

EMBEDDED TECHNOLOGY™
SERIES



The Firmware Handbook

The Definitive Guide to Embedded Firmware Design
and Applications

Edited by

Jack Ganssle



CD-ROM included
Contains source code, an exclusive Firmware
Standards article by Jack Ganssle, and a searchable eBook version of the book



Noorwin

THE FIRMWARE HANDBOOK

THE FIRMWARE HANDBOOK

Edited by

Jack Ganssle



ELSEVIER

AMSTERDAM • BOSTON • HEIDELBERG • LONDON
NEW YORK • OXFORD • PARIS • SAN DIEGO
SAN FRANCISCO • SINGAPORE • SYDNEY • TOKYO

Newnes is an imprint of Elsevier



Newnes

Newnes is an imprint of Elsevier
200 Wheeler Road, Burlington, MA 01803, USA
Linacre House, Jordan Hill, Oxford OX2 8DP, UK

Copyright © 2004, Elsevier Inc. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the publisher.

Permissions may be sought directly from Elsevier's Science & Technology Rights Department in Oxford, UK: phone: (+44) 1865 843830, fax: (+44) 1865 853333, e-mail: permissions@elsevier.com.uk. You may also complete your request on-line via the Elsevier homepage (<http://elsevier.com>), by selecting "Customer Support" and then "Obtaining Permissions."



Recognizing the importance of preserving what has been written, Elsevier prints its books on acid-free paper whenever possible.

Library of Congress Cataloging-in-Publication Data

Ganssle, Jack
The firmware handbook / by Jack Ganssle.
p. cm.
ISBN 0-7506-7606-X
1. Computer firmware. I. Title.

QA76.765.G36 2004
004—dc22

2004040238

British Library Cataloguing-in-Publication Data

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

For information on all Newnes publications
visit our website at www.newnespress.com

04 05 06 07 08 10 9 8 7 6 5 4 3 2 1

Printed in the United States of America

Acknowledgements

I'd like to thank all of the authors whose material appears in this volume. It's been a big job, folks, but we're finally done!

Thanks also to Newnes acquisition editor Carol Lewis, and her husband Jack, who pestered me for a year before inventing the book's concept, in a form I felt that made sense for the readers and the authors. Carol started my book-writing career over a decade ago when she solicited *The Art of Programming Embedded Systems*. Thanks for sticking by me over the years.

And especially, thanks to my wife, Marybeth, for her support and encouragement, as well as an awful lot of work reformatting materials and handling the administrivia of the project.

—Jack Ganssle, February 2004, Anchorage Marina, Baltimore, MD

Contents

Preface	xv
What's on the CD-Rom?	xvi
Section I: Basic Hardware	1
Introduction	3
Chapter 1: Basic Electronics	5
<i>DC Circuits</i>	<i>5</i>
<i>AC Circuits</i>	<i>14</i>
<i>Active Devices</i>	<i>20</i>
<i>Putting it Together—a Power Supply</i>	<i>24</i>
<i>The Scope</i>	<i>27</i>
Chapter 2: Logic Circuits	33
<i>Coding</i>	<i>33</i>
<i>Combinatorial Logic</i>	<i>36</i>
<i>Sequential Logic</i>	<i>43</i>
<i>Logic Wrap-up</i>	<i>47</i>
Chapter 3: Hardware Design Tips	49
<i>Diagnostics</i>	<i>49</i>
<i>Connecting Tools</i>	<i>50</i>
<i>Other Thoughts</i>	<i>51</i>
<i>Summary</i>	<i>52</i>
Section II: Designs	53
Introduction	55
Chapter 4: Tools and Methods for Improving Code Quality	57
<i>Introduction</i>	<i>57</i>
<i>The Traditional Serial Development Cycle of an Embedded Design</i>	<i>57</i>

