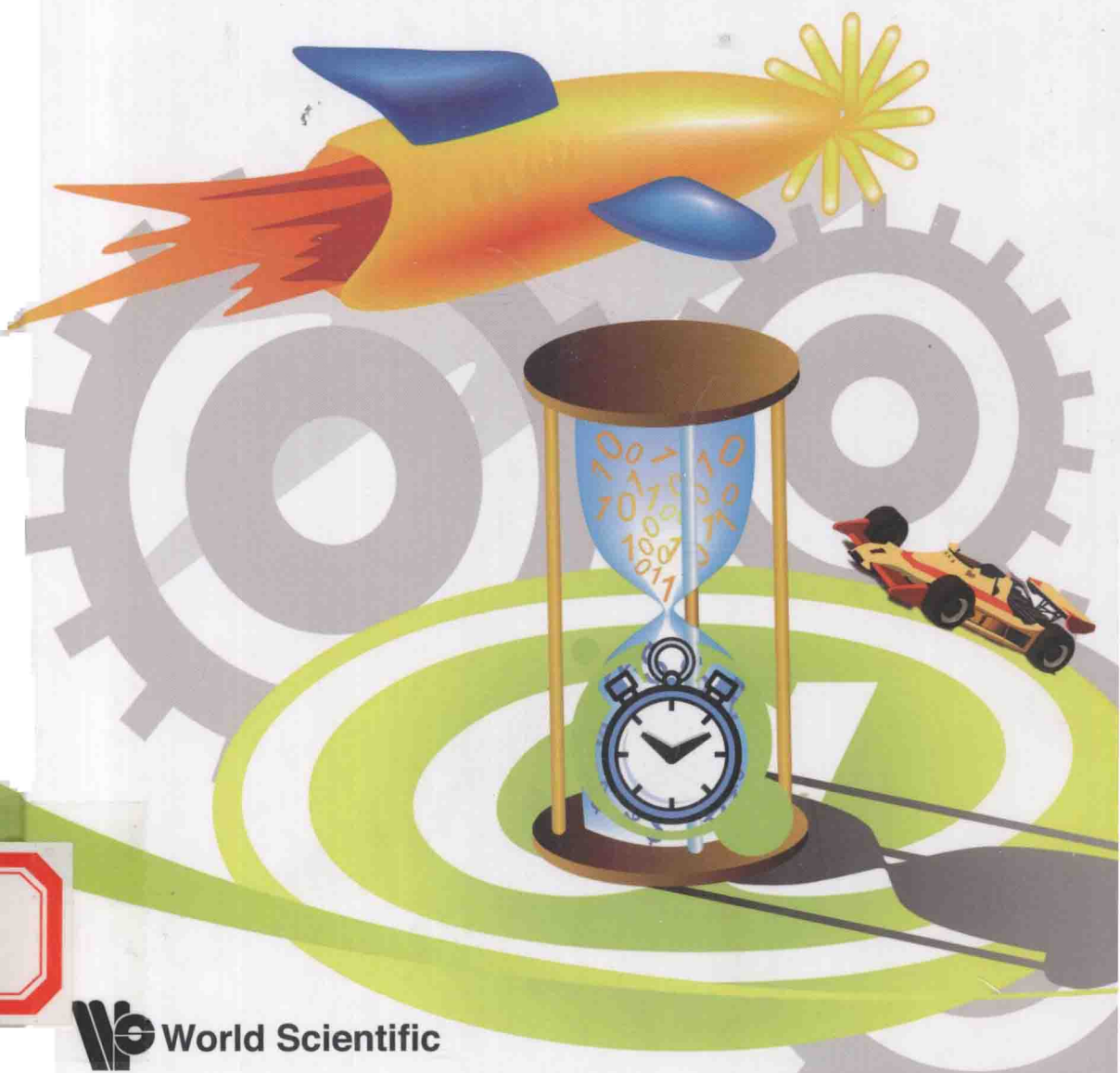


R K Shyamasundar • S Ramesh

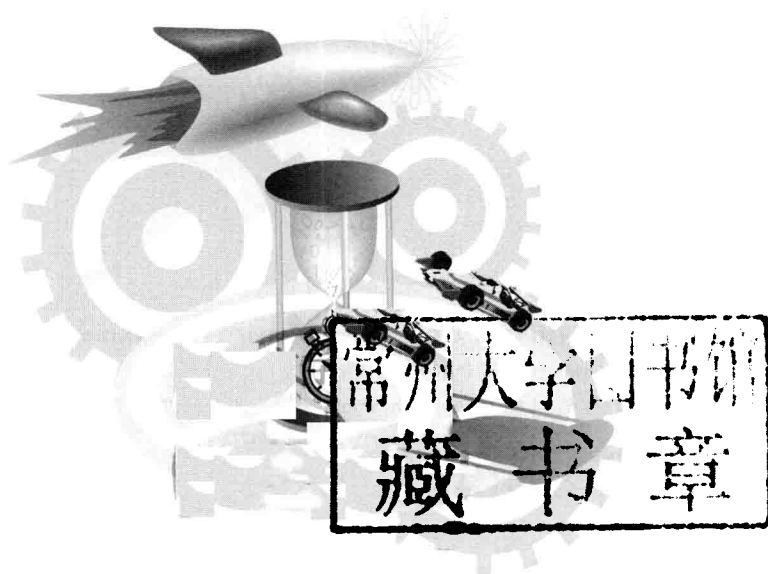
Real Time Programming

Languages, Specification and Verification



Real Time Programming

Languages, Specification and Verification



R K Shyamasundar

Tata Institute of Fundamental Research, Mumbai

S Ramesh

Indian Institute of Technology, Bombay

 **World Scientific**

NEW JERSEY • LONDON • SINGAPORE • BEIJING • SHANGHAI • HONG KONG • TAIPEI • CHENNAI

Published by

World Scientific Publishing Co. Pte. Ltd.

5 Toh Tuck Link, Singapore 596224

USA office: 27 Warren Street, Suite 401-402, Hackensack, NJ 07601

UK office: 57 Shelton Street, Covent Garden, London WC2H 9HE

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library.

REAL TIME PROGRAMMING: LANGUAGES, SPECIFICATION AND VERIFICATION

Copyright © 2010 by World Scientific Publishing Co. Pte. Ltd.

All rights reserved. This book, or parts thereof, may not be reproduced in any form or by any means, electronic or mechanical, including photocopying, recording or any information storage and retrieval system now known or to be invented, without written permission from the Publisher.

For photocopying of material in this volume, please pay a copying fee through the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, USA. In this case permission to photocopy is not required from the publisher.

ISBN-13 978-981-02-2566-7

ISBN-10 981-02-2566-0

Desk Editor: Tjan Kwang Wei

Printed by FuIsland Offset Printing (S) Pte Ltd. Singapore

Preface

