FORMAL VERIFICATION

AN ESSENTIAL TOOLKIT FOR MODERN VLSI DESIGN

ERIK SELIGMAN ■ TOM SCHUBERT
M V ACHUTHA KIRAN KUMAR

MK
MORGAN KAUFMANN

Formal Verification

An Essential Toolkit for Modern VLSI Design

Erik Seligman

Tom Schubert

M V Achutha Kiran Kumar

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO
Morgan Kaufmann is an imprint of Elsevier

Acquiring Editor: Todd Green
Editorial Project Manager: Lindsay Lawrence
Project Manager: Priya Kumaraguruparan
Cover Designer: Alan Studholme

Morgan Kaufmann is an imprint of Elsevier
225 Wyman Street, Waltham, MA 02451, USA

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

ISBN: 978-0-12-800727-3

British Library Cataloguing-in-Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

Formal Verification

Foreword for "Formal Verification: An Essential Toolkit for Modern VLSI Design"

Complex VLSI designs pervade almost every aspect of modern life. We take for granted the existence of cell phones, or antilock braking systems, or web servers, without even being aware of the technology under the hood. As users or customers, we also take it for granted that these devices will function correctly. The impact of an undetected functional failure can range from mild annoyance to major catastrophe, and the potential cost—to individuals, to companies, to society as a whole—can be immense. As just one example, the Pentium® "FDIV" bug in 1994 resulted in Intel Corporation taking a $475M charge against earnings to cover the cost associated with replacement of the defective CPUs.

The danger of an undetected functional failure is compounded by the relentless march of semiconductor process technology—"Moore's Law"—that allows us to put ever more complex designs onto a single piece of silicon. With this complexity come an increasing number of bugs, which need to be found and fixed before the design can be qualified for production use. With shorter design cycles and increased competitive pressure to get products to market faster, verification is now the critical path to production. The authors of the 2011 International Technology Roadmap for Semiconductors described verification as "a bottleneck that has now reached crisis proportions."

The traditional "dynamic testing" approach to design verification has been to:

- Create a verification wrapper or "testbench" around all or part of the design
- Write focused tests, or generate directed random tests, in order to stimulate the design
- Run these tests on an executable software model of the design (typically written at the Register Transfer Level, or RTL for short)
- Debug any resulting failures, fix the offending component (either the RTL, the testbench, or the test itself), and re-run tests until the design is "clean"

However, this approach has a number of drawbacks:

- Testbench development can be a lengthy process—typically of the order of months for complex areas of the design
- Testbench development is an error-prone activity, often creating a number of bugs in the testbench equal to or greater than the number that exist in the actual RTL

- Test execution is an expensive business—Intel dedicates tens of thousands of high-performance computer servers to this problem running around the clock, along with dedicated emulation hardware and FPGA-based solutions
- Tests themselves may contain errors that either mask problems in the RTL (false positives) or wrongly indicate errors that do not in fact exist (false negatives)
- Debug of failing tests is a major effort drain—often the largest single component of validation effort—in part because the failure is often detected only long after the point at which it occurred
- It is in general hard to tell how much of the design has been exercised ("covered") by any given set of tests, so even if all tests are passing it still isn't clear that the design is really clean
- Bug detection via dynamic testing is an inherently sequential process, often referred to as "peeling the onion," since bugs can and do hide behind other bugs
- Some bugs—like the Pentium® FDIV bug mentioned earlier—are data-dependent, or involve such a complex set of microarchitectural conditions that it is highly unlikely that they will be hit by random testing on an RTL model

A different approach, formal verification (FV), has gained acceptance in recent years as a shorter-time, lower-effort, and more comprehensive alternative to dynamic testing. The purpose of this book is to explain what FV is, why it offers at least a partial solution to the limitations of dynamic testing, and how, when, and where it should be applied.

FV is a powerful technique that enables a design or validation engineer to directly analyze and mathematically explore the quality or other aspects of an RTL design, providing 100% coverage of large subsets of the design space without needing to create a simulation testbench or test vectors. Its usage and development have been growing at Intel over the last two decades, and it is now increasingly considered a mainstream technique for both design and validation.

