82] H. Fujiwara and S. Toida. The complexity of fault detection problems for combinational logic circuits. *IEEE Transactions on Computers*, 31(6):555–560, June 1982.

[Fujiw83] H. Fujiwara and T. Shimono. On the acceleration of test generation algorithms. *IEEE Transactions on Computers*, 32(12):1137–1144, December 1983.

[Gaede88] R. Gaede, M. Mercer, K. Butler, and D. Ross. CATAPULT: concurrent automatic testing allowing parallelization and using limited topology. In

Proceedings of the 25th ACM/IEEE Design Automation Conference (DAC), pages 597–600. 1988.

[Gajsk83] D. Gajski and R. Kuhn. New VLSI tools. *Computer Magazine*, pages 11–14, December 1983.

[Gajsk09] D. Gajski, S. Abdi, A. Gerstlauer, and G. Schirner. *Embedded System Design*. Springer, 2009. ISBN 978-1-4419-0503-1.

[Galey61] J. M. Galey, R. E. Norby, and J. P. Roth. Techniques for the diagnosis of switching circuit failures. In *Proceedings of the Second Annual Symposium on Switching Circuit Theory and Logical Design (SWCT)*, pages 152 –160. October 1961.

[Garey79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman Publisher, 1979. ISBN 978-0716710455.

[Goel79] P. Goel and B. C. Rosales. Test generation and dynamic compaction of tests. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 189–192. October 1979.

[Goel80] P. Goel. Test generation costs analysis and projections. In *Proceedings of the 17th Design Automation Conference (DAC)*, pages 77–84. 1980.

[Goel81] P. Goel. An implicit enumeration algorithm to generate tests for combinational logic circuits. *IEEE Transactions on Computers*, C-30(3):215–222, 1981.

[Goldb89] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Professional, 1989. ISBN 978-0201157673.

[Goude91] N. Gouders and R. Kaibel. PARIS: a parallel pattern fault simulator for synchronous sequential circuits. In *Digest of Technical Papers: 1991 IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 542–545. 1991.

[Günth00] W. Günther, N. Drechsler, R. Drechsler, and B. Becker. Verification of designs containing black boxes. In *Proceedings of the 26th Euromicro Conference*, pages 100–105. 2000.

[Hacht96] G. D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, 1996. ISBN 978-0-387-31005-3.

[Hamza98] I. Hamzaoglu and J. H. Patel. New techniques for deterministic test pattern generation. In *Proceedings of the 16th IEEE VLSI Test Symposium (VTS)*, pages 446–452. 1998.

[Harel87] D. Harel and B. Krishnamurthy. Is there hope for linear time fault simulation? In *Proceedings of the 17th International Fault-Tolerant Computing Symposium (FTCS)*, pages 28–33. 1987.

[Hayes86a] J. Hayes. Pseudo-Boolean logic circuits. *IEEE Transactions on Computers*, 100(7):602–612, 1986.

[Hayes86b] J. P. Hayes. Uncertainty, energy, and multiple-valued logics. *IEEE Transactions on Computers*, 35(2):107–114, February 1986.

[Henni64] F. C. Hennie. Fault detecting experiments for sequential circuits. In *Proceedings of the Fifth Annual Symposium on Switching Circuit Theory and Logical Design*, pages 95–110. November 1964.

[Hille12] S. Hillebrecht, M. A. Kochte, H.-J. Wunderlich, and B. Becker. Exact stuck-at fault classification in presence of unknowns. In *Proceedings of the 17th IEEE European Test Symposium (ETS)*, pages 98–103. IEEE Computer Society, 2012.

[Hille13] S. Hillebrecht, M. A. Kochte, D. Erb, H.-J. Wunderlich, and B. Becker. Accurate QBF-based test pattern generation in presence of unknown values. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 436–441. 2013.

[Holst09] S. Holst and H.-J. Wunderlich. A diagnosis algorithm for extreme space compaction. In *Proceedings of the Design Automation and Test in Europe Conference and Exhibition (DATE)*, pages 1355–1360. 2009.

[Hong78] S. J. Hong. Fault simulation strategy for combinational logic networks. In *Digest of Papers: 8th International Symposium on Fault-Tolerant Computing (FTCS)*, pages 96–99. 1978.

[Hsiao97] M. S. Hsiao, E. M. Rudnick, and J. H. Patel. Sequential circuit test generation using dynamic state traversal. In *Proceedings of the European Con-*

ference on Design and Test (EDTC), pages 22–28. IEEE Computer Society, 1997.

[Ibarr75] O. Ibarra and S. Sahni. Polynomially complete fault detection problems. *IEEE Transactions on Computers*, 100(3):242–249, 1975.

[IEEE95] IEEE. IEEE Standard Hardware Description Language Based on the Verilog Hardware Description Language (IEEE Std 1364-1995), 1995. IEEE Computer Society.

[IEEE00] IEEE. IEEE Standard VHDL Language Reference Manual (IEEE Std 1076-2000), 2000. IEEE Computer Society.

[Iguch97] Y. Iguchi, T. Sasao, and M. Matsuura. On properties of Kleene TDDs. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 473–476. January 1997.

[Iguch99] Y. Iguchi, M. Matsuura, T. Sasao, and A. Iseno. Realization of regular ternary logic functions using double-rail logic. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 331–334. 1999.

[Iguch04] Y. Iguchi, T. Sasao, and M. Matsuura. A method to evaluate logic functions in the presence of unknown inputs using LUT cascades. In *Proceedings of the 34th International Symposium on Multiple-Valued Logic*, pages 302–308. May 2004.

[ITRS12] ITRS. International Technology Roadmap for Semiconductors 2012 Update Overview, 2012.

[Iyeng90] V. Iyengar, B. Rosen, and J. Waicukauski. On computing the sizes of detected delay faults. *IEEE Transactions on Computer-Aided Design*, 9(3):299–312, March 1990.

[Jain00] A. Jain, V. Boppana, R. Mukherjee, J. Jain, M. Fujita, and M. Hsiao. Testing, verification, and diagnosis in the presence of unknowns. In *Proceedings IEEE VLSI Test Symposium (VTS)*, pages 263–268. 2000.

[Jenni94] G. Jennings, J. Isaksson, and P. Lindgren. Ordered ternary decision diagrams and the multivalued compiled simulation of unmapped logic. In

Proceedings of the 27th Annual Simulation Symposium, pages 99–105. April 1994.

[Jenni95a] G. Jennings. Accurate ternary-valued compiled logic simulation of complex logic networks by OTDD composition. In *Proceedings of the 28th Annual Simulation Symposium*, pages 303–310. April 1995.

[Jenni95b] G. Jennings. Symbolic incompletely specified functions for correct evaluation in the presence of indeterminate input values. In *Proceedings of the 28th Hawaii International Conference on System Sciences (HICSS)*, pages 23–31. IEEE Computer Society, 1995.

[Kajih04] S. Kajihara, K. K. Saluja, and S. M. Reddy. Enhanced 3-valued logic/fault simulation for full scan circuits using implicit logic values. In *Proceedings of the IEEE European Test Symposium (ETS)*, pages 108–113. 2004.

[Kang03] S. Kang and S. A. Szygenda. Accurate logic simulation by overcoming the unknown value propagation problem. *Simulation*, 79(2):59–68, 2003.

[Kebsc92] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *Proceedings of the 3rd European Conference on Design Automation (EURO-DAC)*, pages 43–47. March 1992.

[Keim96] M. Keim, B. Becker, and B. Stenner. On the (non-)resetability of synchronous sequential circuits. In *Proceedings of the IEEE VLSI Test Symposium (VTS)*, pages 240–245. 1996.

[Keim03] M. Keim. *Symbolic methods for testing digital circuits*. Ph.D. thesis, Albert-Ludwigs-Universität Freiburg im Breisgau, Germany, May 2003.

[Kiefe97] G. Kiefer and H.-J. Wunderlich. Using BIST control for pattern generation. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 347–355. November 1997.

[Kleen52] S. C. Kleene. *Introduction to Metamathematics*. North-Holland Publishing Co., Amsterdam, 1952.

[Kocht08] M. Kochte, C. Zoellin, M. Imhof, and H.-J. Wunderlich. Test set stripping limiting the maximum number of specified bits. In *Proceedings of the 4th IEEE International Symposium on Electronic Design, Test and Applications (DELTA)*, pages 581–586. 2008.

[Kocht09] M. A. Kochte, S. Holst, M. Elm, and H.-J. Wunderlich. Test encoding for extreme response compaction. In *Proceedings of the 14th IEEE European Test Symposium (ETS)*, pages 155–160. 2009.

[Kocht10a] M. A. Kochte, M. Schaal, H.-J. Wunderlich, and C. Zoellin. Efficient fault simulation on many-core processors. In *Proceedings of the 47th ACM/IEEE Design Automation Conference (DAC)*, pages 380–385. June 2010.

