嵌入式系统开发(影印版)

Making Embedded Systems

O'REILLY®
東南大學出版社

Elecia White 著

嵌入式系统开发(影印版)

是否有兴趣开发嵌入式系统？这些系统无法容忍低下的效率，它们需要遵循严格的方式进行编程。这本易于阅读的指南通过经典的软件设计模式和嵌入式编程的全新模式来帮助你培养许多良好的开发习惯。你会学习到如何为处理器而不是操作系统来搭建系统架构，并且了解到处理硬件问题和制造工艺所需的特殊技术。

这本书的作者创建过各种各样的嵌入式系统，从城市监控和DNA扫描仪到儿童玩具。本书适合于中等和有经验的编程人员，无论你使用的是什么平台。

- 优化你的系统以减少损耗并增强性能
- 开发一套能够让你的软件在资源有限的环境中稳定运行的架构
- 探讨传感器、电机和其他I/O设备
- 少花钱多办事：减少内存使用、编码空间、处理器周期和电量损耗
- 学习如何在处理器中直接更新嵌入式代码
- 了解如何在小型处理器上实现复杂数学计算
- 当你寻求一份嵌入式系统工作时明白面试人员在说些什么

“《嵌入式系统开发》是一本适合希望进入有趣（也有利可图）的嵌入式领域的C程序员的书籍。这本书写得非常好——寓教于乐——并且讲解明晰。”

——Jack Ganssle
作家同时也是
嵌入式系统专家

Elecia White是Logical Elegance，一家在圣何塞的嵌入式系统咨询公司的创办人。她喜欢制作有趣的玩意，包括医学设备、惯性测量装置、教育玩具和一套枪声定位系统。

O'REILLY®
oreilly.com.cn

责任编辑：张烨
封面设计：Karen Montgomery，张健

O'Reilly Media, Inc.授权东南大学出版社出版

此影印版仅限于在中华人民共和国境内（但不允许在中国香港、澳门特别行政区和中国台湾地区）销售发行
This Authorized Edition for sale only in the territory of People's Republic of China (excluding Hong Kong, Macao and Taiwan)

ISBN 978-7-5641-3450-1
9 787564 134501 >
定价：58.00元

嵌入式系统开发

（影印版）

White 著

东南大学出版社

O'REILLY®

嵌入式系统开发（影印版）

Making Embedded Systems

Elecia White 著

O'REILLY®

Beijing · Cambridge · Farnham · Köln · Sebastopol · Tokyo

O'Reilly Media, Inc.授权东南大学出版社出版

东南大学出版社

图书在版编目（CIP）数据

嵌入式系统开发：英文/（美）怀特（White, E.）著. —影印本. —南京：东南大学出版社，2012.6

书名原文：Making Embedded Systems

ISBN 978-7-5641-3450-1

I. ①嵌… II. ①怀… III. ①微型计算机－系统设计－英文 IV. ①TP360.2

中国版本图书馆CIP数据核字（2012）第084977号

江苏省版权局著作权合同登记

图字：10-2012-160号

©2011 by O'Reilly Media, Inc.

Reprint of the English Edition, jointly published by O'Reilly Media, Inc. and Southeast University Press, 2012. Authorized reprint of the original English edition, 2012 O'Reilly Media, Inc., the owner of all rights to publish and sell the same.

All rights reserved including the rights of reproduction in whole or in part in any form.

英文原版由O'Reilly Media, Inc. 出版2011。

英文影印版由东南大学出版社出版2012。此影印版的出版和销售得到出版权和销售权的所有者——O'Reilly Media, Inc. 的许可。

版权所有，未得书面许可，本书的任何部分和全部不得以任何形式重制。

嵌入式系统开发（影印版）

出版发行：东南大学出版社
地　　址：南京四牌楼2号　　邮编：210096
出 版 人：江建中
网　　址：http://www.seupress.com
电子邮件：press@seupress.com
印　　刷：扬中市印刷有限公司
开　　本：787毫米×980毫米　16开本
印　　张：20.5
字　　数：401千字
版　　次：2012年6月第1版
印　　次：2012年6月第1次印刷
书　　号：ISBN 978-7-5641-3450-1
定　　价：58.00元（册）

本社图书若有印装质量问题，请直接与营销部联系。电话（传真）：025-83791830

此为试读，需要完整PDF请访问：www.ertongbook.com

Preface

I love embedded systems. The first time a motor turned because I told it to, I was hooked. I quickly moved away from pure software and into a field where I can touch the world. Just as I was leaving software, the seminal work was done on design patterns.* My team went through the book, discussing the patterns and where we'd consider using them. As I got more into embedded systems, I found compilers that couldn't handle C++ inheritance, processors with absurdly small amounts of memory in which to implement the patterns, and a whole new set of problems where design patterns didn't seem applicable. But I never forgot the idea that there are patterns to the way we do engineering. By learning to recognize the patterns, we can use the robust solutions over and over. So much of this book looks at standard patterns and offers some new ones for embedded system development. I've also filled in a number of chapters with other useful information not found in most books.

