Embedded Software Development with eCos™

Anthony J. Massa

ISBN 0-13-035473-2

94999

9 790130 354739

Embedded Software Development with eCos

Anthony J. Massa

PRENTICE HALL
Professional Technical Reference
Upper Saddle River, NJ 07458
www.phptr.com
www.phptr.com/massa/

Library of Congress Cataloging-in-Publication Data

Massa, Anthony J.
Embedded software development with eCos / Anthony J. Massa
p. cm.--(Bruce Perens' Open source series)
ISBN 0-13-035473-2
1. Embedded computer systems--Programming. 2. Application software--Development. 3. Real-time data processing. I. Title. II. Series.
QA76.6 .M364317 2002
005.26--dc21

2002035507

Editorial/production supervision: *Techne Group*
Cover design director: *Jerry Votta*
Cover design: *Anthony Gemmellaro*
Art director: *Gail Cocker-Bogusz*
Interior design: *Meg Van Arsdale*
Manufacturing buyer: *Maura Zaldivar*
Editor-in-Chief: *Mark L. Taub*
Editorial assistant: *Kate Wolf*
Marketing manager: *Bryan Gambrel*
Full-service production manager: *Anne R. Garcia*

Publishing as Prentice Hall Professional Technical Reference
Upper Saddle River, New Jersey 07458

Printed in the United States of America
10 9 8 7 6 5 4 3 2 1

ISBN 0-13-035473-2

Pearson Education LTD.
Pearson Education Australia PTY, Limited
Pearson Education Singapore, Pte. Ltd.
Pearson Education North Asia Ltd.
Pearson Education Canada, Ltd.
Pearson Educación de Mexico, S.A. de C.V.
Pearson Education—Japan
Pearson Education Malaysia, Pte. Ltd.

This book is dedicated to my girls,
Katie and Deanna.
You mean the world to me.
I love you.

About Prentice Hall Professional Technical Reference

With origins reaching back to the industry's first computer science publishing program in the 1960s, Prentice Hall Professional Technical Reference (PH PTR) has developed into the leading provider of technical books in the world today. Formally launched as its own imprint in 1986, our editors now publish over 200 books annually, authored by leaders in the fields of computing, engineering, and business.

Our roots are firmly planted in the soil that gave rise to the technological revolution. Our bookshelf contains many of the industry's computing and engineering classics: *Kernighan and Ritchie's C Programming Language, Nemeth's UNIX System Administration Handbook, Horstmann's Core Java,* and *Johnson's High-Speed Digital Design.*

PH PTR acknowledges its auspicious beginnings while it looks to the future for inspiration. We continue to evolve and break new ground in publishing by providing today's professionals with tomorrow's solutions.

CONTENTS

FOREWORD

In 1997, there were over 100 commercially supported embedded operating systems, none of which had more than a minority share of the overall embedded OS market, not to mention countless thousands of others developed for specific projects (cell phones, radar arrays, networking equipment, etc.) that had no application developer base beyond that specific project. In short, the embedded operating systems market was highly fragmented, and the cost of this fragmentation was beginning to seriously limit the viability of many embedded software projects and the OEMs who funded those software projects.

While it was clear to many that the major embedded software companies needed to change their business models in radical ways, each company believed that it could somehow outlast its competition, and that it could consolidate the market through a strategy of attrition rather than a strategy of innovation. At Cygnus Solutions, we couldn't wait for 90 percent of the market to give up; moreover, we weren't sure we wanted to serve a market that was 90-percent dead. Therefore, we took up our own challenge to create an embedded operating system that could address the incredible variety of possible embedded system designs, from the very small to the highly complex, using a single source base.

In our market research, we found two primary reasons why people wrote their own RTOSes: first, they didn't want to pay per-unit royalties to a third party, and second, they didn't want to suffer the indirect cost of code that they didn't write/control/understand using up resources within their systems. The fact that writing and debugging an RTOS is expensive (in time and money), and the fact that most custom RTOSes required a complete understanding of the entire system in order to make the smallest manual changes, often resulted in systems that were both more expensive to maintain and inferior in functionality to commercial alternatives, but such were the compromises required to avoid direct and indirect per-unit costs.

