REAL-TIME EMBEDDED SYSTEMS

DESIGN PRINCIPLES AND ENGINEERING PRACTICES

Xiaocong Fan

Real-Time Embedded Systems

Design Principles and Engineering Practices

Xiaocong Fan

ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO
Newnes is an imprint of Elsevier

Newnes is an imprint of Elsevier
The Boulevard, Langford Lane, Kidlington, Oxford OX5 1GB, UK
225 Wyman Street, Waltham, MA 02451, USA

Notices

Knowledge and best practice in this field are constantly changing. As new research and experience broaden our understanding, changes in research methods, professional practices, or medical treatment may become necessary.

Practitioners and researchers must always rely on their own experience and knowledge in evaluating and using any information, methods, compounds, or experiments described herein. In using such information or methods they should be mindful of their own safety and the safety of others, including parties for whom they have a professional responsibility.

To the fullest extent of the law, neither the Publisher nor the authors, contributors, or editors, assume any liability for any injury and/or damage to persons or property as a matter of products liability, negligence or otherwise, or from any use or operation of any methods, products, instructions, or ideas contained in the material herein.

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data
A catalog record for this book is available from the Library of Congress

ISBN: 978-0-12-801507-0

Typeset by SPi Global, India

15 16 17 18 10 9 8 7 6 5 4 3 2 1

Real-Time Embedded Systems

Preface

An embedded system is an electronic system that is designed to perform a dedicated function within a larger system. Real-time systems are those that can provide guaranteed worst-case response times to critical events, as well as acceptable average-case response times to noncritical events. When a real-time system is designed as an embedded component, it is called a real-time embedded system. Real-time embedded systems are widespread in consumer, industrial, medical, and military applications.

As more and more of our daily life depends on embedded technologies, the demand for engineers with the skill set for the development of real-time embedded software has soared in recent years. As a consequence, preparing students for the design and implementation of embedded software is becoming increasingly important. This textbook is written especially for advanced undergraduates or master-level students who are pursuing a major in software engineering, computer engineering, or a related discipline. The textbook may also benefit practicing engineers with a concentration in embedded software development.

This book takes a synergetic approach to introducing ideas and topics from real-time systems, embedded systems, and software development principles. Readers will not only gain a thorough understanding of concepts related to microprocessors, interrupts, and the cross-platform development process, and appreciate the importance of real-time modeling and scheduling, they will also be trained in good software engineering practices such as model documentation, model analysis, design patterns, and system standard conformance.

This textbook features three aspects that are essential for the development of real-time embedded software.

First, developing software for real-time embedded systems involves many activities, including specification of requirements, timing analysis, architecture design, multitasking design, and cross-platform testing and debugging. This book covers the whole process of embedded software development, with some topics fully explained and others only briefly mentioned (e.g., debugging and testing). In particular, this book presents various embedded software architectures in a systematic way, with a focus on a real-time operating system, which is the most advanced architecture adopted in large real-time embedded systems. Moreover, we have chosen to place significant emphasis on reusable design solutions. As shown in Table 0.1, this