Contents

<i>Typical Challenges in Today's Embedded Market</i>	58
<i>Generic Methods to Improve Code Quality and Reduce the Time-to-Market</i>	59
<i>Major Time Factors for the Engineering Cycle</i>	60
<i>Where is Most Time Needed?</i>	60
<i>How to Improve Software Development Time and Code Quality</i>	60
<i>How to Reduce Hardware Development Time</i>	64
<i>Outlook and Summary</i>	65
Chapter 5: Tips to Improve Functions	69
<i>Minimize Functionality</i>	69
<i>Encapsulate</i>	70
<i>Remove Redundancies</i>	70
<i>Reduce Real-Time Code</i>	71
<i>Flow With Grace</i>	71
<i>Refactor Relentlessly</i>	72
<i>Employ Standards and Inspections</i>	72
<i>Comment Carefully</i>	73
<i>Summary</i>	75
Chapter 6: Evolutionary Development	77
<i>Introduction</i>	77
1. <i>History</i>	78
2. <i>Issues Addressed by Evo</i>	80
3. <i>How Do We Use Evo in Projects</i>	89
4. <i>Check Lists</i>	92
5. <i>Introducing Evo in New Projects</i>	94
6. <i>Testing With Evo</i>	95
7. <i>Change Requests and Problem Reports</i>	96
8. <i>Tools</i>	97
9. <i>Conclusion</i>	98
<i>Acknowledgment</i>	99
<i>References</i>	99
Chapter 7: Embedded State Machine Implementation	101
<i>State Machines</i>	101
<i>An Example</i>	102
<i>Implementation</i>	104
<i>Testing</i>	108
<i>Crank It</i>	108
<i>References</i>	109

Chapter 8: Hierarchical State Machines	111
<i>Conventional State Machine Example</i>	<i>112</i>
<i>Hierarchical State Machine Example</i>	<i>114</i>
Chapter 9: Developing Safety Critical Applications	121
<i>Introduction</i>	<i>121</i>
<i>Reliability and safety</i>	<i>121</i>
<i>Along came DO-178B</i>	<i>122</i>
<i>Overview of DO-178B</i>	<i>123</i>
<i>Failure Condition Categorization</i>	<i>123</i>
<i>System Architectural Considerations</i>	<i>125</i>
<i>System Architectural Documentation</i>	<i>126</i>
<i>DO-178B Software life cycle</i>	<i>126</i>
<i>Object Oriented Technology and Safety Critical Software Challenges</i>	<i>131</i>
<i>Iterative Process</i>	<i>132</i>
<i>Issues Facing OO Certification</i>	<i>133</i>
<i>Summary</i>	<i>136</i>
<i>References</i>	<i>136</i>
Chapter 10: Installing and Using a Version Control System	137
<i>Introduction</i>	<i>137</i>
<i>The Power and Elegance of Simplicity</i>	<i>138</i>
<i>Version Control</i>	<i>138</i>
<i>Typical Symptoms of Not (Fully) Utilizing a Version Control System</i>	<i>139</i>
<i>Simple Version Control Systems</i>	<i>139</i>
<i>Advanced Version Control Systems</i>	<i>139</i>
<i>What Files to Put Under Version Control</i>	<i>140</i>
<i>Sharing of Files and the Version Control Client</i>	<i>140</i>
<i>Integrated Development Environment Issues</i>	<i>141</i>
<i>Graphical User Interface (GUI) Issues</i>	<i>141</i>
<i>Common Source Code Control Specification</i>	<i>142</i>
<i>World Wide Web Browser Interface or Java Version Control Client</i>	<i>142</i>
<i>Bug Tracking</i>	<i>149</i>
<i>Non-Configuration Management Tools</i>	<i>150</i>
<i>Closing Comments</i>	<i>151</i>
<i>Suggested Reading, References, and Resources</i>	<i>152</i>

Section III: Math	155
Introduction	157
Chapter 11: An Introduction To Machine Calculations	159
<i>Introduction</i>	<i>159</i>
<i>Integer Arithmetic</i>	<i>159</i>
<i>Floating-Point Math</i>	<i>165</i>
<i>Fixed-Point Arithmetic</i>	<i>176</i>
<i>Conclusion</i>	<i>179</i>
<i>Bibliography</i>	<i>179</i>
Chapter 12: Floating Point Approximations	181
<i>General Trig Notes</i>	<i>182</i>
<i>Cosine and Sine</i>	<i>182</i>
<i>Higher Precision Cosines</i>	<i>189</i>
<i>Tangent</i>	<i>190</i>
<i>Higher Precision Tangents</i>	<i>195</i>
<i>Arctangent, Arcsine and Arccosine</i>	<i>196</i>
Chapter 13: Math Functions	201
<i>Gray Code</i>	<i>201</i>
<i>Integer Multiplication by a Constant</i>	<i>201</i>
<i>Computing an Exclusive Or</i>	<i>201</i>
<i>Integer Square Roots</i>	<i>202</i>
<i>Important Basic Math Operations</i>	<i>202</i>
Chapter 14: IEEE 754 Floating Point Numbers	203
<i>Special values</i>	<i>204</i>
Section IV: Real-Time	207
Introduction	209
Chapter 15: Real-Time Kernels	211
<i>Introduction</i>	<i>211</i>
<i>What is a Real-Time Kernel?</i>	<i>211</i>
<i>What is a task?</i>	<i>212</i>
<i>The Clock Tick</i>	<i>215</i>
<i>Scheduling</i>	<i>216</i>
<i>Context Switching</i>	<i>218</i>
<i>Kernel Services</i>	<i>219</i>

