Embedded SOFTWARE *The Works*

CD-ROM INCLUDED containing
- Source code
- PowerPoint® slides
- Spec sheets

COLIN WALLS

Newnes

Embedded Software: The Works

Embedded Software: The Works

Colin Walls

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
30 Corporate Drive, Suite 400, Burlington, MA 01803, USA
525 B Street, Suite 1900, San Diego, California 92101-4495, USA
84 Theobald's Road, London WC1X 8RR, UK

This book is printed on acid-free paper.

Library of Congress Cataloging-in-Publication Data

Walls, Colin.
Embedded software : the works / Colin Walls.
p. cm.
Includes bibliographical references and index.
ISBN 0-7506-7954-9 (alk. paper)
1. Embedded computer systems–Programming. I. Title.
TK7895.E42W35 2005
005.1–dc22

2005014209

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

For all information on all Elsevier Newnes publications
visit our Web site at www.books.elsevier.com

Printed in the United States of America
05 06 07 08 09 9 8 7 6 5 4 3 2 1

Dedication

To Blood Donors Everywhere

Your generosity saves lives every day.

Thank you.

Contents

Foreword

What Do You Expect—Perfection?

A few words from Jack Ganssle . . .

Why are so many firmware projects so late and so bug-ridden? A lot of theories abound about software's complexity and other contributing factors, but I believe the proximate cause is that coding is not something suited to Homo Sapiens. It requires a level of accuracy that is truly super-human. And most of us are not super-human.

Cavemen did not have to get every gazelle they hunted—just enough to keep from starving. Farmers never expect the entire bag of seeds to sprout; a certain wastage is implicit and accepted. Any merchant providing a service expects to delight most, but not all, customers.

The kid who brings home straight A grades thrills his parents. Yet we get an A for being 90% correct. Perfection isn't required. Most endeavors in life succeed if we score an A, if we miss our mark by 10% or less.

Except in software. 90% correct is an utter disaster, resulting in an unusable product. 99.9% correct means we're shipping junk. One-hundred K lines of code with 99.9% accuracy suggests some 100 lurking errors. That's not good enough. Software requires near perfection, which defies the nature of intrinsically error-prone people.

Software is also highly entropic. Anyone can write a perfect 100 line-of-code system, but, as the size soars, perfection, or near perfection, requires ever-increasing investments of energy. It's as if the bits are wandering cattle trying to bust free from the corral; coding cowboys work harder and harder to avoid strays as the size of the herd grows.

So what's the solution? Is there an answer? How good does firmware have to be? How good can it be? Is our search for perfection or near-perfection an exercise in futility?

Complex systems are a new thing in this world. Many of us remember the early transistor radios which sported a half dozen active devices, max. Vacuum tube televisions, common into the 1970s, used 15 to 20 tubes, more or less equivalent to about the same number of transistors. The 1940s-era ENIAC computer required 18,000 tubes—so many that technicians wheeled shopping carts of spares through the room, constantly replacing those that burned out. Though that sounds like a lot of active elements, even the 25-year-old Z80 chip used a quarter of that many transistors, in a die smaller than just one of the hundreds of thousands of resistors in the ENIAC.

Now the Pentium IV, merely one component of a computer, has 45 million transistors. A big memory chip might require one-third of a billion. Intel predicts that later this decade their processors will have a billion transistors. The very simplest of embedded systems, like an electronic greeting card, requires thousands of active elements.

Software has grown even faster, especially in embedded applications. In 1975, 10,000 lines of assembly code was considered huge. Given the development tools of the day—paper tape, cassettes for mass storage, and crude teletypes for consoles—working on projects of this size was very difficult. Today 10,000 lines of C, representing perhaps three to five times as much assembly, is a small program. A cell phone might contain 5 million lines of C or C++, astonishing considering the device's small form factor, miniscule power requirements, and breathtakingly short development times.

Another measure of software size is memory usage. The 256 byte [that's not a typo] EPROMs of 1975 meant even a measly 4k program used 16 devices. Clearly, even small embedded systems were quite pricey. Today? 128k of Flash is nothing, even for a tiny app. The switch from 8 to 16 bit processors, and then from 16 to 32 bitters, is driven more by addressing space requirements than raw horsepower.