Table 0.1 Summary of design patterns

Category	Pattern Name	Where in the Book
ISR	ISR-Pattern-min	Section 4.5.1
	ISR-Pattern-server	Section 4.5.2
	Interrupt chaining	Figure 4.7 in Section 4.5.3
	Interrupt cascading	Figure 4.9 in Section 4.5.4
	Interrupt disabling	Figure 4.11 in Section 4.5.5
	Double buffering	Figure 4.12 in Section 4.5.5
	Honor first request	Figure 12.17 in Section 12.3.2
Subclassing	Abstraction-occurrence	Figure 6.25 in Section 6.3.4
	General hierarchy	Figure 6.27 in Section 6.3.4
Software architecture	Round-robin DAS	Figure 12.10 in Section 12.2.2
	Round robin with interrupts	Figure 12.16 in Section 12.3.2
	FIFO queuing	Figure 12.20 in Section 12.4.1
	Priority queuing	Figure 12.21 in Section 12.4.2
	Serial port design pattern	Figure 14.5 in Section 14.2.2.1
Static task scheduler	Clock based	Section 15.2
	Frame based	Section 15.3
	Timing wheel	Section 22.3
Semaphore/mutex	Rendezvous synchronization pattern	Figure 18.8 in Section 18.3.1
	Multi-instance resource protection	Figure 18.19 in Section 18.4.1
Condition variable	Barrier synchronization pattern	Figure 18.24 in Section 18.5.1
	Producer-consumer pattern	Figure 18.25 in Section 18.5.2
	Read-write lock pattern	Figure 18.30 in Section 18.5.3
Message queue	Unidirectional queuing pattern	Figure 19.5 in Section 19.3.1
	Acked-unidirectional queuing pattern	Figure 19.6 in Section 19.3.2
	Bidirectional queuing pattern	Figure 19.7 in Section 19.3.3
	Client-server queuing pattern	Figure 19.10 in Section 19.3.4
Pipe	Unidirectional piping pattern	Figure 20.3 in Section 20.3
	Bidirectional piping pattern	Figure 20.3 in Section 20.3
Deadlock avoidance	Hierarchical messaging pattern	Figure 21.8 in Section 21.7.3

DAS, detect-acknowledge-service; FIFO, first in first out; ISR, interrupt service routine.

book introduces many design patterns, which represent the best practices that can be reused in a wide range of real-time embedded systems.

Second, Unified Modeling Language (UML) is a graphical language for specifying, visualizing, constructing, and documenting software systems. UML is useful in a variety of engineering problems, from single-process, embedded systems and stand-alone user applications to concurrent, distributed systems. This text features UML 2.4, the latest UML standard as of this writing. Throughout the book, UML diagrams are used for both system designs and concept illustrations. In particular, the UML real-time profile is carefully presented so that students can learn how to document their designs of real-time systems in a professional way.

Third, POSIX (for "portable operating system interface") is an open operating system interface standard that has been developed to promote interoperability and portability of applications across variants of Unix operating systems. Software systems built upon one real-time operating system can be easily ported to other POSIX-compliant operating systems. This text features POSIX.1-2008 (2013 edition). The operating system services and concepts covered in this book are fully compatible with the POSIX.1-2008 standard. The example codes provided in this book have been tested in QNX—a real-time operating system widely adopted in industry. Since QNX is POSIX compliant, the programs may also be compiled, without changing the source code, for execution on another POSIX-compliant operating system.

Briefly, this textbook consists of four parts:

- Part I is dedicated to a basic introduction to real-time embedded systems and the iterative development process. Although our emphasis is on the software aspects, complete isolation from the underlying hardware is neither feasible nor desirable. For such a reason, this part also contains two chapters on microprocessors and interrupts—fundamental topics for software engineers who wish to build embedded systems.
- Part II is dedicated to modeling techniques for real-time systems. In particular, we introduce the modeling tools covered by UML—a standard widely adopted in both academia and the software industry. Moreover, we introduce real-time UML—a profile for specifying real-time-related constraints in system models. UML diagrams are consistently used throughout the book to illustrate key concepts and design patterns.
- Part III is dedicated to the design of software architectures for real-time embedded systems. We start with generic architectures, which lead us to the most complicated architecture—a real-time operating system. The focus is then switched to multitasking and real-time scheduling—two critical issues to be addressed by any designers of real-time embedded systems.
- Part IV is dedicated to system implementation. We especially focus on those mechanisms available on any POSIX-compliant operating systems; this means that the design/implementation patterns given in this book are applicable to other POSIX-compliant operating systems as well.

The four parts together have 23 chapters. A one-semester course can use selected chapters/sections to suit the interests of the instructor and students. For instance, some microprocessor types in Chapter 3 can be skipped in order to fit the materials in one or two lecture time. If UML basic modeling concepts have been covered in a prerequisite course on software engineering principles, chapters 6, 7, and 8 can be used as self-reading assignments or simply used as a reference. Depending on the students' familiarity with basic concepts of operating systems, some topics covered in Part IV (say, message queue, pipe, and signals) can be treated differently.

