

Contents

Foreword	xix
Preface to the First Edition	xxiii
Preface to the Second Edition	xxix
What's on the Website?.....	xxxii
Roadmap to Embedded Software Development.....	xxxv
Chapter 1: Embedded Software.....	1
1.1 What Makes an Embedded Application Tick?	1
1.1.1 Development Challenges	2
1.1.2 Reusable Software	3
1.1.3 Real-Time Operating System.....	4
1.1.4 File System	5
1.1.5 USB.....	5
1.1.6 Graphics	5
1.1.7 Networking	6
1.1.8 Conclusion	8
1.2 Memory in Embedded Systems	8
1.2.1 Memory.....	9
1.2.2 Implementation Challenges	10
1.2.3 When All Goes Wrong.....	11
1.2.4 When All Goes Right.....	11
1.3 Memory Architectures.....	12
1.3.1 The Options.....	12
1.3.2 Flat Single-Space Memory	13
1.3.3 Segmented Memory	13
1.3.4 Bank-Switched Memory	14
1.3.5 Multiple-Space Memory	15
1.3.6 Virtual Memory	17
1.3.7 Cache Memory.....	17
1.3.8 Memory Management Units	17
1.3.9 Conclusions.....	18
1.4 How Software Influences Hardware Design	18
1.4.1 Who Designs the Hardware?	18
1.4.2 Software Leading Hardware	19

Contents

1.4.3	Software/Hardware Trade-Offs.....	19
1.4.4	Debug Hardware	20
1.4.5	Self-Test Support	21
1.4.6	Conclusions.....	22
1.5	Migrating Your Software to a New Processor Architecture	22
1.5.1	Target Specifics	22
1.5.2	RTOS Issues.....	25
1.5.3	Processor Migration and Open Standards.....	26
1.5.4	Conclusions.....	29
1.6	Embedded Software for Transportation Applications	29
1.6.1	Introduction.....	29
1.6.2	Transportation System Characteristics.....	29
1.6.3	Programming Issues.....	30
1.6.4	Real-Time Operating System Factors	31
1.6.5	Conclusions.....	32
1.7	How to Choose a CPU for Your System on Chip Design	33
1.7.1	Design Complexity	33
1.7.2	Design Reuse	33
1.7.3	Memory Architecture and Protection.....	34
1.7.4	CPU Performance	34
1.7.5	Power Consumption.....	35
1.7.6	Costs.....	35
1.7.7	Software Issues	35
1.7.8	Multicore SoCs	35
1.7.9	Conclusions.....	36
1.8	An Introduction to USB Software	36
1.8.1	What Is USB?	36
1.8.2	A USB Peripheral.....	37
1.8.3	USB Communications	37
1.8.4	USB Software	38
1.8.5	USB and Embedded Systems	39
1.8.6	Conclusions.....	40
1.9	Toward USB 3.0	40
1.9.1	Introduction.....	41
1.9.2	Bus Architecture	41
1.9.3	Cables and Connectors.....	41
1.9.4	Packet Routing.....	41
1.9.5	Bidirectional Protocol Flow	41
1.9.6	Bulk Streaming	42
1.9.7	USB 3.0 Power Management.....	43
1.9.8	USB 3.0 Hubs	43
1.9.9	xHCI—New Host Controller Interface	43
1.9.10	Future Applications for USB	43
1.9.11	Conclusions.....	44
	Further Reading.....	45

