

eCos User Guide

eCos User Guide

Copyright © 2001, 2002, 2003, 2004, 2009 Free Software Foundation, Inc.

Documentation licensing terms

This material may be distributed only subject to the terms and conditions set forth in the Open Publication License, v1.0 or later (the latest version is presently available at <http://www.opencontent.org/openpub/>).

Distribution of the work or derivative of the work in any standard (paper) book form is prohibited unless prior permission is obtained from the copyright holder.

Trademarks

Altera® and Excalibur™ are trademarks of Altera Corporation.

AMD® is a registered trademark of Advanced Micro Devices, Inc.

ARM®, StrongARM®, Thumb®, ARM7™, ARM9™ is a registered trademark of Advanced RISC Machines, Ltd.

Cirrus Logic® and Maverick™ are registered trademarks of Cirrus Logic, Inc.

Cogent™ is a trademark of Cogent Computer Systems, Inc.

Compaq® is a registered trademark of the Compaq Computer Corporation.

Fujitsu® is a registered trademark of Fujitsu Limited.

IBM®, and PowerPC™ are trademarks of International Business Machines Corporation.

IDT® is a registered trademark of Integrated Device Technology Inc.

Intel®, i386™, Pentium®, StrataFlash® and XScale™ are trademarks of Intel Corporation.

Intrinsyc® and Cerf™ are trademarks of Intrinsyc Software, Inc.

Linux® is a registered trademark of Linus Torvalds.

Matsushita™ and Panasonic® are trademarks of the Matsushita Electric Industrial Corporation.

Microsoft®, Windows®, Windows NT® and Windows XP® are registered trademarks of Microsoft Corporation, Inc.

MIPS®, MIPS32™ MIPS64™, 4K™, 5K™ Atlas™ and Malta™ are trademarks of MIPS Technologies, Inc.

Motorola®, ColdFire® is a trademark of Motorola, Inc.

NEC® V800™, V850™, V850/SA1™, V850/SB1™, VR4300™, and VRC4375™ are trademarks of NEC Corporation.

PMC-Sierra® RM7000™ and Ocelot™ are trademarks of PMC-Sierra Incorporated.

Red Hat, eCos™, RedBoot™, GNUPro®, and Insight™ are trademarks of Red Hat, Inc.

Samsung® and CalmRISC™ are trademarks or registered trademarks of Samsung, Inc.

Sharp® is a registered trademark of Sharp Electronics Corp.

SPARC® is a registered trademark of SPARC International, Inc., and is used under license by Sun Microsystems, Inc.

Sun Microsystems® and Solaris® are registered trademarks of Sun Microsystems, Inc.

SuperH™ and Renesas™ are trademarks owned by Renesas Technology Corp.

Texas Instruments®, OMAP™ and Innovator™ are trademarks of Texas Instruments Incorporated.

Toshiba® is a registered trademark of the Toshiba Corporation.

UNIX® is a registered trademark of The Open Group.

All other brand and product names, trademarks, and copyrights are the property of their respective owners.

Table of Contents

I. Introduction	xiii
1. Key Features.....	1
2. eCos Overview	3
3. eCos Licence Overview	5
Questions and answers	5
Previous License.....	6
4. Notation and Conventions.....	7
GDB and GCC Command Notation	7
Directory and File System Conventions	7
Version Conventions	7
5. Documentation Roadmap.....	9
II. Installing eCos.....	11
6. System Requirements.....	13
7. Installation on Linux	15
8. Installation on Windows.....	17
9. Target Setup.....	19
Connecting Via Serial Line	19
Connecting Via Ethernet.....	19
Using A Simulator Target.....	19
Using A Synthetic Target	20
III. Programming With eCos.....	21
10. Programming With eCos.....	23
The Development Process	23
eCos Configuration	23
Integrity check of the eCos configuration.....	23
Application Development - Target Neutral Part	23
Application Development - Target Specific Part.....	24
11. Configuring and Building eCos from Source.....	25
eCos Start-up Configurations	25
Configuration Tool on Windows and Linux Quick Start.....	26
Ecosconfig on Windows and Linux Quick Start.....	30
Selecting a Target.....	31
12. Running an eCos Test Case.....	33
Using the Configuration Tool	33
Using the command line.....	34
Testing Filters	36
13. Building and Running Sample Applications.....	37
eCos Hello World	37
eCos hello world program listing.....	37
A Sample Program with Two Threads	38
eCos two-threaded program listing.....	38
14. More Features — Clocks and Alarm Handlers	41
A Sample Program with Alarms.....	41

IV. The eCos Configuration Tool.....	45
15. Getting Started	47
Introduction	47
Invoking the eCos Configuration Tool.....	47
On Linux.....	47
On Windows.....	47
The Component Repository.....	48
eCos Configuration Tool Documents	49
Configuration Save File	49
Save the currently active document.....	49
Open an existing document	50
Open a document you have used recently	50
Create a new blank document based on the Component Registry	51
Save to a different file name.....	51
Build and Install Trees	51
16. Getting Help	53
Context-sensitive Help for Dialogs	53
Context-sensitive Help for Other Windows.....	53
Context-sensitive Help for Configuration Items.....	53
Methods of Displaying HTML Help	53
17. Customization	55
Window Placement.....	55
Settings	55
Settings: Display tab	55
Labels	55
Integer Items.....	56
Font.....	56
Miscellaneous.....	56
Settings: Viewers tab.....	56
View header files	57
View documentation.....	57
18. Screen Layout	59
Configuration Window	59
Disabled items.....	60
Right-Clicking.....	60
Conflicts Window.....	60
Output Window	61
Properties Window	61
Short Description Window	62
19. Updating the Configuration.....	63
Adding and Removing Packages.....	63
Platform Selection	64
Using Templates	65
Resolving conflicts.....	65
Automatic resolution.....	66
20. Searching.....	69
21. Building.....	71
Selecting Build Tools.....	71

Selecting User Tools	72
22. Execution	73
Properties	73
Download Timeout	73
Run time Timeout	74
Connection	74
Executables Tab	74
Output Tab	75
Summary Tab	76
23. Creating a Shell	77
Keyboard Accelerators	77
V. eCos Programming Concepts and Techniques	79
24. CDL Concepts	81
About this chapter	81
Background	81
Configurations	81
Component Repository	81
Component Definition Language	81
Packages	81
Configuration Items	82
Expressions	82
Properties	82
Inactive Items	83
Conflicts	83
Templates	84
25. The Component Repository and Working Directories	85
Component Repository	85
Purpose	86
How is it modified?	86
When is it edited manually?	87
User Applications	87
Examples of files in this hierarchy	87
Build Tree	87
Purpose	87
How is it modified?	88
User applications	88
Examples of files in this hierarchy	88
Install Tree	88
Purpose	88
How is it modified?	88
When is it edited manually?	88
User applications	89
Examples of files in this hierarchy	89
Application Build Tree	89
26. Compiler and Linker Options	91
Compiling a C Application	91
Compiling a C++ Application	91

27. Debugging Techniques	93
Tracing	93
Kernel Instrumentation	94
VI. Configuration and the Package Repository	99
28. Manual Configuration	101
Directory Tree Structure	101
Creating the Build Tree	101
ecosconfig qualifiers	102
ecosconfig commands	103
Conflicts and constraints	104
Building the System	106
Packages	107
Coarse-grained Configuration	107
Fine-grained Configuration	108
Editing an eCos Savefile	109
Header	109
Toplevel Section	110
Conflicts Section	111
Data Section	111
Tel Syntax	120
Editing the Sources	124
Modifying the Memory Layout	125
29. Managing the Package Repository	127
Package Installation	127
Using the Administration Tool	127
Using the command line	128
Package Structure	129
VII. Appendixes	131
A. Target Setup	133
MN10300 stdevall Hardware Setup	133
MN10300 Architectural Simulator Setup	133
AM33 STB Hardware Setup	134
Use with GDB Stub ROM	134
Use with the JTAG debugger	134
Building the GDB stub ROM image	135
TX39 Hardware Setup	136
TX39 Architectural Simulator Setup	136
TX49 Hardware Setup	137
Preparing the GDB stubs	137
Building the GDB stub image with the eCos Configuration Tool	137
Building the GDB stub image with ecosconfig	137
Installing GDB stubs into FLASH	137
VR4300 Hardware Setup	138
VRC4375 Hardware Setup	138
Atlas/Malta Hardware Setup	139
PowerPC Cogent Hardware Setup	139
Installing the Stubs into ROM	139

Preparing the Binaries	139
Building the ROM images with the eCos Configuration Tool	139
Building the ROM images with ecosconfig	140
Installing the Stubs into ROM or FLASH	140
PowerPC MBX860 Hardware Setup	140
Installing the Stubs into FLASH	141
Preparing the Binaries	141
Building the ROM images with the eCos Configuration Tool	141
Building the ROM images with ecosconfig	141
Installing the Stubs into ROM	141
Installing the Stubs into FLASH	142
Program FLASH	142
PowerPC Architectural Simulator Setup	142
SPARClite Hardware Setup	143
Ethernet Setup	143
BOOTP/DHCP service on Linux	144
BOOTP/DHCP boot process	144
Serial Setup	144
SPARClite Architectural Simulator Setup	145
ARM PID Hardware Setup	145
Installing the Stubs into FLASH	145
Preparing the Binaries	146
Building the ROM images with the eCos Configuration Tool	146
Building the ROM images with ecosconfig	146
Building the FLASH Tool with the eCos Configuration Tool	146
Building the FLASH Tool with ecosconfig	147
Prepare the Board for FLASH Programming	147
Program the FLASH	147
Programming the FLASH for big-endian mode	149
Installing the Stubs into ROM	150
ARM AEB-1 Hardware Setup	150
Overview	150
Talking to the Board	150
Downloading the Stubs via the Rom Menu	151
Activating the GDB Stubs	151
Building the GDB Stub FLASH ROM Images	152
Building the GDB Stubs with the eCos Configuration Tool	152
Building the GDB Stub ROMs with ecosconfig	152
ARM Cogent CMA230 Hardware Setup	153
Building the GDB Stub FLASH ROM images	153
Building the GDB Stubs with the eCos Configuration Tool	153
Building the GDB Stub ROMs with ecosconfig	154
Cirrus Logic ARM EP7211 Development Board Hardware Setup	154
Building programs for programming into FLASH	155
Building the GDB Stub FLASH ROM images	155
Building the ROM images with the eCos Configuration Tool	155
Building the ROM images with ecosconfig	155
Loading the ROM Image into On-board Flash	156

Building the Flash Downloader on Linux.....	157
Developing eCos Programs with the ARM Multi-ICE.....	157
Cirrus Logic ARM EP7212 Development Board Hardware Setup.....	158
Cirrus Logic ARM EP7312 Development Board Hardware Setup.....	158
90MHz Operation	159
Cirrus Logic ARM EP7209 Development Board Hardware Setup.....	159
Cirrus Logic ARM CL-PS7111 Evaluation Board Hardware Setup.....	159
StrongARM EBSA-285 Hardware Setup.....	160
Building the GDB Stub FLASH ROM images.....	160
Building the GDB Stubs with the eCos Configuration Tool.....	161
Building the GDB Stub ROMs with ecosconfig.....	161
Loading the ROM Image into On-board Flash.....	161
Running your eCos Program Using GDB and the StubROM.....	162
Compaq iPAQ PocketPC Hardware Setup	163
Arm Industrial Module AIM 711 Hardware Setup	163
Setup Hardware.....	163
Power supply	164
Serial devices.....	164
Ethernet	164
Installing RedBoot into FLASH	164
Using RedBoot	164
Using JTAG.....	165
More documentation	165
SH3/EDK7708 Hardware Setup.....	165
Installing the Stubs into FLASH.....	165
Preparing the Binaries	165
Building the ROM images with the eCos Configuration Tool.....	165
Building the ROM images with ecosconfig.....	166
Installing the Stubs into ROM or FLASH.....	166
SH3/CQ7708 Hardware Setup	166
Preparing the board.....	166
eCos GDB Stubs	167
Preparing the GDB stubs.....	167
Building the GDB stub image with the eCos Configuration Tool.....	167
Building the GDB stub image with ecosconfig.....	167
Programming the stubs in EPROM/FLASH.....	167
SH3/HS7729PCI Hardware Setup.....	168
SH3/SE77x9 Hardware Setup	168
SH4/CQ7750 Hardware Setup	168
Preparing the board.....	169
eCos GDB Stubs	169
Preparing the GDB stubs.....	169
Building the GDB stub image with the eCos Configuration Tool.....	169
Building the GDB stub image with ecosconfig.....	169
Programming the stubs in EPROM/FLASH.....	170
SH4/SE7751 Hardware Setup	170
NEC CEB-V850/SA1 Hardware Setup.....	171
Installing the Stubs into ROM	171

Preparing the Binaries	171
Building the ROM images with the eCos Configuration Tool	171
Building the ROM images with ecosconfig	172
Installing the Stubs into ROM or FLASH	172
Debugging with the NEC V850 I.C.E.	172
INITIAL SETUP	172
BUILD PROCEDURES	173
V850ICE.EXE EXECUTION	173
V850-ELF-GDB EXECUTION	174
MDI INTERFACE VS. GDB INTERFACE	175
eCos THREAD DEBUGGING	175
NEC CEB-V850/SB1 Hardware Setup	176
i386 PC Hardware Setup	176
RedBoot Support	176
Floppy Disk Support	176
GRUB Bootloader Support	177
Debugging FLOPPY and GRUB Applications	178
i386/Linux Synthetic Target Setup	178
Tools	178
B. Real-time characterization	181
Board: ARM AEB-1 Revision B Evaluation Board	181
Board: Atmel AT91/EB40	183
Board: Intel StrongARM EBSA-285 Evaluation Board	186
Board: Cirrus Logic EDB7111-2 Development Board	188
CPU : Cirrus Logic EP7211 73MHz	188
CPU : Cirrus Logic EP7212 73MHz	190
Board: ARM PID Evaluation Board	193
CPU : ARM 7TDMI 20 MHz	193
CPU : ARM 920T 20 MHz	195
Board: Intel IQ80310 XScale Development Kit	198
Board: Toshiba JMR3904 Evaluation Board	200
Board: Toshiba REF 4955	202
Board: Matsushita STDEVAL1 Board	205
Board: Fujitsu SPARClite Evaluation Board	207
Board: Cogent CMA MPC860 (PowerPC) Evaluation	209
Board: NEC VR4373	211
Board: Intel SA1110 (Assabet)	214
Board: Intel SA1100 (Brutus)	216
Board: Motorola MBX	219
Board: Hitachi EDK7708	221
Board: CQ CqREEK SH3 Evaluation Board (cq7708)	223
Board: Hitachi HS7729PCI HS7729 SH3	226
Board: Hitachi Solution Engine 7751 SH4 (se7751)	228
Board: PC	231
Board: NEC V850 Cosmo Evaluation Board	233
Board: NEC V850 Cosmo Evaluation Board	235
Board: ARM Industrial Module AIM711 (S3C4510)	238
C. GNU General Public License	241

List of Tables

11-1. Configuration for various download methods	25
18-1. Cell types	59
23-1. Keyboard accelerators	77
24-1. CDL Expressions.....	82
24-2. Configuration properties.....	83

I. Introduction

Chapter 1. Key Features

- eCos is distributed under the GPL license with an exception which permits proprietary application code to be linked with eCos without itself being forced to be released under the GPL. It is also royalty and buyout free.
- As an Open Source project, eCos is under constant improvement, with an active developer community, based around the eCos web site at <http://ecos.sourceforge.org/>.
- Powerful GUI-based configuration system allowing both large and fine grained configuration of eCos. This allows the functionality of eCos to be customized to the exact requirements of the application.
- Full-featured, flexible, configurable, real time embedded kernel. The kernel provides thread scheduling, synchronization, timer, and communication primitives. It handles hardware resources such as interrupts, exceptions, memory and caches.
- The Hardware Abstraction Layer (HAL) hides the specific features of each supported CPU and platform, so that the kernel and other run-time components can be implemented in a portable fashion.
- Support for μ ITRON and POSIX Application Programmer Interfaces (APIs). It also includes a fully featured, thread-safe ISO standard C library and math library.
- Support for a wide variety of devices including many serial devices, ethernet controllers and FLASH memories. There is also support for PCMCIA, USB and PCI interconnects.
- A fully featured TCP/IP stack implementing IP, IPv6, ICMP, UDP and TCP over ethernet. Support for SNMP, HTTP, TFTP and FTP are also present.
- The RedBoot ROM monitor is an application that uses the eCos HAL for portability. It provides serial and ethernet based booting and debug services during development.
- Many components include test programs that validate the components behaviour. These can be used both to check that hardware is functioning correctly, and as examples of eCos usage.
- eCos documentation included this User Guide, the Reference Manual and the Components Writer's Guide. These are being continually updated as the system develops.

Chapter 2. eCos Overview

eCos is an open source, configurable, portable, and royalty-free embedded real-time operating system. The following text expands on these core aspects that define eCos.

eCos is provided as an open source runtime system supported by the GNU open source development tools. Developers have full and unfettered access to all aspects of the runtime system. No parts of it are proprietary or hidden, and you are at liberty to examine, add to, and modify the code as you deem necessary. These rights are granted to you and protected by the GNU Public License (GPL). An exception clause has been added to the eCos license which limits the circumstances in which the license applies to other code when used in conjunction with eCos. This exception grants you the right to freely develop and distribute applications based on eCos. You are not expected or required to make your embedded applications or any additional components that you develop freely available so long as they are not derived from eCos code. We of course welcome all contributions back to eCos such as board ports, device drivers and other components, as this helps the growth and development of eCos, and is of benefit to the entire eCos community. See [Chapter 3](#) for more details.

One of the key technological innovations in eCos is the configuration system. The configuration system allows the application writer to impose their requirements on the run-time components, both in terms of their functionality and implementation, whereas traditionally the operating system has constrained the application's own implementation. Essentially, this enables eCos developers to create their own application-specific operating system and makes eCos suitable for a wide range of embedded uses. Configuration also ensures that the resource footprint of eCos is minimized as all unnecessary functionality and features are removed. The configuration system also presents eCos as a component architecture. This provides a standardized mechanism for component suppliers to extend the functionality of eCos and allows applications to be built from a wide set of optional configurable run-time components. Components can be provided from a variety of sources including: the standard eCos release; commercial third party developers or open source contributors.

The royalty-free nature of eCos means that you can develop and deploy your application using the standard eCos release without incurring any royalty charges. In addition, there are no up-front license charges for the eCos runtime source code and associated tools. We provide, without charge, everything necessary for basic embedded applications development.

eCos is designed to be portable to a wide range of target architectures and target platforms including 16, 32, and 64 bit architectures, MPUs, MCUs and DSPs. The eCos kernel, libraries and runtime components are layered on the Hardware Abstraction Layer (HAL), and thus will run on any target once the HAL and relevant device drivers have been ported to the target's processor architecture and board. Currently eCos supports a large range of different target architectures:

- ARM, Intel StrongARM and XScale
- Fujitsu FR-V
- Hitachi SH2/3/4
- Hitachi H8/300H
- Intel x86
- MIPS
- Matsushita AM3x
- Motorola PowerPC

- Motorola 68k/Coldfire
- NEC V850
- Sun SPARC

including many of the popular variants of these architectures and evaluation boards.

eCos has been designed to support applications with real-time requirements, providing features such as full pre-emptability, minimal interrupt latencies, and all the necessary synchronization primitives, scheduling policies, and interrupt handling mechanisms needed for these type of applications. eCos also provides all the functionality required for general embedded application support including device drivers, memory management, exception handling, C, math libraries, etc. In addition to runtime support, the eCos system includes all the tools necessary to develop embedded applications, including eCos software configuration and build tools, and GNU based compilers, assemblers, linkers, debuggers, and simulators.

To get the most out of eCos you should visit the eCos open source developers site: <http://ecos.sourceforge.org/>.

The site is dedicated to the eCos developer community and contains a rich set of resources including news, FAQ, online documentation, installation guide, discussion and announcement mailing lists, and runtime and development tools downloads. The site also supports anonymous CVS and WEB CVS access to provide direct access to the latest eCos source base.

eCos is released as open source software because we believe that this is the most effective software development model, and that it provides the greatest benefit to the embedded developer community as a whole. As part of this endeavor, we seek the input and participation of eCos developers in its continuing evolution. Participation can take many forms including:

- providing us with feedback on how eCos might be made more useful to you - by taking part in the ongoing mailing list discussions and by submitting problem reports covering bugs, documentation issues, and missing features
- contributing bug fixes and enhancement patches
- contributing new code including device drivers, board ports, libraries, and other runtime components

Our long term aim is to make eCos a rich and ubiquitous standard infrastructure for the development of deeply embedded applications. This will be achieved with the assistance of the eCos developer community cooperating to improve eCos for all. We would like to take this opportunity to extend our thanks to the many eCos developers who have already contributed feedback, ideas, patches, and code that have augmented and improved this release.

The eCos Maintainers

Chapter 3. eCos Licence Overview

As of May 2002, eCos is released under a modified version of the well known GNU General Public License (GPL) (<http://www.gnu.org/copyleft/gpl.html>), now making it an official GPL-compatible Free Software License (<http://www.gnu.org/philosophy/license-list.html>). An exception clause has been added to the eCos license which limits the circumstances in which the license applies to other code when used in conjunction with eCos. The exception clause is as follows:

```
As a special exception, if other files instantiate templates or use macros
or inline functions from this file, or you compile this file and link it
with other works to produce a work based on this file, this file does not
by itself cause the resulting work to be covered by the GNU General Public
License. However the source code for this file must still be made
available in accordance with section (3) of the GNU General Public
License.
```

```
This exception does not invalidate any other reasons why a work based on
this file might be covered by the GNU General Public License.
```

The goal of the license is to serve the eCos user community as a whole. It allows all eCos users to develop products without paying anybody anything, no matter how many developers are working on the product or how many units will be shipped. The license also guarantees that the eCos source code will always be freely available. This applies not only to the core eCos code itself but also to any changes that anybody makes to the core. In particular, it should prevent any company or individual contributing code to the system and then later claiming that all eCos users are now guilty of copyright or patent infringements and have to pay royalties. It should also prevent any company from making some small improvements, calling the result a completely new system, and releasing this under a new and less generous license.

The license does *not* require users to release the source code of any *applications* that are developed with eCos. However, if anybody makes any changes to code covered by the eCos license, or writes new files derived in any way from eCos code, then we believe that the entire user community should have the opportunity to benefit from this. The license stipulates that these changes must be made available in source code form to all recipients of binaries based on the modified code, either by including the sources along with the binaries you deliver (or with any device containing such binaries) or with a written offer to supply the source code to the general public for three years. It is perhaps most practical for eCos developers to make the source code available online and inform those who are receiving binaries containing eCos code, and probably also the eCos maintainers, about the location of the code. See the full text of the GPL (<http://www.gnu.org/copyleft/gpl.html>) for the most authoritative definition of the obligations.

Although it is not strictly necessary to contribute the modified code back to the eCos open source project, we are always pleased to receive code contributions and hope that developers will also be keen to give back in return for what they received from the eCos project completely free of charge. The eCos maintainers are responsible for deciding whether such contributions should be applied to the public repository. In addition, a copyright assignment (<http://ecos.sourceware.org/assign.html>) is required for any significant changes to the core eCos packages.

The result is a royalty-free system with minimal obligations on the part of application developers. This has resulted in the rapid uptake of eCos. At the same time, eCos is fully open source with all the benefits that implies in terms of quality and innovation. We believe that this is a winning combination.

Questions and answers

The following queries provide some clarification as to the implications of the eCos license. They do not constitute part of the legal meaning of the license.

Q. What is the effect of the eCos license?

A. In the simplest terms, when you distribute anything containing eCos code, you must make the source code to eCos available under the terms of the GPL.

Q. What if I make changes to eCos, or write new code based on eCos code?

A. Then you must make those changes available as well.

Q. Do I have to distribute the source code to my application? Isn't the GPL "viral"?

A. You do not have to distribute any code under the terms of the GPL other than eCos code or code derived from eCos. For example, if you write a HAL port based on copying an existing eCos HAL in any way, you must make the source code available with the binary. However you would not need to make available any other code, such as the code of a wholly separate application linked with eCos.

Q. I would rather stick with the RHEPL code, but I updated my anonymous CVS checkout.

A. You can check out the final version of anonymous CVS before the license change using the CVS tag `last-rhepl`. See the anonymous CVS access page (<http://ecos.sourceware.org/anoncv.html>) for details.

Previous License

Prior to May 2002, eCos was released under the Red Hat eCos Public License (RHEPL) (<http://ecos.sourceware.org/old-license.html>). The RHEPL required any modifications to eCos code to be made available under preferential terms to Red Hat and was therefore incompatible with code licensed under the GPL. The use of eCos source code which was licensed under the RHEPL is not affected by the switch to the modified GPL for later revisions.

Chapter 4. Notation and Conventions

Since there are many supported target architectures, notation conventions are used in this manual to avoid repeating instructions that are very similar.

GDB and GCC Command Notation

Cross-development commands like **gcc** and **gdb** will be shown with a *TARGET-* prefix. You need to replace *TARGET-* with the correct prefix before using the command. Just using **gcc** or **gdb** will use the tools for the host, which is not (usually) what you want.

For example use **arm-elf-gcc** and **arm-elf-gdb** for ARM, Thumb, and StrongARM targets. Use **xscale-elf-gcc** and **xscale-elf-gdb** for Intel Xscale targets. Use **i386-elf-gcc** and **i386-elf-gdb** for IA32 targets. And so on, the exact prefix to use is shown in the documentation for each target.

Note that some versions of the GCC cross compiler generate executable files with the `.exe` suffix on Windows, but not on Linux. The suffix `.exe` will be omitted from executable file names, so you will see `hello` instead of `hello.exe`.

Directory and File System Conventions

The default directory for installing eCos on Windows (usually `C:/Program Files/eCos`) is different from that on Linux (usually `/opt/ecos`). Since many command line examples in the tutorials use these paths, this default (base) directory will be cited as *BASE_DIR*.

Windows and Linux have a similar file system syntax, but the MS-DOS command interpreter on Windows uses the backslash character (`\`) as a path separator, while Linux and POSIX shells (including the Cygwin bash shell for windows) use the forward slash (`/`).

This document will use the POSIX shell convention of forward slashes throughout.

Version Conventions

This manual does not refer explicitly to any particular version of eCos. However, version numbers form part of many file path names. In all of these places the version number will be shown like this: *<version>*.

If you have used anonymous CVS to check eCos out of the CVS repository, the version number will always be *current*, since that is the name of the directory in the repository. When a stable release is made this directory name is changed, in the release, to the number of the release, for example `v2_0` or `v2_1`.

Chapter 5. Documentation Roadmap

The eCos documentation is divided into a three main parts:

User Guide

This document. It includes the following sections:

Installing eCos

This section describes how to install the eCos software, how to set up your hardware and how to test that it is all working.

Programming Under eCos

This section describes how to write programs that run under eCos by running through some examples.

The eCos Configuration Tool

This section describes the eCos graphical configuration tool and how to use it to change how eCos behaves.

eCos Programming Concepts and Techniques

An explanation of the eCos programming cycle, and a description of some debugging facilities that eCos offers.

Configuration and the Package Repository

Information on how to configure eCos manually, including a reference on the **ecosconfig** command, memory layouts, and information on how to manage a package repository using the eCos Package Administration Tool.

Reference Guide

The Reference Guide provides detailed documentation on various aspects of eCos. This document is being constantly updated, so the following list just mentions the more important sections, take a look at the guide itself for the full story.

The eCos Kernel

In-depth description of eCos's native C kernel API Important considerations are given for programming the eCos kernel. The semantics for each kernel function are described, including how they are affected by configuration.

POSIX and μ ITRON APIs

A description of the POSIX and μ ITRON APIs and how they are supported under eCos.

The eCos Hardware Abstraction Layer (HAL)

A description of the structure and functionality of the eCos HAL. This section also includes a porting guide to help moving eCos to different platforms.

Device Drivers

A description of the philosophy behind eCos device drivers, as well as a presentation of the C language APIs for using the current device drivers.

Device driver support includes serial, ethernet and FLASH devices, and support for PCI, PCMCIA and USB interconnects.

RedBoot User's Guide

This describes RedBoot, which provides a complete bootstrap environment for a range of embedded operating systems, such as embedded Linux and eCos, and includes facilities such as network downloading and debugging. It also provides a simple flash file system for boot images.

TCP/IP Stack Support

This describes the Common Networking for eCos package, which provides support for a complete TCP/IP networking stack. The design allows for the actual stack to be modular and at the current time two different implementations, one based on OpenBSD from 2000 and a new version based on FreeBSD, are available.

Other components related to networking, including support for SNMP, DNS, HTTP and FTP, are also described.

Component Writer's Guide

The Component Writer's Guide is intended for developers who need to add or modify parts of eCos itself. It describes the following things:

Overview

An explanation of the configuration technology used in eCos, why it is done this way, how it works and the terminology used.

Package Organization

A description of the eCos package repository, how it is organized and how packages themselves are organized.

The CDL Language

A description of the CDL language and how it is used to control the configuration of eCos components. The document also contains a complete specification of the language.

The Build Process

A description of what happens once a configuration has been created and must be built into a set of executables.

II. Installing eCos

Chapter 6. System Requirements

- Standard Intel architecture PC running Linux (tested on recent Fedora, openSUSE and Ubuntu distributions), Microsoft Windows NT 4 + SP6a, Windows 2000, Windows XP and Windows Vista. Linux distributions from other vendors may also work, but are currently untested.
- Enough disk space for the installed distribution. The eCos installation process will detail the various components of eCos and the compiler toolkit that can be installed, and their disk space requirements.
- 64MB of RAM and a 350MHz or faster Pentium processor.

If you are downloading the eCos release distribution from ecos.sourceforge.org (<http://ecos.sourceforge.org>), you will also need space to store that image and to compile the toolchain and eCos from source.

Chapter 7. Installation on Linux

Full instructions for the downloading and installation of eCos (<http://ecos.sourceforge.org/getstart.html>) on Linux hosts are provided on the eCos website.

Chapter 8. Installation on Windows

Full instructions for the downloading and installation of eCos (<http://ecos.sourceware.org/getstart.html>) on Windows hosts are provided on the eCos website.

Chapter 9. Target Setup

While eCos supports a variety of targets, communication with all the targets happens in one of four ways. These are described in general below. Any details or variations from these descriptions will be found in the eCos documentation for a specific target, in the appendix.

Connecting Via Serial Line

Most targets will have RedBoot or GDB Stubs installed. These normally wait for GDB to connect at 38400 baud, using 8 data bit, no parity bit and 1 stop-bit and no hardware flow control. Check the documentation for your target to ensure it uses this speed. If not, adjust the following instructions accordingly.

The following instructions depend on your having selected the appropriate serial port on the host. That is, the serial port which connects to the target's (primary) serial port. On Linux this could be `/dev/ttyS0`, while the same port on Windows would be named COM1. Substitute the proper serial port name in the below.

Connect to the target by issuing the following commands in GDB console mode:

```
(gdb) set remotebaud 38400
(gdb) target remote /dev/ttyS0
```

In Insight, connect by opening the *File->Target Settings* window and enter:

```
Target: Remote/Serial
Baud Rate: 38400
Port: /dev/ttyS0
```

Set other options according to preference, close the window and select *Run->Connect to target*.

Connecting Via Ethernet

Some targets allow GDB to connect via Ethernet - if so, it will be mentioned in the document describing the target. Substitute the target's assigned IP address or hostname for `<hostname>` in the following. Depending on how RedBoot has been configured, it will either have this address allocated statically, or will acquire it via BOOTP. In both cases RedBoot will report the IP address it is listening on in its startup message printed on the serial port. The `<port>` is the TCP port which RedBoot is listening on, usually 9000. It is also listed in the target document.

Connect to the target by issuing the following command in GDB console mode:

```
(gdb) target remote <hostname>:<port>
```

In Insight, connect by opening the *File->Target Settings* window and enter:

```
Target: Remote/TCP
Hostname: <hostname>
Port: <port>
```

Set other options according to preference, close the window and select *Run->Connect to target*.

Using A Simulator Target

GDB connects to all simulator targets using the same basic command, although each simulator may require additional options. These are listed in the document describing the target, and should be used when connecting.

Connect to the target by issuing the following command in GDB console mode:

```
(gdb) target sim [target specific options]
```

In Insight, connect by opening the *File->Target Settings* window and enter:

```
Target: Simulator  
Options: [target specific options]
```

Set other options according to preference, close the window and select *Run->Connect to target*.

Using A Synthetic Target

Synthetic targets are special in that the built tests and applications actually run as native applications on the host. This means that there is no target to connect to. The test or application can be run directly from the GDB console using:

```
(gdb) run
```

or from Insight by pressing the *Run* icon. There is therefore no need to connect to the target or download the application, so you should ignore GDB “target” and “load” commands in any instructions found in other places in the documentation.

III. Programming With eCos

Chapter 10. Programming With eCos

The following chapters of this manual comprise a simple tutorial for configuring and building eCos, building and running eCos tests, and finally building three stand-alone example programs which use the eCos API to perform some simple tasks.

You will need a properly installed eCos system, with the correct versions of the GNU toolchain. On Windows you will be using the bash command line interpreter that comes with Cygwin, with the environment variables set as described in the toolchain documentation.

The Development Process

Most development projects using eCos would contain some (or most) of the following:

eCos Configuration

eCos is configured to provide the desired API (the inclusion of libc, uitron, and the disabling of certain undesired functions, etc.), and semantics (selecting scheduler, mutex behavior, etc.). See [Chapter 11](#).

It would normally make sense to enable eCos assertion checking at this time as well, to catch as many programming errors during the development phase as possible.

Note that it should not be necessary to spend much time on eCos configuration initially. It may be important to perform fine tuning to reduce the memory footprint and to improve performance later when the product reaches a testable state.

Integrity check of the eCos configuration

While we strive to thoroughly test eCos, the vast number of configuration permutations mean that the particular configuration parameters used for your project may not have been tested. Therefore, we advise running the eCos tests after the project's eCos configuration has been determined. See [Chapter 12](#).

Obviously, this should be repeated if the configuration changes later on in the development process.

Application Development - Target Neutral Part

While your project is probably targeting a specific architecture and platform, possibly custom hardware, it may be possible to perform part of the application development using simulated or synthetic targets.

There are three good reasons for doing this:

- It may be possible by this means to perform application development in parallel with the design/implementation of the target hardware, thus providing more time for developing and testing functionality, and reducing time-to-market.
- The build-run-debug-cycle may be faster when the application does not have to be downloaded to a target via a serial interface. Debugging is also likely to be more responsive when you do not have to communicate with

the remote GDB stubs in RedBoot via serial. It also removes the need for manually or automatically resetting the target hardware.

- New hardware can often be buggy. Comparing the behaviour of the program on the hardware and in the simulator or synthetic target may allow you to identify where the problems lie.

This approach is possible because all targets (including simulators and synthetic ones) provide the same basic API: that is, kernel, libc, libm, uitron, infra, and to some extent, HAL and IO.

Synthetic targets are especially suitable as they allow you to construct simulations of elaborate devices by interaction with the host system, where an IO device API can hide the details from the application. When switching to hardware later in the development cycle, the IO driver is properly implemented.

Simulators can also do this, but it all depends on the design and capabilities of the simulator you use. Some, like SID (<http://sources.redhat.com/sid>) or Bochs (<http://bochs.sourceforge.net/>) provide complete hardware emulation, while others just support enough of the instruction set to run compiled code.

Therefore, select a simulator or synthetic target and use it for as long as possible for application development. That is, configure for the selected target, build eCos, build the application and link with eCos, run and debug. Repeat the latter two steps until you are happy with it.

Obviously, at some time you will have to switch to the intended target hardware, for example when adding target specific feature support, for memory footprint/performance characterization, and for final tuning of eCos and the application.

Application Development - Target Specific Part

Repeat the build-run-debug-cycle while performing final tuning and debugging of application. Remember to disable eCos assertion checking if you are testing any performance-related aspects, it can make a big difference.

It may be useful to switch between this and the previous step repeatedly through the development process; use the simulator/synthetic target for actual development, and use the target hardware to continually check memory footprint and performance. There should be little cost in switching between the two targets when using two separate build trees.

Chapter 11. Configuring and Building eCos from Source

This chapter documents the configuration of eCos. The process is the same for any of the supported targets: you may select a hardware target (if you have a board available), any one of the simulators, or a synthetic target (if your host platform has synthetic target support).

eCos Start-up Configurations

There are various ways to download an executable image to a target board, and these involve different ways of preparing the executable image. In the eCos Hardware Abstraction Layer (HAL package) there are configuration options to support the different download methods. [Table 11-1](#) summarizes the ways in which an eCos image can be prepared for different types of download. This is not an exhaustive list, some targets define additional start-up types of their own. Where a ROM Monitor is mentioned, this will usually be RedBoot, although on some older, or low resource, targets you may need to use CygMon or the GDB stubs ROM, see the target documentation for details.

Table 11-1. Configuration for various download methods

Download method	HAL configuration
Burn hardware ROM	ROM or ROMRAM start-up
Download to ROM emulator	ROM or ROMRAM start-up
Download to board with ROM Monitor	RAM start-up
Download to simulator without ROM Monitor	ROM start-up
Download to simulator with ROM Monitor	RAM start-up
Download to simulator ignoring devices	SIM configuration
Run synthetic target	RAM start-up

Caution

You cannot run an application configured for RAM start-up on the simulator directly: it will fail during start-up. You can only download it to the simulator if you are already running RedBoot in the simulator, as described in the toolchain documentation or you load through the *SID* GDB debugging component. This is not the same as the simulated stub, since it does not require a target program to be running to get GDB to talk to it. It can be done before letting the simulator run or you use the ELF loader component to get a program into memory.

Note: Configuring eCos' HAL package for simulation should rarely be needed for real development; binaries built with such a kernel will not run on target boards at all, and the MN10300 and TX39 simulators can run binaries built for stdev1 and jmr3904 target boards. The main use for a "simulation" configuration is if you are trying to work around problems with the device drivers or with the simulator. Also note that when using a TX39 system configured for simulator start-up you should then invoke the simulator with the `--board=jmr3904pal` option instead of `--board=jmr3904`

Note: If your chosen architecture does not have simulator support, then the combinations above that refer to the simulator do not apply. Similarly, if your chosen platform does not have RedBoot ROM support, the combinations listed above that use RedBoot do not apply.

The debugging environment for most developers will be either a hardware board or the simulator, in which case they will be able to select a single HAL configuration.

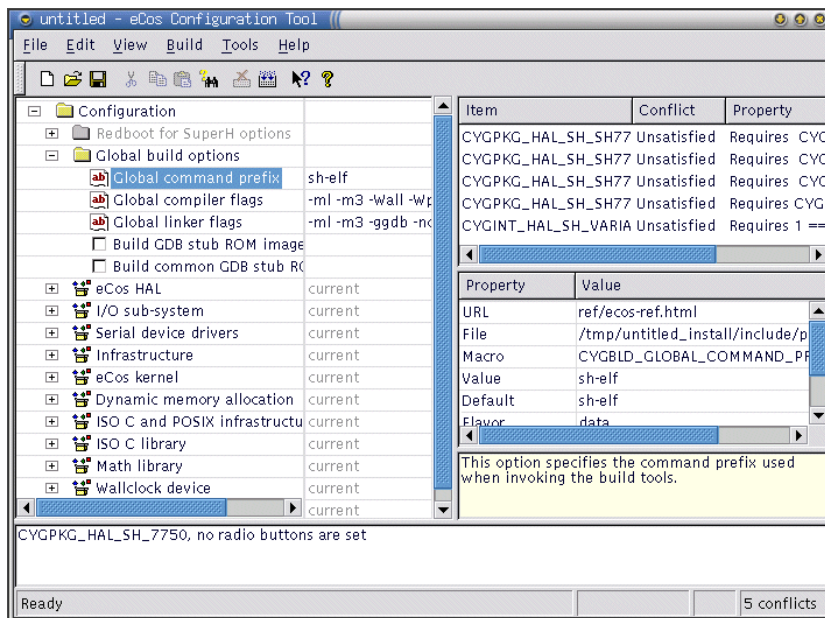
Configuration Tool on Windows and Linux Quick Start

This section describes the GUI based configuration tool. This tool is probably more suited to users who prefer GUI's. The next section describes a CLI based tool which Unix users may prefer.

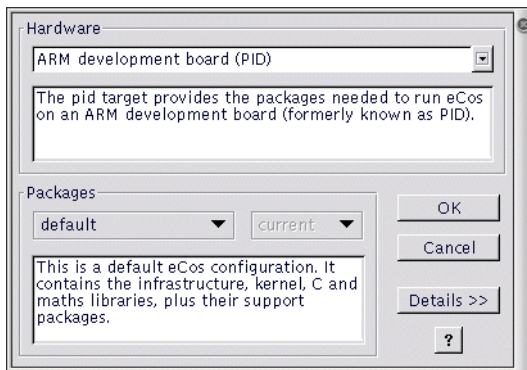
Note that the use of the Configuration Tool is described in detail in [Part IV in eCos User Guide](#).

The Configuration Tool (see [Figure 11-1](#)) has five main elements: the *configuration window*, the *conflicts window*, the *properties window*, the *short description window*, and the *output window*.

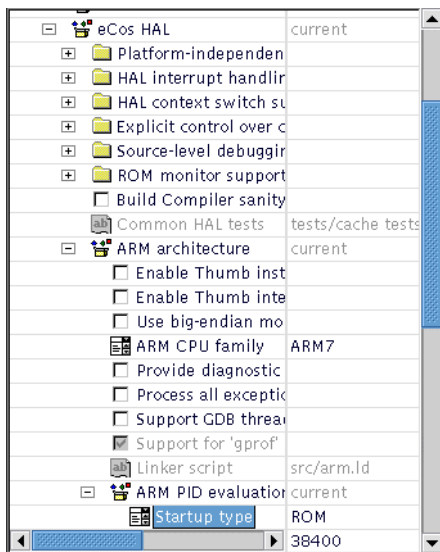
Figure 11-1. Configuration Tool



Start by opening the templates window via **Build->Templates**. Select the desired target (see [Figure 11-2](#)).

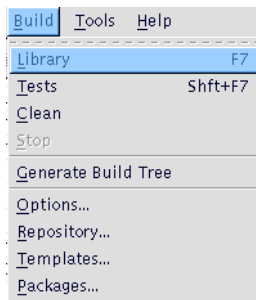
Figure 11-2. Template selection

Make sure that the configuration is correct for the target in terms of endianness, CPU model, Startup type, etc. (see [Figure 11-3](#)).

Figure 11-3. Configuring for the target

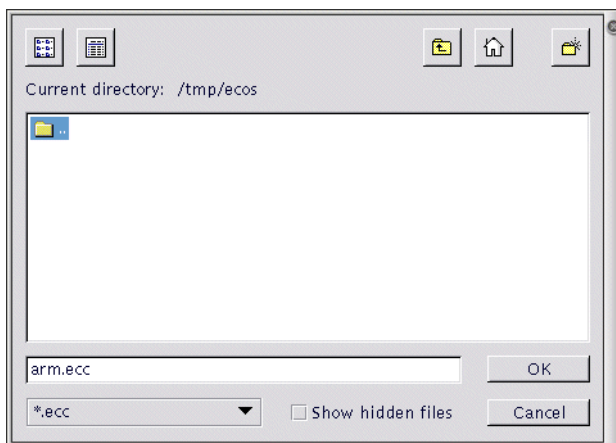
Next, select the *Build->Library* menu item to start building eCos (see [Figure 11-4](#)). The application will configure the sources, prepare a build tree, and build the `libtarget.a` library, which contains the eCos kernel and other packages.

Figure 11-4. Selecting the Build Library menu item

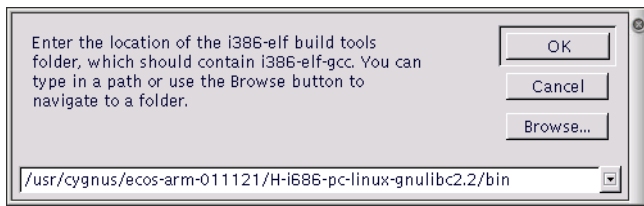


The *Save As* dialog box will appear, asking you to specify a directory in which to place your save file. You can use the default, but it is a good idea to make a subdirectory, called `ecos-work` for example.

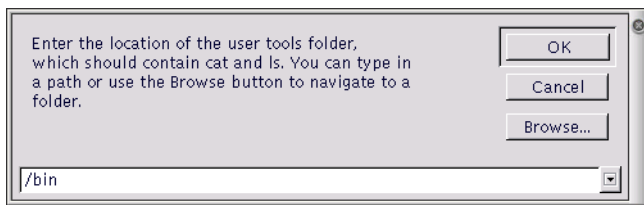
Figure 11-5. Save file dialog



The first time you build an eCos library for a specific architecture, the Configuration Tool may prompt you for the location of the appropriate build tools (including **make** and **TARGET-gcc**) using a *Build Tools* dialog box (as shown in [Figure 11-6](#)). You can select a location from the drop down list, browse to the directory using the *Browse* button, or type in the location of the build tools manually.

Figure 11-6. Build tools dialog

The Configuration Tool may also prompt you for the location of the user tools (such as **cat** and **ls**) using a *User Tools* dialog box (as shown in [Figure 11-7](#)). As with the *Build Tools* dialog, you can select a location from the drop down list, browse to the directory using the *Browse* button, or type in the location of the user tools manually. Note that on Linux, this will often be unnecessary as the tools will already be on your `PATH`.

Figure 11-7. User tools dialog

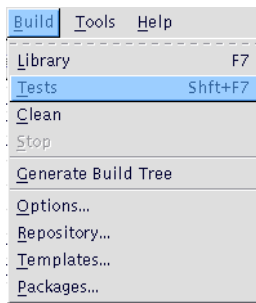
When the tool locations have been entered, the Configuration Tool will configure the sources, prepare a build tree, and build the `libtarget.a` library, which contains the eCos kernel and other packages.

The output from the configuration process and the building of `libtarget.a` will be shown in the output window.

Once the build process has finished you will have a kernel with other packages in `libtarget.a`. You should now build the eCos tests for your particular configuration.

You can do this by selecting *Build -> Tests*. Notice that you could have selected *Tests* instead of *Library* in the earlier step and it would have built *both* the library and the tests, but this would increase the build time substantially, and if you do not need to build the tests it is unnecessary.

Figure 11-8. Selecting the Build Tests menu item



[Chapter 12](#) will guide you through running one of the test cases you just built on the selected target, using GDB.

Ecosconfig on Windows and Linux Quick Start

As an alternative to using the graphical Configuration Tool, it is possible to configure and build a kernel by editing a configuration file manually and using the **ecosconfig** command. Users with a Unix background may find this tool more suitable than the GUI tool described in the previous section.

Manual configuration and the **ecosconfig** command are described in detail in [Chapter 28](#).

To use the **ecosconfig** command you need to start a shell. In Windows you need to start a CygWin **bash** shell, not a DOS command line.

The following instructions assume that the `PATH` and `ECOS_REPOSITORY` environment variables have been setup correctly as described in [Chapter 7](#). They also assume Linux usage but equally well apply to Windows running Cygwin.

Before invoking **ecosconfig** you need to choose a directory in which to work. For the purposes of this tutorial, the default path will be `BASE_DIR/ecos-work`. Create this directory and change to it by typing:

```
$ mkdir BASE_DIR/ecos-work
$ cd BASE_DIR/ecos-work
```

To see what options can be used with **ecosconfig**, type:

```
$ ecosconfig --help
```

The available packages, targets and templates may be listed as follows:

```
$ ecosconfig list
```

Here is sample output from **ecosconfig** showing the usage message.

Example 11-1. Getting help from ecosconfig

```
$ ecosconfig --help
```

```

Usage: ecosconfig [ qualifier ... ] [ command ]
  commands are:
    list                                : list repository contents
    new TARGET [ TEMPLATE [ VERSION ] ] : create a configuration
    target TARGET                        : change the target hardware
    template TEMPLATE [ VERSION ]      : change the template
    add PACKAGE [ PACKAGE ... ]         : add package(s)
    remove PACKAGE [ PACKAGE ... ]      : remove package(s)
    version VERSION PACKAGE [ PACKAGE ... ] : change version of package(s)
    export FILE                          : export minimal config info
    import FILE                          : import additional config info
    check                               : check the configuration
    resolve                             : resolve conflicts
    tree                                : create a build tree
  qualifiers are:
    --config=FILE                       : the configuration file
    --prefix=DIRECTORY                  : the install prefix
    --srcdir=DIRECTORY                  : the source repository
    --no-resolve                        : disable conflict
  resolution
    --version                           : show version and copyright
$

```

Example 11-2. ecosconfig output — list of available packages, targets and templates

```

$ ecosconfig list
Package CYGPKG_CYGMON (CygMon support via eCos):
aliases: cygmon
versions: <version>
Package CYGPKG_DEVICES_WALLCLOCK_DALLAS_DS1742 (Wallclock driver for Dallas 1742):
aliases: devices_wallclock_ds1742 device_wallclock_ds1742
versions: <version>
Package CYGPKG_DEVICES_WALLCLOCK_SH3 (Wallclock driver for SH3 RTC module):
aliases: devices_wallclock_sh3 device_wallclock_sh3
versions: <version>
Package CYGPKG_DEVICES_WATCHDOG_ARM_AEB (Watchdog driver for ARM/AEB board):
aliases: devices_watchdog_aeb device_watchdog_aeb
versions: <version>
Package CYGPKG_DEVICES_WATCHDOG_ARM_EBSA285 (Watchdog driver for ARM/EBSA285 board):
aliases: devices_watchdog_ebsa285 device_watchdog_ebsa285
versions: <version>
...

```

Selecting a Target

To configure for a listed target, type:

```
$ ecosconfig new <target>
```

For example, to configure for the ARM PID development board, type:

```
$ ecosconfig new pid
```

You can then edit the generated file, `ecos.ecc`, setting the options as required for the target (endianess, CPU model, Startup type, etc.). For detailed information about how to edit the `ecos.ecc` file, see the *CDL Writer's Guide* and [the Section called *Editing an eCos Savefile* in Chapter 28](#).

Create a build tree for the configured target by typing:

```
$ ecosconfig tree
```

If there are any problem with the configuration, **ecosconfig** will tell you. The most likely cause of this is mistakes when editing the `ecos.ecc` file. You can check whether the configuration you have made is correct, without building the tree with the following command:

```
$ ecosconfig check
```

If this reports any conflicts you can get **ecosconfig** to try and resolve them itself by typing:

```
$ ecosconfig resolve
```

See [the Section called *Conflicts and constraints* in Chapter 28](#) for more details.

You can now run the command **make** or **make tests**, after which you will be at the same point you would be after running the Configuration Tool — you can start developing your own applications, following the steps in [Chapter 13](#).

The procedure shown above allows you to do very coarse-grained configuration of the eCos kernel: you can select which packages to include in your kernel, and give target and start-up options. But you cannot select components within a package, or set the very fine-grained options.

To select fine-grained configuration options you will need to edit the configuration file `ecos.ecc` in the current directory and regenerate the build tree.

Caution

You should follow the manual configuration process described above very carefully, and you should read the comments in each file to see when one option depends on other options or packages being enabled or disabled. If you do not, you might end up with an inconsistently configured kernel which could fail to build or might execute incorrectly.

Chapter 12. Running an eCos Test Case

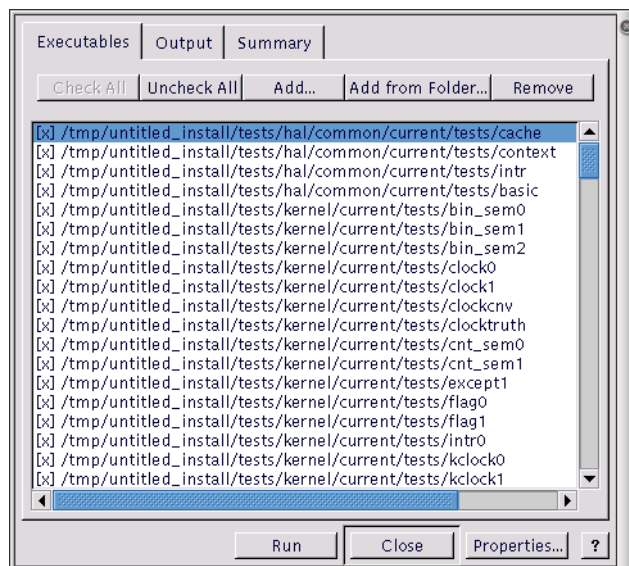
In the Section called *Configuration Tool on Windows and Linux Quick Start* in Chapter 11 or the Section called *Ecosconfig on Windows and Linux Quick Start* in Chapter 11 you created the eCos test cases as part of the build process. Now it is time to try and run one.

Using the Configuration Tool

Test executables that have been linked using the *Build->Tests* operation against the current configuration can be executed by selecting *Tools->Run Tests*.

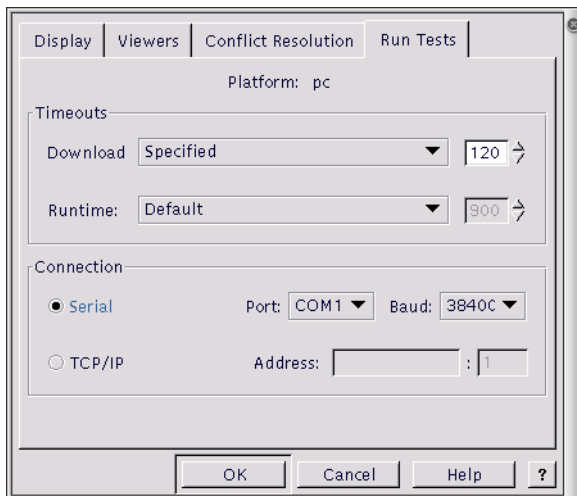
When a test run is invoked, a property sheet is displayed, see [Figure 12-1](#). Press the *Uncheck All* button and then find and check just one test, `bin_sem0` for example.

Figure 12-1. Run tests



Now press the *Properties* button to set up communications with the target. This will bring up a properties dialog shown in [Figure 12-2](#). If you have connected the target board via a serial cable, check the *Serial* radio button, and select the serial port and baud rate for the board. If the target is connected via the network select the *TCP/IP* button and enter the IP address that the board has been given, and the port number (usually 9000).

Figure 12-2. Properties dialog box



Click OK on this dialog and go back to the *Run Tests* dialog. Press the *Run* button and the selected test will be downloaded and run. The *Output* tab will show you how this is progressing. If it seems to stop for a long time, check that the target board is correctly connected, and that eCos has been correctly configured -- especially the start-up type.

When the program runs you should see a couple of line similar to this appear:

```
PASS:<Binary Semaphore 0 OK>
EXIT:<done>
```

This indicates that the test has run successfully.

See [Chapter 22](#) for further details.

Using the command line

Start a command shell (such as a Cygwin shell window in Windows) with the environment variables set as described in the toolchain documentation. Change to the directory in which you set up your build tree, and invoke GDB on the test program.

To run the `bin_sem0` test (which will test the kernel for the correct creation and destruction of binary semaphores) type:

```
$ TARGET-gdb -nw install/tests/kernel/<version>/tests/bin_sem0
```

You should see output similar to the following in the command window:

```
GNU gdb THIS-GDB-VERSION
Copyright 2001 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
```



```
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "--host=THIS-HOST --target=THIS-TARGET".
(gdb)
```

If you are trying to run a synthetic target test on Linux, skip the following connection and download steps. Otherwise, connect to the target by typing:

```
(gdb) set remotebaud 38400
(gdb) target remote /dev/ttyS0
```

on Linux or

```
(gdb) set remotebaud 38400
(gdb) target remote com1
```

on Windows or

```
(gdb) target sim
```

to use a simulator in either host O/S.

Check the documentation for the target board for the actual baud rate to use when connecting to real targets.

You will see output similar to the following:

```
Remote debugging using /dev/ttyS1
0x0000d50c in ?? ()
    at BASE_DIR/kernel/<version>/src/common/kapi.cxx:345

Current language:  auto; currently c++
(gdb)
```

Or if you are using the simulator:

```
Connected to the simulator.
(gdb)
```

Now download the program to the target with

```
(gdb) load
```

You should see output similar to the following on your screen:

```
Loading section .text, size 0x4b04 lma 0x108000
Loading section .rodata, size 0x738 lma 0x10cb08
Loading section .data, size 0x1c0 lma 0x10d240
Start address 0x108000, load size 21500
Transfer rate: 24571 bits/sec, 311 bytes/write.
(gdb)
```

You are now ready to run your program. If you type:

```
(gdb) continue
```

you will see output similar to the following:

```
Continuing.  
PASS:<Binary Semaphore 0 OK>  
EXIT:<done>
```

Note: If you are using a simulator or the synthetic target rather than real hardware, you must use the GDB command “run” rather than “continue” to start your program.