In the late 1970s Seagate introduced the first small Winchester hard disk, a 5Mb 10-pound beauty that cost $1500. Five MB was more disk space than almost anyone needed. Now 20Gb fits into a shirt pocket, is almost free, and fills in the blink of an eye.

So, our systems are growing rapidly in both size and complexity. Are we smart enough to build these huge applications correctly? It's hard to make even a simple application perfect; big ones will possibly never be faultless. As the software grows it inevitably becomes more intertwined; a change in one area impacts other sections, often profoundly. Sometimes this is due to poor design; often, it's a necessary effect of system growth.

The hardware, too, is certainly a long way from perfect. Even mature processors usually come with an errata sheet, one that can rival the datasheet in size. The infamous Pentium divide bug was just one of many bugs. Even today, the Pentium 3's errata sheet [renamed "specification update"] contains 83 issues. Motorola documents nearly a hundred problems in the MPC555.

What is the current state of the reliability of embedded systems? No one knows. It's an area devoid of research. Yet a lot of raw data is available, some of which suggests we're not doing well.

The Mars Pathfinder mission succeeded beyond anyone's dreams, despite a significant error that crashed the software during the lander's descent. A priority inversion problem—noticed on Earth but attributed to a glitch and ignored—caused numerous crashes. A well-designed watchdog timer recovery strategy saved the mission. This was a very instructive failure as it shows the importance of adding external hardware and/or software to deal with unanticipated software errors.

It's clear that the hardware and software are growing in raw size as well as complexity. Pathfinder's priority inversion problem was unheard of in the early days of microprocessors, when the applications just didn't need an RTOS. Today most embedded systems have some sort of operating system, and so are at peril for these sorts of problems.

What are we as developers to do, to cope with the explosion of complexity? Clearly the first step is a lifelong devotion to learning new things. And learning old things. And even re-learning old things we've forgotten. To that end we simply must curl up in the evenings with books such as this, this compendium of the old and the new, from the essentials of C programming to UML and more. Colin is an old hand, who here shares his experience with plenty of practical "how to do this" advice.

We do learn well from experience. But that's the most expensive way to accumulate knowledge. Much better is to take the wisdom of a virtuoso, to, at least for the time it takes to read this book, apprentice oneself to a master.

You'll come away enriched, with new insights and abilities to solve more complex problems with a wider array of tools.

Perfection may elude us, but wise developers will spend their entire careers engaged in the search.

Preface

How This Book Came About

I wonder how many people formulate a plan for their lives and then follow through with it. Some do, I'm sure, but for most of us, even if we have ambitions and ideas for the future, we are often buffeted by the random events around us. And that is how it was for me. I work for Accelerated Technology, a division of Mentor Graphics, which is dedicated to providing embedded software design and development tools and operating systems. I have done a variety of jobs within the company over the years, throughout a number of acquisitions and name changes. Of late, I have been doing outbound marketing—traveling around Europe and North America getting excited about our products and the technology behind them. I often describe my job as "professional enthusiasm." It is a great job, and I work with great people. I enjoy it. But it is also very taxing, with a lot of time spent on the road (or in the air) or just waiting for something to happen. I tend to be away from home quite a lot, but I have always accepted that this price is one I have to pay for such a rewarding job. And then it all changed. . . .

In mid-summer 2004, my wife was suddenly diagnosed with acute myeloid leukemia and immediately admitted to hospital. This meant that my working life was suddenly put on hold. I needed to be home to support her and care for our two teenage daughters. Her treatment—aggressive chemotherapy—progressed through the rest of the year. There have been many ups and downs, but the outcome, at the time of writing at least, is that she is in remission, having monthly blood checks to ensure that this progress continues.

It is under these circumstances that you find out who your friends are—an old cliché but true nevertheless. In my case, I also found out that I was working for the right company. My management and colleagues could not have been more supportive, and I am indebted to all of them. (I'll name names later.) Clearly I could not continue with my usual job, for a while at least. I was put under no pressure, but I gave careful thought as to what I might do that was useful, but would be compatible with my new lifestyle. I had been harboring the idea of writing this book for a while; I suggested the idea to my management, who readily agreed that it would be a worthwhile project.