The design philosophy of eCos was to augment an open-source RTOS (which meant no per-unit royalties) with source-level configuration tools that would enable embedded developers to scale their RTOS from hundreds of bytes to hundreds of kilobytes without needing to manually change a line of source code. Of course, if some code needed to be rewritten to meet some unique requirement, open-source licensing meant the option was there. However, for most cases the 200+ configuration points supported by eCos resulted in systems that were faster to build (all the hard work was coded into the configuration rules) and resulted in smaller systems than manual methods could produce (because the automated rules were more all-seeing and all-knowing than most embedded developers could afford to be).

Since releasing eCos in 1998, we have seen it develop both a healthy user base and a strong base of talented contributors. With the publication of this book, eCos reaches a new milestone: a completely independent source of technical information about eCos, and a rather complete one at that. While this book is primarily targeted at RTOS engineers, it remains accessible to both technical managers and developers who might use, but not actually maintain, an RTOS.

The scope of the book covers the very latest information about eCos, including a section on using eCos compatibility layers to provide POSIX, µItron, and even embedded Linux API compatibility with the EL/IX. Indeed, as high-end embedded system design consolidates around embedded Linux, eCos is becoming even more important for two reasons: because it provides a platform for migrating to Linux APIs without the overhead of running a full Linux-based system, and because eCos is the basis of RedBoot, the new standard ROM monitor that Red Hat supports for its embedded Linux ports.

Anthony's book is easily the most complete treatment of eCos system development. I believe it is destined to become part of every eCos developer's library.

Michael Tiemann
CTO, Red Hat, Inc.

PREFACE

Whether you're working on an existing project or moving on to a new development, eventually you're going to have to decide on what Real-Time Operating System (RTOS) to use. Numerous questions arise, including how much does it cost to get started, are there royalties associated with using the RTOS, what is the quality of the tools, is source code available, what features are available for the RTOS, and so on. In most situations, the lowest-cost solution in both upfront costs and royalties is the best solution, as long as it works. Eliminating royalties is very important for high-volume products, where every nickel counts.

There are also concerns of previous investments made, both in developer knowledge and financially, for the current solution. Anxiety can occur when considering moving existing code to a new software platform, which can be intimidating depending on the size of the project. Porting a new RTOS to your hardware platform can create more trepidation.

Decisions about whether to develop your own RTOS or use an off-the-shelf solution surface in some cases as well; especially when specific functionality is needed for a specialized hardware platform. In some cases, rolling your own RTOS might be the only solution. However, you can put your development way ahead by leveraging software that is already implemented, tested on numerous platforms and in various situations, and, most importantly, proven because it is successfully running on other shipping products. This eliminates the need for implementing functionality that is readily available.

This book focuses on one solution to these concerns: the Embedded Configurable Operating System (eCos). The open-source and royalty-free nature of eCos allows it to be downloaded, set up, and used, and here's the key: at no cost. When finished with this book, you will have a complete embedded software development environment—all the tools necessary to tackle any project.

Since eCos is open source, you, the developer, are in complete control over your embedded software destiny. Even the tools described in the eCos development system are open source, thereby allowing you to become completely self sufficient—although the eCos development community is out there available to lend help when needed.

Book Layout and Overview

Let's take a look at the layout of the book and get an overview of what is covered and where it is located. This enables you to focus on the specific aspects of eCos that you need to understand.

The layout of the book is intended to build on information covered in earlier chapters. We start with understanding the key components within eCos, then move to additional functionality offered in the system, and finally, get down to using eCos and the development environment.

For developers new to the eCos world, or embedded software altogether, it is helpful to understand the components that make up the eCos system by starting at the beginning. This gives the baseline understanding of the different features provided by eCos. You can then implement these software components in an actual system.

More experienced developers looking for an evaluation of eCos can skip to the later chapters and begin experimenting right away. The format of the development platform installation and examples allow a quick setup of the tools and immediate results. This lets you answer the question, "will eCos work for me?"

Current eCos users can fill in any holes that might be present in their eCos knowledge, by looking at some of the eCos concepts from a different point of view.