You can terminate your GDB session with *Control+C*, otherwise it will sit in the “idle” thread and use up CPU time. This is not a problem with real targets, but may have undesirable effects in simulated or synthetic targets. Type **quit** and you are done.

Testing Filters

While most test cases today run solely in the target environment, some packages may require external testing infrastructure and/or feedback from the external environment to do complete testing.

The serial package is an example of this. The network package also contains some tests that require programs to be run on a host. See the network *Tests and Demonstrations* section in the network documentation in the *eCos Reference Guide*. Here we will concentrate on the serial tests since these are applicable to more targets.

Since the serial line is also used for communication with GDB, a filter is inserted in the communication pathway between GDB and the serial device which is connected to the hardware target. The filter forwards all communication between the two, but also listens for special commands embedded in the data stream from the target.

When such a command is seen, the filter stops forwarding data to GDB from the target and enters a special mode. In this mode the test case running on the target is able to control the filter, commanding it to run various tests. While these tests run, GDB is isolated from the target.

As the test completes (or if the filter detects a target crash) the communication path between GDB and the hardware target is re-established, allowing GDB to resume control.

In theory, it is possible to extend the filter to provide a generic framework for other target-external testing components, thus decoupling the testing infrastructure from the (possibly limited) communication means provided by the target (serial, JTAG, Ethernet, etc).

Another advantage is that the host tools do not need to know about the various testing environments required by the eCos packages, since all contact with the target continues to happen via GDB.

Chapter 13. Building and Running Sample Applications

The example programs in this tutorial are included, along with a `Makefile`, in the `examples` directory of the eCos distribution. The first program you will run is a *hello world*-style application, then you will run a more complex application that demonstrates the creation of threads and the use of `cyg_thread_delay()`, and finally you will run one that uses clocks and alarm handlers.

The `Makefile` depends on an externally defined variable to find the eCos library and header files. This variable is `INSTALL_DIR` and must be set to the pathname of the install directory created in [the Section called *Configuration Tool on Windows and Linux Quick Start* in Chapter 11](#).

`INSTALL_DIR` may be either be set in the shell environment or may be supplied on the command line. To set it in the shell do the following in a **bash** shell:

```
$ export INSTALL_DIR=BASE_DIR/ecos-work/arm_install
```

You can then run **make** without any extra parameters to build the examples.

Alternatively, if you can do the following:

```
$ make INSTALL_DIR=BASE_DIR/ecos-work/arm_install
```

eCos Hello World

The following code is found in the file `hello.c` in the `examples` directory:

eCos hello world program listing

```
/* this is a simple hello world program */
#include <stdio.h>
int main(void)
{
    printf("Hello, eCos world!\n");
    return 0;
}
```

To compile this or any other program that is not part of the eCos distribution, you can follow the procedures described below. Type this explicit compilation command (assuming your current working directory is also where you built the eCos kernel):

```
$ TARGET-gcc -g -I$BASE_DIR/ecos-work/install/include hello.c -L$BASE_DIR/ecos-work/install/lib -Ttarget.ld
```

The compilation command above contains some standard GCC options (for example, `-g` enables debugging), as well as some mention of paths (`-I$BASE_DIR/ecos-work/install/include` allows files like `cyg/kernel/kapi.h` to be found, and `-L$BASE_DIR/ecos-work/install/lib` allows the linker to find `-Ttarget.ld`).

The executable program will be called `a.out`.

Note: Some target systems require special options to be passed to gcc to compile correctly for that system. Please examine the Makefile in the examples directory to see if this applies to your target.

You can now run the resulting program using GDB in exactly the same the way you ran the test case before. The procedure will be the same, but this time run **TARGET-gdb** specifying `-nw a.out` on the command line:

```
$ TARGET-gdb -nw a.out
```

For targets other than the synthetic linux target, you should now run the usual GDB commands described earlier. Once this is done, typing the command "continue" at the (gdb) prompt ("run" for simulators) will allow the program to execute and print the string "Hello, eCos world!" on your screen.

On the synthetic linux target, you may use the "run" command immediately - you do not need to connect to the target, nor use the "load" command.

A Sample Program with Two Threads

Below is a program that uses some of eCos' system calls. It creates two threads, each of which goes into an infinite loop in which it sleeps for a while (using `cyg_thread_delay()`). This code is found in the file `twothreads.c` in the examples directory.

eCos two-threaded program listing

```
#include <cyg/kernel/kapi.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

/* now declare (and allocate space for) some kernel objects,
   like the two threads we will use */
cyg_thread thread_s[2]; /* space for two thread objects */

char stack[2][4096]; /* space for two 4K stacks */

/* now the handles for the threads */
cyg_handle_t simple_threadA, simple_threadB;

/* and now variables for the procedure which is the thread */
cyg_thread_entry_t simple_program;

/* and now a mutex to protect calls to the C library */
cyg_mutex_t cliblock;

/* we install our own startup routine which sets up threads */
void cyg_user_start(void)
{
    printf("Entering twothreads' cyg_user_start() function\n");
```

```

cyg_mutex_init(&cliblock);

cyg_thread_create(4, simple_program, (cyg_addrword_t) 0,
"Thread A", (void *) stack[0], 4096,
&simple_threadA, &thread_s[0]);
cyg_thread_create(4, simple_program, (cyg_addrword_t) 1,
"Thread B", (void *) stack[1], 4096,
&simple_threadB, &thread_s[1]);

cyg_thread_resume(simple_threadA);
cyg_thread_resume(simple_threadB);
}

/* this is a simple program which runs in a thread */
void simple_program(cyg_addrword_t data)
{
    int message = (int) data;
    int delay;

    printf("Beginning execution; thread data is %d\n", message);

    cyg_thread_delay(200);

    for (;;) {
        delay = 200 + (rand() % 50);

        /* note: printf() must be protected by a
        call to cyg_mutex_lock() */
        cyg_mutex_lock(&cliblock); {
            printf("Thread %d: and now a delay of %d clock ticks\n",
            message, delay);
        }
        cyg_mutex_unlock(&cliblock);
        cyg_thread_delay(delay);
    }
}

```

When you run the program (by typing **continue** at the (*gdb*) prompt) the output should look like this:

```

Starting program: BASE_DIR/examples/twothreads.exe
Entering twothreads' cyg_user_start()
function
Beginning execution; thread data is 0
Beginning execution; thread data is 1
Thread 0: and now a delay of 240 clock ticks
Thread 1: and now a delay of 225 clock ticks
Thread 1: and now a delay of 234 clock ticks
Thread 0: and now a delay of 231 clock ticks
Thread 1: and now a delay of 224 clock ticks
Thread 0: and now a delay of 249 clock ticks
Thread 1: and now a delay of 202 clock ticks
Thread 0: and now a delay of 235 clock ticks

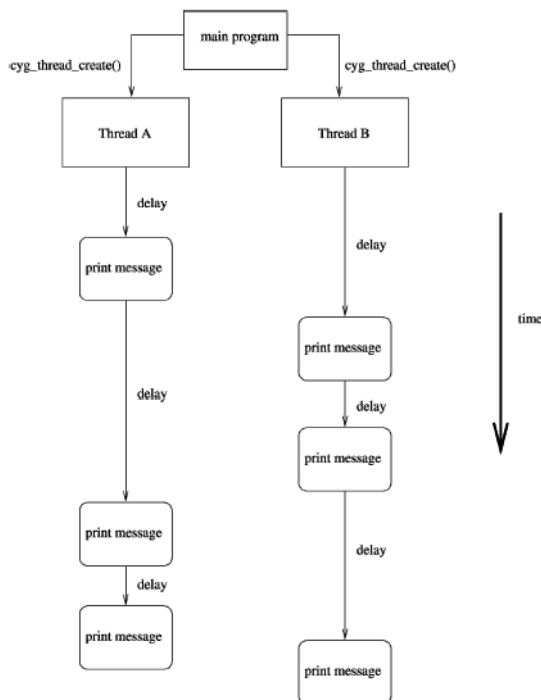
```

Note: When running in a simulator the delays might be quite long. On a hardware board (where the clock speed is 100 ticks/second) the delays should average to about 2.25 seconds. In simulation, the delay will depend on the speed of the host processor and will almost always be much slower than the actual board. You might want to reduce the delay parameter when running in simulation.

Figure 13-1 shows how this multitasking program executes. Note that apart from the thread creation system calls, this program also creates and uses a *mutex* for synchronization between the `printf()` calls in the two threads. This is because the C library standard I/O (by default) is configured not to be thread-safe, which means that if more than one thread is using standard I/O they might corrupt each other. This is fixed by a mutual exclusion (or *mutex*) lockout mechanism: the threads do not call `printf()` until `cyg_mutex_lock()` has returned, which only happens when the other thread calls `cyg_mutex_unlock()`.

You could avoid using the mutex by configuring the C library to be thread-safe (by selecting the component `CYGSEM_LIBC_STDIO_THREAD_SAFE_STREAMS`).

Figure 13-1. Two threads with simple print statements after random delays



Chapter 14. More Features — Clocks and Alarm Handlers

If a program wanted to execute a task at a given time, or periodically, it could do it in an inefficient way by sitting in a loop and checking the real-time clock to see if the proper amount of time has elapsed. But operating systems usually provide system calls which allow the program to be informed at the desired time.

eCos provides a rich timekeeping formalism, involving *counters*, *clocks*, *alarms*, and *timers*. The precise definition, relationship, and motivation of these features is beyond the scope of this tutorial, but these examples illustrate how to set up basic periodic tasks.

Alarms are events that happen at a given time, either once or periodically. A thread associates an alarm handling function with the alarm, so that the function will be invoked every time the alarm “goes off”.

A Sample Program with Alarms

`simple-alarm.c` (in the examples directory) is a short program that creates a thread that creates an alarm. The alarm is handled by the function `test_alarm_func()`, which sets a global variable. When the main thread of execution sees that the variable has changed, it prints a message.

Example 14-1. A sample program that creates an alarm

```
/* this is a very simple program meant to demonstrate
   a basic use of time, alarms and alarm-handling functions in eCos */

#include <cyg/kernel/kapi.h>

#include <stdio.h>

#define NTHREADS 1
#define STACKSIZE 4096

static cyg_handle_t thread[NTHREADS];

static cyg_thread thread_obj[NTHREADS];
static char stack[NTHREADS][STACKSIZE];

static void alarm_prog( cyg_addrword_t data );

/* we install our own startup routine which sets up
   threads and starts the scheduler */
void cyg_user_start(void)
{
    cyg_thread_create(4, alarm_prog, (cyg_addrword_t) 0,
        "alarm_thread", (void *) stack[0],
        STACKSIZE, &thread[0], &thread_obj[0]);
    cyg_thread_resume(thread[0]);
}

/* we need to declare the alarm handling function (which is
```

```

    defined below), so that we can pass it to  cyg_alarm_initialize() */
    cyg_alarm_t test_alarm_func;

/* alarm_prog() is a thread which sets up an alarm which is then
   handled by test_alarm_func() */
static void alarm_prog(cyg_addrword_t data)
{
    cyg_handle_t test_counterH, system_clockH, test_alarmH;
    cyg_tick_count_t ticks;
    cyg_alarm test_alarm;
    unsigned how_many_alarms = 0, prev_alarms = 0, tmp_how_many;

    system_clockH = cyg_real_time_clock();
    cyg_clock_to_counter(system_clockH, &test_counterH);
    cyg_alarm_create(test_counterH, test_alarm_func,
        (cyg_addrword_t) &how_many_alarms,
        &test_alarmH, &test_alarm);
    cyg_alarm_initialize(test_alarmH, cyg_current_time()+200, 200);

/* get in a loop in which we read the current time and
   print it out, just to have something scrolling by */
    for (;;) {
        ticks = cyg_current_time();
        printf("Time is %llu\n", ticks);
        /* note that we must lock access to how_many_alarms, since the
           alarm handler might change it. this involves using the
           annoying temporary variable tmp_how_many so that I can keep the
           critical region short */
        cyg_scheduler_lock();
        tmp_how_many = how_many_alarms;
        cyg_scheduler_unlock();
        if (prev_alarms != tmp_how_many) {
            printf(" --- alarm calls so far: %u\n", tmp_how_many);
            prev_alarms = tmp_how_many;
        }
        cyg_thread_delay(30);
    }
}

/* test_alarm_func() is invoked as an alarm handler, so
   it should be quick and simple. in this case it increments
   the data that is passed to it. */
void test_alarm_func(cyg_handle_t alarmH, cyg_addrword_t data)
{
    ++*((unsigned *) data);
}

```

When you run this program (by typing **continue** at the (*gdb*) prompt) the output should look like this:

```

Starting program: BASE_DIR/examples/simple-alarm.exe
Time is 0
Time is 30
Time is 60
Time is 90
Time is 120

```



```

Time is 150
Time is 180
Time is 210
    --- alarm calls so far: 1
Time is 240
Time is 270
Time is 300
Time is 330
Time is 360
Time is 390
Time is 420
    --- alarm calls so far: 2
Time is 450
Time is 480

```

Note: When running in a simulator the delays might be quite long. On a hardware board (where the clock speed is 100 ticks/second) the delays should average to about 0.3 seconds (and 2 seconds between alarms). In simulation, the delay will depend on the speed of the host processor and will almost always be much slower than the actual board. You might want to reduce the delay parameter when running in simulation.

Here are a few things you might notice about this program:

- It used the `cyg_real_time_clock()` function; this always returns a handle to the default system real-time clock.
- Clocks are based on counters, so the function `cyg_alarm_create()` uses a counter handle. The program used the function `cyg_clock_to_counter()` to strip the clock handle to the underlying counter handle.
- Once the alarm is created it is initialized with `cyg_alarm_initialize()`, which sets the time at which the alarm should go off, as well as the period for repeating alarms. It is set to go off at the current time and then to repeat every 200 ticks.
- The alarm handler function `test_alarm_func()` conforms to the guidelines for writing alarm handlers and other delayed service routines: it does not invoke any functions which might lock the scheduler. This is discussed in detail in the *eCos Reference Manual*, in the chapter *The eCos Kernel*.
- There is a *critical region* in this program: the variable `how_many_alarms` is accessed in the main thread of control and is also modified in the alarm handler. To prevent a possible (though unlikely) race condition on this variable, access to `how_many_alarms` in the principal thread is protected by calls to `cyg_scheduler_lock()` and `cyg_scheduler_unlock()`. When the scheduler is locked, the alarm handler will not be invoked, so the problem is averted.

IV. The eCos Configuration Tool

Chapter 15. Getting Started

Introduction

The eCos Configuration Tool is used to tailor eCos at source level, prior to compilation or assembly, and provides a configuration file and a set of files used to build user applications. The sources and other files used for building a configuration are provided in a *component repository*, which is loaded when the eCos Configuration Tool is invoked. The component repository includes a set of files defining the structure of relationships between the Configuration Tool and other components, and is written in a *Component Definition Language* (CDL). For a description of the concepts underlying component configuration, see [Chapter 24](#).

Invoking the eCos Configuration Tool

On Linux

Add the eCos Configuration Tool install directory to your PATH, for example:

```
export PATH=/opt/ecos/ecos<version>/bin:$PATH
```

You may run configtool with zero, one or two arguments. You can specify the eCos repository location, and/or an eCos save file (extension .ecc) on the command line. The ordering of these two arguments is not significant. For example:

```
configtool /opt/ecos/ecos<version>/packages myfile.ecc
```

The Configuration Tool will be displayed (see [Figure 15-1](#)).

On Windows

There are two ways in which to invoke the eCos Configuration Tool:

- from the desktop explorer or program set up at installation time (by default *Start -> Programs -> eCos -> Configuration Tool*).
- type (at a command prompt or in the *Start* menu's *Run* item): <foldername>\ConfigTool.exe where <foldername> is the full path of the directory in which you installed the eCos Configuration Tool.
- The Configuration Tool will be displayed (see [Figure 15-1](#)).

You may run configtool with zero, one or two arguments. You can specify the eCos repository location, and/or an eCos save file (extension .ecc) on the command line. The ordering of these two arguments is not significant. For example:

```
configtool "c:\Program Files\eCos\packages" myfile.ecc
```

If you invoke the configuration tool from the command line with *--help*, you will see this output:

```
Usage: eCos Configuration Tool [-h] [-e] [-v] [-c] [input file 1] [input file 2]
  -h --help          displays help on the command line parameters
  -e --edit-only      edit save file only
  -v --version        print version
  -c --compile-help  compile online help only
```

This summarizes valid parameters and switches. Switches are shown with both short form and long form.

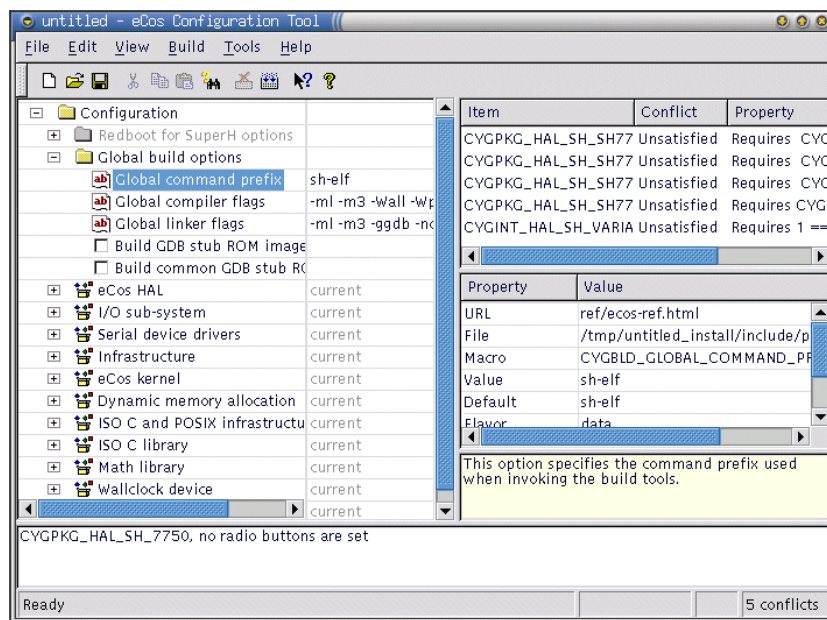
`--help` shows valid options and parameters, as above.

`--edit-only` runs the Configuration Tool in a mode that suppresses creation of a build tree, in case you only want to create and edit save files.

`--version` shows version and build date information, and exits.

`--compile-help` compiles help contents files from the HTML documentation files that the tool finds in the eCos repository, and exits.

Figure 15-1. Configuration Tool



The Component Repository

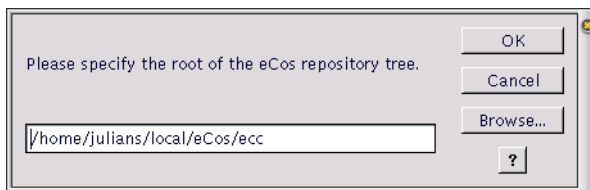
When you invoke the eCos Configuration Tool, it accesses the Component Repository, a read-only location of configuration information. For an explanation of “Component Repository” see [Chapter 24](#).

The eCos Configuration Tool will look for a component repository using (in descending order of preference):

- A location specified on the command line
- The component repository most recently used by the current user
- An eCos distribution under `/opt/ecos` (under Linux) or a default location set by the installation procedure (under Windows)
- User input

The final case above will normally only occur if the previous repository has been moved or (under Windows) installation information stored in the Windows registry has been modified; it will result in a dialog box being displayed that allows you to specify the repository location:

Figure 15-2. Repository relocation dialog box



Note that in order to use the eCos Configuration Tool you are obliged to provide a valid repository location.

In the rare event that you subsequently wish to change the component location, select *Build->Repository* and the above dialog box will then be displayed.

You can check the location of the current repository, the current save file path, and the current hardware template and default package, by selecting *Help->Repository Information....* A summary will be displayed.

eCos Configuration Tool Documents

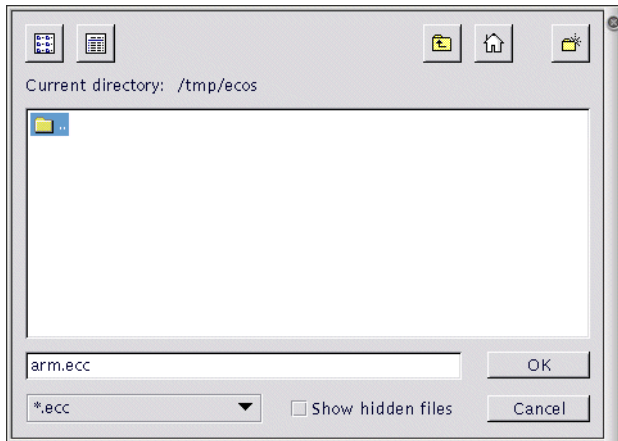
Configuration Save File

eCos configuration settings and other information (such as disabled conflicts) that are set using the eCos Configuration Tool are saved to a file between sessions. By default, when the eCos Configuration Tool is first invoked, it reads and displays information from the Component Registry and displays the information in an untitled blank document. You can perform the following operations on a document:

Save the currently active document

Use the *File->Save* menu item or click the *Save Document* icon on the toolbar; if the current document is unnamed, you will be prompted to supply a name for the configuration save file.

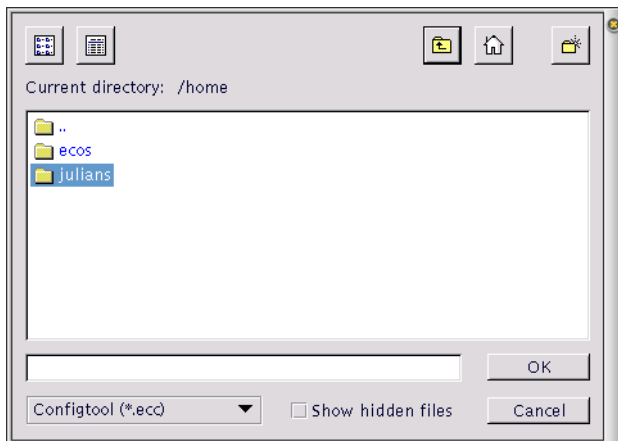
Figure 15-3. Save As dialog box



Open an existing document

Select *File->Open*, or click the *Open Document* icon on the toolbar. You will be prompted to supply a name for the configuration save file.

Figure 15-4. Open dialog box



Open a document you have used recently

Click its name at the bottom of the *File* menu.

Documents may also be opened by:

- double-clicking a Configuration Save File in the desktop explorer (Windows only);
- invoking the eCos Configuration Tool with the name of a Configuration File as command-line argument, or by creating a shortcut to the eCos Configuration Tool with such an argument (under Windows or a suitable Linux desktop environment).

Create a new blank document based on the Component Registry

Select *File->New*, or click the *New Document* icon on the toolbar.

Save to a different file name

Select *File->Save As*. You will be prompted to supply a new name for the configuration save file.

Build and Install Trees

The location of the build and install trees are derived from the eCos save file name as illustrated in the following example:

Save file name = “c:\My eCos\config1.ecc”

Install tree folder = “c:\My eCos\config1_install”

Build tree folder = “c:\My eCos\config1_build”

These names are automatically generated from the name of the save file.

See also [Chapter 24](#).

Chapter 16. Getting Help

The eCos Configuration Tool contains several methods for accessing online help.

Context-sensitive Help for Dialogs

Most dialogs displayed by the eCos Configuration Tool are supplied with context-sensitive help. You can then get help relating to any control within the current dialog box by

- Right-clicking the control (or pressing *F1*)

A “What’s This?” popup menu will be displayed. Click the menu to display a brief description of the function of the selected control.

- Clicking the question mark icon in the dialog caption bar (Windows) or the question mark button on the dialog (Linux).

A question mark cursor will be displayed. Click on any control to display a brief description of its function.

Some dialogs may have a *Help* button. You can press this to display a more general description of the function of the dialog box as a whole. This help will be in HTML form; for more information, see below.

Context-sensitive Help for Other Windows

In the *Help* menu, click *Help On...* and then click on a window (or click on the arrow/question mark button on the toolbar, then click on a window). A small popup window page describing the window will be displayed. The same thing can be achieved by right-clicking on a window and clicking on *What’s This?*.

Context-sensitive Help for Configuration Items

In the configuration window, right-click on a configuration item (or use *Shift+F10*). A context menu will be displayed; select *Visit Documentation* to display the page in the eCos documentation that most closely corresponds to the selected item.

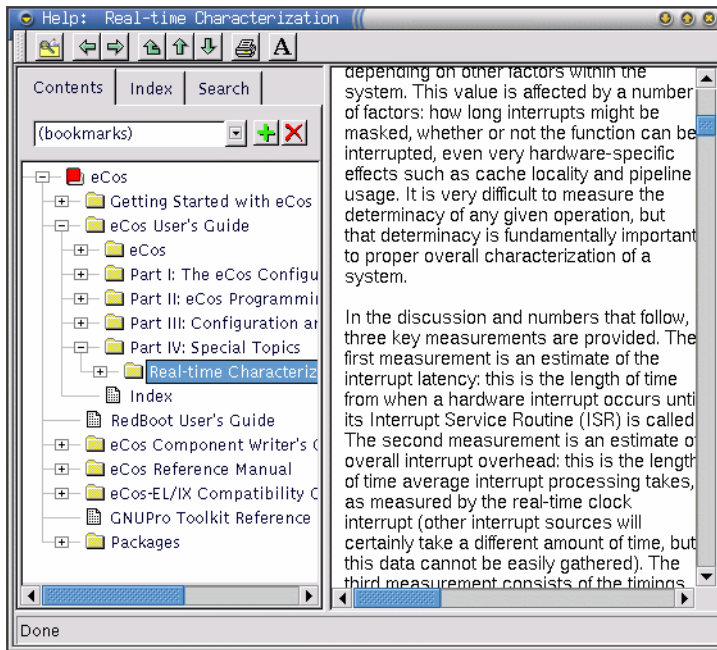
Methods of Displaying HTML Help

1. Using the internal help system. This will show an internal viewer similar to Microsoft HTML Help, with a contents hierarchy on the left and HTML pages on the right; see [Figure 16-1](#). The index is regenerated for each repository. If the documentation in the repository has changed but the contents does not reflect this, please use the Tools Regenerate Help Index menu item.
2. Using the default HTML browser. On Unix, you will need a .mailcap entry similar to this:

```
text/html; netscape -no-about-splash %s
```

3. Using the specified browser.

Figure 16-1. HTML Help viewer



If you wish, you may choose to have *HTML Help* displayed in a browser of your choice. To do this, select *View->Settings* and use the controls in the View Documentation group to select the replacement browser. Note that the Navigation facilities of the built-in *HTML Help* system will be unavailable if you choose this method of displaying help.

Chapter 17. Customization

The following visual aspects of the eCos Configuration Tool can be changed to suit individual preferences. These aspects are saved on a per-user basis, so that when the eCos Configuration Tool is next invoked by the same user, the appearance will be as set in the previous session.

Window Placement

The relative sizes of all windows in the eCos Configuration Tool may be adjusted by dragging the splitter bars that separate the windows. The chosen sizes will be used the next time the eCos Configuration Tool is invoked by the current user.

All windows except the *Configuration Window* may be shown or hidden by using the commands under the *View* menu (for example, *View->Output*) or the corresponding keyboard accelerators (*Alt+1* to *Alt+4*).

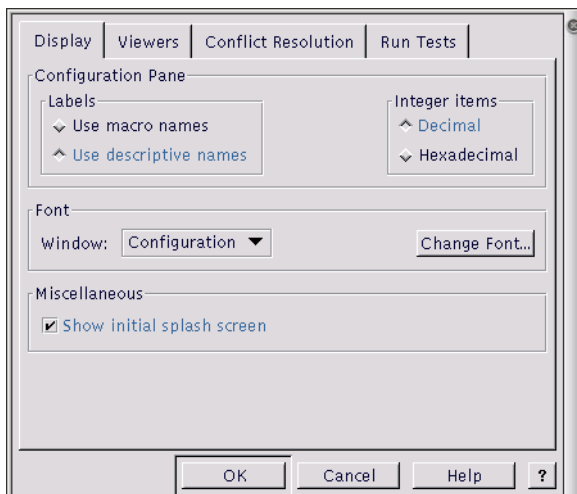
Your chosen set of windows (and their relative sizes) will be preserved between invocations of the eCos Configuration Tool.

Settings

To change other visual aspects, select *View->Settings* and then select the *Display* and *View* tabs depending on the settings you wish to alter. The options are as follows:

Settings: Display tab

Figure 17-1. Settings dialog, Display tab



Labels

In the configuration window, you can choose to have either *descriptive names* (the default) or *macro names* displayed as tree item labels. Descriptive names are generally more comprehensible, but macro names are used in some contexts such as conflict resolution and may be directly related to the source code of the configuration. Note that it is possible to search for an item in the configuration view by selecting *Find->Edit* (see [Chapter 20](#)). Both descriptive names and macro names can be searched.

Integer Items

You can choose to have integer items in the Configuration Window displayed in decimal or hexadecimal format.

Font

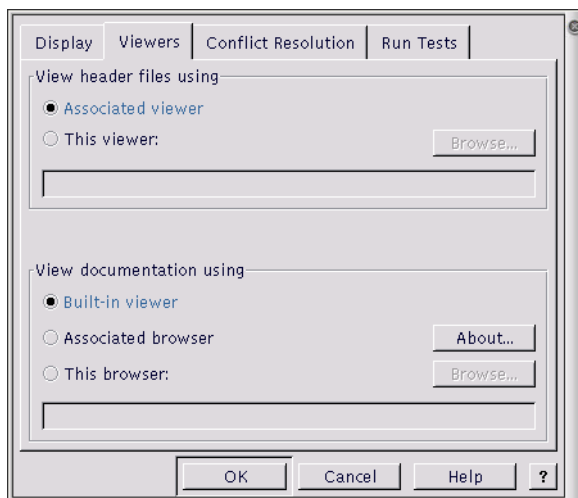
Change the font for a particular window by selecting the window name using the drop-down list, then clicking on *Change Font* to select a font for that window. The changes will be applied when the press *OK* to dismiss the Settings dialog. If you never make font changes, then the windows will take the default setting determined by your current Windows or Unix environment.

Miscellaneous

If the *Splash Screen* checkbox is checked, a *splash* window will appear as the application is loading. Uncheck this to eliminate the splash screen.

Settings: Viewers tab

Figure 17-2. Settings dialog, Viewers tab



View header files

You can change the viewer used to display header files.

View documentation

You can change the viewer used to display HTML files. See [the Section called *Methods of Displaying HTML Help* in Chapter 16](#).

Chapter 18. Screen Layout

The following windows are available within the eCos Configuration Tool:

- Configuration Window
- Properties Window
- Short Description
- Conflicts
- Output


The layout of the windows may be adjusted to suit your preferences: see [the Section called *Settings* in Chapter 17](#).

Configuration Window

This is the principal window used to configure eCos. It takes the form of a tree-based representation of the configuration items within the currently loaded eCos packages.

In the case of items whose values may be changed, controls are available to set the item values. These either take the form of check boxes or radio buttons within the tree itself or cells to the right of the thin vertical splitter bar. Controls in the tree may be used in the usual way; cells, however, must first be activated.

To activate a cell, simply click on it: it will assume a sunken appearance and data can then be edited in the cell. To terminate in-cell editing, click elsewhere in the configuration window or press *ENTER*. To discard the partial results of in-cell editing and revert to the previous value, press *ESCAPE*.

[-] Configuration	
[+] Global build options	
[+] Redboot HAL options	
[+] Intel 82559 ethernet driver	current
[+] PC board ethernet driver	current
[+] eCos HAL	current
[-] I/O sub-system	current
[-] Debug I/O sub-system	
[+] <input checked="" type="checkbox"/> Basic support for file based I/O	
[+] PCI configuration library	current
[+] Serial device drivers	current
[+] Infrastructure	current
[+] eCos kernel	current
[-] Dynamic memory allocation	current
[+] Memory allocator implementatio	
<input checked="" type="checkbox"/> Kernel C API support for memory	
<input type="checkbox"/> malloc(0) returns NULL	
<input checked="" type="checkbox"/> malloc(0) and supporting allocato	
Size of the fallback dynamic me	16384
[+] Common memory allocator pack	
[+] ISO C and POSIX infrastructure	current
[-] 	current

Cells come in three varieties, according to the type of data they accept:

Table 18-1. Cell types

Cell Type	Data Accepted
Integer	Decimal or hexadecimal values
Floating Point	Floating point values
String	Any

In the case of string cells, you can double-click the cell to display a dialog box containing a larger region in which to edit the string value. This is useful in the case of long strings, or those spanning multiple lines.

Disabled items

Some items will appear disabled. In this case the item label and any associated controls and cells will be grayed. It is not possible to change the values of disabled items.

Right-Clicking

You can right-click on an item in the configuration window item to display a pop-up menu which (depending on the type of the item selected) allows you to:

- *Properties* – information relating to the currently selected item is displayed. The information is equivalent to that displayed in the Properties Window.
- *Restore Defaults* - the default value of the currently selected item is restored.
- *Visit Documentation* - causes the HTML page most closely relating to the currently selected item to be displayed. This has the same effect as double-clicking the URL property in the Properties Window.
- *View Header File* – this causes the file containing the items to be displayed. This is equivalent to double-clicking on the File property in the Properties Window. The viewer used for this purpose may be changed using the *View->Settings* menu item (see [the Section called Settings in Chapter 17](#)). Note that this operation is only possible when the current configuration is saved, in order to avoid the possibility of changing the source repository.
- *Unload Package* - this is equivalent to using the *Build->Packages* menu item to select and unload the package in question.

Conflicts Window

This window exists to display any configuration item conflicts. Conflicts are the result of failures to meet the requirements between configuration items expressed in the CDL. See [the Section called Conflicts in Chapter 24](#).

Item	Conflict	Property
CYGPKG_KERNEL_SMP_SUPPORT	Unsatisfied	Requires CYGPKG_HAL_SMP_SUPPORT
CYGPKG_HAL_EXCEPTIONS	Unsatisfied	Requires CYGPKG_KERNEL_EXCEPTIONS

The window comprises three columns:

- *Item*

This is the macro name of the first item involved in the conflict.

- *Conflict*

This is a description of the conflict type. The currently supported types are “unresolved”, “illegal value”, “evaluation exception”, “goal unsatisfied” and “bad data”.

- *Property*

This contains a description of the configuration item’s property that caused the conflict.

Within the conflicts window you can right-click on any item to display a context menu which allows you to choose from one of the following options:

To locate the item involved in the conflict, double-click in the first or third column, or right-click over the item and choose *Locate* from the popup menu.

You can use the *Tools->Resolve Conflicts* menu item, or right-click over the item and select *Resolve* from the popup menu, to resolve conflicts — [the Section called *Resolving conflicts* in Chapter 19](#).

Output Window

This window displays any output generated by execution of external tools and any error messages that are not suitable for display in other forms (for example, as message boxes).

Within the output window you can right-click to display a context menu which allows you to:

- Save the contents of the window to a file
- Clear the contents of the window

Properties Window

This window displays the CDL properties of the item currently selected in the configuration window. The same information may be displayed by right-clicking the item and selecting “properties”.

Property	Value
URL	redirect/the-ecos-hardware-abstraction-layer-hal.html
File	/tmp/untitled_install/include/pkgconf/hal_i386.h
Macro	CYGPKG_HAL_I386
Value	current
Default	current
Parent	CYGPKG_HAL
Hardware	
IncludeDir	cyg/hal
DefineHeader	hal_i386.h
Compile	hal_misc.c context.S i386_stub.c hal_syscall.c
Make	<PREFIX>/lib/target.ld: <PACKAGE>/src/i386.ld \$(CC) -E -P -\

Two properties may be double-clicked as follows:

- *URL* – double-clicking on a URL property causes the referenced HTML page to be displayed. This has the same effect as right-clicking on the item and choosing “Visit Documentation”.
- *File* – double-clicking on a File property in a saved configuration causes the File to be displayed. The viewer used for this purpose may be changed using the *View->Settings* menu item. Note that this operation is only possible when the current configuration is saved, in order to avoid the possibility of changing the source repository.

Short Description Window

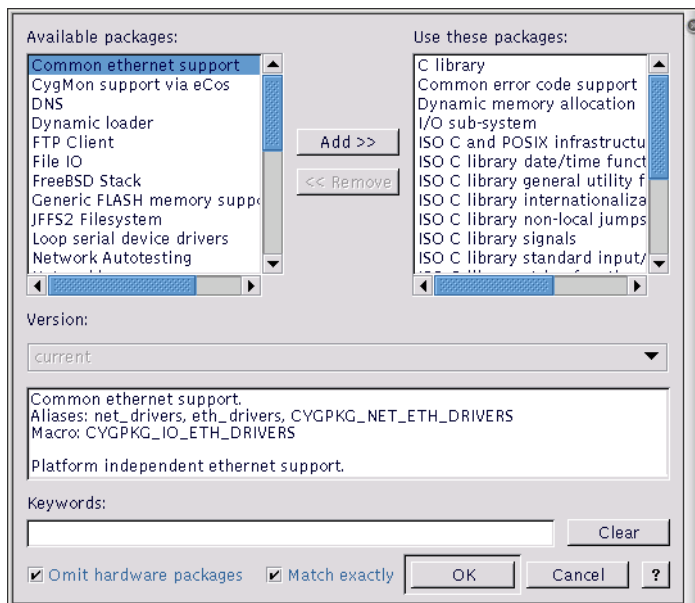
This window displays a short description of the item currently selected in the configuration window. More extensive documentation may be available by right-clicking on the item and choosing “Visit Documentation”.

Chapter 19. Updating the Configuration

Adding and Removing Packages

To add or remove packages from the configuration, select *Build->Packages*. The following dialog box will be displayed:

Figure 19-1. Packages dialog box



The left-hand list shows those packages that are available to be loaded. The right-hand list shows those that are currently loaded. In order to transfer packages from one list to another (that is, to load or unload packages) double-click the selection or click the *Add* or *Remove* buttons.

The version drop-down list displays the versions of the selected packages. When loading packages, this control may be used to load versions other than the most recent (current). Note that if more than one package is selected, the version drop-down list will display only the versions common to all the selected packages.

The window under the version displays a brief description of the selected package. If more than one package is selected, this window will be blank.

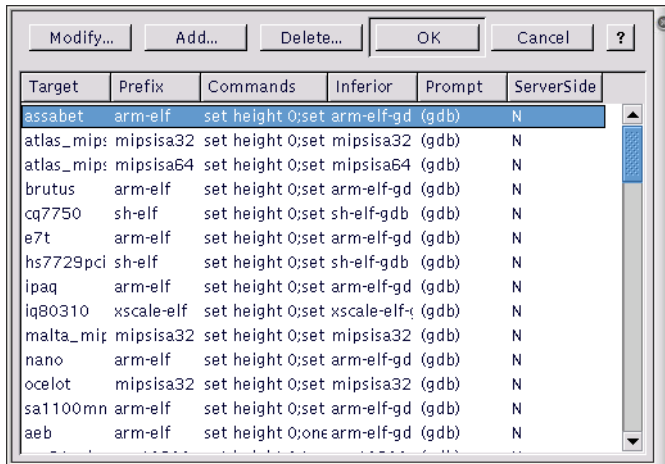
Under the description window there is a *Keywords* control into which you can type a string to be matched against package names, macro names and descriptions. The lists are updated a second or so after typing has stopped. If you type several separate words, all of these words must be associated with a given package for that package to be displayed. If you select the *Match exactly* checkbox, then the string is taken to be a complete fragment and matched against the beginning of a name, macro name or descriptions. All matches are done case-insensitively.

If you check *Omit hardware packages*, only non-hardware packages will be shown.

Platform Selection

To add, modify or remove entries in the list of platforms used for running tests, select *Tools->Platforms*. The following dialog will be displayed:

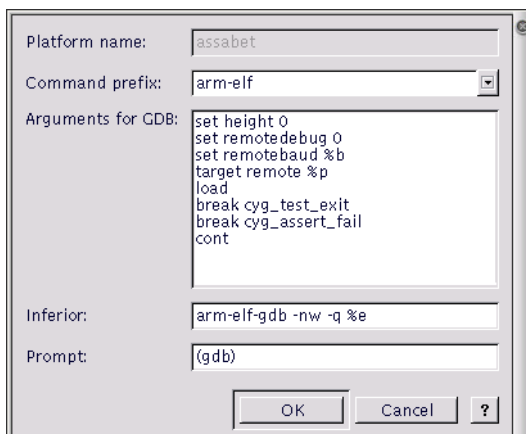
Figure 19-2. Platforms dialog box



You may add, modify or remove platform entries as you wish, but in order to run tests, a platform must be defined to correspond to the currently loaded hardware template. The information associated with each platform name is used to run tests.

To modify a platform, click the *Modify* button with the appropriate platform selected, or double-click on an entry in the list. A dialog will be displayed that allows you to change the command prefix, platform type and arguments for *GDB*.

Figure 19-3. Platform Modify dialog box



To add a new platform, click the *Add* button. A similar dialog will be displayed that allows you to define a new platform. To remove a platform, click the *Delete* button or press the *DEL* key with the appropriate platform selected.

The command prefix is used when running tests in order to determine the names of the executables (such as *gdb*) to be used. For example, if the *gdb* executable name is “*arm-elf-gdb.exe*” the prefix should be set to “*arm-elf*”.

The platform type indicates the capabilities of the platform - whether it is hardware or a simulator, and whether breakpoints are supported.

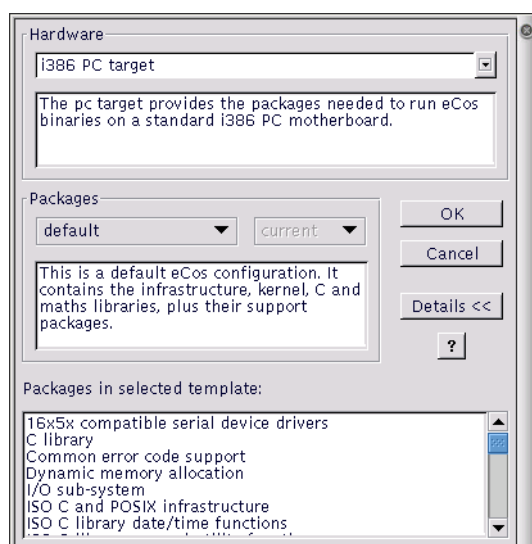
The arguments for the *GDB* field allow additional arguments to be passed to *gdb* when it is used to run a test. This is typically used in the case of simulators linked to *gdb* in order to define memory layout.

Using Templates

To load a configuration based on a template, select *Build->Templates*.

The following dialog box will be displayed:

Figure 19-4. Templates dialog box



Change the hardware template, the packages template, or both. To select a hardware template, choose from the first drop-list. To choose a packages template, choose from the second. Brief descriptions of each kind of template are provided in the corresponding edit boxes.

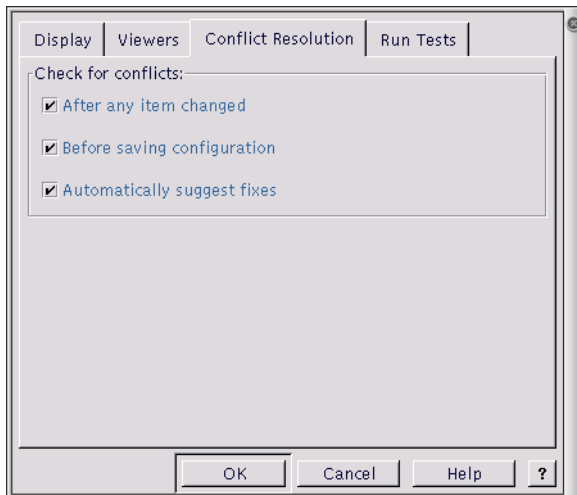
Resolving conflicts

During the process of configuring eCos it is possible that conflicts will be created. For more details of the meaning of conflicts, see [Chapter 24](#).

The Conflicts Window displays all conflicts in the current configuration. Additionally, a window in the status bar displays a count of the conflicts. Because the resolution of conflicts can be time-consuming, a mechanism exists whereby conflicts can be resolved automatically.

You can choose to have a conflicts resolution dialog box displayed by means of the *View->Settings...* menu item, on the *Conflict Resolution* tab of the dialog.

Figure 19-5. Options



You can choose to have conflicts checked under the following circumstances:

- After any item is changed (in other words, as soon as the conflict is created)
- Before saving the configuration (including building)
- Never

The method you chose depends on how much you need your configuration to be free of conflicts. You may want to avoid having to clean up all the conflicts at once, or you may want to keep the configuration consistent at all times. If you have major changes to implement, which may resolve the conflicts, then you might want to wait until after you have completed these changes before you check for conflicts.

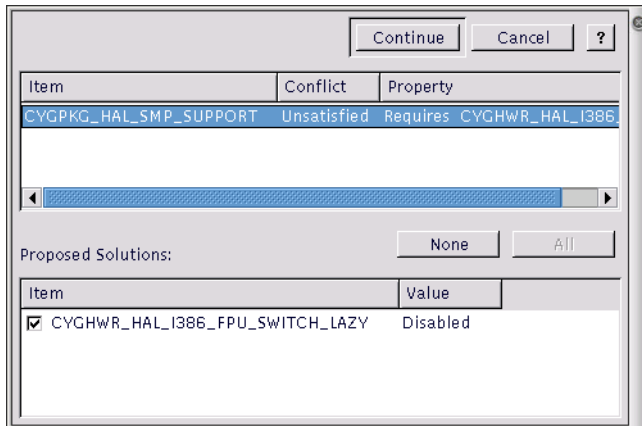
Note: If you choose to check conflicts after any item is changed, only newly arising conflicts are displayed. If you choose to check for conflicts before saving the configuration, the complete set is displayed.

Automatic resolution

If you check the “Automatically suggest fixes” check box, a conflicts resolution dialog box will be displayed whenever new conflicts are created. The same dialog box may be displayed at any stage by means of the *Tools->Resolve Conflicts* menu item.

The conflicts resolution dialog box contains two major windows.

Figure 19-6. Resolve conflicts window



The upper contains the set of conflicts to be addressed; the format of the data being as that of the Conflicts Window. The lower window contains a set of proposed resolutions – each entry is a suggested configuration item value change that as a whole may be expected to lead to the currently selected conflict being resolved.

Note that there is no guarantee:

- that automatic resolutions will be determinable for every conflict.
- that the resolutions for separate conflicts will be independent. In other words, the resolution of one conflict may serve to prevent the resolution of another.
- that the resolution conflicts will not create further conflicts.

The above warnings are, however, conservative. In practice (so long as the number and extent of conflicts are limited) automatic conflict resolution may be used to good effect to correct problems without undue amounts of programmer intervention.

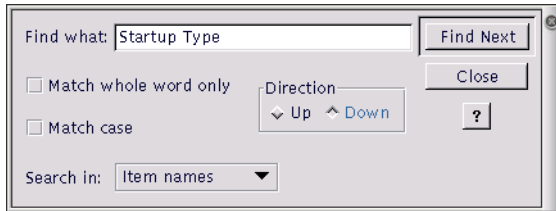
In order to select the conflicts to be applied, select or clear the check boxes against the resolutions for each proposed resolution. By default all resolutions are selected; you can return to the default state (in other words, cause all check boxes for each conflict to again become checked) by pressing the “Reset” button. Note that multiple selection may be used in the resolutions control to allow ranges of check boxes to be toggled in one gesture.

When you are happy to apply the selected resolutions for each conflict displayed, click *Apply*; this will apply the resolutions. Alternatively you may cancel from the dialog box without any resolutions being applied.

Chapter 20. Searching

Select *Edit --> Find*. You will be presented with a Find dialog box:

Figure 20-1. Find dialog box



Using this dialog box you can search for an exact text string in any one of three ways, as specified by your selection in the “Search in” drop-list:

- Macro names - the search is for a text match within configuration item macro names
- Item names - the search is for a text match within configuration item descriptive names
- Short descriptions - the search is for a text match within configuration item short descriptions

Note that to invoke *Find* you can also click the *Find* icon on the toolbar.

Chapter 21. Building

When you have configured eCos, you may build the configuration.

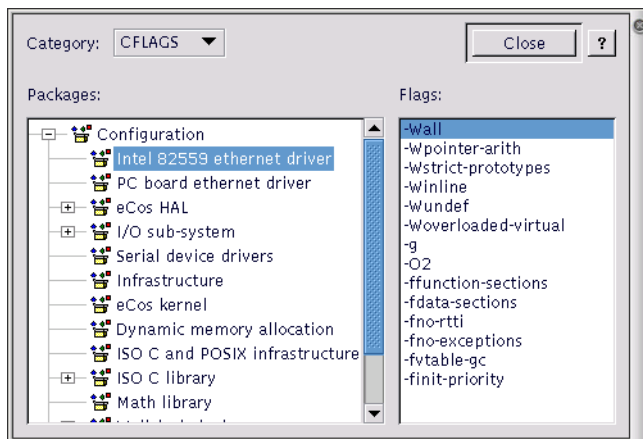
On the *Build* menu, click:

- *Library* (or click the Build Library icon on the toolbar) – this causes the eCos configuration to be built. The result of a successful build will be (among other things) a library against which user code can be linked
- *Tests* – this causes the eCos configuration to be built, and additionally builds the relevant test cases linked against the eCos library
- *Clean* – this removes all intermediate files, thus causing a subsequent build/library or build/tests operation to cause recompilation of all relevant files.
- *Stop* – this causes a currently executing build (any of the above steps) to be interrupted

Build options may be displayed by using the *Build->Options* menu item. This displays a dialog box containing a drop-list control and two windows. The drop-list control allows you to select the type of build option to be displayed (for example “LDFLAGS” are the options applied at link-time. The left-hand window is a tree view of the packages loaded in the current configuration. The right-hand window is a list of the build options that will be used for the currently selected package.

Note that this dialog box currently affords only read-only access to the build options. In order to change build options you must edit the relevant string configuration item.

A single level of inheritance is supported: each package’s build options are combined with the global options (these are to be found in the “Global build options” folder in the configuration view).



Selecting Build Tools

Normally the installation process will supply the information required for the eCosConfiguration Tool to locate the build tools (compiler, linker, etc.) necessary to perform a build. However if this information is not registered, or

it is necessary to specify the location manually (for example, when a new toolchain installation has been made), select *Tools->Paths->Build Tools*. The following dialog box will be displayed:

Figure 21-1. Build tools

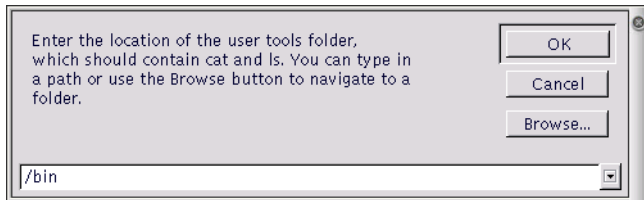


This dialog box allows you to locate the folder containing the build tools.

Selecting User Tools

Normally the installation process will supply the information required for the eCosConfiguration Tool to locate the user tools (cat, ls, etc.) necessary to perform a build. However if this information is not registered, or it is necessary to specify the location manually (for example, when a new toolchain installation has been made), select *Tools->Paths->User Tools*. The following dialog box will be displayed:

Figure 21-2. User tools



Chapter 22. Execution

Test executables that have been linked using the Build/Tests operation against the current configuration can be executed by selecting *Tools->Run Tests*.

When tests are run, the Configuration Tool looks for a platform name corresponding to the currently loaded hardware template. If no such platform is found, a dialog will be displayed for you to define one; this dialog is similar to that displayed by the *Add* function in the *Tools->Platforms* dialog, but in this case the platform name cannot be changed.

When a test run is invoked, a property sheet is displayed, comprising three tabs: *Executables*, *Output* and *Summary*.

Note that the property sheet is resizable.

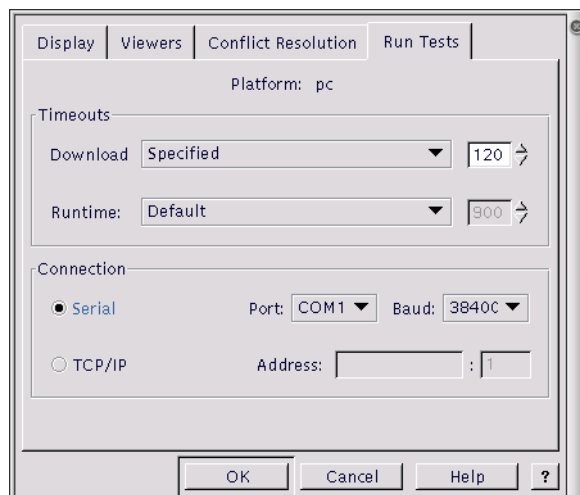
Three buttons appear on the property sheet itself: *Run/Stop*, *Close* and *Properties*.

The *Run* button is used to initiate a test run. Those tests selected on the *Executables* tab are run, and the output recorded on the *Output* and *Summary* tabs. During the course of a run, the *Run* button changes to “Stop”. The button may be used to interrupt a test run at any point.

Properties

The *Properties* button is used to change the connectivity properties for the test run.

Figure 22-1. Properties dialog box



Download Timeout

This group of controls serves to set the maximum time that is allowed for downloading a test to the target board. If the time is exceeded, the test will be deemed to have failed for reason of “Download Timeout” and the execution

of that particular test will be abandoned. This option only applies to tests run on hardware, not to those executed in a simulator. Times are in units of elapsed seconds.

Three options are available using the drop-down list:

- Calculated from file size - an estimate of the maximum time required for download is made using the (stripped) executable size and the currently used baud rate
- Specified - a user-specified value may be entered in the adjacent edit box
- None - no maximum download time is to be applied.

Run time Timeout

This group of controls serves to set the maximum time that is allowed for executing a test on the target board or in a simulator. If the time is exceeded, the test will be deemed to have failed for reason of “Timeout” and the execution of that particular test will be abandoned. In the case of hardware, the time is measured in elapsed seconds: in the case of a simulator it is in CPU seconds.

Three options are available using the drop-down list:

- None - no maximum download time is to be applied.
- Specified - a user-specified value may be entered in the adjacent edit box
- Default - a default value of 30 seconds is used

Connection

The *Connection* controls may be used to specify how the target board is to be accessed.

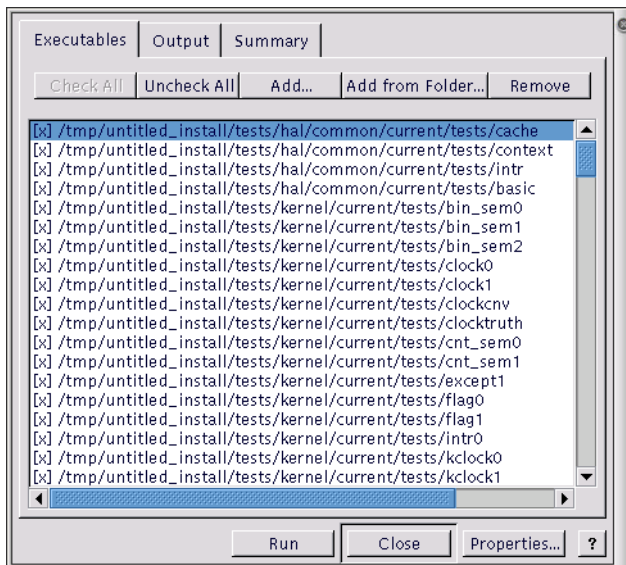
If the target board is connected using a serial cable, the *Serial* radio button should be checked. In this case you can select a port (COM1, COM2, ...) and an appropriate baud rate using drop-list boxes.

If the target board is accessed remotely using GDB remote protocol, the “TCP/IP” radio button should be checked. In this case you can select a host name and TCP/IP port number using edit boxes.

Executables Tab

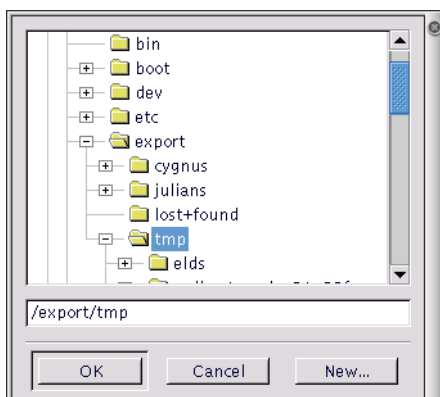
This is used to adjust the set of tests available for execution. A check box against each executable name indicates whether that executable will be included when the *Run* button is pressed. The *Check All* and *Uncheck All* buttons may be used to check or uncheck all items.

When the property sheet is first displayed, it will be pre-populated with those test executables that have been linked using the Build/Tests operation against the current configuration.

Figure 22-2. Run tests

You can right-click in the window to display a context menu containing *Add* and *Remove* items. Clicking *Remove* will remove those executables selected. Clicking *Add* will display a dialog box that allows you to add to the set of items. Equivalently the *Add* button may be used to add executables, and the *DEL* key may be used to remove them.

