

# SCALABLE HARDWARE VERIFICATION WITH SYMBOLIC SIMULATION

# SCALABLE HARDWARE VERIFICATION WITH SYMBOLIC SIMULATION

VALERIA BERTACCO  
University of Michigan



Valeria Bertacco  
The University of Michigan  
Advanced Computer Architecture Lab  
Department of EE & CS  
1301 Beal Avenue, Room 2224  
Ann Arbor, MI 48109  
U.S.A.

## Scalable Hardware Verification with Symbolic Simulation

Cover design by S. Alexander Garcia

Library of Congress Control Number: 20055934803

ISBN-10: 0-387-24411-5  
ISBN-13: 9780387244112

ISBN-10: 0-387-29906-8 (e-book)  
ISBN-13: 9780387299068 (e-book)

Printed on acid-free paper.

© 2006 Springer Science+Business Media, Inc.

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, Inc., 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 in the United States of America.

9 8 7 6 5 4 3 2 1

SPIN 11325031

springeronline.com

*To Roberta, Bruno,  
Livio and Todo.*

# Contents

Dedication	v
List of Figures	xi
List of Tables	xiii
Preface	xv
Acknowledgments	xix
1. INTRODUCTION	1
1.1 Functional validation	2
1.2 Formal verification	3
1.2.1 Symbolic simulation	3
1.3 Organization of the book	5
2. DESIGN AND VERIFICATION OF DIGITAL SYSTEMS	7
2.1 The design flow	7
2.2 RTL verification	11
2.3 Boolean functions and their representation	14
2.3.1 NP-equivalence	16
2.4 Binary decision diagrams	17
2.5 Models for design verification	19
2.5.1 Structural network model	20
2.5.2 State diagrams	22
2.5.3 Mathematical model of finite state machines	23
2.6 Functional validation	24
2.7 Formal verification	28
2.7.1 Symbolic finite state machine traversal	29
2.8 Summary	31
References	31
3. SYMBOLIC SIMULATION	35
3.1 The origins of symbolic simulation	35
3.2 Symbolic simulation of a logic gate	36
3.3 Symbolic simulation, time frame-by-time frame	37
3.3.1 Symbolic simulation to expose design flaws	40
3.4 Close relatives of symbolic simulation	42

3.4.1	Symbolic reachability analysis	42
3.4.2	Symbolic trajectory evaluation	44
3.5	Enhancements and optimizations	45
3.6	The challenge in symbolic simulation	47
3.7	Summary	48
	References	48
4.	COMPACTING INTERMEDIATE STATES	51
4.1	Parametric transformations	51
4.1.1	A formal definition	52
4.1.2	Applications to symbolic simulation	53
4.1.3	A brief history of parametric solutions	56
4.2	Disjoint-support decompositions	57
4.2.1	A canonical form of DSDs	59
4.2.2	Decomposition trees	59
4.3	A BDD-based algorithm to extract DSDs	62
4.3.1	Building decompositions bottom-up	62
4.3.2	Putting it all together: The DEC procedure	64
4.3.3	Complexity analysis and considerations	67
4.3.4	Decomposability experiments	68
4.4	On the decomposability of Boolean functions	75
4.5	Evolution of disjoint-support decompositions	76
4.6	Summary	77
	References	78
5.	APPROXIMATE SIMULATION	81
5.1	Cycle-based symbolic simulation flow	82
5.2	The CBSS algorithm	82
5.3	The reparametrization phase	83
5.3.1	Using functional dependencies	85
5.3.2	How to classify the components of the state vector	87
5.3.3	The remap function	91
5.4	Implementation and insights	93
5.4.1	Experimental results	94
5.5	Quasi-Symbolic Simulation	99
5.5.1	Simulation with X values	100
5.5.2	Approximating and reclassifying symbolic variables	101
5.5.3	Care and Don't care sets	102
5.6	Summary	103
	References	104

