

Applications of Specification and Design Languages for SoCs

Edited by
A. Vachoux

The ChDL series

 Springer

APPLICATIONS OF SPECIFICATION
AND DESIGN LANGUAGES FOR SOCS

Applications of Specification and Design Languages for SoCs

Selected papers from FDL 2005

Edited by

A. VACHOUX

*Ecole Polytechnique Fédérale de Lausanne,
Lausanne, Switzerland*

 Springer

A C.I.P. Catalogue record for this book is available from the Library of Congress.

ISBN-10 1-4020-4997-8 (HB)
ISBN-13 978-1-4020-4997-2 (HB)
ISBN-10 1-4020-4998-6 (e-book)
ISBN-13 978-1-4020-4998-9 (e-book)

Published by Springer,
P.O. Box 17, 3300 AA Dordrecht, The Netherlands.

www.springer.com

Printed on acid-free paper

All Rights Reserved
© 2006 Springer

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.

Contents

List of Figures	xiii
List of Tables	xvii
List of Listings	xix
Preface	xxi
Part I Specification, Design, and Verification Methods	
Introduction	3
<i>Alain Vachoux</i>	
1	
PSL-Based Online Monitoring of Digital Systems	5
<i>Dominique Borrione, Miao Liu, Pierre Ostier, and Laurent Fesquet</i>	
1. Introduction	5
1.1 State of the Art	6
1.2 PSL as a Design Language	7
2. Monitor Construction: Principles	9
2.1 Property Satisfaction	9
2.2 Library of Primitive Components	10
2.3 Structure of a Primitive Monitor	12
2.4 Construction of Complex Monitors	13
3. Validation	15
3.1 Functional Comparison	15
3.2 Area Comparison	16
4. Implementation of Monitors	17
4.1 Design Flow with Assertion Monitors	17
4.2 Example: A Bus Snoop System for Software Verification	19
5. Conclusion	20
Acknowledgments	21
References	21
2	
Refining Synchronous Communication onto Network-on-Chip Best-Effort Services	23
<i>Zhonghai Lu, Ingo Sander, and Axel Jantsch</i>	
1. Introduction	23
2. Related Work	25

3. Refinement Overview	26
3.1 The Perfectly Synchronous Model	26
3.2 Nostrum Communication Services	27
3.3 The Refinement Procedure	28
4. Channel Refinement	29
5. Process Refinement	30
5.1 Interfacing with the Service Channels	30
5.2 Process Synchronization Property	31
5.3 Achieving Synchronization Consistency	32
5.4 Feedback Loops	33
6. Communication Mapping	34
6.1 Channel Mapping	34
6.2 Communication Process Mapping	34
7. Conclusions and Future Work	37
References	37
Part II C/C++-Based System Design	
Introduction	41
<i>Frank Oppenheimer</i>	
3	
Behaviour Separation: A High-Level Methodology Applicable in the SystemC Environment	43
<i>Giovanni B. Vece, Massimo Conti, and Simone Orcioni</i>	
1. Introduction	43
2. Principles of the Behaviour Separation Methodology	45
3. Application for Communication Protocols	47
4. Realization in SystemC	49
5. Application Example Based on an AMBA AHB Master Device	50
6. I/O Adaptation; Limitations and Application Fields	54
7. Modelling of Complete AMBA AHB Master Devices and Results	55
8. Extension to Generic Protocols	56
9. Conclusions	58
References	58
4	
Mixing Synchronous Reactive and Untimed MoCs in SystemC	61
<i>Fernando Herrera and Eugenio Villar</i>	
1. Introduction	62
2. Mapping of SR and Untimed MoCs to SystemC	66
3. Untimed–SR MoC Interfaces	73
4. Conclusions	79
References	80
5	
Interface-Centric Abstraction Level for Rapid Hardware/Software Integration	83
<i>André C. Nácul, Marcello Lajolo, and Tony Givargis</i>	
1. Introduction	83

