

# **Embedded Systems and Software Validation**

Abhik Roychoudhury  
National University of Singapore



*To Jishnu*



# Contents

<b>1</b>	<b>Introduction</b>	<b>7</b>
<b>2</b>	<b>Model Validation</b>	<b>13</b>
2.1	Platform vs System Behavior . . . . .	15
2.2	Criteria for Design Model . . . . .	17
2.3	Informal Requirements: A Case Study . . . . .	19
2.3.1	The Requirements Document . . . . .	21
2.3.2	Simplification of the Informal Requirements . . . . .	21
2.4	Common Modeling Notations . . . . .	24
2.4.1	Finite State Machines (FSM) . . . . .	24
2.4.2	Communicating FSMs . . . . .	29
2.4.3	Message Sequence Chart based Models . . . . .	36
2.5	Remarks about Modeling Notations . . . . .	48
2.6	Model Simulations . . . . .	52
2.6.1	FSM simulations . . . . .	53
2.6.2	Simulating MSC-based System Models . . . . .	59
2.7	Model-based Testing . . . . .	64
2.8	Model Checking . . . . .	73
2.8.1	Property Specification . . . . .	74
2.8.2	Checking procedure . . . . .	90
2.9	The SPIN Validation Tool . . . . .	101
2.10	The SMV Validation Tool . . . . .	106
2.11	Case Study: Air Traffic Controller . . . . .	109
2.12	References . . . . .	112
2.13	Exercises . . . . .	113

<b>3</b>	<b>Communication Validation</b>	<b>117</b>
3.1	Common Incompatibilities . . . . .	120
3.1.1	Sending/receiving signals in different order . . . . .	121
3.1.2	Handling a different signal alphabet . . . . .	123
3.1.3	Mismatch in data format . . . . .	126
3.1.4	Mismatch in data rates . . . . .	128
3.2	Converter Synthesis . . . . .	130
3.2.1	Representing Native Protocols and Converters . . . . .	130
3.2.2	Basic ideas for Converter synthesis . . . . .	132
3.2.3	Various strategies for protocol conversion . . . . .	140
3.2.4	Avoiding no-progress cycles . . . . .	142
3.2.5	Speculative transmission to avoid deadlocks . . . . .	143
3.3	Changing a working design . . . . .	147
3.4	References . . . . .	149
3.5	Exercises . . . . .	149
<b>4</b>	<b>Performance Validation</b>	<b>151</b>
4.1	The Conventional Abstraction of Time . . . . .	153
4.2	Predicting Execution Time of a Program . . . . .	158
4.2.1	WCET Calculation . . . . .	161
4.2.2	Modeling of Micro-architecture . . . . .	175
4.3	Interference within a Processing Element . . . . .	186
4.3.1	Interrupts from Environment . . . . .	186
4.3.2	Contention and Preemption . . . . .	189
4.3.3	Sharing a Processor Cache . . . . .	194
4.4	System level communication analysis . . . . .	199
4.5	Designing Systems with Predictable Timing . . . . .	203
4.5.1	Scratchpad Memories . . . . .	203
4.5.2	Time-triggered Communication . . . . .	208
4.6	Emerging applications . . . . .	211
4.7	References . . . . .	212
4.8	Exercises . . . . .	213
<b>5</b>	<b>Functionality Validation</b>	<b>217</b>
5.1	Dynamic or Trace-based Checking . . . . .	220
5.1.1	Dynamic Slicing . . . . .	225
5.1.2	Fault Localization . . . . .	235
5.1.3	Directed Testing Methods . . . . .	242

5.2	Formal Verification . . . . .	246
5.2.1	Predicate Abstraction . . . . .	250
5.2.2	Software Checking via Predicate Abstraction . . . . .	259
5.2.3	Combining Formal Verification with Testing . . . . .	267
5.3	References . . . . .	271
5.4	Exercises . . . . .	272





# Acknowledgements

This book owes a lot to all my students, colleagues and co-workers. It is by working with them over the last decade, I have discovered the issues and challenges in the field of embedded systems validation. So, first and foremost, I must thank them all.