[Kocht10b] M. A. Kochte, C. G. Zöllin, and H.-J. Wunderlich. Efficient concurrent self-test with partially specified patterns. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 26(5):581–594, Springer, 2010.

[Kocht11a] M. A. Kochte, S. Kundu, K. Miyase, X. Wen, and H.-J. Wunderlich. Efficient BDD-based fault simulation in presence of unknown values. In *Proceedings of the 20th IEEE Asian Test Symposium (ATS)*, pages 383–388. IEEE Computer Society, 2011.

[Kocht11b] M. A. Kochte and H.-J. Wunderlich. SAT-based fault coverage evaluation in the presence of unknown values. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. 2011.

[Kocht12] M. A. Kochte, M. Elm, and H.-J. Wunderlich. Accurate X-propagation for test applications by SAT-based reasoning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(12):1908–1919, 2012.

[Koene91] B. Koenemann. LFSR-coded test patterns for scan designs. In *Proceedings of the European Test Conference*, pages 237–242. 1991.

[Kuehl97] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 263–268. 1997.

[Kundu91] S. Kundu, I. Nair, L. Huisman, and V. Iyengar. Symbolic implication in test generation. In *Proceedings of the Conference on European Design Automation (EURO-DAC)*, pages 492–496. 1991.

[Kunz94] W. Kunz and D. Pradhan. Recursive learning: a new implication technique for efficient solutions to CAD problems-test, verification, and optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(9):1143–1158, 1994.

[Kunz97] W. Kunz, D. Stoffel, and P. Menon. Logic optimization and equivalence checking by implication analysis. *IEEE Transactions CAD of Integrated Circuits and Systems*, 16(3):266 –281, 1997.

[Larra92] T. Larrabee. Test pattern generation using boolean satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(1):4–15, January 1992.

[Lee59] C. Y. Lee. Representation of switching circuits by binary-decision programs. *Bell Systems Technical Journal*, 38:985–999, 1959.

[Lee91] H. K. Lee and D. S. Ha. An efficient, forward fault simulation algorithm based on the parallel pattern single fault propagation. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 946–955. 1991.

[Lee96] D. Lee and M. Yannakakis. Principles and methods of testing finite state machines—a survey. *Proceedings of the IEEE*, 84(8):1090–1123, 1996.

[Li11] H. Li, D. Xu, and K.-T. Cheng. GPU-accelerated fault simulation and its new applications. In *Proceedings of the International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4. April 2011.

[Lind 97] J. Lind-Nielsen. BuDDy - A binary decision diagram package. Technical University of Denmark, 1997.

[Marin12] P. Marin, C. Miller, M. Lewis, and B. Becker. Verification of partial designs using incremental QBF solving. In *Proceedings of the Design, Automation Test in Europe Conference (DATE)*, pages 623–628. March 2012.

[McClu71] E. McCluskey and F. Clegg. Fault equivalence in combinational logic networks. *IEEE Transactions on Computers*, C-20(11):1286–1293, November 1971.

[Mento08] Mentor Graphics. *Design-for-Test Common Resources Manual (Software Version 8.2008_3)*. Mentor Graphics Corporation, 2008.

[Mille07] D. M. Miller and M. A. Thornton. *Multiple Valued Logic: Concepts and Representations*. Synthesis Lectures on Digital Circuits and Systems. Morgan & Claypool Publishers, 2007. ISBN 978-1598291902.

[Minat90] S. Minato. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Proceedings of the 27th ACE/IEEE Design Automation Conference (DAC)*, pages 52–57. 1990.

[Mitra04] S. Mitra and K. S. Kim. X-compact: an efficient response compaction technique. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(3):421–432, March 2004.

[Mitra05] S. Mitra, M. Mitzenmacher, S. Lumetta, and N. Patil. X-tolerant test response compaction. *IEEE Design & Test of Computers*, 22(6):566–574, Nov-Dec 2005.

[Miyas04] K. Miyase and S. Kajihara. Xid: Don't care identification of test patterns for combinational circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, 23(2):321–326, February 2004.

[Moske01] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: engineering an efficient SAT solver. In *Proceedings of the ACM/IEEE Design Automation Conference (DAC)*, pages 530–535. 2001.

[Mruga05] G. Mrugalski, A. Pogiel, J. Rajski, and J. Tyszer. Diagnosis with convolutional compactors in presence of unknown states. In *Proceedings of the IEEE International Test Conference (ITC)*. 2005. Paper 16.3.

[Mukai86] M. Mukaidono. Regular ternary logic functions—ternary logic functions suitable for treating ambiguity. *IEEE Transactions on Computers*, C-35(2):179–183, 1986.

[Muth76] P. Muth. A nine-valued circuit model for test generation. *IEEE Transactions on Computers*, C-25(6):630–636, June 1976.

[Nag02] P. Nag, A. Gattiker, S. Wei, R. Blanton, and W. Maly. Modeling the economics of testing: a DFT perspective. *IEEE Design & Test of Computers*, 19(1):29–41, Jan-Feb 2002.

[Narus03] M. Naruse, I. Pomeranz, S. Reddy, and S. Kundu. On-chip compression of output responses with unknown values using LFSR reseeding. In *Proceedings of the IEEE International Test Conference (IEEE)*, pages 1060–1068. 2003.

[Nassi00] S. Nassif. Design for variability in DSM technologies. In *Proceedings of the IEEE International Symposium on Quality Electronic Design (ISQED)*, pages 451–454. 2000.

[Nierm92] T. Niermann, W.-T. Cheng, and J. Patel. PROOFS: a fast, memory-efficient sequential circuit fault simulator. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 11(2):198–207, February 1992.

[Noppe07] T. Nopper, C. Scholl, and B. Becker. Computation of minimal counterexamples by using black box techniques and symbolic methods. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 273–280. 2007.

[Ogiha88] T. Ogihara, S. Saruyama, and S. Murai. Test generation for sequential circuits using individual initial value propagation. In *Digest of Technical Papers IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 242–247. 1988.

[Patel03] J. Patel, S. Lumetta, and S. Reddy. Application of Saluja-Karpovsky compactors to test responses with many unknowns. In *Proceedings of the 21st VLSI Test Symposium (VTS)*, pages 107–112. 2003.

[Pigor09] F. Pigorsch and C. Scholl. Exploiting structure in an AIG based QBF solver. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1596–1601. European Design and Automation Association, 2009.

[Pomer97] I. Pomeranz and S. M. Reddy. Fault simulation under the multiple observation time approach using backward implications. In *Proceedings of the IEEE/ACM Design Automation Conference (DAC)*, pages 608–613. 1997.

[Pomer03] I. Pomeranz and S. M. Reddy. Theorems for identifying undetectable faults in partial-scan circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 22(8):1092–1097, 2003.

[Pomer06] I. Pomeranz and S. M. Reddy. Reducing the number of specified values per test vector by increasing the test set size. *IEE Proceedings - Computers and Digital Techniques*, 153(1):39–46, 2006.

[Pomer11] I. Pomeranz and S. M. Reddy. Broadside and functional broadside tests for partial-scan circuits. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems.*, 19(6):1104–1108, 2011.

[Psara10] M. Psarakis, D. Gizopoulos, E. Sanchez, and M. Reorda. Microprocessor software-based self-testing. *IEEE Design & Test of Computers*, 27(3):4–19, 2010.

[Rajsk90] J. Rajski and H. Cox. A method to calculate necessary assignments in algorithmic test pattern generation. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 25–34. IEEE Computer Society, 1990.

[Rajsk04] J. Rajski, J. Tyszer, M. Kassab, and N. Mukherjee. Embedded deterministic test. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems.*, 23(5):776–792, May 2004.

[Rajsk05] J. Rajski, J. Tyszer, C. Wang, and S. Reddy. Finite memory test response compactors for embedded test applications. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 24(4):622–634, 2005.

[Roth66] J. Roth. Diagnosis of automata failures: A calculus and a method. *IBM Journal of Research and Development*, 10(4):278–291, 1966.

[Rudel93] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Proceedings of the 1993 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 42–47. IEEE Computer Society Press, 1993.

[Saluj83] K. K. Saluja and M. Karpovsky. Testing computer hardware through data compression in space and time. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 83–89. 1983.

[Sasao97] T. Sasao. Ternary decision diagrams—survey. In *Proceedings of the 27th International Symposium on Multiple-Valued Logic*, pages 241 –250. May 1997.

[Sasao99] T. Sasao. *Switching theory for logic synthesis*. Kluwer Academic Publishers, 1999. ISBN 0-7923-8456-3.

[Sauer13] M. Sauer, S. Reimer, I. Polian, T. Schubert, and B. Becker. Provably optimal test cube generation using quantified boolean formula solving. In *Proceedings of the IEEE Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 533–539. 2013.

[Saxen03] J. Saxena, K. M. Butler, V. B. Jayaram, S. Kundu, N. V. Arvind, P. Sreeprakash, and M. Hachinger. A case study of IR-drop in structured at-speed testing. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1098–1104. 2003.

