

Stephan Eggersgluß · Rolf Drechsler

High Quality Test Pattern Generation and Boolean Satisfiability

 Springer

High Quality Test Pattern Generation and Boolean Satisfiability

Stephan Eggersgluß • Rolf Drechsler

High Quality Test Pattern Generation and Boolean Satisfiability

 Springer

Stephan Eggersglüß
University of Bremen
Bremen, Germany
German Research Center for Artificial
Intelligence (DFKI) – Cyber-Physical
Systems, Bremen, Germany

Rolf Drechsler
University of Bremen
Bremen, Germany
German Research Center for Artificial
Intelligence (DFKI) – Cyber-Physical
Systems, Bremen, Germany

ISBN 978-1-4419-9975-7 e-ISBN 978-1-4419-9976-4
DOI 10.1007/978-1-4419-9976-4
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2012930124

© Springer Science+Business Media, LLC 2012

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The content of the book describes work that has been carried out in the Group of Computer Architecture at the University of Bremen, Germany over the last 5 years. Therefore, we would like to thank all the members of the group for their valuable help. Special thanks go to Daniel Tille and Görschwin Fey for many helpful discussions and support. Both are also co-authors of our previously published book *Test Pattern Generation using Boolean Proof Engines* which describes among other things the basic principles on which the content of this book is based on.

Various chapters of this book are based on scientific papers. Therefore, we would like to acknowledge the work of the co-authors of these papers Görschwin Fey, Hoang M. Le, Juergen Schloeffel and Daniel Tille. Since large parts of the work has been done in collaboration, our special thanks go to the Mentor Graphics Development group in Hamburg, Germany, especially to René Krenz-Bååth (now Hochschule Hamm-Lippstadt). Finally, we would like to thank Lisa Jungmann for her help with the cover design as well as Robert Wille, Judith End and Tom Gmeinder for proof-reading.

Parts of this research work were supported by the German Federal Ministry of Education and Research (BMBF) in the Project MAYA under contract number 01M3172B, by the German Research Foundation (DFG) under contract number DR 287/15-1 and by the Central Research Promotion (ZF) of the University of Bremen under contract number 03/107/05. The authors like to thank these institutions for their support.

Bremen

Stephan Eggersgluß
Rolf Drechsler

Contents

1	Introduction	1
Part I Preliminaries and Previous Work		
2	Circuits and Testing	11
2.1	Post-Production Test	11
2.2	Circuits	14
2.2.1	Scan-Based Testing	15
2.3	Fault Models	16
2.3.1	Stuck-at	17
2.3.2	Delay	20
2.4	Classical ATPG Algorithms	25
2.4.1	ATPG for Stuck-at Faults	25
2.4.2	ATPG for Delay Faults	32
2.5	Industrial Test Environment	36
3	Boolean Satisfiability	41
3.1	Boolean Algebra	41
3.2	SAT Solver	42
3.2.1	DLL-Algorithm	43
3.3	Advanced SAT Techniques	44
3.3.1	Fast Boolean Constraint Propagation	45
3.3.2	Conflict Analysis	46
3.3.3	Conflict-Driven Heuristics	49
3.3.4	Incremental SAT	51
3.3.5	Restarts	52
3.4	Circuit-to-CNF Transformation	52
3.5	Circuit-Oriented SAT and Observability Don't Cares	54
3.5.1	Exploitation of Observability Don't Cares	56

4	ATPG Based on Boolean Satisfiability	59
4.1	SAT-Based ATPG for Boolean Circuits	60
4.1.1	SAT Formulation: Stuck-at Fault Model	60
4.1.2	SAT-Based ATPG Techniques	63
4.2	SAT-Based ATPG for Industrial Circuits	66
4.2.1	Multiple-Valued Logic	66
4.2.2	Hybrid Logic	68
4.2.3	Improving Compactness	68
4.3	Combination with Structural Algorithm	69
Part II New SAT Techniques and their Application in ATPG		
5	Dynamic Clause Activation	73
5.1	Overall Framework for Dynamic Clause Activation	74
5.2	Efficient Activation Methodology	77
5.2.1	Consistent SAT Instance	78
5.2.2	Structural Watch List	79
5.3	Literal-Based Activation	82
5.4	Implicit Observability Don't Cares	84
5.4.1	Clause-Based J-Frontier	85
5.4.2	SCOAP-Based Decision Heuristic	88
5.5	Classical SAT Solver Emulation	89
5.6	SAT Formulation for Test Generation Using DCA	91
5.6.1	Dynamic Clause Activation and Multiple-Valued Logic	93
5.7	Experimental Results	94
5.7.1	Results for the Stuck-at Fault Model	95
5.7.2	Results for the Path Delay Fault Model	98
5.7.3	Results for Industrial Circuits	100
5.8	Summary	105
6	Circuit-Based Dynamic Learning	107
6.1	Integration of Dynamic Learning	108
6.1.1	Pervasive Conflict Clause Identification	108
6.1.2	Variable-Based Activation	110
6.1.3	Combination with Dynamic Clause Activation	111
6.2	Post-Classification Phase	112
6.3	Learning Strategies	113
6.4	Improved SAT Solving Engine	114
6.5	Experimental Results	115
6.5.1	Dynamic Learning for Stuck-at Faults	116
6.5.2	Dynamic Learning for Path Delay Faults	118
6.5.3	Dynamic Learning for Industrial Circuits	119
6.5.4	Combination of Structural and SAT-Based Algorithms	122
6.6	Summary	123

