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
Part I Specification, Design, and Verification Methods

Introduction

Alain Vachoux

1

PSL-Based Online Monitoring of Digital Systems

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

1. Introduction

1.1 State of the Art

1.2 PSL as a Design Language

2. Monitor Construction: Principles

2.1 Property Satisfaction

2.2 Library of Primitive Components

2.3 Structure of a Primitive Monitor

2.4 Construction of Complex Monitors

3. Validation

3.1 Functional Comparison

3.2 Area Comparison

4. Implementation of Monitors

4.1 Design Flow with Assertion Monitors

4.2 Example: A Bus Snoop System for Software Verification

5. Conclusion

Acknowledgments

References

2

Refining Synchronous Communication onto Network-on-Chip Best-Effort Services

Zhonghai Lu, Ingo Sander, and Axel Jantsch

1. Introduction

2. Related Work
## Contents

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  
Roland J. Weiss, Jürgen Ruf, Thomas Kropf, and Wolfgang Rosenstiel  
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  
Deepak A. Mathaikutty, Hiren D. Patel, Sandeep K. Shukla, and Axel Jantsch  
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
Part III  Analog, Mixed-Signal, and Heterogeneous System Design

Introduction

Christoph Grimm

8
Creating Virtual Prototypes of Complex MEMS Transducers Using Reduced-Order Modelling Methods and VHDL-AMS

Torsten Mähne, Kersten Kehr, Axel Franke, Jörg Hauer, and Bertram Schmidt

1. Introduction
2. Theory of the Reduced-Order Modelling Method
3. Micromechanical Yaw Rate Sensor
4. Preparation of the FE Models for the ROM Method
5. Generation of the Reduced-Order Behavioural Models
6. Integration of the Reduced-Order Behavioural Models
7. Simulation of the Complete Sensor System
8. Conclusions

Acknowledgments
References

9
Modeling Uncertainty in Nonlinear Analog Systems with Affine Arithmetic

Wilhelm Heupke, Christoph Grimm, and Klaus Waldschmidt

1. Introduction
2. Semisymbolic Simulation with Affine Arithmetic
   2.1 Basic Concepts of Affine Arithmetic
   2.2 Interval Arithmetic versus Affine Arithmetic
   2.3 SystemC-AMS-Based Implementation
3. Efficient Handling of Additional Terms in Feedback Control Systems
   3.1 Implementation of the Simplification Method
   3.2 Comparison of Efficiency
4. Experimental Results
5. Conclusion
References

10
SystemC-WMS: Mixed-Signal Simulation Based on Wave Exchanges

Simone Orcioni, Giorgio Biagetti, and Massimo Conti

1. Introduction
2. Description and Modeling of Analog Modules in SystemC
   2.1 Module Representation with $a b$ Parameters
   2.2 Wavechannels
3. SystemC-WMS Class Library
4. Application Example
   4.1 Simulation Results
5. Conclusion
References
Contents

11 Automatic Generation of a Coverification Platform 187
Suad Kajtazovic, Christian Steger, Andreas Schuhai, and Markus Pistauer

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
Ian O'Connor, Faress Tissafi-Drissi, Guillaume Révy, and Frédéric Gaffiot

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
Piet van der Putten
### Compiled and Synthesized UML

*Cathy Berthouzoz, François Corthay, Medard Rieder, Rico Steiner, and Thomas Sterren*

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>232</td>
</tr>
<tr>
<td>2. Codesign</td>
<td>232</td>
</tr>
<tr>
<td>3. A Theoretical Codesign Approach</td>
<td>233</td>
</tr>
<tr>
<td>4. A Practical Codesign Approach</td>
<td>235</td>
</tr>
<tr>
<td>5. Translation</td>
<td>236</td>
</tr>
<tr>
<td>5.1 Hardware Thinks Differently</td>
<td>236</td>
</tr>
<tr>
<td>5.2 UML Elements</td>
<td>237</td>
</tr>
<tr>
<td>5.3 UML to VHDL Mapping</td>
<td>238</td>
</tr>
<tr>
<td>6. Experimentation</td>
<td>242</td>
</tr>
<tr>
<td>7. Conclusions</td>
<td>243</td>
</tr>
<tr>
<td>7.1 Tool Chain Improvement</td>
<td>243</td>
</tr>
<tr>
<td>7.2 The 6qx Process</td>
<td>244</td>
</tr>
</tbody>
</table>

### Property-Preservation Synthesis for Unified Control- and Data-Oriented Models

*Oana Florescu, Jeroen Voeten, and Henk Corporaal*

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>247</td>
</tr>
<tr>
<td>2. Related Research</td>
<td>249</td>
</tr>
<tr>
<td>3. Real-Time Systems Models</td>
<td>250</td>
</tr>
<tr>
<td>4. From a Model to Its Realisation</td>
<td>253</td>
</tr>
<tr>
<td>5. Realisation of Systems with Time-Intensive Computations</td>
<td>256</td>
</tr>
<tr>
<td>6. Conclusions and Future Work</td>
<td>260</td>
</tr>
</tbody>
</table>

Acknowledgments

References | 261 |

### Traceability and Interoperability at Different Levels of Abstraction in Model-Driven Engineering

*Lossan Bondé, Pierre Boulet, and Jean-Luc Dekeyser*

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>264</td>
</tr>
<tr>
<td>2. Metamodel for Traceability in Model Transformations</td>
<td>264</td>
</tr>
<tr>
<td>2.1 Concepts and Overview of the Metamodel</td>
<td>265</td>
</tr>
<tr>
<td>2.2 More Details on the Metamodel</td>
<td>266</td>
</tr>
<tr>
<td>3. Generation of the Trace Model</td>
<td>267</td>
</tr>
<tr>
<td>3.1 Principle of TraceModel Generation</td>
<td>268</td>
</tr>
<tr>
<td>3.2 Example</td>
<td>269</td>
</tr>
<tr>
<td>4. Getting Interoperability from Traceability</td>
<td>271</td>
</tr>
<tr>
<td>4.1 Proposed Approach for Interoperability</td>
<td>271</td>
</tr>
<tr>
<td>4.2 Application of the Approach on an Example</td>
<td>273</td>
</tr>
<tr>
<td>5. Conclusion</td>
<td>275</td>
</tr>
</tbody>
</table>

References | 275 |

### Power Simulation of Communication Protocols with StateC

*Luca Negri and Andrea Chiarini*

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>277</td>
</tr>
</tbody>
</table>
Contents

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

Peter Green and Kinika Tasie-Amadi

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