The authors of this book start by describing their goal: helping VLSI designers and validators who focus on RTL to do their jobs more effectively and efficiently by leveraging FV techniques. This approach is sometimes referred to at Intel as the democratization of FV, or "FV for All," since the intent is to expand the use of FV beyond the realm of FV experts and enable much wider use of FV tools and techniques. They briefly describe the history of FV: how early artificial intelligence concepts led to formal verification; theoretical objections to formal verification and why they are not true blocking issues; and general techniques for abstracting problems to make them more tractable for formal verification.

Chapter 2 describes basic FV algorithms in enough detail to convince the reader that FV isn't some form of black magic—these techniques really do gain full coverage without requiring exhaustive (and computationally infeasible) simulation cycles. In particular, the Boolean satisfiability (SAT) problem is explained,

along with a description of the model-checking algorithms and tools that allow it to be solved for many classes of problem.

Chapter 3 provides an introduction to System Verilog Assertions (SVAs): what they are (and the different types of SVAs such as simple Boolean conditions, temporal assertions, etc.) and how they can be combined into sequences and properties. Chapter 4 builds on this by introducing the concept of Formal Property Verification (FPV): what it is; how it compares with dynamic simulation; and usage models such as design exploration, bug hunting, and bounded- and full-proofs.

The heart of the book lies in Chapters 5 and 6, which explain how to make effective use of FPV for Design Exercise and Verification, respectively. The authors' extensive experience with the application of FPV to real-world problems illuminates almost every page of these two chapters with examples gleaned from current and past Intel development projects, along with many helpful hints that will enable the novice user to make effective use of FPV tools.

Chapter 7 switches gears to describe how FPV can be used to create "Apps" for different design domains such as post-silicon bug reproduction, SoC connectivity checking, standard (non-project-specific) protocol verification, unreachable coverage elimination, and control register access policies. These apps enable design and validation engineers who do not have an FV background to quickly apply formal verification methods to the targeted domains, rather than relying on expensive and time-consuming dynamic simulation.

Chapter 8 discusses Formal Equivalence Verification (FEV), which is one of the most mature FV techniques and one that has been deployed at Intel for many years to ensure that RTL matches schematics, and hence that validation results from RTL simulation or formal verification will be applicable to the functional correctness of actual silicon.

Chapter 9—"FV's Greatest Bloopers: False Positives in Formal Verification" is worth the price of the book all by itself. It discusses some of the limitations of formal methods, in particular, the so-called "vacuity" issue, which is an assumption or set of assumptions that rules out legal data and hence leads to false positives (the absence of counter-examples). The examples given in this chapter, and the tips for avoiding these kinds of real-world problems, are worth their weight in gold!

With all of the benefits that FV can provide, the astute reader may be wondering why the technique is not already used more widely as an integral part of the VLSI design process. The answer is that design complexity, and the capacity needed to handle it, can overwhelm even the best FV tools. Chapter 10 addresses this critical issue, first describing the general concepts of complexity and NP-completeness, and then identifying complexity reduction techniques such as black boxing, case splitting, and property simplification that can be used to make FV a tractable solution. Here again, the authors' experience with complex real-world designs enables them to provide practical advice and solutions to what can appear to be daunting problems.

Chapter 11 wraps things up by describing how the reader can introduce the techniques described in this book into his or her organization, and effectively deploy them to increase design and validation productivity. Once again, the emphasis is on the practical: solving real-life problems, using small experiments to demonstrate the power of FPV techniques, getting measurable results and data that help to convince skeptics.

This book is a valuable addition to the library of both experienced and novice users of formal tools and methods. It has my highest recommendation.

Robert Bentley
Former Director,
Formal Verification Center of Expertise, Intel

Acknowledgments

The contents of this book were inspired by the collective efforts of hundreds of our colleagues, at Intel and in the industry beyond; too numerous to name here. But we must thank all the researchers, mathematicians, designers, and validation engineers, whose hard work over many decades has brought us to a technological state where books like this one, and the methods we describe, are accessible and useful for a general engineering audience.

We would also like to individually thank our colleagues who reviewed and sent notes on various draft sections of this book, making major contributions to the quality of the final version. These include: Dan Benua, Jesse Bingham, Claudia Blank, Shalom Bresticker, Ben Cohen, Craig Deaton, Jim Falkenstrom, Harry Foster, Ziyad Hanna, David Hoenig, Robert Jones, Chris Komar, Joe Leslie-Hurd, Mayank Singhal, Brandon Smith, Noppanunt Utamaphethai, Gordon Vreugdenhil, and Ping Yeung.