Part III High Quality Delay Test Generation

7 High Quality ATPG for Transition Faults 127

 7.1 Transition Fault Model: SAT Formulation 128

 7.1.1 Iterative Logic Array 128

 7.1.2 Injection of Stuck-at Faults 129

 7.1.3 Experimental Results 131

 7.2 Long Propagation Paths 134

 7.2.1 Incremental Instance Generation 135

 7.2.2 Output Ordering 136

 7.2.3 Experimental Results 137

 7.3 Timing-Aware ATPG 140

 7.3.1 Motivational Example 140

 7.3.2 Pseudo-Boolean Optimization 142

 7.3.3 PBO Formulation: Timing-Aware ATPG 143

 7.3.4 Using Structural Information 149

 7.3.5 Considering Transition-Dependent Delays 150

 7.3.6 Experimental Results 151

 7.4 Summary 152

8 Path Delay Fault Model 155

 8.1 Related Work 156

 8.1.1 Robust Tests in Combinational Circuits 157

 8.1.2 Incremental SAT and Learning 157

 8.1.3 PDF Test Generation Using CSAT 158

 8.2 Non-Robust Test Pattern Generation 158

 8.3 Robust Test Pattern Generation 160

 8.4 SAT Instance Generation Flow 162

 8.5 As-Robust-As-Possible Test Generation 163

 8.5.1 Test Generation for ARAP Tests 166

 8.5.2 Incremental SAT Formulation for Static Value Justification .. 168

 8.5.3 Considering the Presence of Small Delay Defects 170

 8.6 Experimental Results 172

 8.6.1 Comparison with Competitive Approach 173

 8.6.2 SAT Instance Generation Flow 173

 8.6.3 MONSOON Using DynamicSAT 175

 8.6.4 ARAP Test Generation 177

 8.7 Summary 179

9 Summary and Outlook 181

References 183

Index 191

List of Figures

Fig. 1.1	SAT application flow	3
Fig. 1.2	Developed SAT-based ATPG framework for industrial application	5
Fig. 2.1	Circuit under test	12
Fig. 2.2	Simple representation of an ATPG system	13
Fig. 2.3	Basic gates	15
Fig. 2.4	Sequential circuit	16
Fig. 2.5	SAFM example. (a) Correct circuit, (b) stuck-at-0 fault on branch b and (c) stuck-at-0 fault on branch c	18
Fig. 2.6	Boolean difference	19
Fig. 2.7	Fault collapsing. (a) Uncollapsed fault set and (b) collapsed fault set	20
Fig. 2.8	Non-robust and robust sensitization. (a) Non-robust and (b) robust	23
Fig. 2.9	Propagation paths – transition fault	24
Fig. 2.10	D-algorithm steps. (a) D-drive and (b) consistency	27
Fig. 2.11	Learning of indirect implications. (a) Direct implication of $b = 1$ and (b) indirect implication of $f = 0$	30
Fig. 2.12	Hasse diagram of ten-valued logic [FWA93]	34
Fig. 2.13	Flow in industrial test environment	37
Fig. 2.14	ATPG phase – low compaction	38
Fig. 2.15	ATPG phase – high compaction	39
Fig. 3.1	Watch list example. (a) Initial and (b) Updated	46
Fig. 3.2	Example CNF with implication graph. (a) CNF Φ and (b) Implication graph	48
Fig. 3.3	Implication graph for non-chronological backtracking	49
Fig. 3.4	Example circuit \mathcal{C} for circuit-to-CNF transformation	54
Fig. 3.5	Example circuit with ODCs	56