You can use the *Add from Folder* button to add a number of executables in a specified folder (optionally including subfolders, if you click on *Yes* when asked).

Figure 22-3. Add files from folder

Output Tab

This tab is used to display the output from running tests. The output can be saved to a file or cleared by means of the popup menu displayed when you right-click in the window.

Summary Tab

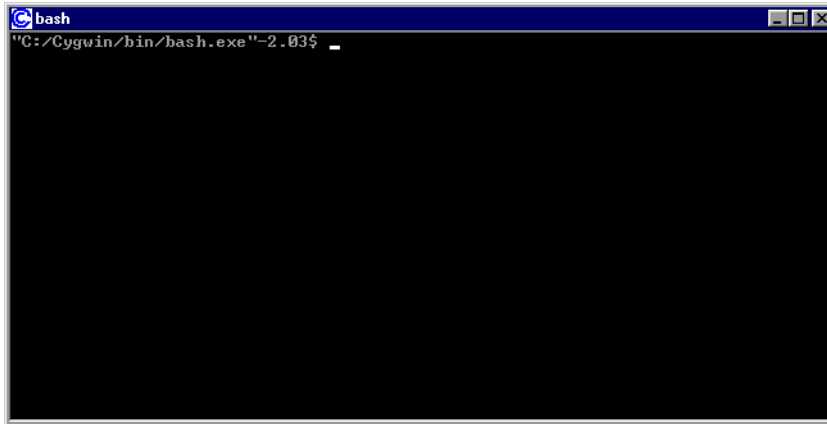
This tab is used to display a record, in summary form, of those tests executed. For each execution, the following information is displayed:

- *Time* - the date and time of execution
- *Host* - the host name of the machine from which the test was downloaded
- *Platform* - the platform on which the test was executed
- *Executable* - the executable (file name) of the test executed
- *Status* - the result of executing the test. This will be one of the following:
 - Not started
 - No result
 - Inapplicable
 - Pass
 - DTimeout
 - Timeout
 - Cancelled
 - Fail
 - Assert fail
- *Size* - the size [stripped/unstripped] of the test executed
- *Download* - the download time [mm:ss/mm:ss] used. The first of the two times displayed represents the actual time used: the second the limit time.
- *Elapsed* - the elapsed time [mm:ss] used.
- *Execution* - the execution time [mm:ss/mm:ss] used. The first of the two times displayed represents the actual time used: the second the limit time.

The output can be saved to a file or cleared by means of the popup menu displayed when you right-click in the window.

Chapter 23. Creating a Shell

To call up a shell window, select *Tools->Shell*. Under Windows, you will get a Cygwin shell similar to the one below. On Linux, you will get a standard Linux shell window.



Keyboard Accelerators

The following table presents the list of keyboard accelerators that can be used with the Configuration Tool.

Table 23-1. Keyboard accelerators

Accelerator	Action	Remarks
<i>Alt+1</i>	hide/show properties window	
<i>Alt+2</i>	hide/show output window	
<i>Alt+3</i>	hide/show short description window	
<i>Alt+4</i>	hide/show conflicts window	
<i>Ctrl+A</i>	select all	output window
<i>Ctrl+C</i>	copy	output window
<i>Ctrl+F</i>	Edit->Find	
<i>Ctrl+N</i>	File->New	
<i>Ctrl+O</i>	File->Open	
<i>Ctrl+S</i>	File->Save	
<i>Ctrl+V</i>	Paste	in-cell editing
<i>Ctrl+X</i>	Cut	in-cell editing
<i>Ctrl+Z</i>	Undo	in-cell editing
<i>F1</i>	Context-sensitive help	
<i>F3</i>	Find next	

Accelerator	Action	Remarks
<i>F7</i>	Build->Library	
<i>Shift+F7</i>	Build->Tests	
<i>Alt+F6</i>	View->Next window	
<i>Shift+Alt+0</i>	View->Previous window	
<i>Shift+Ins</i>	Paste	in-cell editing
<i>Shift+F10</i>	Display context menu	Configuration
<i>Alt+Enter</i>	Display properties dialog box	Configuration
<i>></i>	Increment item value	Configuration
<i><</i>	Decrement item value	Configuration
<i>Space</i>	Toggle item value	Configuration

V. eCos Programming Concepts and Techniques

Programming with eCos is somewhat different from programming in more traditional environments. eCos is a configurable open source system, and you are able to configure and build a system specifically to meet the needs of your application.

Various different directory hierarchies are involved in configuring and building the system: the *component repository*, the *build tree*, and the *install tree*. These directories exist in addition to the ones used to develop applications.

Chapter 24. CDL Concepts

About this chapter

This chapter serves as a brief introduction to the concepts involved in eCos (Embedded Configurable Operating System). It describes the configuration architecture and the underlying technology to a level required for the embedded systems developer to configure eCos. It does not describe in detail aspects such as how to write reusable components for eCos: this information is given in the *Component Writer's Guide*.

Background

Software solutions for the embedded space place particularly stringent demands on the developer, typically represented as requirements for small memory footprint, high performance and robustness. These demands are addressed in eCos by providing the ability to perform compile-time specialization: the developer can tailor the operating system to suit the needs of the application. In order to make this process manageable, eCos is built in the context of a Configuration Infrastructure: a set of tools including a Configuration Tool and a formal description of the process of configuration by means of a *Component Definition Language*.

Configurations

eCos is tailored at source level (that is, before compilation or assembly) in order to create an eCos *configuration*. In concrete terms, an eCos configuration takes the form of a configuration save file (with extension .ecc) and set of files used to build user applications (including, when built, a library file against which the application is linked).

Component Repository

eCos is shipped in source in the form of a *component repository* - a directory hierarchy that contains the sources and other files which are used to build a configuration. The component repository can be added to by, for example, downloading from the net.

Component Definition Language

Part of the component repository is a set of files containing a definition of its structure. The form used for this purpose is the *Component Definition Language* (CDL). CDL defines the relationships between components and other information used by tools such as the eCosConfiguration Tool. CDL is generally formulated by the writers of components: it is not necessary to write or understand CDL in order for the embedded systems developer to construct an eCos configuration.

Packages

The building blocks of an eCos configuration are called *packages*. Packages are the units of software distribution. A set of core packages (such as kernel, C library and math library) is provided by Red Hat; additional third-party packages will be available in future.

A package may exist in one of a number of *versions*. The default version is the *current* version. Only one version of a given package may be present in the component repository at any given time.

Packages are organized in a tree hierarchy. Each package is either at the top-level or is the child of another package.

The eCos Package Administration Tool can be used to add or remove packages from the component repository. The eCos Configuration Tool can be used to include or exclude packages from the configuration being built.

Configuration Items

Configuration items are the individual entities that form a configuration. Each item corresponds to the setting of a C pre-processor macro (for example, `CYGHWR_HAL_ARM_PID_GDB_BAUD`). The code of eCos itself is written to test such pre-processor macros so as to tailor the code. User code can do likewise.

Configuration items come in the following flavors:

- *None*: such entities serve only as place holders in the hierarchy, allowing other entities to be grouped more easily.
- *Boolean* entities are the most common flavor; they correspond to units of functionality that can be either enabled or disabled. If the entity is enabled then there will be a `#define`; code will check the setting using, for example, `#ifdef`
- *Data* entities encapsulate some arbitrary data. Other properties such as a set or range of legal values can be used to constrain the actual values, for example to an integer or floating point value within a certain range.
- *Booldata* entities combine the attributes of *Boolean* and *Data*: they can be enabled or disabled and, if enabled, will hold a data value.

Like packages, configuration items exist in a tree-based hierarchy: each configuration item has a parent which may be another configuration item or a package. Under some conditions (such as when packages are added or removed from a configuration), items may be “re-parented” such that their position in the tree changes.

Expressions

Expressions are relationships between CDL items. There are three types of expression in CDL:

Table 24-1. CDL Expressions

Expression Type	Result	Common Use (see Table 24-2)
Ordinary	A single value	<code>legal_values</code> property
List	A range of values (for example “1 to 10”)	<code>legal_values</code> property
Goal	True or False	<code>requires</code> and <code>active_if</code> properties

Properties

Each configuration item has a set of properties. The following table describes the most commonly used:

Table 24-2. Configuration properties

<i>Property</i>	<i>Use</i>
Flavor	The “type” of the item, as described above
Enabled	Whether the item is enabled
Current_value	The current value of the item
Default_value	An ordinary expression defining the default value of the item
Legal_values	A list expression defining the values the item may hold (for example, 1 to10)
Active_if	A goal expression denoting the requirement for this item to be active (see below: <i>Inactive Items</i>)
Requires	A goal expression denoting requirements this item places on others (see below: <i>Conflicts</i>)
Calculated	Whether the item as non-modifiable
Macro	The corresponding C pre-processor macro
File	The C header file in which the macro is defined
URL	The URL of a documentation page describing the item
Hardware	Indicates that a particular package is related to specific hardware

A complete description of properties is contained in the *Component Writer’s Guide*.

Inactive Items

Descendants of an item that is disabled are inactive: their values may not be changed. Items may also become *inactive* if an *active_if* expression is used to make the item dependent on an expression involving other items.

Conflicts

Not all settings of configuration items will lead to a coherent configuration; for example, the use of a timeout facility might require the existence of timer support, so if the one is required the other cannot be removed. Coherence is policed by means of consistency rules (in particular, the goal expressions that appear as CDL items *requires* and *active_if* attributes [see above]). A violation of consistency rules creates a *conflict*, which must be resolved in order to ensure a consistent configuration. Conflict resolution can be performed manually or with the assistance of the eCos tools. Conflicts come in the following flavors:

- An *unresolved* conflict means that there is a reference to an entity that is not yet in the current configuration

- An *illegal value* conflict is caused when a configuration item is set to a value that is not permitted (that is, a *legal_values* goal expression is failing)
- An *evaluation exception* conflict is caused when the evaluation of an expression would fail (for example, because of a division by zero)
- An *unsatisfied goal* conflict is caused by a failing *requires* goal expression
- A *bad data* conflict arises only rarely, and corresponds to badly constructed CDL. Such a conflict can only be resolved by reference to the CDL writer.

Templates

A *template* is a saved configuration - that is, a set of packages and configuration item settings. Templates are provided with eCos to allow you to get started quickly by instantiating (copying) a saved configuration corresponding to one of a number of common scenarios; for example, a basic eCos configuration template is supplied that contains the infrastructure, kernel, C and math libraries, plus their support packages.

Chapter 25. The Component Repository and Working Directories

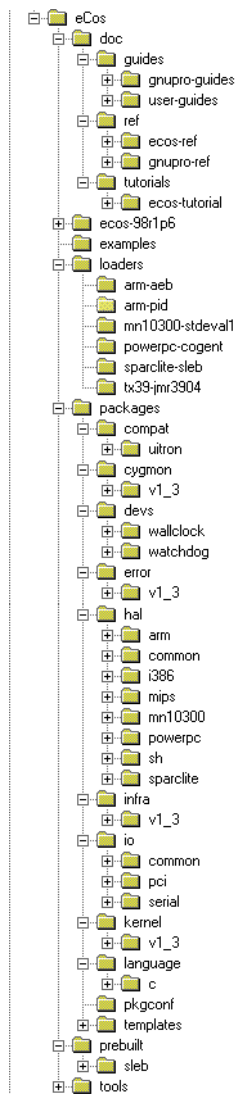
Each of the file trees involved in eCos development has a different role.

Component Repository

The eCos *component repository* contains directories for all the packages that are shipped with eCos or provided by third parties.

The component repository should not be modified as part of application development.

Figure 25-1. Component repository



Purpose

The component repository is the master copy of source code for all system and third party components. It also contains some files needed to administer and build the system, such as **ecosadmin.tcl**.

How is it modified?

You modify it by importing new versions of packages from a distribution or removing existing packages. These

activities are undertaken using the eCos Package Administration Tool.

When is it edited manually?

Files in the component repository should only be edited manually as determined by the component maintainer.

User Applications

User application source code should *not* go into the component repository.

Examples of files in this hierarchy:

`BASE_DIR/doc/ref/ecos-ref.html`

The top level HTML file for the *eCos Reference Manual*.

`BASE_DIR/prebuilt/pid/tests/kernel/<version>/tests/thread_gdb.exe`

`BASE_DIR/prebuilt/linux/tests/kernel/<version>/tests/thread_gdb.exe`

Pre-built tests for the supported platforms, and the synthetic Linux target.

`BASE_DIR/examples/twothreads.c`

One of the example programs.

`BASE_DIR/ecosadmin.tcl`

The Tcl program which is used to import new versions of packages from a distribution or remove existing packages.

`BASE_DIR/packages/language/c/libm/<version>/src/double/portable-api/s_tanh.c`

Implementation of the hyperbolic tangent function in the standard math library.

`BASE_DIR/pkgconf/rules.mak`

A file with **make** rules, used by the `makefile`.

Build Tree

The *build tree* is the directory hierarchy in which all *generated* files are placed. Generated files consist of the `makefile`, the compiled object files, and a dependency file (with a `.d` extension) for each source file.

Purpose

The build tree is where all intermediate object files are placed.

How is it modified?

Recompiling can modify the object files.

User applications

User application source or binary code should *not* go in the build tree.

Examples of files in this hierarchy

```
ecos-work/language/c/libc/<version>/src
```

The directory in which object files for the C library are built.

Install Tree

The *install tree* is the location for all files needed for application development. The `libtarget.a` library, which contains the custom-built eCos kernel and other components, is placed in the install tree, along with all packages' public header files. If you build the tests, the test executable programs will also be placed in the install tree.

By default, the install tree is created by **ecosconfig** in a subdirectory of the build tree called `install`. This can be modified with the `--prefix` option (see [Chapter 28](#)).

Purpose

The install tree is where the custom-built `libtarget.a` library, which contains the eCos kernel and other components, is located. The install tree is also the location for all the header files that are part of a published interface for their component.

How is it modified?

Recompiling can replace `libtarget.a` and the test executables.

When is it edited manually?

Where a memory layout requires modification without use of the eCos Configuration Tool, the memory layout files must be edited directly in the install tree. These files are located at `install/include/pkgconf/mlt_*.*`. Note

that subsequent modification of the install tree using the Configuration Tool will result in such manual edits being lost.

User applications

User application source or binary code should *not* go in the install tree.

Examples of files in this hierarchy

`install/lib/libtarget.a`

The library containing the kernel and other components.

`install/include/cyg/kernel/kapi.h`

The header file for the kernel C language API.

`install/include/pkgconf/mlt_arm_pid_ram.ldi`

The linker script fragment describing the memory layout for linking applications intended for execution on an ARM PID development board using RAM startup.

`install/include/stdio.h`

The C library header file for standard I/O.

Application Build Tree

This tree is not part of eCos itself: it is the directory in which eCos end users write their own applications.

Example applications and their `Makefile` are located in the component repository, in the directory `BASE_DIR/examples`.

There is no imposed format on this directory, but there are certain compiler and linker flags that must be used to compile an eCos application. The basic set of flags is shown in the example `Makefile`, and additional details can be found in [Chapter 26](#).

Chapter 26. Compiler and Linker Options

eCos is built using the GNU C and C++ compilers. eCos relies on certain features of these tools such as constructor priority ordering and selective linking which are not part of other toolchains.

Some GCC options are required for eCos, and others can be useful. This chapter gives a brief description of the required options as well as some recommended eCos-specific options. All other GCC options (described in the GCC manuals) are available.

Compiling a C Application

The following command lines demonstrate the *minimum* set of options required to compile and link an eCos program written in C.

Note: Remember that when this manual shows **TARGET-gcc** you should use the full name of the cross compiler, e.g. **i386-elf-gcc**, **arm-elf-gcc**, or **sh-elf-gcc**. When compiling for the synthetic Linux target, use the native **gcc** which must have the features required by eCos.

```
$ TARGET-gcc -c -IINSTALL_DIR/include file.c
$ TARGET-gcc -o program file.o -LINSTALL_DIR/lib -Ttarget.ld -nostdlib
```

Note: Certain targets may require extra options, for example the SPARClite architectures require the option `-mcpu=sparclite`. Examine the `BASE_DIR/examples/Makefile` or the “Global compiler flags” option (CYGBLD_GLOBAL_CFLAGS) in your generated eCos configuration) to see if any extra options are required, and if so, what they are.

The following command lines use some other options which are recommended because they use the selective linking feature:

```
$ TARGET-gcc -c -IINSTALL_DIR/include -I. -ffunction-sections -fdata-sections -g -O2 file.c
$ TARGET-gcc -o program file.o -ffunction-sections -fdata-sections -Wl,--gc-sections -g -O2 \
    -LINSTALL_DIR/lib -Ttarget.ld -nostdlib
```

Compiling a C++ Application

The following command lines demonstrate the *minimum* set of options required to compile and link an eCos program written in C++.

Note: Remember that when this manual shows **TARGET-g++** you should use the full name of the cross compiler, e.g. **i386-elf-g++**, **arm-elf-g++**, or **sh-elf-g++**. When compiling for the synthetic Linux target, use the native **g++** which must have the features required by eCos.

```
$ TARGET-g++ -c -IINSTALL_DIR/include -fno-rtti -fno-exceptions file.cxx
```

```
$ TARGET-g++ -o program file.o -LINSTALL_DIR/lib -Ttarget.ld -nostdlib
```

Note: Certain targets may require extra options, for example the SPARClite architectures require the option `-mcpu=sparclite`. Examine the `BASE_DIR/packages/targets` file or `BASE_DIR/examples/Makefile` or the “Global compiler flags” option (CYGBLD_GLOBAL_CFLAGS) in your generated eCos configuration) to see if any extra options are required, and if so, what they are.

The following command lines use some other options which are recommended because they use the *selective linking* feature:

```
$ TARGET-g++ -c -IINSTALL_DIR/include -I. -ffunction-sections -fdata-sections -fno-rtti \
    -fno-exceptions -finit-priority -g -O2 file.cxx
$ TARGET-g++ -o program file.o -Wl,--gc-sections -g -O2 -LINSTALL_DIR/lib -Ttarget.ld -nostdlib
```

Chapter 27. Debugging Techniques

eCos applications and components can be debugged in traditional ways, with printing statements and debugger single-stepping, but there are situations in which these techniques cannot be used. One example of this is when a program is getting data at a high rate from a real-time source, and cannot be slowed down or interrupted.

eCos's infrastructure module provides a *tracing* formalism, allowing the kernel's tracing macros to be configured in many useful ways. eCos's kernel provides *instrumentation buffers* which also collect specific (configurable) data about the system's history and performance.

Tracing

To use eCos's tracing facilities you must first configure your system to use *tracing*. You should enable the Asserts and Tracing component (CYGPKG_INFRA_DEBUG) and the Use tracing component within it (CYGDBG_USE_TRACING). These options can be enabled with the Configuration Tool or by editing the file `BUILD_DIR/pkgconf/infra.h` manually.

You should then examine all the tracing-related options in the *Package: Infrastructure* chapter of the *eCos Reference Manual*. One useful set of configuration options are: `CYGDBG_INFRA_DEBUG_FUNCTION_REPORTS` and `CYGDBG_INFRA_DEBUG_TRACE_MESSAGE`, which are both enabled by default when tracing is enabled.

The following “Hello world with tracing” shows the output from running the hello world program (from [the Section called eCos Hello World in Chapter 13](#)) that was built with tracing enabled:

Example 27-1. Hello world with tracing

```
$ mips-tx39-elf-run --board=jmr3904 hello
Hello, eCos world!
ASSERT FAIL: <2>cyg_trac.h [ 623] Cyg_TraceFunction_Report_::set_exitvoid()
TRACE: <1>mlqueue.cxx [ 395] Cyg_ThreadQueue_Implementation::enqueue()
TRACE: <1>mlqueue.cxx [ 395] Cyg_ThreadQueue_Implementation::enqueue()
TRACE: <1>mlqueue.cxx [ 126] Cyg_Scheduler_Implementation::add_thread()
TRACE: <1>thread.cxx [ 654] Cyg_Thread::resume()
TRACE: <1>cstartup.cxx [ 160] cyg_iso_c_start()
TRACE: <1>startup.cxx [ 142] cyg_package_start()
TRACE: <1>startup.cxx [ 150] cyg_user_start()
TRACE: <1>startup.cxx [ 150] cyg_user_start()
TRACE: <1>startup.cxx [ 153] cyg_user_start()
TRACE: <1>startup.cxx [ 157] cyg_user_start()
TRACE: <1>sched.cxx [ 212] Cyg_Scheduler::start()
TRACE: <1>mlqueue.cxx [ 102] Cyg_Scheduler_Implementation::schedule()
TRACE: <1>mlqueue.cxx [ 437] Cyg_ThreadQueue_Implementation::highpri()
TRACE: <1>mlqueue.cxx [ 437] Cyg_ThreadQueue_Implementation::highpri()
TRACE: <1>mlqueue.cxx [ 102] Cyg_Scheduler_Implementation::schedule()
TRACE: <2>intr.cxx [ 450] Cyg_Interrupt::enable_interrupts()
TRACE: <2>intr.cxx [ 450] Cyg_Interrupt::enable_interrupts()
TRACE: <2>thread.cxx [ 69] Cyg_HardwareThread::thread_entry()
TRACE: <2>cstartup.cxx [ 127] invoke_main()
TRACE: <2>cstartup.cxx [ 127] invoke_main()
TRACE: <2>dummysxmain.cxx [ 60] __main()
TRACE: <2>dummysxmain.cxx [ 60] __main()
```

```
TRACE: <2>dummyxxmain.cxx    [ 63] __main()
TRACE: <2>dummyxxmain.cxx    [ 67] __main()
TRACE: <2>memcpy.c           [ 112] _memcpy()
TRACE: <2>memcpy.c           [ 112] _memcpy()
TRACE: <2>memcpy.c           [ 164] _memcpy()
TRACE: <2>cstartup.cxx        [ 137] invoke_main()
TRACE: <2>exit.cxx           [ 71] __libc_exit()
TRACE: <2>exit.cxx           [ 71] __libc_exit()
TRACE: <2>atexit.cxx         [ 84] cyg_libc_invoke_atexit_handlers()
TRACE: <2>atexit.cxx         [ 84] cyg_libc_invoke_atexit_handlers()
```

Scheduler:

Lock: 0
Current Thread: <null>

Threads:

Idle Thread pri = 31 state = R id = 1
 stack base = 800021F0 ptr = 80002510 size = 00000400
 sleep reason NONE wake reason NONE
 queue = 80000C54 wait info = 00000000

<null> pri = 0 state = R id = 2
 stack base = 80002A48 ptr = 8000A968 size = 00008000
 sleep reason NONE wake reason NONE
 queue = 80000BD8 wait info = 00000000

Kernel Instrumentation

Instrument buffers can be used to find out how many events of a given type happened in the kernel during execution of a program.

You can monitor a class of several types of events, or you can just look at individual events.

Examples of *events* that can be monitored are:

- scheduler events
- thread operations
- interrupts
- mutex operations
- binary semaphore operations
- counting semaphore operations
- clock ticks and interrupts

Examples of fine-grained scheduler event types are:

- scheduler lock

- scheduler unlock
- rescheduling
- time slicing

Information about the events is stored in an *event record*. The structure that defines this record has type `struct Instrument_Record`:

The list of records is stored in an array called `instrument_buffer` which you can let the kernel provide or you can provide yourself by setting the configuration option `CYGVAR_KERNEL_INSTRUMENT_EXTERNAL_BUFFER`.

To write a program that examines the instrumentation buffers:

1. Enable instrumentation buffers in the eCos kernel configuration. The component macro is `CYGPKG_KERNEL_INSTRUMENT`.

2. To allocate the buffers yourself, enable the configuration option `CYGVAR_KERNEL_INSTRUMENT_EXTERNAL_BUFFER`.

3. Include the header file `cyg/kernel/instrmnt.h`.

```
#include <cyg/kernel/instrmnt.h>
```

4. The `Instrument_Record` structure is not published in the kernel header file. In the future there will be a cleaner mechanism to access it, but for now you should paste into your code in the following lines:

```
struct Instrument_Record
{
    CYG_WORD16 type; // record type
    CYG_WORD16 thread; // current thread id
    CYG_WORD timestamp; // 32 bit timestamp
    CYG_WORD arg1; // first arg
    CYG_WORD arg2; // second arg
};
```

5. Enable the events you want to record using `cyg_instrument_enable()`, and disable them later. Look at `cyg/kernel/instrmnt.h` and the examples below to see what events can be enabled.

6. Place the code you want to debug between the matching functions `cyg_instrument_enable()` and `cyg_instrument_disable()`.

7. Examine the buffer. For now you need to look at the data in there (the example program below shows how to do that), and future versions of eCos will include a host-side tool to help you understand the data.

Example 27-2. Using instrument buffers

This program is also provided in the `examples` directory.

```
/* this is a program which uses eCos instrumentation buffers; it needs
to be linked with a kernel which was compiled with support for
instrumentation */
```

```
#include <stdio.h>
#include <pkgconf/kernel.h>
#include <cyg/kernel/instrmnt.h>
#include <cyg/kernel/kapi.h>

#ifdef CYGVAR_KERNEL_INSTRUMENT_EXTERNAL_BUFFER
```

```
# error You must configure eCos with CYGVAR_KERNEL_INSTRUMENT_EXTERNAL_BUFFER
#endif

struct Instrument_Record
{
    CYG_WORD16 type; // record type
    CYG_WORD16 thread; // current thread id
    CYG_WORD timestamp; // 32 bit timestamp
    CYG_WORD arg1; // first arg
    CYG_WORD arg2; // second arg
};

struct Instrument_Record instrument_buffer[20];
cyg_uint32 instrument_buffer_size = 20;

int main(void)
{
    int i;

    cyg_instrument_enable(CYG_INSTRUMENT_CLASS_CLOCK, 0);
    cyg_instrument_enable(CYG_INSTRUMENT_CLASS_THREAD, 0);
    cyg_instrument_enable(CYG_INSTRUMENT_CLASS_ALARM, 0);

    printf("Program to play with instrumentation buffer\n");

    cyg_thread_delay(2);

    cyg_instrument_disable(CYG_INSTRUMENT_CLASS_CLOCK, 0);
    cyg_instrument_disable(CYG_INSTRUMENT_CLASS_THREAD, 0);
    cyg_instrument_disable(CYG_INSTRUMENT_CLASS_ALARM, 0);

    for (i = 0; i < instrument_buffer_size; ++i) {
        printf("Record %02d: type 0x%04x, thread %d, ",
            i, instrument_buffer[i].type, instrument_buffer[i].thread);
        printf("time %5d, arg1 0x%08x, arg2 0x%08x\n",
            instrument_buffer[i].timestamp, instrument_buffer[i].arg1,
            instrument_buffer[i].arg2);
    }
    return 0;
}
```

Here is how you could compile and run this program in the `examples` directory, using (for example) the MN10300 simulator target:

```
$ make XCC=mn10300-elf-gcc INSTALL_DIR=/tmp/ecos-work-mn10300/install instrument-test
mn10300-elf-gcc -c -o instrument-test.o -g -Wall -I/tmp/ecos-work-mn10300/install/include \
    -ffunction-sections -fdata-sections instrument-test.c
mn10300-elf-gcc -nostartfiles -L/tmp/ecos-work-mn10300/install/lib -Wl,--gc-sections -o \
    instrument-test instrument-test.o -Ttarget.ld -nostdlib
$ mn10300-elf-run --board=stdevall instrument-test
```

Example 27-3. Instrument buffer output

Here is the output of the **instrument-test** program. Notice that in little over 2 seconds, and with very little activity, and with few event types enabled, it gathered 17 records. In larger programs it will be necessary to select very few event types for debugging.

```

Program to play with instrumentation buffer
Record 00: type 0x0207, thread 2, time 6057, arg1 0x48001cd8, arg2 0x00000002
Record 01: type 0x0202, thread 2, time 6153, arg1 0x48001cd8, arg2 0x00000000
Record 02: type 0x0904, thread 2, time 6358, arg1 0x48001d24, arg2 0x00000000
Record 03: type 0x0905, thread 2, time 6424, arg1 0x00000002, arg2 0x00000000
Record 04: type 0x0906, thread 2, time 6490, arg1 0x00000000, arg2 0x00000000
Record 05: type 0x0901, thread 2, time 6608, arg1 0x48009d74, arg2 0x48001d24
Record 06: type 0x0201, thread 2, time 6804, arg1 0x48001cd8, arg2 0x480013e0
Record 07: type 0x0803, thread 1, time 94, arg1 0x00000000, arg2 0x00000000
Record 08: type 0x0801, thread 1, time 361, arg1 0x00000000, arg2 0x00000000
Record 09: type 0x0802, thread 1, time 548, arg1 0x00000001, arg2 0x00000000
Record 10: type 0x0803, thread 1, time 94, arg1 0x00000000, arg2 0x00000000
Record 11: type 0x0801, thread 1, time 361, arg1 0x00000001, arg2 0x00000000
Record 12: type 0x0903, thread 1, time 513, arg1 0x48009d74, arg2 0x48001d24
Record 13: type 0x0208, thread 1, time 588, arg1 0x00000000, arg2 0x00000000
Record 14: type 0x0203, thread 1, time 697, arg1 0x48001cd8, arg2 0x480013e0
Record 15: type 0x0802, thread 1, time 946, arg1 0x00000002, arg2 0x00000000
Record 16: type 0x0201, thread 1, time 1083, arg1 0x480013e0, arg2 0x48001cd8
Record 17: type 0x0000, thread 0, time 0, arg1 0x00000000, arg2 0x00000000
Record 18: type 0x0000, thread 0, time 0, arg1 0x00000000, arg2 0x00000000
Record 19: type 0x0000, thread 0, time 0, arg1 0x00000000, arg2 0x00000000

```


VI. Configuration and the Package Repository

The following chapters contain information on running **ecosconfig** (the command line tool that manipulates configurations and constructs build trees) and on managing a source repository across multiple versions of eCos.

Chapter 28. Manual Configuration

eCos developers will generally use the graphical Configuration Tool for configuring an eCos system and building the target library. However, some users prefer to use command line tools. These command line tools can also be used for batch operations on all platforms, for example as part of a nightly rebuild and testing procedure.

In the current release of the system the command line tools do not provide exactly the same functionality as the graphical tool. Most importantly, there is no facility to resolve configuration conflicts interactively.

The eCos configuration system, both graphical and command line tools, are under constant development and enhancement. Developers should note that the procedures described may change considerably in future releases.

Directory Tree Structure

When building eCos there are three main directory trees to consider: the source tree, the build tree, and the install tree.

The source tree, also known as the component repository, is read-only. It is possible to use a single component repository for any number of different configurations, and it is also possible to share a component repository between multiple users by putting it on a network drive.

The build tree contains everything that is specific to a particular configuration, including header and other files that contain configuration data, and the object files that result from compiling the system sources for this configuration.

The install tree is usually located in the `install` subdirectory of the build tree. Once an eCos system has been built, the install tree contains all the files needed for application development including the header files and the target library. By making copies of the install tree after a build it is possible to separate application development and system configuration, which may be desirable for some organizations.

Creating the Build Tree

Generating a build tree is a non-trivial operation and should not be attempted manually. Instead, eCos is shipped with a tool called **ecosconfig** that should be used to create a build tree.

Usually **ecosconfig** will be run inside the build tree itself. If you are creating a new build tree then typically you will create a new empty directory using the **mkdir** command, **cd** into that directory, and then invoke **ecosconfig** to create a configuration. By default, the configuration is stored in a file `ecos.ecc` in the current directory. The configuration may be modified by editing this file directly. **ecosconfig** itself deals with a number of coarse-grained configuration options such as the target platform and the packages that should be used.

The **ecosconfig** tool is also used subsequently to generate a build tree for a configuration. Once a build tree exists, it is possible to run **ecosconfig** again inside the same build tree. This will be necessary if you wish to change some of the configuration options.

ecosconfig does not generate the top-level directory of the build tree; you must do this yourself.

```
$ mkdir ecos-work
$ cd ecos-work
```

The next step is to run **ecosconfig**:

```
$ ecosconfig <qualifiers> <command>
```

ecosconfig qualifiers

The available command line qualifiers for **ecosconfig** are as follows. Multiple qualifiers may be used on the command line:

`--help`

Provides basic usage guidelines for the available commands and qualifiers.

`--config=<file>`

Specifies an eCos configuration save file for use by the tool. By default, the file `ecos.ecc` in the current directory is used. Developers may prefer to use a common location for all their eCos configurations rather than keep the configuration information in the base of the build tree.

`--prefix=<dir>`

Specifies an alternative location for the install tree. By default, the install tree resides inside the `install` directory in the build tree. Developers may prefer to locate the build tree in a temporary file hierarchy but keep the install tree in a more permanent location.

`--srcdir=<dir>`

Specifies the location of the component repository. By default, the tool uses the location specified in the `ECOS_REPOSITORY` environment variable. Developers may prefer to use of this qualifier if they are working with more than one repository.

`--no-resolve`

Disables the implicit resolution of conflicts while manipulating the configuration data. developers may prefer to resolve conflicts by editing the eCos configuration save file manually.

`--ignore-errors`

`-i`

By default, `ecosconfig` will exit with an error code if the current configuration contains any conflicts, and it is not possible to generate or update a build tree for such configurations. This qualifier causes `ecosconfig` to ignore such problems, and hence it is possible to generate a build tree even if there are still conflicts. Of course, there are no guarantees that the resulting system will actually do anything.

`--verbose`

`-v`

Display more information.

`--quiet`

`-q`

Display less information.

The `--config`, `--prefix` and `--srcdir` qualifiers can also be written with two arguments, for example:

```
ecosconfig --srcdir <dir> ...
```

This simplifies filename completion with some shells.

ecosconfig commands

The available commands for **ecosconfig** are as follows:

list

Lists the available packages, targets and templates as installed in the eCos repository. Aliases and package versions are also reported.

new <target> [<template> [<version>]]

Creates a new eCos configuration for the specified target hardware and saves it. A software template may also be specified. By default, the template named ‘default’ is used. If the template version is not specified, the latest version is used.

target <target>

Changes the target hardware selection for the eCos configuration. This has the effect of unloading packages supporting the target selected previously and loading the packages which support the new hardware. This command will be used typically when switching between a simulator and real hardware.

template <template> [<version>]

Changes the template selection for the eCos configuration. This has the effect of unloading packages specified by the template selected previously and loading the packages specified by the new template. By default, the latest version of the specified template is used.

remove <packages>

Removes the specified packages from the eCos configuration. This command will be used typically when the template on which a configuration is based contains packages which are not required.

add <packages>

Adds the specified packages to the eCos configuration. This command will be used typically when the template on which a configuration is based does not contain all the packages which are required. For example, add-on packages provided by third parties will not be known to the standard templates, so they will have to be added explicitly.

version <version> <packages>

Selects the specified version of a number of packages in the eCos configuration. By default, the most recent version of each package is used. This command will be used typically when an older version of a package is required.

check

Presents the following information concerning the current configuration:

1. the selected target hardware

2. the selected template
3. additional packages
4. removed packages
5. the selected version of packages where this is not the most recent version
6. conflicts in the current configuration

resolve

Resolves conflicts identified in the current eCos configuration by invoking an inference capability. Resolved conflicts are reported, but not all conflicts may be resolvable. This command will be used typically following manual editing of the configuration.

export <file>

Exports a minimal eCos configuration save file with the specified name. This file contains only those options which do not have their default value. Such files are used typically to transfer option values from one configuration to another.

import <file>

Imports a minimal eCos configuration save file with the specified name. The values of those options specified in the file are applied to the current configuration.

tree

Generates a build tree based on the current eCos configuration. This command will be used typically just before building eCos. Normally a build tree can only be generated if the configuration has no unresolved conflicts, but `--ignore-errors` can be used to override this.

Conflicts and constraints

Configuration options are not completely independent. For example the C library's `strtod()` and `atof()` functions rely on the math library package to provide certain functionality. If the math library package is removed then the C library can no longer provide these functions. Each package describes constraints like these in CDL "*requires*" properties. If a constraint is not satisfied, then the configuration contains a conflict. For any given conflict there can be several resolution options. For example, it would be possible to add the math library package back to the configuration, or to disable the `strtod()` and `atof()` functions.

The eCos configuration tools will report any conflicts in the current configuration. If there are any such conflicts then the configuration is usually unsafe and it makes no sense to build and run eCos in such circumstances. In fact, any attempt at building eCos is likely to fail. In exceptional cases it is possible to override this by using e.g. the `--ignore-errors` qualifier with `ecosconfig`.

Many constraints are fairly simple in nature, and the configuration tools contain an inference engine which can resolve the associated conflicts automatically. For example, if the math library package is removed then the inference engine can resolve the resulting conflict by disabling the configuration option for `strtod()` and `atof()`. All such changes will be reported. Sometimes the inference engine cannot resolve a conflict, for example it is not allowed

to override a change that has been made explicitly by the user. Sometimes it will find a solution which does not match the application's requirements.

A typical session involving conflicts would look something like this:

```
$ ecosconfig new pid
```

This creates a new configuration with the default template. For most targets this will not result in any conflicts, because the default settings for the various options meet the requirements of the default template.

For some targets there may be conflicts and the inference engine would come into play.

```
$ ecosconfig remove libm
U CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT, new inferred value 0
U CYGFUN_LIBC_strtod, new inferred value 0
U CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT, new inferred value 0
```

ecosconfig reports that this change caused three conflicts, all in the C library. The inference engine was able to resolve all the conflicts and update the relevant configuration options accordingly.

To suppress the inference engine `--no-resolve` can be used:

```
$ ecosconfig new pid
$ ecosconfig --no-resolve remove libm
C CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT, "requires" constraint not satisfied:      CYGPKG_LIBM
C CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT, "requires" constraint not satisfied:    CYGPKG_LIBM
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
```

Three unresolved conflicts are reported.

The **check** command can be used to get the current state of the configuration, and the `--verbose` qualifier will provide additional information:

```
$ ecosconfig --srcdir /home/bartv/ecc/ecc --verbose check
Target: pid
Template: default
Removed:
  CYGPKG_LIBM
3 conflict(s):
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
Possible solution:
  CYGFUN_LIBC_strtod -> 0
  CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT -> 0
C CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT, "requires" constraint not satisfied:    CYGPKG_LIBM
Possible solution:
  CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT -> 0
C CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT, "requires" constraint not satisfied:    CYGPKG_LIBM
Possible solution:
  CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT -> 0
```

If the proposed solutions are acceptable, the `resolve` command can be used to apply them:

```
$ ecosconfig resolve
U CYGSEM_LIBC_STDIO_SCANF_FLOATING_POINT, new inferred value 0
U CYGFUN_LIBC_strtod, new inferred value 0
U CYGSEM_LIBC_STDIO_PRINTF_FLOATING_POINT, new inferred value 0
```

The current configuration is again conflict-free and it is possible to generate a build tree. The `--quiet` qualifier can be used to suppress the change messages, if desired.

When changing individual configuration options by editing the `ecos.ecc` file (as described below), the resulting system should be checked and any problems should be resolved. For example, if `CYGFUN_LIBC_strtod` is explicitly enabled in the savefile:

```
$ edit ecos.ecc
$ ecosconfig check
Target: pid
Template: default
Removed:
    CYGPKG_LIBM
1 conflict(s):
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
$ ecosconfig resolve
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
```

In this case the inference engine cannot resolve the conflict automatically because that would involve changing a user setting. Any attempt to generate a build tree will fail:

```
$ ecosconfig --srcdir /home/bartv/ecc/ecc tree
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
Unable to generate build tree, this configuration still contains conflicts.
Either resolve the conflicts or use --ignore-errors
```

It is still possible to generate a build tree:

```
$ ecosconfig --srcdir /home/bartv/ecc/ecc --ignore-errors tree
C CYGFUN_LIBC_strtod, "requires" constraint not satisfied: CYGPKG_LIBM
$ make
```

In this case eCos will fail to build. In other cases of unresolved conflicts eCos may build, but may not run. In general all conflicts should be resolved by editing the `ecos.ecc` file, by letting the inference engine make appropriate changes, or by other means, before any attempt is made to build or run eCos.

Building the System

Once a build tree has been generated with **ecosconfig**, building eCos is straightforward:

```
$ make
```

The build tree contains the subdirectories, makefiles, and everything else that is needed to generate the default configuration for the selected architecture and platform. The only requirement is that the tools needed for that architecture, for example **powerpc-eabi-g++**, are available using the standard search path. If this is not the case then the **make** will fail with an error message. If you have a multiprocessor system then it may be more efficient to use:

```
$ make -j n
```


where n is equal to the number of processors on your system.

Once the **make** process has completed, the install tree will contain the header files and the target library that are needed for application development.

It is also possible to build the system's test cases for the current configuration:

```
$ make tests
```

The resulting test executables will end up in a `tests` subdirectory of the install tree.

If disk space is scarce then it is possible to make the copy of the install tree for application development purposes, and then use:

```
$ make clean
```

The build tree will now use up a minimum of disk space — the bulk of what is left consists of configuration header files that you may have edited and hence should not be deleted automatically. However, it is possible to rebuild the system at any time without re-invoking **ecosconfig**, just by running **make** again.

Under exceptional circumstances it may be necessary to run **make clean** for other reasons, such as when a new release of the toolchain is installed. The toolchain includes a number of header files which are closely tied to the compiler, for example `limits.h`, and these header files are not and should not be duplicated by eCos. The makefiles perform header file dependency analysis, so that when a header file is changed all affected sources will be rebuilt during the next **make**. This is very useful when the configuration header files are changed, but it also means that a build tree containing information about the locations of header files must be rebuilt. If a new version of the toolchain is installed and the old version is removed then this location information is no longer accurate, and **make** will complain that certain dependencies cannot be satisfied. Under such circumstances it is necessary to do a **make clean** first.

Packages

eCos is a component architecture. The system comes as a number of packages which can be enabled or disabled as required, and new packages can be added as they become available. Unfortunately, the packages are not completely independent: for example the μ ITRON compatibility package relies almost entirely on functionality provided by the kernel package, and it would not make sense to try to build μ ITRON if the kernel was disabled. The C library has fewer dependencies: some parts of the C library rely on kernel functionality, but it is possible to disable these parts and thus build a system that has the C library but no kernel. The **ecosconfig** tool has the capability of checking that all the dependencies are satisfied, but it may still be possible to produce configurations that will not build or (conceivably) that will build but not run. Developers should be aware of this and take appropriate care.

By default, **ecosconfig** will include all packages that are appropriate for the specified hardware in the configuration. The common HAL package and the eCos infrastructure must be present in every configuration. In addition, it is always necessary to have one architectural HAL package and one platform HAL package. Other packages are optional, and can be added or removed from a configuration as required.

The application may not require all of the packages; for example, it might not need the μ ITRON compatibility package, or the floating point support provided by the math library. There is a slight overhead when eCos is built because the packages will get compiled, and there is also a small disk space penalty. However, any unused facilities will get stripped out at link-time, so having redundant packages will not affect the final executable.

Coarse-grained Configuration

Coarse-grained configuration of an eCos system means making configuration changes using the **ecosconfig** tool. These changes include:

1. switching to different target hardware
2. switching to a different template
3. adding or removing a package
4. changing the version of a package

Whenever **ecosconfig** generates or updates an eCos configuration, it generates a configuration save file.

Suppose that the configuration was first created using the following command line:

```
$ ecosconfig new stdevall
```

To change the target hardware to the Cogent CMA28x PowerPC board, the following command would be needed:

```
$ ecosconfig target cma28x
```

To switch to the PowerPC simulator instead:

```
$ ecosconfig target psim
```

As the hardware changes, hardware-related packages such as the HAL packages and device drivers will be added to and removed from the configuration as appropriate.

To remove any package from the current configuration, use the **remove** command:

```
$ ecosconfig remove uitron
```

You can disable multiple packages using multiple arguments, for example:

```
$ ecosconfig remove uitron libm
```

If this turns out to have been a mistake then you can re-enable one or more packages with the **add** command:

```
$ ecosconfig add libm
```

Changing the desired version for a package is also straightforward:

```
$ ecosconfig version v2_1 kernel
```

It is necessary to regenerate the build tree and header files following any changes to the configuration before rebuilding eCos:

```
$ ecosconfig tree
```

Fine-grained Configuration

ecosconfig only provides coarse-grained control over the configuration: the hardware, the template and the packages that should be built. Unlike the Configuration Tool, **ecosconfig** does not provide any facilities for manipulating finer-grained configuration options such as how many priority levels the scheduler should support. There are hundreds of these options, and manipulating them by means of command line arguments would not be sensible.

In the current system fine-grained configuration options may be manipulated by manual editing of the configuration file. When a file has been edited in this way, the **ecosconfig** tool should be used to check the configuration for any conflicts which may have been introduced:

```
$ ecosconfig check
```

The **check** command will list all conflicts and will also rewrite the configuration file, propagating any changes which affect other options. The user may choose to resolve the conflicts either by re-editing the configuration file manually or by invoking the inference engine using the **resolve** command:

```
$ ecosconfig resolve
```

The **resolve** command will list all conflicts which can be resolved and save the resulting changes to the configuration.

It is necessary to regenerate the build tree and header files following any changes to the configuration before rebuilding eCos:

```
$ ecosconfig tree
```

All the configuration options and their descriptions are listed in the *eCos Reference Manual*.

Editing an eCos Savefile

The eCos configuration information is held in a single savefile, typically `ecos.ecc`, which can be generated by either the GUI configuration tool or by the command line **ecosconfig** tool. The file normally exists at the top level of the build tree. It is a text file, allowing the various configurations options to be edited inside a suitable text editor or by other programs or scripts, as well as in the GUI config tool.

An eCos savefile is actually a script in the *Tcl* programming language, so any modifications to the file need to preserve Tcl syntax. For most configuration options, any modifications will be trivial and there is no need to worry about Tcl syntax. For example, changing a 1 to a 0 to disable an option. For more complicated options, for example `CYGDAT_UITRON_TASK_EXTERNS`, which involves some lines of C code, more care has to be taken. If an edited savefile is no longer a valid Tcl script then the configuration tools will be unable to read back the data for further processing, for example to generate a build tree. An outline of Tcl syntax is given below. One point worth noting here is that a line that begins with a “#” is usually a comment, and the bulk of an eCos savefile actually consists of such comments, to make it easier to edit.

Header

An eCos savefile begins with a header, which typically looks something like this:

```
# eCos saved configuration
```

```
# ---- commands -----
# This section contains information about the savefile format.
# It should not be edited. Any modifications made to this section
# may make it impossible for the configuration tools to read
# the savefile.

cdl_savefile_version 1;
cdl_savefile_command cdl_savefile_version {};
cdl_savefile_command cdl_savefile_command {};
cdl_savefile_command
cdl_configuration { description hardware template package };
cdl_savefile_command cdl_package { value_source user_value wizard_value inferred_value };
cdl_savefile_command cdl_component { value_source user_value wizard_value inferred_value };
cdl_savefile_command cdl_option { value_source user_value wizard_value inferred_value };
cdl_savefile_command cdl_interface { value_source user_value wizard_value inferred_value };
```

This section of the savefile is intended for use by the configuration system, and should not be edited. If this section is edited then the various configuration tools may no longer be able to read in the modified savefile.

Toplevel Section

The header is followed by a section that defines the configuration as a whole. A typical example would be:

```
# ---- toplevel -----
# This section defines the toplevel configuration object. The only
# values that can be changed are the name of the configuration and
# the description field. It is not possible to modify the target,
# the template or the set of packages simply by editing the lines
# below because these changes have wide-ranging effects. Instead
# the appropriate tools should be used to make such modifications.

cdl_configuration eCos {
description "" ;

# These fields should not be modified.
hardware    pid ;
template    uitron ;
package -hardware CYGPKG_HAL_ARM current ;
package -hardware CYGPKG_HAL_ARM_PID current ;
package -hardware CYGPKG_IO_SERIAL current ;
package -template CYGPKG_HAL current ;
package -template CYGPKG_IO current ;
package -template CYGPKG_INFRA current ;
package -template CYGPKG_KERNEL current ;
package -template CYGPKG_UITRON current ;
package -template CYGPKG_LIBC current ;
package -template CYGPKG_LIBM current ;
package -template CYGPKG_DEVICES_WALLCLOCK current ;
package -template CYGPKG_ERROR current ;
};
```

This section allows the configuration tools to reload the various packages that make up the configuration. Most of the information should not be edited. If it is necessary to add a new package or to remove an existing one then the appropriate tools should be used for this, for example:

```
$ ecosconfig remove CYGPKG_LIBM
```

There are two fields which can be edited. Configurations have a name; in this case eCos. They can also have a description, which is some arbitrary text. The configuration tools do not make use of these fields, they exist so that users can store additional information about a configuration.

Conflicts Section

The toplevel section is followed by details of all the conflicts (if any) in the configuration, for example:

```
# ---- conflicts -----
# There are 2 conflicts.
#
# option CYGNUM_LIBC_TIME_DST_DEFAULT_OFFSET
#   Property LegalValues
#   Illegal current value 100000
#   Legal values are: -90000 to 90000
#
# option CYGSEM_LIBC_TIME_CLOCK_WORKING
#   Property Requires
#   Requires constraint not satisfied: CYGFUN_KERNEL_THREADS_TIMER
```

When editing a configuration you may end up with something that is invalid. Any problems in the configuration will be reported in the conflicts section. In this case there are two conflicts. The option `CYGNUM_LIBC_TIME_DST_DEFAULT_OFFSET` has been given an illegal value: typically this would be fixed by searching for the definition of that option later on in the savefile and modifying the value. The second conflict is more interesting, an unsatisfied *requires* constraint. Configuration options are not independent: disabling some functionality in, say, the kernel, can have an impact elsewhere; in this case the C library. The various dependencies between the options are specified by the component developers and checked by the configuration system. In this case there are two obvious ways in which the conflict could be resolved: re-enabling `CYGFUN_KERNEL_THREADS_TIMER`, or disabling `CYGSEM_LIBC_TIME_CLOCK_WORKING`. Both of these options will be listed later on in the file.

Some care has to be taken when modifying configuration options, to avoid introducing new conflict. For instance it is possible that there might be other options in the system which have a dependency on `CYGSEM_LIBC_TIME_CLOCK_WORKING`, so disabling that option may not be the best way to resolve the conflict. Details of all such dependencies are provided in the appropriate places in the savefile.

It is not absolutely required that a configuration be conflict-free before generating a build tree and building eCos. It is up to the developers of each component to decide what would happen if an attempt is made to build eCos while there are still conflicts. In serious cases there is likely to be a compile-time failure, or possibly a link-time failure. In less serious cases the system may build happily and the application can be linked with the resulting library, but the component may not quite function as intended - although it may still be good enough for the specific needs of the application. It is also possible that everything builds and links, but once in a while the system will unaccountably crash. Using a configuration that still has conflicts is done entirely at the user's risk.

Data Section

The bulk of the savefile lists the various packages, components, and options, including their values and the various dependencies. A number of global options come first, especially those related to the build process such as compiler flags. These are followed by the various packages, and the components and options within those packages, in order.

Packages, components and options are organized in a hierarchy. If a particular component is disabled then all options and sub-components below it will be inactive: any changes made to these will have no effect. The savefile contains information about the hierarchy in the form of comments, for example:

```
cdl_package CYGPKG_KERNEL ...
# >
cdl_component CYGPKG_KERNEL_EXCEPTIONS ...
# >
cdl_option CYGSEM_KERNEL_EXCEPTIONS_DECODE ...
cdl_option CYGSEM_KERNEL_EXCEPTIONS_GLOBAL ...
# <
cdl_component CYGPKG_KERNEL_SCHED ...
# >
cdl_option CYGSEM_KERNEL_SCHED_MLQUEUE ...
cdl_option CYGSEM_KERNEL_SCHED_BITMAP ...
# <
# <
```

This corresponds to the following hierarchy:

```
CYGPKG_KERNEL
  CYGPKG_KERNEL_EXCEPTIONS
    CYGSEM_KERNEL_EXCEPTIONS_DECODE
    CYGSEM_KERNEL_EXCEPTIONS_GLOBAL
  CYGPKG_KERNEL_SCHED
    CYGSEM_KERNEL_SCHED_MLQUEUE
    CYGSEM_KERNEL_SCHED_BITMAP
```

Providing the hierarchy information in this way allows programs or scripts to analyze the savefile and readily determine the hierarchy. It could also be used by a sufficiently powerful editor to support structured editing of eCos savefiles. The information is not used by the configuration tools themselves since they obtain the hierarchy from the original CDL scripts.

Each configurable entity is preceded by a comment, of the following form:

```
# Kernel schedulers
# doc: ref/ecos-ref/ecos-kernel-overview.html#THE-SCHEDULER
# The eCos kernel provides a choice of schedulers. In addition
# there are a number of configuration options to control the
# detailed behaviour of these schedulers.
cdl_component CYGPKG_KERNEL_SCHED {
...
};
```

This provides a short textual alias `Kernel schedulers` for the component. If online documentation is available for the configurable entity then this will come next. Finally there is a short description of the entity as a whole. All this information is provided by the component developers.

Each configurable entity takes the form:

```
<type> <name> {
    <data>
};
```

Configurable entities may not be active. This can be either because the parent is disabled or inactive, or because there are one or more *active_if* properties. Modifying the value of an inactive entity has no effect on the configuration, so this information is provided first:

```
cdl_option CYGSEM_KERNEL_EXCEPTIONS_DECODE {
# This option is not active
# The parent CYGPKG_KERNEL_EXCEPTIONS is disabled
...
};

...

cdl_option CYGIMP_IDLE_THREAD_YIELD {
# This option is not active
# ActiveIf constraint: (CYGNUM_KERNEL_SCHED_PRIORITIES == 1)
#     CYGNUM_KERNEL_SCHED_PRIORITIES == 32
# --> 0
...
};
```

For `CYGIMP_IDLE_THREAD_YIELD` the savefile lists the expression that must be satisfied if the option is to be active, followed by the current value of all entities that are referenced in the expression, and finally the result of evaluating that expression.

Not all options are directly modifiable in the savefile. First, the value of packages (which is the version of that package loaded into the configuration) cannot be modified here.

```
cdl_package CYGPKG_KERNEL {
# Packages cannot be added or removed, nor can their version be changed,
# simply by editing their value. Instead the appropriate configuration
# should be used to perform these actions.
...
};
```

The version of a package can be changed using e.g.:

```
$ ecosconfig version 1.3 CYGPKG_KERNEL
```

Even though a package's value cannot be modified here, it is still important for the savefile to contain the details. In particular packages may impose constraints on other configurable entities and may be referenced by other configurable entities. Also it would be difficult to understand or extract the configuration's hierarchy if the packages were not listed in the appropriate places in the savefile.

Some components (or, conceivably, options) do not have any associated data. Typically they serve only to introduce another level in the hierarchy, which can be useful in the context of the GUI configuration tool.

```
cdl_component CYGPKG_HAL_COMMON_INTERRUPTS {
# There is no associated value.
};
```

Other components or options have a calculated value. These are not user-modifiable, but typically the value will depend on other options which can be modified. Such calculated options can be useful when controlling what gets built or what ends up in the generated configuration header files. A calculated value may also effect other parts of the configuration, for instance, via a *requires* constraint.

```
cdl_option BUFSIZ {
# Calculated value: CYGSEM_LIBC_STDIO_WANT_BUFFERED_IO ? CYGNUM_LIBC_STDIO_BUFSIZE : 0
#     CYGSEM_LIBC_STDIO_WANT_BUFFERED_IO == 1
#     CYGNUM_LIBC_STDIO_BUFSIZE == 256
# Current_value: 256
};
```

A special type of calculated value is the *interface*. The value of an interface is the number of active and enabled options which *implement* that interface. Again the value of an interface cannot be modified directly; only by modifying the options which implement the interface. However, an interface can be referenced by other parts of the configuration.

```
cdl_interface CYGINT_KERNEL_SCHEDULER {
# Implemented by CYGSEM_KERNEL_SCHED_MLQUEUE, active, enabled
# Implemented by CYGSEM_KERNEL_SCHED_BITMAP, active, disabled
# This value cannot be modified here.
# Current_value: 1
# Requires: 1 == CYGINT_KERNEL_SCHEDULER
#     CYGINT_KERNEL_SCHEDULER == 1
#     --> 1

# The following properties are affected by this value
# interface CYGINT_KERNEL_SCHEDULER
#     Requires: 1 == CYGINT_KERNEL_SCHEDULER
};
```

If the configurable entity is modifiable then there will be lines like the following:

```
cdl_option CYGSEM_KERNEL_SCHED_MLQUEUE {
...
# Flavor: bool
# No user value, uncomment the following line to provide one.
# user_value 1
# value_source default
# Default value: 1
...
};
```

Configurable entities can have one of four different flavors: none, bool, data and booldata. Flavor none indicates that there is no data associated with the entity, typically it just acts as a placeholder in the overall hierarchy. Flavor bool is the most common, it is a simple yes-or-no choice. Flavor data is for more complicated configuration choices,

for instance the size of an array or the name of a device. Flavor booldata is a combination of bool and data: the option can be enabled or disabled, and there is some additional data associated with the option as well.