Chapter 1, *An Introduction to the eCos World*, begins with a brief introduction to eCos, which includes a background about the eCos open-source project and the company behind its start. A description of the eCos terminology is detailed as well. This terminology is used throughout the book and in the eCos development community. The beginning of the book is intended to provide developers who are unfamiliar with eCos a means to become acquainted with the eCos open-source project.

Next, we discuss the key components within the eCos system, presenting a closer look under the hood of these major software modules. The key component chapters offer an understanding about how the different software modules work independently and together to provide functionality required by the system.

Chapter 2, *The Hardware Abstraction Layer*, focuses on the software closest to the hardware that enables higher-level software modules to be unaware of the low-level functioning of the hardware.

In Chapter 3, *Exceptions and Interrupts*, we detail exceptions and interrupts and show how they are set up and handled in the eCos system. We discuss virtual vectors in Chapter 4, *Virtual Vectors*, which provide a means to share services between ROM and RAM applications.

The heart of the eCos RTOS, the kernel, is the focus in Chapter 5, *The Kernel*. The kernel supplies the scheduling functionality and the synchronization mechanisms for the software. Moving on to Chapter 6, *Threads and Synchronization Mechanisms*, we discuss the basic unit of

execution in eCos, the thread, and provide a detailed look at the various synchronization mechanisms supported by eCos.

Chapter 7, *Other eCos Architecture Components*, continues with our look at the different eCos components by focusing on timing components, asserts and tracing functionality, and the I/O control system.

Chapter 8, *Additional Functionality and Third-Party Contributions*, includes a broader look at some of the additional features available for eCos developed by the eCos project maintainers and third-party contributors. These include networking support, ROM monitors, file systems, PCI support, USB support, and the GoAhead WebServer.

In Chapter 9, *The RedBoot ROM Monitor*, we focus on the RedBoot ROM monitor. This standalone program is designed for embedded systems to provide a debugging and bootstrap environment. RedBoot is an eCos-based application and uses the eCos Hardware Abstraction Layer (HAL) for its foundation.

We begin our hands-on experience in Chapter 10, *The Host Development Platform*, with the installation of the host development tools. We discuss the Cygwin native tools, the GNU cross-development tools, and the eCos development kit. We also cover the installation of a Concurrent Versions System (CVS) client, WinCVS, to enable access to the online eCos source code repository. This gives you the ability to take advantage of any bug fixes or extended functionality contributed to the eCos source code.

In Chapter 11, *The eCos Toolset*, we delve into the eCos toolset with a detailed look at how the tools operate on the eCos source code, and the layout of the tools. Also included are some other open-source tools to round out and complete our open-source embedded development system. This prepares us for the next step, putting the tools to work to build our application.

Chapter 12, *An Example Application Using eCos*, lets you put your knowledge to work. The chapter starts with an overview of the eCos build process, followed by a build of the RedBoot ROM monitor. We then do a build of the eCos real-time operating system, and, finally, put it all together by building an application. This chapter provides a baseline on which you can then add additional components to assemble a system to meet your embedded software requirements.

Finally, Chapter 13, *Porting eCos*, closes with a look at porting eCos onto another hardware platform. This is key to getting your application running on your new target hardware platform, which is typically the main goal in embedded software development.

Development System and Examples

As mentioned previously, in Chapter 10 we go through the process of setting up an eCos development system. This development system includes the native Cygwin tools for Windows, the GNU cross-development tools (binutils, compiler, and debugger), the eCos configuration and management tools, a CVS client, and a lint program.

This system enables you to configure and build the eCos library, which is then linked with an application to run the eCos RTOS. The RedBoot ROM monitor is also built using this system.

After following the steps in Chapter 10, a complete open-source embedded software development environment is configured.

We go through examples of building RedBoot, the eCos library, and an application in Chapter 12. Rather than requiring a specific development board to run the examples, a second PC is needed for the target platform. This is a better approach to becoming familiar with the development tools since it's typically pretty easy to find a spare PC lying around.

Although GNU cross-development tools binary files are included on the CD-ROM for the Intel x86 and PowerPC processor architectures, the instructions for configuring and building the GNU cross-development tools for other processor architectures are provided in Appendix D, *Building the GNU Cross-Development Tools*.