Fig. 4.1	Influenced circuit parts.....	60
Fig. 4.2	SAT instance for ATPG.....	61
Fig. 4.3	Example for SAT formulation for the SAFM. (a) Good circuit, (b) faulty circuit, and (c) boolean difference.....	62
Fig. 4.4	Combination of structural and SAT-based algorithm. (a) Classic and (b) SAT integration.....	70
Fig. 5.1	Division of the proposed SAT engine.....	75
Fig. 5.2	SAT-based ATPG flow. (a) Classic and (b) dynamic.....	77
Fig. 5.3	Transitive fanin cone with fanouts.....	78
Fig. 5.4	Example circuit \mathcal{C} for dynamic clause activation.....	80
Fig. 5.5	Schematic view of the SWL.....	81
Fig. 5.6	Example implication graph for search process using DCA.....	81
Fig. 5.7	Schematic view of the SWL for literal-based activation.....	84
Fig. 5.8	Example circuit C with IODCs.....	85
Fig. 5.9	Example – J-stack. (a) Conflicting assignment and (b) valid assignment.....	87
Fig. 5.10	SAT emulation flow.....	90
Fig. 5.11	SAT formulation for test generation using DCA.....	92
Fig. 5.12	Example circuit for DCA formulation. (a) Good circuit \mathcal{C}_G and (b) faulty circuit \mathcal{C}_F	92
Fig. 6.1	Illustration of a watch list example.....	110
Fig. 6.2	Example circuit \mathcal{C} for dynamic learning.....	112
Fig. 6.3	Progress of (active) learned clauses – p57k.....	113
Fig. 6.4	Improved search engine.....	115
Fig. 7.1	Example circuit with TF.....	130
Fig. 7.2	Unrolled example circuit with stuck-at fault injection.....	130
Fig. 7.3	Incremental SAT instance generation.....	136
Fig. 7.4	Example circuit for timing-aware ATPG.....	141
Fig. 7.5	d- and j-variables in PB-SAT transformation.....	145
Fig. 7.6	Multiple paths.....	148
Fig. 7.7	Origin of transitions.....	149
Fig. 7.8	Using FFRs.....	149
Fig. 8.1	Example circuit \mathcal{C} for non-robust test generation.....	159
Fig. 8.2	Guaranteeing static values. (a) Boolean modeling and (b) explicit static values.....	160
Fig. 8.3	SAT instance generation flow.....	164
Fig. 8.4	Tests for path $a - d - g - j - m$. (a) Non-robust test; (b) ARAP test.....	165
Fig. 8.5	ARAP test for path $a - d - g - j - m$ considering SDDs.....	171

List of Tables

Table 2.1	Input vectors	19
Table 2.2	Sensitization criteria for robust and non-robust test patterns	22
Table 2.3	Meaning of the values of \mathcal{L}_5	26
Table 3.1	Truth tables of logical operators (a) Unary (b) Binary	42
Table 3.2	Truth table of NAND	53
Table 4.1	Truth table for an AND gate in \mathcal{L}_4	67
Table 4.2	Boolean encoding for \mathcal{L}_4	67
Table 4.3	CNF representation of an AND gate using $\eta_{\mathcal{L}_4}$	67
Table 4.4	CNF size for a 2-input AND gate	68
Table 5.1	CNF and SWL entries for the example circuit \mathcal{C} in Fig. 5.4	80
Table 5.2	SWL entries for literal-based activation for circuit \mathcal{C} in Fig. 5.4	83
Table 5.3	CNF and SWL entries for a BUS gate using $\eta_{\mathcal{L}_4}$	94
Table 5.4	Experimental results for the SAFM	96
Table 5.5	Experimental results for the SAFM – DynamicSAT	97
Table 5.6	Experimental results for the PDFM	99
Table 5.7	Statistical information about industrial circuits – input/output	100
Table 5.8	Statistical information about industrial circuits – size	101
Table 5.9	Experimental results – specified bits	102
Table 5.10	Experimental results – competitive approaches	103
Table 5.11	Experimental results – dynamic clause activation	104
Table 6.1	Experimental results for circuit-based dynamic learning – stuck-at	117
Table 6.2	Number of classified faults with respect to the number of needed restarts for b18	117
Table 6.3	Experimental results for circuit-based dynamic learning – path delay	118