The concept of process control has made *Embedded Systems* all pervasive. The applications range from home appliances (video pumps, cameras, set-top boxes, games), personal telecom and multimedia, medical therapy systems, process control systems, automobiles, avionics, tactical Control, nuclear industry etc. If one considers the processors by numbers or volume, it is of interest to note that only 1% of the total number of processors manufactured are used for general purpose computers (say desktop etc); the rest are all used in embedded systems. While general purpose processors is essentially a dedicated general purpose computational system, an embedded system is a complex system consisting of a package of hardware and software systems that can include a range of mechanical systems interfaced with the real-world to realize repetitive monitoring/actuating of the environment. Its' purpose is a dedicated function rather than general purpose computing. Hence, in embedded systems the focus is on the control logic that governs the interaction of the system with the real world. In other words, the challenges in the design of embedded systems lies in interfacing with the real-world meeting concurrent real-time constraints, and stringent safety considerations to augment component (sensor/actuator) interfaces. Thus, challenges towards the specification, design and realization of embedded systems can be summarized as follows:

1. Rigorous design and verification methodologies for embedded software. The grand challenge advocated by David Harel in this direction can be summarized as follows:
 - Devise frameworks for developing complex reactive systems providing means for describing and analyzing systems with the understanding that the structure be driven and propelled by *behaviour* of the systems. Needless to say tools play a vital role in the realization of such a dream.