In the above example the user has not modified this particular option, as indicated by the comment and by the commented-out `user_value` line. To disable this option the file should be edited to:

```
cdl_option CYGSEM_KERNEL_SCHED_MLQUEUE {
...
# Flavor: bool
# No user value, uncomment the following line to provide one.
user_value 0
# value_source default
# Default value: 1
...
}
```

The comment preceding the `user_value 0` line can be removed if desired, otherwise it will be removed automatically the next time the file is read and updated by the configuration tools.

Much the same process should be used for options with the data flavor, for example:

```
cdl_option CYGNUM_LIBC_TIME_DST_DEFAULT_OFFSET {
# Flavor: data
# No user value, uncomment the following line to provide one.
# user_value 3600
# value_source default
# Default value: 3600
# Legal values: -90000 to 90000
};
```

can be changed to:

```
cdl_option CYGNUM_LIBC_TIME_DST_DEFAULT_OFFSET {
# Flavor: data
user_value 7200
# value_source default
# Default value: 3600
# Legal values: -90000 to 90000 };
```

Note that the original text provides the default value in the `user_value` comment, on the assumption that the desired new value is likely to be similar to the default value. The `value_source` comment does not need to be updated, it will be fixed up if the savefile is fed back into the configuration system and regenerated.

For options with the booldata flavor, the `user_value` line needs take two arguments. The first argument is for the boolean part, the second for the data part. For example:

```
cdl_component CYGNUM_LIBM_COMPATIBILITY {
# Flavor: booldata
# No user value, uncomment the following line to provide one.
# user_value 1 POSIX
# value_source default
# Default value: 1 POSIX
# Legal values:  "POSIX" "IEEE" "XOPEN" "SVID"
```

```
...
};
```

could be changed to:

```
cdl_component CYGNUM_LIBM_COMPATIBILITY {
# Flavor: booldata
user_value 1 IEEE
# value_source default
# Default value: 1 POSIX
# Legal values:  "POSIX" "IEEE" "XOPEN" "SVID"
...
};
```

or alternatively, if the whole component should be disabled, to:

```
cdl_component CYGNUM_LIBM_COMPATIBILITY {
# Flavor: booldata
user_value 0 POSIX
# value_source default
# Default value: 1 POSIX
# Legal values:  "POSIX" "IEEE" "XOPEN" "SVID"
...
};
```

Some options take values that span multiple lines. An example would be:

```
cdl_option CYGDAT_UITRON_MEMPOOLVAR_INITIALIZERS {
# Flavor: data
# No user value, uncomment the following line to provide one.
# user_value \
# "CYG_UIT_MEMPOOLVAR( vpool1, 2000 ), \
#   CYG_UIT_MEMPOOLVAR( vpool2, 2000 ), \
#   CYG_UIT_MEMPOOLVAR( vpool3, 2000 ),"
# value_source default
# Default value: \
#   "CYG_UIT_MEMPOOLVAR( vpool1, 2000 ), \
#     CYG_UIT_MEMPOOLVAR( vpool2, 2000 ), \
#     CYG_UIT_MEMPOOLVAR( vpool3, 2000 ),"
};
```

Setting a user value for this option involves uncommenting and modifying all relevant lines, for example:

```
cdl_option CYGDAT_UITRON_MEMPOOLVAR_INITIALIZERS {
# Flavor: data
user_value \
"CYG_UIT_MEMPOOLVAR( vpool1, 4000 ), \
CYG_UIT_MEMPOOLVAR( vpool2, 4000 ),"
# value_source default
# Default value: \
#   "CYG_UIT_MEMPOOLVAR( vpool1, 2000 ), \
#     CYG_UIT_MEMPOOLVAR( vpool2, 2000 ), \
#     CYG_UIT_MEMPOOLVAR( vpool3, 2000 ),"
```

```
};
```

In such cases appropriate care has to be taken to preserve Tcl syntax, as discussed below.

The configuration system has the ability to keep track of several different values for any given option. All options start off with a default value, in other words their value source is set to `default`. If a configuration involves conflicts and the configuration system's inference engine is allowed to resolve these automatically then it may provide an inferred value instead, for example:

```
cdl_option CYGMFN_KERNEL_SYNCH_CONDVVAR_TIMED_WAIT {
# Flavor: bool
# No user value, uncomment the following line to provide one.
# user_value 1
# The inferred value should not be edited directly.
inferred_value 0
# value_source inferred
# Default value: 1
...
};
```

Inferred values are calculated by the configuration system and should not be edited by the user. If the inferred value is not correct then a user value should be substituted instead:

```
cdl_option CYGMFN_KERNEL_SYNCH_CONDVVAR_TIMED_WAIT {
# Flavor: bool
user_value 1
# The inferred value should not be edited directly.
inferred_value 0
# value_source inferred
# Default value: 1
...
};
```

The inference engine will not override a user value with an inferred one. Making a change like this may well re-introduce a conflict, since the inferred value was only calculated to resolve a conflict. Another run of the inference engine may find a different and more acceptable way of resolving the conflict, but this is not guaranteed and it may be up to the user to examine the various dependencies and work out an acceptable solution.

Inferred values are listed in the savefile because the exact inferred value may depend on the order in which changes were made and conflicts were resolved. If the inferred values were absent then it is possible that reloading a savefile would not exactly restore the configuration. Default values on the other hand are entirely deterministic so there is no actual need for the values to be listed in the savefile. However, the default value can be very useful information so it is provided in a comment.

Occasionally the user will want to do some experimentation, and temporarily switch an option from a user value back to a default or inferred one to see what the effect would be. This could be achieved by simply commenting out the user value. For instance, if the current savefile contains:

```
cdl_option CYGMFN_KERNEL_SYNCH_CONDVVAR_TIMED_WAIT {
# Flavor: bool
user_value 1
# The inferred value should not be edited directly.
inferred_value 0
```

```
# value_source user
# Default value: 1
...
};
```

then the inferred value could be restored by commenting out or removing the `user_value` line:

```
cdl_option CYGMFN_KERNEL_SYNC_CONDVAR_TIMED_WAIT {
# Flavor: bool
# user_value 1
# The inferred value should not be edited directly.
inferred_value 0
# value_source user
# Default value: 1
...
};
```

This is fine for simple values. However if the value is complicated then there is a problem: commenting out the `user_value` line or lines means that the user value becomes invisible to the configuration system, so if the savefile is loaded and then regenerated the information will be lost. An alternative approach is to keep the `user_value` but explicitly set the `value_source` line, for example:

```
cdl_option CYGMFN_KERNEL_SYNC_CONDVAR_TIMED_WAIT {
# Flavor: bool
user_value 1
# The inferred value should not be edited directly.
inferred_value 0
value_source inferred
# Default value: 1
...
};
```

In this case the configuration system will use the inferred value for the purposes of dependency analysis etc., even though a user value is present. To restore the user value the `value_source` line can be commented out again. If there is no explicit `value_source` then the configuration system will just use the highest priority one: the user value if it exists; otherwise the inferred value if it exists; otherwise the default value which always exists.

The default value for an option can be a simple constant, or it can be an expression involving other options. In the latter case the expression will be listed, together with the values for all options referenced in the expression and the current result of evaluating that expression. This is for informational purposes only, the default value is always recalculated deterministically when loading in a savefile.

```
cdl_option CYGBLD_GLOBAL_COMMAND_PREFIX {
# Flavor: data
# No user value, uncomment the following line to provide one.
# user_value arm-elf
# value_source default
# Default value: CYGHWL_THUMB ? "thumb-elf" : "arm-elf"
#     CYGHWL_THUMB == 0
#     --> arm-elf
};
```

For options with the data or booldata flavor, there are likely to be constraints on the possible values. If the value is supposed to be a number in a given range and a user value of “hello world” is provided instead then there are likely to be compile-time failures. Component developers can specify constraints on the legal values, and these will be listed in the savefile.

```
cdl_option X_TLOSS {
# Flavor: data
# No user value, uncomment the following line to provide one.
# user_value 1.41484755040569E+16
# value_source default
# Default value: 1.41484755040569E+16
# Legal values: 1 to 1e308
};

cdl_component CYGNUM_LIBM_COMPATIBILITY {
# Flavor: booldata
# No user value, uncomment the following line to provide one.
# user_value 1 POSIX
# value_source default
# Default value: 1 POSIX
# Legal values:  "POSIX" "IEEE" "XOPEN" "SVID"
...
};
```

In some cases the legal values list may be an expression involving other options. If so then the current values of the referenced options will be listed, together with the result of evaluating the list expression, just as for default value expressions.

If an illegal value is provided then this will result in a conflict, listed in the conflicts section of the savefile. For more complicated options a simple legal values list is not sufficient to allow the current value to be validated, and the configuration system will be unable to flag conflicts. This issue will be addressed in future releases of the configuration system.

Following the value-related fields for a given option, any *requires* constraints belonging to this option will be listed. These constraints are only effective if the option is active and, for bool and booldata flavors, enabled. If some aspect of eCos functionality is inactive or disabled then it cannot impose any constraints on the rest of the system. As usual, the full expression will be listed followed by the current values of all options that are referenced and the result of evaluating the expression:

```
cdl_option CYGSEM_LIBC_TIME_TIME_WORKING {
...
# Requires: CYGPKG_DEVICES_WALLCLOCK
#          CYGPKG_DEVICES_WALLCLOCK == current
#          --> 1
};
```

When modifying the value of an option it is useful to know not only what constraints the option imposes on the rest of the system but also what other options in the system depend in some way on this one. The savefile provides this information:

```
cdl_option CYGFUN_KERNEL_THREADS_TIMER {
...
# The following properties are affected by this value
```

```
# option CYGMFN_KERNEL_SYNCH_CONDVAR_TIMED_WAIT
#   Requires: CYGFUN_KERNEL_THREADS_TIMER
# option CYGIMP_UITRON_STRICT_CONFORMANCE
#   Requires: CYGFUN_KERNEL_THREADS_TIMER
# option CYGSEM_LIBC_TIME_CLOCK_WORKING
#   Requires: CYGFUN_KERNEL_THREADS_TIMER
};
```

Tcl Syntax

eCos savefiles are implemented as Tcl scripts, and are read in by running the data through a standard Tcl interpreter that has been extended with a small number of additional commands such as `cdl_option` and `cdl_configuration`. In many cases this is an implementation detail that can be safely ignored while editing a savefile: simply replacing a 1 with a 0 to disable some functionality is not going to affect whether or not the savefile is still a valid Tcl script and can be processed by a Tcl interpreter. However, there are more complicated cases where an understanding of Tcl syntax is at least desirable, for example:

```
cdl_option CYGDAT_UITRON_MEMPOOLVAR_EXTERNS {
    # Flavor: data
    user_value \
        "static char vpool1\[ 2000 \], \\\
        vpool2\[ 2000 \], \\\
        vpool3\[ 2000 \];"
    # value_source default
    # Default value: \
        #   "static char vpool1\[ 2000 \], \\\
        #   vpool2\[ 2000 \], \\\
        #   vpool3\[ 2000 \];"
};
```

The backslash at the end of the `user_value` line is processed by the Tcl interpreter as a line continuation character. The quote marks around the user data are also interpreted by the Tcl interpreter and serve to turn the entire data field into a single argument. The backslashes preceding the opening and closing square brackets prevent the Tcl interpreter from treating these characters specially, otherwise there would be an attempt at *command substitution* as described below. The double backslashes at the end of each line of the data will be turned into a single backslash by the Tcl interpreter, rather than escaping the newline character, so that the actual data seen by the configuration system is:

```
static char vpool1[ 2000 ], \
    vpool2[ 2000 ], \
    vpool3[ 2000 ];
```

This is of course the data that should end up in the μ ITRON configuration header file. The opening and closing braces surrounding the entire body of the option data are also significant and cause this body to be passed as a single argument to the **`cdl_option`** command. The closing semicolon is optional in this example, but provides a small amount of additional robustness if the savefile is edited such that it is no longer a valid Tcl script. If the data contained any `$` characters then these would have to be treated specially as well, via a backslash escape.

In spite of what all the above might seem to suggest, Tcl is actually a very simple yet powerful scripting language: the syntax is defined by just eleven rules. On occasion this simplicity means that Tcl's behaviour is subtly different from other languages, which can confuse newcomers.

When the Tcl interpreter is passed some data such as `puts Hello`, it splits this data into a command and its arguments. The command will be terminated by a newline or by a semicolon, unless one of the quoting mechanisms is used. The command and each of its arguments are separated by white space. So in the following example:

```
puts Hello
set x 42
```

will result in two separate commands being executed. The first command is `puts` and is passed a single argument, `Hello`. The second command is `set` and is passed two arguments, `x` and `42`. The intervening newline character serves to terminate the first command, and a semi-colon separator could be used instead:

```
puts Hello;set x 42
```

Any white space surrounding the semicolon is just ignored because it does not serve to separate arguments.

Now consider the following:

```
set x Hello world
```

This is not valid Tcl. It is an attempt to invoke the `set` command with three arguments: `x`, `Hello`, and `world`. The `set` only takes two arguments, a variable name and a value, so it is necessary to combine the data into a single argument by quoting:

```
set x "Hello world"
```

When the Tcl interpreter encounters the first quote character it treats all subsequent data up to but not including the closing quote as part of the current argument. The quote marks are removed by the interpreter, so the second argument passed to the `set` command is just `Hello world` without the quote characters. This can be significant in the context of eCos savefiles. For instance, consider the following configuration option:

```
cdl_option CYGDAT_LIBC_STDIO_DEFAULT_CONSOLE {
# Flavor: data
# No user value, uncomment the following line to provide one.
# user_value "\"/dev/ttydiag\""
# value_source default
# Default value: "\"/dev/ttydiag\""
};
```

The desired value of the configuration option should be a valid C string, complete with quote characters. If the savefile was edited to:

```
cdl_option CYGDAT_LIBC_STDIO_DEFAULT_CONSOLE {
# Flavor: data
user_value "/dev/ttydiag"
# value_source default
# Default value: "\"/dev/ttydiag\""
};
```

then the Tcl interpreter would remove the quote marks when the savefile is read back in, so the option's value would not have any quote marks and would not be a valid C string. The configuration system is not yet able to perform the required validation so the following `#define` would be generated in the configuration header file:

```
#define CYGDAT_LIBC_STDIO_DEFAULT_CONSOLE /dev/ttydiag
```

This is likely to cause a compile-time failure when building eCos.

A quoted argument continues until the closing quote character is encountered, which means that it can span multiple lines. This can also be encountered in eCos savefiles, for instance, in the `CYGDAT_UITRON_MEMPOOLVAR_EXTERNS` example mentioned earlier. Newline or semicolon characters do not terminate the current command in such cases.

The Tcl interpreter supports much the same forms of backslash substitution as other common programming languages. Some backslash sequences such as `\n` will be replaced by the appropriate character. The sequence `\\` will be replaced by a single backslash. A backslash at the very end of a line will cause that backslash, the newline character, and any white space at the start of the next line to be replaced by a single space. Hence the following two Tcl commands are equivalent:

```
puts "Hello\nworld\n"
puts \
"Hello
world
"
```

In addition to quote and backslash characters, the Tcl interpreter treats square brackets, the `$` character, and braces specially. Square brackets are used for command substitution, for example:

```
puts "The answer is [expr 6 * 9]"
```

When the Tcl interpreter encounters the square brackets it will treat the contents as another command that should be executed first, and the result of executing that is used when continuing to process the script. In this case the Tcl interpreter will execute the command `expr 6 * 9`, yielding a result of 54, and then the Tcl interpreter will execute `puts "The answer is 54"`. It should be noted that the interpreter contains only one level of substitution: if the result of performing command substitution performs further special characters such as square brackets then these will not be treated specially.

Command line substitution is very unlikely to prove useful in the context of an eCos savefile, but it is part of the Tcl language and hence cannot be easily suppressed while reading in a savefile. As a result care has to be taken when savefile data involves square brackets. Consider the following:

```
cdl_option CYGDAT_UITRON_MEMPOOLFIXED_EXTERNS {
    ...
    user_value \
"static char fpool1[ 2000 ],
fpool2[ 2000 ];"
    ...
};
```

The Tcl interpreter will interpret the square brackets as an attempt at command substitution and hence it will attempt to execute the command `2000` with no arguments. No such command is defined by the Tcl language or by the savefile-related extensions provided by the configuration system, so this will result in an error when an attempt is made to read back the savefile. Instead it is necessary to backslash-escape the square brackets and thus suppress command substitution:

```
cdl_option CYGDAT_UITRON_MEMPOOLFIXED_EXTERNS {
    ...
    user_value \
"static char fpool1\[ 2000 \],
fpool2\[ 2000 \];"
    ...
};
```



```
};
```

Similarly the `$` character is used in Tcl scripts to perform variable substitution:

```
set x [expr 6 * 9]
puts "The answer is $x"
```

Variable substitution, like command substitution, is very unlikely to prove useful in the context of an eCos savefile. Should it be necessary to have a `$` character in configuration data then again a backslash escape needs to be used.

```
cdl_option FOODAT_MONITOR_PROMPT {
    ...
    user_value "\\$ "
    ...
};
```

Braces are used to collect a sequence of characters into a single argument, just like quotes. The difference is that variable, command and backslash substitution do not occur inside braces (with the sole exception of backslash substitution at the end of a line). So, for example, the `CYGDAT_UITRON_MEMPOOL_EXTERNFIXED_EXTERNS` value could be written as:

```
cdl_option CYGDAT_UITRON_MEMPOOLFIXED_EXTERNS {
    ...
    user_value \
{static char fpool1[ 2000 ],
 fpool2[ 2000 ];}
    ...
};
```

The configuration system does not use this when generating savefiles because for simple edits of a savefile by inexperienced users the use of brace characters is likely to be a little bit more confusing than the use of quotes.

At this stage it is worth noting that the basic format of each configuration option in the savefile makes use of braces:

```
cdl_option <name> {
    ...
};
```

The configuration system extends the Tcl language with a small number of additional commands such as `cdl_option`. These commands take two arguments, a name and a body, where the body consists of all the text between the braces. First a check is made that the specified option is actually present in the configuration. Then the body is executed in a recursive invocation of the Tcl interpreter, this time with additional commands such as `user_value` and `value_source`. If, after editing, the braces are not correctly matched up then the savefile will no longer be a valid Tcl script and errors will be reported when the savefile is loaded again.

Comments in Tcl scripts are introduced by a hash character `#`. However, a hash character only introduces a comment if it occurs where a command is expected. Consider the following:

```
# This is a comment
puts "Hello" # world
```

The first line is a valid comment, since the hash character occurs right at the start where a command name is expected. The second line does not contain a comment. Instead it is an attempt to invoke the `puts` command with

three arguments: `Hello`, `#` and `world`. These are not valid arguments for the `puts` command so an error will be raised.

If the second line was rewritten as:

```
puts "Hello"; # world
```

then this is a valid Tcl script. The semicolon identifies the end of the current command, so the hash character occurs at a point where the next command would start and hence it is interpreted as the start of a comment.

This handling of comments can lead to subtle behaviour. Consider the following:

```
cdl_option WHATEVER {  
    # This is a comment }  
    user_value 42  
    ...  
}
```

Consider the way the Tcl interpreter processes this. The command name and the first argument do not pose any special difficulties. The opening brace is interpreted as the start of the next argument, which continues until a closing brace is encountered. In this case the closing brace occurs on the second line, so the second argument passed to `cdl_option` is `\n # This is a comment`. This second argument is processed in a recursive invocation of the Tcl interpreter and does not contain any commands, just a comment. Toplevel savefile processing then resumes, and the next command that is encountered is `user_value`. Since the relevant savefile code is not currently processing a configuration option this is an error. Later on the Tcl interpreter would encounter a closing brace by itself, which is also an error. Fortunately this sequence of events is very unlikely to occur when editing generated savefiles.

This should be sufficient information about Tcl to allow for safe editing of eCos savefiles. Further information is available from a wide variety of sources, for example the book *Tcl and the Tk Toolkit* by John K Ousterhout.

Editing the Sources

For many users, controlling the packages and manipulating the available configuration options will be sufficient to create an embedded operating system that meets the application's requirements. However, since eCos is shipped entirely in source form, it is possible to go further when necessary: you can edit the eCos sources themselves. This requires some understanding of the way the eCos build system works.

The most obvious place to edit the source code is directly in the component repository. For example, you could edit the file `kernel/<version>/src/sync/mutex.cxx` to change the way kernel mutexes work, or possibly just to add some extra diagnostics or assertions. Once the file has been edited, it is possible to invoke **make** at the top level of the build tree and the target library will be rebuilt as required. A small optimization is possible: the build tree is largely a mirror of the component repository, so it too will contain a subdirectory `kernel/<version>`; if `make` is invoked in this directory then it will only check for changes to the kernel sources, which is a bit more efficient than checking for changes throughout the component repository.

Editing a file in the component repository is fine if this tree is used for only one eCos configuration. If the repository is used for several different configurations, however, and especially if it is shared by multiple users, then making what may be experimental changes to the master sources would be a bad idea. The build system provides an alternative. It is possible to make a copy of the file in the build tree, in other words copy `mutex.cxx` from the `kernel/<version>/src/sync` directory in the component repository to `kernel/<version>/src/sync` in the

build tree, and edit the file in the build tree. When **make** is invoked it will pick up local copies of any of the sources in preference to the master versions in the component repository. Once you have finished modifying the eCos sources you can install the final version back in the component repository. If the changes were temporary in nature and only served to aid the debugging process, then you can discard the modified version of the sources.

The situation is slightly more complicated for the header files that a package may export, such as the C library's `stdio.h` header file, which can be found in the directory `language/c/libc/<version>/include`. If such a header file is changed, either directly in the component repository or after copying it to the build tree, then **make** must be invoked at the top level of the build tree. In cases like this it is not safe to rebuild just the C library because other packages may depend on the contents of `stdio.h`.

Modifying the Memory Layout

Each eCos platform package is supplied with linker script fragments which describe the location of memory regions on the evaluation board and the location of memory sections within these regions. The correct linker script fragment is selected and included in the eCos linker script `target.ld` when eCos is built.

It is not necessary to modify the default memory layouts in order to start development with eCos. However, it will be necessary to edit a linker script fragment when the memory map of the evaluation board is changed. For example, if additional memory is added, the linker must be notified that the new memory is available for use. As a minimum, this would involve modifying the length of the corresponding memory region. Where the available memory is non-contiguous, it may be necessary to declare a new memory region and reassign certain linker output sections to the new region.

Linker script fragments and memory layout header files should be edited within the eCos install tree. They are located at `include/pkgconf/mlt_*.*`. Where multiple start-up types are in use, it will be necessary to edit multiple linker script fragments and header files. The information provided in the header file and the corresponding linker script fragment must always match. A typical linker script fragment is shown below:

Example 28-1. eCos linker script fragment

```
MEMORY
{
    rom : ORIGIN = 0x40000000, LENGTH = 0x80000
    ram : ORIGIN = 0x48000000, LENGTH = 0x200000
}

SECTIONS
{
    SECTIONS_BEGIN
    SECTION_rom_vectors (rom, 0x40000000, LMA_EQ_VMA)
    SECTION_text (rom, ALIGN (0x1), LMA_EQ_VMA)
    SECTION_fini (rom, ALIGN (0x1), LMA_EQ_VMA)
    SECTION_rodata (rom, ALIGN (0x1), LMA_EQ_VMA)
    SECTION_rodata1 (rom, ALIGN (0x1), LMA_EQ_VMA)
    SECTION_fixup (rom, ALIGN (0x1), LMA_EQ_VMA)
    SECTION_gcc_except_table (rom, ALIGN (0x1), LMA_EQ_VMA)
    SECTION_data (ram, 0x48000000, FOLLOWING (.gcc_except_table))
    SECTION_bss (ram, ALIGN (0x4), LMA_EQ_VMA)
    SECTIONS_END
}
```

The file consists of two blocks, the `MEMORY` block contains lines describing the address (`ORIGIN`) and the size (`LENGTH`) of each memory region. The `MEMORY` block is followed by the `SECTIONS` block which contains lines describing each of the linker output sections. Each section is represented by a macro call. The arguments of these macros are ordered as follows:

1. The memory region in which the section will finally reside.
2. The final address (`VMA`) of the section. This is expressed using one of the following forms:

n

at the absolute address specified by the unsigned integer *n*

`ALIGN (n)`

following the final location of the previous section with alignment to the next *n*-byte boundary

3. The initial address (`LMA`) of the section. This is expressed using one of the following forms:

`LMA_EQ_VMA`

the `LMA` equals the `VMA` (no relocation)

`AT (n)`

at the absolute address specified by the unsigned integer *n*

`FOLLOWING (.name)`

following the initial location of section *name*

In order to maintain compatibility with linker script fragments and header files exported by the eCos Configuration Tool, the use of other expressions within these files is not recommended.

Note that the names of the linker output sections will vary between target architectures. A description of these sections can be found in the specific GCC documentation for your architecture.

Chapter 29. Managing the Package Repository

A source distribution of eCos consists of a number of packages, such as the kernel, the C library, and the μ ITRON subsystems. These are individually versioned in the tree structure of the source code, to support distribution on a per-package basis and to support third party packages whose versioning systems might be different. The eCos Package Administration Tool is used to manage the installation and removal of packages from a variety of sources with potentially multiple versions.

The presence of the version information in the source tree structure might be a hindrance to the use of a separate source control system such as *CVS* or *SourceSafe*. To work in this way, you can rename all the version components to some common name (such as “current”) thus unifying the structure of source trees from distinct eCos releases.

The eCos build system will treat any such name as just another version of the package(s), and support building in exactly the same way. However, performing this rename invalidates any existing build trees that referred to the versioned source tree, so do the rename first, before any other work, and do a complete rebuild afterwards.

Package Installation

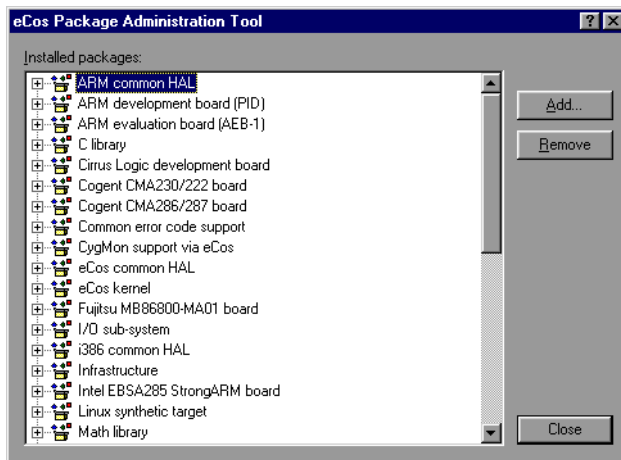
Package installation and removal is performed using the eCos Package Administration Tool. This tool is a Tcl script named **ecosadmin.tcl** which allows the user to add new eCos packages and new versions of existing packages to an eCos repository. Such packages must be distributed as a single file in the eCos package distribution format. Unwanted packages may also be removed from the repository using this tool. A graphical version of the tool is provided as part of the eCos Configuration Tool.

Using the Administration Tool

The graphical version of the eCos Package Administration Tool, provided as part of the eCos Configuration Tool, provides functions equivalent to the command-line version for those who prefer a Windows-based interface.

It may be invoked in one of two ways:

- from the start menu (by default Start->Programs-> eCos->Package Administration Tool)
- from the eCos Configuration Tool via the Tools->Administration menu item



The main window of the tool displays the packages which are currently installed in the form of a tree. The installed versions of each package may be examined by expanding the tree.

Packages may be added to the eCos repository by clicking on the *Add* button. The eCos package distribution file to be added is then selected via a *File Open* dialog box.

Packages may be removed by selecting a package in the tree and then clicking on the *Remove* button. If a package node is selected, all versions of the selected package will be removed. If a package version node is selected, only the selected version of the package will be removed.

Using the command line

The `ecosadmin.tcl` script is located in the base of the eCos repository. Use a command of the following form under versions of UNIX:

```
$ tclsh ecosadmin.tcl <command>
```

Under Windows, a command of the following form may be used at the Cygwin command line prompt:

```
$ cygtclsh80 ecosadmin.tcl <command>
```

The following commands are available:

add <file>

Adds the packages contained with the specified package distribution file to the eCos repository and updates the package database accordingly. By convention, eCos package distribution files are given the `.epk` suffix.

remove <package> [--version=<version>]

Removes the specified package from the eCos repository and updates the package database accordingly. Where the optional version qualifier is used, only the specified version of the package is removed.

list

Produces a list of the packages which are currently installed and their versions. The available templates and hardware targets are also listed.

Note that it is possible to remove critical packages such as the common HAL package using this tool. Users should take care to avoid such errors since core eCos packages may only be re-installed in the context of a complete re-installation of eCos.

Package Structure

The files in an installed eCos source tree are organized in a natural tree structure, grouping together files which work together into *Packages*. For example, the kernel files are all together in:

```
BASE_DIR/kernel/<version>/include/
BASE_DIR/kernel/<version>/src/
BASE_DIR/kernel/<version>/tests/
```

and μ ITRON compatibility layer files are in:

```
BASE_DIR/compat/uitron/<version>/include/
BASE_DIR/compat/uitron/<version>/src/
BASE_DIR/compat/uitron/<version>/tests/
```

The feature of these names which is of interest here is the *<version>* near the end. It may seem odd to place a version number deep in the path, rather than having something like *BASE_DIR/<version>/...everything...* or leaving it up to you to choose a different install-place when a new release of the system arrives.

There is a rationale for this organization: as indicated, the kernel and the μ ITRON compatibility subsystem are examples of software packages. For the first few releases of eCos, all the packages will move along in step, i.e. Release 1.3.x will feature Version 1.3.x of every package, and so forth. But in future, especially when third party packages become available, it is intended that the package be the unit of software distribution, so it will be possible to build a system from a selection of packages with different version numbers, and even differing versioning *schemes*. A Tcl script **ecosadmin.tcl** is provided in the eCos repository to manage the installation and removal of packages in this way.

Many users will have their own source code control system, version control system or equivalent, and will want to use it with eCos sources. In that case, since a new release of eCos comes with different pathnames for all the source files, a bit of work is necessary to import a new release into your source repository.

One way of handling the import is to rename all the version parts to some common name, for example “current”, and continue to work. “current” is suggested because **ecosconfig** recognizes it and places it first in any list of versions. In the future, we may provide a tool to help with this, or an option in the install wizard. Alternatively, in a POSIX shell environment (Linux or Cygwin on Windows) use the following command:

```
find . -name <version> -type d -printf 'mv %p %h/current\n' | sh
```

Having carried out such a renaming operation, your source tree will now look like this:

```
BASE_DIR/kernel/current/include/
BASE_DIR/kernel/current/src/
```

```
BASE_DIR/kernel/current/tests/  
...  
BASE_DIR/compat/uitron/current/include/  
BASE_DIR/compat/uitron/current/src/  
BASE_DIR/compat/uitron/current/tests/
```

which is a suitable format for import into your own source code control system. When you get a subsequent release of eCos, do the same thing and use your own source code control system to manage the new source base, by importing the new version from

```
NEW_BASE_DIR/kernel/current/include/
```

and so on.

The eCos build tool will now offer only the “current” version of each package; select this for the packages you wish to use.

Making such a change has implications for any build trees you already have in use. A configured build tree contains information about the selected packages and their selected versions. Changing the name of the “versioning” folder in the source tree invalidates this information, and in consequence it also invalidates any local configuration options you have set up in this build tree. So if you want to change the version information in the source tree, do it first, before investing any serious time in configuring and building your system. When you create a new build tree to deal with the new source layout, it will contain default settings for all the configuration options, just like the old build tree did before you configured it. You will need to redo that configuration work in the new tree.

Moving source code around also invalidates debugging information in any programs or libraries built from the old tree; these will need to be rebuilt.

VII. Appendixes

Appendix A. Target Setup

The following sections detail the setup of many of the targets supported by eCos.

Caution

This information is presented here only temporarily. It is intended that there will be separate documents detailing this information for each target in future releases. Consequently not much effort has been put into bringing the following documentation up to date -- much of it is obsolete, bogus or just plain wrong.

MN10300 stdevall1 Hardware Setup

The eCos Developer's Kit package comes with a pair of EPROMs which provide GDB support for the Matsushita MN10300 (AM31) series evaluation board using CygMon, the Cygnus ROM monitor. Images of these EPROMs are also provided at `BASE_DIR/loaders/mn10300-stdevall1/cygmon.bin`. The LSB EPROM (LROM) is installed to socket IC8 on the board and the MSB EPROM (UROM) is installed to socket IC9. Attention should be paid to the correct orientation of these EPROMs during installation.

The CygMon stubs allows communication with GDB by way of the serial port at connector CN2. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit, and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a standard RS232C serial cable (not a null modem cable). A gender changer may also be required.

MN10300 Architectural Simulator Setup

The MN10300 simulator is an architectural simulator for the Matsushita MN10300 that implements all features of the microprocessor necessary to run eCos. The current implementation provides accurate simulation of the instruction set, interrupt controller, timers, and serial I/O.

In this release, you can run the same eCos binaries in the simulator that can run on target hardware, if built for ROM start-up, with the exception of those that use the watchdog timer.

However, note that AM33 devices required to run eCos are not simulated; therefore you cannot run eCos binaries built for the AM33 under the simulator. For the AM33, the simulator is effectively an instruction-set only simulator.

To simplify connection to the simulator, you are advised to create a GDB macro by putting the following code in your personal GDB start-up file (`gdb.ini` on Windows and `.gdbinit` on UNIX).

```
define msim
  target sim --board=stdevall --memory-region 0x34004000,0x8

  rbreak cyg_test_exit
  rbreak cyg_assert_fail
end
```

You can then connect to the simulator by invoking the command

```
msim
```

on the command line:

```
(gdb) msim
```

You can achieve the same effect by typing out the macro's content on the command line if necessary.

AM33 STB Hardware Setup

The Matsushita AM33 STB System Reference Board may be used in two modes: via a JTAG debugger, or by means of a GDB stub ROM.

Use with GDB Stub ROM

The eCos Developer's Kit package comes with a ROM image which provides GDB support for the Matsushita(R) AM33 STB System Reference Board. To install the GDB stub ROM requires the use of the JTAG debugger and the Flash ROM programming code available from Matsushita. An image of this ROM is also provided at `loaders/am33-stb/gdbload.bin` under the root of your eCos installation.

Ensure that there is a Flash ROM card in MAIN MEMORY SLOT <0>. Follow the directions for programming a Flash ROM supplied with the programming software.

The final programming of the ROM will need to be done with a command similar to the following:

```
fdown "gdbload.bin",0x80000000,16,1
```

Once the ROM has been programmed, close down the JTAG debugger, turn the STB off, and disconnect the JTAG cable. Ensure that the hardware switches are in the following configuration:

U U D D D U D D

D = lower part of rocker switch pushed in

U = upper part of rocker switch pushed in

This is also the configuration required by the Flash programming code, so it should not be necessary to change these.

Restart the STB and the stub ROM will now be able to communicate with GDB. eCos programs should be built with RAM startup.

Programs can then be downloaded via a standard RS232 null modem serial cable connected to the SERIAL1 connector on the STB front panel (the AM33's serial port 0). This line is programmed to run at 38400 baud, 8 data bits, no parity and 1 stop bit (8-N-1) with no flow control. A gender changer may also be required. Diagnostic output will be output to GDB using the same connection.

This procedure also applies for programming ROM startup eCos programs into ROM, given a binary format image of the program from

```
mn10300-elf-objcopy.
```

Use with the JTAG debugger

To use eCos from the JTAG debugger, executables must be built with ROM startup and then downloaded via the JTAG debugger. For this to work there must be an SDRAM memory card in SUB MEMORY SLOT <0> and the hardware switches on the front panel set to the following:

D U D D D U D D

D = lower part of rocker switch pushed in

U = upper part of rocker switch pushed in

Connect the JTAG unit and run the debugger as described in the documentation that comes with it.

eCos executables should be renamed to have a “.out” extension and may then be loaded using the debugger's “l” or “lp” commands.

Diagnostic output generated by the program will be sent out of the AM33's serial port 0 which is connected to the SERIAL1 connector on the STB front panel. This line is programmed to run at 38400 baud, 8 data bits, no parity, and one stop bit (8-N-1) with no flow control. Connection to the host computer should be using a standard RS232 null modem serial cable. A gender changer may also be required.

Building the GDB stub ROM image

eCos comes with a pre-built GDB stub ROM image for the AM33-STB platform. This can be found at `loaders/am33-stb/gdbload.bin` relative to the eCos installation directory.

If necessary, the ROM image can be re-built as follows:

1. On Windows hosts, open a Bash session using *Start->Programs->Red Hat eCos->eCos Development Environment*
2. Create a build directory and cd into it
3. Run (all as one line):

```
cygtoolsh80 BASE_DIR/packages/pkgconf.tcl \
--target=mn10300_am33 --platform stb --startup rom \
--disable-kernel --disable-uitron --disable-libc --disable-libm \
--disable-io --disable-io_serial --disable-wallclock \
--disable-watchdog
```

where BASE_DIR is the path to the eCos installation directory.

4. Edit the configuration file `pkgconf/hal.h` in the build directory tree by ensuring the following configuration options are set as follows:

```
#define CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS
#define CYGDBG_HAL_DEBUG_GDB_BREAK_SUPPORT
#undef CYGDBG_HAL_DEBUG_GDB_CTRL_C_SUPPORT
#define CYGDBG_HAL_DEBUG_GDB_THREAD_SUPPORT
#define CYG_HAL_ROM_MONITOR
```

5. Run: `make`
6. Run: `make -C hal/common/current/current/src/stubrom`

7. The file `hal/common/current/src/stubrom` will be an ELF format executable of the ROM image. Use `mn10300-elf-objcopy` to convert this to the appropriate format for loading into the Matsushita FLASH ROM programmer, mode “binary” in this case:

```
$ mn10300-elf-objcopy -O binary hal/common/current/src/stubrom/ \
  stubrom stubrom.img
```

TX39 Hardware Setup

The eCos Developer’s Kit package comes with a pair of ROMs that provide GDB support for the Toshiba JMR-TX3904 RISC processor reference board by way of CygMon.

Images of these ROMs are also provided at `BASE_DIR/loaders/tx39-jmr3904/cygmon50.bin` and `BASE_DIR/loaders/tx39-jmr3904/cygmon66.bin` for 50 MHz and 66 MHz boards respectively. The ROMs are installed to sockets IC6 and IC7 on the memory daughterboard according to their labels. Attention should be paid to the correct orientation of these ROMs during installation.

The GDB stub allows communication with GDB using the serial port (channel C) at connector PJ1. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit, and 1 stop bit (8-N-1). No handshaking is employed. Connection to the host computer should be made using an RS232C null modem cable.

CygMon and eCos currently provide support for a 16Mbyte 60ns 72pin DRAM SIMM fitted to the PJ21 connector. Different size DRAMs may require changes in the value stored in the DCCR0 register. This value may be found near line 211 in `hal/mips/arch/<version>/src/vectors.S` in eCos, and near line 99 in `libstub/mips/tx39jmr/tx39jmr-power.S` in CygMon. eCos does not currently use the DRAM for any purpose itself, so it is entirely available for application use.

TX39 Architectural Simulator Setup

The TX39 simulator is an architectural simulator which implements all the features of the Toshiba TX39 needed to run eCos. The current implementation provides accurate simulation of the instruction set, interrupt controller, and timers, as well as having generic support for diagnostic output, serial I/O, and exceptions.

In this release, you can run the same eCos binaries in the simulator that can run on target hardware, if it is built for ROM start-up.

To simplify connection to the simulator, you are advised to create a GDB macro by putting the following code in your personal GDB start-up file (`gdb.ini` on Windows and `.gdbinit` on UNIX).

```
define tsim
  target sim --board=jmr3904pal --memory-region 0xffff8000,0x900 \
    --memory-region 0xffffe000,0x4 \
    --memory-region 0xb2100000,0x4
  rbreak cyg_test_exit
  rbreak cyg_assert_fail
end
```

You can then connect to the simulator by invoking the command **tsim** on the command line:

```
(gdb) tsim
```

You can achieve the same effect by typing out the macro's content on the command line if necessary.

TX49 Hardware Setup

The eCos installation CD contains a copy of the eCos GDB stubs in SREC format which must be programmed into the board's FLASH memory.

Preparing the GDB stubs

These stub preparation steps are not strictly necessary as the eCos distribution ships with pre-compiled stubs in the directory `loaders/tx49-ref4955` relative to the installation root.

Building the GDB stub image with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the TX49 REF4955 hardware.
3. While still displaying the *Build->Templates* dialog box, select the stubs package template to build a GDB stub. Click *OK*.
4. Build eCos stubs using *Build->Library*.
5. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub images have the prefix `gdb_module`.

Building the GDB stub image with ecosconfig

1. Make an empty directory to contain the build tree, and `cd` into it.
2. To build a GDB stub ROM image, enter the command:


```
$ ecosconfig new ref4955 stubs
```
3. Enter the commands:


```
$ ecosconfig tree
$ make
```
4. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub images have the prefix `gdb_module`.

Installing GDB stubs into FLASH

Boot into the board's firmware in little-endian mode:

Set the switches like this:

SW1: 10000000 (first lever up, the rest down) SW2: 10000010

Connect serial cable on the lower connector, configure terminal emulator for 38400, 8-N-1.

When booting the board, you should get this prompt:

```
HCP5 rev 0.9B .  
HCP5?
```

Select o (option), a (FLASH) and b (boot write). You should see this:

```
Boot ROM Write  
ROM address-ffffffffbd000000, Boot Bus-[32bit]  
ID2 0 4 ffffffff002ad40  
zzz SS-40000 IV-1 CS-20000 CC-2  
Flash ROM-[28F640J5], [16bit chip] * 2 * 1  
Block size-00040000 count-64  
ROM adr ffffffffbd000000-fffffffbfe000000 mask-00fc0000  
Send Srecord file sa=00000000 size=fffffffffffffff  
ra=fffffffe000000
```

Now send the stub SREC data down to the board using the terminal emulator's 'send ASCII' (or similar) functionality.

Red Hat has experienced some sensitivity to how fast the data is written to the board. Under Windows you should configure Minicom to use a line delay of 100 milliseconds. Under Linux, use the `slow_cat.tcl` script:

```
% cd BASE_DIR/packages/hal/mips/ref4955/<version>/misc  
% slow_cat.tcl < [path]/gdb_module.srec > /dev/ttyS0
```

Power off the board, and change it to boot the GDB stubs in big-endian mode by setting the switches like this:

SW1: 00000000 (all levers down) SW2: 10001010

The GDB stubs allow communication with GDB using the serial port at connector PJ7A (lower connector). The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a straight through serial cable.

VR4300 Hardware Setup

The eCos Developer's Kit package comes with an EPROM which provides GDB support for the NEC VRC4373 evaluation board. An image of this EPROM is also provided at `loaders/vr4300-vrc4373/gdbload.bin` under the root of your eCos installation.

The EPROM is installed to socket U12 on the board. Attention should be paid to the correct orientation of the EPROM during installation. Only replace the board's existing ROM using a proper PLCC extraction tool, as the socket would otherwise risk getting damaged.

The GDB stub in the EPROM allows communication with GDB using the serial port at connector J1. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a straight-through serial cable.

VRC4375 Hardware Setup

For information about setting up the VRC4375 to run with RedBoot, consult the RedBoot User's Guide. If using serial debugging, the serial line runs at 38400 baud 8-N-1 and should be connected to the debug host using the cable supplied with the board.

Atlas/Malta Hardware Setup

For information about setting up the Atlas and Malta boards to run with RedBoot, consult the RedBoot User's Guide.

PowerPC Cogent Hardware Setup

The eCos Developer's Kit package comes with an EPROM which provides GDB support for the Cogent evaluation board. An image of this EPROM is also provided at `loaders/powerpc-cogent/gdbload.bin` under the root of your eCos installation. The same EPROM and image can be used on all three supported daughterboards: CMA287-23 (MPC823), CMA287-50 (MPC850), and CMA286-60 (MPC860).

The EPROM is installed to socket U4 on the board. Attention should be paid to the correct orientation of the EPROM during installation.

If you are going to burn a new EPROM using the binary image, be careful to get the byte order correct. It needs to be big-endian. If the EPROM burner software has a hex-editor, check that the first few bytes of the image look like:

```
00000000: 3c60 fff0 6063 2000 7c68 03a6 4e80 0020 <`..`c.|h..N..
```

If the byte order is wrong you will see 603c instead of 3c60 etc. Use the EPROM burner software to make a byte-swap before you burn to image to the EPROM.

If the GDB stub EPROM you burn does not work, try reversing the byte-order, even if you think you have it the right way around. At least one DOS-based EPROM burner program is known to have the byte-order upside down.

The GDB stub in the EPROM allows communication with GDB using the serial port at connector P12 (CMA101) or P3 (CMA102). The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a dedicated serial cable as specified in the Cogent CMA manual.

Installing the Stubs into ROM

Preparing the Binaries

These two binary preparation steps are not strictly necessary as the eCos distribution ships with pre-compiled binaries in the directory `loaders/powerpc-cogent` relative to the installation root.

Building the ROM images with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the PowerPC CMA28x hardware.
3. While still displaying the *Build->Templates* dialog box, select the “stubs” package template to build a GDB stub. Click *OK*.
4. Build eCos using *Build->Library*.
5. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. GDB stub ROM images have the prefix “gdb_module”.

Building the ROM images with ecosconfig

1. Make an empty directory to contain the build tree, and cd into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new cma28x stubs
```
3. Enter the commands:

```
$ ecosconfig tree  
$ make
```
4. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. GDB stub ROM images have the prefix “gdb_module”.

Installing the Stubs into ROM or FLASH

1. Program the binary image file gdb_module.bin into ROM or FLASH referring to the instructions of your ROM programmer.
2. Plug the ROM/FLASH into socket U4 as described at the beginning of this *Hardware Setup* section.

PowerPC MBX860 Hardware Setup

The eCos Developer’s Kit package comes with an EPROM which provides GDB support for the Motorola PowerPC MBX860 evaluation board. An image of this EPROM is also provided at `loaders/powerpc-mbx/gdbload.bin` under the root of your eCos installation.

The EPROM is installed to socket XU1 on the board. Attention should be paid to the correct orientation of the EPROM during installation. Only replace the board's existing ROM using a proper PLCC extraction tool, as the socket would otherwise risk getting damaged.

The GDB stub in the EPROM allows communication with GDB using the serial port at connector SMC1/COM1. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a suitable serial cable.

In order to make the board execute the EPROM that you just installed (rather than the on-board FLASH memory), it may be necessary move some links on the board. Specifically, ensure that link J4 is in position 1-2. If in doubt, refer to the MBX documentation from Motorola, ensuring that Boot Port Size=8 Bits/ROM for BOOT (CS#7), in their terminology.

Installing the Stubs into FLASH

Preparing the Binaries

These two binary preparation steps are not strictly necessary as the eCos distribution ships with pre-compiled binaries in the directory `loaders/powerpc-mbx` relative to the installation root.

Building the ROM images with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the PowerPC Motorola MBX860/821 hardware.
3. While still displaying the *Build->Templates* dialog box, select the “stubs” package template to build a GDB stub. Click *OK*.
4. Build eCos using *Build->Library*.
5. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub ROM images have the prefix “gdb_module”.

Building the ROM images with ecosconfig

1. Make an empty directory to contain the build tree, and `cd` into it.
2. To build a GDB stub ROM image, enter the command:


```
$ ecosconfig new mbx stubs
```
3. Enter the commands:


```
$ ecosconfig tree
$ make
```
4. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub ROM images have the prefix “gdb_module”.

Installing the Stubs into ROM

1. Program the binary image file `gdb_module.bin` into ROM or FLASH referring to the instructions of your ROM programmer.
2. Plug the ROM/FLASH into socket XU1 as described near the beginning of this *Hardware Setup* section.

Installing the Stubs into FLASH

This assumes you have EPPC-Bug in the on-board FLASH. This can be determined by setting up the board according to the below instructions and powering up the board. The EPPC-Bug prompt should appear on the SMC1 connector at 9600 baud, 8N1.

1. Set jumper 3 to 2-3 [allow XU2 FLASH to be programmed]
2. Set jumper 4 to 2-3 [boot EPPC-Bug]

Program FLASH

1. Prepare EPPC-Bug for download:

```
EPPC-Bug>lo 0
```

At this point the monitor is ready for input. It will not return the prompt until the file has been downloaded.

2. Use the terminal emulator's ASCII download feature (or a simple clipboard copy/paste operation) to download the `gdb_module.srec` data. Note that on Linux, Minicom's ASCII download feature seems to be broken. A workaround is to load the file into Emacs (or another editor) and copy the full contents to the clipboard. Then press the mouse paste-button (usually the middle one) over the Minicom window.

3. Program the FLASH with the downloaded data:

```
EPPC-Bug>pflash 40000 60000 fc000000
```

4. Switch off the power, and change jumper 4 to 1-2. Turn on the power again. The board should now boot using the newly programmed stubs.

PowerPC Architectural Simulator Setup

The PowerPC simulator is an architectural simulator which implements all the features of the PowerPC needed to run eCos. The current implementation provides accurate simulation of the instruction set and timers, as well as having generic support for diagnostic output and exceptions.

The simulator also allows devices to be simulated, but no device simulation support has been defined for the serial device drivers in this release.

To simplify connection to the simulator, you are advised to create a GDB macro by putting the following code in your personal GDB start-up file (`gdb.ini` on Windows and `.gdbinit` on UNIX).

```
define psim
  target sim -o '/iobus/pal@0xf0001000/reg 0xf0001000 32'
  rbreak cyg_test_exit
  rbreak cyg_assert_fail
end
```

You can then connect to the simulator by invoking the command **psim** on the command line:

```
(gdb) psim
```

You can achieve the same effect by typing out the macro's content on the command line if necessary.

Note: The PowerPC simulator cannot execute binaries built for any of the supported hardware targets. You must generate a configuration using the PowerPC simulator platform:

```
$ ecosconfig new psim
```

or some such.

SPARClite Hardware Setup

The eCos Developer's Kit package comes with a ROM which provides GDB support for the Fujitsu SPARClite Evaluation Board by way of CygMon.

An image of this ROM is also provided at `BASE_DIR/loaders/sparclite-sleb/cygmon.bin`. The ROM is installed in socket IC9 on the evaluation board. Attention should be paid to the correct orientation of the ROM during installation.

The GDB stub allows communication with GDB using a TCP channel via the ethernet port at connector J5.

Ethernet Setup

The ethernet setup is described in the board's manual, but here is a recapitulation.

Set the board's ethernet address using SW1 on the motherboard:

SW1-4	SW1-3	SW1-2	SW1-1	Ethernet Address
----	----	----	----	-----
OFF	OFF	OFF	OFF	No ethernet, use serial
OFF	OFF	OFF	ON	00:00:0E:31:00:01
OFF	OFF	ON	OFF	00:00:0E:31:00:02
OFF	OFF	ON	ON	00:00:0E:31:00:03
OFF	ON	OFF	OFF	00:00:0E:31:00:04
OFF	ON	OFF	ON	00:00:0E:31:00:05
OFF	ON	ON	OFF	00:00:0E:31:00:06
OFF	ON	ON	ON	00:00:0E:31:00:07
ON	OFF	OFF	OFF	00:00:0E:31:00:08
ON	OFF	OFF	ON	00:00:0E:31:00:09
ON	OFF	ON	OFF	00:00:0E:31:00:0A
ON	OFF	ON	ON	00:00:0E:31:00:0B

ON	ON	OFF	OFF	00:00:0E:31:00:0C
ON	ON	OFF	ON	00:00:0E:31:00:0D
ON	ON	ON	OFF	00:00:0E:31:00:0E
ON	ON	ON	ON	00:00:0E:31:00:0F

BOOTP/DHCP service on Linux

Configure the BOOTP or DHCP server on the network to recognize the evaluation board's ethernet address so it can assign the board an IP address. Below is a sample DHCP server configuration from a Linux system (/etc/dhcpd.conf). It shows a setup for three evaluation boards.

```
#
# DHCP server configuration.
#
allow bootp;

subnet 192.168.1.0 netmask 255.255.255.0 {
    host mb831evb {
        hardware ethernet 00:00:0e:31:00:01;
        fixed-address mb831evb;
    }
    host mb832evb {
        hardware ethernet 00:00:0e:31:00:02;
        fixed-address mb832evb;
    }
    host mb833evb {
        hardware ethernet 00:00:0e:31:00:03;
        fixed-address mb833evb;
    }
}
```

BOOTP/DHCP boot process

Even when configured to use a TCP channel, CygMon will still print a boot message to the serial channel. If the BOOTP process was successful and an IP address was found, a message “BOOTP found xxx.xxx.xxx.xxx” will be printed where xxx.xxx.xxx.xxx is the IP address assigned by the BOOTP or DHCP server. If the BOOTP process fails, a message indicating failure will be printed and the serial port will be used as the debug channel.

Once the board finds an IP address it will respond to ICMP echo request packets (ping). This gives a simple means to test the health of the board.

As described in “Ethernet Setup” on page 72, it should now be possible to connect to the SPARClite board from within GDB by using the command:

```
(gdb) target remote <host>:1000
```

Serial Setup

The CygMon stubs also allow communication with GDB by way of the serial port at connector CON1. The communication parameters are fixed at 19200 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a null modem cable. A gender changer may also be required.

SPARClite Architectural Simulator Setup

The ESA SPARClite simulator is an architectural simulator which implements all the features of the SPARClite needed to run eCos. The current implementation provides accurate simulation of the instruction set, interrupt controller, and timers, as well as having generic support for diagnostic output and exceptions.

Note that the ESA SPARClite simulator is unsupported, but is included in the release as a convenience.

To simplify connection to the simulator, you are advised to create a GDB macro by putting the following code in your personal GDB start-up file (gdb.ini on Windows and .gdbinit on UNIX).

```
define ssim
  target sim -nfp -sparclite -dumbio
  rbreak cyg_test_exit
  rbreak cyg_assert_fail
end
```

You can then connect to the simulator by invoking the command **ssim** on the command line:

```
(gdb) ssim
```

You can achieve the same effect by typing out the macro's content on the command line if necessary.

ARM PID Hardware Setup

eCos comes with two ROM images that provide GDB support for the ARM PID board. The first ROM image provides a port of the CygMon ROM monitor, which includes a command-line interface and a GDB remote stub. The second ROM image provides a remote GDB stub only, which is a minimal environment for downloading and debugging eCos programs solely using GDB.

eCos, CygMon and the GDB stubs all support the PID fitted with both ARM7T and ARM9 daughterboards. CygMon and the stubs can be programmed into either the programmable ROM (U12) or the FLASH (U13). Pre-built forms of both ROM images are provided in the directory loaders/arm-pid under the root of your eCos installation, along with a tool that will program the stubs into the FLASH memory on the board. CygMon images are prefixed with the name 'cygmon' and GDB stub ROM images are given the prefix 'gdb_module'. Images may be provided in a number of formats including ELF (.img extension), binary (.bin extension) and SREC (.srec extension). Note that some unreliability has been experienced in downloading files using Angel 1.00. Angel 1.02 appears to be more robust in this application.

Installing the Stubs into FLASH

Preparing the Binaries

These two binary preparation steps are not strictly necessary as the eCos distribution ships with pre-compiled binaries in the directory `loaders/arm-pid` relative to the installation root.

Building the ROM images with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build -> Templates* menu item, and then select the ARM PID hardware.
3. While still displaying the *Build -> Templates* dialog box, select either the "stubs" package template to build a GDB stub image, or the "cygmon" template to build the CygMon ROM Monitor. Click *OK*.
4. Build eCos using *Build -> Library*
5. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub ROM images have the prefix "gdb_module". CygMon images have the prefix "cygmon".

Building the ROM images with ecosconfig

1. Make an empty directory to contain the build tree, and `cd` into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new pid stubs
```


or to build a CygMon ROM monitor image, enter the command:

```
$ ecosconfig new pid cygmon
```
3. Enter the commands:

```
$ ecosconfig tree  
$ make
```
4. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub ROM images have the prefix "gdb_module". CygMon images have the prefix "cygmon".