<i>Contents</i>	vii
2. Related Work	85
3. Terminology	86
4. System-Level API	87
4.1 Interface Synthesis	90
4.2 RTOS Synthesis	93
4.3 Our Hardware/Software Codesign Environment	94
5. Hardware/Software Integration	95
6. Conclusions	97
References	97
6	
Efficient and Customizable Integration of Temporal Properties into SystemC	101
<i>Roland J. Weiss, Jürgen Ruf, Thomas Kropf, and Wolfgang Rosenstiel</i>	
1. Introduction	101
2. Property Synthesis	102
2.1 Intermediate Language	104
3. Integrating Temporal Properties into SystemC	104
3.1 Property Specification	106
3.2 Property Checking	106
3.3 Customizing Actions with Policies	108
4. Experimental Results	108
4.1 Memory Consumption	109
4.2 Run-time Performance	109
5. Related Work	110
6. Conclusions and Future Work	111
Acknowledgments	112
References	112
7	
UMoC++: A C++-Based Multi-MoC Modeling Environment	115
<i>Deepak A. Mathaikutty, Hiren D. Patel, Sandeep K. Shukla, and Axel Jantsch</i>	
1. Introduction	116
2. Related Work	117
3. Generic MoCs	117
3.1 Preliminary Notations	118
3.2 Generic MoCs Formulation in SML-Sys	119
3.3 Untimed Model of Computation	119
4. Essential Concepts from FL mapped to C++ for our Implementation	121
4.1 Polymorphic Types	121
4.2 Higher-Order Functions	122
4.3 Partial Application	122
5. Generic MoCs Formulation in C++	123
5.1 UMoC++ Framework	124
5.2 Process Constructors	124
5.3 Process Combinators	125
6. Example of Models in our Framework	126
6.1 Petri Net Style Modeling Using UMoC++	126
6.2 Synchronous Data Flow Style Modeling Using UMoC++	127
6.3 Cosimulation With SystemC	128

viii	<i>Contents</i>
7. Conclusion	128
References	129
Part III Analog, Mixed-Signal, and Heterogeneous System Design	
Introduction	133
<i>Christoph Grimm</i>	
8	
Creating Virtual Prototypes of Complex MEMS Transducers Using Reduced-Order Modelling Methods and VHDL-AMS	135
<i>Torsten Mähne, Kersten Kehr, Axel Franke, Jörg Hauer, and Bertram Schmidt</i>	
1. Introduction	136
2. Theory of the Reduced-Order Modelling Method	137
3. Micromechanical Yaw Rate Sensor	138
4. Preparation of the FE Models for the ROM Method	140
5. Generation of the Reduced-Order Behavioural Models	140
6. Integration of the Reduced-Order Behavioural Models	143
7. Simulation of the Complete Sensor System	146
8. Conclusions	150
Acknowledgments	151
References	151
9	
Modeling Uncertainty in Nonlinear Analog Systems with Affine Arithmetic	155
<i>Wilhelm Heupke, Christoph Grimm, and Klaus Waldschmidt</i>	
1. Introduction	155
2. Semisymbolic Simulation with Affine Arithmetic	157
2.1 Basic Concepts of Affine Arithmetic	157
2.2 Interval Arithmetic versus Affine Arithmetic	158
2.3 SystemC-AMS-Based Implementation	159
3. Efficient Handling of Additional Terms in Feedback Control Systems	161
3.1 Implementation of the Simplification Method	162
3.2 Comparison of Efficiency	163
4. Experimental Results	165
5. Conclusion	166
References	168
10	
SystemC-WMS: Mixed-Signal Simulation Based on Wave Exchanges	171
<i>Simone Orcioni, Giorgio Biagetti, and Massimo Conti</i>	
1. Introduction	171
2. Description and Modeling of Analog Modules in SystemC	173
2.1 Module Representation with a b Parameters	174
2.2 Wavechannels	176
3. SystemC-WMS Class Library	178
4. Application Example	179
4.1 Simulation Results	183
5. Conclusion	184
References	184

