Software-Centric and Interaction-Oriented System-on-Chip Verification

by

Xiao Xi Xu
B.E. (Automatic Control)
Shanghai Jiao Tong University, China, 1996

A thesis submitted for the
degree of Doctor of Philosophy

in
School of Electrical and Electronic Engineering
Faculty of Engineering, Computer and Mathematical Sciences

March 2009
Declaration of Authorship

This work contains no material which has been accepted for the award of any other degree or diploma in any university or other tertiary institution and, to the best of my knowledge and belief, contains no material previously published or written by another person, except where due reference has been made in the text.

I give consent to this copy of my thesis, when deposited in the University Library, being made available for loan and photocopying, subject to the provisions of the Copyright Act 1968.

Signature:

Date:
Abstract

As the complexity of very-large-scale-integrated-circuits (VLSI) soars, the complexity of verifying them increases even faster. Design verification becomes the biggest bottleneck in VLSI design, consuming around 70% of the effort and time in a typical design cycle. The problem is even more severe as the system-on-chip (SoC) design paradigm is gaining popularity.

Unfortunately, the development in verification techniques has not kept up with the growth of the design capability, and is being left further behind in the SoC era. In recent years, a new generation of hardware-modelling-languages alongside the best practices to use them have emerged and evolved in an attempt to productively build an intelligent stimulation-observation environment referred to as the test-bench. Ironically, as test-benches are becoming more powerful and sophisticated under these best practices known as verification methodologies, the overall verification approaches today are still officially described as ad hoc and experimental and are in great need of a methodological breakthrough.

Our research was carried out to seek the desirable methodological breakthrough, and this thesis presents the research outcome: a novel and holistic methodology that brings an opportunity to address the SoC verification problems. Furthermore, our methodology is a solution completely independent of the underlying simulation technologies; therefore, it could extend its applicability into future VLSI designs.

Our methodology presents two ideas. (a) We propose that system-level verification should resort to the SoC-native languages rather than the test-bench construction languages; the software native to the SoC should take more critical responsibilities than the test-benches. (b) We challenge the fundamental assumption that “objects-under-test” and “tests” are distinct entities; instead, they should be understood as one type of entities – the interactions; interactions, together with the interference between interactions, i.e., the parallelism and resource-competitions, should be treated as the focus in system-level verification.

The above two ideas, namely, software-centric verification and interaction-oriented verification have yielded practical techniques. This thesis elaborates on these techniques, including the transfer-resource-graph based test-generation method targeting the parallelism, the coverage measures of the concurrency completeness using Petri-nets, the automation of the test-programs which can execute smartly in an event-driven manner, and a software observation mechanism that gives insights into the system-level behaviours.
I thank my supervisors Prof. Cheng-Chew Lim and Prof. Michael Liebelt. They provided me with this research position, and they are the co-authors of my research publications. I am grateful to their advice and feedback during the development of this thesis. Cheng-Chew’s help comes in all forms, including the resources he guarantees, the meetings he organises, the peer review he performs and the presentations he rehearses.

I would like to extend special thanks to Mr. Adriel Cheng, who has kindly opened his source codes in the SALVEM (Software Application Level Verification Methodology) approach to me. These codes have guided me to learn new programming languages, new tools and new technologies, and more importantly, I was able to understand the nature of software-based verification from them. It is these codes that have invited me to form my own idea. In addition, the Nios SoC used in my research was generated for the SALVEM project. I appreciate many scintillating talks with Adriel about SoC verification.

I would also like to thank Mr. Kiet To for interesting conversations on more general topics about typical computer structures.

This research work is supported by Australian Research Council Linkage Project (LP0454838). And the Australian Postgraduate Award (Industry) allows me to concentrate on the research.

Finally, I must thank my wife Hongqi Wu, who has given me energy and support throughout the research.
# Contents

Declaration of Authorship iii  
Abstract iv  
Acknowledgements v  
List of Figures xi  
List of Tables xiii  
Abbreviations xv  

## 1 Introduction 1  
1.1 Motivation and Contribution 1  
1.1.1 Motivation 1  
1.1.2 Contribution 3  
1.2 Thesis Overview 5  
1.3 Publications 7  

## 2 Background 9  
2.1 General Verification Practice 9  
2.1.1 Overview 9  
2.1.2 Simulation-Based Verification 10  
2.1.3 Formal Verification 22  
2.2 System-Level Verification Problem 27  
2.2.1 Overview: System-Level Bugs 27  
2.2.2 Formal Methods: Not in the Position 28  
2.2.3 Simulation: DUT-TB Dualism 28  
2.2.4 Software: the Third Entity 31  
2.3 Our Solution: Software-Centric and Interaction-Oriented Verification 33
List of Figures

1.1 The Verification Gap ........................................... 2
1.2 The Verification Gap from Simulation Point of View .......... 4
1.3 Thesis Structure ............................................. 6

2.1 Canonical Test-bench .................................... 16
2.2 Layered Test-bench ....................................... 22
2.3 Model Checking ........................................... 24
2.4 SAT-based Bounded Model Checking ........................... 26
2.5 Test-bench Stimulates and Observes Design-under-test ......... 29
2.6 Both Test-bench and Test-program Stimulate and Observe DUT 32
2.7 Connection between Components Using Channels and Ports ...... 36
2.8 The Nios SoC ............................................... 39
2.9 The Nios Interrupt Sub-system ................................ 40