2. Software Plays an important role in the design of embedded systems. A study shows that at least 60% of development time spent is spent on software coding. This has made a paradigm shift from hardware to software. This is also necessitated by the need to include late specification changes, shorter lifetime of embedded systems, and the need for the reuse of previous design functions independent of the platform. Thus, the main challenge is to invent design structures that match with the application domain. Developing a language independent platform to support such designs is another serious challenge.

In this book, we shall be focussing on the first goal.

Embedded control applications are concurrent and often real-time in nature: a controller runs concurrently with the physical system being controlled and is required to respond to the changes in the physical system state not only correctly but at the right times. For instance, a brake-by-wire subsystem of a car needs to brake the wheels of the car within a few milliseconds after detecting the pressing of the brake pedal. The development of real-time concurrent systems is many orders of magnitude more difficult than conventional data processing applications due to simultaneous evolution of concurrent components.

An emerging methodology of development of embedded control applications is *model-based development*. One of the main features of model based development paradigm is the use of models which are high level abstractions of software (and systems) and can be easily developed from requirements and are executable; the models can be automatically translated into low level code which can be mapped to target platform and integrated. Executable models help in early debugging of design leading to shorter and fast design cycle.

Synchronous programming methodology is one of the successful model based development methodologies of real-time embedded applications and hardware. This methodology was developed at around the same time by three French groups. An important feature of this methodology is a simplified and elegant abstraction of real-time: the synchrony hypothesis. According to this hypothesis, the reaction time of the controller implemented in software is zero. The validity of this assumption stems from the fact that the controllers are executed by powerful micro-controllers and processors which are quite fast compared to the slow physical system being controlled. The synchrony hypothesis greatly simplifies the design and verification of real-time systems as it reduces the number of interleaved executions of concurrent systems. Synchronous programs can also be efficiently translated

into low level code that can run on a variety of platforms. Recently these techniques have been commercialized into a number of tools that are being used in the aerospace and process control industries.

Spurred by the success of the synchronous methodology in many important industrial control applications, and due to the power and relative unawareness of this technology, we decided to devote this monograph to this methodology. In this monograph, we focus on the basic elements of synchronous methodology and include a detailed description of three synchronous languages: ESTEREL , Lustre and Argos.

Synchronous Languages are suitable for centralized, single processor and sequential applications. But many complex real-time embedded systems are often implemented over distributed platforms. Recently, the authors of the monograph have extended the synchronous methodology to such applications, developing modeling languages like Communicating Reactive Processes (CRP), Multi-clock ESTEREL and Communicating Reactive State Machines (CRSM). We discuss aspects of these formalisms and illustrate applications of the same.

Organization of the Monograph

The organization of the monograph is as follows. Chapters 1 - 4 discuss the general aspects of real time and reactive systems. Chapters 5 - 10 contain detailed descriptions of the ESTEREL language. While Section 5 gives in detail all the important constructs of ESTEREL , Chapter 6 gives a number of small case studies highlighting the features and tools of ESTEREL . Chapters 7 and 8 are concerned with advanced constructs of ESTEREL . Chapter 9 discusses a large case study in ESTEREL while the formal aspects of ESTEREL are given in Chapter 10. Chapters 11 - 12 deal with the other synchronous language, Lustre. While Section 11 introduces the features of Lustre, Section 12 discusses the modeling of Time Triggered Protocol in Lustre followed by a discussion of graphical language Argos motivated from Statecharts in Section 13. Chapter 14 discusses the verification methods used for ESTEREL and followed by observer based verification for Lustre.

While many reactive systems can be described using synchronous languages, large distributed applications demand a more flexible approach of combining both synchronous and asynchronous features. Chapter 16 introduces Communicating Reactive Processes (CRP), one of the earliest approach to combining synchrony and asynchrony. The semantic challenges of CRP is described in chapter 17 along with a formal semantics of CRP.