[Schub10] T. Schubert, M. Lewis, and B. Becker. Antom—solver description. *SAT Race*, 2010.

[Schul87] M. H. Schultz and F. Brglez. Accelerated transition fault simulation. In *Proceedings of the 24th ACM/IEEE Design Automation Conference (DAC)*, pages 237–243. ACM, New York, NY, USA, 1987.

[Schul88] M. H. Schulz, E. Trischler, and T. M. Sarfert. SOCRATES: a highly efficient automatic test pattern generation system. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 7(1):126–137, 1988.

[Shann38] C. E. Shannon. A symbolic analysis of relay and switching circuits. *Transactions of the American Institute of Electrical Engineers (AIEE)*, 57:713–723, 1938.

[Sharm05] M. Sharma and W.-T. Cheng. X-filter: Filtering unknowns from compacted test responses. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1090–1098. 2005.

[Shi05] J. Shi, G. Fey, R. Drechsler, A. Glowatz, F. Hapke, and J. Schloffel. PASSAT: efficient SAT-based test pattern generation for industrial circuits. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 212–217. 2005.

[Singh00] M. Singh and S. Nowick. Synthesis for logical initializability of synchronous finite-state machines. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 8(5):542–557, October 2000.

[Somen99] F. Somenzi. Binary decision diagrams. In M. Broy and R. Steinbrüggen, editors, *Calculational System Design*, volume 173 of *NATO Science Series*

F.III: Computer and Systems Sciences, pages 303–366. IOS Press, 1999. ISBN 978-90-5199-459-9.

[Srini90] A. Srinivasan, T. Ham, S. Malik, and R. Brayton. Algorithms for discrete function manipulation. In *Digest of Technical Papers: IEEE International Conference on Computer-Aided Design (ICCAD)*, pages 92–95. November 1990.

[Stani91] T. Stanion and D. Bhattacharya. TSUNAMI: a path oriented scheme for algebraic test generation. In *Digest of Papers: 21st International Symposium on Fault-Tolerant Computing (FTCS)*, pages 36–43. 1991.

[Takam85] Y. Takamatsu and K. Kinoshita. An efficient test generation method by 10-V algorithm. In *Proceedings of the IEEE International Symposium Circuits and Systems (ISCAS)*, pages 679–682. 1985.

[Tang06] Y. Tang, H.-J. Wunderlich, H. P. E. Vranken, F. Hapke, M. Wittke, P. Engelke, I. Polian, and B. Becker. X-masking during logic BIST and its impact on defect coverage. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 14(2):442–451, 2006.

[Tani93] S. Tani, K. Hamaguchi, and S. Yajima. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *Proceedings of the 4th International Symposium on Algorithms and Computation (ISAAC'93)*, pages 389–398. Springer, 1993.

[Tille08] D. Tille and R. Drechsler. Incremental SAT instance generation for SAT-based ATPG. In *11th IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 1–6. April 2008.

[Tille10] D. Tille, S. Eggersglüß, R. Krenz-Baath, J. Schlöffel, and R. Drechsler. Improving CNF representations in SAT-based ATPG for industrial circuits using BDDs. In *Proceedings of the 15th IEEE European Test Symposium (ETS)*, pages 176–181. IEEE Computer Society, 2010.

[Touba07] N. Touba. X-canceling MISR - an X-tolerant methodology for compacting output responses with unknowns using a MISR. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1–10. 2007.

[Tseit68] G. Tseitin. On the complexity of derivation in propositional calculus. *Studies in constructive mathematics and mathematical logic*, 2(115-125):10–13, 1968.

[Turpi03] M. Turpin. The dangers of living with an X (bugs hidden in your Verilog). In *Boston Synopsys Users Group Meeting*, pages 1–34. 2003.

[Ulric73] E. G. Ulrich and T. Baker. The concurrent simulation of nearly identical digital networks. In *Proceedings of the 10th Workshop on Design Automation*, pages 145–150. 1973.

[Unger95] S. Unger. Hazards, critical races, and metastability. *IEEE Transactions on Computers*, 44(6):754–768, June 1995.

[Volke03] E. H. Volkerink and S. Mitra. Efficient seed utilization for reseeding based compression. In *Proceedings of the 21st IEEE VLSI Test Symposium (VTS)*, pages 232–237. 2003.

[Wadsa78] R. L. Wadsack. Fault modeling and logic simulation of CMOS and MOS integrated circuits. *Bell System Technical Journal*, 57(5):1449–1474, May-June 1978.

[Waicu85] J. Waicukauski, E. Eichelberger, D. Forlenza, E. Lindbloom, and T. McCarthy. Fault simulation for structured VLSI. *VLSI Systems Design*, 6(12):20–32, December 1985.

[Waicu87] J. Waicukauski, E. Lindbloom, B. Rosen, and V. Iyengar. Transition fault simulation. *IEEE Design & Test of Computers*, 4(2):32–38, 1987.

[Wang06] L.-T. Wang, C.-W. Wu, and X. Wen. *VLSI Test Principles and Architectures: Design for Testability*. Morgan Kaufmann, 2006. ISBN 978-0123705976.

[Wang08a] L.-T. Wang, C. E. Stroud, and N. A. Touba, editors. *System-on-Chip Test Architectures: Nanometer Design for Testability*. Morgan Kaufmann, 2008. ISBN 978-0123739735.

[Wang08b] L.-T. Wang, X. Wen, S. Wu, Z. Wang, Z. Jiang, B. Sheu, and X. Gu. Virtualscan: Test compression technology using combinational logic and one-pass ATPG. *IEEE Design & Test of Computers*, 25(2):122–130, March 2008.

[Wen05] X. Wen, H. Tamamoto, K. K. Saluja, and K. Kinoshita. Fault diagnosis of physical defects using unknown behavior model. *Journal of Computer Science and Technology*, 20(2):187–194, Kluwer Academic Publishers, 2005.

[Wen11] X. Wen, K. Enokimoto, K. Miyase, Y. Yamato, M. Kochte, S. Kajihara, P. Girard, and M. Tehranipoor. Power-aware test generation with guaranteed launch safety for at-speed scan testing. In *Proceedings of the 29th IEEE VLSI Test Symposium (VTS)*, pages 166–171. May 2011.

[Whitt01] J. Whittemore, J. Kim, and K. Sakallah. SATIRE: a new incremental satisfiability engine. In *Proceedings of the 38th IEEE/ACM Annual Design Automation Conference (DAC)*, pages 542–545. ACM, 2001.

[Willi73] M. J. Y. Williams and J. B. Angell. Enhancing testability of large-scale integrated circuits via test points and additional logic. *IEEE Transactions on Computers*, 22(1):46–60, January 1973.

[Wilso00] C. Wilson, D. Dill, and R. Bryant. Symbolic simulation with approximate values. In *Formal Methods in Computer-Aided Design*, volume 1954 of *LNCS*, pages 507–522. Springer, 2000.

[Wohl07] P. Wohl, J. Waicukauski, and S. Ramnath. Fully X-tolerant combinational scan compression. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1–10. 2007.

[Wu11] S. Wu, L.-T. Wang, X. Wen, Z. Jiang, L. Tan, Y. Zhang, Y. Hu, W.-B. Jone, M. Hsiao, J.-M. Li, J.-L. Huang, and L. Yu. Using launch-on-capture for testing scan designs containing synchronous and asynchronous clock domains. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 30(3):455–463, March 2011.

[Wunde89] H. Wunderlich. The design of random-testable sequential circuits. In *Proceedings of the International Symposium on Fault-Tolerant Computing (FTCS)*, pages 110–117. 1989.

[Wunde91] H.-J. Wunderlich. *Hochintegrierte Schaltungen: Prüfgerechter Entwurf und Test*. Springer, 1991. ISBN 978-3540534563.

[Wunde10] H.-J. Wunderlich, editor. *Models in Hardware Testing*. Frontiers in Electronic Testing, Vol. 43. Springer, 2010. ISBN 978-90-481-3281-2.

[Yoeli64] M. Yoeli and S. Rinon. Application of ternary algebra to the study of static hazards. *Journal of the ACM*, 11(1):84–97, January 1964.

[Zhang01] L. Zhang, C. Madigan, M. Moskewicz, and S. Malik. Efficient conflict driven learning in a Boolean satisfiability solver. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, pages 279–285. 2001.

[Zhang02] L. Zhang and S. Malik. Conflict driven learning in a quantified Boolean satisfiability solver. In *Proceedings of the 2002 IEEE/ACM international conference on Computer-aided design (ICCAD)*, pages 442–449. 2002.

[Zoell10] C. G. Zoellin and H.-J. Wunderlich. Low-power test planning for arbitrary at-speed delay-test clock schemes. In *Proceedings of the 28th IEEE VLSI Test Symposium (VTS)*, pages 93–98. 2010.