6. EXACT PARAMETRIZATIONS	105
6.1 Re-encoding the state function using DSDs	105
6.1.1 Reduction at free points	108
6.1.2 Elimination of prime functions	111
6.1.3 Removal of non-dominant variables	114
6.2 The DSD-based simulator	117
6.2.1 Experimental results	117
6.3 Parametrization in the micro-processor domain	122
6.3.1 Structural decompositions	123
6.3.2 Parametrization for data-space partitions	124
6.4 Summary	125
References	125
7. CONCLUSION	127
7.1 Enabling techniques for symbolic simulation	128
7.2 Scalable symbolic simulation techniques	128
References	129
Appendices	130
A Disjoint-Support Decompositions	131
A.1 Function decompositions	131
A.2 The unique maximal Disjoint-Support Decomposition	132
A.2.1 Partitions and representative elements	133
A.2.2 Uniqueness of the kernel function	134
A.2.3 Uniqueness of the actuals list	136
A.3 The canonical decomposition tree	141
A.3.1 Extracting all decompositions from the canonical tree	142
A.4 Building the decomposition tree from a BDD	145
A.4.1 Case 1. Neither $A_{10}$ nor $A_{11}$ is constant and $A_{10} \neq \overline{A_{11}}$	147
A.4.1.1 Case 1.a - <i>PRIME</i> decomposition	147
A.4.1.2 Case 1.b - Associative decomposition	148
A.4.2 Case 2. Exactly one of $A_{10}$ , $A_{11}$ is constant	149
A.4.2.1 Case 2.a - <i>PRIME</i> decomposition	150
A.4.2.2 Case 2.b - Associative decomposition	150
A.4.3 Case 3. $A_{10} = \overline{A_{11}}$ and $A_{10}$ is not a constant	151
A.4.3.1 Case 3.a - <i>PRIME</i> decomposition	152
A.4.3.2 Case 3.b - Associative decomposition	152
A.4.4 New decompositions	153
A.4.4.1 Case NEW.a - <i>AND</i> or <i>OR</i> decomposition	153
A.4.4.2 Case NEW.b - <i>XOR</i> decomposition	153
A.4.4.3 Case NEW.c - <i>PRIME</i> decomposition	153
A.5 The DEC procedure	160
A.5.1 Inherited decompositions	160
A.5.1.1 <i>OR</i> decompositions	160

A.5.1.2 <i>XOR</i> decompositions	161
A.5.1.3 <i>PRIME</i> decompositions	161
A.5.2 New decompositions	164
A.5.2.1 <i>OR</i> and <i>XOR</i> decompositions	165
A.5.2.2 <i>PRIME</i> decompositions	165
References	166
References	168
Index	175



# List of Figures

2.1	Conceptual design flow of a digital system	8
2.2	Approaches to verification: Validation vs. Formal Verification	13
2.3	Examples of Binary Decision Diagrams	18
2.4	Graphic symbols for basic logic gates	20
2.5	Structural network model schematic	21
2.6	Network model of a 3-bits up/down counter with reset	21
2.7	State diagram for a 3-bits up/down counter	22
2.8	State diagram for a 1-hot encoded 3-bits counter	23
2.9	Compiled logic simulator	25
2.10	Pseudo-code for a cycle-based logic simulator	26
3.1	Comparison of logic and symbolic simulation	36
3.2	Simulation of a netlist by composition of symbolic expressions	37
3.3	Schematic of the iterative model of symbolic simulation	38
3.4	Symbolic simulation for Example 3.1 - Initialization phase	39
3.5	Symbolic simulation for Example 3.1 - Simulation Step 2	40
3.6	Pseudo-code for frame-by-frame symbolic simulation	41
3.7	Pseudo-code for symbolic reachability analysis	43
4.1	Parametrization of the state vector during symbolic simulation	53
4.2	Parametrization of the state vector during symbolic simulation	54
4.3	Three steps of symbolic simulation for the counter of Example 2.2 and possible parametrizations of the reached state sets	55
4.4	General form of a disjoint-support decomposition (DSD)	57
4.5	Three different disjoint-support decompositions for Example 4.3	58
4.6	A decomposition tree for Example 4.4	60
4.7	Decomposition data structure for the function of Example 4.5	61
4.8	Pseudo-code for the <code>decompose_node</code> procedure	65
4.9	Pseudo-code for the <code>decompose</code> procedure	65
4.10	Pseudo-code for <code>decompose_INHERITED</code>	66
4.11	Pseudo-code for <code>decompose_NEW</code>	66
5.1	Flow of cycle-based symbolic simulation algorithm	83

5.2	Pseudo-code for cycle-based symbolic simulation	84
5.3	The parametrized frontier subset $\mathbf{PS}_{@k}$	86
5.4	Pseudo-code for the <code>Parametrize</code> function of CBSS	87
5.5	Pseudo-code for classifying simple and complex support variables	89
5.6	Pseudo-code for classifying shared support variables	91
5.7	Comparison of CBSS vs. logic simulation	98
5.8	Definition of logic operations over the ternary set $\{0, 1, X\}$	100
5.9	MTBDD for the function $(a + X)b$	101
5.10	Quasi-symbolic simulation for Example 5.7	102
6.1	The decomposed state vector for a small design	107
6.2	The parameterized frontier set $\mathbf{PS}_{@k}$	108
6.3	A vector function and its free points	109
6.4	Free points elimination for Example 6.1	111
6.5	General case for prime function elimination: (a) before and (b) after the transformation	113
6.6	Prime elimination in test <i>s1196</i> for Example 6.2	113
6.7	Non-dominant variable removal for Example 6.4	116
6.8	Comparison of DSD simulation vs. cycle-based symbolic simulation and vs. logic simulation	122
6.9	Design decomposition for Example 6.5	124
A.1	Decomposition tree for Example A.5.	143
A.2	PRIME decomposition.	155
A.3	Function for Example A.11.	158
A.4	Pseudo-code for <code>decompose_INHERITED_OR_123 . b</code>	161
A.5	Pseudo-code for <code>decompose_INHERITED_PRIME_1 . a</code>	163
A.6	Pseudo-code for <code>decompose_INHERITED_PRIME_2 . a</code>	164
A.7	Two functions and the construction of their $Max(G, H)$ tree.	166