<i>Kernel Services, Semaphores</i>	219
<i>Kernel Services, Message Queues</i>	223
<i>Kernel Services, Memory Management</i>	225
<i>Do You Need a Kernel?</i>	225
<i>Can You Use a Kernel?</i>	226
<i>Selecting a Kernel?</i>	227
<i>Conclusion</i>	229
Chapter 16: Reentrancy	231
<i>Atomic Variables</i>	231
<i>Two More Rules</i>	233
<i>Keeping Code Reentrant</i>	234
<i>Recursion</i>	235
<i>Asynchronous Hardware/Firmware</i>	236
<i>Race Conditions</i>	237
<i>Options</i>	238
<i>Other RTOSes</i>	239
<i>Metastable States</i>	240
<i>Firmware, not Hardware</i>	242
Chapter 17: Interrupt Latency	245
<i>Taking Data</i>	248
Chapter 18: Understanding Your C Compiler:	
How to Minimize Code Size	251
<i>Modern C Compilers</i>	252
<i>Tips on Programming</i>	259
<i>Final Notes</i>	265
<i>Acknowledgements</i>	266
Chapter 19: Optimizing C and C++ Code	267
<i>Adjust Structure Sizes to Power of Two</i>	267
<i>Place Case Labels in Narrow Range</i>	267
<i>Place Frequent Case Labels First</i>	267
<i>Break Big Switch Statements into Nested Switches</i>	268
<i>Minimize Local Variables</i>	269
<i>Declare Local Variables in the Innermost Scope</i>	269
<i>Reduce the Number of Parameters</i>	269
<i>Use References for Parameter Passing and Return Value for Types Bigger than 4 Bytes</i> ..	269
<i>Don't Define a Return Value if Not Used</i>	270

Contents

<i>Consider Locality of Reference for Code and Data</i>	270
<i>Prefer int over char and short</i>	270
<i>Define Lightweight Constructors</i>	271
<i>Prefer Initialization Over Assignment</i>	272
<i>Use Constructor Initialization Lists</i>	272
<i>Do Not Declare “Just in Case” Virtual Functions</i>	273
<i>Inline 1 to 3 Line Functions</i>	273
Chapter 20: Real-Time Asserts	275
<i>Embedded Issues</i>	275
<i>Real-Time Asserts</i>	276
Section V: Errors and Changes	281
Introduction	283
Chapter 21: Implementing Downloadable Firmware With Flash Memory ..	285
<i>Introduction</i>	285
<i>The Microprogrammer</i>	286
<i>Advantages of Microprogrammers</i>	286
<i>Disadvantages of Microprogrammers</i>	287
<i>Receiving a Microprogrammer</i>	287
<i>A Basic Microprogrammer</i>	288
<i>Common Problems and Their Solutions</i>	289
<i>Hardware Alternatives</i>	295
<i>Separating Code and Data</i>	295
Chapter 22: Memory Diagnostics	299
<i>ROM Tests</i>	299
<i>RAM Tests</i>	301
Chapter 23: Nonvolatile Memory	307
<i>Supervisory Circuits</i>	307
<i>Multi-byte Writes</i>	309
<i>Testing</i>	311
<i>Conclusion</i>	312
Chapter 24: Proactive Debugging	313
<i>Stacks and Heaps</i>	313
<i>Seeding Memory</i>	315
<i>Wandering Code</i>	316