<i>Contents</i>	ix
11	
Automatic Generation of a Coverification Platform	187
<i>Suad Kajtazovic, Christian Steger, Andreas Schuhai, and Markus Pistauer</i>	
1. Introduction	187
2. Related Work	188
2.1 Summary	189
3. Design Methodology	190
3.1 System Design Level	190
3.2 Language Level	191
3.3 Simulator Level	191
4. Design of a Cosimulation Interface	191
4.1 Interfacing Between Simulators	191
4.2 Communication	193
4.3 Data Type Conversion	194
4.4 Synchronization	195
4.5 Cosimulation Interface	195
5. Automatic Code Generation	196
6. Experimental Example	198
6.1 Design Steps	198
6.2 Results	200
7. Conclusion	202
References	202
12	
UML/XML-Based Approach to Hierarchical AMS Synthesis	205
<i>Ian O'Connor, Faress Tissafi-Drissi, Guillaume Révy, and Frédéric Gaffiot</i>	
1. Introduction	205
2. AMS IP Element Requirements for Synthesis Tools	206
3. UML in AMS Design	210
3.1 Reasons for Using UML in Analogue Synthesis	210
3.2 Mapping AMS IP Requirements to UML Concepts	211
3.3 Modelling Analogue Synthesis with Activity Diagrams	213
4. Extensions to Existing Analogue Synthesis Tool (runeII)	214
4.1 AMS Soft-IP Definition	216
4.2 AMS Firm-IP Synthesis	216
5. Example	218
5.1 Class Diagram Example	219
5.2 Soft-IP XML File Example	220
5.3 Optimisation Scenario Example	220
5.4 Firm-IP XML Output File Example	220
6. Conclusion	220
Acknowledgments	224
References	224
Part IV UML-Based System Specification and Design	
Introduction	229
<i>Piet van der Putten</i>	

x	<i>Contents</i>
13	
Compiled and Synthesized UML	231
<i>Cathy Berthouzoz, François Corthay, Medard Rieder, Rico Steiner, and Thomas Sterren</i>	
1. Introduction	232
2. Codesign	232
3. A Theoretical Codesign Approach	233
4. A Practical Codesign Approach	235
5. Translation	236
5.1 Hardware Thinks Differently	236
5.2 UML Elements	237
5.3 UML to VHDL Mapping	238
6. Experimentation	242
7. Conclusions	243
7.1 Tool Chain Improvement	243
7.2 The 6qx Process	244
References	245
14	
Property-Preservation Synthesis for Unified Control- and Data-Oriented Models	247
<i>Oana Florescu, Jeroen Voeten, and Henk Corporaal</i>	
1. Introduction	247
2. Related Research	249
3. Real-Time Systems Models	250
4. From a Model to Its Realisation	253
5. Realisation of Systems with Time-Intensive Computations	256
6. Conclusions and Future Work	260
Acknowledgments	261
References	261
15	
Traceability and Interoperability at Different Levels of Abstraction in Model-Driven Engineering	263
<i>Lossan Bondé, Pierre Boulet, and Jean-Luc Dekeyser</i>	
1. Introduction	264
2. Metamodel for Traceability in Model Transformations	264
2.1 Concepts and Overview of the Metamodel	265
2.2 More Details on the Metamodel	266
3. Generation of the Trace Model	267
3.1 Principle of <i>TraceModel</i> Generation	268
3.2 Example	269
4. Getting Interoperability from Traceability	271
4.1 Proposed Approach for Interoperability	271
4.2 Application of the Approach on an Example	273
5. Conclusion	275
References	275
16	
Power Simulation of Communication Protocols with StateC	277
<i>Luca Negri and Andrea Chiarini</i>	
1. Introduction	277

2. The StateC Flow	279
3. Implementation-Independent Model	281
3.1 Statecharts Modeling of a Protocol Stack	281
3.2 Logical Activities Identification and Localization	282
4. Implementation-Dependent Model	283
4.1 Model Characterization	283
4.2 Training and Validating the Model	284
5. Power Simulation	285
5.1 Automatic Simulator Generation	285
5.2 Simulator Usage	288
6. Experimental Results	289
6.1 Implementation-Independent Models	289
6.2 Power Characterization of the Models	290
6.3 Simulator Performance	291
7. Conclusions and Future Work	292
References	293
17	
Integrating Model-Checking with UML-Based SoC Development	295
<i>Peter Green and Kinika Tasie-Amadi</i>	
1. Introduction	296
2. Overview of the Approach	296
3. Background	297
3.1 Overview of CSP and FDR	298
3.2 Previous Approaches to the Checking of UML Models	298
3.3 UML State Machines	299
4. Translating State Machines to CSP	300
4.1 Flattening State Machines	301
4.2 Realizing State Machine Semantics in CSP	302
5. Mapping the Models to CSP	304
5.1 Use Case Models	304
5.2 Interaction Models	304
5.3 The Composite Object Model	305
6. The UML2CSP Tool	306
7. Applying FDR to Translated Specifications	307
8. Partial Case Study	308
9. Conclusions	310
References	311