Building the FLASH Tool with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the ARM PID hardware.
3. Enable the "Build flash programming tool" option in the ARM PID HAL (CYGBLD_BUILD_FLASH_TOOL) and resolve any resulting configuration conflicts.
4. Build eCos using *Build -> Library*

5. When the build completes, the FLASH tool image file can be found in the bin/ subdirectory of the install tree, with the prefix "prog_flash"

Building the FLASH Tool with ecosconfig

1. Make an empty directory to contain the build tree, and cd into it
2. Enter the command:

```
$ ecosconfig new pid
```
3. Edit the file ecos.ecc and enable the option CYGBLD_BUILD_FLASH_TOOL by uncommenting its user_value property and setting it to 1.
4. Enter the commands:

```
$ ecosconfig resolve
```

[there will be some output]

```
$ ecosconfig tree
```

```
$ make
```
5. When the build completes, the FLASH tool image file can be found in the bin/ subdirectory of the install tree, with the prefix "prog_flash"

Prepare the Board for FLASH Programming

Each time a new image is to be programmed in the FLASH, the jumpers on the board must be set to allow Angel to run:

1. Set jumper 7-8 on LK6 [using the Angel code in the 16 bit EPROM]
2. Set jumper 5-6 on LK6 [select 8bit ROM mode]
3. Set jumper LK18 [ROM remap - this is also required for eCos]
4. Set S1 to 0-0-1-1 [20MHz operation]
5. Open jumper LK4 [enable little-endian operation] Attach a serial cable from Serial A on the PID board to connector 1 on the development system. This is the cable through which the binaries will be downloaded. Attach a serial cable from Serial B on the PID board to connector 2 on the development system (or any system that will work as a terminal). Through this cable, the FLASH tool will write its instructions (at 38400 baud).

Program the FLASH

1. Download the FLASH ROM image onto the PID board. For example, for the GDB stubs image:

```
bash$ arm-elf-gdb -nw gdb_module.img
GNU gdb 4.18-DEVTOOLSVERSION
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute copies
```

```
of it under certain conditions. Type "show copying" to see the conditions.
There is absolutely no warranty for GDB. Type "show warranty" for details.
This GDB was configured as "--host=i586-pc-cygwin32 --target=arm-elf".
(no debugging symbols found)...
(gdb) target rdi s=com1
Angel Debug Monitor for PID (Built with Serial(x1), Parallel, DCC) 1.00
(Advanced RISC Machines SDT 2.10)
Angel Debug Monitor rebuilt on Jan 20 1997 at 02:33:43
Connected to ARM RDI target.
(gdb) load
Loading section .rom_vectors, size 0x44 lma 0x60000
Loading section .text, size 0x1f3c lma 0x60044
Loading section .rodata, size 0x2c lma 0x61f80
Loading section .data, size 0x124 lma 0x61fac
Start address 0x60044 , load size 8400
Transfer rate: 5169 bits/sec.
(gdb) q
The program is running.  Exit anyway? (y or n) y
```

Note: On a UNIX or Linux system, the serial port must be /dev/ttyS0 instead of COM1. You need to make sure that the /dev/ttyS0 files have the right permissions:

```
$ su
Password:
# chmod o+rw /dev/ttyS0*
# exit
```

If you are programming the GDB stub image, it will now be located at 0x60000..0x64000. If you are programming the Cygmon ROM Monitor, it will be located at 0x60000..0x80000.

2. Now download the FLASH programmer tool

```
bash$ arm-elf-gdb prog_flash.img
GNU gdb 4.18-DEVTOOLSVERSION
Copyright 1998 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License,
and you are welcome to change it and/or distribute
copies of it under certain conditions. Type "show copying" to see
the conditions. There is absolutely no warranty for GDB. Type "show
warranty" for details.
This GDB was configured as "--host=i586-pc-cygwin32 --target=arm-elf".
(gdb) target rdi s=com1
Angel Debug Monitor for PID (Built with Serial(x1), Parallel, DCC) 1.00
(Advanced RISC Machines SDT 2.10)
Angel Debug Monitor rebuilt on Jan 20 1997 at 02:33:43
Connected to ARM RDI target.
(gdb) load
Loading section .rom_vectors, size 0x44 lma 0x40000
Loading section .text, size 0x44a4 lma 0x40044
Loading section .rodata, size 0x318 lma 0x444e8
Loading section .data, size 0x1c8 lma 0x44800
Start address 0x40044 , load size 18888
Transfer rate: 5596 bits/sec.
(gdb) c
```

3. The FLASH tool will output some text on the board serial port B at 38400 baud:

```
ARM
eCos

FLASH here!
manuf: 8, device: 40
Error: Wrong Manufacturer: 08
... Please change FLASH jumper
```

4. This text is repeated until you remove the jumper 7-8 on LK6. Then the output will be:

```
manuf: 1F, device: A4
AT29C040A recognised
About to program FLASH using data at 60000..64000
*** Press RESET now to abort!
```

5. You have about 10 seconds to abort the operation by pressing reset. After this timeout, the FLASH programming happens:

```
...Programming FLASH
All done!
```

6. Quit/kill the GDB process, which will hang.
7. Next time you reset the board, the stub will be in control, communicating on Serial A at 38400 baud.

Note: If you do not have two serial ports available on your host computer, you may still verify the FLASH programming completed successfully by quitting/killing the GDB process after running "c" in step 2 above. Then switch the serial cable on the PID from Serial A to Serial B and run a terminal emulator on the host computer. In a few seconds you should see the the repeated text described in step 2 above and you may continue the remaining steps as normal.

Programming the FLASH for big-endian mode

The process is almost identical to the previous instructions which apply to a PID board running in little-endian mode only.

The only adjustments to make are that if programming a *GDB* stub ROM image (or CygMon ROM monitor image), you must enable the option "Use Big-endian mode" in the *eCos Configuration Tool* (CYGHWR_HAL_ARM_BIGENDIAN if using ecosconfig and editing ecos.ecc).

When programming the FLASH there are two options:

1. Program FLASH using the little-endian FLASH tool. After powering off, replace the ROM controller with the special big-endian version which can be acquired from ARM. (This has not been tested by Red Hat).
2. Use a special big-endian version of the FLASH tool which byte-swaps all the words as they are written to the FLASH.

Build this tool by enabling the "Build flash programming tool for BE images on LE boards" option (CYGBLD_BUILD_FLASH_TOOL_BE), resulting in a utility with the prefix "prog_flash_BE_image_LE_system" which should be used instead of "prog_flash".

Note that there is a limitation to this method: no sub-word data can be read from the ROM. To work around this, the .rodata section is folded into the .data section and thus copied to RAM before the system starts.

Given that Thumb instructions are 16 bit, it is not possible to run ROM-startup Thumb binaries on the PID board using this method.

When the image has been programmed, power off the board, and set jumper LK4 to enable big-endian operation.

Installing the Stubs into ROM

1. Program the binary image file `gdb_module.bin` into ROM referring to the instructions of your ROM programmer.
2. Plug the ROM into socket U12 and install jumper LK6 pins 7-8 to enable the ROM.

ARM AEB-1 Hardware Setup

Overview

The ARM AEB-1 comes with tools in ROM. These include a simple FLASH management tool and the Angel® monitor. eCos for the ARM AEB-1 comes with GDB stubs suitable for programming into the onboard FLASH. GDB is the preferred debug environment for GDB, and while Angel provides a subset of the features in the eCos GDB stub, Angel is unsupported.

Both eCos and the stubs support both Revision B and Revision C of the AEB-1 board. Stub ROM images for both types of board can be found in the `loaders/arm-aeb` directory under the root of your eCos installation. You can select which board you are using by selecting either the `aeb` or `aebC` platform by selecting the appropriate platform HAL in the *eCos Configuration Tool*.

The GDB stub can be downloaded to the board for programming in the FLASH using the board's on-board ROM monitor:

1. talk to the AEB-1 board with a terminal emulator (or a real terminal!)
2. use the board's rom menu to download a UU-encoded version of the GDB stubs which will act as a ROM monitor
3. tell the board to use this new monitor, and then hook GDB up to it for real debugging

Talking to the Board

Connect a terminal or computer's serial port to the ARM AEB-1. On a PC with a 9-pin serial port, you can use the cable shipped by ARM with no modification.

Set the terminal or terminal emulator to 9600N1 (9600 baud, no parity, 1 stop bit).

Reset the board by pressing the little reset button on the top. You will see the following text:

```
ARM Evaluation Board Boot Monitor 0.01 (19 APR 1998)
Press ENTER within 2 seconds to stop autoboot
```

Press ENTER quickly, and you will get the boot prompt:

```
Boot:
```

Downloading the Stubs via the Rom Menu

Using the AEB-1 rom menu to download the GDB stubs from the provided ".UU" file.

Note: This is an annotated 'terminal' session with the AEB-1 monitor:

```
+Boot: help
Module is BootStrap          1.00 (14 Aug 1998)

Help is available on:

Help      Modules      ROMModules  UnPlug      PlugIn
Kill      SetEnv       UnSetEnv    PrintEnv    DownLoad
Go        GoS          Boot        PC           FlashWrite
FlashLoad FlashErase

Boot: download c000
Ready to download. Use 'transmit' option on terminal
emulator to download file.

... at this point, download the ASCII file "loaders/arm-aeb/
gdb_module.img.UU". The details of this operation differ
depending on which terminal emulator is used. It may be
necessary to enter "^D" (control+D) when the download completes
to get the monitor to return to command mode.

Loaded file gdb_module.img.bin at address
0000c000, size = 19392
```

Activating the GDB Stubs

Commit the GDB stubs module to FLASH:

```
Boot: flashwrite 4018000 C000 8000
```

Verify that the eCos/"GDB stubs" module is now added in the list of modules in the board:

```
Boot: rommodules
```

You should see output similar to the following:

```
Header      Base      Limit
04000004 04000000 040034a8 BootStrap      1.00 (14 Aug 1998)
04003a74 04003800 04003bc0 Production Test 1.00 (13 Aug 1998)
0400e4f4 04004000 0400e60f Angel          1.02 (12 MAY 1998)
0401c810 04018000 0401cbc0 eCos            1.3  (27 Jan 2000)
GDB stubs
```

Now make the eCos/"GDB stubs" module be the default monitor:

```
Boot: plugin eCos
```

Note: Since the GDB stubs are always linked at the same address (0x4018000), the operation of writing to the FLASH and selecting the stubs as default monitor is an idempotent operation. You can download a new set of stubs following the same procedure - you do not have to unregister or delete anything.

Building the GDB Stub FLASH ROM Images

Pre-built GDB stubs images are provided in the directory loaders/arm-aeb relative to the root of your eCos installation, but here are instructions on how to rebuild them if you should ever need to.

Building the GDB Stubs with the eCos Configuration Tool

1. Start with a new document - selecting the *File -> New* menu item if necessary to do this.
2. Choose the *Build -> Templates* menu item, and then select the ARM AEB-1 hardware.
3. While still displaying the *Build->Templates* dialog box, select the "stubs" package template to build a GDB stub image. Click *OK*.
4. If applicable, set the "AEB board revision" option to "C" from "B" depending on the board revision being used.
5. Build eCos using *Build -> Library*.
6. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. The GDB stub ROM images have the prefix "gdb_module".

Building the GDB Stub ROMs with ecosconfig

1. Make an empty directory to contain the build tree, and cd into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new aeb stubs
```

3. If applicable, edit `ecos.ecc` and set the AEB board revision. (`CYGHWR_HAL_ARM_AEB_REVISION`) from the default "B" to "C" by uncommenting the `user_value` property and setting it to "C".

4. Enter the commands

```
$ ecosconfig tree
$ make
```

5. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. The GDB stub ROM images have the prefix `"gdb_module"`.

ARM Cogent CMA230 Hardware Setup

The eCos Developer's Kit package comes with an EPROM which provides GDB support for the Cogent evaluation board. An image of this EPROM is also provided at `loaders/arm-cma230/gdbload.bin` under the root of your eCos installation.

The EPROM is installed to socket U3 on the board. Attention should be paid to the correct orientation of the EPROM during installation.

If you are going to burn a new EPROM using the binary image, be careful to get the byte order correct. It needs to be little-endian, which is usually the default in PC based programmer software.

If the GDB stub EPROM you burn does not work, try reversing the byte-order, even if you think you have it the right way around. At least one DOS-based EPROM burner program is known to have the byte-order upside down.

The GDB stub in the EPROM allows communication with GDB using the serial port at connector P12 (CMA101) or P3 (CMA102). The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a dedicated serial cable as specified in the Cogent CMA manual.

Building the GDB Stub FLASH ROM images

Pre-built GDB stubs images are provided in the directory `loaders/arm-cma230` relative to the root of your eCos installation, but here are instructions on how to rebuild them if you should ever need to.

CygMon images are prefixed with the name `'cygmon'` and GDB stub ROM images

are given the prefix `'gdb_module'`. Images may be provided in a number of formats including ELF (`.img` extension), binary (`.bin` extension) and SREC (`.srec` extension).

Building the GDB Stubs with the eCos Configuration Tool

1. Start with a new document - selecting the File->New menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the ARM CMA230 hardware.

3. While still displaying the *Build -> Templates* dialog box, select the "stubs" package template to build a GDB stub image. Click *OK*.
4. Build eCos using *Build -> Library*
5. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. The GDB stub ROM images have the prefix "gdb_module".

Building the GDB Stub ROMs with ecosconfig

1. Make an empty directory to contain the build tree, and cd into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new cma230 stubs
```
3. Enter the commands:

```
$ ecosconfig tree  
$ make
```
4. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. The GDB stub ROM images have the prefix "gdb_module".

Cirrus Logic ARM EP7211 Development Board Hardware Setup

eCos comes with two Flash ROM images that provide GDB support for the Cirrus Logic EP7211 Development Board (also known as the EDB7211).. Note that on some board revisions, the board is silk-screened as EDB7111-2. The first Flash ROM image provides a port of the CygMon ROM monitor, which includes a command-line interface and a GDB remote stub. The second Flash ROM image provides a remote GDB stub only.

Both ROM images are provided in the directory loaders/arm-edb7211 under the root of your eCos installation. CygMon images are prefixed with the name 'edb7211_cygmon' and are provided in a number of formats including binary (.bin extension) and SREC (.srec) extension. GDB stub ROM images are given the prefix 'edb7211_gdb_module'.

The ROM images provided for the EP7211 Development Board must be programmed into the FLASH. Please refer to the section titled "Loading the ROM image into On-Board flash" on how to program the ROM onto the board.

Both Cygmon and GDB Stub ROMS allow communication with GDB via the serial connector labelled 'UART 1'. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a null modem cable. A gender changer may also be required. Note that the GDB Configuration tool uses the serial port identifiers 0 and 1 to identify the EB7211 serial ports UART1 and UART2 respectively.

Both eCos and the ROM images assume the core clock is generated with a 3.6864 MHz PLL input. The CPU will be configured to run at 73.728MHz.

Note: The EP7211 CPU needs a two step RESET process. After pressing the `URESET` pushbutton, the `WAKEUP` pushbutton must be pressed to complete the process.

Note: When an eCos program is run on an EDB7211 board fitted with either CygMon or a GDB stub ROM, then the code in ROM loses control. This means that if you require the ability to remotely stop execution on the target, or want thread debugging capabilities, you must include GDB stub support when configuring eCos.

Building programs for programming into FLASH

If your application is to be run directly from FLASH, you must configure eCos appropriately for "ROM" startup. This can be done in the *eCos Configuration Tool* by setting the "Startup type" HAL option to "ROM". If using the *ecosconfig* utility, set the *user_value* of the *CYG_HAL_STARTUP* option in *ecos.ecc* to "ROM".

When you have linked your application with eCos, you will then have an ELF executable. To convert this into a format appropriate for the Cirrus Logic FLASH download utility, or the *dl_7xxx* utility on Linux, you can use the utility *arm-elf-objcopy*, as in the following example:

```
$ arm-elf-objcopy -O binary helloworld.exe helloworld.bin
```

This will produce a binary format image *helloworld.bin* which can be downloaded into FLASH.

Building the GDB Stub FLASH ROM images

Pre-built GDB stubs images are provided in the directory *loaders/arm-edb7211* relative to the root of your eCos installation, but here are instructions on how to rebuild them if you should ever need to.

CygMon images are prefixed with the name 'cygmon' and GDB stub ROM images are given the prefix 'gdb_module'. Images may be provided in a number of formats including ELF (.img extension), binary (.bin extension) and SREC (.srec extension).

Building the ROM images with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the "Cirrus Logic development board" hardware.
3. While still displaying the *Build -> Templates* dialog box, select either the "stubs" package template to build a GDB stub image, or the "cygmon" template to build the CygMon ROM Monitor. Click *OK*.
4. Build eCos using *Build -> Library*
5. When the build completes, the image files can be found in the *bin/* subdirectory of the install tree. GDB stub ROM images have the prefix "gdb_module". CygMon images have the prefix "cygmon".

Building the ROM images with ecosconfig

1. Make an empty directory to contain the build tree, and cd into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new edb7xxx stubs
```

or to build a CygMon ROM monitor image, enter the command:

```
$ ecosconfig new edb7xxx cygmon
```

3. Enter the commands:

```
$ ecosconfig tree
$ make
```

4. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. GDB stub ROM images have the prefix "gdb_module". CygMon images have the prefix "cygmon".

Loading the ROM Image into On-board Flash

Program images can be written into Flash memory by means of a bootstrap program which is built into the EDB7211. This program communicates with a support program on your host to download and program an image into the Flash memory.

Cirrus Logic provides such a program for use with Windows/DOS. eCos comes with a similar program which will run under Linux. The basic operation of both programs is the same.

1. Connect a serial line to 'UART 1'.
2. Power off the EDB7211.
3. Install jumper 'PROGRAM ENABLE' which enables this special mode for downloading Flash images. Note that some board revisions have this jumper labelled "BOOT ENABLE".
4. Power on the EDB7211.
5. Execute the Flash writing program on your host. On Linux, this would be:

```
# dl_edb7xxx <PATH>/gdb_module.bin
```

where '<PATH>' is the path to the binary format version of the ROM image you wish to load, either as built in the previous section or the "loaders/arm-edb7211/" subdirectory of your eCos installation. The download tool defaults to 38400 baud and device /dev/ttyS1 for communication. To change these, specify them as parameters, e.g.

```
# dl_edb7xxx <PATH>/gdb_module.bin 9600 /dev/ttyS0
```

6. The download program will indicate that it is waiting for the board to come alive. At this point, press 'RESET' and then 'WAKEUP' switches in order. There should be some indication of progress, first of the code being downloaded, then of the programming process.
7. Upon completion of the programming, power off the EDB7211.
8. Remove the 'PROGRAM ENABLE/BOOT ENABLE' jumper.

9. Power on the EDB7211, press 'RESET' and 'WAKEUP'. The new ROM image should now be running on the board.
10. The GDB debugger will now be able to communicate with the board to download and debug RAM based programs. This procedure also applies for loading ROM-startup eCos programs into the on-board FLASH memory, given a binary format image of the program from arm-elf-objcopy. Loading a ROM-startup eCos program into Flash will overwrite the GDB Stub ROM/CygMon in Flash, so you would have to reload the GDB Stub ROM/CygMon to return to normal RAM-startup program development.

Building the Flash Downloader on Linux

eCos provides a Flash download program suitable for use with the EP7211 Development Board which will run on Linux. Follow these steps to build this program. Note: at the time of the writing of these instructions, the download program is built directly within the eCos source repository since it is not configuration specific.

```
# cd <eCos install dir>/packages/hal/arm/edb7xxx/<version>/support
# make
```

(where '#' is your shell prompt)

Note: this program was adapted from the Cirrus Logic original DOS program and still contains some vestiges of that environment.

Developing eCos Programs with the ARM Multi-ICE

The EP7211 Development Board supports use of the ARM Multi-processor EmbeddedICE(tm), also known as the Multi-ICE. Full instructions on how to install and use the Multi-ICE in conjunction with GDB are provided in the *"GNUPro Toolkit Reference for eCos ARM/Thumb"* manual. However, the following platform-specific details should be noted.

You will need an ARM Multi-ICE Server configuration file for the EP7211 Development Board. Here is a suggested configuration file to use:

```
===== File "720T.cfg" =====
;Total IR length = 4
[TITLE]
Multi-ICE configuration for EP7211

[TAP 0]
ARM720T

[TAPINFO]
YES

[Timing]
Low=0
High=0
Adaptive=OFF
=====
```

You must ensure that the board has the appropriate soldered connections. For the EP7211 this involves connecting TEST0 and TEST1 of the EP7211 to ground. To do this you must solder a wire from ground at JP33 to TP8 and TP9.

With respect to using multiple devices simultaneously, note that the EP7211 is not ID sensitive.

If you wish to view diagnostic output from your program that was downloaded via the Multi-ICE, you will note that by default the output on the serial line (as viewed by a terminal such as Hyperterm in Windows, or cu in Unix) is in the form of GDB packets.

To get legible output, the solution is to set the "GDB Serial port" to a different device from the "Diagnostic serial port", and you should use the Diagnostic serial port to view the diagnostic output.

Warning: The multi-ice-gdb-server will fail on startup if the board has not been both reset and awakened before running the server.

To resolve this, it is necessary to free up the connection from within the ARM Multi-ICE server itself. However when this happens, the next time you use GDB to load the program into the board, you will see lots of "Readback did not match original data" messages in the output of the multi-ice-gdb-server program. This indicates your program did not load correctly, and you should restart the multi-ice-gdb-server program, taking care to reset the board correctly before reconnecting.

As a reminder, you must specify --config-dialog to the multi-ice-gdb-server program to connect to the board correctly. If you do not, the multi-ice-gdb-server program will not be able to connect.

Cirrus Logic ARM EP7212 Development Board Hardware Setup

The Cirrus Logic EP7212 Development Board is almost identical to the EP7211 Development Board from a hardware setup viewpoint, and is based on the same port of eCos. Therefore the earlier documentation for the EP7211 Development Board can be considered equivalent, but with the following changes:

- The first serial port is silk screened as "UART 1" on the EP7211 Development Board, but is silk screened as "Serial Port 0" on the EP7212 Development Board. Similarly "UART 2" is silk screened as "Serial Port 1" on the EP7212 Development Board.
- JP2 (used to control reprogramming of the FLASH) is not silkscreened with "Boot Enable".
- To setup the EP7212 Development Board for use with the ARM Multi-ICE JTAG debugging interface unit, it is necessary to connect TEST0 and TEST1 of the EP7212 to ground. On the Development Board, this is accomplished by placing shorting blocks on JP47 and JP48. When the shorting blocks are fitted, the board can only be operated through the Multi-ICE - debugging over a serial line is not possible.
- Pre-built GDB stubs are provided in the directory `loaders/arm-edb7212` relative to the root of your eCos installation
- When rebuilding the GDB stub ROM image, change the "Cirrus Logic processor variant" option (CYGHWR_HAL_ARM_EDB7XXX_VARIANT) from the EP7211 to the EP7212. This can be selected in the *eCos Configuration Tool*, or if using `ecosconfig`, can be set by uncommenting the `user_value` property of this option in `ecos.ecc` and setting it to "EP7212".

Cirrus Logic ARM EP7312 Development Board Hardware Setup

The Cirrus Logic EP7312 Development Board is similar to the EP7212 Development Board from a hardware setup viewpoint, and is based on the same port of eCos.

When rebuilding the RedBoot ROM image or an eCos application, change the "Cirrus Logic processor variant" option (CYGHWR_HAL_ARM_EDB7XXX_VARIANT) from the EP7211 to the EP7312. This can be selected in the *eCos Configuration Tool*, or if using ecosconfig, can be set by uncommenting the user_value property of this option in ecos.ecc and setting it to "EP7312".

See the RedBoot documentation for building and installing RedBoot for this target. Only RedBoot is supported as a boot image; ROMRAM startup is recommended.

90MHz Operation

The EP7xxx targets offer a choice of clock speeds, from 18MHz to a maximum, normally, of 72MHz. These are described as kHz values 18432 36864 49152 and 73728 within the configuration tool. If you have a release which supports it, you will also see 90317 as an available option here, for 90MHz operation.

This option only applies to certain EP7312 hardware, not all EP7312 boards support it. Do not select 90MHz when building RedBoot or your eCos application unless you are absolutely sure that your board supports it.

If you do have a 90MHz board and wish to execute at 90MHz, it is in fact not necessary to build RedBoot specially, if you build your eCos application configured for 90MHz. RedBoot will run at 72MHz and your application will run at 90MHz. If you do install a 90MHz RedBoot, then you must build eCos for 90MHz or timing and baud rates on serial I/O will be wrong.

In other words, code (either eCos app or RedBoot) built for 90MHz will "change up a gear" when it starts up; but code built for 72MHz, because it needs to run correctly on boards without the "gearbox" does not change back down, so if you mix the two, unexpected timing can result. To run a non-eCos application without any hardware initialization code at 90MHz, you must install a specially-built RedBoot.

Cirrus Logic ARM EP7209 Development Board Hardware Setup

Note: At time of writing, no EP7209 Development Board is available, and consequently eCos has not been verified for use with the EP7209 Development Board.

The Cirrus Logic EP7209 Development Board is almost identical to the EP7212 Board in all respects, except that it is not fitted with DRAM, nor has it a DRAM controller.

The only valid configuration for the EDB7209 is ROM based. The STUBS and RAM startup modes are not available as no DRAM is fitted.

Cirrus Logic ARM CL-PS7111 Evaluation Board Hardware Setup

The implementation of the port of eCos to the Cirrus Logic ARM CL-PS7111 Evaluation Board (also known as EB7111) is based on the EP7211 Development Board port.

For that reason, the setup required is identical to the EP7211 Development Board as described above, with the following exceptions:

- The Cygmon ROM monitor is not supported
- The ARM Multi-ICE is not supported
- Pre-built GDB stubs are provided in the directory loaders/arm-eb7111 relative to the root of your eCos installation
- If rebuilding the GDB stub ROM image, change the "Cirrus Logic processor variant" option (CYGHWR_HAL_ARM_EDB7XXX_VARIANT) from the EP7211 to the CL_PS7111. This can be selected in the *eCos Configuration Tool*, or if using ecosconfig, can be set by uncommenting the user_value property of this option in ecos.ecc and setting it to "CL_PS7111"

All remote serial communication is done with the serial I/O connector

```
/misc  
% slow_cat.tcl < [path]/gdb_module.srec > /dev/ttyS0
```

Power off the board, and change it to boot the GDB stubs in big-endian mode by setting the switches like this:

SW1: 00000000 (all levers down) SW2: 10001010

The GDB stubs allow communication with GDB using the serial port at connector PJ7A (lower connector). The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a straight through serial cable.

StrongARM EBSA-285 Hardware Setup

The eCos Developer's Kit package comes with a ROM image which provides GDB support for the Intel® StrongARM® Evaluation Board EBSA-285. Both eCos and the Stub ROM image assume the clocks are: 3.6864 MHz PLL input for generating the core clock, and 50MHz osc input for external clocks. An image of this ROM is also provided at loaders/arm-ebsa285/gdbload.bin under the root of your eCos installation.

The ROM monitor image (an eCos GDB stub) provided for the EBSA-285 board must be programmed into the flash, replacing the Angel monitor on the board. Please refer to the section titled "Loading the ROM Image into On-Board flash" on how to program the ROM onto the board.

The Stub ROM allows communication with GDB via the serial connector on the bulkhead mounting bracket COM0. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed.

Building the GDB Stub FLASH ROM images

Pre-built GDB stubs images are provided in the directory `loaders/arm-ebsa285` relative to the root of your eCos installation, but here are instructions on how to rebuild them if you should ever need to.

Building the GDB Stubs with the eCos Configuration Tool

1. Start with a new document - selecting the *File -> New* menu item if necessary to do this.
2. Choose the *Build -> Templates* menu item, and then select the StrongARM EBSA285 hardware.
3. While still displaying the *Build -> Templates* dialog box, select the "stubs" package template to build a GDB stub image. Click *OK*.
4. Build eCos using *Build -> Library*
5. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. The GDB stub ROM images have the prefix "gdb_module".

Building the GDB Stub ROMs with ecosconfig

(See "Using ecosconfig on UNIX" on page 72)

1. Make an empty directory to contain the build tree, and `cd` into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new ebsa285 stubs
```
3. Enter the commands:

```
$ ecosconfig tree
$ make
```
4. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. The GDB stub ROM images have the prefix "gdb_module".

Loading the ROM Image into On-board Flash

There are several ways to install the eCos gdb stub ROM image in the EBSA board's flash memory. Once installed, the gdb stub ROM provides standard eCos download and debug via the EBSA board's serial port. The options available include the Linux based EBSA flash upgrade utility provided by Red Hat, direct writing of the flash via MultiICE (JTAG) hardware debugger, and other flash management utilities from Intel (these only support DOS, and proprietary ARM tools and image formats). Only the Red Hat flash upgrade tool is supported and tested in this release.

The flash upgrade tool requires the EBSA board to be configured as a PCI slave (rather than a master, its normal operating mode) and plugged into a Linux host computer's PCI bus.

Configuring the board for flash loading: Follow the instructions in the EBSA-285 Reference Manual, pages A-2 and A-3 to configure the board as an add-in card, and enable flash blank programming. Briefly: assuming the board

was in the default setting to execute as a bus master ("Host Bridge") make jumper 9 (J9), move jumper 10 (J10) to external reset (PCI_RST), and move jumper 15 (J15) link 4-6-5 to connect 5-6 instead of 4-6.

Configuring the board for execution of eCos programs: Follow the instructions in the EBSA-285 Reference Manual, pages A-2 and A-3 to configure the board as a "Host Bridge" with "Central Function". Briefly: unset J9, move J10 to on-board reset (BRD_RST), and set J15 to make 4-6 instead of 5-6 (see page A-8 also). Plug the card into its own PCI bus, not the Linux PC used for the flash-programming process.

Building the Linux software: the Linux software sources are in directory

```
<BASE_DIR>/packages/hal/arm/ebsa285/v1_3/support/linux/safl_util
```

in the eCos source repository. There are two parts to the system: a loadable kernel module and the flash utility. The loadable kernel module is safl.o and the utility is sa_flash. To build:

cd to this directory, or a copy of it.

make

This builds safl.o and sa_flash. The kernel module must be installed, and a device file created for it. Both of these operations require root permissions. Create the device file by:

```
% mknod /dev/safl c 10 178
```

Programming the flash: switch off the EBSA-285, and remove the EBSA-285 board from its PCI bus. Take appropriate anti-static precautions. Configure it for flash loading as above, halt your Linux system and turn it off. Install the EBSA-285 board in the PCI bus of the Linux system and boot it up. (Single user is good enough, assuming your image and safl_util build dir are on a local disc partition.) Change directory to the safl_util directory, then, to load the kernel module and flash an image onto the eval board (as root):

```
% insmod safl.o
% sa_flash <image_file>
```

Halt and turn off the Linux machine and remove the EBSA-285 card. Take appropriate anti-static precautions. Configure it for execution of eCos programs as above, and plug it into its own PCI bus. Restart the Linux machine however you wish.

This information is replicated in the README file within the safl_util directory and its parents, and in the EBSA-285 Reference Manual from Intel, appendix A "Configuration Guide". If in doubt, please refer to those documents also.

This procedure also applies for loading ROM-startup eCos programs into the on-board flash memory, given a binary format image of the program from arm-elf-objcopy. Loading a ROM-startup eCos program into flash will overwrite the StubROM in flash, so you would have to reload the StubROM to return to normal RAM-startup program development.

Running your eCos Program Using GDB and the StubROM

Note: You must first load the StubROM image into the flash memory on the EBSA-285 board before doing this. See "Loading the ROM Image into On-board Flash", page 93 for details.

Connect to the StubROM in the board and run your eCos program <PROGRAM> as follows:

```
$ arm-elf-gdb -nw <PROGRAM>
(gdb) set remotebaud 38400
(gdb) target remote <DEVICE>
```

Where <DEVICE> is /dev/ttyS0 or COM1: or similar, depending on your environment and how you connected your serial line to the host computer. Expect some output here, for example:

```
Remote debugging using /dev/ttyS0
0x410026a4 in ?? ()
```

then, to load the program

```
(gdb) load
```

which will report locations and sizes of sections as they load, then begin execution using

```
(gdb) continue
```

If you have no eCos program yet, but you want to connect to the board just to verify serial communications, tell gdb "set endian little" before anything else, so that it understands the board (GDB normally infers this from information within the eCos program).

Note: When an eCos program is run on the EBSA-285 board, the GDB stub in ROM loses control. This means that if you require the ability to stop execution on the target remotely, or want thread debugging capabilities, you must include GDB stub support when configuring eCos.

Compaq iPAQ PocketPC Hardware Setup

For setting up the iPAQ to run with RedBoot, see the the *RedBoot User's Guide*. Connections may be made using the Compact Flash Ethernet interface. A serial cable may be connected directly, or via the cradle. Serial communication uses the parameters 38400,8,N,1. The LCD/Touchscreen may also be used as an interface to RedBoot and eCos applications.

Arm Industrial Module AIM 711 Hardware Setup

The Arm Industrial Module AIM 711 comes with RedBoot installed as the default boot loader.

For developing without having a finished custom board, a starter-kit with a minimally configured board is available. It offers all the connectors needed for development, including serial device, Ethernet, power supply and an extra connector for the external bus.

Setup Hardware

Power supply

A 6V - 7.5V power supply must be connected to J2 or TB1. At J2 the inner pin is V+ and at TB1 it is pin 1.

Serial devices

The AIM 711 has 3 serial devices, which are the debug and diagnostic channel COM0 (/dev/ser0), the high performance 16550 UART COM1 (/dev/ser1) and the second internal device COM2 (/dev/ser2).

To use the debug channel, which is also the default for RedBoot, the supplied DB9-male cable must be connected to CN4. If the also available service board is used, the above connector must be disabled by setting JP1.

COM1 is available over the RJ45 connector CN2. This device can be configured as a RS232, RS422, RS485 or TTL level

COM2 is only available with TTL level at CN5.

Ethernet

The RJ45 connector CN1 is for Ethernet.

Installing RedBoot into FLASH

Using RedBoot

In order that Redboot can overwrite itself, Redboot is built as a ROMRAM image.

Load the RedBoot binary to the next free space:

```
RedBoot> load -m tftp -h 192.168.1.36 -r -b 0x40000 redboot.bin
Raw file loaded 0x00040000-0x00063233, assumed entry at 0x00040000
```

Store it in FLASH:

```
RedBoot> fis create RedBoot
An image named 'RedBoot' exists - continue (y/n)? y
... Erase from 0x02000000-0x02025000: .....
... Program from 0x00040000-0x00063234 at 0x02000000: .....
.....
... Erase from 0x021ff000-0x02200000: .
... Program from 0x007ff000-0x00800000 at 0x021ff000: .
```

Restart the AIM 711:

```
RedBoot> reset
... Resetting.
```

Using JTAG

To rewrite the FLASH using JTAG the service board must be used, which includes a JTAG connector.

More documentation

For more information please look at <http://www.visionsystems.de/arm7.html>.

SH3/EDK7708 Hardware Setup

The eCos Developer's Kit package comes with a ROM which provides GDB support for the Hitachi EDK7708 board (a big-endian and a little-endian version). Images of these ROMs are also provided at `loaders/sh-edk7708/gdbload.bin` and `loaders/sh-edk7708le/gdbload.bin` under the root of your eCos installation.

The ROM is installed to socket U6 on the board. When using the big-endian ROM, jumper 9 must be set to 2-3. When using the little-endian ROM, jumper 9 must be set to 1-2. Attention should be paid to the correct orientation of the ROM during installation. Only replace the board's existing ROM using a proper PLCC extraction tool, as the socket would otherwise risk being damaged.

If you are going to program a new ROM or FLASH using the binary image, you may have to experiment to get the right byte-order in the device. Depending on the programming software you use, it might be necessary to enable byte-swapping. If the GDB stub ROM/FLASH you program does not work, try reversing the byte-order.

The GDB stub in the EPROM allows communication with GDB using the serial port at connector J1. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using the dedicated serial cable included in the EDK package.

Installing the Stubs into FLASH

Preparing the Binaries

These two binary preparation steps are not strictly necessary as the eCos distribution ships with pre-compiled binaries in the directory `loaders/sh-edk7708` and `loaders/sh-edk7708le` relative to the installation root.

Building the ROM images with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the SH EDK7708 hardware.

3. While still displaying the *Build->Templates* dialog box, select the “stubs” package template to build a GDB stub. Click *OK*.
4. If building a little-endian image, disable the “Use big-endian mode” option in the SH EDK7708 HAL (CYGHWR_HAL_SH_BIGENDIAN).
5. Build eCos using *Build->Library*.
6. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. GDB stub ROM images have the prefix “gdb_module”.

Building the ROM images with ecosconfig

1. Make an empty directory to contain the build tree, and cd into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new edk7708 stubs
```
3. If building a little-endian image, uncomment the user value in ecos.ecc for CYGHWR_HAL_SH_BIGENDIAN and change it to 0.
4. Enter the commands:

```
$ ecosconfig tree
$ make
```
5. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. GDB stub ROM images have the prefix “gdb_module”.

Installing the Stubs into ROM or FLASH

1. Program the binary image file gdb_module.bin into ROM or FLASH referring to the instructions of your ROM programmer.
2. Plug the ROM/FLASH into socket U6. If the image is little-endian set jumper 9 to 1-2. If the image is big-endian set jumper 9 to 2-3.

SH3/CQ7708 Hardware Setup

Preparing the board

Make sure the DIP switches on the board are set as follows:

```
SW1-1  ON
SW1-2  OFF
SW1-3  ON
SW1-4  OFF
```

```
SW2-1  ON
SW2-2  ON
SW2-3  OFF
SW2-4  OFF
```

If you are using a straight through serial cable which has flow control lines, you will also need to cut JP12 (5-6) as the flow control lines can cause NMIs.

eCos GDB Stubs

The eCos installation CD contains a copy of the eCos GDB stubs in binary format which must be programmed into an EPROM or FLASH and installed on the board.

Preparing the GDB stubs

These stub preparation steps are not strictly necessary as the eCos distribution ships with pre-compiled stubs in the directory `loaders/sh3-cq7708` relative to the installation root.

Building the GDB stub image with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the SH3 cq7708 hardware.
3. While still displaying the *Build->Templates* dialog box, select the stubs package template to build a GDB stub. Click *OK*.
4. Build eCos stubs using *Build->Library*.
5. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub images have the prefix `gdb_module`.

Building the GDB stub image with ecosconfig

1. Make an empty directory to contain the build tree, and `cd` into it.
2. To build a GDB stub ROM image, enter the command:


```
$ ecosconfig new cq7708 stubs
```
3. Enter the commands:


```
$ ecosconfig tree
$ make
```
4. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub images have the prefix `gdb_module`.

Programming the stubs in EPROM/FLASH

The board can use different sizes of ROMs. Use this table to adjust the board's jumpers to the ROM sizes you are using.

size (kbit)	JP7	JP9	JP10	JP11
256	2-3	2-3	open	open
512	1-2	2-3	open	open
1000	1-2	open	open	2-3
2000	1-2	1-2	open	2-3
4000	1-2	1-2	short	2-3
8000	1-2	1-2	short	1-2

There are two ways to program the stubs. We advise you to use method 1, since it is simpler. Method 2 is unsupported and requires a bit of fiddling.

Method 1:

Program the binary stub image into two EPROMs, E and O. EPROM E should contain the even bytes, and O the odd bytes (your EPROM programmer should have the ability to split the image).

EPROM E should be installed in socket IC8, and EPROM O should be installed in socket IC4.

Set JP6 to 16 bit mode (1-2 soldered, 2-3 cut) Set SW1-4 to ON and SW2-1 to OFF.

Method2:

Assuming that the stub binary is smaller than 32 KB, you can install it in a single EPROM.

Compile the `mkcqrom.c` program found in the `misc` directory.

Use it to convert the binary image to the required format. See the `mkcqrom.c` source for a description of what is done, and why it is necessary.

```
% mkcqrom gdb_module.bin gdb_mangled.bin
```

Program the `gdb_mangled.bin` file into an EPROM and install it in socket IC4

Set JP6 to 8 bit mode (cut 1-2, solder 2-3)

The GDB stubs allow communication with GDB using the serial port at connector CN7. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a straight through serial cable.

SH3/HS7729PCI Hardware Setup

Please see the RedBoot manual for instructions on how to prepare the board for use with eCos.

SH3/SE77x9 Hardware Setup

Please see the RedBoot manual for instructions on how to prepare the board for use with eCos.

SH4/CQ7750 Hardware Setup

Preparing the board

Make sure the DIP switches on the board are set as follows:

SW1-1 ON
SW1-2 OFF
SW1-3 ON
SW1-4 OFF

SW2-1 ON
SW2-2 ON
SW2-3 OFF
SW2-4 OFF

If you are using a straight through serial cable which has flow control lines, you will also need to cut JP12 (5-6) as the flow control lines can cause NMIs.

eCos GDB Stubs

The eCos installation CD contains a copy of the eCos GDB stubs in binary format which must be programmed into an EPROM or FLASH and installed on the board.

Preparing the GDB stubs

These stub preparation steps are not strictly necessary as the eCos distribution ships with pre-compiled stubs in the directory `loaders/sh3-cq7708` relative to the installation root.

Building the GDB stub image with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the SH3 cq7708 hardware.
3. While still displaying the *Build->Templates* dialog box, select the stubs package template to build a GDB stub. Click *OK*.
4. Build eCos stubs using *Build->Library*.
5. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub images have the prefix `gdb_module`.

Building the GDB stub image with ecosconfig

1. Make an empty directory to contain the build tree, and `cd` into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new cq7708 stubs
```

3. Enter the commands:

```
$ ecosconfig tree
$ make
```

4. When the build completes, the image files can be found in the `bin/` subdirectory of the install tree. GDB stub images have the prefix `gdb_module.`

Programming the stubs in EPROM/FLASH

The board can use different sizes of ROMs. Use this table to adjust the board's jumpers to the ROM sizes you are using.

size(kbit)	JP7	JP9	JP10	JP11
256	2-3	2-3	open	open
512	1-2	2-3	open	open
1000	1-2	open	open	2-3
2000	1-2	1-2	open	2-3
4000	1-2	1-2	short	2-3
8000	1-2	1-2	short	1-2

There are two ways to program the stubs. We advise you to use method 1, since it is simpler. Method 2 is unsupported and requires a bit of fiddling.

Method 1:

Program the binary stub image into two EPROMs, E and O. EPROM E should contain the even bytes, and O the odd bytes (your EPROM programmer should have the ability to split the image).

EPROM E should be installed in socket IC8, and EPROM O should be installed in socket IC4.

Set JP6 to 16 bit mode (1-2 soldered, 2-3 cut) Set SW1-4 to ON and SW2-1 to OFF.

Method2:

Assuming that the stub binary is smaller than 32 KB, you can install it in a single EPROM.

Compile the `mkcqrom.c` program found in the `misc` directory.

Use it to convert the binary image to the required format. See the `mkcqrom.c` source for a description of what is done, and why it is necessary.

```
% mkcqrom gdb_module.bin gdb_mangled.bin
```

Program the `gdb_mangled.bin` file into an EPROM and install it in socket IC4

Set JP6 to 8 bit mode (cut 1-2, solder 2-3)

The GDB stubs allow communication with GDB using the serial port at connector CN7. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a straight through serial cable.

SH4/SE7751 Hardware Setup

Please see the RedBoot manual for instructions on how to prepare the board for use with eCos.

NEC CEB-V850/SA1 Hardware Setup

The CEB-V850 board is fitted with a socketed EPROM. The internal Flash of the V850 supplied with the CEB-V850 boards defaults to vectoring into this EPROM. A GDB stub image should be programmed into an EPROM fitted to this board, and a pre-built image is provided at `loaders/v850-ceb_v850/v850sa1/gdb_module.bin` under the root of your eCos installation.

The EPROM is installed to the socket labelled U7 on the board. Attention should be paid to the correct orientation of the EPROM during installation.

When programming an EPROM using the binary image, be careful to get the byte order correct. It needs to be little-endian. If the EPROM burner software has a hex-editor, check that the first few bytes of the image look similar to:

```
00000000: 0018 8007 5e02 0000 0000 0000 0000 0000
```

If the byte order is wrong you will see 1800 instead of 0018 etc. Use the EPROM burner software to make a byte-swap before you burn to image to the EPROM.

If the GDB stub EPROM you burn does not work, try reversing the byte-order, even if you think you have it the right way around. At least one DOS-based EPROM burner program is known to have the byte-order upside down.

The GDB stub in the EPROM allows communication with GDB using the serial port. The communication parameters are fixed at 38400 baud, 8 data bits, no parity bit and 1 stop bit (8-N-1). No flow control is employed. Connection to the host computer should be made using a dedicated serial cable as specified in the CEB-V850/SA1 manual.

Installing the Stubs into ROM

Preparing the Binaries

These two binary preparation steps are not strictly necessary as the eCos distribution ships with pre-compiled binaries in the directory `loaders/v850-ceb_v850` relative to the installation root.

Building the ROM images with the eCos Configuration Tool

1. Start with a new document - selecting the *File->New* menu item if necessary to do this.
2. Choose the *Build->Templates* menu item, and then select the NEC CEB-V850/SA1 hardware.
3. While still displaying the *Build->Templates* dialog box, select the “stubs” package template to build a GDB stub. Click *OK*.
4. Build eCos using *Build->Library*.

5. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. GDB stub ROM images have the prefix “gdb_module”.

Building the ROM images with ecosconfig

1. Make an empty directory to contain the build tree, and cd into it.
2. To build a GDB stub ROM image, enter the command:

```
$ ecosconfig new ceb-v850 stubs
```

3. Enter the commands:

```
$ ecosconfig tree  
$ make
```

4. When the build completes, the image files can be found in the bin/ subdirectory of the install tree. GDB stub ROM images have the prefix “gdb_module”.

Installing the Stubs into ROM or FLASH

1. Program the binary image file gdb_module.bin into ROM or FLASH referring to the instructions of your ROM programmer.
2. Plug the ROM/FLASH into the socket as described at the beginning of this section.

Debugging with the NEC V850 I.C.E.

eCos applications may be debugged using the NEC V850 In Circuit Emulator (I.C.E.) A PC running Microsoft Windows is required in order to run the NEC ICE software and drivers. In addition Red Hat have developed a “libremote” server application named v850ice.exe which is used on the PC connected to the I.C.E. in order to allow connections from GDB.

The I.C.E. must be physically connected to a Windows NT system through NEC's PCI or PC Card interface. A driver, DLLs, and application are provided by NEC to control the I.C.E.

v850ice is a Cygwin based server that runs on the NT system and provides an interface between the gdb client and the I.C.E. software. v850-elf-gdb may be run on the Windows NT system or on a remote system. v850-elf-gdb communicates with the libremote server using the gdb remote protocol over a TCP/IP socket. v850ice communicates with the I.C.E. by calling functions in the NECMSG.DLL provided by NEC.

INITIAL SETUP

1. Configure the hardware including the I.C.E., SA1 or SB1 Option Module, and target board. Install the interface card in the Windows NT system. Reference NEC's documentation for interface installation, jumper settings, etc.
2. Install the Windows NT device driver provided by NEC.

3. Copy the NEC DLLs, MDI application, and other support files to a directory on the Windows NT system. The standard location is C:\NecTools32. This directory will be referred to as the "libremote server directory" in this document. v850ice.exe must also be copied to this directory after being built. The required files are: cpu.cfg, Nec.cfg, MDI.EXE, NECMSG.DLL, EX85032.DLL, V850E.DLL, IE850.MON, IE850E.MON, and D3037A.800.
4. Make certain the file cpu.cfg contains the line:

```
CpuOption=SA1
```

 if using a V850/SA1 module, or:

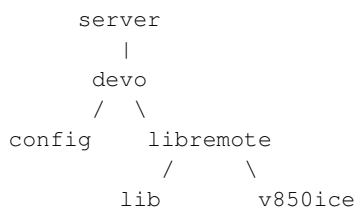
```
CpuOption=SB1
```

 if using a V850/SB1 module.
5. Set the environment variable IEPATH to point to the libremote server directory.

BUILD PROCEDURES

A pre-built v850ice.exe executable is supplied in the loaders/v850-ceb_v850 directory relative to the root of the eCos installation. However the following process will allow the rebuilding of this executable if required:

For this example assume the v850ice libremote tree has been copied to a directory named "server". The directory structure will be similar to the following diagram:



Build the v850ice source as follows. Be sure to use the native Cygwin compiler tools that were supplied alongside eCos.

```
cd server mkdir build cd build ../devo/configure --target=v850-elf --host=i686-pc-cygwin make
```

The resultant libremote server image (v850ice.exe) can be found in build/libremote/v850ice. Copy v850ice.exe to the lib remote server directory.

V850ICE.EXE EXECUTION

The v850ice command line syntax is:

```
v850ice [-d] [-t addr] [port number]
```

The optional -d option enables debug output. The -t option is associated with thread debugging - see the "eCos thread debugging" section below for details. By default v850ice listens on port 2345 for an attach request from a gdb client. A different port number may be specified on the command line.

To run the libremote server:

1. Power on the I.C.E. and target board.
2. Open a Cygwin window.
3. Run v850ice.
4. You will see the MDI interface window appear. In this window you should see the "Connected to In-Circuit Emulator" message. In the Cygwin window, the libremote server will indicate it is ready to accept a gdb client connection with the message "v850ice: listening on port 2345."

V850-ELF-GDB EXECUTION

Run the v850-elf-gdb client to debug the V850 target. It is necessary to issue certain configuration commands to the I.C.E. software. These commands may be issued directly in the MDI window or they may be issued from the gdb client through the "monitor" command.

On the Cosmo CEB-V850 board, on-chip Flash is mapped at address 0x0, the on-board EPROM at 0x100000 and the on-board RAM at 0xfc0000. Since a stand alone V850 will start executing from address 0x0 on reset, it is normal to load either an application or a bootstrap loader for Flash at this address. eCos programs may be built to boot from Flash or the on-board EPROM. If building for the on-board EPROM, it would be expected that the Flash will contain the default CEB-V850 flash contents. An ELF format version of the default contents may be found in the eCos distribution with the name v850flash.img.

In stand alone operation, normally the code in this flash image would have been programmed into the V850 on the Cosmo board, and this would cause it to vector into the on-board EPROM to run the application located there. In the case of eCos, this application may be a GDB stub ROM application, allowing the further download to RAM over serial of actual applications to debug.

As an example, we shall demonstrate how to use the I.C.E. to download the v850flash.img and GDB stub EPROM image using I.C.E. emulator memory only, and not requiring any actual programming of devices.

```
v850-elf-gdb -nw (gdb) file v850flash.img (gdb) target remote localhost:2345 (gdb) monitor reset (gdb) monitor
cpu r=256 a=16 (gdb) monitor map r=0x100000-L 0x80000 (gdb) monitor map u=0xfc0000-L 0x40000 (gdb)
monitor pinmask k (gdb) monitor step (gdb) monitor step (gdb) monitor step (gdb) monitor step (gdb) load (gdb)
detach (gdb) file gdb_module.img (gdb) target remote localhost:2345 (gdb) load (gdb) continue
```

NOTE: The four "monitor step" commands are only required the first time the board is connected to the I.C.E., otherwise the program will fail.

This is because of a limitation of the I.C.E. hardware that means that the first time it is used, the "map" commands are not acted on and the addresses "0x100000" and "0xfc0000" are not mapped. This can be observed using the command "td e-20" in the MDI application's console to display the trace buffer, which will show that the contents of address 0x100000 are not valid. Subsequent runs do not require the "monitor step" commands.

It is unusual to load two executable images to a target through gdb. From the example above notice that this is accomplished by attaching to the libremote server, loading the flash image, detaching, reattaching, and loading the ROM/RAM image. It is more normal to build an executable image that can be executed directly. In eCos this is achieved by selecting either the ROM or ROMRAM startup type, and optionally enable building for the internal FLASH. The I.C.E. emulator memory can emulate both the internal FLASH and the EPROM, so real hardware programming is not required.

Upon running this example you will notice that the libremote server does not exit upon detecting a detach request, but simply begins listening for the next attach request. To cause v850ice to terminate, issue the "monitor quit" or "monitor exit" command from the gdb client. v850ice will then terminate with the next detach request. (You can also enter control-c in the Cygwin/DOS window where v850ice is running.)

MDI INTERFACE VS. GDB INTERFACE

If a filename is referenced in an MDI command, whether the command is entered in the MDI window or issued from the gdb client with the monitor command, the file must reside on the Windows NT libremote server system. When specifying a filename when entering a command in the MDI window it is obvious that a server local file is being referenced. When issuing an MDI command from the gdb client, the user must remember that the command line is simply passed to the I.C.E. software on the server system. The command is executed by the I.C.E. software as though it were entered locally.

Executable images may be loaded into the V850 target by entering the "load" command in the MDI window or with the gdb "load" command. If the MDI load command is used, the executable image must be located on the server system and must be in S Record format. If the gdb load command is used, the executable image must be located on the client system and must be in ELF format.

Be aware that the gdb client is not aware of debugger commands issued from the MDI window. It is possible to cause the gdb client and the I.C.E. software to get out of sync by issuing commands from both interfaces during the same debugging session.

eCos THREAD DEBUGGING

eCos and the V850 I.C.E. libremote server have been written to work together to allow debugging of eCos threads. This is an optional feature, disabled by default because of the overheads trying to detect a threaded program involves.

Obviously thread debugging is not possible for programs with "RAM" startup type, as they are expected to operate underneath a separate ROM monitor (such as a GDB stub ROM), that itself would provide its own thread debugging capabilities over the serial line. Thread debugging is relevant only for programs built for Flash, ROM, or ROMRAM startup.

To configure the libremote server to support thread debugging, use the command:

```
(gdb) monitor syscallinfo ADDRESS
```

at the GDB console prompt, where ADDRESS is the address of the syscall information structure included in the applications. In eCos this has been designed to be located at a consistent address for each CPU model (V850/SA1 or V850/SB1). It may be determined from an eCos executable using the following command at a cygwin bash prompt:

```
v850-elf-nm EXECUTABLE | grep hal_v85x_ice_syscall_info
```

At the current time, this address is 0xfc0400 for a Cosmo board fitted with a V850/SA1, or 0xfc0540 for a Cosmo board fitted with a V850/SB1.

So for example, the GDB command for the SB1 would be:

```
(gdb) monitor syscallinfo 0xfc0540
```

Given that the syscallinfo address is fixed over all eCos executables for a given target, it is possible to define it on the libremote command line as well using the "-t" option, for example:

```
bash$ v850ice -t 0xfc0400
v850ice: listening on port 2345
```

NEC CEB-V850/SB1 Hardware Setup

The instructions for setting up the CEB-V850/SB1 are virtually identical to those of the CEB-V850/SA1 above. The only significant differences are that pre-built loaders are available at loaders/v850-ceb_v850/v850sb1 within the eCos installation. Binaries supporting boards with both 16MHz and 8MHz clock speeds are supplied. Also when building applications, or rebuilding the stubs for a V850/SB1 target, then the V850 CPU variant must be changed in the CEB-V850 HAL to the SB1.

i386 PC Hardware Setup

eCos application on the PC can be run in three ways: via RedBoot, loaded directly from a floppy disk, or loaded by the GRUB bootloader.

RedBoot Support

For information about setting up the PC to run with RedBoot, consult the RedBoot User's Guide. If using serial debugging, the serial line runs at 38400 baud 8-N-1 and should be connected to the debug host using a null modem cable. If ethernet debugging is required, an i82559 compatible network interface card, such as an Intel EtherExpress Pro 10/100, should be installed on the target PC and connected to the development PC running GDB. When RedBoot is configured appropriately to have an IP address set, then GDB will be able to debug directly over TCP/IP to the target PC.

Floppy Disk Support

If an application is built with a startup type of FLOPPY, then it is configured to be a self-booting image that must be written onto a formatted floppy disk. This will erase any existing file system or data that is already on the disk, so proceed with caution.

To write an application to floppy disk, it must first be converted to a pure binary format. This is done with the following command:

```
$ i386-elf-objcopy -O binary app.elf app.bin
```

Here `app.elf` is the final linked application executable, in ELF format (it may not have a `.elf` extension). The file `app.bin` is the resulting pure binary file. This must be written to the floppy disk with the following command:

```
$ dd conv=sync if=app.bin of=/dev/fd0
```

For NT Cygwin users, this can be done by first ensuring that the raw floppy device is mounted as `/dev/fd0`. To check if this is the case, type the command **mount** at the Cygwin bash prompt. If the floppy drive is already mounted, it will be listed as something similar to the following line:

```
\\.\a: /dev/fd0 user binmode
```

If this line is not listed, then mount the floppy drive using the command:

```
$ mount -f -b //./a: /dev/fd0
```

To actually install the boot image on the floppy, use the command:

```
$ dd conv=sync if=app.bin of=/dev/fd0
```

Insert this floppy in the A: drive of the PC to be used as a target and ensure that the BIOS is configured to boot from A: by default. On reset, the PC will boot from the floppy and the eCos application will load itself and execute immediately.

NOTE: Unreliable floppy media may cause the write to silently fail. This can be determined if the RedBoot image does not correctly boot. In such cases, the floppy should be (unconditionally) reformatted using the **fdformat** command on Linux, or **format a: /u** on DOS/Windows. If this fails, try a different disk.

GRUB Bootloader Support

If an application is built with the GRUB startup type, it is configured to be loaded by the GRUB bootloader.

