

# Debugging at the Electronic System Level



Frank Rogin • Rolf Drechsler

# Debugging at the Electronic System Level

Frank Rogin  
Fraunhofer - Institut für  
Integrierte Schaltungen  
Institutsteil  
Entwurfsautomatisierung  
Zeunerstr. 38  
01069 Dresden  
Germany  
frank.rogin@web.de

Rolf Drechsler  
Universität Bremen  
AG Rechnerarchitektur  
Bibliothekstr. 1  
28359 Bremen  
Germany  
drechsle@Informatik.Uni-Bremen.de

ISBN 978-90-481-9254-0 e-ISBN 978-90-481-9255-7  
DOI 10.1007/978-90-481-9255-7  
Springer Dordrecht Heidelberg London New York

Library of Congress Control Number: 2010929863

© Springer Science+Business Media B.V. 2010

No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by means, electronic, mechanical, photocopying, microfilming, recording or otherwise, without written permission from the Publisher, with the exception of any material supplied specifically for the purpose of being entered and executed on a computer system, for exclusive use by the purchaser of the work.

*Cover design:* eStudio Calamar S.L.

Printed on acid-free paper

Springer is part of Springer Science+Business Media ([www.springer.com](http://www.springer.com))

# Contents

List of Figures	xi
List of Tables	xv
Preface	xvii
Acknowledgements	xix
1. INTRODUCTION	1
1 General Objective of the Book	2
2 Summary of Contributions	5
3 Book Outline	7
2. ESL DESIGN AND VERIFICATION	9
1 ESL Design	9
1.1 ESL Design Flow	10
1.2 System Level Language SystemC	15
1.3 Transaction Level Modeling in SystemC	16
2 ESL Verification	18
2.1 Simulation	19
2.2 Formal Verification	19
2.3 Semi-Formal Verification	20
2.4 Verifying SystemC Models	21
2.4.1 Simulation in SystemC	21
2.4.2 Semi-Formal Verification in SystemC	22
2.4.3 Formal Verification in SystemC	24
3 Our Debugging Approach	25
3.1 Terms	25
3.2 General Debug Process	26
3.3 Hierarchy of Debugging Techniques	27

3.4	SIMD Data Transfer Example	28
3.	EARLY ERROR DETECTION	33
1	Deduction Techniques in a Nutshell	34
1.1	Preliminaries	34
1.2	Related Work	38
2	Static Analysis Framework	40
2.1	Requirements	41
2.2	General Architecture	41
2.3	Generic Framework Components	43
2.3.1	Generic Symbol Table	43
2.3.2	Generic Dataflow Analysis	45
2.3.3	Generic Structural Analysis	49
2.4	Configuration and Adaptation	51
2.4.1	Implementation of Analyses	51
2.4.2	Tailoring to the Analyzed Language	53
2.5	Approach Rating	55
2.5.1	General Benefits	55
2.5.2	General Limitations and Risks	56
2.5.3	Benefits of REGATTA	56
2.5.4	Limitations of REGATTA	57
3	SystemC Design Analysis System	57
3.1	Implementation Effort	58
3.2	Configuration and Adaption	58
3.3	Example Analysis Flow	61
4	Experimental Results	63
4.1	SIMD Data Transfer Example Continued	63
4.2	Industrial SystemC Verification Environment	65
5	Summary and Future Work	68
4.	HIGH-LEVEL DEBUGGING AND EXPLORATION	71
1	Observation Techniques in a Nutshell	72
1.1	Overview	72
1.1.1	Logging	73
1.1.2	Debugging	73
1.1.3	Visualization	75
1.2	Related Work	75
2	System-Level Debugging	77
2.1	Requirements	77
2.2	Methodology	78

- 2.2.1 Debug Levels 78
    - 2.2.2 Debug Flow 80
    - 2.2.3 Debugging Support 82
  - 3 High-Level SystemC Debugging 83
    - 3.1 General Architecture 83
    - 3.2 Debug Pattern Support 84
      - 3.2.1 Scenario-Based Guidance 84
      - 3.2.2 Partially Automated Process 85
      - 3.2.3 Supplied Debug Patterns 86
    - 3.3 SystemC Debugger 86
      - 3.3.1 User Layer 88
      - 3.3.2 API Layer 88
      - 3.3.3 Data Layer 88
    - 3.4 SystemC Visualizer 91
    - 3.5 Implementation Issues 93
  - 4 Experimental Results 95
    - 4.1 SIMD Data Transfer Example Continued 95
    - 4.2 Industrial Examples 97
      - 4.2.1 Efficiency Discussion 98
      - 4.2.2 SHIELD in Practice 99
      - 4.2.3 SHIELD Debugger vs. CoWare SystemC Shell 101
  - 5 Summary and Future Work 103
- 5. LEARNING ABOUT THE DESIGN 105
  - 1 Induction Techniques in a Nutshell 106
    - 1.1 Overview 106
    - 1.2 Related Work 108
  - 2 Automatic Generation of Properties 110
    - 2.1 Generation Methodology 110
    - 2.2 Generation Algorithm 111
      - 2.2.1 Preliminaries 111
      - 2.2.2 Algorithm Description 112
      - 2.2.3 Complexity and Termination 114
    - 2.3 Design Flow Integration 115
  - 3 Dynamic Invariant Analysis on Simulation Traces 116
    - 3.1 General Architecture 117
    - 3.2 Basic Property Generation 118
    - 3.3 Property Filtering 122
    - 3.4 Property Encoding 123
    - 3.5 Complex Property Generation 123