Finally, we would like to thank our families, whose patience and support made this book possible.

Contents

Foreword for "Formal Verification: An Essential Toolkit for Modern VLSI Design" xiii
Acknowledgments xvii

CHAPTER 1 Formal verification: from dreams to reality 1
What Is FV? 2
Why This Book? 3
A Motivating Anecdote 4
FV: The Next Level of Depth 6
Overall Advantages of FV 6
General Usage Models for FV 7
FV for Complete Coverage 8
FV Methods Not Discussed in This Book 11
The Emergence of Practical FV 13
Early Automated Reasoning 13
Applications to Computer Science 14
Model Checking Becomes Practical 14
The Standardizing of Assertions 15
Challenges in Implementing FV 16
Fundamental Limitations of Mathematics 16
Complexity Theory 17
The Good News 17
Amplifying the Power of Formal 18
Getting the Most Out of This Book 19
Practical Tips from This Chapter 20
Further Reading 21

CHAPTER 2 Basic formal verification algorithms 23
Formal Verification (FV) in the Validation Process 23
A Simple Vending Machine Example 24
Comparing Specifications 26
Cones of Influence 28
Formalizing Operation Definitions 29
Building Truth Tables Intelligently 29
Adding Sequential Logic 30

Boolean Algebra Notation 30
Basic Boolean Algebra Laws 32
Comparing Specifications 32
BDDs 35
Computing a BDD for a Circuit Design 37
Model Checking 38
Boolean Satisfiability 39
Bounded Model Checking 40
Solving the SAT Problem 41
The Davis–Putnam SAT Algorithm 43
The Davis Logemann Loveland (DLL) SAT Algorithm 45
Chapter Summary 46
Further Reading 47

CHAPTER 3 Introduction to systemverilog assertions 49
Basic Assertion Concepts 50
A Simple Arbiter Example 50
What Are Assertions? 51
What Are Assumptions? 51
What Are Cover Points? 52
Clarification on Assertion Statements 52
SVA Assertion Language Basics 53
Immediate Assertions 54
Writing Immediate Assertions 55
Complications of Procedural Code and Motivation for Assert Final 56
Location in Procedural Blocks 58
Boolean Building Blocks 58
Concurrent Assertion Basics and Clocking 59
Sampling and Assertion Clocking 60
Sampled Value Functions 61
Concurrent Assertion Clock Edges 62
Concurrent Assertion Reset (Disable) Conditions 64
Setting Default Clock and Reset 65
Sequences, Properties, and Concurrent Assertions 66
Sequence Syntax and Examples 67
Property Syntax and Examples 71
Named Sequences and Properties 74
Assertions and Implicit Multithreading 75
Writing the Properties 76
Summary 83
Practical Tips from This Chapter 84
Further Reading 86

CHAPTER 4 Formal property verification 87
What Is FPV? 87
Example for This Chapter: Combination Lock 90
Bringing Up a Basic FPV Environment 91
How Is FPV Different from Simulation? 103
What Types and Sizes of Models Can Be Run? 105
How Do We Reach Targeted Behaviors? 105
What Values Are Checked? 106
Deciding Where and How to Run FPV 111
Summary 115
Practical Tips from This Chapter 116
Further Reading 117

CHAPTER 5 Effective FPV for design exercise 119
Example for This Chapter: Traffic Light Controller 120
Creating a Design Exercise Plan 123
Design Exercise Goals 124
Major Properties for Design Exercise 125
Complexity Staging Plan 126
Exit Criteria 129
Putting It All Together 130
Setting Up the Design Exercise FPV Environment 131
Cover Points 131
Assumptions 132
Assertions 132
Clocks and Resets 133
Sanity-Checks 133
Wiggling the Design 133
The Wiggling Process 133
Wiggling Stage 1: Our First Short Waveform 135
Debugging Another Short Waveform 137
Exploring More Interesting Behaviors 140
Answering Some New Questions 140
Proving the Assertions 143
Removing Simplifications and Exploring More Behaviors 146
Facing Complexity Issues 149
Summary 150
Practical Tips from This Chapter 151
Further Reading 152

CHAPTER 6 Effective FPV for verification 153
Deciding on Your FPV Goals 154
Bug Hunting FPV 154
Full Proof FPV 155

