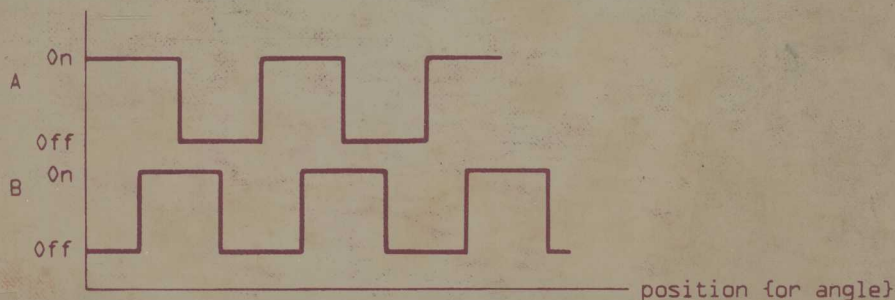
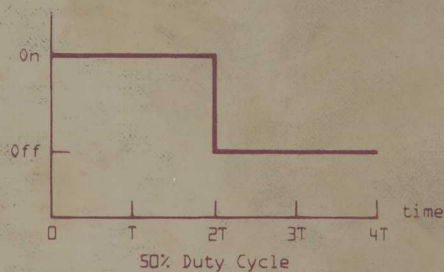
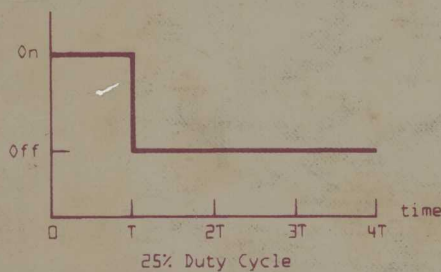


DAVID M. AUSLANDER
CHENG H. THAM

REAL-TIME SOFTWARE FOR CONTROL



PROGRAM EXAMPLES IN C

TP31
A932

9064405

REAL-TIME SOFTWARE
FOR CONTROL:
PROGRAM EXAMPLES
IN C



David M. Auslander
University of California, Berkeley

Cheng H. Tham
Sherpa Corporation



E9064405



Prentice Hall
Englewood Cliffs, New Jersey 07632

Library of Congress Cataloging-in-Publication Data

Auslander, David M.

Real time software for control : program examples in C / David M. Auslander, Cheng H. Tham.

p. cm.

Includes bibliographical references.

ISBN 0-13-762824-2

1. Real-time control--Computer programs. 2. C (Computer program language) I. Tham, Cheng H. (Cheng Haam) II. Title.

TJ217.7.A87 1989

629.8'955133--dc20

89-36478

CIP

Editorial/production supervision

and interior design: *Jacqueline A. Jeglinski*

Cover design: *Ben Santora*

Manufacturing buyer: *Denise Duggan/Mary Ann Gloriande*



© 1990 by Prentice-Hall, Inc.

A division of Simon & Schuster

Englewood Cliffs, New Jersey 07632

The publisher offers discounts on this book when ordered in bulk quantities. For more information, write:

Special Sales/College Marketing

Prentice-Hall, Inc.

College Technical and Reference Division

Englewood Cliffs, NJ 07632

All rights reserved. No part of this book may be reproduced, in any form or by any means, without permission in writing from the publisher.

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

ISBN 0-13-762824-2

Prentice-Hall International (UK) Limited, *London*

Prentice-Hall of Australia Pty. Limited, *Sydney*

Prentice-Hall Canada Inc., *Toronto*

Prentice-Hall Hispanoamericana, S.A., *Mexico*

Prentice-Hall of India Private Limited, *New Delhi*

Prentice-Hall of Japan, Inc., *Tokyo*

Simon & Schuster Asia Pte. Ltd., *Singapore*

Editora Prentice-Hall do Brasil, Ltda., *Rio de Janeiro*

REAL-TIME SOFTWARE
FOR CONTROL:
PROGRAM EXAMPLES
IN C

PREFACE

Engineers from all disciplines must be able to conceptualize, design, and prototype systems that depend on computers as operational components. The software in these computers is *real-time*, in the sense that its operation must be synchronized with events occurring in the physical system and with time.