Chapter 18 discusses a pictorial variant of CRP, called Communicating Reactive State Machines Chapter 19 demonstrates how real time systems can be captured within the synchronous framework of ESTEREL . Multi-clock ESTEREL is a generalization the basic ESTEREL and is discussed in Chapter 20; multiclock ESTEREL permits modeling subsystems with different clocks. Chapter 21 summarizes the topics covered with a few important observations.

Dependence of the chapters

For the convenience of the reader, the dependence of the chapters are given below, where $a \rightarrow b$ denotes that Chapter b requires reading of Chapter a

$1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 6 \rightarrow 7 \rightarrow 8 \rightarrow 9$

$5 \rightarrow 6 \rightarrow 10, 5 \rightarrow 11 \rightarrow 12 \rightarrow 13$

$7 \rightarrow 12, 5 \rightarrow 14, 5 \rightarrow 15, 11 \rightarrow 16$

$5 \rightarrow 17 \rightarrow 18, 17 \rightarrow 19, 5 \rightarrow 20, 5 \rightarrow 21$

The intended audience for the monograph include advanced graduate students and researchers interested in synchronous languages and embedded software engineering; practicing engineers involved in embedded software development would also benefit from the book. The monograph would also provide useful material for part of a graduate course on embedded systems.

Acknowledgement

It is great pleasure to thank the inventors of synchronous languages ESTEREL (Gerard Berry), LUSTRE (Paul Caspi, Nicolas Halbwachs), SIGNAL (P. Le Guernic , Albert Benveniste), Statecharts (David Harel, Amir Pnueli), and ARGOS (Florence Maraninchi), with whom the authors had close collaborations and discussions. We also thank the Synchronous Programming Community (the authors have gained a lot in their participation in the yearly Synchronous Programming Workshops) with whom we have had a long association.

The major part of the work was done while the authors were with Tata Institute of Fundamental Research, Mumbai, and Indian Institute of Technology Bombay, Mumbai respectively. We gratefully acknowledge the generous support of these Institutions. Many of the joint works in the area came through the projects under Indo-French Centre for Promotion of Advanced Research (IFCPAR), New Delhi. It is pleasure to thank IFCPAR for the generous support. Professor R.K. Shyamasundar thanks IBM India Research Lab., New Delhi, with which he was affiliated during 2005-2008 and Prof. S. Ramesh thanks India Science Lab General Motors R&D Bangalore, with which he is currently affiliated, for the support and permission to use the resources towards the finalization of the material.

Thanks go to a large number students and collaborators who worked with us. Specially we thank Basant Rajan, Tata Institute of Fundamental Research who mainly worked and developed the multiclock Esterel. A special thanks goes to K Kalyanasundaram who developed and tested examples on Lustre and TTP, and Prahladavaradan Sampath, General Motors R&D Bangalore for reading the entire draft and giving very useful comments that improved the presentation of the monograph.

The authors would like to express their sincere appreciation for the patience and understanding shown by the publishers without which this monograph would not have seen the light of the day.

Contents

PART I: Real Time Systems — Background	1
1 Real Time System Characteristics	3
1.1 Real-time and Reactive Programs	4
2 Formal Program Development Methodologies	9
2.1 Requirement Specification	10
2.1.1 An Example	12
2.2 System Specifications	13
3 Characteristics of Real-Time Languages	17
3.1 Modelling Features of Real-Time Languages	19
3.2 A Look at Classes of Real-Time Languages	22
4 Programming Characteristics of Reactive Systems	25
4.1 Execution of Reactive Programs	26
4.2 Perfect Synchrony Hypothesis	26
4.3 Multiform Notion of Time	27
4.4 Logical Concurrency and Broadcast Communication	27
4.5 Determinism and Causality	28
PART II: Synchronous Languages	29
5 ESTEREL Language: Structure	31
5.1 Top Level Structure	31
5.1.1 Signals and Events	32
5.1.2 Module Instantiation	33
5.2 ESTEREL Statements	34
5.2.1 Data Handling Statements	36
5.2.2 Reactive Statements	36