About This Book

After seeing embedded systems in medical devices, race cars, airplanes, children's toys, and gunshot location systems, I've found a lot of commonalities. There are a lot of things I wish I knew then on how to go about designing and implementing software for an embedded system. This book contains some of what I've learned. It is a book about good software design in resource-constrained environments.

It is also a book about understanding what interviewers look for when you apply for an embedded systems job. Each section ends with an interview question. These are generally not language specific; instead, they attempt to divine how you think. Good interview questions don't have a single correct answer. Instead of trying to document all the paths, the notes after each question provide hints about what an interviewer might look for in your response. You'll have to get the job (and the answers) on your own merits.

* Gamma, Erich, Richard Helm, Ralph Johnson, and John Vlissides, 1995. *Design Patterns: Elements of Reusable Object-Oriented Software*. Boston: Addison-Wesley.

One note, though: my embedded systems don't have operating systems. The software runs on the bare metal. When the software says "turn that light on," it says it to the processor without an intermediary. This isn't a book about an embedded operating system (OS). But the concepts translate to processors running OSs, so if you stick around, you may learn about the undersides of OSs too. Working without one helps you really appreciate what an OS does.

This book describes the archetypes and principles that are commonly used in creating embedded system software. I don't cover any particular platform, processor, compiler, or language, because if you get a good foundation from this book, specifics can come later.

About the Author

In the field of embedded systems, I have worked on DNA scanners, inertial measurement units for airplanes and race cars, toys for preschoolers, a gunshot location system for catching criminals, and assorted medical and consumer devices.

I have specialized in signal processing, hardware integration, complex system design, and performance. Having been through FAA and FDA certification processes, I understand the importance of producing high-quality designs and how they lead to high-quality implementations.

I've spent several years in management roles, but I enjoy hands-on engineering and the thrill of delivering excellent products. I'm happy to say that leaving management has not decreased my opportunities to provide leadership and mentoring.

Acknowledgments

This book didn't happen in a vacuum. It started with a colleague who said, "Hey, do you know a book I can give to one of my junior engineers?" From that planted seed came months of writing; I learned to really appreciate understanding (and encouraging) friends. Then there were the engineers who gave their time to look at the technical material (any remaining issues are my fault, not theirs, of course). Finally, O'Reilly provided tremendous support through the whole process.

Thanking each person as they properly deserve would take pages and pages, so I will just go through them all in one breath, in no particular order: Phillip King, Ken Brown, Jerry Ryle, Matthew Hughes, Eric Angell, Scott Fitzgerald, John Catsoulis, Robert P. J. Day, Rebecca Demarest, and Jen Costillo. These folks made a difference in the flavor of this book. There are additional thank-yous spread throughout the book, where I got help from a particular person on a particular area, so you may see these names again (or a few different ones).

Two people are left out of that list. Andy Oram is an excellent editor at O'Reilly, and I was lucky to get him. He's made a big difference to this book and to my writing.

Finally, authors always give gushing thanks to their spouses; it is a cliché. However, having written a book, I see why. Christopher White, my favorite drummer, physicist, and embedded systems engineer, thank you most of all. For everything.

Organization of This Book

I read nonfiction for amusement. I read a lot more fiction than nonfiction, but still, I like any good book. I wrote this book to be read almost as a story, from cover to cover. The information is technical (extremely so in spots), but the presentation is casual. You don't need to program along with it to get the material (though trying out the examples and applying the recommendations to your code will give you a deeper understanding).

This isn't intended to be a technical manual where you can skip into the middle and read only what you want. I mean, you can do that, but you'll miss a lot of information with the search-and-destroy method. You'll also miss the jokes, which is what I really would feel bad about. I hope that you go through the book in order. Then, when you are hip-deep in alligators and need to implement a function fast, pick up the book, flip to the right chapter, and, like a wizard, whip up a command table or fixed point implementation of variance.

Or you can skip around, reading about solutions to your crisis of the week. I understand. Sometimes you just have to solve the problem. If that is the case, I hope you find the chapter interesting enough to come back when you are done fighting that fire.

The order of chapters is:

Chapter 1, Introduction
: What is an embedded system? How is development different from traditional software?

Chapter 2, Creating a System Architecture
: How to create (and document) a system architecture.