Chapter 2: Design and Development	47
2.1 Emerging Technology for Embedded Systems Software Development	47
2.1.1 Microprocessor Device Technology	48
2.1.2 System Architecture	48
2.1.3 Design Composition	50
2.1.4 Software Content	51
2.1.5 Programming Languages	52
2.1.6 Software Team Size and Distribution	52
2.1.7 UML and Modeling	53
2.1.8 Key Technologies.....	53
2.1.9 Conclusions.....	54
2.2 Making Development Tool Choices	54
2.2.1 The Development Tool Chain	54
2.2.2 Compiler Features.....	55
2.2.3 Extensions for Embedded Systems.....	56
2.2.4 Optimizations.....	58
2.2.5 Build Tools: Key Issues Recapped.....	59
2.2.6 Debugging.....	59
2.2.7 Debug Tools: Key Issues Recapped	63
2.2.8 Standards and Development Tool Integration.....	63
2.2.9 Implications of Selections.....	64
2.2.10 Conclusions.....	65
2.3 Eclipse—Bringing Embedded Tools Together	65
2.3.1 Introduction.....	65
2.3.2 Eclipse Platform Philosophy.....	66
2.3.3 Platform	66
2.3.4 How Eclipse Gets Embedded.....	67
2.3.5 Conclusions.....	69
2.4 A Development System That Crosses RTOS Boundaries	69
2.4.1 Are Standards the Solution?.....	69
2.4.2 The Eclipse Solution.....	70
2.4.3 Eclipse Plug-Ins	70
2.4.4 Eclipse Licensing.....	71
2.4.5 Eclipse User Advantages.....	71
2.4.6 Perspectives.....	71
2.4.7 Nonembedded Plug-Ins.....	72
2.5 Embedded Software and UML.....	73
2.5.1 Why Model in UML?.....	73
2.5.2 Separating Application from Architecture	77
2.5.3 xtUML Code Generation	81
2.5.4 Conclusions.....	84
2.6 User Interface Development.....	85
2.6.1 User Interface Diversity	85
2.6.2 Implementing a User Interface	86

Contents

2.6.3	A Rationalized UI Solution.....	88
2.6.4	Conclusions.....	90
2.7	Software and Power Consumption.....	90
2.7.1	Introduction.....	90
2.7.2	Software Issues.....	92
2.7.3	Power Control in Software.....	94
2.7.4	Multicore.....	96
2.7.5	Hardware Issues.....	97
2.7.6	Virtual Programming.....	99
2.7.7	Conclusions.....	99
Chapter 3: Programming.....		101
3.1	Programming for Exotic Memories.....	101
3.1.1	Exotic Memories.....	101
3.1.2	Nonvolatile RAM.....	102
3.1.3	Shared Memory.....	104
3.1.4	Conclusions.....	105
3.2	Self-Testing in Embedded Systems.....	105
3.2.1	Memory Testing.....	105
3.2.2	Input/Output Devices.....	108
3.2.3	Multithreading Issues.....	108
3.2.4	Watchdogs.....	109
3.2.5	Self-Test Failures.....	109
3.2.6	Final Points.....	109
3.3	A Command-Line Interpreter.....	109
3.3.1	Embedded Systems Diagnostics.....	110
3.3.2	An Embedded System Comes Alive.....	111
3.3.3	A Command-Line Interpreter—Requirements.....	111
3.3.4	Designing a Command-Line Interpreter.....	111
3.3.5	A CLI Implementation.....	112
3.3.6	CLI Prototype Code.....	113
3.3.7	Conclusions.....	118
3.4	Traffic Lights: An Embedded Software Application.....	118
3.4.1	The Application.....	119
3.4.2	Hardware Configuration.....	119
3.4.3	Program Implementation.....	119
3.4.4	Main Loop.....	120
3.4.5	Interrupts.....	121
3.4.6	Time Delays.....	122
3.4.7	Lights.....	122
3.4.8	Using Global Variables.....	123
Chapter 4: C Language.....		125
4.1	C Common.....	125
4.2	Using C Function Prototypes.....	128