List of Figures

1.1	Property status and monitor output	10
1.2	Property monitor for P1	11
1.3	Structure of a primitive monitor	13
1.4	Tree structure of PSL property P1: <code>always (req -> next((next_e[1:2](done)) until grant)) rising_edge(clk);</code>	14
1.5	Comparison flow	15
1.6	Design flow to implement assertion monitors	18
1.7	The bus snoop system implemented on an FPGA with assertion monitor	19
2.1	An NoC design flow	24
2.2	The digital equalizer	26
2.3	Communication refinement overview	27
2.4	Processes for synchronization	32
2.5	Read/write adapters for a process with strong synchronization	33
2.6	Read/write adapters for a process with strict synchronization	35
2.7	The equalizer mapped on an NoC	36
3.1	Protocol rules classification in fixed and free protocol rules	45
3.2	Generic architecture for communication protocol-bound devices	47
3.3	Device architecture in the SystemC environment	49
3.4	Bound unit I/O adaptation	55
3.5	Device architecture for protocols concerning internal behaviour	57
4.1	Heterogeneity and abstraction in the specification	63
4.2	SR reactive process styles in SystemC	68
4.3	Untimed MoCs P. O. over the SystemC time axis	70
4.4	SR MoC adds T. O. in the reactive chain and perfect synchrony	71
4.5	SR reactive chain with a feedback loop	73
4.6	Perfect synchrony and P. O fulfilling in CSP-SR connection	74
4.7	Untimed-SR MoC interface in the MoC interface taxonomy	75
4.8	Timing and blocking semantics in Untimed-SR interfaces	76

4.9	Dynamic check of reaction time	77
4.10	Types of border processes in KPN–SR MoC interface	78
4.11	Border channels in KPN–SR interface	78
5.1	Interface synthesis for hardware-to-hardware communication	90
5.2	Interface synthesis for software-to-software communication	91
5.3	Interface synthesis for hardware-to-software communication	92
5.4	Interface synthesis for multiprocessor communication	93
5.5	Code example	94
5.6	Hardware/Software integration	96
6.1	Example of an AR-automaton for a simple FLTL property. The state labeled with R is the rejecting state.	103
6.2	Outline of the IL approach	105
6.3	Memory consumption for properties P1 and P2	109
6.4	Run-time comparison for arbiter example	110
7.1	Parallel, sequential, and feedback operators	120
7.2	Mapping of the Amplifier PN to a Petri net	127
7.3	An FIR Filter cosimulated using UMoC++ and SystemC	128
8.1	Yaw rate sensor manufactured by Robert Bosch GmbH	139
8.2	Steps to prepare the coupled FE models of the yaw rate sensor for the generation of its in-plane and out-of-plane ROMs: (a) single-domain FE models, (b) preparation of the coupled FE models, and (c) electromechanically coupled FE models	141
8.3	Steps to generate the reduced-order models of the yaw rate sensor	142
8.4	Structure of the yaw rate sensor full model with the in-plane and out-of-plane ROMs	144
8.5	Structure of the test bench for the yaw rate sensor full model	147
8.6	Simulated self-excitation of the yaw rate sensor	148
8.7	Simulated rate detection of the yaw rate sensor	149
9.1	Simulated system with nonlinear block	164
9.2	Step responses of a feedback loop containing a nonlinear block	167
10.1	Example of interconnection problem	174
10.2	Wavechannel symbols for parallel and series interconnections	176
10.3	Half-bridge inverter: electrical schematic diagram	180
10.4	Simulation results of the half-bridge inverter	183
11.1	System description	190
11.2	Simulator interfacing principle	192
11.3	Proposed coupling mechanism	193