To aid instructors and students in using this text, we provide a supplements package on Elsevier's companion website: http://booksite.elsevier.com/9780128015070. This package includes PowerPoint slides and source code.

In this text, I have not been able to cover every major topic concerned with real-time embedded systems. I have exercised my best judgement in deciding which topics are suitable for software engineers, which to emphasize, and which to omit. Seriously interested readers may refer to other textbooks for different perspectives.

Comments from colleagues are encouraged and welcome. Please feel free to send suggestions to Xiaocong Fan, Behrend College, Pennsylvania State University, Erie, PA 16563, USA (e-mail: xfan@psu.edu). I look forward to hearing from you about your experiences with the text.

Erie, PA, August 2014 *Xiaocong Fan*

Acknowledgments

First of all, I am grateful for the opportunity to learn from many excellent texts available on real-time systems or embedded systems, the authors including Jane W.S. Liu, David E. Simon, Rob Williams, Qing Li, and Bruce P. Douglass, to mention only a few. They may recognize their influence in some parts of the book, directly or indirectly.

I am indebted to the reviewers for numerous comments and enlightening suggestions. I would like to thank my students, who have been influential in shaping my thoughts on how best to organize and teach a course on real-time embedded systems.

The final manuscript was ably edited by SPi. A most special thanks go to Tim Pitts, Charlie Kent and Nicky Carter at Elsevier. This book would not be possible without their graceful management and great patience.

To my wonderful wife, Yan,
and our precious sons—Mutian and Aaron

Acronyms

AIC	Advanced interrupt controller
ANSI	American National Standards Institute
BSP	Board support package
CAN	Controller area network
CISC	Complex instruction set computing
COFF	Common Object File Format
COM	Serial communication port
CPU	Central processing unit
CSPR	Current program status register
DAS	Detect-acknowledge-service
DMA	Direct memory access
DSP	Digital signal processor
DTCM	Data tightly coupled memory
EDF	Earliest deadline first
EEPROM	Electrically erasable programmable read-only memory
ELF	Executable and Linking Format
EOI	End of interrupt
EPROM	Erasable programmable read-only memory
FIFO	First in first out
GPR	General-purpose register
GRM	General Resource Modeling
GRM	Graphical user interface
HLP	Highest locker protocol
HNERT	Highest-priority nonempty ready-thread list
I2C	Inter-Integrated Circuit
IC	Integrated circuit
ICE	In-circuit emulator
ICR	Interrupt command/status register
IEEE	Institute of Electrical and Electronics Engineers

IMR	Interrupt mask register
INV	Interrupt vector number
IP	Internet Protocol
IQR	Interrupt request
IRR	Interrupt request register
ISR	Interrupt service routine
LNA	Low-noise amplifier
LSb	Least significant bit
LSB	Least significant byte
MMU	Memory management unit
MOF	Meta Object Facility
MSB	Most significant byte
NVM	Nonvolatile memory
OCL	Object Constraint Language
OMG	Object Management Group
OOM	Object-oriented modeling
OOP	Object-oriented programming
OS	Operating system
PC	Personal computer
PCP	Priority ceiling protocol
PCR	Program counter register
PDF	Probability distribution function
PIC	Programmable interrupt controller
PIT	Programmable interval timer
POSIX	Portable Operating System Interface
PRF	Pulse repetition frequency
QoS	Quality of service
RAM	Random-access memory
RF	Radio frequency
RISC	Reduced instruction set computing
RMA	Rate-monotonic assignment
ROM	Read-only memory
RTOS	Real-time operating system
RT-UML	Real-time Unified Modeling Language
SFR	Special-function register
SPI	Serial peripheral interface
SRAM	Static read-access memory
TCM	Tightly coupled memory
UART	Universal asynchronous receiver-transmitter
UML	Unified Modeling Language
UTC	Coordinated Universal Time

Contents