The embedded software development tools are installed, and examples are built, on a Windows development system. However, the necessary files to get an embedded software development system up and running on a Linux system are included on the CD-ROM. Since the eCos configuration tools are able to run on both Linux and Windows, the procedure for building and running the examples applies to both host operating systems.

The CD-ROM accompanying this book contains the files needed to set up the complete embedded software development system for eCos as detailed in Chapter 10. The examples in Chapters 12 and 13 are also contained on the CD-ROM under the `examples` directory.

A web site is available to download the source code and updates. The site is located online at:

www.phptr.com/massa/

If you find any errors that need correction, feel free to contact me and I will update the source code accordingly.

Some Notes about the Book

All example code in this book is in C or assembly language. eCos also uses the Component Definition Language (CDL), an extension of the existing Tool Command Language (Tcl) scripting language. We cover this in Chapter 11.

In the text where a 32-bit hexadecimal value is shown, the most significant 16 bits and least significant 16 bits are separated by an underscore ("_") (e.g., 0xABCD_EF12) for readability.

Sidebars are also included, which are used to point out important or additional information. The sidebars look like the following:

NOTE Example of a sidebar.

This book includes several Uniform Resource Locator (URL) links showing where additional information can be obtained on the Internet. The most up-to-date links are included in the text; however, we all know that links can have a finite existence.

Item lists throughout the book detail the eCos kernel API functions. These lists contain a field named "Context." This field shows the context from where the specified function can be

called. The different contexts are Initialization, Thread, ISR, and DSR. "Any" is used to designate that the function can be called from any of the contexts.

A Brief Take on Open Source Development

There have always been "agnostics" in the debate of open source versus closed source—it sometimes comes down to whether the funding is available to purchase software tools and support. Each has its benefits and shortcomings, so let's take a brief look at some of the pros and cons developers have found when working on both sides of the open-source and closed-source fence. What it ultimately comes down to is, does the product work, and can the schedule and cost budget be met using this solution?

Open source can be a confusing term. In an article written by Daniel Benenstein,[1] he states that open source is free software in the sense of freedom of knowledge exchange. Much more reliance is placed on the individual developer to find problems, correct the problems, and make the solution available to others in the open-source community. This iterative process is the method that creates a more robust and bug-free code base.

Presumably, because the draw of talent is worldwide, the best and brightest developers are working to make the product better. With closed source, a development support team is assembled to work on fixing problems for code they probably did not create. The team assembled might be very talented; however, the pool of talent from which to draw is very small compared to a worldwide talent pool.

Often times, closed-source, or proprietary, software and open-source software work hand in hand to complement each other. Many proprietary products are derived from open-source projects. Proprietary software must be innovative and differentiate itself from the open-source alternative; otherwise, why would people pay for something that they can get for free? Looking at things from this point of view, open source pushes proprietary software to the extreme of innovation.

There are some general advantages and disadvantages to using open-source software. One advantage is that since the source is available if you are not able to get support from the open-source community, you can dive right in and find out exactly what is going on with the code. There is no need to wait for support from an external source, which can reduce debug time drastically.

Another advantage of open-source software is that it is not tied to any one specific company. In the case of proprietary software, if the owning company changes direction, or disappears, then the application developers using that solution are left out in the cold. Open-source software prevents this because developers have full access to the code and can choose the path for their own system software.

Sometimes vendors selling proprietary software are not always the most responsive to all customers using their product. In cases where a project is not destined to produce large volumes, the vendor's response to questions might not be as rapid as needed. The reality is that a company with higher revenue products will get the preferential treatment.

[1] Benenstein, Daniel. "Galleo Linux Multimedia Communicator." *Embedded Linux Journal* (July/August 2001): 14–19.

Having the source also gives you the ability to implement your own changes and customize the code exactly the way you need to for your specific application. In addition, only with open-source can the source-level configuration method, as we find in eCos, be used.

Although some proprietary software vendors offer their source code to developers, there is often a fee associated with getting said code.

Security is often looked at as a negative aspect of open-source developments. Because all of the source code is available to everyone, malicious developers can exploit security holes. On the other hand, the community of developers supporting the open-source project work quickly on a fix because it is in their best interest as well. You are not at the mercy of a single source providing a fix to the security problem.