Appendices

A Sets, Partially Ordered Sets, and Lattices

A.1 Sets and Relations

A set $A = \{e_1, e_2, \ldots, e_n\}$ with $e_i \in A$ for $1 \leq i \leq n$ is a collection of discernible entities e_i, called elements or members. The number of entities in set $|A|$ is called the cardinality of A and can be finite or infinite. The entities are taken from a universe of discourse, or universal set denoted U. The empty set $\emptyset = \{\}$ is the set of no elements.

The set B is called subset of A, written $B \subseteq A$, if all elements of B are also elements of A: $\forall e \in B : e \in A$. B is a proper subset of A, written $B \subset A$, if $B \subseteq A$ and $A \neq B$. In particular, any set A is a subset of the universal set U.

For sets, the unary operation complement as well as the binary operations intersection, union and difference are defined as follows:

- Complement of A in universe U: $\overline{A} = \{x \mid x \in U \wedge x \notin A\}$
- Intersection of sets A and B: $A \cap B = \{x \mid x \in A \wedge x \in B\}$
- Union of sets A and B: $A \cup B = \{x \mid x \in A \vee x \in B\}$
- Difference of set A and set B: $A - B = A \cap \overline{B} = \{x \mid x \in A \wedge x \notin B\}$.

The power set $\mathcal{P}(A)$ of set A is the set which consists of all subsets of A: $\mathcal{P}(A) = \{S \mid S \subseteq A\}$.

The Cartesian product $A \times B$ of two sets A and B is the set of all ordered pairs (a, b) with $a \in A$ and $b \in B$: $A \times B = \{(a, b) \mid a \in A \wedge b \in B\}$. The Cartesian product of n sets $A_1, A_2, \ldots A_n$ is the set of all ordered n-tuples $(a_1, a_2, \ldots a_n)$ with $a_i \in A_i$: $A_1 \times A_2 \times \ldots A_n = \{(a_1, a_2, \ldots a_n) \mid a_i \in A_i, 1 \leq i \leq n\}$.

Relations A binary relation R between two sets A and B is defined as a subset of the Cartesian product of A and B: $R \subseteq A \times B$. Two elements $a \in A$ and $b \in B$ are in relation R, if and only if $(a, b) \in R$, also denoted as aRb. An n-ary relation is defined by a subset of the Cartesian product of n sets.

A binary relation $R \subseteq A \times A$ with identical domain and range is called a relation over A. A relation over A can be represented as a directed graph $G = (V, E)$ where the vertices V represent the elements in A, and the edges $E \subseteq V \times V$ are given by the relation R such that an edge exists between two vertices if and only if the two represented elements of A are in relation R.

Important properties of binary relations over a set A include symmetry and anti-symmetry, transitivity and reflexivity:

Symmetry: Relation $R \subseteq A \times A$ is symmetric if and only if for each tuple $(a, a') \in R$, $(a', a) \in R$ as well.

Anti-symmetry: Relation $R \subseteq A \times A$ is anti-symmetric if and only if $(a, a') \in R$ and $(a', a) \in R$ implies the equivalence of a and a': $a = a'$.

Reflexivity: Relation $R \subseteq A \times A$ is reflexive if and only if for each $a \in A$, $(a, a) \in R$.

Transitivity: Relation $R \subseteq A \times A$ is transitive if and only if $(a, a') \in R$ and $(a', a'') \in R$ implies that a and a'' are in relation: $(a, a'') \in R$.

A symmetric, reflexive and transitive relation is called an *equivalence relation*. An anti-symmetric, reflexive and transitive relation is called a *partial order* and forms the foundation of the predominantly used simulation algorithms for the analysis of the behavior of digital circuits.

A.2 Partially Ordered Sets

A set P over which a partial order $\mathcal{R}_{\leq} \subseteq P \times P$ is defined, is called a partially ordered set $(P, \mathcal{R}_{\leq})$, or poset. The two binary operations *greatest lower bound* and *least upper bound* are defined. Element $m \in P$ is a lower bound of elements $a, b \in P$ if $(m, a) \in \mathcal{R}_{\leq}$ and $(m, b) \in \mathcal{R}_{\leq}$, i.e. m is less than a and b w.r.t. the partial order $\mathcal{R}_{\leq}$. The lower bound m of a and b is the greatest lower bound if and only if for any lower bound m' of a and b, $(m', m) \in \mathcal{R}_{\leq}$. In a similar way, element $m \in P$ is an upper bound of elements a and

b, if $(a, m) \in \mathcal{R}_{\leq}$ and $(b, m) \in \mathcal{R}_{\leq}$. An upper bound m is the least upper bound of a and b, if for any upper bound m' of a and b, $(m, m') \in \mathcal{R}_{\leq}$. In the following, the greatest lower bound operation is denoted by the binary operator $\wedge$, and the least upper bound operation by the binary operator $\vee$.

Since $\mathcal{R}_{\leq}$ is a subset of $P \times P$, some pairs of elements of P may be incomparable. In consequence, the existence of lower and upper bounds of pairs of elements is not guaranteed. Even if an upper or lower bound exists, the existence of their least upper or greatest lower bound depends on $\mathcal{R}_{\leq}$. If the least upper or greatest lower bound exists, they are unique as per their definition.

The elements of a partially ordered set P and their mutual relation can be graphically represented in a directed graph $H = (V, E)$. If only the transitive reduction of the partial order is represented, the graph is called Hasse diagram. The vertices V are the elements of P. The edges E express the order relation $\mathcal{R}_{\leq}$ of P. The edges are implicitly assumed to point upwards, i.e. the direction is expressed by the layout of the graph. If edge (a, b) connects two vertices a and b, then $(a, b) \in \mathcal{R}_{\leq}$. Edges that represent the reflexive or transitive elements of $\mathcal{R}_{\leq}$ are omitted.

Figure A.1 depicts Hasse diagrams of different partially ordered sets. Figure A.1 a) shows a partially ordered set with two elements 0 and 1, for which $\mathcal{R}_{\leq} = \{(0, 1)\}$. The set in Figure A.1 b) has four elements, but elements a and b are not comparable by the partial order. For elements a and b, the greatest lower bound is c, i.e. $a \wedge b = c$, but the pair does not have a least upper bound. In the partially ordered sets depicted in Figure A.1 c) and A.1 d), all pairs of elements have least upper and greatest lower bounds.

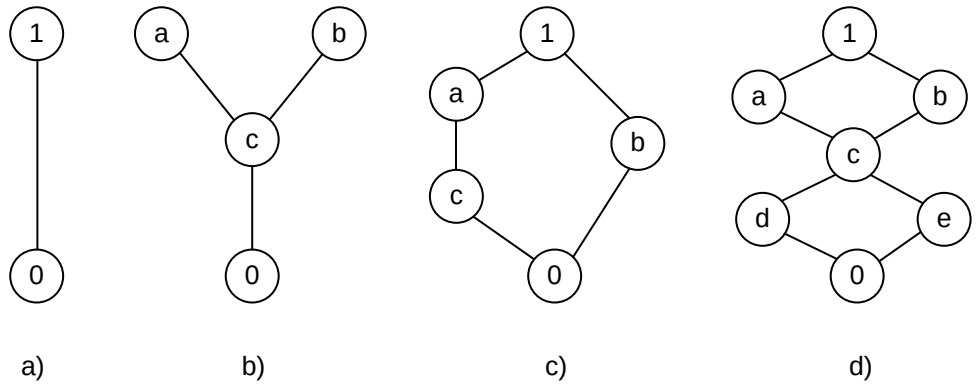


Figure A.1: Hasse diagrams of partially ordered sets

A.3 Lattices

A lattice is a partially ordered set $(L, \mathcal{R}_{\leq})$ in which the least upper bound and greatest lower bound exist for any pair of elements of L. A finite lattice has a unique greatest element $1 \in L$, and a unique least element $0 \in L$. In a lattice, the following four properties hold:

- ▷ Idempotent laws: $a \vee a = a$, $a \wedge a = a$,
- ▷ Commutative laws: $a \vee b = b \vee a$, $a \wedge b = b \wedge a$,
- ▷ Associative laws: $a \vee (b \vee c) = (a \vee b) \vee c$, $a \wedge (b \wedge c) = (a \wedge b) \wedge c$,
- ▷ Absorption laws: $a \wedge (a \vee b) = a$, $a \vee (a \wedge b) = a$.

In Figure A.1, the partially ordered sets a), c) and d) are lattices. Set b) is no lattice because for elements a and b the least upper bound is not defined.

A lattice satisfying the distributive laws below is called a distributive lattice:

- ▷ Distributive laws: $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$, $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$.

Figure A.1 a) and d) are distributive lattices, while the distributive laws do not hold in lattice c).

The element $\neg a \in L$ is called the complement of element $a \in L$ if and only if $a \vee \neg a = 1$ and $a \wedge \neg a = 0$. If for all elements in a lattice a complement exists, then the lattice is complemented. The lattices in Figure A.1 a), c) and d) are complemented lattices, but element b in the lattice c) has no unique complement. In a complemented distributive lattice, the complements are unique.