4.2.1	Before Prototypes	128
4.2.2	Applying Prototypes	129
4.2.3	Prototypes in Use	129
4.3	Interrupt Functions and ANSI Keywords	130
4.3.1	Interrupt Functions	130
4.3.2	ANSI C const Keyword	132
4.3.3	ANSI C Volatile Keyword	133
4.4	Bit by Bit	134
4.4.1	Bitwise Operators	135
4.4.2	Binary Constants	135
4.4.3	Bit Fields in Structures	135
4.4.4	Microprocessor Bit-Field Instructions	136
4.4.5	I/O Devices and Bit Fields	137
4.4.6	Conclusions	138
4.5	Programming Floating-Point Applications	138
4.5.1	A Test Case	138
4.5.2	Running the Test Case	139
4.5.3	Troubleshooting	140
4.5.4	Lessons Learned	140
4.6	Looking at C—A Different Perspective	141
4.6.1	Static Things	141
4.6.2	All Those Semicolons	142
4.6.3	Pointers and Pointer Arithmetic	142
4.6.4	When Being Clever Is Not Being Smart	142
4.6.5	Conclusions	143
4.7	Reducing Function Call Overhead	144
4.7.1	Compilers and Structured Code	144
4.7.2	Inline Functions	145
4.7.3	Function Calls	145
4.7.4	Parameter Passing	145
4.7.5	Local Storage	146
4.7.6	Stack Frame Generation	146
4.7.7	Return Values	148
4.7.8	Conclusions	148
4.8	Structure Layout—Become an Expert	148
4.8.1	Key Concepts	149
4.8.2	Bit Fields	153
4.8.3	Tips and Techniques	154
4.9	Memory and Programming in C	162
4.9.1	Memory	163
4.9.2	Sections	163
4.9.3	Conclusions	163
4.10	Pointers and Arrays in C and C++	164
4.10.1	Pointers and Pointer Arithmetic	164
4.10.2	Arrays and Pointers	165

Contents

4.10.3	Conclusions.....	166
4.11	Using Dynamic Memory in C and C++	166
4.11.1	C/C++ Memory Spaces	166
4.11.2	Dynamic Memory in C	168
4.11.3	Dynamic Memory in C++	169
4.11.4	Issues and Problems.....	170
4.11.5	Memory Fragmentation	171
4.11.6	Memory with an RTOS.....	173
4.11.7	Real-Time Memory Solutions.....	174
4.11.8	Conclusions.....	175
Chapter 5: C++		177
5.1	C++ in Embedded Systems—A Management Perspective	177
5.1.1	Embedded Systems Development Teams	177
5.1.2	Object-Oriented Programming	178
5.1.3	Team Management and Object-Oriented Techniques.....	178
5.1.4	C++ as an Object-Oriented Language.....	178
5.1.5	Overheads	179
5.1.6	The Way Forward.....	179
5.2	Why Convert from C to C++?	179
5.2.1	Hide Implementation Details	180
5.2.2	Reuse Class Code.....	180
5.2.3	Reuse Generic Classes	181
5.2.4	Extend Operators	181
5.2.5	Derive Classes from Base Classes	181
5.2.6	Avoid Errors Through Function Prototyping	182
5.2.7	Add Parameters Without Changing Function Calls	182
5.2.8	Using Safer, Simpler I/O.....	182
5.2.9	Improve Performance with Fast Inline Functions.....	183
5.2.10	Overload Function Names	183
5.2.11	Embedded System Support.....	184
5.2.12	Change Involves Effort	184
5.2.13	Massage C Code into C++	184
5.2.14	The Hard Part: Designing Objects	185
5.2.15	If It Ain't Broke, Don't Fix it.....	185
5.3	Clearing the Path to C++	185
5.3.1	A Strategy for Transition.....	186
5.3.2	Evolutionary Steps	186
5.3.3	Applying Reusability	186
5.3.4	Writing Clean C	187
5.3.5	C+—Nearly C++	191
5.3.6	Conclusions—The Path Ahead	194
5.4	C++ Templates—Benefits and Pitfalls.....	194
5.4.1	What Are Templates?.....	195
5.4.2	Template Instantiation.....	197