Staging Your FPV Efforts 156
Example for This Chapter: Simple ALU 157
Understanding the Design 158
Creating the FPV Verification Plan 161
FPV Goals 162
Major Properties for FPV 163
Addressing Complexity 167
Quality Checks and Exit Criteria 170
Initial Cover Points 176
Extended Wiggling 177
Expanding the Cover Points 182
Removing Simplifications and Exploring More Behaviors 183
The Road to Full Proof FPV 184
Summary 185
Practical Tips from This Chapter 186
Further Reading 188

CHAPTER 7 FPV "Apps" for specific SOC problems 189
Reusable Protocol Verification 190
Basic Design of the Reusable Property Set 191
Property Direction Issues 193
Verifying Property Set Consistency 195
Checking Completeness 197
Unreachable Coverage Elimination 198
The Role of Assertions in UCE 200
Covergroups and Other Coverage Types 201
Connectivity Verification 201
Model Build for Connectivity 202
Specifying the Connectivity 204
Possible Connectivity FPV Complications 205
Control Register Verification 207
Specifying Control Register Requirements 209
SVA Assertions for Control Registers 210
Major Challenges of Control Register Verification 212
Post-Silicon Debug 215
Building the FPV Environment 215
The Paradox of Too Much Information 218
Using Semiformal Design Exploration 219
Proving Your Bug Fixes Correct 220
Summary 221
Practical Tips from This Chapter 221
Further Reading 224

CHAPTER 8 Formal equivalence verification 225
Types of Equivalence to Check 226
Combinational Equivalence 226
Sequential Equivalence 228
Transaction-Based Equivalence 229
FEV Use Cases 231
RTL to Netlist FEV 231
RTL to RTL FEV 233
Running FEV 238
Choosing the Models 238
Key Point Selection and Mapping 240
Assumptions and Constraints 245
Debugging Mismatches 247
Additional FEV Challenges 248
Latch/Flop Optimizations 248
Conditional Equivalence 249
Don't Care Space 251
Complexity 252
Summary 256
Practical Tips from This Chapter 256
Further Reading 259

CHAPTER 9 Formal verification's greatest bloopers: the danger of false positives 261
Misuse of the SVA Language 263
The Missing Semicolon 263
Assertion at Both Clock Edges 264
Short-Circuited Function with Assertion 265
Subtle Effects of Signal Sampling 266
Liveness Properties That Are Not Alive 267
Preventing SVA-Related False Positives 268
Vacuity Issues 269
Misleading Cover Point with Bad Reset 270
Proven Memory Controller That Failed Simulation 271
Contradiction Between Assumption and Constraint 272
The Queue That Never Fills, Because It Never Starts 274
Preventing Vacuity: Tool and User Responsibility 275
Implicit or Unstated Assumptions 277
Libraries with Schematic Assumptions 277
Expectations for Multiply-Driven Signals 279
Unreachable Logic Elements: Needed or Not? 279
Preventing Misunderstandings 281

Division of Labor 282
Loss of Glue Logic 282
Case-Splitting with Missing Cases 283
Undoing Temporary Hacks 284
Safe Division of Labor 284
Summary 285
Practical Tips from This Chapter 286
Further Reading 287

CHAPTER 10 Dealing with complexity 289
Design State and Associated Complexity 289
Example for This Chapter: Memory Controller 291
Observing Complexity Issues 293
Simple Techniques for Convergence 294
Choosing the Right Battle 294
Engine Tuning 295
Blackboxing 297
Parameters and Size Reduction 297
Case-Splitting 298
Property Simplification 300
Cut Points 302
Semiformal Verification 305
Incremental FEV 306
Helper Assumptions . . . and Not-So-Helpful Assumptions 307
Writing Custom Helper Assumptions 308
Leveraging Proven Assertions 308
Do You Have Too Many Assumptions? 309
Generalizing Analysis Using Free Variables 310
Exploiting Symmetry with Rigid Free Variables 310
Other Uses of Free Variables 311
Downsides of Free Variables 312
Abstraction Models for Complexity Reduction 313
Counter Abstraction 313
Sequence Abstraction 315
Memory Abstraction 316
Shadow Models 319
Summary 320
Practical Tips from This Chapter: Summary 321
Further Reading 323

CHAPTER 11 Your new FV-aware lifestyle 325
Uses of FV 326
Design Exercise 326