Performance, reliability, and cost are products of the integral design of a machine and its control intelligence. Part count reductions, self-diagnostic ability, adaptation to changing environments, faster operation—these are all benefits of appropriate replacement of hardware with software. Our hypothesis is that *all* engineers need to know the fundamentals of integrated real-time software, whether they are designing, supervising design, purchasing, or using sophisticated engineering systems.

Real-time software functions in an environment in which the various system components operate asynchronously. The events associated with changes in software state, changes in state of the physical system, and time, do not repeat in a predictable way. This environment puts a premium on good design practice, since system “bugs” cannot be reproduced at will for diagnosis as they usually can in purely numerical programs. Furthermore, the asynchronous nature of the system increases the likelihood that a system will contain very low probability bugs, bugs that don’t show up in laboratory testing, but could appear in production versions of the system long after its initial release.

Our approach to real-time software emphasizes design practices that result in fewer bugs in the first place: modular programs, data hiding, mutual exclusion,

task isolation, simulation. The text uses a graduated approach, starting with strictly synchronous software (although the complete system remains asynchronous), adding interrupts, simple scheduling, and then event-driven scheduling.

All of the text material is illustrated with extensive examples, with complete code in C included. The text presents solutions in a language-independent form, with C-specific discussion of the examples following the text material in each chapter.

Motor-driven systems are used as the physical example throughout the text. Motors are ubiquitous in the engineering world, and they also can be small, inexpensive, and safe—ideal properties for use in a teaching laboratory.

The material in this text is the subject of a one-semester graduate course in the Mechanical Engineering Department, University of California at Berkeley. It is supplemented with a C-language tutorial. The course has no formal prerequisites, and is successfully completed by students from Mechanical Engineering as well as other engineering departments. In particular, students are not expected to have any prior C or real-time experience.

The course is heavily laboratory-based. In the early part of the course, example programs from the text are used as the basis for lab exercises. The students are asked to test, modify, or enhance these programs. This provides a functioning starting point for students, and helps to minimize the frustration commonly associated with development of real-time software from scratch. It also helps teach an incremental style of design—test, enhance the design, retest, etc.—which we feel allows students to build confidence in their ability to get a job done.

The text can be used even in situations in which an actual lab is not available, since the use of simulations is encouraged throughout. Simulations are carried all the way to real-time operation, with the physical system existing as a simulation in a separate task module.

The book is also intended for use as a professional self-study text. In that case, some prior study in C would be useful. Because the example code includes simulation-based programs, completion of the text does not require an extensive laboratory.

REAL-TIME SOFTWARE
FOR CONTROL:
PROGRAM EXAMPLES
IN C

CONTENTS

PREFACE

xiii

CHAPTER 1 SYNCHRONOUS PROGRAMMING

1

1.1	MOTOR SPEED CONTROL	2
1.2	THE CONTROL ALGORITHM	2
1.3	PROGRAM STRUCTURE	3
1.4	DEBUGGING: SIMULATING REAL-TIME	6
1.5	INTEGER AND FLOATING POINT: REAL-TIME CONSIDERATIONS	8
1.6	RUNNING IN REAL-TIME	11
	INTRODUCTION TO EXAMPLE PROGRAMS	13
1.7	ABOUT EXAMPLE PROGRAMS	13
1.8	THE C LANGUAGE STANDARD	13
1.9	COMPILERS	14
1.10	PROGRAMMING PRACTICES	14
1.10.1	FILE MODULES	14
1.10.2	THE ENVIRONMENT FILE—ENVIR.H	16
1.10.3	INPUT AND OUTPUT PORTS—INOUT.H	17