A great source for getting viewpoints from some of the leading figures behind the open-source movement is *Open Sources: Voices from the Open Source Revolution*.[2]

Acknowledgments

This book could not have been completed without the hard work and tremendous effort of other people. I would like to start by thanking the technical reviewers, Jonathan Larmour (a.k.a. Jifl)—I appreciate your very insightful comments—and the technical support (occasionally real-time) throughout the development of this book. Other reviewers I am indebted to for sharing their keen comments and extremely valuable views include Grant Edwards, Bart Veer, Bill Gatliff, Larry Mittag, and Paul Beskeen. I would like to thank Michael Tiemann for writing the Foreword for this book.

I would like to thank the editor, Mark Taub—I appreciate the feedback and support you gave throughout this process.

Many thanks to the companies and open-source projects whose software is included on this CD-ROM—some brilliant stuff there.

Closing on a Personal Note

I would like to thank my Nonno and Nonna for all their support throughout my life. They are always there for me providing whatever is needed whether it's encouragement, a getaway to the backcountry for a ride and lunch, or a trip to the Boll Weevil. I sure miss Nonna. I love you both very much.

I would like to thank my brother, Laurie, and sister, Catherine, for their encouragement and understanding. You two are the best brother and sister anyone can ask for. By the way, I'd also like to congratulate my sister on passing the CPA exam, although I do believe the score is now 2 to 1—in my favor. :) I love you guys.

Thank you Mom and Dad for your never-ending support and encouragement for things I attempt in my career, and especially for things I attempt throughout my life. You are always there for me in whatever I do. I am very thankful to have such wonderful parents. I love you with all my heart.

[2] DiBona, Chris; Ockman, Sam; Stone, Mark. *Open Sources: Voices from the Open Source Revolution* (O'Reilly, 1999).

I would like to thank my wonderful daughter, Katie. You always could sense when I needed to take a break while working on this book. You not only knew that I should take a break, but you insisted that I leave the office immediately and forced me to watch one of your shows—although my preference is *Seinfeld*. Thanks for that; it really helped me to clear my head and refocus. You are very special to me and I love you with all my heart.

And, last but certainly not least, a big thank you to my wife, Deanna. Well, it's all over now. Thank you for supporting me on this effort and all efforts I undertake for us. Thank you for giving me the time to work on the book. I know it was difficult at times and put a lot of responsibility on you, but I hope the journey was worth it. Thank you not only for being a wonderful wife, but also for being my best friend. I love you, always.

Finally, I hope you all enjoy this book.

Anthony J. Massa
amassa@san.rr.com

CHAPTER 1

An Introduction to the eCos World

In this first chapter, we take a brief look at the origins of the Embedded Configurable Operating System (eCos) and the people and company behind it. We then get an overview of the configurable architecture of eCos, the core functionality, the different processors and evaluation platforms supported, and technical assistance options available.

Lastly, we get an overview of the eCos architecture and a look at the terminology used to describe the different pieces of the configuration system. The overview gives us a general idea of the components we detail in later chapters, and the terminology described in this chapter is used throughout the book.

1.1 Where It All Started—Cygnus Solutions

Michael Tiemann, David Henkel-Wallace, and John Gilmore founded Cygnus Solutions in 1989. The idea behind Cygnus Solutions was to provide high-quality support and development for open source software. It was initially unclear whether this business model would work out; however, by the end of the first year it was obvious from the value of the support and development contracts that the business was real. The workload was enormous for the five-person company (the three founders, a salesperson, and a part-time graduate student).

It was clear that the engineering support model worked; however, the costs to fulfill these contracts were very high. In order to generate income at a lower cost, the engineers had to put their heads together to come up with an idea. The plan was to focus their development efforts on a small set of open-source technology that could be sold. The key to maintaining this development on an order that could be handled by the group was to keep the focus very small. What they came up with was selling the GNU compiler (GCC) and debugger (GDB) as shrink-wrapped software. This was the right team of people to do the job. Michael Tiemann, who contributed

numerous GNU compiler ports and also wrote the first native C++ compiler (GNU C++ or G++), took on the task of working on GCC; David Henkel-Wallace worked on the binary utilities (binutils) and the library; and John Gilmore worked on GDB.