5.4.3	Problems with Templates	197
5.4.4	Multiple Template Parameters	198
5.4.5	Other Template Applications	199
5.4.6	Conclusions.....	199
5.4.7	Postscript.....	199
5.5	Exception Handling in C++	200
5.5.1	Error Handling in C	200
5.5.2	Does Not Involve Interrupts.....	200
5.5.3	C++ Exception Handling.....	201
5.5.4	Special Cases	203
5.5.5	EHS and Embedded Systems.....	206
5.5.6	Conclusions.....	206
5.6	Looking at Code Size and Performance with C++	207
5.6.1	How Efficient Is C++ Compared to C?	207
5.6.2	How C++ Affects Application Memory Requirements.....	208
5.6.3	Doing C++ Right.....	212
5.6.4	Conclusions.....	213
5.7	Write-Only Ports in C++	214
5.7.1	Encapsulating Expertise.....	214
5.7.2	Defining the Problem	214
5.7.3	A Solution in C	216
5.7.4	A First Attempt in C++	216
5.7.5	Using Overloaded Operators	217
5.7.6	Enhancing the wop Class.....	218
5.7.7	Addressing Reentrancy	219
5.7.8	Using an RTOS	221
5.7.9	Expertise Encapsulated.....	222
5.7.10	Other Possibilities	223
5.7.11	The Way Forward.....	223
5.8	Using Nonvolatile RAM with C++	223
5.8.1	Requirements of Using Nonvolatile RAM with C++	223
5.8.2	NVRAM Implementation	224
5.8.3	A C++ nvram Class	224
5.8.4	Enhancing the nvram Class.....	227
5.8.5	Conclusions.....	227
	Further Reading	228
Chapter 6: Real Time.....		229
6.1	Real-Time Systems	229
6.1.1	Implementing an RTS	230
6.1.2	A Processing Loop	230
6.1.3	Interrupts.....	230
6.1.4	Multitasking	231
6.1.5	Using an RTOS	232

Contents

6.2	Visualizing Program Models of Embedded Systems	232
6.2.1	Which Programming Model Is Best for Building Real-Time Systems?....	232
6.2.2	What Purpose Do Models Serve for a Real-Time System?	232
6.2.3	What Differences Exist Between Models, and What Gains Require What Sacrifices?	233
6.2.4	What Is the Single-Threaded Programming Model?	233
6.2.5	What Are the Advantages and Disadvantages of the Single-Threaded Programming Model?.....	233
6.2.6	Is a Polling Loop a Single-Threaded Program?.....	233
6.2.7	Is a State Machine a Single-Threaded Program?.....	234
6.2.8	What Is a Multi-Threaded System?	234
6.2.9	What Are the Advantages and Disadvantages of the Multi-Threaded Programming Model?.....	234
6.2.10	Can Multiple Threads of Execution Really Run Simultaneously on One CPU?	235
6.2.11	How Can a Multi-Threaded Environment for a Real-Time System Be Acquired?.....	235
6.3	Event Handling in Embedded Systems	236
6.3.1	What Is an Event?	236
6.3.2	Is a Signal the Same Thing as an Event?	237
6.3.3	What Events Are the Most Time Critical?	237
6.3.4	What Does the Microprocessor Do When it Detects an Exception?	237
6.3.5	Are All Exceptions the Same?	237
6.3.6	What Is a Synchronous Exception?	237
6.3.7	What Is an Asynchronous Exception?	238
6.3.8	How Do Interrupts Get Generated and Serviced?.....	238
6.3.9	What State Gets Saved by the CPU?	238
6.3.10	Is the Machine State the Same as the Thread State?.....	238
6.3.11	Should I Write Exception Handlers in Assembly Language or in C?.....	239
6.3.12	How Do I Avoid Doing Work in the Exception Handler?.....	239
6.4	Programming for Interrupts.....	239
6.4.1	Setting up Interrupts.....	240
6.4.2	Interrupt Service Routines	240
6.4.3	Interrupt Vector	241
6.4.4	Initialization	241
6.4.5	Conclusions.....	241