Table 6.4	Summary of the experimental evaluation of all learning strategies	120
Table 6.5	Experimental results – dynamic learning	121
Table 6.6	Experimental results – post-classification phase	122
Table 6.7	Experimental results – combination	123
Table 7.1	Experimental results – transition faults – FAN/PASSAT	132
Table 7.2	Experimental results – transition faults – DynamicSAT+ (Hybrid)	133
Table 7.3	Impact on fault coverage/fault efficiency	134
Table 7.4	Experimental results – long propagation paths	138
Table 7.5	Experimental results – long propagation paths – n -steps	139
Table 7.6	Pseudo-Boolean and CNF representation for an AND gate $a \cdot b = c$	142
Table 7.7	PB representation of implications	147
Table 7.8	Experimental results – run time	152
Table 7.9	Experimental results – path length	153
Table 8.1	Encoding of \mathcal{L}_7 [CG96]	157
Table 8.2	CNF representation of an AND gate using $\eta_{\mathcal{L}_7}$ [CG96]	157
Table 8.3	Truth table for an AND gate in \mathcal{L}_{6s}	161
Table 8.4	Boolean encoding $\eta_{\mathcal{L}_{19s}}$ for \mathcal{L}_{19s}	162
Table 8.5	Multiple-valued logics and logic mapping	163
Table 8.6	CNF size of incremental SAT formulation	169
Table 8.7	Boolean encoding $\eta_{\mathcal{L}_{16}}$ for \mathcal{L}_{16}	169
Table 8.8	CNF for an AND gate using $\eta_{\mathcal{L}_{AB}}$	170
Table 8.9	CNF description for static value justification for an AND gate using $\eta_{\mathcal{L}_{AB}}$	170
Table 8.10	Run time comparison with competitive approach – non-robust	173
Table 8.11	Information about the industrial circuits	174
Table 8.12	Robust path delay test generation	175
Table 8.13	Experimental results – MONSOON using DynamicSAT – non-robust	176
Table 8.14	Experimental results – MONSOON using DynamicSAT – robust	176
Table 8.15	Experimental results – run time	177
Table 8.16	Experimental results – ARAP tests	178

List of Acronyms

ARAP	As-Robust-As-Possible
ATE	Automatic Test Equipment
ATPG	Automatic Test Pattern Generation
BCP	Boolean Constraint Propagation
BDD	Binary Decision Diagram
CNF	Conjunctive Normal Form
CUT	Circuit Under Test
DCA	Dynamic Clause Activation
DFT	Design-For-Test
DTPG	Deterministic Test Pattern Generation
GDFM	Gate Delay Fault Model
IC	Integrated Circuit
IG	Implication Graph
ISAT	Incremental SAT
LC	Logic Class
LWL	Learned Watch List
ODC	Observability Don't Care
PBO	Pseudo-Boolean Optimization
PB-SAT	Pseudo-Boolean SAT
PDF	Path Delay Fault
PDFM	Path Delay Fault Model
PI	Primary Input
PO	Primary Output
PPI	Pseudo Primary Input
PPO	Pseudo Primary Output
RTPG	Random Test Pattern Generation
s-a-0	Stuck-at-0
s-a-1	Stuck-at-1
SAFM	Stuck-At Fault Model
SAT	Boolean Satisfiability, Satisfiable

SDD	Small Delay Defect
SWL	Structural Watch List
TF	Transition Fault
TFM	Transition Fault Model
UNSAT	Unsatisfiable

List of Symbols

\downarrow	falling transition
\uparrow	rising transition
\cdot	Boolean AND operator
$+$	Boolean OR operator
\odot	resolution operator
\oplus	Boolean XOR operator
$\bar{\cdot}$	Boolean NOT of \cdot
\rightarrow	implies
\leftrightarrow	equivalence
Φ	CNF, set of clauses
Φ_{dyn}	dynamically extended CNF
Φ_F	fault-specific constraints
Ψ	set of pseudo-Boolean constraints
ψ	pseudo-Boolean constraint of Ψ
η	Boolean encoding
κ	conflict
λ	arbitrary literal
ω	clause of Φ
ω_C	conflict clause
$@_x$	at decision level x
\mathbb{B}	set of Boolean values $\{0, 1\}$
\mathcal{C}	circuit
\mathcal{F}	set of flip-flops
F	fault
f	flip-flop $\in \mathcal{C}$, faulty line of gate
$\mathcal{F}(g)$	transitive fanin of g
\mathcal{G}	set of gates
g	gate $\in \mathcal{C}$
h	successor gate of g
\mathcal{I}	set of primary inputs
i_1, \dots, i_n	primary inputs of \mathcal{C}

\mathcal{J}	J-stack
\mathcal{L}_x	multiple-valued logic with x values
\mathcal{O}	set of primary outputs
o_1, \dots, o_m	primary outputs of \mathcal{C}
\mathcal{P}	structural path of \mathcal{C}
\mathcal{S}	set of signal lines or connections
s	signal line, connection
t_i	initial (current) time frame
t_{i+1}	final (next) time frame
V	vector
v	(Boolean) value
X	set of Boolean variables, don't care value
x_1, \dots, x_n	Boolean variables