5.2.3	Derived Statements	41
5.3	Illustrations of ESTEREL Program Behaviour	43
5.4	Causality Problems	45
5.5	A Historical Perspective	46
6	Program Development in ESTEREL	49
6.1	A Simulation Environment	49
6.2	Verification Environment	52
7	Programming Controllers in ESTEREL	55
7.1	Auto Controllers	55
7.1.1	A Very Simple Auto Controller	55
7.1.2	A Complex Controller	56
7.1.3	A Cruise Controller	58
7.1.4	A Train Controller	61
7.1.5	A Mine Pump Controller	63
8	Asynchronous Interaction in ESTEREL	67
9	Futurebus Arbitration Protocol: A Case Study	71
9.1	Arbitration Process	71
9.2	Abstraction of the Protocol	72
9.3	Solution in ESTEREL	74
10	Semantics of ESTEREL	79
10.1	Semantic Structure	79
10.2	Transition Rules	81
10.2.1	Rules for Signal Statement	84
10.3	Illustrative Examples	87
10.4	Discussions	89
10.5	Semantics of Esterel with exec	90
PART III: Other Synchronous Languages		95
11	Synchronous Language LUSTRE	97
11.1	An Overview of LUSTRE	97
11.2	Flows and Streams	97
11.3	Equations, Variables and Expressions	98
11.4	Program Structure	99
11.4.1	Illustrative Example	101

11.5	Arrays in LUSTRE	102
11.6	Further Examples	103
11.6.1	A Very Simple Auto Controller	103
11.6.2	A Complex Controller	103
11.6.3	A Cruise Controller	104
11.6.4	A Train Controller	106
11.6.5	A Mine Pump Controller	107
12	Modelling Time-Triggered Protocol (TTP) in LUSTRE	111
12.1	Time-Triggered Protocol	111
12.1.1	Clock Synchronization	113
12.1.2	Bus Guardian	114
12.2	Modelling TTP in LUSTRE	115
13	Synchronous Language ARGOS	123
13.1	ARGOS Constructs	123
13.2	Illustrative Example	125
13.3	Discussions	128
PART IV: Verification of Synchronous Programs		133
14	Verification of ESTEREL Programs	133
14.1	Transition System Based Verification of ESTEREL Programs	133
14.1.1	Detailed Discussion	134
14.2	ESTEREL Transition System	135
14.2.1	Abstraction and Hiding	136
14.2.2	Observation Equivalence Reduction	137
14.2.3	Context Filtering	139
14.3	Temporal Logic Based Verification	140
14.4	Observer-based Verification	141
14.5	First Order Logic Based Verification	143
15	Observer Based Verification of Simple LUSTRE Programs	145
15.1	A Simple Auto Controller	145
15.2	A Complex Controller	146
15.3	A Cruise Controller	146
15.4	A Train Controller	147
15.5	A Mine Pump Controller	148

PART V: Integration of Synchrony and Asynchrony	151
16 Communicating Reactive Processes	151
16.1 An Overview of CRP	151
16.2 Communicating Reactive Processes: Structure	153
16.2.1 Syntax of CRP	154
16.2.2 Realizing Watchdog Timers in CRP	155
16.3 Behavioural Semantics of CRP	156
16.4 An Illustrative Example: Banker Teller Machine	157
16.5 Implementation of CRP	160
17 Semantics of Communicating Reactive Processes	165
17.1 A Brief Overview of CSP	165
17.2 Translation of CSP to CRP	166
17.3 Cooperation of CRP Nodes	168
17.4 Ready-Trace Semantics of CRP	168
17.5 Ready-Trace Semantics of CSP	168
17.5.1 Semantic Definition	170
17.5.2 Semantics of Parallel Composition	170
17.5.3 Semantics of ‘send’ Action	170
17.5.4 Semantics of ‘receive’ Action	171
17.5.5 Semantics of Assignment Statement	171
17.5.6 Semantics of Sequential Composition	171
17.5.7 Semantics of Guarded Selection	171
17.6 Extracting CSP Ready-trace Semantics from CRP Semantics	172
17.6.1 Behavioural Traces of CRP Programs	172
17.7 Correctness of the Translation	174
17.8 Translation into MELJE Process Calculus	176
18 Communicating Reactive State Machines	181
18.1 CRSM Constructs	182
18.2 Semantics of CRSM	183
19 Multiclock ESTEREL	187
19.1 Need for a Multiclock Synchronous Paradigm	187
19.2 Informal Introduction	189
19.2.1 Latched Signals	191
19.2.2 Expressions	192
19.2.3 Multiclock ESTEREL Statements	193