Chapter 7: Real-Time Operating Systems243

7.1	Debugging Techniques with an RTOS	243
7.1.1	Introduction.....	243
7.1.2	Multiprocess Concept	244
7.1.3	Execution Environment.....	245
7.1.4	Target Connection	246
7.1.5	Debugging Modes.....	247
7.1.6	RTOS-Aware Debugging Functionality	248

7.1.7	Shared Code	250
7.1.8	Task-Aware Breakpoints	250
7.1.9	Dependent Tasks	251
7.1.10	Memory Management Units	252
7.1.11	Multiple Processors.....	253
7.1.12	Conclusions.....	254
7.2	A Debugging Solution for a Custom Real-Time Operating System	254
7.2.1	Implementing Task-Aware Debugging	255
7.2.2	Task-Awareness Facilities	256
7.2.3	Conclusions.....	258
7.3	Debugging—Stack Overflows.....	258
7.3.1	Conclusions.....	259
7.4	Bring in the Pros—When to Consider a Commercial RTOS	259
7.4.1	Commercial and Custom RTOSes	260
7.4.2	Advantages of a Commercial RTOS	260
7.4.3	Commercial RTOS Downsides	260
7.4.4	Why Write a Custom RTOS?.....	261
7.4.5	Reasons Not to Create a Custom RTOS	262
7.4.6	Conclusions.....	263
7.5	On the Move.....	265
7.5.1	Migrating from One RTOS to Another	265
7.5.2	Code Migration	265
7.5.3	Wrappers.....	266
7.5.4	Drivers and More	269
7.5.5	Debugging Issues.....	269
7.5.6	Conclusions.....	269
7.6	Introduction to RTOS Driver Development	271
7.6.1	The Two Sides of a Device Driver	272
7.6.2	Data Corruption	272
7.6.3	Thread Control.....	272
7.6.4	Program Logic	273
7.6.5	Conclusions.....	274
7.7	Scheduling Algorithms and Priority Inversion	274
7.7.1	Introduction.....	274
7.7.2	Real-Time Requirements	274
7.7.3	Scheduling Algorithms	275
7.7.4	Implications for Operating Systems and Applications	275
7.7.5	Conclusions.....	277
7.8	Time Versus Priority Scheduling.....	278
7.8.1	RTOS Scheduling	278
7.8.2	Perfect World	278
7.8.3	Real World with Priority Scheduling.....	279
7.8.4	Time Domain Bounded Without Relinquish.....	279
7.8.5	Time Domain Bounded With Relinquish.....	280
7.8.6	Conclusions.....	280

Contents

7.9	An Embedded File System	281
7.9.1	Requirements of an Embedded File System	282
7.9.2	MS-DOS File System Overview	282
7.9.3	Long Filenames.....	283
7.9.4	Formatting.....	283
7.9.5	Partitioning.....	283
7.9.6	Devices.....	283
7.10	OSEK—An RTOS Standard	283
7.10.1	About OSEK	284
7.10.2	OSEK Requirements.....	284
7.10.3	OSEK Tasks	285
7.10.4	Alarms.....	286
7.10.5	Error Treatment.....	286
Chapter 8:	Networking.....	287
8.1	What's Wi-Fi?.....	287
8.1.1	Wireless Datacom	288
8.1.2	IEEE 802.11	289
8.1.3	802.11 Basics	289
8.1.4	Wi-Fi and Bluetooth.....	290
8.1.5	Where Next?	291
8.2	Who Needs a Web Server?	292
8.2.1	Introduction.....	292
8.2.2	Three Primary Capabilities	292
8.2.3	Web Servers at Work.....	294
8.2.4	Brief Summary of the Web Server's Capabilities	296
8.2.5	What Else Should You Consider?	297
8.2.6	Conclusion	297
8.3	Introduction to SNMP	298
8.3.1	Why SNMP?	298
8.3.2	The Role of Network Manager	299
8.3.3	Architectural Model	299
8.3.4	A Common Misperception.....	299
8.3.5	Application-Level Manager-Agents.....	299
8.3.6	How to Write Your MIB.....	300
8.3.7	Terminology.....	301
8.3.8	Conclusions.....	302
8.4	IPv6—The Next Generation Internet Protocol.....	302
8.4.1	Limitations of the Internet Protocol.....	303
8.4.2	Introduction to IP Version 6.....	303
8.4.3	Dual Stack Eases Transition	304
8.4.4	How IPv6 Works.....	304
8.4.5	RFC Support	308
8.5	The Basics of DHCP	309
8.5.1	A DHCP Server.....	309