Chapter 1

Introduction

The *Integrated Circuit* (IC) was invented in the 1950s. At the beginning, ICs were mainly used in computers. With the advancing miniaturization of the components, the significance of ICs as part of our daily life grows. Many consumer products such as mobile music players or cell phones use ICs (or “chips”) as core engines. A failure of these devices usually results in problems of lesser extent for the owner. But today, ICs also control safety critical applications. For example, chips are responsible for the correct mode of operation in car control systems, avionics or medical equipment. A failure of a chip can be life-threatening in the worst-case. Consequently, the correct mode of operation of a fabricated chip is crucial.

Due to the ever shrinking component sizes of today’s designs, the vulnerability of chips to flaws in the manufacturing process increases. The IC manufacturers put much effort in guaranteeing the integrity of their products. A large part of the manufacturing costs is spent for the detection of defects caused by the manufacturing process. Every fabricated chip is subjected to a *post-production test* (or *manufacturing test*) to avoid that defective chips are delivered to customers (and by this could cause failures in operation mode). Thus, the purpose of such a post-production test is to detect any defects caused by the manufacturing process.

Stimuli are applied to the inputs of the *Circuit Under Test* (CUT) during this test. The output responses are monitored. If one or more of the output responses are inconsistent with the specification, the chip will be rejected as erroneous. However, the complexity of modern designs does not allow for a complete test of all possible input stimuli. A complete test for each fabricated chip would be far too time-consuming or costly, since the number of possible tests is exponential in the number of inputs. Instead, a test set is pre-computed that covers a large range of possible defects. Logical fault models are used to abstract from physical defects. The fault model most widespread is the *Stuck-at Fault Model* (SAFM) [Eld59].

This test set is applied to each fabricated chip by *Automatic Test Equipment* (ATE). Because the memory and bandwidth of an ATE is limited, the applied

test set has to be as small as possible. A large test set size signifies not only a long test application time but also immense test costs. The computation of the test set, which is known as *Automatic Test Pattern Generation* (ATPG), is the main subject of this book. Due to the large number of potential faults, ATPG is a computationally intensive task and fast algorithms are needed for obtaining a test set in acceptable run time.

Classical ATPG algorithms are mostly based on the D-algorithm proposed by Roth in 1966 [Rot66]. These algorithms work directly on the circuit structure, i.e. a flat gate-level netlist, and typically benefit significantly from their knowledge about the structure of the problem. Several techniques and powerful heuristics have been proposed over the years to improve the effectiveness of ATPG. At the turn of the millennium, the ATPG problem was considered to be solved. The existing algorithms were fast enough and provided sufficient fault coverage. This has changed in the last years.

The size of new designs doubles every 18 months according to Moore's law [Moo65]. Today's circuits consist of multi-million gates and the classical structural ATPG algorithms reach their limits. Tests for a large number of faults can still be generated very quickly. But the size of the set of faults for which no test can be generated in acceptable run time increases significantly. As a result, the high fault coverage demands of the chip manufacturers can barely be met. Thus, the overall quality of the test set decreases due to the lower fault coverage. As a consequence, there is a need for new robust ATPG algorithms especially when considering future design sizes.

Furthermore, another crucial point has been emerged in the field of manufacturing test. Due to the increased operation speed beyond the GHz mark and the small manufacturing technologies, the number of timing-related defects which affects the product quality has increased. This trend is expected to grow with the process technology scaling down towards very deep sub-micron devices. Therefore, *delay testing* has become mandatory to filter out defective devices and to assure that the desired performance specifications are met.

Test generation for delay faults generally needs more computational effort than ATPG for stuck-at faults because test patterns have to be computed over at least two time frames. A large number of delay faults remain unclassified during ATPG for modern designs. Moreover, in contrast to the SAFM, test patterns for delay faults can be classified in different quality levels like robust and non-robust test patterns [KC98]. The quality of a test can be – simply spoken – defined as the probability of fault detection. High quality test patterns are more desirable but usually harder to obtain. As a result, the need for new robust ATPG algorithms is even more urgent in the field of delay test generation. In the last years, *Small Delay Defects* (SDDs) have become more and more serious. Therefore, the ability to detect SDDs has become an important indicator for judging the quality of a delay test. However, ATPG approaches struggle with the generation of tests dedicated to detect SDDs due to the high complexity.

A promising solution to close the gap between test quality requirements and ATPG effectiveness is the application of solvers for *Boolean Satisfiability* (SAT).