11.4	Simple example of the CsBlock concept	194
11.5	Simulator interface structure	196
11.6	Class diagram of the cosimulation interface generator	197
11.7	Flow of the automatic code generator	197
11.8	A system overview of an automotive power management system	199
11.9	System overview	199
11.10	Configuration of the Cosimulation Platform	200
11.11	Cosimulation results	201
11.12	Signal comparison: generator and board voltage in simulation and cosimulation	201
12.1	AMS synthesis loop showing AMS IP facet use	209
12.2	UML representation of AMS IP hierarchical dependencies	212
12.3	UML class definitions for AMS IP blocks	212
12.4	Activity diagram for TIA block synthesis process	215
12.5	UML/XML use flow in runeII	215
12.6	Screenshot of the runeII GUI	217
12.7	TIA and amplifier in an integrated optical link	218
12.8	TIA and resistive feedback classes in UML	219
13.1	Traditional embedded system development cycle	232
13.2	Different degrees of partitioning	234
13.3	Model-driven codesign of embedded systems	234
13.4	A practical codesign approach	235
13.5	UML object diagram	238
13.6	VHDL representation (a) of a single UML state (b)	241
13.7	Principle of the chronometer codesign demonstrator	242
13.8	Overview of the 6qx codesign process	244
14.1	SHE method for real-time systems design	250
14.2	Example of a timed labelled transition system	251
14.3	Two phases of model execution	252
14.4	A timed trace of the transition system	252
14.5	The UML model of a simple controller	252
14.6	The timed labelled transition system of the model	253
14.7	A timed trace of the controller	253
14.8	Timed traces ϵ -close	254
14.9	Implementation of the controller in physical time	255
14.10	Y-chart scheme for real-time systems design	256
14.11	Observational-equivalent model timed trace	257
14.12	Implementation of the equivalent model in physical time	258
14.13	A possible execution that still preserves the properties	259

15.1	Overview of the metamodel	266
15.2	Detailed metamodel	268
15.3	Mapping ModTransf rules to trace element	269
15.4	Simple UML class translated into a Java class	270
15.5	Interoperability bridging from trace information	272
15.6	A simple PIM model transformed into two PSM models	273
16.1	The StateC power modeling and simulation flow	279
16.2	Common pattern of communication between layers of the stack	282
16.3	Automated Statecharts to SystemC transformation	286
16.4	State template for SystemC simulator. Dark gray shaded texts are the only parts that change from one state to the other.	288
16.5	Partial view of full Bluetooth Statecharts model; subset of states used for controlling inquiry procedures. Logical activities are highlighted with bold arrows.	290
16.6	Partial view of full 802.11 Statecharts model; subset of states used for packet transmission in distributed coordination function (DCF) mode.	291
17.1	Hierarchical state machine and its flat-equivalent	300
17.2	Concurrent state machine and its flat-equivalent	301
17.3	Flattening and completion transitions	302
17.4	Object state machine and its simplified representation in CSP	303
17.5	Simple sequence diagram	305
17.6	Modified object state machine facilitating dynamic object creation/destruction	306
17.7	Structure of the UML2CSP tool	306
17.8	Use case monitor water level	309
17.9	Checking an interaction against a use case	309
17.10	Desirable property and the results of a refinement check	310

List of Tables

1.1	Components in the library	11
1.2	Parameters	12
1.3	List of properties for library validation	16
1.4	List of complex properties with nested temporal operators	17
1.5	Area comparison results	17
3.1	Code data for different master devices	56
5.1	The API functions	87
6.1	The categorized IL statements	105
9.1	Affine expressions and their interval counterparts	158
9.2	Measured computation time	165
12.1	AMS IP block facets	208
12.2	Mapping of AMS IP requirements to class structure	214