B Experimental Results

This appendix provides additional details on the experiments and used circuits, as well as tabulated results.

B.1 Implementation

The proposed algorithms (logic and fault simulation based on n-valued, restricted symbolic logic, and BDDs; accurate logic and fault simulation; three-valued ATPG; accurate QBF-based ATPG) are implemented in C. The algorithms use the incremental SAT solver Antom [Schub10] as well as the extended QBF-version of Antom, called Quantom. The approximate BDD-based fault simulation of [Kocht11a] is implemented in Java. The BDD package BuDDy Version 2.4 is used for the BDD operations [Lind 97]. The experiments are conducted on compute servers with Intel Xeon CPUs (3.3 GHz).

B.2 Benchmark Circuits

In the experiments, benchmark circuits from the ISCAS'85 and ISCAS'89 circuit collections are used. In addition, industrial circuits kindly provided by NXP (named *p*k*) are used. The characteristics of these circuits are given in Table B.1.

Table B.1: Benchmark circuit characteristics.

Circuit	Inputs		Gates	Collapsed
	Primary	Pseudo-prim.		stuck-at faults
c2670	233	—	1269	3523
c5315	178	—	1726	5424
c6288	32	—	2416	8704
c7552	207	—	3513	9756
s5378	35	179	2993	7494
s9234	36	211	5597	13892
s13207	62	631	7951	20184
s15850	77	534	9772	24543
s35932	35	1728	16065	51649
s38417	28	1636	22179	56605
s38584	38	1426	19253	53845
p35k	741	2173	31435	110494
p45k	1408	2331	39786	107814
p77k	89	3386	68146	186645
p78k	171	2977	74243	225476
p81k	154	3877	76759	272322
p89k	331	4301	83466	228888
p100k	156	5735	90712	247376
p141k	790	10501	116219	434363
p239k	299	18398	216101	617080
p259k	320	18398	236187	661258
p267k	757	16528	262451	640221
p269k	758	16528	263543	515299
p279k	362	17713	260662	671492
p286k	636	17713	283988	949733
p295k	43	18465	274872	728246
p330k	785	16775	330014	840690
p378k	848	14885	371215	1127364
p388k	942	23789	387454	1287689
p418k	1232	29205	417965	950555
p469k	305	332	48733	179182
p483k	857	32409	431608	1138218

B.3 Logic Simulation in Presence of Unknown Values

Application to Larger Circuits The simulation pessimism of three-valued, restricted symbolic, BDD-based approximate, and accurate SAT-based simulation for an X-ratio of 5% are given in Table B.2. The runtime for the simulation of 1000 random patterns is given in seconds.

Table B.2: Simulation pessimism and runtime of three-valued (Sim-3V), restricted symbolic (RSS), local BDD-based (BDD), and accurate simulation for an X-ratio of 5%.

Circuit	Sim-3V		RSS		BDD		Accurate
	Pess. [%]	Time [s]	Pess. [%]	Time [s]	Pess. [%]	Time [s]	Time [s]
p35k	17.62	0.52	6.92	3.19	6.58	3.84	4.62
p45k	25.95	0.56	8.23	3.40	5.26	5.26	5.68
p77k	36.79	0.85	18.21	5.62	3.88	8.23	12.18
p78k	34.11	0.92	8.28	6.74	1.55	12.72	17.28
p81k	14.27	1.17	7.38	7.95	2.96	12.44	15.96
p89k	13.21	1.08	5.73	6.86	5.48	9.15	9.75
p100k	27.69	1.21	9.26	7.63	3.47	12.19	17.65
p141k	24.03	2.27	7.09	14.34	4.77	21.27	22.05
p267k	14.43	3.54	5.98	23.11	5.41	34.96	34.06
p286k	20.66	4.75	6.88	33.43	5.37	43.41	47.35
p330k	25.66	4.58	6.70	33.84	3.71	50.72	53.30
p378k	34.12	4.85	8.56	38.67	1.62	75.47	171.33
p388k	29.21	6.60	7.61	45.13	2.96	73.66	95.78

Accuracy of Simulation Using Indirect Implications The experiment is conducted as described in [Kajih04]. Table B.3 reports the results for the ISCAS'85 and ISCAS'89 circuits evaluated in [Kajih04]. For circuit s38417, for instance, the method of [Kajih04] increases the outputs with binary value in average by 1.9 outputs per pattern, while accurate simulation identifies 44.7 outputs with binary value over three-valued simulation [Kocht12].

Table B.3: Average increase in outputs with binary value over three-valued simulation computed by [Kajih04] and accurate simulation [Kocht12].

Circuit	Indirect Impl. [Kajih04]	Accurate simulation
c2670	1.0	1.9
c5315	0.2	3.4
s5378	3.0	5.8
s9234	2.4	8.9
s13207	2.5	13.7
s15850	7.0	18.9
s35932	0.1	0.9
s38417	1.9	44.7
s38584	24.0	68.9

B.4 Fault Simulation in Presence of Unknown Values

Application to Larger Circuits For an X-ratio of 5%, Table B.4 shows the increase in stuck-at fault coverage of restricted symbolic logic based, local BDD-based [Kocht11a], and accurate SAT-based fault simulation [Erb14a] w.r.t. three-valued fault simulation. The runtime of these algorithms and the runtime of three-valued fault simulation for the simulation of 1000 random patterns are given in seconds.

Table B.4: Increase in fault coverage (percent points) and runtime of restricted symbolic (RSS), local BDD-based (BDD), and accurate fault simulation w.r.t. three-valued fault simulation (Sim-3V) for an X-ratio of 5%.

Circuit	Sim-3V	RSS		BDD		Accurate	
	Time [s]	Fcov. Inc.	Time [s]	Fcov. Inc.	Time [s]	Fcov. Inc.	Time [s]
p77k	19.9	0.31	687	0.48	723	0.62	26695
p78k	6.4	7.46	448	7.83	476	8.25	1761
p81k	11.2	0.50	620	0.96	1822	1.54	29055
p89k	8.2	0.63	509	0.64	1112	1.56	1881
p100k	10.5	0.93	535	1.35	612	1.74	4292
p141k	22.7	2.32	637	2.39	749	3.11	9889
p267k	29.8	0.88	803	0.88	958	1.31	13719
p295k	45.9	0.95	1011	0.95	1007	8.50	36607
p330k	38.7	2.47	991	2.82	1192	3.82	181308
p378k	33.2	7.10	1009	8.41	1180	9.00	170132
p388k	66.1	1.97	1369	2.33	1552	4.80	201917

Pessimism of Fault Simulation using Indirect Implications Table B.5 shows the average number of additionally detected stuck-at faults per pattern over three-valued fault simulation when using the method of [Kajih04] and the accurate algorithm. For example for circuit c5315, [Kajih04] detects on average 0.4 faults per pattern in addition to three-valued fault simulation. The accurate algorithm is able to classify in average 8.0 additional faults per pattern as detected [Kocht12].

Table B.5: Additionally detected faults (average per pattern) over three-valued fault simulation using [Kajih04] and the accurate algorithm [Kocht12].

Circuit	Indirect Impl. [Kajih04]	Accurate simulation
c2670	5.0	8.3
c5315	0.4	8.0
s5378	6.0	12.8
s9234	3.0	14.7
s13207	3.0	28.5
s15850	16.0	45.7
s35932	0.5	3.7
s38417	10.0	78.5
s38584	27.0	81.8

Accurate Transition Delay Fault Simulation Table B.6 shows the increase in transition delay fault coverage by accurate fault simulation over three-valued fault simulation for an X-ratio of 5%.

Table B.6: Increase of transition delay fault coverage (in percent points) of accurate fault simulation w.r.t. to three-valued fault simulation and runtime of accurate simulation [Erb14a].

Circuit	Fault coverage increase	Runtime [s]
p77k	0.61	366
p78k	12.66	116
p81k	4.83	242
p89k	1.51	587
p100k	1.59	563
p141k	5.64	1451
p267k	1.85	4877
p295k	3.34	7090
p330k	4.82	8523
p378k	12.80	4241
p388k	8.31	17980

B.5 Test Pattern Generation in Presence of Unknown Values

Combinational ATPG Table B.7 lists the increase in stuck-at fault coverage in percent points of the accurate QBF-based ATPG over three-valued ATPG for X-ratios of 1%, 2%, and 5% [Hille13]. The table also lists the runtime of accurate QBF-based ATPG in seconds. For circuit c6288, the X-ratios of 1% and 2% result in the same number of X-sources.

Table B.7: Stuck-at fault coverage increase (in percent points) of accurate QBF-based ATPG w.r.t. to three-valued ATPG and runtime of the QBF-based ATPG.