3.1 Transfer Life Cycle ......................................... 45
3.2 Generalisation of Transfer-types ............................. 50
3.3 Transfer Life-expectancy Affects Test Quality ................. 52
3.4 The TRG for the Nios SoC ................................ 60
3.5 The Petri-net Derived from the TRG of the Nios SoC ......... 68

4.1 Pseudo Code of a Polling-Based Test-Program .................. 76
4.2 Scheduler and Transfers .................................... 77
4.3 Event-driven Test-program: Scheduler and Its Action Table .... 78
4.4 Execution of Different Test-program Structure ................. 80
4.5 Modelling the UART Transmission as a Virtual-Transfer ........ 84
4.6 Enhanced Transfer Model ................................... 86
4.7 General ISR Structure ...................................... 91
4.8 General Soft-Transfer Structure ................................ 93

5.1 The TP-TB-DUT Continuum .................................. 97
5.2 Position the Test-Program and the Test-Bench in the Verification Framework 98
5.3 The Test-Program to Test-Bench Interface ....................... 100
5.4 Test-bench Observes the Software ............................ 104

6.1 Verification Environment .................................... 110
6.2  Toggle and Conditional Coverage Comparison .......................... 112
6.3  State-changes Against Simulation Cycles ............................ 114
6.4  New State Rate Against Known States ............................... 114
6.5  Petri-Net Based State and Transition Coverages with and without Feedback 117
6.6  Hardware Simulation Profiling ......................................... 119
6.7  Basic Test-program Profiling ......................................... 121
6.8  Function-Interrupt Cross ............................................. 124
6.9  Interrupt Nesting Depth Profiling .................................. 124
6.10 Exact Interrupt Nesting Sequences .................................. 125
6.11 Using Profiling to Compare Different Test-Program Structure ........ 126
A.1  The EOP Problem Symptom ......................................... 137
A.2  The Transient Interrupt-Request Problem ......................... 140
B.1  The Register-Window Mechanism of the Nios SoC .................. 144
C.1  Implementation of the Test-Generator ............................. 148
List of Tables

2.1 Simulation-based Verification and Formal Verification .................. 10
2.2 Abstraction Levels ................................................. 12
2.3 Simulation Platforms ................................................ 14
2.4 Combinations of Abstraction-levels, Modelling-languages and Simulation Platforms ................................................. 15
2.5 Verification Methodologies .............................................. 20
3.1 Implement Different Categories of Transfers ........................... 49
3.2 Different Levels of Interactions ........................................ 54
3.3 Typical Physical Resources and Resource Contentions .................. 56
4.1 Incorporating Interrupts into the Transfer Model ...................... 85
5.1 Test-Benches and Test-Programs’ Capabilities to Control and to Observe .......................... 96
7.1 Methodology Differentiation ............................................. 133
## Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALU</td>
<td>Arithmetic Logic Unit</td>
</tr>
<tr>
<td>BDD</td>
<td>Binary Decision Diagram</td>
</tr>
<tr>
<td>BFM</td>
<td>Bus Functional Model</td>
</tr>
<tr>
<td>CTL</td>
<td>Computation Tree Logic</td>
</tr>
<tr>
<td>DMA</td>
<td>Direct Memory Access</td>
</tr>
<tr>
<td>DUT</td>
<td>Design Under Test</td>
</tr>
<tr>
<td>EDA</td>
<td>Electronics Design Automation</td>
</tr>
<tr>
<td>FPGA</td>
<td>Field Programmable Gate Array</td>
</tr>
<tr>
<td>FSM</td>
<td>Finite State Machine</td>
</tr>
<tr>
<td>HDL</td>
<td>Hardware Description Language</td>
</tr>
<tr>
<td>HVL</td>
<td>Hardware Verification Language</td>
</tr>
<tr>
<td>HW</td>
<td>Hardware</td>
</tr>
<tr>
<td>IC</td>
<td>Integrated Circuit</td>
</tr>
<tr>
<td>ISR</td>
<td>Interrupt Service Routine</td>
</tr>
<tr>
<td>OOP</td>
<td>Object Oriented Programming</td>
</tr>
<tr>
<td>OS</td>
<td>Operating System</td>
</tr>
<tr>
<td>RISC</td>
<td>Reduced Instruction Set Computer</td>
</tr>
<tr>
<td>RTL, RT-Level</td>
<td>Register Transfer Level</td>
</tr>
<tr>
<td>SoC</td>
<td>System on Chip</td>
</tr>
<tr>
<td>SW</td>
<td>Software</td>
</tr>
<tr>
<td>TB</td>
<td>Test-Bench</td>
</tr>
<tr>
<td>TLM</td>
<td>Transaction Level Model(ing)</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>TP</td>
<td>Test-Program</td>
</tr>
<tr>
<td>UART</td>
<td>Universal Asynchronous Receiver and Transmitter</td>
</tr>
<tr>
<td>VLSI</td>
<td>Very Large Scale Integration</td>
</tr>
</tbody>
</table>
Dedicated to my girls: Hongqi, Jingyi and Grace.