

Ingredients for Successful System Level Design Methodology

Hiren D. Patel Sandeep K. Shukla

Ingredients for Successful System Level Design Methodology

 Springer

Dr. Hiren D. Patel
Electrical Engineering
and Computer Science
University of California, Berkeley
545K Cory Hall
Berkeley CA 94720
USA
hiren@eecs.berkeley.edu

Dr. Sandeep K. Shukla
Department of Electrical
and Computer Engineering
Virginia Tech
340 Whittemore Hall
Blacksburg VA 24061
USA
shukla@vt.edu

ISBN 978-1-4020-8471-3

e-ISBN 978-1-4020-8472-0

Library of Congress Control Number: 2008927073

All Rights Reserved

© 2008 Springer Science+Business Media B.V.

No part of this work may be reproduced, stored in a retrieval system, or transmitted in any form or by any 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.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

To those affected by April 16, 2007,
the Blacksburg and Virginia Tech. community,
the friends and families, and
all our fellow Hokies.

Hiren D. Patel
and
Sandeep K. Shukla
February 3, 2008

Preface

ESL or “Electronic System Level” is a buzz word these days, in the electronic design automation (EDA) industry, in design houses, and in the academia. Even though numerous trade magazine articles have been written, quite a few books have been published that have attempted to define ESL, it is still not clear what exactly it entails. However, what seems clear to every one is that the “Register Transfer Level” (RTL) languages are not adequate any more to be the design entry point for today’s and tomorrow’s complex electronic system design. There are multiple reasons for such thoughts. First, the continued progression of the miniaturization of the silicon technology has led to the ability of putting almost a billion transistors on a single chip. Second, applications are becoming more and more complex, and integrated with communication, control, ubiquitous and pervasive computing, and hence the need for ever faster, ever more reliable, and more robust electronic systems is pushing designers towards a productivity demand that is not sustainable without a fundamental change in the design methodologies. Also, the hardware and software functionalities are getting interchangeable and ability to model and design both in the same manner is gaining importance.

Given this context, we assume that any methodology that allows us to model an entire electronic system from a system perspective, rather than just hardware with discrete-event or cycle based semantics is an ESL methodology of some kind. One could argue about the applicability, effectiveness and designer friendliness of one ESL methodology over another. One could also argue about the right abstraction level at which systems should be modeled, whether direct algorithmic synthesis from system models to hardware/software should be available, and whether the ESL model should be used only as a reference model without any hope of direct mapping into implementation.

All these are valid arguments which need to be sorted out through continued interaction, discussion, innovation and standardization efforts from industry and academia. These arguments also lie at the very core of the work presented in this book. In fact, as the title implies, we have here a set of prescriptions and solutions that we think is one direction that ESL methodology

should take, if it has to succeed with designers. Of course, we have strong reasons to believe that the current directions in the market place in the ESL domain are not necessarily the correct ones, and hence we have spent several years researching alternative visions, methodologies, language extensions, and implemented prototype of tools that demonstrate the effectiveness of our methods and views.

A word of caution for the readers is probably justified at this point. The work reported in this monograph is the work of one PhD student over about four years, and hence one should not expect that this book provides a panacea for the ESL problems, nor should one expect that the tools described here are industry strength tools that can be downloaded and directly used for the next design project. It is meant to present our academic perspective which is a fruit of our interaction with various design companies as well as EDA companies, and which is enriched at various points in time by discussions with many of our academic colleagues. So it is but an attempt to bring out what we have envisioned and worked out as a few strands in the whole ESL DNA that would eventually spawn the ESL based electronic design industry.

Berkeley, CA
Blacksburg, VA
February 2008

Hiren D. Patel
Sandeep K. Shukla

Acknowledgments

We acknowledge the support received from the NSF CAREER grant CCR-0237947, the NSF NGS program grant ACI-0204028, an SRC Integrated Systems Grant and industrial grant from Bluespec Inc.