List of Listings

6.1	The IL code for formula $G(a \rightarrow X b)$. The left column gives the code location and the statement's opcode, separated by a colon.	105
6.2	The main loop of the checker process	107
7.1	Mealy-based process constructor in SML-Sys	120
7.2	Mealy-based process constructor in C++	125
7.3	Sequential composition	126
11.1	Basic synchronization algorithm	195
12.1	Entity/functional and structural model DTD template	217
12.2	Entity/functional model description output in XML	221
12.3	Structural model description output in XML	221
12.4	TransimpedanceAmplifier/RFeedback optimisation scenario description in Java	222
12.5	Firm-IP synthesis results in XML	223
13.1	Generated code of UML diagram in Figure 13.5	240
14.1	POOSL model of the simple controller	253
14.2	Observational-equivalent model of the controller	257
14.3	Example of model without observational equivalence	260

Preface

The Forum on specification and Design Languages (FDL) is the premier European forum to exchange experiences and learn about new trends in the application of languages and models for the specification and modeling of electronic systems. FDL'05 was organized around four thematic areas that cover essential aspects of system-level design methods and tools: “C/C++-Based System Design” (chaired by Frank Oppenheimer, OFFIS, Germany), “Analog, Mixed-Signal, and Heterogeneous System Design” (chaired by Christoph Grimm, University of Hannover, Germany), “UML-Based System Specification and Design” (chaired by Piet van der Putten, TU Eindhoven, The Netherlands), and “Specification, Design, and Verification Methods” (chaired by Alain Vachoux, EPFL, Switzerland). This book includes a collection of outstanding contributions to FDL'05 that have been carefully selected by the thematic area Chairs and thoroughly revised by the authors. The book has 17 chapters grouped in four parts. Each part groups chapters related to one thematic area and is introduced by its respective FDL'05 Chair:

- Part I, “Specification, Design, and Verification Methods,” includes two chapters covering system design issues of growing importance, namely assertion-based design and network-on-chip (NoC) design flow.
- Part II, “C/C++-Based System Design,” includes five chapters mostly addressing issues related to the development and the use of SystemC for hardware/software system-level design.
- Part III, “Analog, Mixed-Signal, and Heterogeneous System Design,” includes five chapters discussing how the design of mixed-signal/mixed-technology systems may be handled at system level.
- Part IV, “UML-Based System Specification and Design,” concludes the book with five chapters exploring modeling methodologies that can map abstract models of complex systems onto efficient implementations.

The book is providing an excellent coverage of recent achievements in the use of languages and models for the specification and the design of systems-on-chip (SoCs). It also highlights the diversity of the issues that have to be tackled

with in today's system designs and the creativity of researchers to develop efficient solutions. Last, but not least, the book shows the importance of the FDL event as the place to discuss issues related to electronic system design.

On a final note, I would like to warmly thank Torsten Mähne for his excellent work in helping me editing the book. Torsten efficiently managed all the subtle \LaTeX issues that arose and highly contributed to make all chapters in the book consistent.

Alain Vachoux
FDL'05 General Chair
Ecole Polytechnique Fédérale de Lausanne (EPFL)
Lausanne, Switzerland
March 2006

Chapter 1

PSL-Based Online Monitoring of Digital Systems

Dominique Borrione, Miao Liu, Pierre Ostier, and Laurent Fesquet

TIMA

46 Avenue Félix Viallet

38031 Grenoble cedex

France

Abstract We present an original method for generating monitors that capture the occurrence of events, specified by logical and temporal properties under the form of assertions in declarative form, written in the PSL standard. The method includes a library of primitive digital components and a technique to interconnect them, resulting in a synthesizable digital module that can be properly connected to a digital system under verification, or to a set of input signals under scrutiny. The complexity of the generation is proportional to the size of the PSL expression. A prototype emulation system has been implemented.

Keywords: property; monitor; PSL.

1. Introduction

Today's increasing design complexity requires innovative methods for verification and debug. With verification consuming up to 70% of the design cycle, assertion-based design (Foster et al., 2003) is viewed as one key method for improving productivity. An assertion is a design property that is declared to be true and should be evaluated by one or more techniques among simulation, emulation, or formal verification. The introduction of new standard languages such as Property Specification Language (PSL) or SystemVerilog has made assertions more easy to write and very powerful. An assertion can also be seen as a high-level functional specification for a circuit intended for monitoring of events over time.