1.10.4	FLOATS AND DOUBLES	17	
	SYNCHRONOUS PROGRAMMING EXAMPLES	18	
1.11	INTRODUCTION	18	
1.12	MAIN MODULE—MAIN0.C	19	
1.13	CONTROL MODULE—CNTR10.C	21	
1.13.1	THE CONTROL LOOP	21	
1.13.2	PROGRAMMING NOTES	21	
1.14	SIMULATION MODULE—RTSIM0.C	22	
1.15	METRABYTE DASH-8 A/D DRIVER—DASH8.C	24	
1.15.1	PRINCIPLES OF OPERATION	24	
1.15.2	INITIALIZATION	25	
1.15.3	MORE BIT FIDDLING	25	
1.16	METRABYTE DAC-02 D/A DRIVER—DAC2.C	26	
CHAPTER 2	TIME		41
2.1	PROPORTIONAL PLUS INTEGRAL (PI) CONTROL	41	
2.2	CLOCKS	43	
2.3	CLOCK IMPLEMENTATION	43	
2.4	USING TIME IN CONTROL—PARALLEL PROCESSES	45	
2.5	ACHIEVING PARALLELISM	46	
2.6	INTERRUPT HARDWARE	47	
2.7	SIGNAL (XIGNAL): A SOFTWARE INTERRUPT CONTROLLER	49	
	THE TIME ELEMENT	50	
2.8	INTRODUCTION	50	
2.9	SIMPLE PI CONTROL	51	
2.9.1	TIMER SIMULATION	51	
2.10	REAL-TIME	52	
2.10.1	USING <i>FTIME()</i>	52	
2.10.2	USING THE XIGNAL PACKAGE	52	
2.10.3	USING TIMER INTERRUPTS DIRECTLY	53	
2.11	IN CONCLUSION	55	
CHAPTER 3	ASYNCHRONOUS SIGNAL PROCESSING		84
3.1	PULSE-WIDTH MODULATION	84	
3.2	PROGRAMMING FOR PULSE-WIDTH MODULATION	85	
3.3	PULSE-FREQUENCY MODULATION (PFM)	88	
3.4	UNIDIRECTIONAL VELOCITY MEASUREMENT	89	

3.5	SOFTWARE IMPLEMENTATION OF MULTIPLE TIMERS	91
3.6	POSITION CONTROL	93
3.7	MEASURING POSITION	95
	ASYNCHRONOUS PROGRAMMING EXAMPLES	98
3.8	INTRODUCTION	98
3.9	PROGRAM TERMINATION	99
3.10	TWO SOFTWARE TIMERS	100
	3.10.1 INITIALIZATION	102
	3.10.2 TIMER INTERRUPT SERVICE ROUTINE	102
3.11	THE PARALLEL PRINTER PORT	103
3.12	PULSE-WIDTH MODULATION	105
	3.12.1 INITIALIZATION	105
	3.12.2 PULSE-WIDTH MODULATION	105
	3.12.3 PULSE FREQUENCY	105
	3.12.4 DUTY CYCLE	106
3.13	PULSE-FREQUENCY MODULATION	106
	3.13.1 INITIALIZATION	107
	3.13.2 PULSE INPUT	107
	3.13.3 CALCULATING VELOCITY	107
	3.13.4 ADAPTIVE GROUP SIZE	108
3.14	QUADRATURE DECODING	109
	3.14.1 INITIALIZATION	110
	3.14.2 POSITION DECODING	110
	3.14.3 CALCULATING VELOCITY	111
3.15	MULTIPLE TIMERS IN SOFTWARE	112
	3.15.1 INITIALIZATION	112
	3.15.2 TIMER ALLOCATION	113
	3.15.3 CHRONOS	113
	3.15.4 MULTI-TASKING USING ONLY CHRONOS	115
3.16	CASCADE CONTROL	115

CHAPTER 4 DATA STRUCTURES

151

4.1	DATA ORGANIZATIONS	151
4.2	POINTER VARIABLES	153
4.3	POINTER ARITHMETIC	154
4.4	LINKED LISTS	155
4.5	A SIMPLE INTERPRETER FOR SETPOINT CALCULATION	157
4.6	DYNAMICS MEMORY ALLOCATION	159
4.7	STORAGE MANAGEMENT	161