## List of Tables

4.1	Disjoint Support Decomposition results (Part 1)	70
4.2	Disjoint Support Decomposition results (Part 2)	71
4.3	Disjoint Support Decomposition results (Part 3)	72
4.4	Disjoint Support Decomposition results (Part 4)	73
4.5	Disjoint Support Decomposition results (Part 5)	74
4.6	Disjoint Support Decomposition results (Part 6)	75
5.1	Cycle Based Symbolic Simulation results (Part 1)	95
5.2	Cycle Based Symbolic Simulation results (Part 2)	96
6.1	DSD-based Symbolic Simulation (Part 1)	119
6.2	DSD-based Symbolic Simulation (Part 2)	120

# Preface

In the past 40 years, electronic systems have become a pervasive force in modern society. Digital integrated circuits (ICs) are at the heart of a large majority of these systems. These digital ICs are complex systems comprised of millions of interconnected transistors in a very small area. Moreover, the underlying semiconductor fabrication technology used to manufacture these ICs allows for the doubling of the number of transistors in the same area approximately every 18 months.

The design of digital systems is an intricate and time consuming process that progresses through various phases and levels of abstraction relying heavily on CAD (Computer-Aided Design) software tools. Within this context, ensuring the correctness of these digital systems is a critical consideration, especially because failure costs are becoming increasingly high. One of the most famous, recent examples of the importance of correct design is the Intel Pentium flaw in the floating point divide unit in 1994 that eventually forced Intel to replace many of the Pentium chips that were already in the market. In many cases, the possibility of failure is plainly unacceptable. Examples of these applications are transportation systems, medical applications and financial systems. Driven by the importance of correct design, the cost of verification in modern computing systems has grown to dominate the cost of system design in terms of the time and human resources dedicated to it. In contrast, even though guaranteeing the correctness of a design is such a central aspect of its development, current verification methodologies are still inadequate to tackle the complex systems that are being developed nowadays. Hardware design companies try to compensate for mediocre CAD tools by dedicating the majority of their resources to verification, yet are still unable to guarantee correct functionality over the entire design space.

In industry, the scalability, flexibility and predictable run-time behavior of logic simulation makes it the most widely accepted technique for ensuring the correctness of digital ICs. The technique is based on verifying a digital system

by providing sequences of binary values for each of the inputs of the system and checking that the corresponding outputs are correct, based on what the design team expected to see or described in a specification document. However, logic simulation can usually visit only a small fraction of all the possible configurations of a system - also called the state space - and, thus, the discovery of bugs heavily relies on the expertise of the designer to select a few crucial configurations to verify.

Symbolic simulation is another verification method that is attracting increasing interest because it allows the verification engineer to explore all, or a major portion, of a circuit's state space without the need to design time-consuming test stimuli. However, this approach poses a high demand on the resources of the simulating host, and in particular, on the memory system, because of the complexity of the algorithms involved and their unpredictable runtime behavior. Thus, the scalability of this approach has been the main limiting factor to its mainstream deployment, with the consequence that, thus far, its scope has been limited to small systems.

## About this book

This book presents recent advancements in symbolic simulation-based solutions which radically improve scalability. We overview current verification techniques, both based on logic simulation and on formal verification methods, and we describe in detail the baseline technique of symbolic simulation. The core of this book focuses on new techniques that narrow the performance gap between the complexity of digital systems and the limited ability to verify them. In particular we cover a range of solutions that exploit approximation and parametrization methods in order to achieve this goal. In the direction of approximation techniques, we comprehensively cover quasi-symbolic simulation – an aggressive technique aiming at simulating only the portion of the design necessary for the verification goal at hand – and cycle-based symbolic simulation – a unique combination of formal methods and logic simulation that can stimulate a circuit with a very large number of input combinations and sequences in parallel. Cycle-based symbolic simulation is a hybrid solution that uses both approximation and parametrization to attain its scalability goal. Its key concept is the use of a parametric form to represent the set of states visited during simulation. This approach maintains a high degree of scalability, in line with current logic simulation techniques, while achieving better efficiency.