Circuit	X-ratio 1%		X-ratio 2%		X-ratio 5%	
	Fcov. Inc.	Time [s]	Fcov. Inc.	Time [s]	Fcov. Inc.	Time [s]
c6288	11.06	406	–	–	24.39	1207
c7552	0.08	726	1.98	1026	9.24	3011
s9234	0.53	25	0.63	43	1.41	78
s13207	0.11	44	2.07	64	2.03	71
s38417	0.04	923	0.24	1957	0.53	2698
s38584	0.47	162	0.44	175	0.55	256
p45k	0.52	3801	0.68	5399	1.00	7750
p78k	0.98	4494	2.44	10480	7.38	36928
p100k	0.39	31041	1.24	48143	3.02	77170

Sequential ATPG Table B.8 shows the stuck-at fault coverage of three-valued and accurate QBF-based multi-cycle ATPG for circuit s5378 and an X-ratio of 5%. Up to nine cycles are considered.

Table B.9 shows the increase in stuck-at fault coverage (in percent points) of three-valued and accurate QBF-based multi-cycle ATPG over single-cycle three-valued ATPG for an X-ratio of 5%, considering up to three cycles (column C). The runtime of the accurate multi-cycle QBF-based ATPG is given in seconds.

Table B.8: Stuck-at fault coverage of three-valued and accurate QBF-based multi-cycle ATPG for circuit s5378 and an X-ratio of 5% [Erb13].

Number of cycles	Fault coverage [%]	
	Three-valued	Accurate QBF-based
1	74.5	76.4
2	86.4	87.9
3	86.9	88.2
4	86.9	88.3
5	86.9	88.3
6	86.9	88.3
7	87.0	88.3
8	87.0	88.3
9	87.0	88.3

Table B.9: Fault coverage increase (in percent points) of multi-cycle three-valued and accurate QBF-based ATPG over single-cycle three-valued ATPG for an X-ratio of 5%, and runtime of accurate QBF-based ATPG (in seconds) [Erb13].

Circuit	C	Three-val. multi-cycle ATPG				Accurate multi-cycle ATPG				
		Detect	FC Inc.	UT	Abort	Detect	FC Inc.	UT	Abort	Time
s5378	1	5583	0.00	349	1561	5727	1.91	1767	0	3
	2	6473	11.88	55	966	6585	13.37	901	8	286
	3	6511	12.38	56	927	6611	13.72	776	106	1797
s9234	1	9841	0.00	605	3447	10089	1.79	3198	0	45
	2	10496	4.72	68	3328	11120	9.21	2457	248	3752
	3	10755	6.58	68	3069	11311	10.58	1901	613	8589
s13207	1	15863	0.00	923	3398	16392	2.62	2869	0	15
	2	17341	7.32	125	2718	17804	9.61	2245	11	531
	3	17551	8.36	125	2508	18466	12.89	1285	309	5349
s35932	1	41333	0.00	3064	7252	41333	0.00	7252	0	22
	2	45257	7.60	1996	4396	46291	9.60	3363	0	105
	3	45928	8.90	1999	3722	46745	10.48	2905	0	205
s38417	1	48083	0.00	1625	6897	48349	0.47	6547	84	1257
	2	52916	8.54	34	3655	53280	9.18	2795	496	6578
	3	53621	9.78	34	2950	54865	11.98	883	828	13580
p45k	1	91568	0.00	2412	13834	92830	1.17	12392	181	3535
	2	101976	9.65	92	5746	103247	10.83	1920	2555	32152
	3	103769	11.32	92	3952	104610	12.10	449	2663	35581
p78k	1	192074	0.00	4848	28554	208926	7.47	10897	806	15989
	2	210116	8.00	10	15349	220681	12.69	2824	1961	28199
	3	214751	10.06	10	10715	222450	13.47	892	2124	29869

B.6 Impact on X-Handling Approaches

MATE: Masking Aware Test Set Encoding Table B.10 shows the results after encoding the test patterns using MATE. The table lists the bandwidth requirements of the original circuits without scan chain splitting and test decompressor, and an X-compactor attached to the outputs of the scan chains (column bw_o). The X-compactor is parameterized according to [Mitra04] for vector lengths between 257 and 512, resulting in a signature of 10 bits.

The bandwidth of the MATE approach is given in column bw_M. It is the sum of the required injects to the decompressor (column Inj.) and one parity response bit per scan cycle. The reduction in the required bandwidth (column Δbw) and test time with respect to the original circuit is given in column Δt.

Table B.10: Bandwidth and test time reduction, fault coverage and diagnostic success after applying MATE [Kocht09].

Circuit	bw_o	bw_M	Inj.	Δbw	Δt	Fault Coverage		Diagnostic Success		
				[×]	[×]	Missed	Add. det.	Orig.	MATE	Δ
p100k	24	2	1	12.0	8.2	0	81	84%	80%	–4%
p141k	30	7	6	4.3	5.0	35	8	85%	85%	+0%
p239k	47	2	1	23.5	1.3	0	6	87%	88%	+1%
p259k	47	2	1	23.5	1.2	3	0	80%	82%	+2%
p267k	52	3	2	17.3	1.3	12	16	81%	82%	+1%
p269k	52	3	2	17.3	1.6	15	14	86%	85%	–1%
p279k	62	3	2	20.7	1.1	0	111	77%	82%	+5%
p286k	62	3	2	20.7	1.7	2	12	73%	79%	+6%
p295k	16	3	2	5.3	11.9	1	18	77%	73%	–5%
p330k	71	4	3	17.8	2.7	87	13	81%	79%	–2%
p378k	335	2	1	167.5	0.3	0	1	81%	88%	+7%
p388k	57	3	2	19.0	1.6	0	155	86%	86%	+0%
p418k	71	3	2	23.7	1.5	1	96	82%	83%	+1%
p469k	2	3	2	0.7	7.5	0	0	54%	55%	+1%
p483k	79	3	2	26.3	0.7	1	902	84%	87%	+3%
Avg.			2.1	31.2	2.4			80.9%	82.5%	+1.6%

After applying MATE, only very few faults could not be encoded and remain undetected (column Missed). In some cases, MATE leads to additionally detected faults (column Add. det.) which were previously aborted by ATPG and are now detected since the encoded patterns are filled with different bits by the decompressor.

For a sample of 400 faults, the diagnostic quality of the encoded test set and compacted responses is assessed. The sample faults are selected randomly out of four fault models (stuck-at, crosstalk, transition, single-victim bridge) with 100 instances each. A diagnosis is considered a *success* if there is a single top-candidate which correctly localizes the injected fault. The table compares the diagnostic success rate for the uncompacted response data of the original circuits with the proposed MATE test architecture in the last three columns.

Overmasking in Scan Vector and Scan Chain X-Masking Table B.11 reports the scan chain (vector) overmasking ratio for three X-source distributions (cf. Section 8.2.2) with an X-ratio of 1%.

Table B.11: Overmasking of scan chains (vectors) when using pessimistic three-valued logic simulation for min., avg., max. X-source configuration and an X-ratio of 1% [Kocht12].

Circuit	Overmasking ratio [%]					
	Min. Config.		Avg. Config.		Max. Config.	
	Chains	Vectors	Chains	Vectors	Chains	Vectors
p77k	29.9	8.8	4.0	5.6	2.4	2.9
p78k	17.1	0.8	12.0	1.1	1.8	0.0
p81k	11.3	4.2	7.3	7.4	12.1	7.3
p89k	12.3	30.5	2.4	7.4	3.1	2.0
p100k	8.2	11.6	3.5	6.6	2.2	5.5
p141k	5.6	0.0	18.3	5.0	12.5	3.9
p239k	15.9	0.9	15.3	2.5	13.5	0.9
p259k	32.5	6.9	32.6	6.3	35.0	9.2
p267k	3.4	9.8	0.6	2.9	0.3	0.3
p269k	3.7	3.7	1.3	0.1	0.3	0.5
p279k	4.1	0.8	2.4	1.9	1.7	4.1
p295k	4.2	1.7	2.3	1.0	0.9	0.6
p330k	1.9	0.9	1.6	0.9	1.8	0.5
p378k	13.7	0.0	12.8	0.0	11.2	1.0
p388k	24.0	1.8	24.4	2.8	21.7	2.0
p418k	1.7	2.7	1.8	2.9	1.4	0.1

Error Masking Probability in X-Compact Space Compaction Table B.12 lists the masking probability of $t = 1$, $t = 3$, and $t = 5$ error bits in a scan vector in the X-

compact space compactor for three-valued and accurate analysis of X-propagation in the circuits.

Table B.12: Masking probability [%] of error bits in a scan vector for three-valued and accurate X-propagation analysis of the test set and an X-ratio of 1%.