4.8	CONTROL COMMAND LANGUAGE	162
	DATA STRUCTURE EXAMPLES	164
4.9	INTRODUCTION	164
4.10	REVIEW OF DATA STRUCTURES AND TYPES IN C	164
4.11	CASCADE CONTROL REVISTED	166
4.11.1	CONTROL LOOP DESCRIPTORS	166
4.11.2	INITIALIZATION	168
4.11.3	CONTROL	169
4.12	LINKED LISTS IN CASCADE CONTROL	170
4.13	COMMAND INTERPRETATION USING TABLES	172
4.13.1	INTERPRETATION	173
4.13.2	END OF TABLE	175
4.14	SIMPLE INPUT CHECKING	175

CHAPTER 5 MULTIPLE INDEPENDENT PROCESSES 193

5.1	DATA STRUCTURE FOR MULTIPLE MOTOR CONTROL	193
5.2	ALGORITHM IMPLEMENTATION: PRECISION	194
5.3	TASK SCHEDULING	198
5.4	BACKGROUND SCHEDULING	200
5.5	BACKGROUND SCHEDULING CAVEATS	202
5.6	SIMULATING REAL-TIME	204
	MULTIPLE INDEPENDENT PROCESSES EXAMPLES	206
5.7	INTRODUCTION	206
5.8	EXPLICIT BACKGROUND SCHEDULER	207
5.8.1	MAIN MODULE	207
5.8.2	CONTROL MODULE	208
5.8.2.1	INITIALIZATION	208
5.8.2.2	SCHEDULING	209
5.8.3	SIMULATION MODULE	210
5.9	GENERAL BACKGROUND SCHEDULER	211
5.9.0.1	START	212
5.9.0.2	SCHEDULING	212

CHAPTER 6 THE OPERATOR'S CONSOLE 240

6.1	CONSOLE DEVICES	241
6.2	THE SERIAL INTERFACE	243

6.3	ASYNCHRONOUS (INTERRUPT) PROCESSING	244
6.4	ECHOES, SPECIAL CHARACTERS, AND HANDSHAKING	246
6.4.1	IMPLEMENTING THE ECHO	247
6.4.2	HANDSHAKING/BUFFER FULL	248
6.4.3	IMPLEMENTATION WITH BUFFER CONTROL	248
6.5	MESSAGE ENCODING/DECODING	252
6.6	OPERATOR FUNCTIONS FOR MULTI-MOTOR CONTROL	253
6.7	FORMATTED SCREEN OPERATOR INTERFACES	256
6.7.1	LABELLED VALUES	256
6.7.2	ACTION INPUTS	257
6.7.3	SCREEN MANAGEMENT PROGRAM	257
6.7.4	SCREEN BACKGROUND	259
6.7.5	NESTED SCREENS	259
	OPERATOR CONSOLE EXAMPLES	260
6.8	INTRODUCTION	260
6.9	APPLICATION INTERFACE LAYER	261
6.9.1	BUFFERS	262
6.9.2	TERMINAL CONTROL	263
6.9.2.1	KEYBOARD INPUT	264
6.9.2.2	SCREEN OUTPUT	265
6.9.2.3	TRANSMISSION LINE ERROR	265
6.9.2.4	OUTPUT BUFFER CONTROL	265
6.10	SERIAL TERMINAL AS CONSOLE	266
6.11	VIDEO DISPLAY AND KEYBOARD AS CONSOLE	268
6.11.1	VIDEO DISPLAY OUTPUT	268
6.11.2	KEYBOARD INPUT	268
6.11.3	MISCELLANEOUS ROUTINES	269
6.12	THE DEMONSTRATION PROGRAM	269
6.12.1	INITIALIZATION	270
6.12.2	KEYBOARDS COMMANDS	270
6.12.3	PLANT ROUTINE	270
6.13	CONCLUSIONS	271
	OPERATOR CONSOLE EXAMPLES: FORMATTED SCREEN	272
6.14	INTRODUCTION	272
6.15	FILES	272
6.16	SAMPLE PROBLEM	272
6.17	SCREEN MANAGEMENT FUNCTIONS	273
6.18	MS/PCDOS INTERFACE	273