Contents

Preface	VII
Acknowledgments	IX
1 Introduction	1
1.1 Motivation	1
1.1.1 Why SystemC?	3
1.1.2 Modeling	3
1.1.3 Simulation	4
1.1.4 Validation	5
1.1.5 Dynamic Integration of Multiple Tools	7
1.2 Organization	8
2 Related Work	11
2.1 System Level Design Languages and Frameworks	11
2.1.1 Ptolemy II	11
2.1.2 SystemC	13
2.1.3 SystemC-H	15
2.1.4 HetSC	20
2.1.5 YAPI	20
2.1.6 Metropolis	21
2.1.7 ForSyDE and SML-Sys	21
2.1.8 Bluespec BSC	22
2.1.9 API and PLI Approaches	22
2.1.10 Term Rewriting Systems	22
2.1.11 TRS Modeling	23
2.2 Verification of SystemC Designs	24
2.3 Reflection and Introspection	26
2.3.1 Existing Tools for Structural Reflection	26
2.3.2 Edison Group's C/C++ Front-end (EDG)	26

2.3.3	Pinapa	27
2.3.4	SystemCXML	27
2.3.5	SystemPerl	27
2.3.6	Karlsruhe SystemC Parser Suite	28
2.3.7	ESys.NET Framework and Introspection in SystemC ...	28
2.3.8	BALBOA Framework	29
2.3.9	Meta-model Based Visual Component Composition Framework (MCF)	30
2.3.10	Java, C# .NET Framework, C++ RTTI	30
2.4	Service-orientation	31
2.4.1	Adaptive Communication Environment (ACE) and the ACE Orb Way (TAO)	31
2.4.2	Service-oriented Software	31
3	Background	33
3.1	Fidelity, Expressiveness and Multiple Models of Computation	33
3.1.1	Synchronous Data Flow MoC (SDF)	35
3.1.2	Finite State Machines MoC (FSMs)	36
3.1.3	Communicating Sequential Processes (CSP)	37
3.1.4	Abstract State Machines (ASMs)	42
3.1.5	SpecExplorer	43
3.1.6	Discrete-Event Semantics Using ASMs	43
3.1.7	Exploration in SpecExplorer	44
4	Behavioral Hierarchy with Hierarchical FSMs (HFSMs) ...	47
4.1	Behavioral Modeling versus Structural Modeling	49
4.2	Finite State Machine Terminology	50
4.3	Requirements for Behavioral Hierarchy in SystemC	51
4.3.1	Requirements for Hierarchical FSMs	52
4.3.2	Additional Semantic Requirement for Hierarchical FSMs	52
4.4	Execution Semantics for Hierarchical FSMs	53
4.4.1	Abstract Semantics	53
4.4.2	Basic Definitions for HFSMs	54
4.4.3	Execution Semantics for Starcharts	56
4.4.4	Our Execution Semantics for Hierarchical FSMs	57
4.5	Implementation of Hierarchical FSMs	63
4.5.1	Graph Representation Using Boost Graph Library	63
4.5.2	Graph Representation for HFSMs	64
4.5.3	HFSM Graph Representing the HFSM Model	68
4.6	Modeling Guidelines for HFSM	71
4.7	HFSM Example: Power Model	74