GRUB is an open source boot loader that supports many different operating systems. It is available from <http://www.gnu.org/software/grub>. The latest version of GRUB should be downloaded from there and installed. In Red Hat Linux version 7.2 and later it is the default bootloader for Linux and therefore is already installed.

To install GRUB on a floppy disk from Linux you need to execute the following commands:

```
$ mformat a:
$ mount /mnt/floppy
$ grub-install --root-directory=/mnt/floppy '(fd0)'
Probing devices to guess BIOS drives. This may take a long time.
Installation finished. No error reported.
This is the contents of the device map /mnt/floppy/boot/grub/device.map.
Check if this is correct or not. If any of the lines is incorrect,
```

```
fix it and re-run the script 'grub-install'.

(fd0) /dev/fd0
$ cp $ECOS_REPOSITORY/packages/hal/i386/pc/current/misc/menu.lst /mnt/floppy/boot/grub
$ umount /mnt/floppy
```

The file `menu.lst` is an example GRUB menu configuration file. It contains menu items to load some of the standard eCos tests from floppy or from partition zero of the first hard disk. You should, of course, customize this file to load your own application. Alternatively you can use the command-line interface of GRUB to input commands yourself.

Applications can be installed, or updated simply by copying them to the floppy disk at the location expected by the `menu.lst` file. For booting from floppy disks it is recommended that the executable be stripped of all debug and symbol table information before copying. This reduces the size of the file and can make booting faster.

To install GRUB on a hard disk, refer to the GRUB documentation. Be warned, however, that if you get this wrong it may compromise any existing bootloader that exists on the hard disk and may make any other operating systems unbootable. Practice on floppy disks or sacrificial hard disks first. On machines running Red Hat Linux version 7.2 and later, you can just add your own menu items to the `/boot/grub/menu.lst` file that already exists.

Debugging FLOPPY and GRUB Applications

When RedBoot loads an application it also provides debugging services in the form of GDB remote protocol stubs. When an application is loaded stand-alone from a floppy disk, or by GRUB, these services are not present. To allow these application to be debugged, it is possible to include GDB stubs into the application.

To do this, set the "Support for GDB stubs" (`CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS`) configuration option. Following this any application built will allow GDB to connect to the debug serial port (by default serial device 0, also known as COM1) whenever the application takes an exception, or if a Control-C is typed to the debug port. Ethernet debugging is not supported.

The option "Enable initial breakpoint" (`CYGDBG_HAL_DEBUG_GDB_INITIAL_BREAK`) causes the HAL to take a breakpoint immediately before calling `cyg_start()`. This gives the developer a chance to set any breakpoints or inspect the system state before it proceeds. The configuration sets this option by default if GDB stubs are included, and this is not a RedBoot build. To make the application execute immediately either disable this option, or disable `CYGDBG_HAL_DEBUG_GDB_INCLUDE_STUBS`.

i386/Linux Synthetic Target Setup

When building for the synthetic Linux target, the resulting binaries are native Linux applications with the HAL providing suitable bindings between the eCos kernel and the Linux kernel.

Note: Please be aware that the current implementation of the Linux synthetic target does not allow thread-aware debugging.

These Linux applications cannot be run on a Windows system. However, it is possible to write a similar HAL emulation for the Windows kernel if such a testing target is desired.

Tools

For the synthetic target, eCos relies on features not available in native compilers earlier than gcc-2.95.1. It also requires version 2.9.5 or later of the GNU linker. If you have gcc-2.95.1 or later and ld version 2.9.5 or later, then you do not need to build new tools. eCos does not support earlier versions. You can check the compiler version using **gcc -v** and the linker version using **ld -v**.

If you have native tools that are sufficiently recent for use with eCos, you should be aware that by default eCos assumes that the tools **i686-pc-linux-gnu-gcc**, **i686-pc-linux-gnu-ar**, **i686-pc-linux-gnu-ld**, and **i686-pc-linux-gnu-objcopy** are on your system and are the correct versions for use with eCos. But instead, you can tell eCos to use your native tools by editing the configuration value "Global command prefix" (CYGBLD_GLOBAL_COMMAND_PREFIX) in your eCos configuration. If left empty (i.e. set to the empty string) eCos will use your native tools when building.

If you have any difficulties, it is almost certainly easiest overall to rebuild the tools as described on: <http://ecos.sourceware.org/getstart.html>

Appendix B. Real-time characterization

For a discussion of real-time performance measurement for eCos, see the kernel documentation in the eCos Reference Manual.

Caution

As with the target setup descriptions in the previous appendix, this information will eventually be merged into per-target documents.

Sample numbers:

Board: ARM AEB-1 Revision B Evaluation Board

Board: ARM AEB-1 Revision B Evaluation Board

CPU : Sharp LH77790A 24MHz

Startup, main stack : stack used 404 size 2400
Startup : Interrupt stack used 128 size 2048
Startup : Idlethread stack used 80 size 2048

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 13 'ticks' overhead
... this value will be factored out of all other measurements
Clock interrupt took 193.49 microseconds (290 raw clock ticks)

Testing parameters:

Clock samples: 32
Threads: 7
Thread switches: 128
Mutexes: 32
Mailboxes: 32
Semaphores: 32
Scheduler operations: 128
Counters: 32
Alarms: 32

			Confidence			Function
Ave	Min	Max	Var	Ave	Min	
=====	=====	=====	=====	=====	=====	=====
110.19	104.67	116.00	3.26	42%	28%	Create thread
34.00	34.00	34.00	0.00	100%	100%	Yield thread [all suspended]
24.67	24.67	24.67	0.00	100%	100%	Suspend [suspended] thread
25.05	24.67	25.33	0.33	57%	42%	Resume thread
37.14	36.67	37.33	0.27	71%	28%	Set priority

Appendix B. Real-time characterization

3.81	3.33	4.00	0.27	71%	28%	Get priority
80.00	80.00	80.00	0.00	100%	100%	Kill [suspended] thread
33.90	33.33	34.00	0.16	85%	14%	Yield [no other] thread
45.90	44.00	46.67	0.54	57%	14%	Resume [suspended low prio] thread
24.57	24.00	24.67	0.16	85%	14%	Resume [runnable low prio] thread
42.29	36.67	43.33	1.61	85%	14%	Suspend [runnable] thread
33.90	33.33	34.00	0.16	85%	14%	Yield [only low prio] thread
24.67	24.67	24.67	0.00	100%	100%	Suspend [runnable->not runnable]
80.00	80.00	80.00	0.00	100%	100%	Kill [runnable] thread
43.33	43.33	43.33	0.00	100%	100%	Destroy [dead] thread
106.29	101.33	107.33	1.41	85%	14%	Destroy [runnable] thread
144.95	141.33	166.00	6.01	85%	85%	Resume [high priority] thread
78.31	76.67	254.67	2.75	99%	99%	Thread switch
4.00	4.00	4.00	0.00	100%	100%	Scheduler lock
16.37	16.00	16.67	0.33	56%	43%	Scheduler unlock [0 threads]
16.37	16.00	16.67	0.33	56%	43%	Scheduler unlock [1 suspended]
16.37	16.00	16.67	0.33	56%	43%	Scheduler unlock [many suspended]
16.37	16.00	16.67	0.33	56%	43%	Scheduler unlock [many low prio]
10.67	10.67	10.67	0.00	100%	100%	Init mutex
28.67	28.67	28.67	0.00	100%	100%	Lock [unlocked] mutex
30.44	30.00	31.33	0.33	59%	37%	Unlock [locked] mutex
25.42	25.33	26.00	0.15	87%	87%	Trylock [unlocked] mutex
22.50	22.00	22.67	0.25	75%	25%	Trylock [locked] mutex
5.75	5.33	6.00	0.31	62%	37%	Destroy mutex
185.33	185.33	185.33	0.00	100%	100%	Unlock/Lock mutex
20.17	20.00	20.67	0.25	75%	75%	Create mbox
2.92	2.67	3.33	0.31	62%	62%	Peek [empty] mbox
32.42	32.00	32.67	0.31	62%	37%	Put [first] mbox
3.00	2.67	3.33	0.33	100%	50%	Peek [1 msg] mbox
32.50	32.00	32.67	0.25	75%	25%	Put [second] mbox
2.92	2.67	3.33	0.31	62%	62%	Peek [2 msgs] mbox
32.83	32.67	33.33	0.25	75%	75%	Get [first] mbox
32.67	32.67	32.67	0.00	100%	100%	Get [second] mbox
31.33	31.33	31.33	0.00	100%	100%	Tryput [first] mbox
27.58	27.33	28.00	0.31	62%	62%	Peek item [non-empty] mbox
32.83	32.67	33.33	0.25	75%	75%	Tryget [non-empty] mbox
26.50	26.00	26.67	0.25	75%	25%	Peek item [empty] mbox
28.00	28.00	28.00	0.00	100%	100%	Tryget [empty] mbox
3.25	2.67	3.33	0.15	87%	12%	Waiting to get mbox
3.25	2.67	3.33	0.15	87%	12%	Waiting to put mbox
30.83	30.67	31.33	0.25	75%	75%	Delete mbox
101.08	100.67	101.33	0.31	62%	37%	Put/Get mbox
11.17	10.67	11.33	0.25	75%	25%	Init semaphore
24.17	24.00	24.67	0.25	75%	75%	Post [0] semaphore
27.08	26.67	27.33	0.31	62%	37%	Wait [1] semaphore
22.75	22.67	23.33	0.15	87%	87%	Trywait [0] semaphore
22.21	22.00	22.67	0.29	68%	68%	Trywait [1] semaphore
7.33	7.33	7.33	0.00	100%	100%	Peek semaphore
5.92	5.33	6.00	0.15	87%	12%	Destroy semaphore
110.04	110.00	110.67	0.08	93%	93%	Post/Wait semaphore

9.54	9.33	10.00	0.29	68%	68%	Create counter
3.92	3.33	4.00	0.15	87%	12%	Get counter value
4.00	4.00	4.00	0.00	100%	100%	Set counter value
30.92	30.67	31.33	0.31	62%	62%	Tick counter
5.75	5.33	6.00	0.31	62%	37%	Delete counter
13.83	13.33	14.00	0.25	75%	25%	Create alarm
46.67	46.67	46.67	0.00	100%	100%	Initialize alarm
3.67	3.33	4.00	0.33	100%	50%	Disable alarm
45.67	45.33	46.00	0.33	100%	50%	Enable alarm
8.33	8.00	8.67	0.33	100%	50%	Delete alarm
36.33	36.00	36.67	0.33	100%	50%	Tick counter [1 alarm]
214.67	214.67	214.67	0.00	100%	100%	Tick counter [many alarms]
62.67	62.67	62.67	0.00	100%	100%	Tick & fire counter [1 alarm]
1087.04	1075.33	1278.67	21.91	93%	93%	Tick & fire counters [>1 together]
246.35	240.67	412.00	10.35	96%	96%	Tick & fire counters [>1 separately]
168.01	167.33	237.33	1.08	99%	99%	Alarm latency [0 threads]
187.36	168.00	234.67	3.60	86%	1%	Alarm latency [2 threads]
187.37	167.33	235.33	3.59	85%	1%	Alarm latency [many threads]
303.12	280.00	508.67	3.21	98%	0%	Alarm -> thread resume latency
36.65	36.00	38.67	0.00			Clock/interrupt latency
65.79	52.00	152.67	0.00			Clock DSR latency
316	316	316	(main stack: 752)	Thread stack used (1120 total)		
All done, main stack			:	stack used	752	size 2400
All done			:	Interrupt stack used	280	size 2048
All done			:	Idlethread stack used	268	size 2048
Timing complete - 30390 ms total						
PASS:<Basic timing OK>						
EXIT:<done>						

Board: Atmel AT91/EB40

Board: Atmel AT91/EB40
 CPU : AT91R40807 (ARM7TDMI core), 32MHz
 512KB RAM, 64K Flash

Startup, main stack	:	stack used	420	size	2400
Startup	:	Interrupt stack used	144	size	4096
Startup	:	Idlethread stack used	84	size	2048

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 3 'ticks' overhead
 ... this value will be factored out of all other measurements

Appendix B. Real-time characterization

Clock interrupt took 127.53 microseconds (130 raw clock ticks)

Testing parameters:

```

Clock samples:      32
Threads:            25
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32

```

				Confidence			Function
Ave	Min	Max	Var	Ave	Min		
=====	=====	=====	=====	=====	=====	=====	
86.48	71.29	101.56	7.99	48%	28%	Create thread	
20.70	20.51	21.48	0.31	80%	80%	Yield thread [all suspended]	
17.15	16.60	17.58	0.48	56%	44%	Suspend [suspended] thread	
17.07	16.60	17.58	0.49	52%	52%	Resume thread	
25.51	25.39	26.37	0.21	88%	88%	Set priority	
3.16	2.93	3.91	0.36	76%	76%	Get priority	
52.34	51.76	52.73	0.47	60%	40%	Kill [suspended] thread	
20.70	20.51	21.48	0.31	80%	80%	Yield [no other] thread	
28.98	28.32	30.27	0.48	60%	36%	Resume [suspended low prio] thread	
17.11	16.60	17.58	0.49	52%	48%	Resume [runnable low prio] thread	
27.85	26.37	28.32	0.52	96%	4%	Suspend [runnable] thread	
20.70	20.51	21.48	0.31	80%	80%	Yield [only low prio] thread	
17.23	16.60	17.58	0.45	64%	36%	Suspend [runnable->not runnable]	
52.34	51.76	52.73	0.47	60%	40%	Kill [runnable] thread	
33.01	32.23	33.20	0.31	80%	20%	Destroy [dead] thread	
72.03	70.31	72.27	0.38	80%	4%	Destroy [runnable] thread	
96.99	95.70	112.30	1.22	64%	96%	Resume [high priority] thread	
51.48	49.80	164.06	1.76	99%	99%	Thread switch	
2.78	1.95	2.93	0.26	84%	15%	Scheduler lock	
11.81	11.72	12.70	0.17	90%	90%	Scheduler unlock [0 threads]	
11.81	11.72	12.70	0.17	90%	90%	Scheduler unlock [1 suspended]	
11.81	11.72	12.70	0.17	90%	90%	Scheduler unlock [many suspended]	
11.81	11.72	12.70	0.17	90%	90%	Scheduler unlock [many low prio]	
5.49	4.88	5.86	0.46	62%	37%	Init mutex	
20.20	19.53	20.51	0.42	68%	31%	Lock [unlocked] mutex	
24.44	24.41	25.39	0.06	96%	96%	Unlock [locked] mutex	
18.25	17.58	18.55	0.42	68%	31%	Trylock [unlocked] mutex	
16.11	15.63	16.60	0.49	100%	50%	Trylock [locked] mutex	
6.10	5.86	6.84	0.37	75%	75%	Destroy mutex	
124.21	124.02	125.00	0.30	81%	81%	Unlock/Lock mutex	
9.28	8.79	9.77	0.49	100%	50%	Create mbox	
2.93	2.93	2.93	0.00	100%	100%	Peek [empty] mbox	
22.58	22.46	23.44	0.21	87%	87%	Put [first] mbox	
2.44	1.95	2.93	0.49	100%	50%	Peek [1 msg] mbox	
22.58	22.46	23.44	0.21	87%	87%	Put [second] mbox	

Appendix B. Real-time characterization

2.44	1.95	2.93	0.49	100%	50%	Peek [2 msgs] mbox
22.71	22.46	23.44	0.37	75%	75%	Get [first] mbox
22.71	22.46	23.44	0.37	75%	75%	Get [second] mbox
21.18	20.51	21.48	0.42	68%	31%	Tryput [first] mbox
18.98	18.55	19.53	0.48	56%	56%	Peek item [non-empty] mbox
22.46	22.46	22.46	0.00	100%	100%	Tryget [non-empty] mbox
18.31	17.58	18.55	0.37	75%	25%	Peek item [empty] mbox
19.53	19.53	19.53	0.00	100%	100%	Tryget [empty] mbox
2.69	1.95	2.93	0.37	75%	25%	Waiting to get mbox
2.93	2.93	2.93	0.00	100%	100%	Waiting to put mbox
23.86	23.44	24.41	0.48	56%	56%	Delete mbox
67.60	67.38	68.36	0.33	78%	78%	Put/Get mbox
5.37	4.88	5.86	0.49	100%	50%	Init semaphore
16.97	16.60	17.58	0.46	62%	62%	Post [0] semaphore
18.98	18.55	19.53	0.48	56%	56%	Wait [1] semaphore
15.81	15.63	16.60	0.30	81%	81%	Trywait [0] semaphore
15.29	14.65	15.63	0.44	65%	34%	Trywait [1] semaphore
5.62	4.88	5.86	0.37	75%	25%	Peek semaphore
6.35	5.86	6.84	0.49	100%	50%	Destroy semaphore
72.36	72.27	73.24	0.17	90%	90%	Post/Wait semaphore
7.08	6.84	7.81	0.37	75%	75%	Create counter
3.17	2.93	3.91	0.37	75%	75%	Get counter value
3.05	2.93	3.91	0.21	87%	87%	Set counter value
24.11	23.44	24.41	0.42	68%	31%	Tick counter
5.49	4.88	5.86	0.46	62%	37%	Delete counter
10.92	10.74	11.72	0.30	81%	81%	Create alarm
31.46	31.25	32.23	0.33	78%	78%	Initialize alarm
3.05	2.93	3.91	0.21	87%	87%	Disable alarm
31.49	31.25	32.23	0.37	75%	75%	Enable alarm
7.02	6.84	7.81	0.30	81%	81%	Delete alarm
31.16	30.27	31.25	0.17	90%	9%	Tick counter [1 alarm]
309.26	304.69	425.78	7.28	96%	96%	Tick counter [many alarms]
44.83	43.95	44.92	0.17	90%	9%	Tick & fire counter [1 alarm]
781.68	774.41	893.55	13.62	93%	93%	Tick & fire counters [>1 together]
324.16	320.31	433.59	6.84	96%	96%	Tick & fire counters [>1 separately]
114.26	113.28	167.97	0.84	57%	42%	Alarm latency [0 threads]
126.91	113.28	159.18	8.20	50%	31%	Alarm latency [2 threads]
127.11	113.28	158.20	8.09	51%	28%	Alarm latency [many threads]
196.49	189.45	331.05	2.10	98%	0%	Alarm -> thread resume latency
23.50	23.44	25.39	0.00			Clock/interrupt latency
40.31	33.20	514.65	0.00			Clock DSR latency
300	271	312	(main stack: 832)			Thread stack used (1120 total)
All done, main stack			: stack used	832	size	2400
All done			: Interrupt stack used	288	size	4096
All done			: Idlethread stack used	272	size	2048

Timing complete - 30350 ms total

PASS:<Basic timing OK>

EXIT:<done>

Board: Intel StrongARM EBSA-285 Evaluation Board

Board: Intel StrongARM EBSA-285 Evaluation Board

CPU : Intel StrongARM SA-110 228MHz

```
Startup, main stack      : stack used  404 size 2400
Startup                  : Interrupt stack used 136 size 4096
Startup                  : Idlethread stack used  80 size 2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 1 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 4.61 microseconds (16 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

Ave	Min	Max	Var	Confidence	Function
=====	=====	=====	=====	=====	=====
4.97	3.26	7.34	0.60	50% 4%	Create thread
0.73	0.54	2.17	0.14	60% 37%	Yield thread [all suspended]
0.98	0.82	2.99	0.23	81% 68%	Suspend [suspended] thread
0.54	0.27	1.63	0.03	92% 6%	Resume thread
0.83	0.54	1.90	0.10	73% 14%	Set priority
0.21	0.00	0.54	0.21	25% 48%	Get priority
2.25	1.90	10.05	0.37	96% 67%	Kill [suspended] thread
0.70	0.54	1.09	0.14	53% 45%	Yield [no other] thread
0.96	0.82	1.36	0.14	50% 48%	Resume [suspended low prio] thread
0.53	0.27	0.82	0.03	92% 6%	Resume [runnable low prio] thread
0.90	0.82	1.63	0.13	70% 70%	Suspend [runnable] thread
0.70	0.54	0.82	0.13	57% 42%	Yield [only low prio] thread
0.55	0.54	0.82	0.01	98% 98%	Suspend [runnable->not runnable]
1.64	1.63	2.17	0.02	98% 98%	Kill [runnable] thread
0.97	0.82	4.62	0.20	98% 64%	Destroy [dead] thread
2.17	1.90	2.17	0.01	98% 1%	Destroy [runnable] thread

Appendix B. Real-time characterization

6.06	5.16	10.60	0.53	59%	31% Resume [high priority] thread
1.69	1.63	5.98	0.11	90%	90% Thread switch
0.14	0.00	1.36	0.14	99%	50% Scheduler lock
0.37	0.27	0.54	0.13	62%	62% Scheduler unlock [0 threads]
0.38	0.27	0.54	0.13	60%	60% Scheduler unlock [1 suspended]
0.37	0.27	0.54	0.13	63%	63% Scheduler unlock [many suspended]
0.37	0.27	0.54	0.13	63%	63% Scheduler unlock [many low prio]
0.34	0.00	1.90	0.15	78%	6% Init mutex
0.88	0.54	4.62	0.37	93%	71% Lock [unlocked] mutex
0.79	0.54	4.35	0.26	93%	53% Unlock [locked] mutex
0.59	0.27	2.17	0.10	93%	3% Trylock [unlocked] mutex
0.50	0.27	0.82	0.09	78%	18% Trylock [locked] mutex
0.18	0.00	0.54	0.13	59%	37% Destroy mutex
3.85	3.80	5.16	0.08	96%	96% Unlock/Lock mutex
0.64	0.27	3.53	0.24	81%	15% Create mbox
0.61	0.27	2.17	0.21	68%	18% Peek [empty] mbox
0.87	0.54	5.16	0.31	59%	87% Put [first] mbox
0.08	0.00	0.54	0.12	71%	71% Peek [1 msg] mbox
0.71	0.54	1.09	0.14	56%	40% Put [second] mbox
0.08	0.00	0.27	0.12	68%	68% Peek [2 msgs] mbox
0.89	0.54	4.89	0.31	62%	81% Get [first] mbox
0.76	0.54	1.09	0.17	43%	37% Get [second] mbox
0.76	0.54	3.26	0.21	96%	50% Tryput [first] mbox
0.65	0.54	2.45	0.17	81%	81% Peek item [non-empty] mbox
0.76	0.54	2.72	0.19	53%	43% Tryget [non-empty] mbox
0.58	0.54	0.82	0.06	87%	87% Peek item [empty] mbox
0.61	0.54	0.82	0.10	75%	75% Tryget [empty] mbox
0.10	0.00	0.54	0.13	65%	65% Waiting to get mbox
0.10	0.00	0.54	0.13	65%	65% Waiting to put mbox
0.77	0.54	3.26	0.20	53%	43% Delete mbox
2.10	1.90	6.25	0.30	93%	93% Put/Get mbox
0.34	0.27	1.09	0.11	81%	81% Init semaphore
0.60	0.27	1.09	0.12	68%	6% Post [0] semaphore
0.59	0.54	0.82	0.08	81%	81% Wait [1] semaphore
0.59	0.54	2.17	0.10	96%	96% Trywait [0] semaphore
0.48	0.27	0.82	0.11	71%	25% Trywait [1] semaphore
0.24	0.00	0.82	0.09	78%	18% Peek semaphore
0.19	0.00	0.54	0.13	62%	34% Destroy semaphore
2.28	2.17	4.08	0.18	93%	90% Post/Wait semaphore
0.43	0.00	2.72	0.23	90%	6% Create counter
0.40	0.00	1.63	0.25	68%	28% Get counter value
0.13	0.00	0.82	0.15	96%	59% Set counter value
0.71	0.54	1.63	0.16	50%	46% Tick counter
0.16	0.00	0.54	0.14	53%	43% Delete counter
0.47	0.27	1.36	0.15	59%	37% Create alarm
1.58	1.09	7.07	0.44	71%	68% Initialize alarm
0.12	0.00	1.09	0.16	96%	65% Disable alarm
1.01	0.82	2.45	0.17	53%	43% Enable alarm
0.21	0.00	0.27	0.09	78%	21% Delete alarm

Appendix B. Real-time characterization

```
0.78    0.54    1.90    0.12    71%  25% Tick counter [1 alarm]
3.90    3.80    4.35    0.13    68%  68% Tick counter [many alarms]
1.25    1.09    1.63    0.14    53%  43% Tick & fire counter [1 alarm]
19.88   19.84   20.11   0.07    84%  84% Tick & fire counters [>1 together]
4.37    4.35    4.62    0.05    90%  90% Tick & fire counters [>1 separately]
3.83    3.80    7.61    0.06    99%  99% Alarm latency [0 threads]
4.46    3.80    7.88    0.27    71%  24% Alarm latency [2 threads]
16.06   13.59   26.36    1.05    54%  10% Alarm latency [many threads]
6.67    6.52   22.83    0.29    98%  98% Alarm -> thread resume latency

1.89    0.82    9.78    0.00                    Clock/interrupt latency

2.17    1.09    7.34    0.00                    Clock DSR latency

11      0      316 (main stack: 744) Thread stack used (1120 total)
All done, main stack          : stack used 744 size 2400
All done                      : Interrupt stack used 288 size 4096
All done                      : Idlethread stack used 268 size 2048

Timing complete - 30210 ms total

PASS:<Basic timing OK>
EXIT:<done>
```

Board: Cirrus Logic EDB7111-2 Development Board

CPU : Cirrus Logic EP7211 73MHz

Board: Cirrus Logic EDB7111-2 Development Board

CPU : Cirrus Logic EP7211 73MHz

```
Startup, main stack          : stack used 404 size 2400
Startup                      : Interrupt stack used 136 size 4096
Startup                      : Idlethread stack used 88 size 2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 0 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 356.69 microseconds (182 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
```

```

Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32

```

				Confidence		
Ave	Min	Max	Var	Ave	Min	Function
=====	=====	=====	=====	=====	=====	=====
22.71	17.58	37.11	3.07	46%	34%	Create thread
4.36	3.91	5.86	0.70	76%	76%	Yield thread [all suspended]
4.24	3.91	7.81	0.56	84%	84%	Suspend [suspended] thread
4.09	1.95	7.81	0.45	85%	3%	Resume thread
5.31	3.91	11.72	0.92	65%	32%	Set priority
2.11	1.95	3.91	0.28	92%	92%	Get priority
11.54	9.77	25.39	0.99	62%	28%	Kill [suspended] thread
4.46	3.91	9.77	0.82	75%	75%	Yield [no other] thread
7.57	5.86	13.67	0.69	75%	20%	Resume [suspended low prio] thread
3.94	1.95	5.86	0.18	92%	3%	Resume [runnable low prio] thread
7.02	5.86	13.67	1.05	53%	45%	Suspend [runnable] thread
4.42	3.91	9.77	0.79	76%	76%	Yield [only low prio] thread
4.24	1.95	5.86	0.61	79%	1%	Suspend [runnable->not runnable]
11.29	9.77	27.34	1.14	57%	37%	Kill [runnable] thread
6.29	3.91	11.72	0.84	71%	4%	Destroy [dead] thread
13.52	11.72	31.25	0.90	70%	25%	Destroy [runnable] thread
24.50	21.48	42.97	1.69	79%	12%	Resume [high priority] thread
8.79	7.81	19.53	1.05	99%	53%	Thread switch
1.66	0.00	3.91	0.52	83%	15%	Scheduler lock
2.59	1.95	3.91	0.86	67%	67%	Scheduler unlock [0 threads]
2.62	1.95	3.91	0.88	65%	65%	Scheduler unlock [1 suspended]
2.61	1.95	3.91	0.87	66%	66%	Scheduler unlock [many suspended]
2.58	1.95	3.91	0.85	67%	67%	Scheduler unlock [many low prio]
2.69	1.95	5.86	0.96	65%	65%	Init mutex
4.88	3.91	9.77	1.10	96%	56%	Lock [unlocked] mutex
4.64	3.91	11.72	1.05	71%	71%	Unlock [locked] mutex
3.97	1.95	7.81	0.47	81%	9%	Trylock [unlocked] mutex
3.48	1.95	3.91	0.67	78%	21%	Trylock [locked] mutex
1.77	0.00	3.91	0.44	84%	12%	Destroy mutex
31.92	29.30	42.97	1.65	71%	18%	Unlock/Lock mutex
4.09	3.91	9.77	0.35	96%	96%	Create mbox
1.83	0.00	3.91	0.34	87%	9%	Peek [empty] mbox
5.31	3.91	9.77	0.96	62%	34%	Put [first] mbox
1.59	0.00	1.95	0.60	81%	18%	Peek [1 msg] mbox
5.19	3.91	9.77	1.04	56%	40%	Put [second] mbox
1.65	0.00	3.91	0.62	78%	18%	Peek [2 msgs] mbox
5.43	3.91	9.77	0.86	68%	28%	Get [first] mbox
5.31	3.91	7.81	0.96	59%	34%	Get [second] mbox
4.76	3.91	9.77	1.07	62%	62%	Tryput [first] mbox
4.82	1.95	9.77	1.15	93%	3%	Peek item [non-empty] mbox
5.55	3.91	11.72	0.82	71%	25%	Tryget [non-empty] mbox
3.97	1.95	7.81	0.59	75%	12%	Peek item [empty] mbox

Appendix B. Real-time characterization

4.33	3.91	7.81	0.69	81%	81% Tryget [empty] mbox
1.59	0.00	3.91	0.79	68%	25% Waiting to get mbox
1.71	0.00	3.91	0.53	81%	15% Waiting to put mbox
5.25	3.91	9.77	1.01	59%	37% Delete mbox
17.82	15.63	29.30	1.14	65%	18% Put/Get mbox
2.69	1.95	5.86	0.96	65%	65% Init semaphore
3.78	1.95	7.81	0.46	84%	12% Post [0] semaphore
4.27	3.91	7.81	0.62	84%	84% Wait [1] semaphore
3.72	1.95	7.81	0.66	75%	18% Trywait [0] semaphore
3.29	1.95	5.86	0.92	62%	34% Trywait [1] semaphore
2.32	1.95	3.91	0.59	81%	81% Peek semaphore
1.89	0.00	3.91	0.24	90%	6% Destroy semaphore
15.75	13.67	29.30	1.07	68%	21% Post/Wait semaphore
2.69	1.95	5.86	0.96	65%	65% Create counter
1.83	0.00	1.95	0.23	93%	6% Get counter value
1.53	0.00	3.91	0.76	71%	25% Set counter value
4.82	3.91	5.86	0.97	53%	53% Tick counter
1.89	0.00	1.95	0.12	96%	3% Delete counter
3.78	1.95	7.81	0.46	84%	12% Create alarm
7.99	5.86	15.63	0.70	81%	9% Initialize alarm
1.71	0.00	1.95	0.43	87%	12% Disable alarm
7.14	5.86	11.72	1.04	56%	40% Enable alarm
2.50	1.95	3.91	0.79	71%	71% Delete alarm
4.94	3.91	7.81	1.04	96%	50% Tick counter [1 alarm]
19.47	17.58	23.44	0.36	87%	9% Tick counter [many alarms]
7.63	5.86	11.72	0.55	81%	15% Tick & fire counter [1 alarm]
99.06	97.66	105.47	1.05	59%	37% Tick & fire counters [>1 together]
22.15	21.48	27.34	0.96	71%	71% Tick & fire counters [>1 separately]
359.16	357.42	378.91	0.87	71%	25% Alarm latency [0 threads]
364.03	357.42	402.34	3.03	58%	15% Alarm latency [2 threads]
408.25	402.34	416.02	2.89	53%	24% Alarm latency [many threads]
381.16	376.95	492.19	2.48	95%	46% Alarm -> thread resume latency
9.79	5.86	19.53	0.00		Clock/interrupt latency
12.13	5.86	31.25	0.00		Clock DSR latency
12	0	316	(main stack: 752)	Thread stack used (1120 total)	
All done, main stack			:	stack used	752 size 2400
All done			:	Interrupt stack used	288 size 4096
All done			:	Idlethread stack used	276 size 2048
Timing complete - 30450 ms total					
PASS:<Basic timing OK>					
EXIT:<done>					

CPU : Cirrus Logic EP7212 73MHz

Board: Cirrus Logic EDB7111-2 Development Board

CPU : Cirrus Logic EP7212 73MHz

```
Startup, main stack      : stack used  404 size 2400
Startup                  : Interrupt stack used 136 size 4096
Startup                  : Idlethread stack used  88 size 2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 0 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 356.32 microseconds (182 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

				Confidence		
Ave	Min	Max	Var	Ave	Min	Function
=====	=====	=====	=====	=====	=====	=====
22.43	15.63	33.20	3.02	68%	18%	Create thread
4.48	3.91	5.86	0.81	70%	70%	Yield thread [all suspended]
4.42	3.91	7.81	0.78	75%	75%	Suspend [suspended] thread
4.12	1.95	5.86	0.49	82%	3%	Resume thread
5.62	3.91	11.72	0.64	78%	18%	Set priority
2.17	1.95	3.91	0.38	89%	89%	Get priority
11.54	9.77	27.34	0.88	70%	25%	Kill [suspended] thread
4.64	3.91	9.77	0.96	65%	65%	Yield [no other] thread
7.51	5.86	15.63	0.72	76%	21%	Resume [suspended low prio] thread
3.88	1.95	9.77	0.42	82%	10%	Resume [runnable low prio] thread
7.14	5.86	13.67	1.00	59%	39%	Suspend [runnable] thread
4.52	3.91	7.81	0.86	70%	70%	Yield [only low prio] thread
4.15	1.95	7.81	0.49	85%	1%	Suspend [runnable->not runnable]
11.26	9.77	27.34	1.17	56%	39%	Kill [runnable] thread
6.22	3.91	13.67	0.88	70%	7%	Destroy [dead] thread
13.64	11.72	33.20	1.02	64%	26%	Destroy [runnable] thread
24.17	21.48	41.02	1.49	82%	12%	Resume [high priority] thread
8.80	7.81	21.48	1.08	98%	54%	Thread switch
1.60	0.00	1.95	0.58	82%	17%	Scheduler lock
2.61	1.95	3.91	0.87	66%	66%	Scheduler unlock [0 threads]

Appendix B. Real-time characterization

2.59	1.95	3.91	0.86	67%	67% Scheduler unlock [1 suspended]
2.61	1.95	3.91	0.87	66%	66% Scheduler unlock [many suspended]
2.59	1.95	3.91	0.86	67%	67% Scheduler unlock [many low prio]
2.62	1.95	3.91	0.88	65%	65% Init mutex
4.82	3.91	9.77	1.09	96%	59% Lock [unlocked] mutex
4.39	3.91	9.77	0.79	81%	81% Unlock [locked] mutex
3.84	1.95	7.81	0.36	87%	9% Trylock [unlocked] mutex
3.54	1.95	5.86	0.69	75%	21% Trylock [locked] mutex
1.83	0.00	3.91	0.34	87%	9% Destroy mutex
34.61	31.25	46.88	1.68	78%	9% Unlock/Lock mutex
3.97	1.95	7.81	0.24	93%	3% Create mbox
1.83	0.00	3.91	0.34	87%	9% Peek [empty] mbox
4.76	3.91	9.77	1.07	62%	62% Put [first] mbox
1.71	0.00	3.91	0.64	75%	18% Peek [1 msg] mbox
5.00	3.91	9.77	1.10	96%	50% Put [second] mbox
1.65	0.00	1.95	0.52	84%	15% Peek [2 msgs] mbox
5.31	3.91	11.72	1.05	59%	37% Get [first] mbox
5.13	3.91	7.81	0.99	56%	40% Get [second] mbox
4.76	3.91	11.72	1.12	96%	65% Tryput [first] mbox
4.46	3.91	7.81	0.82	75%	75% Peek item [non-empty] mbox
5.55	3.91	9.77	0.82	68%	25% Tryget [non-empty] mbox
4.03	1.95	7.81	0.58	78%	9% Peek item [empty] mbox
4.27	3.91	5.86	0.59	81%	81% Tryget [empty] mbox
1.77	0.00	3.91	0.44	84%	12% Waiting to get mbox
1.59	0.00	1.95	0.60	81%	18% Waiting to put mbox
5.37	3.91	9.77	0.91	65%	31% Delete mbox
16.66	13.67	27.34	1.42	90%	3% Put/Get mbox
2.62	1.95	5.86	0.92	68%	68% Init semaphore
3.84	1.95	7.81	0.47	81%	12% Post [0] semaphore
4.21	3.91	7.81	0.53	87%	87% Wait [1] semaphore
3.48	1.95	5.86	0.76	71%	25% Trywait [0] semaphore
3.60	1.95	5.86	0.62	78%	18% Trywait [1] semaphore
2.26	1.95	5.86	0.53	87%	87% Peek semaphore
1.89	0.00	1.95	0.12	96%	3% Destroy semaphore
16.05	13.67	29.30	1.40	59%	18% Post/Wait semaphore
2.38	1.95	3.91	0.67	78%	78% Create counter
2.01	0.00	3.91	0.35	84%	6% Get counter value
1.89	0.00	3.91	0.24	90%	6% Set counter value
4.58	3.91	5.86	0.88	65%	65% Tick counter
1.71	0.00	1.95	0.43	87%	12% Delete counter
3.84	1.95	7.81	0.36	87%	9% Create alarm
7.99	5.86	15.63	0.47	93%	3% Initialize alarm
2.01	0.00	3.91	0.35	84%	6% Disable alarm
6.53	5.86	13.67	1.01	75%	75% Enable alarm
2.32	1.95	3.91	0.59	81%	81% Delete alarm
4.76	3.91	7.81	1.01	59%	59% Tick counter [1 alarm]
19.53	17.58	23.44	0.24	90%	6% Tick counter [many alarms]
7.57	5.86	13.67	0.75	75%	21% Tick & fire counter [1 alarm]
98.57	97.66	105.47	1.14	96%	62% Tick & fire counters [>1 together]
22.15	21.48	27.34	0.96	71%	71% Tick & fire counters [>1 separately]

```

359.18  357.42  384.77    1.10   65%  31% Alarm latency [0 threads]
362.63  357.42  396.48    2.55   43%  27% Alarm latency [2 threads]
408.22  402.34  416.02    2.73   55%  21% Alarm latency [many threads]
378.63  375.00  494.14    2.56   93%  71% Alarm -> thread resume latency

    9.78    5.86   19.53    0.00                Clock/interrupt latency

    12.21   5.86   31.25    0.00                Clock DSR latency

    12      0     316 (main stack: 752) Thread stack used (1120 total)
All done, main stack      : stack used 752 size 2400
All done      : Interrupt stack used 288 size 4096
All done      : Idlethread stack used 276 size 2048

Timing complete - 30550 ms total

PASS:<Basic timing OK>
EXIT:<done>

```

Board: ARM PID Evaluation Board

CPU : ARM 7TDMI 20 MHz

Board: ARM PID Evaluation Board

CPU : ARM 7TDMI 20 MHz

```

Startup, main stack      : stack used 404 size 2400
Startup      : Interrupt stack used 136 size 4096
Startup      : Idlethread stack used 84 size 2048

```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 6 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 120.74 microseconds (150 raw clock ticks)

Testing parameters:

```

Clock samples:      32
Threads:           50
Thread switches:    128
Mutexes:           32
Mailboxes:         32
Semaphores:        32
Scheduler operations: 128

```

Appendix B. Real-time characterization

Counters: 32
Alarms: 32

				Confidence		
Ave	Min	Max	Var	Ave	Min	Function
=====	=====	=====	=====	=====	=====	=====
99.01	68.00	129.60	15.62	50%	26%	Create thread
21.60	21.60	21.60	0.00	100%	100%	Yield thread [all suspended]
15.65	15.20	16.00	0.39	56%	44%	Suspend [suspended] thread
15.79	15.20	16.00	0.31	74%	26%	Resume thread
23.65	23.20	24.00	0.39	56%	44%	Set priority
2.26	1.60	2.40	0.24	82%	18%	Get priority
51.39	51.20	52.00	0.29	76%	76%	Kill [suspended] thread
21.60	21.60	21.60	0.00	100%	100%	Yield [no other] thread
29.47	28.00	29.60	0.22	86%	2%	Resume [suspended low prio] thread
15.60	15.20	16.00	0.40	100%	50%	Resume [runnable low prio] thread
27.73	24.00	28.00	0.40	74%	2%	Suspend [runnable] thread
21.60	21.60	21.60	0.00	100%	100%	Yield [only low prio] thread
15.65	15.20	16.00	0.39	56%	44%	Suspend [runnable->not runnable]
51.39	51.20	52.00	0.29	76%	76%	Kill [runnable] thread
27.66	27.20	28.80	0.41	54%	44%	Destroy [dead] thread
68.93	64.80	69.60	0.35	72%	2%	Destroy [runnable] thread
91.26	90.40	107.20	0.64	66%	32%	Resume [high priority] thread
49.14	48.80	49.60	0.39	57%	57%	Thread switch
2.20	1.60	2.40	0.30	75%	25%	Scheduler lock
10.20	9.60	10.40	0.30	75%	25%	Scheduler unlock [0 threads]
10.20	9.60	10.40	0.30	75%	25%	Scheduler unlock [1 suspended]
10.20	9.60	10.40	0.30	75%	25%	Scheduler unlock [many suspended]
10.20	9.60	10.40	0.30	75%	25%	Scheduler unlock [many low prio]
6.85	6.40	7.20	0.39	56%	43%	Init mutex
18.40	18.40	18.40	0.00	100%	100%	Lock [unlocked] mutex
19.57	19.20	20.00	0.40	53%	53%	Unlock [locked] mutex
16.55	16.00	16.80	0.34	68%	31%	Trylock [unlocked] mutex
14.55	14.40	15.20	0.24	81%	81%	Trylock [locked] mutex
3.55	3.20	4.00	0.39	56%	56%	Destroy mutex
119.85	119.20	120.00	0.24	81%	18%	Unlock/Lock mutex
12.85	12.80	13.60	0.09	93%	93%	Create mbox
1.65	1.60	2.40	0.09	93%	93%	Peek [empty] mbox
20.70	20.00	20.80	0.17	87%	12%	Put [first] mbox
1.65	1.60	2.40	0.09	93%	93%	Peek [1 msg] mbox
20.70	20.00	20.80	0.17	87%	12%	Put [second] mbox
1.65	1.60	2.40	0.09	93%	93%	Peek [2 msgs] mbox
20.85	20.80	21.60	0.09	93%	93%	Get [first] mbox
20.85	20.80	21.60	0.09	93%	93%	Get [second] mbox
19.90	19.20	20.00	0.17	87%	12%	Tryput [first] mbox
17.60	17.60	17.60	0.00	100%	100%	Peek item [non-empty] mbox
20.90	20.80	21.60	0.17	87%	87%	Tryget [non-empty] mbox
16.80	16.80	16.80	0.00	100%	100%	Peek item [empty] mbox
17.65	17.60	18.40	0.09	93%	93%	Tryget [empty] mbox
1.85	1.60	2.40	0.34	68%	68%	Waiting to get mbox
1.85	1.60	2.40	0.34	68%	68%	Waiting to put mbox

Appendix B. Real-time characterization

19.40	19.20	20.00	0.30	75%	75%	Delete mbox
65.05	64.80	65.60	0.34	68%	68%	Put/Get mbox
7.05	6.40	7.20	0.24	81%	18%	Init semaphore
15.55	15.20	16.00	0.39	56%	56%	Post [0] semaphore
17.35	16.80	17.60	0.34	68%	31%	Wait [1] semaphore
14.60	14.40	15.20	0.30	75%	75%	Trywait [0] semaphore
14.20	13.60	14.40	0.30	75%	25%	Trywait [1] semaphore
4.55	4.00	4.80	0.34	68%	31%	Peek semaphore
3.75	3.20	4.00	0.34	68%	31%	Destroy semaphore
70.85	70.40	71.20	0.39	56%	43%	Post/Wait semaphore
6.05	5.60	6.40	0.39	56%	43%	Create counter
2.25	1.60	2.40	0.24	81%	18%	Get counter value
2.25	1.60	2.40	0.24	81%	18%	Set counter value
19.70	19.20	20.00	0.37	62%	37%	Tick counter
3.45	3.20	4.00	0.34	68%	68%	Delete counter
9.05	8.80	9.60	0.34	68%	68%	Create alarm
29.60	29.60	29.60	0.00	100%	100%	Initialize alarm
2.15	1.60	2.40	0.34	68%	31%	Disable alarm
29.35	28.80	29.60	0.34	68%	31%	Enable alarm
5.10	4.80	5.60	0.37	62%	62%	Delete alarm
23.20	23.20	23.20	0.00	100%	100%	Tick counter [1 alarm]
138.00	137.60	138.40	0.40	100%	50%	Tick counter [many alarms]
40.40	40.00	40.80	0.40	100%	50%	Tick & fire counter [1 alarm]
704.25	697.60	804.00	12.47	93%	93%	Tick & fire counters [>1 together]
155.20	155.20	155.20	0.00	100%	100%	Tick & fire counters [>1 separately]
105.20	104.80	151.20	0.76	99%	94%	Alarm latency [0 threads]
117.57	104.80	149.60	7.13	57%	25%	Alarm latency [2 threads]
117.49	104.80	148.80	7.10	58%	26%	Alarm latency [many threads]
192.59	177.60	316.00	1.93	98%	0%	Alarm -> thread resume latency
22.10	21.60	24.00	0.00			Clock/interrupt latency
38.69	32.80	61.60	0.00			Clock DSR latency
297	276	316	(main stack: 752)			Thread stack used (1120 total)
All done, main stack			: stack used	752	size	2400
All done			: Interrupt stack used	288	size	4096
All done			: Idlethread stack used	272	size	2048

Timing complete - 30350 ms total

PASS:<Basic timing OK>

EXIT:<done>

CPU : ARM 920T 20 MHz

Board: ARM PID Evaluation Board

CPU : ARM 920T 20 MHz

```
Startup, main stack          : stack used  404 size 2400
Startup                     : Interrupt stack used 136 size 4096
Startup                     : Idlethread stack used  84 size 2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 15 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 291.41 microseconds (364 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            50
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

				Confidence		
Ave	Min	Max	Var	Ave	Min	Function
=====	=====	=====	=====	=====	=====	=====
257.78	168.00	568.00	48.70	56%	28%	Create thread
50.21	49.60	50.40	0.29	76%	24%	Yield thread [all suspended]
36.26	36.00	36.80	0.35	68%	68%	Suspend [suspended] thread
37.20	36.80	37.60	0.40	100%	50%	Resume thread
56.24	56.00	56.80	0.34	70%	70%	Set priority
5.20	4.80	5.60	0.40	100%	50%	Get priority
122.75	122.40	123.20	0.39	56%	56%	Kill [suspended] thread
50.19	49.60	50.40	0.31	74%	26%	Yield [no other] thread
69.49	66.40	69.60	0.21	92%	2%	Resume [suspended low prio] thread
37.01	36.80	37.60	0.31	74%	74%	Resume [runnable low prio] thread
64.75	55.20	65.60	0.38	80%	2%	Suspend [runnable] thread
50.19	49.60	50.40	0.31	74%	26%	Yield [only low prio] thread
36.24	36.00	36.80	0.34	70%	70%	Suspend [runnable->not runnable]
122.75	122.40	123.20	0.39	56%	56%	Kill [runnable] thread
67.76	67.20	68.00	0.34	70%	30%	Destroy [dead] thread
167.07	158.40	168.00	0.35	92%	2%	Destroy [runnable] thread
213.49	212.00	249.60	1.46	84%	90%	Resume [high priority] thread
122.81	120.00	389.60	4.17	99%	99%	Thread switch
4.70	4.00	4.80	0.17	87%	12%	Scheduler lock
23.70	23.20	24.00	0.37	62%	37%	Scheduler unlock [0 threads]
23.60	23.20	24.00	0.40	100%	50%	Scheduler unlock [1 suspended]

Appendix B. Real-time characterization

23.70	23.20	24.00	0.37	62%	37%	Scheduler unlock [many suspended]
23.60	23.20	24.00	0.40	100%	50%	Scheduler unlock [many low prio]
15.65	15.20	16.00	0.39	56%	43%	Init mutex
42.40	42.40	42.40	0.00	100%	100%	Lock [unlocked] mutex
45.37	44.80	46.40	0.36	65%	31%	Unlock [locked] mutex
39.20	39.20	39.20	0.00	100%	100%	Trylock [unlocked] mutex
34.45	34.40	35.20	0.09	93%	93%	Trylock [locked] mutex
8.00	8.00	8.00	0.00	100%	100%	Destroy mutex
284.42	284.00	284.80	0.40	53%	46%	Unlock/Lock mutex
29.40	28.80	29.60	0.30	75%	25%	Create mbox
3.35	3.20	4.00	0.24	81%	81%	Peek [empty] mbox
49.35	48.80	49.60	0.34	68%	31%	Put [first] mbox
3.35	3.20	4.00	0.24	81%	81%	Peek [1 msg] mbox
49.35	48.80	49.60	0.34	68%	31%	Put [second] mbox
3.35	3.20	4.00	0.24	81%	81%	Peek [2 msgs] mbox
49.15	48.80	49.60	0.39	56%	56%	Get [first] mbox
49.15	48.80	49.60	0.39	56%	56%	Get [second] mbox
47.80	47.20	48.00	0.30	75%	25%	Tryput [first] mbox
41.40	40.80	41.60	0.30	75%	25%	Peek item [non-empty] mbox
49.40	48.80	49.60	0.30	75%	25%	Tryget [non-empty] mbox
40.15	40.00	40.80	0.24	81%	81%	Peek item [empty] mbox
40.95	40.80	41.60	0.24	81%	81%	Tryget [empty] mbox
4.05	4.00	4.80	0.09	93%	93%	Waiting to get mbox
4.05	4.00	4.80	0.09	93%	93%	Waiting to put mbox
45.60	45.60	45.60	0.00	100%	100%	Delete mbox
153.27	152.80	153.60	0.39	59%	40%	Put/Get mbox
16.80	16.80	16.80	0.00	100%	100%	Init semaphore
36.60	36.00	36.80	0.30	75%	25%	Post [0] semaphore
39.60	39.20	40.00	0.40	100%	50%	Wait [1] semaphore
34.80	34.40	35.20	0.40	100%	50%	Trywait [0] semaphore
33.35	32.80	33.60	0.34	68%	31%	Trywait [1] semaphore
10.30	9.60	10.40	0.17	87%	12%	Peek semaphore
8.80	8.80	8.80	0.00	100%	100%	Destroy semaphore
166.92	166.40	167.20	0.36	65%	34%	Post/Wait semaphore
13.60	13.60	13.60	0.00	100%	100%	Create counter
4.85	4.80	5.60	0.09	93%	93%	Get counter value
4.80	4.80	4.80	0.00	100%	100%	Set counter value
45.25	44.80	45.60	0.39	56%	43%	Tick counter
7.75	7.20	8.00	0.34	68%	31%	Delete counter
20.80	20.80	20.80	0.00	100%	100%	Create alarm
69.30	68.80	69.60	0.37	62%	37%	Initialize alarm
4.80	4.80	4.80	0.00	100%	100%	Disable alarm
67.35	67.20	68.00	0.24	81%	81%	Enable alarm
11.80	11.20	12.00	0.30	75%	25%	Delete alarm
54.80	54.40	55.20	0.40	100%	50%	Tick counter [1 alarm]
372.35	363.20	652.80	17.53	96%	96%	Tick counter [many alarms]
95.50	95.20	96.00	0.37	62%	62%	Tick & fire counter [1 alarm]
1757.92	1707.20	1996.80	81.43	81%	81%	Tick & fire counters [>1 together]
404.37	404.00	404.80	0.40	53%	53%	Tick & fire counters [>1 separately]
256.57	254.40	395.20	2.17	98%	97%	Alarm latency [0 threads]

Appendix B. Real-time characterization

```
296.60 255.20 359.20 23.53 53% 31% Alarm latency [2 threads]
307.49 265.60 357.60 27.52 53% 53% Alarm latency [many threads]
467.04 432.00 788.80 5.03 97% 1% Alarm -> thread resume latency

55.63 54.40 60.80 0.00 Clock/interrupt latency

101.23 80.80 1433.60 0.00 Clock DSR latency

316 316 316 (main stack: 752) Thread stack used (1120 total)
All done, main stack : stack used 752 size 2400
All done : Interrupt stack used 288 size 4096
All done : Idlethread stack used 272 size 2048

Timing complete - 30780 ms total

PASS:<Basic timing OK>
EXIT:<done>
```

Board: Intel IQ80310 XScale Development Kit

Board: Intel IQ80310 XScale Development Kit

CPU: Intel XScale 600MHz

```
Startup, main stack : stack used 388 size 2400
Startup : Interrupt stack used 148 size 4096
Startup : Idlethread stack used 76 size 1120
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 73 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 12.11 microseconds (399 raw clock ticks)

Testing parameters:

```
Clock samples: 32
Threads: 64
Thread switches: 128
Mutexes: 32
Mailboxes: 32
Semaphores: 32
Scheduler operations: 128
Counters: 32
Alarms: 32
```

Confidence

```
Ave Min Max Var Ave Min Function
```

=====	=====	=====	=====	=====	=====
6.53	5.48	8.55	0.50	53%	23% Create thread
0.37	0.03	3.24	0.18	87%	1% Yield thread [all suspended]
0.24	0.00	2.06	0.12	87%	1% Suspend [suspended] thread
0.25	0.00	0.73	0.06	71%	1% Resume thread
0.36	0.09	0.82	0.10	89%	1% Set priority
0.03	0.00	0.42	0.05	90%	90% Get priority
1.07	0.52	6.39	0.18	92%	1% Kill [suspended] thread
0.33	0.06	0.91	0.08	78%	3% Yield [no other] thread
0.55	0.03	1.06	0.09	85%	1% Resume [suspended low prio] thread
0.28	0.00	1.79	0.11	84%	4% Resume [runnable low prio] thread
0.43	0.00	1.00	0.12	76%	1% Suspend [runnable] thread
0.31	0.00	1.24	0.09	82%	4% Yield [only low prio] thread
0.21	0.00	0.42	0.04	73%	1% Suspend [runnable->not runnable]
1.00	0.88	1.45	0.04	78%	4% Kill [runnable] thread
0.59	0.42	3.97	0.13	81%	87% Destroy [dead] thread
1.43	1.27	1.94	0.07	78%	7% Destroy [runnable] thread
3.12	2.58	5.09	0.33	56%	34% Resume [high priority] thread
0.87	0.36	1.39	0.07	86%	0% Thread switch
0.15	0.00	1.39	0.21	81%	81% Scheduler lock
0.16	0.00	0.64	0.08	85%	7% Scheduler unlock [0 threads]
0.16	0.00	0.64	0.08	75%	8% Scheduler unlock [1 suspended]
0.16	0.00	0.70	0.08	78%	6% Scheduler unlock [many suspended]
0.16	0.00	0.64	0.07	81%	4% Scheduler unlock [many low prio]
0.45	0.00	1.39	0.34	56%	46% Init mutex
0.43	0.18	3.27	0.23	87%	87% Lock [unlocked] mutex
0.48	0.09	3.88	0.26	84%	71% Unlock [locked] mutex
0.35	0.21	2.24	0.21	87%	84% Trylock [unlocked] mutex
0.26	0.00	0.67	0.13	78%	9% Trylock [locked] mutex
0.21	0.00	1.27	0.24	78%	75% Destroy mutex
2.58	2.09	3.09	0.13	75%	9% Unlock/Lock mutex
0.99	0.21	2.48	0.41	65%	28% Create mbox
0.04	0.00	0.39	0.07	90%	87% Peek [empty] mbox
0.47	0.27	3.48	0.29	90%	78% Put [first] mbox
0.02	0.00	0.39	0.03	90%	90% Peek [1 msg] mbox
0.29	0.15	0.58	0.04	68%	3% Put [second] mbox
0.02	0.00	0.45	0.04	93%	93% Peek [2 msgs] mbox
0.48	0.21	3.67	0.26	84%	87% Get [first] mbox
0.35	0.09	0.82	0.11	75%	3% Get [second] mbox
0.50	0.21	3.18	0.33	90%	68% Tryput [first] mbox
0.39	0.15	1.39	0.19	78%	68% Peek item [non-empty] mbox
0.43	0.18	3.33	0.23	87%	90% Tryget [non-empty] mbox
0.28	0.03	0.79	0.06	68%	3% Peek item [empty] mbox
0.28	0.21	0.58	0.05	71%	65% Tryget [empty] mbox
0.01	0.00	0.36	0.02	96%	90% Waiting to get mbox
0.05	0.00	0.45	0.09	87%	84% Waiting to put mbox
0.42	0.09	2.88	0.20	84%	12% Delete mbox
1.39	1.27	2.39	0.14	87%	87% Put/Get mbox
0.35	0.00	1.36	0.45	75%	68% Init semaphore
0.19	0.00	0.45	0.04	81%	3% Post [0] semaphore
0.25	0.21	0.88	0.06	84%	81% Wait [1] semaphore