19.2.4	Informal Development of Programs in Multiclock ESTEREL	194
19.3	Formal Semantics	197
19.3.1	Specification of Clocks	197
19.4	Embedding CRP	212
19.5	Modelling a VHDL Subset	218
19.6	Discussion	219
20	Modelling Real-Time Systems in ESTEREL	221
20.1	Interpretation of a Global Clock in terms of <code>exec</code>	222
20.2	Modelling Real-Time Requirements in ESTEREL	222
20.2.1	Deadline Specification	222
20.2.2	Periodic Activities	223
20.2.3	Guaranteed Activities	224
21	Putting it Together	231
	Bibliography	235
	Index	243

Part I: Real Time Systems: Background

Summary

In the next four sections, we shall provide (i) an overview of the general characteristics of real-time systems and reactive systems, (ii) a general discussion on formal development methodologies for real-time systems, (iii) characteristics of real-time languages and (iv) programming characteristics of reactive systems and the synchrony hypothesis.

Chapter 1

Real Time System Characteristics

Real-Time systems are designed to cater to many applications ranging from simple home appliances and laboratory instruments to complex control systems for chemical and nuclear plants, flight guidance of aircrafts and ballistic missiles. All these applications require a computational system (including both the computer and software) interacting with physical equipments like sensors and actuators. Such systems are often referred to as *embedded systems*.

An important feature of many of these systems is the ability to provide *continual* and *timely* response to unpredictable changes in the state of the environment. Hence, these systems have relatively rigid performance requirements. Further, these systems have to satisfy stringent *fail-safe* reliability requirements as failure in many of the applications will result in economic, human or ecological catastrophes. For these reasons, these systems are called *safety-critical* or *time-critical* systems.

In general, the interface between a real-time system and its environment tends to be complex, asynchronous, and distributed. This is due to the fact that the environment of the system consists of a number of physical entities that have autonomous behavior and that interact with the systems asynchronously; it is probably the complexity of the environment that necessitates computer support in the first place. Such systems can be extraordinarily hard to test. The complexity of the environment interface is one obstacle, and the fact that these programs often cannot be tested in their operational environments is another. It is not feasible to test flight-guidance software by flying with it, nor to test ballistic-missile-defense software un-

der battle conditions. In summary, some of the important characteristics of real-time systems are:

- The environment that a system interacts with, is highly nondeterministic and often consists of asynchronous distributed units; there is no way to anticipate in advance the precise order of different external events.
- High speed external events may affect the flow of control in the system easily.
- Responses to external events should be within strict bounded time limits.
- They tend to be large, complex and extraordinarily hard to test.
- In some real time applications, the mission time is long and the system, during its mission time should not only deal with ordinary situations but also must be able to recover from some exceptional situations.

In view of the above characteristics, the design of real-time systems poses serious challenges. There is a definite need for systematic methods and methodologies for designing them. In the design of quality software, high level programming languages and abstract models have a major role to play. The focus of this monograph is on some of the high level programming abstractions that have been found to be useful in designing provably correct real-time programs.

1.1 Real-time and Reactive Programs

There are many dichotomies of programs such as determinism/non determinism, synchrony/asynchrony, off-line/on-line, virtual time/ real-time, sequential/concurrent. However, depending on the way they interact with their environment, programs can be classified into the following three broad kinds[9]:

1. *Transformational Programs*: These programs compute outputs given the input; programs interact with their environment once at the beginning to get inputs and once at the end to give outputs. Compilers are examples belonging to this category.