3.6	Property Selection	126
3.7	Implementation Issues	128
4	Experimental Results	128
4.1	SIMD Data Transfer Example Continued	129
4.1.1	Improve Design Understanding	130
4.1.2	Detect Abnormal Behavior	131
4.2	Industrial Hardware Designs	131
4.2.1	Generated Properties	132
4.2.2	Generation Statistics	135
4.2.3	Case Study: Traffic Light Control	138
4.2.4	Case Study: SIMD MP Design	139
5	Summary and Future Work	140
6.	ISOLATING FAILURE CAUSES	143
1	Experimentation Techniques in a Nutshell	143
1.1	Overview	144
1.2	Related Work	145
2	Automatic Isolation of Failure Causes	147
2.1	Requirements	147
2.2	Methodology	148
2.3	Approach Rating	148
3	Automatic Isolation of Failure Causes in SystemC	150
3.1	Debugging Process Schedules	150
3.1.1	Deterministic Record/Replay Facility	152
3.1.2	Isolating Failure Causes	154
3.1.3	Root-Cause Analysis	156
3.2	Debugging Program Input	156
4	Experimental Results	157
4.1	SIMD Data Transfer Example Continued	157
4.2	Failure Causes in Process Schedules	160
4.2.1	Producer–Consumer Application	160
4.2.2	Deadlock Example	161
5	Summary and Future Work	162
7.	SUMMARY AND CONCLUSION	165
	Appendix A. FDC Language	169
1	FDC Syntax	169
2	FDC Semantic	169



Appendix B. Debug Pattern Catalog	173
1 General Format	173
2 COMPETITION Pattern	174
3 TIMELOCK Pattern	177
References	181
List of Acronyms	193
Index of Symbols	195
Index	197



# List of Figures

1.1	Debugging techniques in a simplified ESL design flow	3
1.2	Verification efficiency using only late system level simulation (chart trend according to [Fos06])	4
1.3	Improved verification efficiency using static and dynamic analysis techniques (chart trend according to [Fos06])	4
1.4	Hierarchy of proposed debugging techniques	5
2.1	Idealized ESL design flow (taken from [BMP07])	13
2.2	Designing the hardware part of an SoC	14
2.3	TLM notion in an example (taken from [TLM2])	17
2.4	Reasoning hierarchy in a simplified ESL design flow	29
2.5	General architecture of the SIMD data transfer example	30
2.6	Example of a read/write data transfer sequence	30
3.1	Early error detection in system models	34
3.2	Structure of a front-end for static analysis	34
3.3	EST for the word “aa_bb_n”	36
3.4	General architecture of REGATTA	42
3.5	Extract of the generic symbol table class hierarchy	45
3.6	Control flow patterns recognized in REGATTA	47
3.7	Algorithm to determine the <i>gen</i> sets for each basic block	48
3.8	Algorithm to determine the <i>kill</i> sets for each basic block	48
3.9	<i>gen</i> and <i>kill</i> sets in an example graph	49
3.10	Algorithm to detect undefined variables (according to Aho et al. [ASU03])	49

3.11	Detection of the Composite design pattern using <i>CrocoPat</i>	51
3.12	EBNF syntax of the FDC language	53
3.13	Abstract FDC specification counting arbitrary language elements	54
3.14	Configuration of the FDC “ComponentCnt”	59
3.15	Symbol analysis in SystemC	60
3.16	CFG with annotated variable accesses	60
3.17	Detecting undefined variable accesses	61
3.18	Created relations in a SystemC design as used by <i>CrocoPat</i>	61
3.19	REGATTA exemplary analysis flow	62
3.20	Collect names of found <code>SC_METHOD</code> processes	63
3.21	Create a write transaction in the SIMD processor unit	64
3.22	SDAS error report indicating an undefined address in case of write transactions	64
3.23	Number of analyzed files	65
3.24	Number of analyzed lines of code	66
3.25	Number of violations per month for selected coding checks	66
3.26	Violations per line of code for all coding checks	67
3.27	Number of coding violations existing from the beginning	68
4.1	Debugging and exploration of system models at a higher level	72
4.2	Static vs. dynamic slicing	74
4.3	Hierarchy of debug levels	79
4.4	Debug flow to guide debugging at the ESL	81
4.5	General architecture of SHIELD	83
4.6	<i>lsb</i> command at the API layer	89
4.7	Class hierarchy of the debug database	90
4.8	Dynamic slice for variable <i>pl</i> (code of the SIMD example)	92
4.9	Visualization debugging command <i>vtrace_at</i>	93
4.10	Visualization debugging command <i>vlsb</i>	94
4.11	Preloading a SystemC kernel method	94
4.12	SHIELD screenshot of a SIMD debugging session	96
5.1	Learning about the design using an induction technique	106
5.2	General property generation algorithm	113
5.3	Property generation in the design flow from Figure 2.2	115