CHAPTER 7 PRIORITY SCHEDULING**314**

7.1	FOREGROUND/BACKGROUND	314
7.2	BACKGROUND PRIORITIES	315
7.3	FOREGROUND SCHEDULING	316
7.4	TASK STRUCTURE FOR CONTROL	317
7.5	TIME-BASED FIXED-PRIORITY SCHEDULING	320
7.6	TASK SUSPENSION	321
7.7	STACKS	322
7.8	ARGUMENTS AND LOCAL VARIABLES	324
7.9	PRIVATE STACKS	326
7.10	THE TASK CONTROL BLOCK	326
	CONTROL OF A THERMAL SYSTEM	328
7.11	INTRODUCTION	328
7.12	PROGRAM FILES	328
7.13	CONTROL OBJECT	329
7.14	THE CONTROLLER	330
7.15	"PURE" SIMULATION	330
7.16	REAL-TIME OPERATION	331
7.17	MUTUAL EXCLUSION	332
7.18	AUTOMATIC/MANUAL OPERATION	332
7.19	OPERATOR INTERFACE	333
7.20	DISPLAY FUNCTIONS	334
7.21	OPERATOR INPUT	337

CHAPTER 8 EVENT DRIVEN SCHEDULING**362**

8.1	TASK CONTROL	363
8.1.1	DEFINING EVENTS	363
8.1.2	TASK SUSPENSION	364
8.1.3	SIGNALLING	364
8.2	TASK STRUCTURE	365
8.3	THE MULTI-MOTOR CONTROL EXAMPLE	366
8.4	MULTI-TASKING	367
8.5	MUTUAL EXCLUSION	367
	EVENT DRIVEN SCHEDULING EXAMPLE	372
8.6	INTRODUCTION	372
8.7	INITIALIZATION	372
8.8	PRIORITIES	373
8.9	EVENT HARDWARE INTERFACE	373
8.10	GENERATING THE EXAMPLE PROGRAM	373

THE CLOTHO REAL-TIME KERNEL	374
8.11 INTRODUCTION	374
8.12 GETTING STARTED	374
8.13 SCHEDULING	377
8.13.1 FIXED SAMPLE TIME SCHEDULING	377
8.13.2 TIME-SLICE SCHEDULING	379
8.13.2.1 DEMO1.C	380
8.13.3 EVENT DRIVEN SCHEDULING	380
8.14 SYNCHRONIZATION	382
8.14.1 DISABLING INTERRUPTS	382
8.14.2 TEST AND SET	382
8.14.3 BLOCKING SEMAPHORES	383
8.15 INTER-TASK COMMUNICATION	385
8.16 SYSTEM ORGANIZATION	386
8.16.1 TASK STRUCTURE	386
8.16.2 TASK QUEUES	387
8.16.3 CONTEXT SWITCHING	388
8.16.4 TIME—CHRONOS	389
8.16.5 MEMORY ORGANIZATION	390
8.17 FUNCTION SUMMARIES	390
 CHAPTER 9 TASK ORGANIZATION AND SCHEDULING FOR A CONTROL SYSTEM	 438
9.1 THE PROCESS AND ITS MODEL	439
9.2 CONTROLLER CONFIGURATION	440
9.3 TASK STRUCTURE: NORMAL OPERATION	441
9.4 ALARMS	442
9.5 INTER-TASK COMMUNICATION	443
9.6 SIMULATED REAL-TIME FOR DEBUGGING	445
9.7 TASK SPECIFICATIONS	445
9.8 RESULTS	448
9.9 PARALLEL PROCESSING	451
CLOTHO IMPLEMENTATION OF SCHEDULING FOR A CONTROL SYSTEM	452
9.10 TIME-TELLING PROGRAM	452
9.11 THE TASKS	454
9.12 THE CONTROL PROGRAM	454
9.12.1 DATA TRANSFER	454
9.12.2 CONTROL TASK STRUCTURE	455
9.12.3 RESOURCE ALLOCATION: DATA LOGGING	455
9.12.4 ALARM	456
9.12.5 TASK SUSPENSION	456