5	Simulation Semantics for Heterogeneous Behavioral Hierarchy	77
5.1	Abstract Semantics	77
5.2	Basic Definitions	78
5.3	Execution Semantics for Starcharts	80
5.4	Our Execution Semantics for Hierarchical FSMs	83
5.4.1	Additional Definitions Specific for HFSMs	83
5.4.2	Redefinition for Heterogeneous Behavioral Hierarchical FSMs and SDFs	84
5.5	Implementing Heterogeneous Behavioral Hierarchy	85
5.5.1	Graph Representation Using Boost Graph Library	85
5.5.2	Implementing the Execution Semantics	86
5.6	Examples	89
5.6.1	Polygon In-fill Processor	89
6	Bluespec ESL and its Co-simulation with SystemC DE	93
6.1	Advantages of this Work	95
6.2	Design Flow	96
6.3	BS-ESL Language	97
6.3.1	BS-ESL Modules	98
6.3.2	BS-ESL Rules	99
6.3.3	BS-ESL Methods	100
6.3.4	Interfaces	101
6.3.5	BS-ESL Primitive Modules	102
6.3.6	Higher Abstraction Mechanisms	103
6.4	BS-ESL Execution	107
6.5	An Example Demonstrating BS-ESL and SystemC Integration	107
6.6	Summary	108
6.7	Interoperability between SystemC and BS-ESL	109
6.7.1	Problem Statement: Interoperability between SystemC and BS-ESL	110
6.7.2	Interoperability Issues at RTL	110
6.7.3	Interoperability Issues at TL	110
6.8	Problem Description	111
6.8.1	DE Simulation Semantics for RTL	112
6.8.2	BS-ESL's Rule-based Semantics	115
6.8.3	Composing HDL RTL Models	116
6.8.4	Composing HDL RTL & BS-ESL Models	117
6.9	Solution: Our Interoperability Technique	119
6.9.1	HDL RTL with BS-ESL RTL	119
6.9.2	Example	121
6.9.3	Simulation Results	123
6.10	Summary	124

7	Model-driven Validation of SystemC Designs	125
7.1	Overview of this Work	127
7.2	Design Flow	128
7.2.1	Semantic Modeling and Simulation	128
7.2.2	SystemC Modeling, Simulation and Wrappers	129
7.2.3	C# Interface for SpecExplorer and SystemC	130
7.2.4	Validation, Test Case Generation and Execution	131
7.3	Results: Validation of FIFO, FIR and GCD	132
7.3.1	FIFO	132
7.3.2	GCD	138
7.3.3	FIR	140
7.4	Our Experience	141
7.5	Evaluation of this Approach	142
7.5.1	Benefits of this Approach	142
7.5.2	Drawbacks of this Approach	144
7.6	Summary	147
8	Service-orientation for Dynamic Integration of Multiple Tools	149
8.1	CARH's Capabilities	150
8.2	Issues and Inadequacies of Current SDLs and Dynamic Validation Frameworks	151
8.3	Our Generic Approach to Addressing these Inadequacies	153
8.3.1	Service Orientation	153
8.3.2	Introspection Architecture	153
8.3.3	Test Generation and Coverage Monitor	154
8.3.4	Performance Analysis	154
8.3.5	Visualization	155
8.4	CARH's Software Architecture	156
8.5	Services Rendered by CARH	158
8.5.1	Reflection Service	158
8.5.2	Testbench Generator	163
8.5.3	d-VCD Service	166
8.6	Usage Model of CARH	168
8.7	Simulation Results	171
8.8	Our Experience with CARH	172
9	Summary Evaluations	177
9.1	Modeling and Simulating Heterogeneous Behaviors in SystemC	177
9.2	Validating Corner Case Scenarios for SystemC	179
9.3	Dynamic Integration of Multiple Tools	180
10	Conclusion and Future work	181

- A Parsing SystemC using C/C++ Front-end Compiler** 187
 - A.1 Tool Flow 187
 - A.2 Parsing SystemC 188

- B Eclipse-based Plugin for a SystemC IDE** 193
 - B.1 Project Overview 193
 - B.2 SystemC IDE Feature 193
 - B.3 SystemC IDE Plug-in 194
 - B.3.1 Plug-in Source 194
 - B.4 Setting up the SystemC IDE 195
 - B.4.1 Setup the SystemC Preferences 195
 - B.4.2 Creating a Managed SystemC Project 195
 - B.4.3 Adding Files to the Project 197
 - B.4.4 Executing a SystemC Project 197
 - B.5 A Little About Implementation 197
 - B.5.1 The Plug-In File 197
 - B.5.2 The Editor 198
 - B.5.3 The Preference Page 199
 - B.5.4 The Wizards 199

- References** 201