Circuit	t=1		t=3		t=5	
	Three-val.	Accurate	Three-val.	Accurate	Three-val.	Accurate
p78k	44.5	24.9	12.8	4.3	5.4	1.8
p81k	20.0	6.5	3.1	0.9	1.3	0.6
p89k	5.4	2.9	0.8	0.5	0.5	0.4
p100k	5.0	3.3	0.7	0.5	0.5	0.4
p141k	27.1	18.6	5.0	2.8	2.0	1.2
p239k	48.6	34.5	15.4	7.6	6.7	3.1
p259k	40.5	16.7	10.5	2.4	4.3	1.1
p267k	30.0	25.0	5.9	4.4	2.4	1.8
p269k	28.4	22.5	5.4	3.7	2.2	1.5
p279k	21.1	16.6	3.3	2.4	1.4	1.1
p295k	50.9	48.3	17.1	15.2	7.6	6.6
p330k	29.2	21.4	5.7	3.4	2.3	1.4
p378k	63.7	44.7	29.1	12.9	15.2	5.4
p388k	54.6	31.2	20.1	6.4	9.4	2.5
p418k	38.5	32.6	9.5	6.9	3.8	2.7

C Contributions of the Author

The following paragraphs concisely list the contributions of the author over the state-of-the-art in logic simulation, fault simulation, and test pattern generation in presence of unknown values, as well as in the field of test data compression. The paragraphs also indicate the cooperations involved in this work.

Chapter 5: Logic Simulation in Presence of Unknown Values Novel algorithms for combinational and sequential simulation at gate level have been developed to reduce simulation pessimism in presence of unknown values [Kocht11a] (Section 5.3.4), or even compute the accurate result [Elm10, Kocht12, Erb14a] (Sections 5.2.5, 5.4, and 5.5.3). Accuracy, robustness and scalability of these algorithms have been achieved by integration of both heuristics and formal reasoning based on Boolean satisfiability and binary decision diagrams [Kocht11a, Hille12, Erb14a]. The simulator in [Hille12, Erb14a] has been developed in cooperation with University of Freiburg. The sequential simulator has been mainly developed by Dominik Erb. The developed algorithms allow for the first time to thoroughly assess the simulation pessimism in conventional simulation algorithms even for large circuits (Section 5.6).

Chapter 6: Fault Simulation in Presence of Unknown Values The state-of-the-art in fault simulation has been advanced by novel algorithms that significantly increase the simulation accuracy in presence of unknown values [Kocht11b, Kocht12] (Section 6.5.4), or even allow to trade-off the required computational resources and the accuracy of the result [Kocht11a] (Section 6.5.5). Furthermore, the first accurate fault simulation algorithm has been developed and evaluated [Hille12] (Section 6.4), which provides the accurate fault coverage in presence of unknown values. The algorithms are applied to stuck-at and transition delay fault simulation. An extension to sequential multi-cycle fault simulation has been proposed in [Erb14a] to further increase fault coverage.

The fault simulator in [Hille12] has been developed in cooperation with University of Freiburg. The extension to sequential fault simulation [Erb14a] has been mainly developed by Dominik Erb.

Chapter 7: Test Pattern Generation in Presence of Unknown Values State-of-the-art commercial and academic test pattern generation algorithms are unable to accurately reason about fault testability in presence of unknown values. In this work, test pattern generation in combinational circuits has been mapped to the satisfiability of quantified Boolean formulas (QBF) for the first time [Hille13] (Section 7.4). This allows to generate a test pattern for each testable fault, or otherwise prove its untestability. In an efficient test pattern generation framework, the QBF-based algorithm has been complemented with hybrid two- and three-valued SAT-based test pattern generation, as well as accurate fault simulation to reduce runtime as much as possible. Combinational test generation has been extended to multi-cycle test generation for sequential or partial scan circuits to further increase fault coverage [Erb13] (Section 7.4.7). The proposed algorithms allow to quantify the pessimism in conventional test generation algorithms and to process larger circuits accurately.

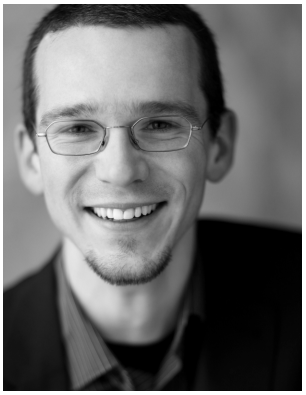
The test pattern generation framework in [Hille13, Erb13] has been developed in cooperation with University of Freiburg. Multi-cycle test generation [Erb13] has been mainly developed by Dominik Erb.

Chapter 8: Application to Test Data Compression and Compaction A test compression algorithm for extreme response compaction [Kocht09] is presented in Section 8.1.4. It uses a heuristic algorithm to identify the input assignments in test patterns that are not required for fault detection [Kocht08]. Such input assignments behave like X-sources in the circuit. The resulting partially specified test set is input to the compression algorithm, which exploits the small number of specified input bits to minimize input and output bandwidth requirements on the test equipment.

Test response compaction is highly sensitive to X-values. A classification of response compaction schemes and X-handling approaches is presented. For the first time, the benefits of accurate reasoning for X-handling approaches is demonstrated for scan chain and vector X-masking schemes [Kocht12] and an X-tolerant compactor (Section 8.2.2). The results reveal a large degree of overmasking of response data by conventional approaches.

Index

Curriculum Vitae of the Author



Michael Kochte received a Diploma degree in Computer Science (Diplom-Informatik) from the University of Stuttgart, Germany in 2005. From 2006 to 2007 he worked as Research Scholar in the IBM Thomas J. Watson Research Center, Yorktown Heights, NY, USA on the test of the Blue Gene/P supercomputer memory system, and the development of distributed cellular applications. In 2007 he joined the Institute of Computer Architecture and Computer Engineering (Institut für Technische Informatik, ITI) at the University of Stuttgart.

The author has been working as a research and teaching assistant under the supervision of Prof. Dr. rer. nat. habil. H.-J. Wunderlich and supported the graduate courses "Design and Test of Systems-on-a-Chip," "Hardware-based Fault Tolerance," and "Hardware Verification and Quality Assessment." He also supervised students in several seminars, Master and Diploma theses, and undergraduate projects.

He has been involved in the research projects "DFG Forschergruppe 460: Development of Concepts and Methods for Reliability Evaluation of Mechatronic Systems in Early Development Phases," "OASIS: Online Failure Prediction for Microelectronic Circuits Using Aging Signatures," and "OTERA: Online Test Strategies for Reliable Reconfigurable Architectures (DFG Priority Project SPP-1500)," supported by the German Research Foundation (DFG). Furthermore, he contributed to the project "VIGONI: Combining Fault Tolerance and Offline Test Strategies for Nanoscaled Electronics," supported by the German Academic Exchange Service (DAAD). From 2010 to 2011, he joined the Department of Creative Informatics of the Kyushu Institute of Technology, Iizuka, Japan as visiting researcher in the group of Prof. X. Wen, Ph.D., supported by a doctoral scholarship of the DAAD. The author has been a member of the program committee of the IEEE Workshop of RTL and High Level Testing (WRTLT) since 2012. His research interests include hardware reliability, test generation and hardware verification.

Publications of the Author

Book Chapters

H.-J. Wunderlich, M. Elm, and M. A. Kochte. Bewertung und Verbesserung der Zuverlässigkeit von mikroelektronischen Komponenten in mechatronischen Systemen, in *Zuverlässigkeit mechatronischer Systeme: Grundlagen und Bewertung in frühen Entwicklungsphasen*, pages 391–464. Springer, 2009. ISBN 978-3540850892.

Journal Publications

1. The Blue Gene Team: Overview of the IBM Blue Gene/P Project. *IBM Journal of Research and Development*, 52(1/2):199-220, 2008.

2. M. A. Kochte, C. G. Zoellin, and H.-J. Wunderlich. Efficient concurrent self-test with partially specified patterns. *Journal of Electronic Testing: Theory and Applications (JETTA)*, 26(5):581–594, Springer, 2010.

3. R. Baranowski, S. Di Carlo, N. Hatami, M. E. Imhof, M. A. Kochte, P. Prinetto, H.-J. Wunderlich, and C. G. Zoellin. Efficient multi-level fault simulation of HW/SW systems for structural faults. *SCIENCE CHINA Information Sciences*, 54(9):1784–1796, 2011.

4. M. A. Kochte, M. Elm, and H.-J. Wunderlich. Accurate X-propagation for test applications by SAT-based reasoning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 31(12):1908–1919, 2012.

5. Y. Yamato, X. Wen, M. A. Kochte, K. Miyase, S. Kajihara, L.-T. Wang. LCTI-SS: Low-Clock-Tree-Impact Scan Segmentation for Avoiding Shift Timing Failures in Scan Testing. *IEEE Design & Test*, 30(4):60-70, 2013.

6. L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, E. Schneider, H. Zhang, J. Henkel, and H.-J. Wunderlich. Test strategies for reliable runtime reconfigurable architectures. *IEEE Transactions on Computers*, 62(8):1494–1507, 2013.

7. D. Erb, M. A. Kochte, M. Sauer, S. Hillebrecht, T. Schubert, H.-J. Wunderlich, and B. Becker. Exact logic and fault simulation in presence of unknowns. *Accepted for publication in ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 2014.