I have written this book off and on, in course of my teaching and research work at the National University of Singapore (NUS). Funding from a University Research Council project at NUS is gratefully acknowledged.

A leave from NUS in 2007 to the Indian Institute of Science (IISc) infused in me the energy to start writing the book. The calm environs of the IISc campus helped set the mood for writing this book.

The support of Elsevier staff were instrumental in ensuring that the book proceeds on schedule.

Finally, playing with my five year old son Jishnu allowed me to absorb the pressures of writing the book in the midst of various deadlines and commitments. Thanks, Jishnu!

Singapore

19 January 2009.



# Preface

This book attempts cover the issues in validation of embedded software and systems. There are lot of books on “embedded software and systems” as a web search with the appropriate search terms will reveal. So, **why this book?**

There are several ways to answer the question. The first, most direct, answer is that the current books mostly deal with the programming and/or co-design of embedded systems. Validation is often discussed almost as an after-thought. In this book, we treat validation as a first class citizen in the design process, weaving it into the design process itself.

The focus of our book is on validation, but from a embedded software and systems perspective. The methods we have covered (testing/model-checking) can also be covered from a completely general perspective, focusing only on the techniques, rather than how they fit into the system design process. But we have not done so. Even though the focus of the book is on validation methods, we clearly show how it fits into system design. As an example, we present and discuss the model checking method twice in two different ways — once at the level of system model (Chapter 2) and again at the level of system implementation (Chapter 5).

Finally, being rooted in embedded software and systems — the focus of our book is not restricted to functionality validation. We have covered at least two other aspects — debugging of performance and communication behavior. As a result, this book contains analysis methods which are rarely found in a single book — testing (informal validation), model checking (formal validation), worst-case execution time analysis (static analysis for program performance), schedulability analysis (system level performance analysis) and so on — all blended under one cover, with the goal of reliable embedded system design.

As for the chapters of the book, Chapter 1 gives a general introduction to

the issues in embedded system validation. Differences between functionality and performance validation are discussed at a general level.

Chapter 2 discusses model-level validation. It starts with a generic discussions on system structure and behavior and zooms into behavioral modeling notations such as Finite-state machines (FSMs) and Message Sequence Charts (MSCs). Simulation, testing and formal verification of these models are discussed. We discuss model-based testing, where test cases generated from the model are tried out on the system implementation. We also discuss property verification, and the well-known model checking method. The chapter wraps with a nice hands-on discussion on practical validation tools such as SPIN and SMV. Thus, this chapter corresponds to *model-level debugging*.

Chapter 3 discusses the issues in resolving communication incompatibilities between embedded system components. We discuss different strategies for resolving such incompatibilities, such as endowing the components with appropriate interfaces, and/or constructing a centralized communication protocol converter. Thus, this chapter corresponds to *communication debugging*.

Chapter 4 discusses system level performance validation. We start with software timing analysis, in particular Worst-case Execution Time (WCET) analysis. This is followed by the estimation of time spent due to different interferences in a program execution — from the external environment, or due to other executing programs on same/different processing elements. Suitable analysis methods to estimate the time due to such interferences are discussed. We then discuss mechanisms to combat execution time unpredictability via system level support. In particular, we discuss compiler controlled memories or scratchpad memories. The chapter concludes with a discussion on time predictability issues in emerging applications. Thus, this chapter corresponds to *performance debugging*.

Chapter 5 discusses functionality debugging of embedded software. We discuss both formal and informal approaches, with almost equal emphasis on testing and formal verification. The first half of the chapter involves validation methods built on testing or dynamic analysis. The second half of the chapter concentrates on formal verification, in particular, software model checking. The chapter concludes with a discussion on combining formal verification with testing. Thus, this chapter corresponds to *software debugging*.

Apart from some debugging/validation methods being common to Chapters 2 and 5, the readers may try to read the chapters independently. A senior undergraduate or graduate course on this topic may however read the chapters in sequence, that is, chapters 2, 3, 4, 5.