Where This Book Came From

It is nearly 20 years since I last wrote a book (*Programming Dedicated Microprocessors,* Macmillan Education, 1986). That book was about embedded software, even though the term was not in common use at that time. Writing that book was a realization of an ambition, and the couple of copies that I still have are among my most treasured possessions. Writing a book is a time-consuming activity. Back when I wrote my first book, my job was relatively undemanding, and we had no children. Finding the time then was not so difficult. I have long since wanted to write another book and have had various ideas for one, but spare time has been in short supply.

Nevertheless, I have still been writing over the years—lots of technical articles, conference papers, lessons for training classes, and presentations. Then I had a thought: why not gather a whole bunch of this writing together in a book? To simplify the process, I concentrated my search for material within Accelerated Technology because the company owns the copyrights. A few other pieces did eventually get donated, but that was later.

At Microtec Research, *NewBits*, a quarterly newsletter, evolved from a simple newsletter into a technical journal, and I was a regular contributor from the early 1990s. *NewBits* continued to be published after the acquisition of Microtec by Mentor Graphics, but the journal was eventually phased out. Recently, after revitalizing the embedded software team by acquiring Accelerated Technology, we decided to revive *NewBits*, and it has gone from strength to strength. Since I collected every issue of *NewBits*, I had all of the research material at my fingertips. I quickly determined that many of my articles and those by several other writers would be appropriate for inclusion in this book. Other sources of material were white papers and another Accelerated Technology newsletter called *Nucleus Reactor*.

I have endeavored to make as few alterations as possible to the articles. Some required almost no changes; others needed a little updating or the removal of product-specific information. Each article includes an introductory paragraph that describes the origins of the article and puts it into context.

What You Will Find Here

In selecting the material for this book, my goal was to ensure that every article was relevant to current embedded software development practice and technology. Of course, many pieces have a historical perspective, but that alone was not sufficient qualification for their inclusion. Quite a few articles were rejected because they covered a technology that did not catch on or has since been superseded by something else. Maybe those topics will appear when I write *A History of Embedded Software*, which I slate for publication on the fiftieth anniversary of the start of it all (in 2020—50 years after the Intel 4004 was announced).

NewBits
Microtec Research Inc
NewBits
NEW BITS
Quarterly Embedded News from Accelerated Technology
IN THIS ISSUE
Nucleus EDGE: Eclipse for Embedded Development
Separation of Application from Architecture
RTOS Migration On the Move
Xilinx Platform Studio EDK 6.3i
Compiler Update
Win an iPod!
Accelerated Technology

In this book, you will find a selection of articles covering just about every facet of embedded software: its design, development, management, debugging procedures, licensing, and reuse.

Who This Book Is For

If you are interested in embedded software, this book includes something for you. The articles cover a wide range, and hence, it is of interest to newcomers to the field as well as old hands. If your background is in more conventional software, some pieces will give you a perspective on the "close to the hardware" stuff. If your background is in hardware design, you may gain some understanding of "the other side."

If you teach—either in a commercial or academic context—you may find some of the articles provide useful background material for your students. Some of the CD-ROM content is designed with just you in mind. See the section "What's on the CD-ROM?" following this preface.

How to Use This Book

There is no "right" way to use this book. I have attempted to sequence the articles so that reading the book from front to back would make sense. I have also done my best to categorize the articles across the chapters to help you find what might be of particular interest, with a few cross-references where I felt they might be helpful. I am frustrated by inadequate indexes in reference books, so I have tried to make the index in this book an effective means of finding what interests you.

Acknowledgments

It is difficult to write this section without it sounding like an acceptance speech for an Academy Award.

Before naming individuals, I want to make a more general acknowledgment. I have spoken to many colleagues within Accelerated Technology and in the wider world of Mentor Graphics about this book project. The response has been universal encouragement and enthusiasm. This helped.

When I started working with Elsevier, my editor was Carol Lewis, who started the project. In due course, the baton was passed to Tiffany Gasbarrini, who helped me see it through to conclusion. I enjoyed working with both of them, admired their professionalism, and appreciated their guidance.