Chapter 3, Getting Your Hands on the Hardware
: Hardware/software integration during board bring-up can be scary, but there are some ways to make it smoother.

Chapter 4, Outputs, Inputs, and Timers
: The embedded systems version of "Hello World" is making an LED blink. It can be more complex than you might expect.

Chapter 5, Managing the Flow of Activity
: This chapter describes how to set up your system, where to use interrupts (and how not to), and how to make a state machine.

Chapter 6, Communicating with Peripherals
: Different serial communication forms rule embedded systems (UART, SSP, SPI, I^2C, USB, etc.). Networking, bit-bang, and parallel buses are not to be discounted.

Chapter 7, Updating Code
: When you need to replace the program running in your processor, you have a few options ranging from internal updaters to building your own solution.

Chapter 8, Doing More with Less
: This covers methods for reducing consumption of RAM, code space, and processor cycles.

Chapter 9, Math
: Most embedded systems need to do some form of analysis. Understanding how mathematical operations and floating points work (and don't work) will make your system faster.

Chapter 10, Reducing Power Consumption
: From reducing processor cycles to system architecture suggestions, this chapter will help you if your system runs on batteries.

The information is presented in the order that I want my engineers to start thinking about these things. It may seem odd that architecture is first, considering that most people don't get to it until later in their careers. However, I want the people I work with to be thinking about how their code fits in the system long before I want them to worry about optimization.

Conventions Used in This Book

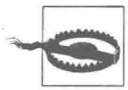
Typography

The following typographical conventions are used in this book:

Italic
: Indicates new terms, URLs, filenames, and file extensions.

`Constant width`
: Used for program listings, as well as within paragraphs to refer to program elements such as variable or function names, data types, and keywords.

This icon signifies a tip, suggestion, or general note.



This icon indicates a warning or caution.

Terminology

A *microcontroller* is a processor with onboard goodies like RAM, code space (usually flash), and various peripheral interfaces (e.g., I/O lines). Your code runs on a processor, or *central processing unit* (CPU). A *microprocessor* is a small processor, but the definition of "small" changes.

A DSP (*digital signal processor)* is a specialized form of microcontroller that focuses on signal processing, usually sampling analog signals and doing something interesting with the result. Usually a DSP is also a microcontroller, but it has special tweaks to make it perform math operations faster (in particular, multiply and add).

As I wrote this book, I wanted to use the correct terminology so you'd get used to it. However, with so many names for the piece of the system that is running your code, I didn't want to add confusion by changing the name. So, I stick with the term *processor* to represent whatever it is you're using to implement your system. Most of the material is applicable to whatever you actually have.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "*Making Embedded Systems* by Elecia White (O'Reilly). Copyright 2012 Elecia White, 978-1-449-30214-6."

If you feel your use of code examples falls outside fair use or the permission given above, feel free to contact us at *permissions@oreilly.com*.

•

Safari® Books Online

Safari Books Online is an on-demand digital library that lets you easily search over 7,500 technology and creative reference books and videos to find the answers you need quickly.

With a subscription, you can read any page and watch any video from our library online. Read books on your cell phone and mobile devices. Access new titles before they are available for print, and get exclusive access to manuscripts in development and post feedback for the authors. Copy and paste code samples, organize your favorites, download chapters, bookmark key sections, create notes, print out pages, and benefit from tons of other time-saving features.

O'Reilly Media has uploaded this book to the Safari Books Online service. To have full digital access to this book and others on similar topics from O'Reilly and other publishers, sign up for free at *http://my.safaribooksonline.com*.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information. You can access this page at:

http://shop.oreilly.com/product/0636920017776.do

To comment or ask technical questions about this book, send email to:

bookquestions@oreilly.com

For more information about our books, conferences, Resource Centers, and the O'Reilly Network, see our website at:

http://www.oreilly.com

此为试读，需要完整PDF请访问：www.ertongbook.com

Table of Contents

Preface ix

1. Introduction 1
Compilers, Languages, and Object-Oriented Programming 1
Embedded System Development 2
Debugging 2
More Challenges 4
Principles to Confront Those Challenges 5
Further Reading 7

2. Creating a System Architecture 9
Creating System Diagrams 10
The Block Diagram 10
Hierarchy of Control 13
Layered View 14
From Diagram to Architecture 16
Encapsulate Modules 16
Delegation of Tasks 17
Driver Interface: Open, Close, Read, Write, IOCTL 18
Adapter Pattern 19
Getting Started with Other Interfaces 20
Example: A Logging Interface 21
A Sandbox to Play In 27
Further Reading 32

3. Getting Your Hands on the Hardware 35
Hardware/Software Integration 35
Ideal Project Flow 36
Board Bring-Up 37
Reading a Datasheet 38
Datasheet Sections You Need When Things Go Wrong 40