Appendix B. Real-time characterization

0.32	0.06	1.79	0.21	78%	68% Trywait [0] semaphore
0.20	0.00	0.52	0.06	62%	3% Trywait [1] semaphore
0.07	0.00	0.45	0.10	84%	81% Peek semaphore
0.06	0.00	0.52	0.06	71%	78% Destroy semaphore
1.45	1.42	1.79	0.04	87%	87% Post/Wait semaphore
0.70	0.00	2.88	0.47	43%	34% Create counter
0.05	0.00	0.42	0.09	87%	84% Get counter value
0.02	0.00	0.45	0.04	93%	93% Set counter value
0.38	0.12	0.58	0.06	59%	3% Tick counter
0.03	0.00	0.48	0.05	93%	78% Delete counter
1.10	0.39	4.30	0.47	62%	53% Create alarm
0.58	0.03	3.12	0.18	87%	3% Initialize alarm
0.04	0.00	0.42	0.07	90%	90% Disable alarm
0.54	0.36	1.36	0.12	84%	43% Enable alarm
0.03	0.00	0.70	0.06	84%	84% Delete alarm
0.50	0.24	0.97	0.08	84%	6% Tick counter [1 alarm]
5.30	5.12	5.97	0.14	84%	75% Tick counter [many alarms]
0.82	0.64	1.36	0.11	78%	43% Tick & fire counter [1 alarm]
14.13	13.85	14.55	0.09	78%	3% Tick & fire counters [>1 together]
5.56	5.45	6.00	0.09	78%	71% Tick & fire counters [>1 separately]
9.69	9.45	12.52	0.22	64%	71% Alarm latency [0 threads]
9.98	9.48	12.76	0.23	69%	14% Alarm latency [2 threads]
10.38	9.48	24.67	0.59	74%	45% Alarm latency [many threads]
11.72	11.30	21.33	0.32	81%	58% Alarm -> thread resume latency
1.87	1.82	10.42	0.00		Clock/interrupt latency
3.02	2.58	7.67	0.00		Clock DSR latency
9	0	260	(main stack: 776)	Thread stack used (1120 total)	
All done, main stack : stack used 776 size 2400					
All done : Interrupt stack used 268 size 4096					
All done : Idlethread stack used 244 size 1120					
Timing complete - 30300 ms total					
PASS:<Basic timing OK>					
EXIT:<done>					

Board: Toshiba JMR3904 Evaluation Board

Board: Toshiba JMR3904 Evaluation Board

CPU : TMPR3904F 50MHz

eCOS Kernel Timings

Note: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 0 'ticks' overhead
 ... this value will be factored out of all other measurements
 Clock interrupt took 29.68 microseconds (45 raw clock ticks)

Testing parameters:

Clock samples: 32
 Threads: 24
 Thread switches: 128
 Mutexes: 32
 Mailboxes: 32
 Semaphores: 32
 Scheduler operations: 128
 Counters: 32
 Alarms: 32

				Confidence			Function
Ave	Min	Max	Var	Ave	Min		
=====	=====	=====	=====	=====	=====		
13.62	11.72	27.99	1.51	79%	54%		
2.77	2.60	3.91	0.26	79%	79%		
3.31	2.60	6.51	0.27	83%	12%		
2.58	1.95	7.81	0.47	58%	37%		
4.94	4.56	11.07	0.60	95%	79%		
0.71	0.65	1.95	0.10	95%	95%		
14.97	14.32	25.39	0.87	95%	95%		
2.25	1.95	9.11	0.57	95%	95%		
7.27	6.51	12.37	0.42	79%	16%		
2.28	1.95	7.16	0.51	95%	79%		
4.31	3.26	12.37	0.75	87%	79%		
2.17	1.95	7.16	0.42	95%	95%		
2.39	1.95	6.51	0.51	95%	58%		
13.43	12.37	22.79	0.80	91%	91%		
22.30	20.83	37.76	1.76	91%	91%		
4.62	4.56	11.07	0.13	98%	98%		
1.51	1.30	2.60	0.29	68%	68%		
2.36	1.95	3.26	0.31	61%	37%		
2.39	1.95	5.21	0.32	62%	36%		
2.38	1.95	4.56	0.32	61%	37%		
2.38	1.95	5.21	0.32	61%	37%		
0.90	0.65	3.26	0.35	71%	71%		
2.48	1.95	8.46	0.50	50%	46%		
2.83	2.60	9.11	0.42	93%	93%		
2.30	1.95	6.51	0.45	96%	65%		
1.99	1.30	5.86	0.24	84%	12%		
0.04	0.00	1.30	0.08	96%	96%		
42.40	42.32	44.92	0.16	96%	96%		
1.44	1.30	5.86	0.28	96%	96%		
0.51	0.00	1.30	0.25	71%	25%		
2.93	2.60	9.11	0.51	96%	78%		
0.51	0.00	1.30	0.25	71%	25%		

Appendix B. Real-time characterization

4.19	3.91	5.21	0.34	59%	59% Put [second] mbox
0.45	0.00	0.65	0.28	68%	31% Peek [2 msgs] mbox
3.28	2.60	10.42	0.45	65%	31% Get [first] mbox
3.34	2.60	9.77	0.40	78%	18% Get [second] mbox
2.69	1.95	9.11	0.40	78%	18% Tryput [first] mbox
2.75	1.95	7.81	0.32	93%	3% Peek item [non-empty] mbox
3.15	2.60	9.11	0.48	53%	43% Tryget [non-empty] mbox
2.22	1.95	6.51	0.41	96%	78% Peek item [empty] mbox
2.40	1.95	5.86	0.42	50%	46% Tryget [empty] mbox
0.47	0.00	0.65	0.26	71%	28% Waiting to get mbox
0.59	0.00	1.30	0.15	84%	12% Waiting to put mbox
4.01	3.26	10.42	0.40	81%	15% Delete mbox
26.18	26.04	30.60	0.28	96%	96% Put/Get mbox
0.92	0.65	3.91	0.38	71%	71% Init semaphore
2.24	1.95	6.51	0.43	96%	75% Post [0] semaphore
2.32	1.95	7.16	0.48	96%	65% Wait [1] semaphore
2.03	1.30	5.86	0.24	90%	6% Trywait [0] semaphore
1.91	1.30	4.56	0.23	78%	18% Trywait [1] semaphore
0.77	0.00	1.95	0.30	65%	9% Peek semaphore
0.61	0.00	1.95	0.15	84%	12% Destroy semaphore
22.62	22.14	30.60	0.61	96%	62% Post/Wait semaphore
0.92	0.65	3.91	0.38	71%	71% Create counter
0.69	0.65	1.95	0.08	96%	96% Get counter value
0.41	0.00	1.30	0.33	56%	40% Set counter value
3.21	2.60	5.86	0.27	71%	21% Tick counter
0.65	0.00	3.26	0.16	84%	12% Delete counter
1.57	1.30	4.56	0.38	71%	71% Create alarm
4.52	3.91	13.02	0.57	50%	46% Initialize alarm
0.61	0.00	1.95	0.15	84%	12% Disable alarm
4.43	3.91	9.11	0.43	56%	40% Enable alarm
0.87	0.65	2.60	0.32	71%	71% Delete alarm
2.93	2.60	6.51	0.43	96%	65% Tick counter [1 alarm]
14.83	14.32	22.79	0.60	96%	59% Tick counter [many alarms]
4.88	4.56	11.07	0.51	96%	78% Tick & fire counter [1 alarm]
83.25	82.03	102.86	1.23	96%	93% Tick & fire counters [>1 together]
17.58	16.93	27.34	0.61	50%	46% Tick & fire counters [>1 separately]
26.18	24.74	40.36	0.30	97%	0% Alarm latency [0 threads]
33.88	29.30	56.64	1.70	85%	6% Alarm latency [2 threads]
36.37	29.30	61.20	3.25	53%	24% Alarm latency [many threads]
7.85	6.51	14.97	0.00		Clock/interrupt latency
Timing complete - 23540 ms total					
PASS:<Basic timing OK>					
EXIT:<done>					

Board: Toshiba REF 4955

Board: Toshiba REF 4955

CPU : Toshiba TX4955 66MHz

```
Startup, main stack      : stack used   960 size  2936
Startup                  : Interrupt stack used 168 size  4096
Startup                  : Idlethread stack used 372 size  2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 3 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 4.00 microseconds (264 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

				Confidence			
Ave	Min	Max	Var	Ave	Min	Function	
=====	=====	=====	=====	=====	=====	=====	
11.21	9.58	14.11	0.95	48%	34%	Create thread	
0.66	0.65	1.29	0.02	98%	98%	Yield thread [all suspended]	
0.63	0.53	3.06	0.17	82%	82%	Suspend [suspended] thread	
0.54	0.53	1.06	0.02	98%	98%	Resume thread	
0.78	0.74	1.39	0.05	93%	93%	Set priority	
0.05	0.05	0.36	0.01	98%	98%	Get priority	
2.06	1.89	6.65	0.25	95%	79%	Kill [suspended] thread	
0.65	0.65	0.68	0.00	98%	98%	Yield [no other] thread	
1.15	1.02	3.03	0.20	81%	81%	Resume [suspended low prio] thread	
0.54	0.52	1.18	0.03	96%	96%	Resume [runnable low prio] thread	
0.94	0.88	1.27	0.01	95%	1%	Suspend [runnable] thread	
0.65	0.65	0.68	0.00	98%	98%	Yield [only low prio] thread	
0.54	0.53	0.86	0.01	98%	96%	Suspend [runnable->not runnable]	
1.97	1.89	2.98	0.12	84%	84%	Kill [runnable] thread	
1.03	0.92	4.94	0.17	89%	89%	Destroy [dead] thread	
2.55	2.33	4.38	0.24	89%	70%	Destroy [runnable] thread	
5.62	4.11	13.23	0.99	65%	40%	Resume [high priority] thread	
1.84	1.83	2.79	0.02	98%	98%	Thread switch	
0.12	0.02	0.65	0.15	74%	74%	Scheduler lock	
0.35	0.35	0.35	0.00	100%	100%	Scheduler unlock [0 threads]	
0.35	0.35	0.35	0.00	100%	100%	Scheduler unlock [1 suspended]	
0.43	0.35	1.17	0.13	78%	78%	Scheduler unlock [many suspended]	

Appendix B. Real-time characterization

0.45	0.35	1.17	0.15	75%	75%	Scheduler unlock [many low prio]
0.46	0.15	3.38	0.30	62%	50%	Init mutex
0.73	0.64	3.27	0.16	96%	96%	Lock [unlocked] mutex
0.77	0.65	4.50	0.23	96%	96%	Unlock [locked] mutex
0.58	0.55	1.42	0.05	96%	96%	Trylock [unlocked] mutex
0.51	0.50	0.83	0.02	96%	96%	Trylock [locked] mutex
0.12	0.11	0.41	0.02	96%	96%	Destroy mutex
4.72	4.70	5.58	0.05	96%	96%	Unlock/Lock mutex
1.01	0.67	3.48	0.40	71%	71%	Create mbox
0.02	0.00	0.53	0.03	96%	96%	Peek [empty] mbox
0.89	0.68	4.20	0.29	96%	71%	Put [first] mbox
0.02	0.00	0.33	0.02	96%	96%	Peek [1 msg] mbox
0.69	0.68	0.76	0.01	50%	46%	Put [second] mbox
0.02	0.00	0.30	0.02	96%	96%	Peek [2 msgs] mbox
0.81	0.71	3.83	0.19	96%	96%	Get [first] mbox
0.72	0.71	1.02	0.02	96%	96%	Get [second] mbox
0.81	0.65	2.74	0.22	96%	71%	Tryput [first] mbox
0.67	0.62	2.27	0.10	96%	96%	Peek item [non-empty] mbox
0.77	0.71	2.41	0.10	96%	96%	Tryget [non-empty] mbox
0.59	0.58	0.88	0.02	96%	96%	Peek item [empty] mbox
0.62	0.62	0.67	0.00	96%	96%	Tryget [empty] mbox
0.03	0.02	0.32	0.02	96%	96%	Waiting to get mbox
0.02	0.02	0.06	0.01	50%	46%	Waiting to put mbox
0.75	0.65	3.59	0.18	96%	96%	Delete mbox
2.80	2.77	3.59	0.05	96%	96%	Put/Get mbox
0.37	0.18	0.88	0.28	71%	71%	Init semaphore
0.48	0.47	0.80	0.02	96%	96%	Post [0] semaphore
0.60	0.59	0.67	0.01	50%	46%	Wait [1] semaphore
0.53	0.50	1.41	0.06	96%	96%	Trywait [0] semaphore
0.51	0.50	0.71	0.01	96%	50%	Trywait [1] semaphore
0.09	0.09	0.15	0.00	96%	96%	Peek semaphore
0.12	0.11	0.41	0.02	96%	96%	Destroy semaphore
3.05	3.05	3.05	0.00	100%	100%	Post/Wait semaphore
0.57	0.17	2.76	0.24	59%	25%	Create counter
0.06	0.05	0.58	0.03	96%	96%	Get counter value
0.06	0.03	0.64	0.04	96%	96%	Set counter value
0.73	0.71	1.02	0.02	96%	96%	Tick counter
0.12	0.11	0.15	0.01	50%	46%	Delete counter
0.89	0.64	3.15	0.34	84%	71%	Create alarm
1.00	0.95	2.41	0.09	96%	96%	Initialize alarm
0.09	0.06	0.68	0.04	96%	96%	Disable alarm
1.05	1.00	2.48	0.09	96%	96%	Enable alarm
0.18	0.17	0.50	0.02	96%	96%	Delete alarm
0.90	0.89	1.11	0.01	96%	96%	Tick counter [1 alarm]
5.60	5.59	5.88	0.02	96%	96%	Tick counter [many alarms]
1.53	1.52	2.11	0.04	96%	96%	Tick & fire counter [1 alarm]
25.48	25.47	25.76	0.02	96%	96%	Tick & fire counters [>1 together]
6.22	6.21	6.44	0.01	96%	96%	Tick & fire counters [>1 separately]
2.59	2.56	6.17	0.07	98%	98%	Alarm latency [0 threads]
4.06	3.95	6.24	0.08	78%	57%	Alarm latency [2 threads]

```

5.03    2.56    9.03    0.89    59% 10% Alarm latency [many threads]
5.68    5.59   15.45    0.15    99% 99% Alarm -> thread resume latency

2.52    1.41    8.12    0.00                Clock/interrupt latency

2.05    1.17    6.00    0.00                Clock DSR latency

34      0      1072 (main stack: 1320) Thread stack used (1912 total)
All done, main stack      : stack used 1320 size 2936
All done                  : Interrupt stack used 136 size 4096
All done                  : Idlethread stack used 996 size 2048

Timing complete - 30360 ms total

PASS:<Basic timing OK>
EXIT:<done>

```

Board: Matsushita STDEVAL1 Board

Board: Matsushita STDEVAL1 Board

CPU : MN103002A 60MHz

eCOS Kernel Timings

Note: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 18 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 13.73 microseconds (205 raw clock ticks)

Testing parameters:

```

Clock samples:      32
Threads:            24
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32

```

				Confidence			
Ave	Min	Max	Var	Ave	Min	Function	
=====	=====	=====	=====	=====	=====	=====	
14.36	11.53	23.53	1.81	54%	33%	Create thread	
2.64	2.53	5.07	0.20	95%	95%	Yield thread [all suspended]	
2.25	1.93	4.80	0.31	45%	83%	Suspend [suspended] thread	
2.19	2.00	4.93	0.28	91%	91%	Resume thread	
3.42	3.00	8.40	0.47	95%	87%	Set priority	

Appendix B. Real-time characterization

0.31	0.13	1.20	0.19	79%	58% Get priority
8.26	7.40	18.80	0.93	95%	87% Kill [suspended] thread
2.58	2.47	5.13	0.21	95%	95% Yield [no other] thread
5.07	4.53	8.67	0.44	62%	50% Resume [suspended low prio] thread
2.27	2.07	4.53	0.23	87%	87% Resume [runnable low prio] thread
4.76	4.07	9.40	0.65	66%	75% Suspend [runnable] thread
2.63	2.53	4.73	0.18	95%	95% Yield [only low prio] thread
2.09	1.87	4.27	0.27	91%	79% Suspend [runnable->not runnable]
10.79	10.00	18.20	0.81	95%	79% Kill [runnable] thread
20.30	18.40	28.80	1.42	79%	54% Resume [high priority] thread
5.53	5.47	12.13	0.11	98%	97% Thread switch
0.28	0.27	2.20	0.03	97%	97% Scheduler lock
1.14	1.13	2.00	0.01	99%	99% Scheduler unlock [0 threads]
1.14	1.13	2.40	0.02	99%	99% Scheduler unlock [1 suspended]
1.16	1.13	3.33	0.06	95%	95% Scheduler unlock [many suspended]
1.23	1.20	3.13	0.05	95%	95% Scheduler unlock [many low prio]
1.29	1.00	4.20	0.25	65%	50% Init mutex
2.65	2.47	5.27	0.23	93%	87% Lock [unlocked] mutex
3.26	3.07	6.80	0.28	93%	87% Unlock [locked] mutex
2.48	2.33	5.07	0.21	90%	87% Trylock [unlocked] mutex
2.20	2.07	4.67	0.21	93%	87% Trylock [locked] mutex
0.23	0.20	1.00	0.05	96%	93% Destroy mutex
25.11	24.73	27.53	0.21	65%	31% Unlock/Lock mutex
2.49	2.00	5.73	0.32	81%	37% Create mbox
0.11	0.00	1.60	0.15	84%	81% Peek [empty] mbox
3.01	2.60	9.47	0.52	96%	78% Put [first] mbox
0.10	0.00	1.67	0.15	87%	81% Peek [1 msg] mbox
3.09	2.60	8.33	0.50	93%	75% Put [second] mbox
0.06	0.00	1.13	0.08	96%	87% Peek [2 msgs] mbox
3.10	2.80	7.93	0.40	93%	84% Get [first] mbox
3.13	2.80	7.53	0.43	90%	78% Get [second] mbox
2.99	2.60	8.53	0.52	93%	75% Tryput [first] mbox
2.65	2.33	6.80	0.42	90%	78% Peek item [non-empty] mbox
3.05	2.73	7.60	0.42	93%	78% Tryget [non-empty] mbox
3.16	2.93	6.27	0.31	84%	84% Peek item [empty] mbox
2.48	2.27	5.73	0.30	84%	84% Tryget [empty] mbox
0.23	0.13	2.07	0.14	96%	87% Waiting to get mbox
0.22	0.13	1.93	0.13	96%	75% Waiting to put mbox
3.08	2.80	7.93	0.42	84%	84% Delete mbox
16.01	15.53	19.00	0.52	78%	59% Put/Get mbox
0.85	0.67	3.27	0.19	96%	50% Init semaphore
2.00	1.93	3.87	0.12	96%	90% Post [0] semaphore
2.05	2.00	3.47	0.09	96%	96% Wait [1] semaphore
1.85	1.80	3.47	0.10	96%	96% Trywait [0] semaphore
1.82	1.80	2.53	0.04	96%	96% Trywait [1] semaphore
0.36	0.33	1.33	0.06	96%	96% Peek semaphore
0.38	0.33	1.87	0.09	96%	96% Destroy semaphore
12.38	12.20	16.27	0.30	93%	87% Post/Wait semaphore
1.18	0.73	4.07	0.24	78%	18% Create counter
0.20	0.13	1.40	0.11	87%	87% Get counter value

0.24	0.20	1.40	0.08	93%	93% Set counter value
3.17	3.13	4.20	0.07	93%	93% Tick counter
0.44	0.40	1.73	0.08	96%	96% Delete counter
2.24	1.67	5.13	0.47	68%	65% Create alarm
3.86	3.40	9.67	0.51	90%	78% Initialize alarm
0.15	0.07	1.60	0.12	96%	68% Disable alarm
3.76	3.47	7.67	0.35	93%	75% Enable alarm
0.57	0.47	2.73	0.16	96%	84% Delete alarm
3.64	3.60	4.73	0.07	96%	96% Tick counter [1 alarm]
21.72	21.67	23.27	0.10	96%	96% Tick counter [many alarms]
6.13	6.07	8.07	0.12	96%	96% Tick & fire counter [1 alarm]
101.40	99.53	132.73	2.75	93%	93% Tick & fire counters [>1 together]
24.21	24.13	26.40	0.14	96%	96% Tick & fire counters [>1 separately]
11.74	11.60	22.67	0.26	98%	98% Alarm latency [0 threads]
14.58	11.73	24.93	1.59	54%	28% Alarm latency [2 threads]
18.18	15.20	41.07	1.96	60%	43% Alarm latency [many threads]
3.06	2.13	10.33	0.00		Clock/interrupt latency

Timing complete - 23480 ms total

PASS:<Basic timing OK>

EXIT:<done>

Board: Fujitsu SPARClite Evaluation Board

Board: Fujitsu SPARClite Evaluation Board

CPU : Fujitsu SPARClite MB8683X 100MHz

eCOS Kernel Timings

Note: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 0 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 17.19 microseconds (17 raw clock ticks)

Testing parameters:

Clock samples:	32
Threads:	24
Thread switches:	128
Mutexes:	32
Mailboxes:	32
Semaphores:	32
Scheduler operations:	128
Counters:	32
Alarms:	32

Appendix B. Real-time characterization

Ave	Min	Max	Var	Confidence		Function
				Ave	Min	
=====	=====	=====	=====	=====	=====	=====
48.59	47.00	63.01	1.41	66%	70%	Create thread
2.13	2.00	5.00	0.24	95%	95%	Yield thread [all suspended]
2.92	2.00	10.00	0.69	58%	37%	Suspend [suspended] thread
2.13	1.00	10.00	0.66	75%	20%	Resume thread
2.79	2.00	11.00	0.86	95%	54%	Set priority
1.00	0.00	5.00	0.33	79%	16%	Get priority
7.17	5.00	34.00	2.24	95%	95%	Kill [suspended] thread
2.42	2.00	12.00	0.80	95%	95%	Yield [no other] thread
3.46	2.00	14.00	1.10	75%	83%	Resume [suspended low prio] thread
2.00	1.00	9.00	0.58	66%	29%	Resume [runnable low prio] thread
4.21	3.00	20.00	1.38	95%	91%	Suspend [runnable] thread
2.33	2.00	10.00	0.64	95%	95%	Yield [only low prio] thread
2.00	1.00	9.00	0.67	58%	33%	Suspend [runnable->not runnable]
5.79	4.00	30.00	2.07	95%	95%	Kill [runnable] thread
39.34	37.00	75.01	3.36	91%	91%	Resume [high priority] thread
15.20	15.00	31.00	0.40	97%	97%	Thread switch
1.04	1.00	4.00	0.08	97%	97%	Scheduler lock
1.42	1.00	5.00	0.51	60%	60%	Scheduler unlock [0 threads]
1.41	1.00	5.00	0.50	61%	61%	Scheduler unlock [1 suspended]
1.41	1.00	5.00	0.50	60%	60%	Scheduler unlock [many suspended]
1.40	1.00	5.00	0.50	62%	62%	Scheduler unlock [many low prio]
1.19	1.00	6.00	0.35	93%	93%	Init mutex
2.34	2.00	12.00	0.64	93%	93%	Lock [unlocked] mutex
3.41	3.00	13.00	0.71	96%	87%	Unlock [locked] mutex
2.16	1.00	10.00	0.49	87%	9%	Trylock [unlocked] mutex
1.78	1.00	7.00	0.59	59%	37%	Trylock [locked] mutex
0.72	0.00	2.00	0.45	65%	31%	Destroy mutex
25.25	24.00	41.00	0.98	71%	25%	Unlock/Lock mutex
1.44	1.00	9.00	0.68	96%	78%	Create mbox
0.94	0.00	3.00	0.23	84%	12%	Peek [empty] mbox
3.06	2.00	13.00	0.62	71%	25%	Put [first] mbox
0.69	0.00	3.00	0.52	59%	37%	Peek [1 msg] mbox
2.44	2.00	10.00	0.68	96%	78%	Put [second] mbox
0.78	0.00	3.00	0.44	68%	28%	Peek [2 msgs] mbox
3.78	3.00	14.00	0.83	96%	53%	Get [first] mbox
2.97	2.00	9.00	0.61	56%	31%	Get [second] mbox
2.53	2.00	12.00	0.80	96%	75%	Tryput [first] mbox
2.72	2.00	12.00	0.81	96%	56%	Peek item [non-empty] mbox
2.63	2.00	13.00	0.94	90%	75%	Tryget [non-empty] mbox
1.97	1.00	6.00	0.42	68%	21%	Peek item [empty] mbox
2.09	1.00	9.00	0.49	78%	15%	Tryget [empty] mbox
0.84	0.00	4.00	0.42	71%	25%	Waiting to get mbox
0.81	0.00	4.00	0.46	68%	28%	Waiting to put mbox
2.38	2.00	11.00	0.66	96%	87%	Delete mbox
23.41	22.00	47.00	1.47	96%	96%	Put/Get mbox
1.03	0.00	6.00	0.31	84%	12%	Init semaphore
2.66	2.00	8.00	0.66	96%	50%	Post [0] semaphore

1.97	1.00	10.00	0.55	68%	28% Wait [1] semaphore
1.78	1.00	8.00	0.63	56%	40% Trywait [0] semaphore
1.84	1.00	8.00	0.58	62%	34% Trywait [1] semaphore
1.00	0.00	5.00	0.25	84%	12% Peek semaphore
0.81	0.00	4.00	0.46	68%	28% Destroy semaphore
19.03	18.00	41.00	1.37	96%	96% Post/Wait semaphore
1.38	1.00	6.00	0.56	75%	75% Create counter
1.09	1.00	3.00	0.18	93%	93% Get counter value
1.00	0.00	5.00	0.31	78%	15% Set counter value
3.09	2.00	6.00	0.35	78%	9% Tick counter
0.91	0.00	5.00	0.40	75%	21% Delete counter
2.53	2.00	9.00	0.70	96%	65% Create alarm
6.03	5.00	22.00	1.00	50%	46% Initialize alarm
0.78	0.00	4.00	0.49	65%	31% Disable alarm
2.91	2.00	13.00	0.91	87%	50% Enable alarm
0.97	0.00	5.00	0.30	81%	15% Delete alarm
2.69	2.00	9.00	0.69	96%	50% Tick counter [1 alarm]
12.00	11.00	23.00	0.69	62%	34% Tick counter [many alarms]
4.16	3.00	13.00	0.55	84%	12% Tick & fire counter [1 alarm]
72.69	72.01	87.01	1.03	96%	96% Tick & fire counters [>1 together]
13.66	13.00	23.00	0.82	96%	62% Tick & fire counters [>1 separately]
13.26	13.00	42.00	0.51	98%	98% Alarm latency [0 threads]
16.75	11.00	53.01	2.78	64%	16% Alarm latency [2 threads]
24.06	18.00	58.01	3.55	67%	25% Alarm latency [many threads]
3.61	2.00	13.00	0.00		Clock/interrupt latency

Timing complete - 23590 ms total

PASS:<Basic timing OK>

EXIT:<done>

Board: Cogent CMA MPC860 (PowerPC) Evaluation

Board: Cogent CMA MPC860 (PowerPC) Evaluation
 CPU : MPC860, revision A3 33MHz

eCOS Kernel Timings

Note: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 0 'ticks' overhead
 ... this value will be factored out of all other measurements
 Clock interrupt took 14.46 microseconds (30 raw clock ticks)

Testing parameters:

Clock samples: 32
 Threads: 24

Appendix B. Real-time characterization

```

Thread switches:      128
Mutexes:              32
Mailboxes:            32
Semaphores:           32
Scheduler operations: 128
Counters:             32
Alarms:               32

```

				Confidence		
Ave	Min	Max	Var	Ave	Min	Function
=====	=====	=====	=====	=====	=====	=====
26.78	23.52	41.76	1.97	66%	37%	Create thread
4.00	3.84	4.80	0.23	70%	70%	Yield thread [all suspended]
3.78	3.36	7.68	0.38	50%	45%	Suspend [suspended] thread
3.56	3.36	7.68	0.37	95%	91%	Resume thread
5.28	4.32	12.96	0.76	83%	66%	Set priority
0.84	0.48	3.84	0.39	91%	54%	Get priority
11.76	10.08	32.16	1.70	95%	95%	Kill [suspended] thread
4.14	3.84	8.64	0.45	95%	75%	Yield [no other] thread
7.14	5.76	17.76	1.07	79%	70%	Resume [suspended low prio] thread
3.60	3.36	8.16	0.42	95%	87%	Resume [runnable low prio] thread
6.10	5.28	14.88	0.80	62%	70%	Suspend [runnable] thread
4.00	3.84	5.76	0.25	79%	79%	Yield [only low prio] thread
3.66	3.36	8.64	0.47	95%	79%	Suspend [runnable->not runnable]
11.66	10.08	30.24	1.58	79%	91%	Kill [runnable] thread
31.12	27.84	53.28	2.35	87%	50%	Resume [high priority] thread
7.52	7.20	15.84	0.30	50%	48%	Thread switch
1.00	0.48	2.88	0.21	63%	14%	Scheduler lock
2.57	2.40	3.84	0.23	65%	65%	Scheduler unlock [0 threads]
2.58	2.40	4.32	0.23	64%	64%	Scheduler unlock [1 suspended]
2.59	2.40	4.32	0.24	62%	62%	Scheduler unlock [many suspended]
2.59	2.40	4.32	0.24	61%	61%	Scheduler unlock [many low prio]
1.69	1.44	5.76	0.37	96%	71%	Init mutex
4.15	3.84	10.56	0.47	96%	75%	Lock [unlocked] mutex
5.82	5.28	10.56	0.38	62%	28%	Unlock [locked] mutex
3.70	3.36	8.64	0.41	96%	59%	Trylock [unlocked] mutex
3.42	2.88	6.72	0.26	75%	15%	Trylock [locked] mutex
0.36	0.00	1.92	0.25	62%	34%	Destroy mutex
43.41	42.72	45.12	0.34	81%	3%	Unlock/Lock mutex
3.27	2.88	8.16	0.39	96%	50%	Create mbox
0.57	0.00	2.40	0.34	50%	21%	Peek [empty] mbox
6.16	5.76	11.04	0.48	87%	87%	Put [first] mbox
0.48	0.00	1.92	0.27	50%	28%	Peek [1 msg] mbox
5.92	5.28	10.56	0.35	90%	6%	Put [second] mbox
0.60	0.00	2.40	0.30	62%	12%	Peek [2 msgs] mbox
4.69	4.32	12.00	0.54	93%	93%	Get [first] mbox
4.68	4.32	11.52	0.52	93%	93%	Get [second] mbox
5.86	5.28	11.04	0.47	62%	31%	Tryput [first] mbox

Appendix B. Real-time characterization

4.00	3.36	9.12	0.38	87%	9% Peek item [non-empty] mbox
4.59	3.84	12.48	0.61	71%	75% Tryget [non-empty] mbox
3.75	3.36	7.68	0.34	53%	43% Peek item [empty] mbox
3.93	3.36	9.60	0.45	65%	31% Tryget [empty] mbox
0.63	0.00	2.40	0.28	68%	6% Waiting to get mbox
0.54	0.00	1.92	0.19	75%	9% Waiting to put mbox
4.84	4.32	12.00	0.47	56%	40% Delete mbox
24.18	23.52	29.76	0.66	81%	75% Put/Get mbox
1.72	0.96	3.84	0.33	90%	6% Init semaphore
3.15	2.88	6.24	0.34	96%	62% Post [0] semaphore
3.85	3.36	8.64	0.30	68%	28% Wait [1] semaphore
3.24	2.88	6.24	0.34	46%	46% Trywait [0] semaphore
3.22	2.88	6.24	0.32	50%	46% Trywait [1] semaphore
0.96	0.48	2.88	0.12	84%	12% Peek semaphore
0.99	0.96	1.92	0.06	96%	96% Destroy semaphore
24.71	24.00	28.80	0.40	87%	6% Post/Wait semaphore
2.31	1.44	6.24	0.77	46%	56% Create counter
0.45	0.00	0.96	0.08	87%	9% Get counter value
0.42	0.00	0.96	0.16	75%	18% Set counter value
4.14	3.84	4.80	0.26	50%	43% Tick counter
0.91	0.48	2.40	0.19	71%	21% Delete counter
5.23	4.32	7.68	0.61	65%	53% Create alarm
5.58	4.80	12.96	0.72	68%	84% Initialize alarm
0.75	0.48	1.92	0.30	90%	56% Disable alarm
8.02	7.20	14.40	0.53	84%	68% Enable alarm
1.32	0.96	3.84	0.29	56%	40% Delete alarm
4.63	4.32	6.24	0.28	53%	43% Tick counter [1 alarm]
23.67	23.52	25.44	0.23	78%	78% Tick counter [many alarms]
7.24	6.72	10.56	0.21	84%	12% Tick & fire counter [1 alarm]
106.83	106.56	110.40	0.35	96%	65% Tick & fire counters [>1 together]
26.18	25.44	29.76	0.46	81%	9% Tick & fire counters [>1 separately]
10.79	10.08	29.28	0.66	53%	55% Alarm latency [0 threads]
17.20	13.92	35.52	1.48	67%	21% Alarm latency [2 threads]
29.69	22.56	47.04	3.58	57%	17% Alarm latency [many threads]
7.66	3.84	19.20	0.00		Clock/interrupt latency

Timing complete - 23530 ms total

PASS:<Basic timing OK>

EXIT:<done>

Board: NEC VR4373

Board: NEC VR4373

CPU : NEC VR4300 133MHz

```
Startup, main stack      : stack used 1304 size 3576
Startup                  : Interrupt stack used 980 size 4096
Startup                  : Idlethread stack used 494 size 2552
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 3 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 6.49 microseconds (431 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            16
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

				Confidence			
Ave	Min	Max	Var	Ave	Min	Function	
=====	=====	=====	=====	=====	=====	=====	
17.21	16.18	22.14	0.88	75%	68%	Create thread	
0.84	0.78	1.29	0.10	81%	81%	Yield thread [all suspended]	
0.90	0.62	3.20	0.35	87%	87%	Suspend [suspended] thread	
0.74	0.65	1.16	0.12	81%	68%	Resume thread	
1.11	0.90	1.70	0.25	75%	68%	Set priority	
0.11	0.05	0.35	0.09	75%	75%	Get priority	
2.93	2.24	8.27	0.78	93%	75%	Kill [suspended] thread	
0.88	0.78	1.92	0.16	93%	81%	Yield [no other] thread	
1.82	1.20	4.71	0.62	87%	62%	Resume [suspended low prio] thread	
0.70	0.63	0.86	0.09	68%	68%	Resume [runnable low prio] thread	
1.21	1.07	1.61	0.13	81%	68%	Suspend [runnable] thread	
0.86	0.78	1.58	0.13	81%	81%	Yield [only low prio] thread	
0.69	0.62	0.84	0.09	68%	68%	Suspend [runnable->not runnable]	
2.64	2.24	4.35	0.43	81%	62%	Kill [runnable] thread	
1.50	1.07	5.82	0.56	93%	87%	Destroy [dead] thread	
3.66	2.75	7.74	0.82	50%	56%	Destroy [runnable] thread	
13.65	8.33	27.88	3.70	50%	43%	Resume [high priority] thread	
2.04	1.89	3.32	0.15	46%	49%	Thread switch	
0.19	0.05	0.83	0.13	48%	44%	Scheduler lock	
0.50	0.41	1.59	0.13	89%	73%	Scheduler unlock [0 threads]	
0.52	0.41	1.29	0.14	89%	64%	Scheduler unlock [1 suspended]	

Appendix B. Real-time characterization

0.56	0.41	1.49	0.15	42%	47%	Scheduler unlock [many suspended]
0.56	0.41	1.41	0.15	43%	47%	Scheduler unlock [many low prio]
0.57	0.20	2.33	0.27	65%	50%	Init mutex
0.89	0.75	3.35	0.20	96%	75%	Lock [unlocked] mutex
0.90	0.74	4.38	0.25	96%	93%	Unlock [locked] mutex
0.77	0.65	2.63	0.17	96%	75%	Trylock [unlocked] mutex
0.66	0.59	1.16	0.10	75%	75%	Trylock [locked] mutex
0.07	0.00	0.45	0.09	75%	75%	Destroy mutex
7.95	7.71	9.49	0.19	50%	46%	Unlock/Lock mutex
1.04	0.81	3.44	0.27	93%	68%	Create mbox
0.10	0.02	0.57	0.11	71%	68%	Peek [empty] mbox
1.15	0.83	4.71	0.31	53%	71%	Put [first] mbox
0.10	0.02	0.57	0.12	68%	68%	Peek [1 msg] mbox
1.01	0.83	3.83	0.22	93%	75%	Put [second] mbox
0.09	0.02	0.57	0.10	71%	71%	Peek [2 msgs] mbox
1.03	0.81	5.02	0.27	96%	87%	Get [first] mbox
0.93	0.81	1.61	0.14	84%	62%	Get [second] mbox
1.07	0.77	4.18	0.23	68%	50%	Tryput [first] mbox
0.89	0.72	3.49	0.21	93%	71%	Peek item [non-empty] mbox
1.04	0.83	4.09	0.26	90%	81%	Tryget [non-empty] mbox
0.79	0.68	1.97	0.15	87%	68%	Peek item [empty] mbox
0.84	0.72	2.36	0.17	93%	68%	Tryget [empty] mbox
0.13	0.02	0.59	0.13	87%	62%	Waiting to get mbox
0.13	0.02	0.90	0.13	90%	62%	Waiting to put mbox
0.93	0.77	3.23	0.21	90%	71%	Delete mbox
4.74	4.51	8.80	0.32	93%	78%	Put/Get mbox
0.50	0.21	1.95	0.29	90%	50%	Init semaphore
0.86	0.57	2.87	0.29	93%	56%	Post [0] semaphore
1.01	0.74	3.62	0.28	93%	56%	Wait [1] semaphore
0.87	0.60	3.17	0.28	90%	59%	Trywait [0] semaphore
0.74	0.62	1.70	0.14	93%	56%	Trywait [1] semaphore
0.36	0.11	1.11	0.26	65%	56%	Peek semaphore
0.25	0.12	1.19	0.14	93%	56%	Destroy semaphore
7.85	7.52	8.93	0.21	62%	43%	Post/Wait semaphore
0.90	0.44	3.08	0.29	65%	28%	Create counter
0.07	0.05	0.89	0.05	96%	96%	Get counter value
0.06	0.05	0.33	0.02	96%	96%	Set counter value
0.88	0.86	1.62	0.05	96%	96%	Tick counter
0.13	0.12	0.41	0.02	96%	96%	Delete counter
1.37	0.81	2.95	0.27	62%	25%	Create alarm
1.35	1.17	6.03	0.31	96%	93%	Initialize alarm
0.11	0.08	0.65	0.05	90%	90%	Disable alarm
1.23	1.14	3.05	0.15	93%	87%	Enable alarm
0.21	0.18	0.47	0.04	90%	90%	Delete alarm
1.03	0.99	2.11	0.07	96%	96%	Tick counter [1 alarm]
4.96	4.96	4.96	0.00	100%	100%	Tick counter [many alarms]
1.70	1.67	2.51	0.05	96%	96%	Tick & fire counter [1 alarm]
26.39	26.38	26.71	0.02	96%	96%	Tick & fire counters [>1 together]
5.65	5.64	5.91	0.02	96%	96%	Tick & fire counters [>1 separately]
2.55	2.38	9.86	0.19	96%	54%	Alarm latency [0 threads]

Appendix B. Real-time characterization

```
5.37    3.80    9.73    0.95    50%  34% Alarm latency [2 threads]
8.79    5.83   16.12    1.29    57%  14% Alarm latency [many threads]

5.85    2.26   16.24    0.00                Clock/interrupt latency

1540    1536    1544 (main stack: 1664) Thread stack used (2552 total)
All done, main stack          : stack used 1664 size 3576
All done                      : Interrupt stack used 312 size 4096
All done                      : Idlethread stack used 1440 size 2552

Timing complete - 23810 ms total

PASS:<Basic timing OK>
EXIT:<done>
```

Board: Intel SA1110 (Assabet)

Board: Intel SA1110 (Assabet)

CPU : StrongARM 221.2 MHz

```
Microseconds for one run through Dhrystone:      3.3
Dhrystones per Second:                          306748.5
VAX MIPS rating =      174.586
```

```
Startup, main stack          : stack used 420 size 2400
Startup                      : Interrupt stack used 136 size 4096
Startup                      : Idlethread stack used 84 size 2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

```
Reading the hardware clock takes 0 'ticks' overhead
... this value will be factored out of all other measurements
Clock interrupt took      3.20 microseconds (11 raw clock ticks)
```

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

Confidence

Ave	Min	Max	Var	Ave	Min	Function
=====	=====	=====	=====	=====	=====	=====
5.98	4.88	14.38	0.70	57%	35%	Create thread
0.86	0.81	1.90	0.08	87%	87%	Yield thread [all suspended]
1.05	0.81	3.53	0.19	46%	39%	Suspend [suspended] thread
1.07	0.81	3.80	0.18	48%	35%	Resume thread
1.36	1.09	5.97	0.22	45%	39%	Set priority
0.73	0.54	1.90	0.19	85%	50%	Get priority
2.93	2.44	13.56	0.39	79%	70%	Kill [suspended] thread
0.89	0.81	4.34	0.14	89%	89%	Yield [no other] thread
1.63	1.36	4.61	0.17	57%	29%	Resume [suspended low prio] thread
1.03	0.81	3.53	0.19	46%	42%	Resume [runnable low prio] thread
1.74	1.36	6.51	0.22	87%	6%	Suspend [runnable] thread
0.93	0.81	4.61	0.18	98%	78%	Yield [only low prio] thread
1.06	0.81	3.26	0.19	42%	39%	Suspend [runnable->not runnable]
2.56	1.90	13.02	0.41	87%	34%	Kill [runnable] thread
2.02	1.63	7.05	0.22	92%	3%	Destroy [dead] thread
3.09	2.44	15.19	0.51	78%	46%	Destroy [runnable] thread
6.77	5.43	13.02	0.59	75%	17%	Resume [high priority] thread
1.81	1.63	7.87	0.18	49%	49%	Thread switch
0.25	0.00	1.36	0.05	89%	10%	Scheduler lock
0.51	0.27	1.36	0.06	85%	13%	Scheduler unlock [0 threads]
0.51	0.27	1.09	0.06	85%	13%	Scheduler unlock [1 suspended]
0.51	0.27	1.09	0.07	85%	14%	Scheduler unlock [many suspended]
0.51	0.27	1.09	0.06	85%	13%	Scheduler unlock [many low prio]
0.52	0.27	2.17	0.15	62%	31%	Init mutex
0.97	0.54	4.34	0.28	84%	65%	Lock [unlocked] mutex
1.05	0.81	5.15	0.28	96%	96%	Unlock [locked] mutex
0.86	0.54	3.26	0.24	65%	31%	Trylock [unlocked] mutex
0.79	0.54	3.53	0.23	43%	46%	Trylock [locked] mutex
0.33	0.27	1.63	0.11	90%	90%	Destroy mutex
4.16	3.80	8.95	0.30	75%	96%	Unlock/Lock mutex
0.70	0.54	2.98	0.21	96%	65%	Create mbox
0.59	0.27	1.63	0.14	75%	9%	Peek [empty] mbox
1.33	1.09	5.70	0.31	96%	93%	Put [first] mbox
0.61	0.27	1.63	0.13	81%	3%	Peek [1 msg] mbox
1.35	1.09	5.43	0.31	96%	87%	Put [second] mbox
0.58	0.27	1.36	0.11	78%	6%	Peek [2 msgs] mbox
1.38	1.09	4.88	0.25	59%	37%	Get [first] mbox
1.40	1.09	5.15	0.26	62%	34%	Get [second] mbox
1.27	0.81	4.88	0.28	90%	65%	Tryput [first] mbox
1.34	0.81	4.61	0.22	59%	6%	Peek item [non-empty] mbox
1.47	1.09	5.15	0.27	84%	12%	Tryget [non-empty] mbox
1.12	0.81	4.34	0.23	59%	31%	Peek item [empty] mbox
1.14	0.81	4.07	0.24	71%	25%	Tryget [empty] mbox
0.59	0.27	1.36	0.12	78%	6%	Waiting to get mbox
0.59	0.27	1.36	0.12	78%	6%	Waiting to put mbox
1.28	0.81	5.43	0.32	87%	78%	Delete mbox
2.64	2.17	10.31	0.48	96%	96%	Put/Get mbox
0.47	0.27	2.17	0.19	46%	46%	Init semaphore
0.77	0.54	3.80	0.26	90%	56%	Post [0] semaphore

Appendix B. Real-time characterization

0.90	0.54	4.07	0.26	75%	21% Wait [1] semaphore
0.85	0.54	3.26	0.21	56%	28% Trywait [0] semaphore
0.69	0.54	2.17	0.18	96%	62% Trywait [1] semaphore
0.44	0.27	2.17	0.19	96%	56% Peek semaphore
0.38	0.27	1.90	0.17	96%	75% Destroy semaphore
2.74	2.44	9.49	0.42	96%	96% Post/Wait semaphore
0.43	0.27	1.90	0.18	96%	56% Create counter
0.49	0.00	2.17	0.18	56%	3% Get counter value
0.33	0.00	1.63	0.13	78%	6% Set counter value
1.03	0.81	2.44	0.22	84%	50% Tick counter
0.42	0.27	1.90	0.20	90%	65% Delete counter
0.70	0.54	2.44	0.20	93%	62% Create alarm
1.65	1.36	6.78	0.40	96%	81% Initialize alarm
0.75	0.54	1.63	0.18	43%	43% Disable alarm
1.75	1.36	7.05	0.38	65%	81% Enable alarm
0.81	0.54	2.44	0.15	62%	28% Delete alarm
1.01	0.81	2.17	0.16	56%	40% Tick counter [1 alarm]
4.19	4.07	5.43	0.16	96%	68% Tick counter [many alarms]
1.48	1.36	3.80	0.20	96%	78% Tick & fire counter [1 alarm]
20.23	20.07	22.52	0.21	96%	65% Tick & fire counters [>1 together]
4.70	4.61	6.78	0.16	87%	87% Tick & fire counters [>1 separately]
2.81	2.71	14.38	0.20	98%	98% Alarm latency [0 threads]
3.19	2.71	13.56	0.38	73%	59% Alarm latency [2 threads]
9.71	7.87	18.17	1.25	59%	53% Alarm latency [many threads]
5.77	5.43	45.57	0.68	97%	97% Alarm -> thread resume latency
2.38	0.81	9.49	0.00		Clock/interrupt latency
2.02	1.09	7.32	0.00		Clock DSR latency
11	0	316	(main stack: 764)	Thread stack used (1120 total)	
All done, main stack : stack used 764 size 2400					
All done : Interrupt stack used 287 size 4096					
All done : Idlethread stack used 272 size 2048					
Timing complete - 30220 ms total					

Board: Intel SA1100 (Brutus)

Board: Intel SA1100 (Brutus)

CPU : StrongARM 221.2 MHz

Microseconds for one run through Dhrystone: 3.3
 Dhrystones per Second: 306748.5
 VAX MIPS rating = 174.586

Startup, main stack : stack used 404 size 2400
 Startup : Interrupt stack used 136 size 4096

Startup : Idlethread stack used 87 size 2048

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 0 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 3.09 microseconds (11 raw clock ticks)

Testing parameters:

Clock samples:	32
Threads:	64
Thread switches:	128
Mutexes:	32
Mailboxes:	32
Semaphores:	32
Scheduler operations:	128
Counters:	32
Alarms:	32

Ave	Min	Max	Var	Confidence	Ave	Min	Function
=====	=====	=====	=====	=====	=====	=====	=====
6.63	5.43	18.99	0.77	70%	37%		Create thread
0.83	0.81	2.17	0.04	98%	98%		Yield thread [all suspended]
1.27	0.81	5.15	0.30	68%	73%		Suspend [suspended] thread
1.25	0.81	5.15	0.25	82%	1%		Resume thread
1.52	1.09	7.87	0.30	78%	75%		Set priority
0.97	0.54	2.71	0.28	64%	51%		Get priority
3.45	2.71	19.53	0.66	84%	76%		Kill [suspended] thread
0.90	0.81	6.24	0.17	98%	98%		Yield [no other] thread
1.86	1.36	6.24	0.33	68%	50%		Resume [suspended low prio] thread
1.25	0.81	5.15	0.25	82%	1%		Resume [runnable low prio] thread
2.01	1.63	10.04	0.32	70%	84%		Suspend [runnable] thread
0.90	0.81	6.24	0.17	98%	98%		Yield [only low prio] thread
1.25	0.81	5.15	0.24	84%	1%		Suspend [runnable->not runnable]
2.92	1.90	18.72	0.57	85%	43%		Kill [runnable] thread
2.45	1.90	10.31	0.33	95%	54%		Destroy [dead] thread
3.95	2.71	23.60	0.89	68%	54%		Destroy [runnable] thread
8.55	6.24	19.53	1.15	60%	23%		Resume [high priority] thread
1.85	1.63	11.94	0.21	49%	49%		Thread switch
0.25	0.00	1.63	0.05	89%	10%		Scheduler lock
0.52	0.27	1.90	0.07	85%	13%		Scheduler unlock [0 threads]
0.51	0.27	1.36	0.06	85%	13%		Scheduler unlock [1 suspended]
0.51	0.27	1.36	0.06	85%	13%		Scheduler unlock [many suspended]
0.51	0.27	1.63	0.06	85%	13%		Scheduler unlock [many low prio]
0.58	0.27	3.53	0.20	71%	21%		Init mutex
1.07	0.54	5.70	0.35	87%	59%		Lock [unlocked] mutex
1.14	0.81	6.51	0.40	96%	81%		Unlock [locked] mutex
0.96	0.54	5.15	0.34	68%	65%		Trylock [unlocked] mutex
0.94	0.54	4.88	0.34	65%	65%		Trylock [locked] mutex
0.33	0.27	2.17	0.11	96%	96%		Destroy mutex

Appendix B. Real-time characterization

4.21	3.80	10.85	0.41	71%	96% Unlock/Lock mutex
0.76	0.54	4.07	0.25	96%	56% Create mbox
0.75	0.54	1.90	0.20	84%	50% Peek [empty] mbox
1.56	1.09	6.78	0.39	68%	59% Put [first] mbox
0.75	0.54	1.90	0.20	84%	50% Peek [1 msg] mbox
1.55	1.09	6.78	0.40	68%	62% Put [second] mbox
0.77	0.54	1.63	0.17	46%	37% Peek [2 msgs] mbox
1.67	1.09	6.24	0.31	87%	34% Get [first] mbox
1.63	1.09	6.24	0.31	75%	34% Get [second] mbox
1.50	1.09	6.51	0.40	56%	62% Tryput [first] mbox
1.58	1.09	5.43	0.37	68%	53% Peek item [non-empty] mbox
1.79	1.09	7.05	0.43	71%	25% Tryget [non-empty] mbox
1.29	1.09	5.15	0.32	87%	87% Peek item [empty] mbox
1.33	1.09	5.97	0.37	96%	84% Tryget [empty] mbox
0.73	0.54	1.90	0.21	84%	56% Waiting to get mbox
0.76	0.54	1.90	0.19	40%	43% Waiting to put mbox
1.47	1.09	6.78	0.39	59%	84% Delete mbox
2.70	2.17	12.75	0.63	96%	96% Put/Get mbox
0.47	0.27	2.71	0.20	96%	50% Init semaphore
0.89	0.54	4.88	0.33	56%	75% Post [0] semaphore
0.96	0.54	5.15	0.33	71%	75% Wait [1] semaphore
0.86	0.54	4.88	0.32	96%	81% Trywait [0] semaphore
0.69	0.54	3.26	0.22	96%	75% Trywait [1] semaphore
0.49	0.27	3.26	0.28	84%	84% Peek semaphore
0.39	0.27	2.44	0.19	96%	78% Destroy semaphore
2.83	2.44	11.66	0.55	96%	96% Post/Wait semaphore
0.52	0.27	3.26	0.20	56%	40% Create counter
0.59	0.00	2.71	0.34	81%	46% Get counter value
0.36	0.00	2.44	0.21	81%	9% Set counter value
1.13	0.81	2.98	0.26	59%	37% Tick counter
0.39	0.27	1.90	0.19	90%	78% Delete counter
0.86	0.54	4.07	0.24	65%	31% Create alarm
1.86	1.36	9.77	0.54	96%	90% Initialize alarm
0.77	0.54	2.71	0.23	84%	50% Disable alarm
1.86	1.36	9.22	0.51	96%	75% Enable alarm
0.89	0.54	3.26	0.25	65%	21% Delete alarm
0.99	0.81	3.26	0.21	96%	59% Tick counter [1 alarm]
4.22	4.07	6.78	0.22	96%	71% Tick counter [many alarms]
1.51	1.36	4.61	0.24	96%	78% Tick & fire counter [1 alarm]
20.29	20.07	23.33	0.23	96%	53% Tick & fire counters [>1 together]
4.71	4.61	7.87	0.20	96%	96% Tick & fire counters [>1 separately]
2.88	2.71	23.87	0.33	99%	99% Alarm latency [0 threads]
3.24	2.71	17.36	0.40	79%	58% Alarm latency [2 threads]
15.71	12.48	27.40	1.47	53%	17% Alarm latency [many threads]
5.95	5.43	64.56	1.02	97%	97% Alarm -> thread resume latency
3.25	0.81	14.11	0.00		Clock/interrupt latency
2.68	1.09	12.75	0.00		Clock DSR latency
29	0	316	(main stack: 764)	Thread stack used (1120 total)	
All done, main stack			:	stack used	764 size 2400


```

All done          : Interrupt stack used 288 size 4096
All done          : Idlethread stack used 260 size 2048

```

Timing complete - 30280 ms total

Board: Motorola MBX

Board: Motorola MBX

CPU : Motorola MPC860 66MHZ

```

Startup, main stack      : stack used 643 size 5664
Startup                  : Interrupt stack used 427 size 4096
Startup                  : Idlethread stack used 236 size 2048

```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 0 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 25.36 microseconds (79 raw clock ticks)

Testing parameters:

```

Clock samples:      32
Threads:            16
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32

```

				Confidence		
Ave	Min	Max	Var	Ave	Min	Function
=====	=====	=====	=====	=====	=====	=====
27.58	25.60	44.16	2.07	93%	93%	Create thread
5.94	5.76	7.04	0.22	93%	62%	Yield thread [all suspended]
6.06	5.44	10.56	0.57	75%	75%	Suspend [suspended] thread
5.42	4.80	9.60	0.53	87%	81%	Resume thread
7.10	6.40	14.08	0.90	93%	87%	Set priority
0.86	0.64	1.92	0.22	93%	50%	Get priority
16.74	15.04	36.48	2.47	93%	93%	Kill [suspended] thread
6.14	5.76	10.56	0.55	93%	93%	Yield [no other] thread
9.74	8.96	18.56	1.10	93%	93%	Resume [suspended low prio] thread
5.28	4.80	9.28	0.54	93%	81%	Resume [runnable low prio] thread
9.40	8.32	18.56	1.14	93%	93%	Suspend [runnable] thread

Appendix B. Real-time characterization

6.04	5.76	8.96	0.38	93%	93% Yield [only low prio] thread
5.68	5.12	9.60	0.52	68%	75% Suspend [runnable->not runnable]
16.10	14.40	35.20	2.39	93%	93% Kill [runnable] thread
8.54	7.68	16.00	0.94	93%	87% Destroy [dead] thread
20.20	18.56	40.64	2.55	93%	93% Destroy [runnable] thread
39.02	36.48	57.28	3.28	87%	87% Resume [high priority] thread
13.13	12.80	22.08	0.15	78%	20% Thread switch
0.59	0.32	1.60	0.09	82%	16% Scheduler lock
3.67	3.52	5.12	0.17	99%	54% Scheduler unlock [0 threads]
3.67	3.52	4.80	0.17	99%	53% Scheduler unlock [1 suspended]
3.67	3.52	4.80	0.17	54%	54% Scheduler unlock [many suspended]
3.69	3.52	5.12	0.17	99%	50% Scheduler unlock [many low prio]
2.41	2.24	5.44	0.25	96%	75% Init mutex
6.83	6.40	11.84	0.34	75%	90% Lock [unlocked] mutex
6.74	6.40	13.12	0.40	96%	96% Unlock [locked] mutex
5.53	5.12	9.60	0.25	84%	12% Trylock [unlocked] mutex
4.84	4.48	7.36	0.17	78%	15% Trylock [locked] mutex
0.34	0.00	0.96	0.06	90%	3% Destroy mutex
56.10	55.68	59.52	0.21	93%	3% Unlock/Lock mutex
4.72	4.48	10.24	0.37	96%	96% Create mbox
0.75	0.64	1.92	0.16	75%	75% Peek [empty] mbox
6.79	6.40	12.80	0.41	96%	90% Put [first] mbox
0.46	0.32	1.60	0.19	93%	68% Peek [1 msg] mbox
6.68	6.40	12.16	0.37	96%	96% Put [second] mbox
0.50	0.32	1.60	0.20	93%	56% Peek [2 msgs] mbox
7.13	6.40	14.08	0.49	90%	46% Get [first] mbox
6.97	6.40	13.44	0.47	84%	78% Get [second] mbox
6.24	5.76	11.52	0.38	78%	81% Tryput [first] mbox
5.98	5.44	11.20	0.39	78%	62% Peek item [non-empty] mbox
6.52	6.08	13.12	0.49	93%	81% Tryget [non-empty] mbox
5.50	5.12	10.24	0.30	68%	28% Peek item [empty] mbox
5.76	5.44	10.88	0.32	96%	96% Tryget [empty] mbox
0.50	0.32	1.60	0.19	96%	53% Waiting to get mbox
0.50	0.32	1.60	0.19	96%	53% Waiting to put mbox
7.45	7.04	15.04	0.49	96%	93% Delete mbox
37.47	36.80	48.64	0.70	96%	96% Put/Get mbox
2.49	2.24	6.08	0.28	96%	56% Init semaphore
5.09	4.80	8.64	0.27	46%	46% Post [0] semaphore
6.25	5.76	10.88	0.32	93%	3% Wait [1] semaphore
4.84	4.48	8.32	0.23	68%	25% Trywait [0] semaphore
4.98	4.80	8.00	0.26	96%	71% Trywait [1] semaphore
1.66	1.28	3.84	0.20	68%	15% Peek semaphore
1.24	0.96	3.20	0.17	65%	31% Destroy semaphore
40.74	40.32	49.28	0.53	96%	96% Post/Wait semaphore
2.65	2.24	6.08	0.23	84%	9% Create counter
0.85	0.64	2.24	0.22	90%	53% Get counter value
0.68	0.64	1.92	0.08	96%	96% Set counter value
7.13	6.72	8.64	0.24	78%	18% Tick counter
1.30	0.96	3.20	0.12	84%	12% Delete counter