In the realm of parametric solutions, we discuss a range of approaches, including various applications of parametric symbolic simulation to industrial microprocessor designs. An in-depth analysis is dedicated to another solution that we recently proposed, disjoint-support decomposition-based symbolic simulation, where the parametrization makes use of the disjoint-support decomposition properties of a Boolean function. This simulation technique is

rooted on a novel algorithm that exposes the disjoint decomposition properties of a Boolean function by restructuring its BDD representation. The new algorithm is very efficient in the sense that it has worst-case complexity that is only quadratic in the size of the initial BDD, compared to that of previous solutions which had exponential complexity in the number of input variables of the function. We deploy this algorithm to decompose of the state functions in symbolic simulation. Then, by restructuring the next-state functions using their disjoint components, it is possible to transform them into a simpler parametric form without sacrificing simulation accuracy. Results show that this technique is effective in reducing the memory requirements of symbolic simulation while maintaining exact state exploration. When the design complexity becomes overwhelming, it can trade-off search breadth for performance, and proceed to simulate very large trace sets in parallel, thus maintaining a simulation speed and memory profile that are close to logic simulation.

In structuring this book, the hope was to provide an interesting reading for a broad range of readers. Chapters 1, 2 and 3 constitute a panoramic flight over the world of digital systems' design and, in particular, verification. Chapter 3 reviews some of the mainstream symbolic techniques in formal verification, dedicating most of the focus to symbolic simulation.

We use Chapter 4 to cover the necessary principles of parametric forms and disjoint-support decompositions. In particular, we attempt to keep the material at a level that facilitates understanding, but without too many formal details. While there is a range of resources discussing parametric forms and parametrizations for Boolean functions, we felt that the topic of disjoint-support decompositions was not as readily available. For that reason Appendix A complements Chapter 4 in providing a more formal presentation of the topic and derivation of the theoretical results.

Chapters 5 and 6 focus on a range of recent symbolic simulation techniques, which we grouped in approximate solutions, and exact parametrizations. Finally, Chapter 7 wraps up the presentation and outlines possible future research directions.

# Acknowledgments

I would like to acknowledge several people who were critical in making this book a reality through their work, advice and support. The main solutions discussed by this book were developed during my Ph.D. work at Stanford, with the supervision of Kunle Olukotun, who had always been prompt and available in supporting whatever direction of research, and of life, I pursued. In our technical interactions, he would always move straight to the results of my work and challenge me on their practical contribution to improve the quality of verification in industrial designs. David Dill has been the person I could always go to bounce ideas off of and to have illuminating technical discussions. When my ideas could survive his dissecting analysis, I knew I could publish them.

My years at Synopsys have played a central role in shaping my understanding of design verification as an industrial challenge first and a research area later. My colleagues have been crucial in providing me with invaluable opportunities: Ghulam Nurie, Swami Venkat and the Vera Group team, who gave me early opportunities to interact with customers. Those customer meetings have always been enlightening in my quest towards understanding the needs of hardware designers; Pei-Hsin Ho, my manager in the Advanced Technology Group of Synopsys, showed me how to efficiently achieve technology transfers, by taking academic research and deploying it in software solutions for the hardware-design community. My undergraduate advisor, Maurizio Damiani, first introduced me to research and to the area of Computer-Aided Design for integrated circuits. I would like to thank him for the numerous interactions and collaborations that lasted long after my undergraduate studies and spurred many of the publications that led to this research work.

On a personal level, I would like to thank my parents for teaching me the first concepts of mathematics and logic and for introducing me, early in my life, to the pursuit of both education and industry experience. I also want to thank my family for supporting my choices in my path through life. My brother, Livio, provided all sorts of technical support and advice and solved many sys-

tem crashes, most often connecting from some remote location in Europe. My colleagues and my students at the University of Michigan have been an important source of support in continuing this research and in helping shuffle all the work and deadlines that are always overlapping: Kai-hui Chang, Steve Plaza, Smitha Shyam and Ilya Wagner. I would also like to thank Todd Austin for the numerous technical discussions and for taking the time to review this book multiple times. In addition, I would like to thank Gloria Cadavid, Alex Garcia, Tim Wright and Azita Emami.

Finally, I would like to express gratitude to my editors Michael Hackett and Alex Greene and their collaborators, Rose Antonelli, Melissa Guasch and Rebecca Olson at Springer for their help in preparing the final manuscript and keeping me on task, particularly during a year of transition and growth in the company.