I have always enjoyed reading the work of Jack Ganssle—his books and regular columns in *Embedded Systems Programming* magazine—who manages to convey useful technical information in a thought-provoking and, more often than not, humorous way. So, when I started planning this book, I sought his advice, which he enthusiastically provided. I was also delighted when he agreed to provide the Foreword. Thanks Jack.

I have a debt of gratitude to my management and colleagues, who have stood by me and gone that extra mile at a difficult time. Apart from (perhaps) maintaining some of my sanity, they were instrumental in making this book possible. To name a few: Neil Henderson, Robert Day, Michelle Hale, Gordon Cameron, Joakim Hedenstedt. There are many others. Thanks guys.

As I have mentioned, a great deal of the material in this book had its origins in the Microtec newsletter *NewBits*. So, I am indebted to all the folks who were involved in its

publication over the years. Lucille Woo [née Ching] was the managing editor of *NewBits*, developing it from a simple newsletter into a solid technical journal, with graphics and design work by Gianfranco Paolozzi. At Microtec, various people looked after the journal over the years, including Eugene Castillo, Melanie Gill, and Rob van Blommestein. The "revival" of *NewBits* at Accelerated Technology was led by Charity Mason, who had a hard act to follow but rose to the challenge admirably.

I am allergic to one facet of business: anything with the word "legal" involved. I accept the necessity for all the procedures, but I am grateful that people like Jodi Charter in Mentor Graphics Procurement can take care of the contract processing for me.

I was pleased by the prompt and positive response I got from David Vornholt at Xilinx and Bob Garrett at Altera when I sought solid background material on FPGA processors.

Last, and by no means least, I would like to thank Archie's proud parents, Andrea and Barry Byford, for their permission to use his photographs in the Afterword.

Contributors

I wrote about half the words in this book. The rest were contributed cooperatively or unwittingly by these individuals: Zeeshan Altaf, Fakhir Ansari, Antonio Bigazzi, Sarah Bigazzi, Paul Carrol, Lily Chang, Robert Day, Michael Eager, Michael Fay, Jack Ganssle, Bob Garrett, Kevin George, Ken Greenberg, Donald Grimes, Larry Hardin, Neil Henderson, C.C. Hung, Meador Inge, Stephen Mellor, Glen Johnson, Pravat Lall, Tammy Leino, Nick Lethaby, Steven Lewis, Alasdar Mullarney, Stephen Olsen, Doug Phillips, Uriah Pollock, James Ready, John Schneider, Robin Smith, Dan Schiro, Richard Vlamynck, David Vornholt, Fu-Hwa Wang, John Wolfe.

I would like to thank every one of them for their contributions and apologize in advance if I managed to forget attributing anyone.

It is easy to think of a book as just a bunch of words between two covers. But you will find that many articles here, as in most technical publications, include illustrations that help reinforce the message. I am happy with writing words, but I am no artist. So I am grateful for the help I received from Chase Matthews and Dan Testen, who provided the necessary artistic talent.

A Good Cause

I was delighted when my management generously agreed that all proceeds from this book could be donated to a charitable organization of my choice. Naturally, I thought about my circumstances and decided to support a group that produced tangible results. I chose the LINC Fund (Leukaemia and Intensive Chemotherapy—www.lincfund.co.uk), which is based at the center where my wife was treated.

LINC is a charitable organization dedicated to the support and well-being of patients with leukemia and related conditions. Money collected by LINC is used to purchase equipment and aid research at Cheltenham General Hospital Oncology Unit. Since the onset of these conditions can be very sudden, with patients embarking on lengthy treatment programs with little or no notice, some people find themselves in financial difficulties. The LINC Fund also provides help to such families at a time when they need it most.

On their behalf, thank you for your contribution, which I know will be spent wisely.

Contact Me

If you have comments or questions about this book, or indeed anything to do with embedded software, I would be very pleased to hear from you. Email is the best way to reach me:

`colin_walls@mentor.com`

If you wish to reach other contributors, please email me, and I will endeavor to put you in contact with them.

To see the latest information about this book—updates, errata, downloads—please visit www.EmbeddedSoftwareWorks.com.

Colin Walls