8.5.2	Theory of Operation.....	310
8.5.3	RFC Support	314
8.6	NAT Explained.....	315
8.6.1	NAT Explained.....	315
8.6.2	RFC Support	317
8.6.3	Protocol Support	317
8.6.4	Application Level Gateways	318
8.6.5	The Private Network Address Assignment	318
8.7	PPP—Point-to-Point Protocol.....	318
8.7.1	Introduction.....	318
8.7.2	How PPP Works.....	319
8.7.3	PPP Details	321
8.7.4	RFC Support	324
8.8	Introduction to SSL.....	324
8.8.1	Introduction.....	325
8.8.2	How SSL Works.....	325
8.8.3	Some SSL Details	327
8.9	DHCP Debugging Tips.....	328
8.10	IP Multicasting	331
8.10.1	Initializing Multicasting.....	332
8.10.2	IGMP Protocol.....	333
8.10.3	Implementing Multicasting.....	333
8.10.4	Bringing it All Together	335
Chapter 9: Open Source, Embedded Linux, and Android		337
9.1	GNU Toolchain for Embedded Development: Build or Buy	337
9.1.1	Introduction.....	337
9.1.2	Toolchain Components	338
9.1.3	Building the Toolchain.....	340
9.1.4	Validating the Toolchain	344
9.1.5	Testing Multiple Options	347
9.1.6	Conclusions.....	349
9.2	Introduction to Linux for Embedded Systems	350
9.2.1	Introduction.....	350
9.2.2	Challenges Using Open Source	350
9.2.3	OpenEmbedded.....	352
9.2.4	Understanding Metadata	353
9.2.5	Project Workflow	355
9.2.6	Summary	355
9.3	Android Architecture and Deployment	355
9.3.1	What Is Android?	355
9.3.2	Android Architecture.....	356
9.3.3	Application Development	357
9.3.4	The Android UI.....	359

Contents

9.3.5	Extending Android Beyond Mobile	359
9.3.6	Conclusion	360
9.4	Android, MeeGo, and Embedded Linux in Vertical Markets	360
9.4.1	Introduction	360
9.4.2	How Vertical Markets Are Different	360
9.4.3	The Appeal of Android	361
9.4.4	The Promise of MeeGo	362
9.4.5	The Versatility of Embedded Linux	363
9.4.6	Conclusion	363
Chapter 10:	Multicore Embedded Systems	365
10.1	Introduction to Multicore	365
10.1.1	System Architecture	365
10.1.2	Power	366
10.1.3	Challenges	367
10.2	Multiple Cores: Multiple Operating Systems	367
10.2.1	SMP Hardware for AMP	368
10.2.2	AMP Hardware Architecture	368
10.2.3	AMP Software Architecture	369
10.2.4	The Importance of Inter-Process Communication	370
10.2.5	AMP Development Tools	370
10.2.6	Difficulties	371
10.2.7	AMP Use Cases	372
10.2.8	The Use of a Hypervisor	373
10.2.9	Conclusions	374
10.3	Selecting Multiple Operating Systems for Multiple Cores	374
10.3.1	Introduction	374
10.3.2	Types of OS	375
10.3.3	OS Selection	375
10.3.4	Multicore Systems	377
10.3.5	Conclusions	378
10.4	CPU to CPU Communication: MCAPI	378
10.4.1	Introduction	378
10.4.2	Multicore	379
10.4.3	The MCAPI	379
10.4.4	Conclusion	382
Afterword		383
Index		385