VRTX IMPLEMENTATION OF SCHEDULING FOR A CONTROL SYSTEM		457
9.13	TIME-TELLING PROGRAM	458
9.14	THE TASKS	459
9.15	THE CONTROL PROGRAM	459
9.15.1	MAILBOXES	460
9.15.2	MESSAGE QUEUES	460
9.15.3	TASK SUSPENSION	461
BIBLIOGRAPHY		485
INDEX		487

CHAPTER 1

SYNCHRONOUS PROGRAMMING

Real-time computer systems must interact with the outside world on terms that are dictated by events taking place there. The computations that are done in response to those events must not only produce the correct results, but they must also produce those results at the right time. Unlike a real-time system, the success of a scientific or engineering computation is rarely related to when the result appears, although the user's patience and total computing expenses are related to the computation time. A further distinction in real-time computing is that the total computing environment consists of many semi-independent tasks that must be synchronized properly.

Many varieties of computers and systems qualify as "real-time." In this text, our concerns will focus on engineering systems in which there are interactions between a computer and some form of physical system. There are also often interactions with an operator. The physical system usually contains several measuring devices, which the computer must interrogate to get information, and several actuators, which receive signals from the computer to control their actions. Some systems have only one or the other of sensors or actuators, while most have both (Fig. 1.1). The computer (or computers) used can range from thumbnail size to room size (microprocessors to superminis), but the basic techniques for designing effective real-time systems are the same: careful conceptual design, systematic implementation, exhaustive validation, and thoughtful choice of software and hardware development tools. A major focus here will be on the use of high-level computing languages for implementation of real-time systems.

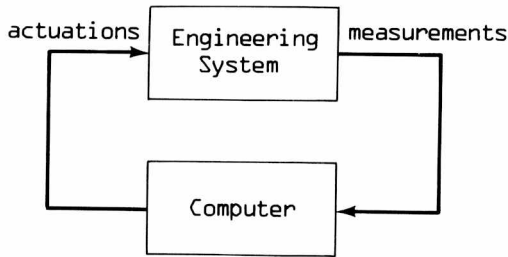


Figure 1.1

1.1 MOTOR SPEED CONTROL

We have chosen the control of electric motors as our theme. Motors are widely used and appear in so many different kinds of engineering systems that they cross virtually all disciplinary boundaries. When different methods of actuation and speed and position measurement are considered, motors also offer examples of situations that are typical of almost any real-time system. Motor systems are also easy and inexpensive to build in a laboratory, and so offer an excellent learning environment. On the other hand, the programs developed in the course of exploring the theme of motor control are generic to other control problems, and could be applied to many of them with little or no change.

A simple motor control system is shown schematically in Fig. 1.2. From the point of view of real-time system design, the simplicity of the job, even for this very simple-looking physical system, will depend on how much we demand of the computer. If the analog-to-digital (A/D) and digital-to-analog (D/A) converters can operate with little or no intervention from the computer, if the only interaction with the operator takes place at the beginning and end of an experiment, and if the algorithm chosen for computing the output signal to the power amplifier as a function of the measured motor speed depends only on the most recent measurement, then the real-time system will also be quite simple. With these restrictions, we can embark on our first example.

1.2 THE CONTROL ALGORITHM

At the heart of most real-time computation systems there are usually some key calculations. This could be a trend analysis of incoming data, spectral analysis for recognizing changes in system characteristics, generation of waveforms for system excitation, or, in this case, computation of the actuation signal on the basis of the measured motor velocity. Although these calculations are absolutely critical to proper system operation, the actual amount of program code devoted to them is usually embarrassingly small!

Control of motor speed is accomplished by increasing the voltage to the power amplifier if the speed is too low, and decreasing it if the speed is too high. A simple rule for doing this is to make the change in actuation voltage proportional to the velocity error, the difference between the actual velocity and the