Important Text for Software Developers 42
Evaluating Components Using the Datasheet 45
Your Processor Is a Language 48
Reading a Schematic 50
Having a Debugging Toolbox (and a Fire Extinguisher) 53
Keep Your Board Safe 53
Toolbox 54
Digital Multimeter 55
Oscilloscopes and Logic Analyzers 56
Testing the Hardware (and Software) 59
Building Tests 60
Flash Test Example 61
Command and Response 64
Command Pattern 67
Dealing with Errors 69
Consistent Methodology 70
Error-Handling Library 70
Debugging Timing Errors 71
Further Reading 72

4. Outputs, Inputs, and Timers 75
Toggling an Output 75
Starting with Registers 76
Set the Pin to Be an Output 77
Turn On the LED 79
Blinking the LED 80
Troubleshooting 80
Separating the Hardware from the Action 82
Board-Specific Header File 82
I/O-Handling Code 83
Main Loop 85
Facade Pattern 86
The Input in I/O 87
A Simple Interface to a Button 88
Momentary Button Press 90
Interrupt on a Button Press 90
Configuring the Interrupt 91
Debouncing Switches 91
Runtime Uncertainty 94
Dependency Injection 95
Using a Timer 96
Timer Pieces 97
Doing the Math 99

A Long Wait Between Timer Ticks 103
Using the Timer 104
Using Pulse-Width Modulation 104
Shipping the Product 106
Further Reading 108

5. Managing the Flow of Activity 109
Scheduling and Operating System Basics 109
Tasks 109
Communication Between Tasks 110
Avoiding Race Conditions 110
Priority Inversion 112
State Machines 113
State Machine Example: Stoplight Controller 114
State-Centric State Machine 115
State-Centric State Machine with Hidden Transitions 115
Event-Centric State Machine 116
State Pattern 117
Table-Driven State Machine 118
Choosing a State Machine Implementation 121
Interrupts 121
An IRQ Happens 122
Save the Context 129
Get the ISR from the Vector Table 131
Calling the ISR 133
Restore the Context 136
When to Use Interrupts 136
How Not to Use Interrupts 137
Polling 138
System Tick 138
Time-Based Events 140
A Very Small Scheduler 141
Watchdog 142
Further Reading 144

6. Communicating with Peripherals 147
The Wide Reach of Peripherals 147
External Memory 147
Buttons and Key Matrices 148
Sensors 150
Actuators 153
Displays 158
So Many Ways of Communicating 163

Serial 165
Parallel 173
Ethernet and WiFi 175
Putting Peripherals and Communication Together 176
Data Handling 176
Adding Robustness to the Communication 186
Changing Data 189
Changing Algorithms 191
Further Reading 193

7. Updating Code 197
Onboard Bootloader 198
Build Your Own Updater 199
Modifying the Resident Updater 201
Brick Loader 202
Copy Loader to RAM 203
Run the Loader 204
Copy New Code to Scratch 205
Dangerous Time: Erase and Program 205
Reset to New Code 205
Security 206
Linker Scripts 207
Summary 210

8. Doing More with Less 213
Code Space 214
Reading a Map File (Part 1) 214
Process of Elimination 217
Libraries 218
Functions and Macros 219
Constants and Strings 220
RAM 221
Remove malloc 221
Reading a Map File (Part 2) 223
Registers and Local Variables 224
Function Chains 226
Pros and Cons of Globals 228
Memory Overlays 228
Speed 229
Profiling 230
Optimizing 234
Summary 243
Further Reading 244

9. Math 247
Identifying Fast and Slow Operations 248
Taking an Average 249
Use an Existing Algorithm 252
Designing and Modifying Algorithms 255
Factor Polynomials 255
Taylor Series 256
Dividing by a Constant 258
Scaling the Input 259
Lookup Tables 260
Fake Floating-Point Numbers 267
Rational Numbers 268
Precision 269
Addition (and Subtraction) 270
Multiplication (and Division) 271
Determining the Error 272
Further Reading 276

10. Reducing Power Consumption 279
Understanding Power Consumption 280
Turn Off the Light When You Leave the Room 282
Turn Off Peripherals 282
Turn Off Unused I/O devices 283
Turn Off Processor Subsystems 283
Slowing Down to Conserve Energy 284
Putting the Processor to Sleep 285
Interrupt-Based Code Flow Model 286
A Closer Look at the Main Loop 288
Processor Watchdog 289
Avoid Frequent Wake-Ups 290
Chained Processors 290
Further Reading 290

Index 293

此为试读,需要完整PDF请访问: www.ertongbook.com