This task grew to monumental proportions. One advantage, or so it seemed, was that John Gilmore decided to become the new GDB maintainer. Making this known to the Internet community immediately flooded him with different versions of GDB. Now came the task of integrating these new version features.

Eventually, the hard work paid off in what today is called the GNUPro Developers Kit. The kit includes:

- **GCC—**the highly optimized ANSI-C compiler.
- **G++—**ANSI-tracking C++ compiler.
- **GDB—**source- and assembly-level debugger.
- **GAS—**GNU assembler.
- **LD—**GNU linker.
- **Cygwin—**UNIX environment for Windows.
- **Insight—**a graphical user interface (GUI) for GDB.
- **Source-Navigator—**source code comprehension tool.

1.2 The Origins of eCos

Initial design discussions for eCos began in the spring of 1997. The primary goal was to bring a cost-effective, high-quality embedded software solution to the marketplace. This new development would also complement the existing GNUPro tools, thereby expanding Cygnus' product offering.

Another essential requirement was that eCos needed to be designed in such a way that a small resource footprint could be constructed. By working with different semiconductor companies, Cygnus was able to architect a real-time operating system (RTOS) that abstracted the hardware layer and was highly configurable. This enabled the RTOS to fit into many diverse embedded systems. The highly configurable nature of eCos also allowed companies to reduce time to market for embedded products.

Reducing cost is always a concern in embedded systems. By using the open-source model, eCos was available with no initial costs. It could be downloaded and "test driven" free of charge. In addition to eliminating startup costs, another attractive cost-saving feature was that eCos had no backend charges—it had to be royalty-free.

Developers have full access to the entire software source code, including the tools, which can be modified as necessary (see Appendix B, *eCos License*, for the eCos license). There are no up-front license fees for the eCos run-time source code or any of the associated tools; everything needed to set up a complete embedded software development environment can be accomplished

for free. Developers do not have to contribute back any additional components or applications developed; however, they are required to contribute back modifications to the eCos code itself. These contributions help the open-source community develop a better product.

Today, numerous companies are using eCos, and many successful products have been launched running eCos, including the Brother HL-2400 CeN network color laser printer, Delphi Communiport, and the Iomega Hip Zip Digital Audio Player.

1.2.1 In a Word: Configurability

In order to get an understanding of the eCos architecture, it is important to appreciate the component framework that makes up the eCos system. This component framework is specifically targeted at embedded systems and meeting the requirements associated in embedded design. Using this framework, an enormous amount of functionality for an application can be built from reusable software components or software building blocks. The eCos component framework has been designed to control components to minimize memory use, allow users to control timing behavior to meet real-time requirements, and use usual programming languages (e.g., C, C++, and assembly for certain implementations in the Hardware Abstraction Layer [HAL]).

Most embedded software today provides more functionality than what might actually be needed for a particular application. Often, extra code is included in a software system that gives generic support for functionality that embedded developers are not concerned with and is not needed. This extra code makes the software unnecessarily more complex. Furthermore, the more code, the greater the chance of something going wrong. An example would be a simple “Hello World” program. With most RTOSes, full support for mutexes, task switching, and other features would be included, even though it is not necessary for such a simplistic application. eCos gives the developer ultimate control over run-time components where functionality that is not needed can easily be removed. eCos can be scaled from a few hundred bytes up to hundreds of kilobytes when features such as networking stacks are included and third-party contributions such as Web servers are used.

Developers are able to select components that satisfy basic application needs, and configure that particular component for the specific implementation requirements for the application. This could mean enabling or disabling a particular feature within a component, or selecting a particular implementation for the component. An example of this is in the kernel scheduler configuration. eCos offers the developer options such as the ability to select the number of priority levels and whether time slicing is used. Any code unnecessary to meeting the developer’s requirements is eliminated in the final image of the application.

Configurability allows a company to build an internal foundation of reusable components with access to the source code of the component. This can reduce development time and time to market because the components are highly portable and can be used in a wide range of applications. The eCos framework encourages third-party development to extend the features and functionality of the core eCos components. As more and more developers work toward extending the functionality on products and contribute these components back to the eCos project, the growth