<i>Special Decoders</i>	318
<i>MMUs</i>	318
<i>Conclusion</i>	319
Chapter 25: Exception Handling in C++	321
<i>The Mountains (Landmarks of Exception Safety)</i>	322
<i>A History of this Territory</i>	323
<i>The Tar Pit</i>	324
<i>Tar!</i>	326
<i>The Royal Road</i>	326
<i>The Assignment Operator—A Special Case</i>	328
<i>In Bad Weather</i>	329
<i>Looking back</i>	332
<i>References</i>	337
Chapter 26: Building a Great Watchdog	339
<i>Internal WDTs</i>	342
<i>External WDTs</i>	343
<i>Characteristics of Great WDTs</i>	345
<i>Using an Internal WDT</i>	347
<i>An External WDT</i>	349
<i>WDTs for Multitasking</i>	350
<i>Summary and Other Thoughts</i>	352
Appendix A: ASCII Table	355
Appendix B: Byte Alignment and Ordering	357
<i>Byte Alignment Restrictions</i>	357
<i>Byte Ordering</i>	359
Index	361

Preface

Welcome to The Firmware Handbook! This book fills a critical gap in the embedded literature. There's a crying need for a compendium of ideas and concepts, a handbook, a volume that lives on engineers' desks, one they consult for solutions to problems and for insight into forgotten areas. Though the main topic is firmware, the gritty reality of the embedded world is that the code and hardware are codependent. Neither exists in isolation; in no other field of software is there such a deep coupling of the real and the virtual.

Analog designers tout the fun of their profession. Sure, it's a gas to slew an op amp. But those poor souls know nothing of the excitement of making things move, lights blink, gas flow. We embedded developers sequence motors, pump blood, actuate vehicles' brakes, control the horizontal and vertical on TV sets and eject CDs. Who can beat the fuzzy line between firmware and the real world for fun?

The target audience for the book is the practicing firmware developer, those of us who write the code that makes the technology of the 21st century work.

The book is not intended as a tutorial or introduction to writing firmware. Plenty of other volumes exist whose goal is teaching people the essentials of developing embedded systems.

Nor is it an introduction to the essentials of software engineering. Every developer should be familiar with Boehm's COCOMO software estimation model, the tenants of eXtreme Programming, Fagen and Gilb's approaches to software inspections, Humphrey's Personal Software Process, the Software Engineering Institute's Capability Maturity Model, and numerous other well-known and evolving methodologies. Meticulous software engineering is a critical part of success in building any large system, but there's little about it that's unique to the embedded world.

The reader also will find little about RTOSes, TCP/IP, DSP and other similar topics. These are hot topics, critical to an awful lot of embedded applications. Yet each subject needs an entire volume to itself, and many, many such books exist.

What's on the CD-Rom?

Included on the accompanying CD-ROM:

- A full searchable eBook version of the text in Adobe pdf format
- A directory containing the source code for all of the example programs in the book

Refer to the ReadMe file for more details on CD-ROM content.

SECTION

I

Basic Hardware

Introduction

The earliest electronic computers were analog machines, really nothing more than a collection of operational amplifiers. “Programs” as we know them did not exist; instead, users created algorithms using arrays of electronic components placed into the amps’ feedback loops. Programming literally meant rewiring the machine. Only electrical engineers understood enough of the circuitry to actually use these beasts.

In the 1940s the advent of the stored program digital computer transformed programs from circuits to bits saved on various media... though nothing that would look familiar today! A more subtle benefit was a layer of abstraction between the programmer and the machine. The digital nature of the machines transcended electrical parameters; nothing existed other than a zero or a one. Computing was opened to a huge group of people well beyond just electrical engineers. Programming became its own discipline, with its own skills, none of which required the slightest knowledge of electronics and circuits.

Except in embedded systems. Intel’s 1971 invention of the microprocessor brought computers back to their roots. Suddenly computers cost little and were small enough to build into products. Despite cheap CPUs, though, keeping the overall system cost low became the new challenge for designers *and* programmers of these new smart products.

This is the last bastion of computing where the hardware and firmware together form a unified whole. It’s often possible to reduce system costs by replacing components with smarter code, or to tune performance problems by taking the opposite tack. Our firmware works intimately with peripherals, sensors and a multitude of real-world phenomena. Though there’s been a trend to hire software-only people to build firmware, the very best developers will always be those with a breadth of experience that encompasses the very best software development practices with a working knowledge of the electronics.

Every embedded developer should be—no, *must* be—familiar with the information in this chapter. You can’t say “well, I’m a firmware engineer, so I don’t need to know what a resistor is.” You’re an *embedded* engineer, tasked with building a system that’s more than just code.

A fantastic reference that gives much more insight into every aspect of electronics is *The Art of Electronics* by Horowitz and Hill.