8. A. Herkersdorf, H. Aliee, M. Engel, M. Glaß, C. Gimmler-Dumont, J. Henkel, V. B. Kleeberger, M. A. Kochte, J. M. Kühn, D. Mueller-Gritschneder, S. R. Nassif, H. Rauchfuss, W. Rosenstiel, U. Schlichtmann, M. Shafique, M. B. Tahoori, J. Teich, N. Wehn, C. Weis, and H.-J. Wunderlich. Resilience articulation point (RAP): Cross-layer dependability modeling for nanometer system-on-chip resilience. *Accepted for publication in Elsevier Microelectronics Reliability Journal (In press)*, 2014.

Conference Proceedings

1. M. A. Kochte, R. Baranowski, and H.-J. Wunderlich. Zur Zuverlässigkeitsmodellierung von Hardware-Software-Systemen. In *2. GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf (ZuE)*, volume 57, pages 83–90. VDE VERLAG GMBH, 2008.

2. M. A. Kochte, C. G. Zoellin, M. E. Imhof, and H.-J. Wunderlich. Test set stripping limiting the maximum number of specified bits. In *Proceedings of the 4th IEEE International Symposium on Electronic Design, Test and Applications (DELTA)*, pages 581–586. 2008.

3. M. A. Kochte and R. Natarajan. A framework for scheduling parallel DBMS user-defined programs on an attached high-performance computer. In *Proceedings of the Conference on Computing frontiers (CF'08)*, pages 97–104. ACM, 2008.

4. M. A. Kochte, C. G. Zoellin, M. E. Imhof, R. Salimi Khaligh, M. Radetzki, H.-J. Wunderlich, S. Di Carlo, and P. Prinetto. Test exploration and validation using transaction level models. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 1250–1253. IEEE Computer Society, 2009.

5. M. A. Kochte, S. Holst, M. Elm, and H.-J. Wunderlich. Test encoding for extreme response compaction. In *Proceedings of the 14th IEEE European Test Symposium (ETS)*, pages 155–160. 2009.

6. M. A. Kochte, C. G. Zoellin, and H.-J. Wunderlich. Concurrent self-test with partially specified patterns for low test latency and overhead. In *Proceedings of the 14th IEEE European Test Symposium (ETS)*, pages 53–58. IEEE Computer Society, 2009.

7. M. A. Kochte, M. Schaal, H.-J. Wunderlich, and C. G. Zoellin. Efficient fault simulation on many-core processors. In *Proceedings 47th ACM/IEEE Design Automation Conference (DAC)*, pages 380–385. June 2010.

8. M. A. Kochte, C. G. Zoellin, R. Baranowski, M. E. Imhof, H.-J. Wunderlich, N. Hatami, S. Di Carlo, and P. Prinetto. Effiziente Simulation von strukturellen Fehlern für die Zuverlässigkeitsanalyse auf Systemebene. In *4. GMM/GI/ITG-Fachtagung Zuverlässigkeit und Entwurf (ZuE)*, volume 66, pages 25–32. VDE VERLAG GMBH, 2010.

9. M. A. Kochte, C. G. Zoellin, R. Baranowski, M. E. Imhof, H.-J. Wunderlich, N. Hatami, S. Di Carlo, and P. Prinetto. Efficient simulation of structural faults for the reliability evaluation at system-level. In *Proceedings of the IEEE 19th Asian Test Symposium (ATS)*, pages 3–8. IEEE Computer Society, 2010.

10. M. Elm, M. A. Kochte, and H.-J. Wunderlich. On determining the real output Xs by SAT-based reasoning. In *Proceedings of the IEEE Asian Test Symposium (ATS)*, pages 39–44. 2010.

11. M. A. Kochte, C. G. Zoellin, R. Baranowski, M. E. Imhof, H.-J. Wunderlich, N. Hatami, S. Di Carlo, and P. Prinetto. System reliability evaluation using concurrent multi-level simulation of structural faults. In *Proceedings of the IEEE International Test Conference (ITC)*. IEEE Computer Society, 2010.

12. M. Kochte and H. Wunderlich. SAT-based fault coverage evaluation in the presence of unknown values. In *Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1–6. 2011.

13. X. Wen, K. Enokimoto, K. Miyase, Y. Yamato, M. A. Kochte, S. Kajihara, P. Girard, and M. Tehranipoor. Power-aware test generation with guaranteed launch safety for at-speed scan testing. In *Proceedings of the 29th IEEE VLSI Test Symposium (VTS)*, pages 166–171. 2011.

14. Y. Yamato, X. Wen, M. A. Kochte, K. Miyase, S. Kajihara, and L.-T. Wang. A novel scan segmentation design method for avoiding shift timing failures in scan testing. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1–8. IEEE Computer Society, 2011.

15. M. A. Kochte, S. Kundu, K. Miyase, X. Wen, and H.-J. Wunderlich. Efficient BDD-based fault simulation in presence of unknown values. In *Proceedings of the 20th IEEE Asian Test Symposium (ATS)*, pages 383–388. IEEE Computer Society, 2011.

16. M. A. Kochte, K. Miyase, X. Wen, S. Kajihara, Y. Yamato, K. Enokimoto, and H.-J. Wunderlich. SAT-based capture-power reduction for at-speed broadcast-scan-based test compression architectures. In *Proceedings of the 17th IEEE/ACM International Symposium on Low Power Electronics and Design (ISLPED)*, pages 33–38. IEEE Computer Society, 2011.

17. S. Hillebrecht, M. A. Kochte, H.-J. Wunderlich, and B. Becker. Exact stuck-at fault classification in presence of unknowns. In *Proceedings of the 17th IEEE European Test Symposium (ETS)*, pages 98–103. IEEE Computer Society, 2012.

18. M. S. Abdelfattah, L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, H. Zhang, J. Henkel, and H.-J. Wunderlich. Transparent structural online test for reconfigurable systems. In *Proceedings of the 18th IEEE International On-Line Testing Symposium (IOLTS)*, pages 37–42. IEEE Computer Society, 2012.

19. L. Bauer, C. Braun, M. E. Imhof, M. A. Kochte, H. Zhang, H.-J. Wunderlich, and J. Henkel. OTERA: Online test strategies for reliable reconfigurable architectures. In *Proceedings of the NASA/ESA Conference on Adaptive Hardware and Systems (AHS)*, pages 38–45. IEEE Computer Society, 2012.

20. R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Modeling, verification and pattern generation for reconfigurable scan networks. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1–9. IEEE Computer Society, 2012.

21. S. Hillebrecht, M. A. Kochte, D. Erb, H.-J. Wunderlich, and B. Becker. Accurate QBF-based test pattern generation in presence of unknown values. In *Proceedings of the Conference on Design, Automation and Test in Europe (DATE)*, pages 436–441. 2013.

22. R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Scan pattern retargeting and merging with reduced access time. In *Proceedings of IEEE European Test Symposium (ETS)*, pages 39–45. IEEE Computer Society, 2013.

23. H. Zhang, L. Bauer, M. A. Kochte, E. Schneider, C. Braun, M. E. Imhof, H.-J. Wunderlich, and J. Henkel. Module diversification: Fault tolerance and aging mitigation for runtime reconfigurable architectures. In *Proceedings of the IEEE International Test Conference (ITC)*, pages 1–10. 2013.

24. A. Dalirsani, M. A. Kochte, and H.-J. Wunderlich. SAT-based code synthesis for fault-secure circuits. In *Proceedings 16th IEEE Symp. Defect and Fault Tolerance in VLSI and Nanotechnology Systems (DFTS)*, pages 38–44. 2013.

25. D. Erb, M. A. Kochte, M. Sauer, H.-J. Wunderlich, and B. Becker. Accurate multi-cycle ATPG in presence of X-values. In *Proceedings of the 22nd IEEE Asian Test Symposium (ATS)*, pages 245–250. 2013.

26. R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Securing access to reconfigurable scan networks. In *Proceedings of the 22nd IEEE Asian Test Symposium (ATS)*, pages 295–300. 2013.

27. R. Baranowski, M. A. Kochte, and H.-J. Wunderlich. Verifikation rekonfigurierbarer Scan-Netze. In *Proceedings of the 17. Workshop Methoden und Beschreibungssprachen zur Modellierung und Verifikation von Schaltungen und Systemen (MBMV)*, pages 137–146. 2014.

28. H. Zhang, M. A. Kochte, M. E. Imhof, L. Bauer, H.-J. Wunderlich, and J. Henkel. GUARD: Guaranteed reliability in dynamically reconfigurable systems. To appear in *Proceedings of the 51st ACM/IEEE Design Automation Conference (DAC)*. 2014.

Declaration

All the work contained within this thesis, except where otherwise acknowledged, was solely the effort of the author. At no stage was any collaboration entered into with any other party.

Michael A. Kochte