5.4	DIANOSIS property generation flow	116
5.5	DIANOSIS general architecture	118
5.6	Finite-state machine of the mutual exclusion checker	120
5.7	Property generation example	126
5.8	Example for a property generation report	127
5.9	Efficient storage of signal values	128
5.10	Optimization of the workload	129
5.11	Incompletely inferred Req_Grant1_Grant2 property	134
5.12	Valid binary property candidates for the SATA FIFO	136
5.13	Valid complex property candidates for the SATA FIFO	136
6.1	Automatic isolation of failure causes in system designs	144
6.2	General delta debugging algorithm according to [ZH02]	149
6.3	Isolation of failure-inducing process schedules	152
6.4	Extract of a recorded process schedule	153
6.5	Simple producer–consumer application	155
6.6	Isolating failure-inducing simulation input	157
6.7	Isolate the failure-inducing instruction in program 433	158
6.8	<i>dd</i> algorithm analysis result for Example 24	160
6.9	Architecture of the deadlock example	161
A.1	EBNF syntax of the FDC language	170
B.1	COMPETITION pattern flowchart	175
B.2	Exemplary race condition	176
B.3	TIMELOCK pattern flowchart	178
B.4	Exemplary TIMELOCK pattern	179



# List of Tables

3.1	Comparison of REGATTA and SDAS	58
4.1	Selection of system-level debugging commands	87
4.2	Selection of visualization debugging commands	91
4.3	SystemC debugger setup times	98
4.4	Exemplary performance slow down due to tracing	99
4.5	Debugging effort in SHIELD and GDB	99
4.6	Comparing SHIELD debugger and CoWare's <i>scsh</i>	102
5.1	Selection of basic property checkers	119
5.2	Test bench characteristics	132
5.3	Found basic properties	133
5.4	Found complex properties	135
5.5	Statistical evaluation	137
6.1	ISOC debugging commands	151
6.2	Illustrating the first steps of the <i>dd</i> algorithm	156
6.3	<i>dd</i> algorithm applied on 13 erroneous programs	159
6.4	Running <i>dd</i> on the deadlock example	162





# Preface

Debugging becomes more and more the bottleneck to chip design productivity, especially while developing modern complex and heterogenous integrated circuits and systems at the *Electronic System Level* (ESL). Today, debugging is still an unsystematic and lengthy process. Here, a simple reporting of a failure is not enough, anymore. Rather, it becomes more and more important not only to find many errors early during development but also to provide efficient methods for their isolation. In this book the state-of-the-art of modeling and verification of ESL designs is reviewed. There, a particular focus is taken onto SystemC. Then, a reasoning hierarchy is introduced. The hierarchy combines well-known debugging techniques with whole new techniques to improve the verification efficiency at ESL. The proposed systematic debugging approach is supported amongst others by static code analysis, debug patterns, dynamic program slicing, design visualization, property generation, and automatic failure isolation. All techniques were empirically evaluated using real-world industrial designs. Summarized, the introduced approach enables a systematic search for errors in ESL designs. Here, the debugging techniques improve and accelerate error detection, observation, and isolation as well as design understanding.



## Acknowledgements

We would like to thank all colleagues of Division Design Automation of Fraunhofer Institute for Integrated Circuits in Dresden and all members of the research group for computer architecture in Bremen for the amazing atmosphere during research and work and the many fruitful discussions.

Moreover, we are grateful to all our coauthors of the papers that make up an important part of this book: Thomas Berndt, Erhard Fehlauer, Görschwin Fey, Christian Genz, Christian Haufe, Thomas Klotz, Sebastian Ohnewald, and Steffen Rülke. Additionally, we would like to thank our students who had contributed many prototype implementations: Lars Ehrler, Dirk Linke, Dominic Scharfe, Tom Schiekkel, and Richard Wähler.

Erhard Fehlauer, Görschwin Fey, and Steffen Rülke helped us in proof-reading and improving the presentation.

FRANK ROGIN AND ROLF DRECHSLER  
Bremen, March 2010