```

3.69    3.52    7.68    0.29    96%   84% Create alarm
8.98    8.32   17.60    0.61    68%   62% Initialize alarm
0.96    0.64    2.88    0.14    71%   21% Disable alarm
8.76    8.32   17.60    0.59    96%   87% Enable alarm
1.99    1.60    5.12    0.21    81%   12% Delete alarm
7.44    7.36    9.92    0.15    96%   96% Tick counter [1 alarm]
21.68   21.44   24.64    0.25    96%   53% Tick counter [many alarms]
10.95   10.56   15.04    0.26    78%   18% Tick & fire counter [1 alarm]
132.79  132.48  136.32    0.23    59%   37% Tick & fire counters [>1 together]
25.18   24.96   28.80    0.29    96%   65% Tick & fire counters [>1 separately]
23.06   22.72   47.36    0.40    98%   98% Alarm latency [0 threads]
31.53   27.20   56.00    0.63    96%    0% Alarm latency [2 threads]
36.86   30.40   58.88    4.15    50%   28% Alarm latency [many threads]

11.41    8.96   16.32    0.00                Clock/interrupt latency

609      603      651 (main stack: 1059) Thread stack used (1704 total)
All done, main stack          : stack used 1059 size 5664
All done                      : Interrupt stack used 251 size 4096
All done                      : Idlethread stack used 587 size 2048

Timing complete - 23690 ms total

PASS:<Basic timing OK>
EXIT:<done>

```

Board: Hitachi EDK7708

Board: Hitachi EDK7708

CPU: Hitachi SH3/7708 60MHz

```

Startup, main stack          : stack used 444 size 4112
Startup                      : Interrupt stack used 76 size 4096
Startup                      : Idlethread stack used 96 size 2048

```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

```

Reading the hardware clock takes 2 'ticks' overhead
... this value will be factored out of all other measurements
Clock interrupt took 14.75 microseconds (55 raw clock ticks)

```

Testing parameters:

```

Clock samples:      32
Threads:            16
Thread switches:    128

```

Appendix B. Real-time characterization

```

Mutexes:          32
Mailboxes:        32
Semaphores:       32
Scheduler operations: 128
Counters:         32
Alarms:           32

```

				Confidence			Function
Ave	Min	Max	Var	Ave	Min	Function	
=====	=====	=====	=====	=====	=====	=====	
15.43	13.60	24.00	1.29	62%	50%	Create thread	
3.33	3.20	4.27	0.18	93%	68%	Yield thread [all suspended]	
2.90	2.40	5.33	0.36	81%	62%	Suspend [suspended] thread	
2.93	2.67	4.80	0.27	93%	87%	Resume thread	
4.30	3.73	10.13	0.73	93%	93%	Set priority	
0.65	0.27	2.13	0.28	68%	62%	Get priority	
9.72	8.53	21.33	1.45	93%	93%	Kill [suspended] thread	
3.33	3.20	4.53	0.20	93%	75%	Yield [no other] thread	
5.30	4.80	10.13	0.65	93%	87%	Resume [suspended low prio] thread	
2.80	2.40	4.53	0.27	81%	75%	Resume [runnable low prio] thread	
4.82	4.00	8.27	0.49	68%	25%	Suspend [runnable] thread	
3.32	3.20	4.00	0.16	93%	68%	Yield [only low prio] thread	
2.82	2.40	4.27	0.25	81%	12%	Suspend [runnable->not runnable]	
9.45	8.53	19.47	1.25	93%	93%	Kill [runnable] thread	
5.30	4.53	11.20	0.74	87%	93%	Destroy [dead] thread	
11.83	10.67	25.07	1.65	93%	93%	Destroy [runnable] thread	
19.53	17.33	31.20	1.88	75%	75%	Resume [high priority] thread	
6.70	6.67	11.47	0.07	99%	99%	Thread switch	
0.33	0.27	0.80	0.10	75%	75%	Scheduler lock	
1.74	1.60	2.67	0.14	99%	50%	Scheduler unlock [0 threads]	
1.72	1.60	3.20	0.14	99%	57%	Scheduler unlock [1 suspended]	
1.81	1.60	3.20	0.10	75%	23%	Scheduler unlock [many suspended]	
1.86	1.60	3.20	0.02	94%	4%	Scheduler unlock [many low prio]	
1.22	1.07	3.20	0.20	96%	65%	Init mutex	
3.21	2.93	5.87	0.17	68%	28%	Lock [unlocked] mutex	
3.36	2.93	7.47	0.30	84%	75%	Unlock [locked] mutex	
2.83	2.67	5.33	0.22	96%	65%	Trylock [unlocked] mutex	
2.53	2.40	2.93	0.14	96%	53%	Trylock [locked] mutex	
0.28	0.27	0.80	0.03	96%	96%	Destroy mutex	
20.09	19.73	23.20	0.23	84%	12%	Unlock/Lock mutex	
2.38	2.13	4.53	0.17	59%	34%	Create mbox	
0.45	0.27	1.33	0.15	56%	40%	Peek [empty] mbox	
3.70	3.20	7.20	0.29	84%	59%	Put [first] mbox	
0.45	0.27	0.80	0.13	62%	34%	Peek [1 msg] mbox	
3.67	3.20	5.60	0.23	81%	6%	Put [second] mbox	
0.42	0.27	0.53	0.13	59%	40%	Peek [2 msgs] mbox	
3.98	3.47	7.47	0.24	59%	9%	Get [first] mbox	
3.97	3.47	4.80	0.24	59%	12%	Get [second] mbox	
3.51	3.20	6.67	0.28	56%	78%	Tryput [first] mbox	
3.29	2.93	5.60	0.29	59%	65%	Peek item [non-empty] mbox	
4.06	3.47	7.20	0.26	68%	3%	Tryget [non-empty] mbox	

Appendix B. Real-time characterization

3.03	2.67	5.33	0.19	93%	3% Peek item [empty] mbox
3.36	3.20	4.80	0.18	96%	56% Tryget [empty] mbox
0.57	0.27	1.33	0.09	84%	3% Waiting to get mbox
0.52	0.27	1.07	0.11	62%	21% Waiting to put mbox
3.88	3.47	7.47	0.30	78%	65% Delete mbox
12.04	11.73	17.33	0.33	96%	96% Put/Get mbox
1.17	1.07	2.40	0.16	71%	71% Init semaphore
2.67	2.40	4.27	0.15	62%	25% Post [0] semaphore
3.00	2.67	4.53	0.17	65%	12% Wait [1] semaphore
2.54	2.40	4.80	0.20	96%	71% Trywait [0] semaphore
2.42	2.40	2.93	0.03	96%	96% Trywait [1] semaphore
0.79	0.53	2.13	0.15	59%	28% Peek semaphore
0.77	0.53	1.87	0.12	71%	25% Destroy semaphore
12.64	12.27	17.07	0.28	84%	96% Post/Wait semaphore
1.27	1.07	2.93	0.17	53%	43% Create counter
0.54	0.27	1.33	0.13	59%	21% Get counter value
0.47	0.27	1.60	0.17	46%	43% Set counter value
3.47	3.20	4.80	0.16	53%	28% Tick counter
0.80	0.53	2.13	0.13	62%	25% Delete counter
1.86	1.60	4.00	0.21	43%	40% Create alarm
5.12	4.80	9.07	0.36	93%	75% Initialize alarm
0.44	0.27	1.33	0.19	87%	53% Disable alarm
4.77	4.27	9.60	0.35	87%	62% Enable alarm
1.02	0.80	2.67	0.18	53%	40% Delete alarm
3.56	3.47	5.33	0.15	84%	84% Tick counter [1 alarm]
15.04	14.93	16.27	0.16	71%	71% Tick counter [many alarms]
5.75	5.60	8.00	0.21	96%	68% Tick & fire counter [1 alarm]
79.60	79.47	81.07	0.17	96%	65% Tick & fire counters [>1 together]
17.04	16.80	18.93	0.15	65%	31% Tick & fire counters [>1 separately]
12.44	12.27	29.60	0.31	96%	96% Alarm latency [0 threads]
14.06	12.27	27.20	0.53	82%	4% Alarm latency [2 threads]
19.62	17.07	38.40	1.44	57%	34% Alarm latency [many threads]
2.79	2.40	6.13	0.00		Clock/interrupt latency
376	376	376	(main stack: 764)	Thread stack used (992 total)	
All done, main stack : stack used 764 size 4112					
All done : Interrupt stack used 176 size 4096					
All done : Idlethread stack used 352 size 2048					
Timing complete - 23860 ms total					
PASS:<Basic timing OK>					
EXIT:<done>					

Board: CQ CqREEK SH3 Evaluation Board (cq7708)

Board: CQ CqREEK SH3 Evaluation Board (cq7708)

CPU: Hitachi SH3/7708 60MHz

```
Startup, main stack      : stack used   448 size  4112
Startup                  : Interrupt stack used   80 size  4096
Startup                  : Idlethread stack used   96 size  2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 2 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 19.17 microseconds (71 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

				Confidence			
Ave	Min	Max	Var	Ave	Min	Function	
=====	=====	=====	=====	=====	=====	=====	
20.62	14.40	26.93	3.23	48%	26%	Create thread	
3.16	2.93	4.27	0.09	78%	20%	Yield thread [all suspended]	
2.91	2.40	5.87	0.17	57%	1%	Suspend [suspended] thread	
2.73	2.40	6.40	0.19	64%	15%	Resume thread	
4.05	3.73	11.47	0.27	62%	90%	Set priority	
0.82	0.27	2.67	0.17	56%	3%	Get priority	
9.07	8.53	24.27	0.51	78%	71%	Kill [suspended] thread	
3.19	2.93	7.20	0.14	70%	28%	Yield [no other] thread	
5.45	4.53	17.87	0.49	78%	17%	Resume [suspended low prio] thread	
2.67	2.40	5.07	0.15	56%	28%	Resume [runnable low prio] thread	
4.95	4.27	11.47	0.28	82%	14%	Suspend [runnable] thread	
3.15	2.93	4.53	0.11	73%	25%	Yield [only low prio] thread	
2.82	2.40	5.60	0.21	84%	10%	Suspend [runnable->not runnable]	
8.92	8.00	24.27	0.51	84%	14%	Kill [runnable] thread	
5.10	4.53	12.00	0.27	59%	39%	Destroy [dead] thread	
11.81	10.93	37.33	0.81	87%	95%	Destroy [runnable] thread	
22.15	20.80	54.67	1.27	92%	92%	Resume [high priority] thread	
6.85	6.67	13.60	0.19	99%	50%	Thread switch	
0.27	0.27	1.07	0.01	99%	99%	Scheduler lock	
1.74	1.60	2.67	0.14	99%	50%	Scheduler unlock [0 threads]	
1.74	1.60	2.93	0.14	99%	50%	Scheduler unlock [1 suspended]	
1.81	1.60	4.27	0.11	72%	26%	Scheduler unlock [many suspended]	

Appendix B. Real-time characterization

1.75	1.60	4.00	0.15	50%	49% Scheduler unlock [many low prio]
1.22	1.07	4.27	0.23	96%	78% Init mutex
3.18	2.93	7.20	0.27	96%	53% Lock [unlocked] mutex
3.40	3.20	8.00	0.31	96%	96% Unlock [locked] mutex
2.77	2.40	5.87	0.22	87%	9% Trylock [unlocked] mutex
2.35	2.13	3.47	0.14	65%	31% Trylock [locked] mutex
0.78	0.53	2.67	0.14	68%	28% Destroy mutex
22.80	22.40	28.80	0.51	96%	71% Unlock/Lock mutex
2.61	2.40	6.13	0.26	96%	62% Create mbox
0.52	0.27	1.60	0.19	40%	37% Peek [empty] mbox
3.54	3.20	7.73	0.35	93%	78% Put [first] mbox
0.50	0.27	1.60	0.17	46%	37% Peek [1 msg] mbox
3.62	3.20	6.93	0.34	59%	65% Put [second] mbox
0.52	0.27	2.13	0.23	31%	43% Peek [2 msgs] mbox
3.93	3.47	10.13	0.43	65%	65% Get [first] mbox
3.92	3.47	7.47	0.40	56%	56% Get [second] mbox
3.37	2.93	6.93	0.36	59%	68% Tryput [first] mbox
3.30	2.67	6.93	0.38	84%	40% Peek item [non-empty] mbox
3.93	3.47	9.33	0.44	65%	71% Tryget [non-empty] mbox
2.94	2.67	6.13	0.25	43%	43% Peek item [empty] mbox
3.23	2.93	6.67	0.27	56%	84% Tryget [empty] mbox
0.58	0.27	2.67	0.20	62%	21% Waiting to get mbox
0.55	0.27	1.87	0.14	62%	21% Waiting to put mbox
3.82	3.47	9.87	0.39	96%	93% Delete mbox
13.35	12.80	21.33	0.50	87%	78% Put/Get mbox
1.22	1.07	2.93	0.19	96%	59% Init semaphore
2.42	2.13	4.27	0.12	81%	15% Post [0] semaphore
2.96	2.67	5.07	0.16	68%	21% Wait [1] semaphore
2.37	2.13	4.53	0.17	62%	34% Trywait [0] semaphore
2.29	2.13	3.47	0.17	96%	53% Trywait [1] semaphore
0.66	0.53	2.13	0.17	96%	68% Peek semaphore
0.81	0.53	2.93	0.13	75%	21% Destroy semaphore
14.47	14.13	21.33	0.43	96%	96% Post/Wait semaphore
1.44	1.07	3.47	0.29	56%	71% Create counter
0.62	0.27	1.07	0.14	62%	3% Get counter value
0.56	0.27	1.60	0.17	50%	25% Set counter value
3.39	3.20	4.27	0.16	53%	40% Tick counter
0.83	0.53	1.87	0.14	68%	15% Delete counter
2.02	1.87	4.00	0.21	93%	68% Create alarm
5.06	4.27	11.73	0.46	78%	18% Initialize alarm
0.73	0.27	2.40	0.22	84%	3% Disable alarm
4.82	4.27	11.47	0.48	81%	65% Enable alarm
1.19	0.80	3.47	0.22	87%	9% Delete alarm
3.63	3.47	5.60	0.20	96%	59% Tick counter [1 alarm]
15.01	14.93	16.53	0.13	87%	87% Tick counter [many alarms]
5.50	5.33	8.00	0.22	96%	65% Tick & fire counter [1 alarm]
74.27	74.13	76.80	0.21	96%	78% Tick & fire counters [>1 together]
16.90	16.53	19.47	0.23	81%	15% Tick & fire counters [>1 separately]
16.70	16.53	36.27	0.33	98%	98% Alarm latency [0 threads]
17.85	16.53	34.40	0.47	73%	0% Alarm latency [2 threads]

Appendix B. Real-time characterization

```
63.26  58.40  80.00  2.64  52%  32% Alarm latency [many threads]
30.37  29.33 124.80  1.68  98%  97% Alarm -> thread resume latency

7.37   5.07  17.87  0.00           Clock/interrupt latency

9.00   4.53  26.93  0.00           Clock DSR latency

106      0    376 (main stack: 764) Thread stack used (992 total)
All done, main stack      : stack used 764 size 4112
All done                  : Interrupt stack used 176 size 4096
All done                  : Idlethread stack used 352 size 2048

Timing complete - 30310 ms total

PASS:<Basic timing OK>
EXIT:<done>
```

Board: Hitachi HS7729PCI HS7729 SH3

Board: Hitachi HS7729PCI HS7729 SH3

CPU: Hitachi SH3/7729 132MHz

```
Startup, main stack      : stack used 464 size 4112
Startup                  : Interrupt stack used 92 size 4096
Startup                  : Idlethread stack used 94 size 2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 3 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 18.10 microseconds (149 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

```

                                Confidence
      Ave      Min      Max      Var      Ave      Min      Function
=====
=====
```


18.33	15.52	28.24	1.47	53%	28% Create thread
3.08	2.91	6.79	0.13	78%	89% Yield thread [all suspended]
3.23	3.03	6.18	0.16	59%	70% Suspend [suspended] thread
2.70	2.55	6.18	0.15	54%	82% Resume thread
4.12	4.00	7.52	0.16	96%	81% Set priority
0.61	0.48	1.33	0.07	57%	28% Get priority
9.14	8.61	18.91	0.42	85%	57% Kill [suspended] thread
3.04	2.91	4.48	0.07	68%	20% Yield [no other] thread
5.12	4.73	7.88	0.29	60%	53% Resume [suspended low prio] thread
2.54	2.42	3.03	0.09	39%	40% Resume [runnable low prio] thread
5.00	4.36	9.45	0.21	75%	1% Suspend [runnable] thread
3.04	2.91	4.61	0.07	65%	21% Yield [only low prio] thread
2.91	2.79	3.27	0.08	43%	31% Suspend [runnable->not runnable]
8.82	8.12	15.39	0.36	68%	29% Kill [runnable] thread
5.07	4.48	12.73	0.37	76%	50% Destroy [dead] thread
11.17	10.55	22.91	0.52	78%	67% Destroy [runnable] thread
22.43	21.45	32.73	0.61	81%	50% Resume [high priority] thread
7.99	7.88	13.58	0.14	98%	86% Thread switch
0.37	0.36	1.33	0.02	97%	97% Scheduler lock
1.74	1.70	2.06	0.06	70%	70% Scheduler unlock [0 threads]
1.75	1.70	2.06	0.07	92%	64% Scheduler unlock [1 suspended]
1.71	1.70	2.42	0.03	89%	89% Scheduler unlock [many suspended]
1.76	1.70	3.64	0.08	96%	64% Scheduler unlock [many low prio]
4.23	3.88	10.67	0.41	96%	93% Unlock [locked] mutex
3.12	2.91	6.91	0.29	96%	87% Trylock [unlocked] mutex
2.54	2.42	2.91	0.11	18%	46% Trylock [locked] mutex
0.88	0.73	3.15	0.14	65%	96% Destroy mutex
22.33	22.06	25.94	0.23	81%	62% Unlock/Lock mutex
1.92	1.82	4.73	0.19	96%	93% Create mbox
0.61	0.48	1.70	0.15	84%	75% Peek [empty] mbox
4.00	3.64	9.45	0.36	96%	87% Put [first] mbox
0.30	0.24	0.73	0.09	84%	75% Peek [1 msg] mbox
3.82	3.64	6.67	0.22	90%	84% Put [second] mbox
0.32	0.24	1.33	0.12	81%	81% Peek [2 msgs] mbox
4.19	3.76	9.21	0.34	84%	50% Get [first] mbox
3.91	3.76	5.21	0.16	84%	75% Get [second] mbox
3.51	3.27	8.12	0.34	93%	87% Tryput [first] mbox
3.25	2.91	7.15	0.30	62%	56% Peek item [non-empty] mbox
3.86	3.52	8.73	0.37	93%	84% Tryget [non-empty] mbox
2.87	2.79	3.76	0.12	84%	71% Peek item [empty] mbox
3.15	3.03	4.24	0.10	46%	40% Tryget [empty] mbox
0.34	0.24	1.33	0.10	43%	46% Waiting to get mbox
0.36	0.24	1.45	0.09	53%	37% Waiting to put mbox
4.49	4.24	10.91	0.41	96%	96% Delete mbox
12.67	12.36	19.52	0.43	96%	96% Put/Get mbox
0.87	0.85	1.45	0.05	93%	93% Init semaphore
2.74	2.55	4.48	0.18	50%	50% Post [0] semaphore
3.39	3.15	4.24	0.14	78%	50% Wait [1] semaphore
2.62	2.42	5.33	0.21	96%	65% Trywait [0] semaphore
2.76	2.67	3.27	0.08	46%	43% Trywait [1] semaphore
1.09	0.85	2.91	0.19	68%	56% Peek semaphore

Appendix B. Real-time characterization

0.97	0.73	3.39	0.17	90%	65% Destroy semaphore
13.09	12.85	16.12	0.19	84%	65% Post/Wait semaphore
1.57	1.45	3.88	0.15	96%	93% Create counter
0.91	0.73	2.18	0.16	46%	68% Get counter value
0.55	0.48	0.97	0.09	90%	62% Set counter value
4.19	4.00	5.82	0.13	84%	75% Tick counter
0.87	0.73	3.15	0.16	93%	93% Delete counter
2.50	2.30	5.21	0.18	81%	90% Create alarm
6.16	5.70	12.97	0.47	96%	71% Initialize alarm
0.50	0.36	1.70	0.11	62%	34% Disable alarm
5.16	4.85	8.73	0.29	78%	78% Enable alarm
1.18	1.09	2.30	0.12	84%	65% Delete alarm
5.22	5.09	7.39	0.14	96%	93% Tick counter [1 alarm]
52.37	52.12	52.73	0.20	37%	56% Tick counter [many alarms]
6.73	6.55	8.24	0.13	78%	68% Tick & fire counter [1 alarm]
108.65	108.61	109.21	0.07	87%	87% Tick & fire counters [>1 together]
54.25	54.06	54.79	0.11	65%	18% Tick & fire counters [>1 separately]
17.36	17.09	29.82	0.23	82%	57% Alarm latency [0 threads]
19.75	17.09	28.00	1.65	46%	40% Alarm latency [2 threads]
39.02	34.06	50.67	2.00	53%	15% Alarm latency [many threads]
29.31	28.36	105.09	1.27	98%	97% Alarm -> thread resume latency
5.08	3.88	11.15	0.00		Clock/interrupt latency
7.32	5.09	16.73	0.00		Clock DSR latency
6	0	380	(main stack: 820)		Thread stack used (992 total)
All done,	main stack		: stack used	820	size 4112
All done		:	Interrupt stack used	196	size 4096
All done		:	Idlethread stack used	360	size 2048

Timing complete - 29960 ms total
PASS:<Basic timing OK>
EXIT:<done>

Board: Hitachi Solution Engine 7751 SH4 (se7751)

Board: Hitachi Solution Engine 7751 SH4 (se7751)

CPU: Hitachi SH4/7751 162MHz

Startup, main stack	:	stack used	464	size	4112
Startup	:	Interrupt stack used	92	size	4096
Startup	:	Idlethread stack used	94	size	2048

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 1 'ticks' overhead
 ... this value will be factored out of all other measurements
 Clock interrupt took 14.27 microseconds (96 raw clock ticks)

Testing parameters:

Clock samples: 32
 Threads: 64
 Thread switches: 128
 Mutexes: 32
 Mailboxes: 32
 Semaphores: 32
 Scheduler operations: 128
 Counters: 32
 Alarms: 32

			Confidence			Function
Ave	Min	Max	Var	Ave	Min	
=====	=====	=====	=====	=====	=====	=====
8.06	5.63	12.15	1.37	46%	29%	Create thread
1.15	1.04	5.19	0.15	98%	98%	Yield thread [all suspended]
1.13	0.89	5.04	0.27	89%	62%	Suspend [suspended] thread
1.11	0.89	5.19	0.26	89%	71%	Resume thread
1.45	1.19	3.56	0.23	53%	53%	Set priority
0.21	0.15	1.19	0.10	90%	79%	Get priority
4.15	3.56	13.04	0.53	68%	64%	Kill [suspended] thread
1.12	1.04	3.70	0.12	98%	70%	Yield [no other] thread
1.75	1.33	8.00	0.38	59%	65%	Resume [suspended low prio] thread
1.10	0.89	4.59	0.25	87%	73%	Resume [runnable low prio] thread
1.59	1.33	5.93	0.33	81%	79%	Suspend [runnable] thread
1.13	1.04	4.30	0.13	98%	71%	Yield [only low prio] thread
1.09	0.89	3.56	0.21	89%	70%	Suspend [runnable->not runnable]
4.96	4.30	11.70	0.44	68%	39%	Kill [runnable] thread
1.95	1.48	8.00	0.34	75%	57%	Destroy [dead] thread
4.41	3.85	10.37	0.47	53%	57%	Destroy [runnable] thread
13.15	11.41	23.85	1.11	73%	39%	Resume [high priority] thread
3.10	2.96	6.22	0.11	41%	39%	Thread switch
0.13	0.00	1.33	0.06	74%	21%	Scheduler lock
0.76	0.74	1.78	0.03	96%	96%	Scheduler unlock [0 threads]
0.76	0.74	1.78	0.03	96%	96%	Scheduler unlock [1 suspended]
0.77	0.74	2.67	0.05	95%	95%	Scheduler unlock [many suspended]
0.76	0.74	2.37	0.04	95%	95%	Scheduler unlock [many low prio]
0.52	0.15	2.67	0.26	65%	34%	Init mutex
1.23	1.04	5.63	0.32	93%	93%	Lock [unlocked] mutex
1.45	1.19	5.33	0.31	90%	87%	Unlock [locked] mutex
1.13	0.89	4.15	0.28	90%	84%	Trylock [unlocked] mutex
1.00	0.89	2.96	0.17	87%	87%	Trylock [locked] mutex
0.37	0.30	1.78	0.13	90%	84%	Destroy mutex
9.09	8.59	12.59	0.43	71%	71%	Unlock/Lock mutex
0.93	0.59	4.30	0.40	84%	71%	Create mbox
0.26	0.00	1.19	0.17	71%	59%	Peek [empty] mbox
3.03	2.52	6.37	0.47	50%	59%	Put [first] mbox

Appendix B. Real-time characterization

0.23	0.00	0.74	0.14	68%	15% Peek [1 msg] mbox
2.93	2.52	4.74	0.46	71%	59% Put [second] mbox
0.22	0.00	0.59	0.13	68%	15% Peek [2 msgs] mbox
2.07	1.63	5.93	0.37	84%	59% Get [first] mbox
2.06	1.63	4.74	0.34	78%	59% Get [second] mbox
1.48	1.04	5.48	0.37	62%	53% Tryput [first] mbox
1.31	1.04	4.89	0.32	96%	75% Peek item [non-empty] mbox
1.47	1.04	5.78	0.38	84%	65% Tryget [non-empty] mbox
1.15	0.89	3.11	0.18	71%	56% Peek item [empty] mbox
1.20	1.04	3.85	0.21	93%	84% Tryget [empty] mbox
0.21	0.00	0.74	0.14	68%	18% Waiting to get mbox
0.19	0.00	0.44	0.10	43%	15% Waiting to put mbox
2.19	1.93	5.78	0.27	93%	71% Delete mbox
10.23	9.93	11.56	0.15	53%	37% Put/Get mbox
0.37	0.15	1.33	0.26	71%	71% Init semaphore
0.98	0.89	2.52	0.13	96%	68% Post [0] semaphore
1.08	0.89	3.26	0.15	68%	93% Wait [1] semaphore
0.98	0.89	3.41	0.16	93%	93% Trywait [0] semaphore
0.73	0.59	1.63	0.07	71%	25% Trywait [1] semaphore
0.33	0.30	1.33	0.07	93%	93% Peek semaphore
0.34	0.30	1.78	0.09	96%	96% Destroy semaphore
9.36	8.74	10.37	0.33	56%	31% Post/Wait semaphore
0.54	0.15	3.26	0.23	59%	37% Create counter
0.13	0.00	0.59	0.07	68%	25% Get counter value
0.14	0.00	0.59	0.07	68%	25% Set counter value
3.74	3.56	5.33	0.17	53%	75% Tick counter
0.32	0.15	2.07	0.12	71%	21% Delete counter
1.59	1.19	3.11	0.29	71%	43% Create alarm
1.89	1.48	6.37	0.44	87%	78% Initialize alarm
0.20	0.15	0.74	0.09	87%	84% Disable alarm
1.62	1.33	5.63	0.41	87%	84% Enable alarm
0.40	0.30	1.33	0.13	87%	62% Delete alarm
4.03	3.70	5.78	0.27	68%	56% Tick counter [1 alarm]
14.18	13.93	15.70	0.27	81%	75% Tick counter [many alarms]
4.81	4.59	5.93	0.13	81%	15% Tick & fire counter [1 alarm]
30.77	30.52	33.63	0.20	75%	65% Tick & fire counters [>1 together]
15.10	14.52	17.04	0.23	71%	3% Tick & fire counters [>1 separately]
8.78	8.59	18.22	0.20	97%	89% Alarm latency [0 threads]
11.29	9.33	17.48	1.02	56%	22% Alarm latency [2 threads]
18.70	15.70	26.37	1.45	54%	22% Alarm latency [many threads]
19.40	18.81	57.48	0.65	97%	97% Alarm -> thread resume latency
4.18	2.81	8.89	0.00		Clock/interrupt latency
3.98	2.52	11.56	0.00		Clock DSR latency
6	0	380	(main stack: 728)	Thread stack used (992 total)	
All done, main stack			:	stack used	728 size 4112
All done			:	Interrupt stack used	196 size 4096
All done			:	Idlethread stack used	360 size 2048

Timing complete - 29790 ms total

PASS:<Basic timing OK>

EXIT:<done>

Board: PC

Board: PC

CPU: 433MHz Celeron

```
Startup, main stack      : stack used 124 size 2912
Startup                  : Interrupt stack used 280 size 4108
Startup                  : Idlethread stack used 62 size 2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 8 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 6.75 microseconds (8 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
```

				Confidence			Function
Ave	Min	Max	Var	Ave	Min		
3.93	1.68	8.38	0.93	68%	3%	Create thread	
0.71	0.00	3.35	0.84	59%	59%	Yield thread [all suspended]	
0.65	0.00	5.03	0.84	64%	64%	Suspend [suspended] thread	
0.63	0.00	1.68	0.79	62%	62%	Resume thread	
0.76	0.00	1.68	0.83	54%	54%	Set priority	
0.39	0.00	1.68	0.60	76%	76%	Get priority	
1.34	0.00	6.70	0.67	73%	25%	Kill [suspended] thread	
0.68	0.00	1.68	0.81	59%	59%	Yield [no other] thread	
0.92	0.00	1.68	0.83	54%	45%	Resume [suspended low prio] thread	
0.63	0.00	1.68	0.79	62%	62%	Resume [runnable low prio] thread	
0.84	0.00	1.68	0.84	100%	50%	Suspend [runnable] thread	
0.73	0.00	1.68	0.82	56%	56%	Yield [only low prio] thread	
0.58	0.00	1.68	0.76	65%	65%	Suspend [runnable->not runnable]	
1.26	0.00	3.35	0.67	71%	26%	Kill [runnable] thread	
0.86	0.00	3.35	0.86	98%	50%	Destroy [dead] thread	

Appendix B. Real-time characterization

1.44	0.00	1.68	0.40	85%	14% Destroy [runnable] thread
4.45	3.35	6.70	0.89	53%	40% Resume [high priority] thread
1.62	0.00	1.68	0.10	96%	3% Thread switch
0.41	0.00	1.68	0.61	75%	75% Scheduler lock
0.48	0.00	1.68	0.69	71%	71% Scheduler unlock [0 threads]
0.59	0.00	1.68	0.76	64%	64% Scheduler unlock [1 suspended]
0.45	0.00	1.68	0.65	73%	73% Scheduler unlock [many suspended]
0.45	0.00	1.68	0.65	73%	73% Scheduler unlock [many low prio]
0.52	0.00	1.68	0.72	68%	68% Init mutex
0.79	0.00	5.03	0.93	96%	59% Lock [unlocked] mutex
0.84	0.00	5.03	0.94	96%	56% Unlock [locked] mutex
0.63	0.00	1.68	0.79	62%	62% Trylock [unlocked] mutex
0.52	0.00	1.68	0.72	68%	68% Trylock [locked] mutex
0.58	0.00	1.68	0.76	65%	65% Destroy mutex
3.40	3.35	5.03	0.10	96%	96% Unlock/Lock mutex
0.99	0.00	1.68	0.81	59%	40% Create mbox
0.47	0.00	1.68	0.68	71%	71% Peek [empty] mbox
0.79	0.00	5.03	0.93	96%	59% Put [first] mbox
0.42	0.00	1.68	0.63	75%	75% Peek [1 msg] mbox
0.79	0.00	1.68	0.83	53%	53% Put [second] mbox
0.37	0.00	1.68	0.57	78%	78% Peek [2 msgs] mbox
0.73	0.00	3.35	0.87	59%	59% Get [first] mbox
0.73	0.00	1.68	0.82	56%	56% Get [second] mbox
0.79	0.00	3.35	0.88	56%	56% Tryput [first] mbox
0.68	0.00	3.35	0.85	62%	62% Peek item [non-empty] mbox
0.73	0.00	3.35	0.87	59%	59% Tryget [non-empty] mbox
0.63	0.00	1.68	0.79	62%	62% Peek item [empty] mbox
0.68	0.00	1.68	0.81	59%	59% Tryget [empty] mbox
0.26	0.00	1.68	0.44	84%	84% Waiting to get mbox
0.63	0.00	1.68	0.79	62%	62% Waiting to put mbox
0.73	0.00	3.35	0.87	59%	59% Delete mbox
3.25	1.68	3.35	0.20	93%	6% Put/Get mbox
0.63	0.00	1.68	0.79	62%	62% Init semaphore
0.63	0.00	1.68	0.79	62%	62% Post [0] semaphore
0.63	0.00	1.68	0.79	62%	62% Wait [1] semaphore
0.52	0.00	1.68	0.72	68%	68% Trywait [0] semaphore
0.52	0.00	1.68	0.72	68%	68% Trywait [1] semaphore
0.52	0.00	1.68	0.72	68%	68% Peek semaphore
0.21	0.00	1.68	0.37	87%	87% Destroy semaphore
3.30	1.68	3.35	0.10	96%	3% Post/Wait semaphore
0.79	0.00	3.35	0.88	56%	56% Create counter
0.42	0.00	1.68	0.63	75%	75% Get counter value
0.37	0.00	1.68	0.57	78%	78% Set counter value
0.73	0.00	1.68	0.82	56%	56% Tick counter
0.63	0.00	1.68	0.79	62%	62% Delete counter
0.89	0.00	3.35	0.89	96%	50% Create alarm
0.84	0.00	1.68	0.84	100%	50% Initialize alarm
0.52	0.00	1.68	0.72	68%	68% Disable alarm
0.89	0.00	3.35	0.89	96%	50% Enable alarm

```

0.58    0.00    1.68    0.76    65%    65% Delete alarm
0.63    0.00    1.68    0.79    62%    62% Tick counter [1 alarm]
5.03    3.35    6.70    0.10    93%     3% Tick counter [many alarms]
0.94    0.00    1.68    0.82    56%    43% Tick & fire counter [1 alarm]
11.16   10.06   11.73    0.76    65%    34% Tick & fire counters [>1 together]
5.19    5.03    6.70    0.28    90%    90% Tick & fire counters [>1 separately]
0.01    0.00    1.68    0.03    99%    99% Alarm latency [0 threads]
0.13    0.00    1.68    0.24    92%    92% Alarm latency [2 threads]
0.94    0.00    3.35    0.85    53%    45% Alarm latency [many threads]
1.75    1.68    6.70    0.15    96%    96% Alarm -> thread resume latency

41      0      368 (main stack: 1036) Thread stack used (1712 total)
All done, main stack      : stack used 1036 size 2912
All done      : Interrupt stack used 368 size 4108
All done      : Idlethread stack used 288 size 2048

Timing complete - 28520 ms total

PASS:<Basic timing OK>
EXIT:<done>

```

Board: NEC V850 Cosmo Evaluation Board

Board: NEC V850 Cosmo Evaluation Board

CPU: NEC CEB-V850/SA1 17MHz

```

Startup, main stack : stack used 552 size 2936
Startup             : Interrupt stack used 120 size 4096
Startup             : Idlethread stack used 206 size 2048

```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

```

Reading the hardware clock takes 27 'ticks' overhead
... this value will be factored out of all other measurements
Clock interrupt took 280.04 microseconds (1190 raw clock ticks)

```

Testing parameters:

```

Clock samples:      32
Threads:            7
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32

```

```

                                Confidence
Ave    Min    Max    Var Ave Min Function

```

Appendix B. Real-time characterization

=====	=====	=====	=====	=====	=====	=====
288.71	280.24	297.18	4.84	42%	28%	Create thread
70.76	70.59	70.82	0.10	71%	28%	Yield thread [all suspended]
59.06	59.06	59.06	0.00	100%	100%	Suspend [suspended] thread
60.00	60.00	60.00	0.00	100%	100%	Resume thread
77.38	77.18	77.41	0.06	85%	14%	Set priority
3.13	3.06	3.29	0.10	71%	71%	Get priority
187.46	187.29	187.53	0.10	71%	28%	Kill [suspended] thread
70.76	70.59	70.82	0.10	71%	28%	Yield [no other] thread
104.40	103.29	104.71	0.32	85%	14%	Resume [suspended low prio] thread
59.06	59.06	59.06	0.00	100%	100%	Resume [runnable low prio] thread
97.11	91.06	98.12	1.73	85%	14%	Suspend [runnable] thread
70.76	70.59	70.82	0.10	71%	28%	Yield [only low prio] thread
59.06	59.06	59.06	0.00	100%	100%	Suspend [runnable->not runnable]
187.46	187.29	187.53	0.10	71%	28%	Kill [runnable] thread
95.63	95.29	97.18	0.44	85%	85%	Destroy [dead] thread
241.28	236.94	242.12	1.24	85%	14%	Destroy [runnable] thread
378.55	370.35	427.06	13.86	85%	85%	Resume [high priority] thread
198.77	183.76	452.94	18.77	96%	96%	Thread switch
2.59	2.59	2.59	0.00	100%	100%	Scheduler lock
41.29	41.18	41.41	0.12	100%	50%	Scheduler unlock [0 threads]
40.82	40.71	40.94	0.12	100%	50%	Scheduler unlock [1 suspended]
41.29	41.18	41.41	0.12	100%	50%	Scheduler unlock [many suspended]
41.29	41.18	41.41	0.12	100%	50%	Scheduler unlock [many low prio]
17.94	17.88	18.12	0.09	75%	75%	Init mutex
68.71	68.71	68.71	0.00	100%	100%	Lock [unlocked] mutex
72.10	72.00	73.41	0.15	96%	71%	Unlock [locked] mutex
57.88	57.88	57.88	0.00	100%	100%	Trylock [unlocked] mutex
52.24	52.24	52.24	0.00	100%	100%	Trylock [locked] mutex
12.41	12.24	12.47	0.09	75%	25%	Destroy mutex
427.06	427.06	427.06	0.00	100%	100%	Unlock/Lock mutex
34.94	34.82	35.06	0.12	100%	50%	Create mbox
0.76	0.71	0.94	0.09	75%	75%	Peek [empty] mbox
75.29	75.29	75.29	0.00	100%	100%	Put [first] mbox
1.24	1.18	1.41	0.09	75%	75%	Peek [1 msg] mbox
75.76	75.76	75.76	0.00	100%	100%	Put [second] mbox
0.76	0.71	0.94	0.09	75%	75%	Peek [2 msgs] mbox
80.12	80.00	80.24	0.12	100%	50%	Get [first] mbox
79.65	79.53	79.76	0.12	100%	50%	Get [second] mbox
70.12	70.12	70.12	0.00	100%	100%	Tryput [first] mbox
65.76	65.65	65.88	0.12	100%	50%	Peek item [non-empty] mbox
78.00	77.88	78.12	0.12	100%	50%	Tryget [non-empty] mbox
63.12	63.06	63.29	0.09	75%	75%	Peek item [empty] mbox
67.82	67.76	68.00	0.09	75%	75%	Tryget [empty] mbox
1.94	1.88	2.12	0.09	75%	75%	Waiting to get mbox
1.47	1.41	1.65	0.09	75%	75%	Waiting to put mbox
75.59	75.53	75.76	0.09	75%	75%	Delete mbox
252.76	252.71	252.94	0.09	75%	75%	Put/Get mbox
20.24	20.24	20.24	0.00	100%	100%	Init semaphore
54.35	54.35	54.35	0.00	100%	100%	Post [0] semaphore

66.59	66.59	66.59	0.00	100%	100%	Wait [1] semaphore
52.24	52.24	52.24	0.00	100%	100%	Trywait [0] semaphore
53.41	53.41	53.41	0.00	100%	100%	Trywait [1] semaphore
10.65	10.59	10.82	0.09	75%	75%	Peek semaphore
12.65	12.47	12.71	0.09	75%	25%	Destroy semaphore
276.94	276.94	276.94	0.00	100%	100%	Post/Wait semaphore
14.94	14.82	15.06	0.12	100%	50%	Create counter
2.18	2.12	2.35	0.09	75%	75%	Get counter value
3.06	3.06	3.06	0.00	100%	100%	Set counter value
78.12	78.12	78.12	0.00	100%	100%	Tick counter
13.82	13.65	13.88	0.09	75%	25%	Delete counter
26.94	26.82	27.06	0.12	100%	50%	Create alarm
104.18	104.00	104.24	0.09	75%	25%	Initialize alarm
7.65	7.53	7.76	0.12	100%	50%	Disable alarm
104.94	104.94	104.94	0.00	100%	100%	Enable alarm
19.47	19.29	19.53	0.09	75%	25%	Delete alarm
88.53	88.47	88.71	0.09	75%	75%	Tick counter [1 alarm]
418.61	411.29	645.41	14.17	96%	96%	Tick counter [many alarms]
139.59	139.53	139.76	0.09	75%	75%	Tick & fire counter [1 alarm]
2150.21	2096.71	2367.53	83.59	78%	78%	Tick & fire counters [>1 together]
478.15	462.35	733.41	29.61	93%	93%	Tick & fire counters [>1 separately]
219.89	218.59	369.88	2.34	99%	99%	Alarm latency [0 threads]
292.11	218.59	371.53	37.85	50%	25%	Alarm latency [2 threads]
292.96	218.59	370.59	38.12	49%	25%	Alarm latency [many threads]
540.90	495.76	1677.41	17.76	98%	0%	Alarm -> thread resume latency
79.01	78.59	104.71	0.00			Clock/interrupt latency
123.41	85.88	1982.82	0.00			Clock DSR latency
522	516	536	(main stack: 1124) Thread stack used (1912 total)			
All done, main stack : stack used 1124 size 2936						
All done : Interrupt stack used 288 size 4096						
All done : Idlethread stack used 488 size 2048						
Timing complete - 32540 ms total						

Board: NEC V850 Cosmo Evaluation Board

Board: NEC V850 Cosmo Evaluation Board

CPU: NEC CEB-V850/SB1 16MHz (in internal Flash)

Startup, main stack	:	stack used	572	size	2936
Startup	:	Interrupt stack used	132	size	4096
Startup	:	Idlethread stack used	210	size	2048

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Appendix B. Real-time characterization

Reading the hardware clock takes 8 'ticks' overhead
 ... this value will be factored out of all other measurements
 Clock interrupt took 118.15 microseconds (472 raw clock ticks)

Testing parameters:

```

Clock samples:      32
Threads:            7
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Alarms:             32
  
```

				Confidence			Function
Ave	Min	Max	Var	Ave	Min	Function	
=====	=====	=====	=====	=====	=====	=====	
113.68	111.00	116.50	1.63	42%	28%	Create thread	
30.00	30.00	30.00	0.00	100%	100%	Yield thread [all suspended]	
29.57	29.50	29.75	0.10	71%	71%	Suspend [suspended] thread	
27.43	27.25	27.50	0.10	71%	28%	Resume thread	
34.11	34.00	34.25	0.12	57%	57%	Set priority	
1.57	1.50	1.75	0.10	71%	71%	Get priority	
72.96	72.75	73.00	0.06	85%	14%	Kill [suspended] thread	
30.00	30.00	30.00	0.00	100%	100%	Yield [no other] thread	
42.75	42.75	42.75	0.00	100%	100%	Resume [suspended low prio] thread	
27.00	27.00	27.00	0.00	100%	100%	Resume [runnable low prio] thread	
43.64	41.25	44.25	0.68	85%	14%	Suspend [runnable] thread	
30.00	30.00	30.00	0.00	100%	100%	Yield [only low prio] thread	
29.57	29.50	29.75	0.10	71%	71%	Suspend [runnable->not runnable]	
72.93	72.75	73.00	0.10	71%	28%	Kill [runnable] thread	
44.89	44.75	45.75	0.24	85%	85%	Destroy [dead] thread	
103.00	101.50	103.25	0.43	85%	14%	Destroy [runnable] thread	
175.21	171.50	197.50	6.37	85%	85%	Resume [high priority] thread	
84.11	79.50	197.25	1.77	98%	0%	Thread switch	
1.00	1.00	1.00	0.00	100%	100%	Scheduler lock	
20.06	20.00	20.25	0.09	75%	75%	Scheduler unlock [0 threads]	
20.00	20.00	20.00	0.00	100%	100%	Scheduler unlock [1 suspended]	
20.06	20.00	20.25	0.09	75%	75%	Scheduler unlock [many suspended]	
20.06	20.00	20.25	0.09	75%	75%	Scheduler unlock [many low prio]	
4.00	4.00	4.00	0.00	100%	100%	Init mutex	
33.00	33.00	33.00	0.00	100%	100%	Lock [unlocked] mutex	
36.77	36.75	37.25	0.03	96%	96%	Unlock [locked] mutex	
28.13	28.00	28.25	0.13	100%	50%	Trylock [unlocked] mutex	
25.13	25.00	25.25	0.13	100%	50%	Trylock [locked] mutex	
4.88	4.75	5.00	0.13	100%	50%	Destroy mutex	
187.00	187.00	187.00	0.00	100%	100%	Unlock/Lock mutex	
10.00	10.00	10.00	0.00	100%	100%	Create mbox	
0.69	0.50	0.75	0.09	75%	25%	Peek [empty] mbox	

34.75	34.75	34.75	0.00	100%	100%	Put [first] mbox
0.69	0.50	0.75	0.09	75%	25%	Peek [1 msg] mbox
35.00	35.00	35.00	0.00	100%	100%	Put [second] mbox
0.69	0.50	0.75	0.09	75%	25%	Peek [2 msgs] mbox
36.00	36.00	36.00	0.00	100%	100%	Get [first] mbox
36.00	36.00	36.00	0.00	100%	100%	Get [second] mbox
31.00	31.00	31.00	0.00	100%	100%	Tryput [first] mbox
29.50	29.50	29.50	0.00	100%	100%	Peek item [non-empty] mbox
35.25	35.25	35.25	0.00	100%	100%	Tryget [non-empty] mbox
27.69	27.50	27.75	0.09	75%	25%	Peek item [empty] mbox
31.06	31.00	31.25	0.09	75%	75%	Tryget [empty] mbox
0.94	0.75	1.00	0.09	75%	25%	Waiting to get mbox
0.94	0.75	1.00	0.09	75%	25%	Waiting to put mbox
37.81	37.75	38.00	0.09	75%	75%	Delete mbox
112.00	112.00	112.00	0.00	100%	100%	Put/Get mbox
3.19	3.00	3.25	0.09	75%	25%	Init semaphore
25.38	25.25	25.50	0.13	100%	50%	Post [0] semaphore
32.63	32.50	32.75	0.13	100%	50%	Wait [1] semaphore
24.25	24.25	24.25	0.00	100%	100%	Trywait [0] semaphore
25.00	25.00	25.00	0.00	100%	100%	Trywait [1] semaphore
4.00	4.00	4.00	0.00	100%	100%	Peek semaphore
4.88	4.75	5.00	0.13	100%	50%	Destroy semaphore
124.50	124.50	124.50	0.00	100%	100%	Post/Wait semaphore
6.50	6.50	6.50	0.00	100%	100%	Create counter
1.25	1.25	1.25	0.00	100%	100%	Get counter value
1.44	1.25	1.50	0.09	75%	25%	Set counter value
36.25	36.25	36.25	0.00	100%	100%	Tick counter
5.25	5.25	5.25	0.00	100%	100%	Delete counter
12.25	12.25	12.25	0.00	100%	100%	Create alarm
49.13	49.00	49.25	0.13	100%	50%	Initialize alarm
2.81	2.75	3.00	0.09	75%	75%	Disable alarm
48.50	48.50	48.50	0.00	100%	100%	Enable alarm
8.25	8.25	8.25	0.00	100%	100%	Delete alarm
46.50	46.50	46.50	0.00	100%	100%	Tick counter [1 alarm]
485.42	482.25	580.00	5.91	96%	96%	Tick counter [many alarms]
64.00	64.00	64.00	0.00	100%	100%	Tick & fire counter [1 alarm]
1109.76	1100.50	1198.00	16.53	90%	90%	Tick & fire counters [>1 together]
505.85	502.00	621.00	7.20	96%	96%	Tick & fire counters [>1 separately]
96.26	95.75	161.25	1.02	99%	99%	Alarm latency [0 threads]
159.20	95.75	160.75	2.52	97%	0%	Alarm latency [2 threads]
159.73	110.50	161.75	1.53	97%	0%	Alarm latency [many threads]
218.45	211.25	445.75	3.55	97%	1%	Alarm -> thread resume latency
28.24	25.25	43.25	0.00			Clock/interrupt latency
60.15	40.50	221.50	0.00			Clock DSR latency
472	424	572	(main stack: 1052) Thread stack used (1912 total)			
All done, main stack			:	stack used	1052 size	2936
All done			:	Interrupt stack used	280 size	4096
All done			:	Idlethread stack used	516 size	2048

Appendix B. Real-time characterization

Timing complete - 30590 ms total

PASS:<Basic timing OK>

EXIT:<done>

Board: ARM Industrial Module AIM711 (S3C4510)

Board: ARM Industrial Module AIM711 (S3C4510)

CPU : S3C4510B (ARM7TDMI core), 50MHz

8MB RAM, 2MB Flash

```
Startup, main stack :          stack used    82 size  2400
Startup              : Interrupt stack used  134 size  4096
Startup              : Idlethread stack used   91 size  2048
```

eCos Kernel Timings

Notes: all times are in microseconds (.000001) unless otherwise stated

Reading the hardware clock takes 33 'ticks' overhead

... this value will be factored out of all other measurements

Clock interrupt took 17.68 microseconds (884 raw clock ticks)

Testing parameters:

```
Clock samples:      32
Threads:            64
Thread switches:    128
Mutexes:            32
Mailboxes:          32
Semaphores:         32
Scheduler operations: 128
Counters:           32
Flags:              32
Alarms:             32
```

				Confidence			Function
Ave	Min	Max	Var	Ave	Min		
=====	=====	=====	=====	=====	=====	=====	
22.99	15.24	36.98	4.01	50%	26%		Create thread
2.96	2.88	8.30	0.17	98%	98%		Yield thread [all suspended]
3.57	3.36	8.76	0.26	93%	71%		Suspend [suspended] thread
3.64	3.00	8.74	0.33	65%	20%		Resume thread
5.44	4.78	15.10	0.42	75%	26%		Set priority
0.77	0.20	1.98	0.25	59%	17%		Get priority
14.46	12.40	33.02	1.03	85%	9%		Kill [suspended] thread
2.95	2.88	7.44	0.14	98%	98%		Yield [no other] thread
6.73	5.40	15.60	0.44	78%	6%		Resume [suspended low prio] thread
3.59	2.98	7.18	0.28	56%	21%		Resume [runnable low prio] thread
5.77	4.78	13.46	0.44	71%	18%		Suspend [runnable] thread
2.97	2.88	8.86	0.18	98%	98%		Yield [only low prio] thread
3.40	2.86	6.26	0.26	59%	17%		Suspend [runnable->not runnable]
14.15	12.08	30.54	1.09	78%	23%		Kill [runnable] thread

Appendix B. Real-time characterization

11.00	9.74	23.38	0.75	70%	31%	Destroy [dead] thread
20.35	17.72	43.00	1.26	73%	14%	Destroy [runnable] thread
23.77	21.02	45.38	1.59	68%	35%	Resume [high priority] thread
8.40	8.30	15.38	0.18	89%	89%	Thread switch
0.10	0.08	1.52	0.03	92%	92%	Scheduler lock
2.01	1.98	3.80	0.06	92%	92%	Scheduler unlock [0 threads]
2.01	1.98	3.80	0.06	92%	92%	Scheduler unlock [1 suspended]
2.01	1.98	4.08	0.06	92%	92%	Scheduler unlock [many suspended]
2.01	1.98	3.68	0.05	92%	92%	Scheduler unlock [many low prio]
0.67	0.54	3.90	0.21	96%	96%	Init mutex
4.55	4.14	12.40	0.53	96%	87%	Lock [unlocked] mutex
4.84	4.12	12.78	0.56	65%	56%	Unlock [locked] mutex
3.72	3.18	8.86	0.41	68%	56%	Trylock [unlocked] mutex
3.22	2.76	5.38	0.26	65%	28%	Trylock [locked] mutex
0.49	0.34	3.26	0.26	93%	84%	Destroy mutex
33.13	32.42	43.64	0.66	90%	81%	Unlock/Lock mutex
1.21	1.06	5.12	0.25	96%	96%	Create mbox
0.63	0.46	2.66	0.22	96%	71%	Peek [empty] mbox
4.57	3.64	11.12	0.50	75%	18%	Put [first] mbox
0.52	0.10	2.74	0.23	62%	18%	Peek [1 msg] mbox
5.39	4.46	12.00	0.56	75%	43%	Put [second] mbox
0.51	0.10	2.38	0.22	62%	18%	Peek [2 msgs] mbox
5.06	4.00	13.86	0.60	81%	18%	Get [first] mbox
5.01	4.36	9.20	0.38	68%	25%	Get [second] mbox
5.56	4.70	11.22	0.55	75%	37%	Tryput [first] mbox
4.25	3.14	10.64	0.49	75%	9%	Peek item [non-empty] mbox
5.10	3.82	14.02	0.78	78%	40%	Tryget [non-empty] mbox
3.86	3.12	9.72	0.47	81%	21%	Peek item [empty] mbox
4.13	3.28	11.20	0.54	87%	59%	Tryget [empty] mbox
0.60	0.14	2.34	0.22	68%	9%	Waiting to get mbox
0.61	0.14	2.90	0.27	78%	15%	Waiting to put mbox
4.51	3.66	11.20	0.53	84%	50%	Delete mbox
26.55	26.00	31.46	0.37	78%	40%	Put/Get mbox
0.53	0.44	2.68	0.15	96%	90%	Init semaphore
3.08	2.76	5.02	0.29	43%	46%	Post [0] semaphore
3.64	3.20	7.72	0.40	53%	50%	Wait [1] semaphore
3.08	2.66	7.40	0.39	50%	50%	Trywait [0] semaphore
2.72	2.62	5.88	0.20	96%	96%	Trywait [1] semaphore
0.85	0.52	3.30	0.32	50%	50%	Peek semaphore
0.80	0.34	3.74	0.39	46%	37%	Destroy semaphore
21.87	21.54	25.64	0.28	68%	65%	Post/Wait semaphore
1.18	1.04	4.92	0.24	96%	96%	Create counter
0.69	0.52	2.84	0.24	93%	71%	Get counter value
0.26	0.14	1.76	0.18	78%	78%	Set counter value
3.73	3.24	5.62	0.14	78%	12%	Tick counter
0.79	0.36	3.58	0.19	78%	15%	Delete counter
0.53	0.44	3.06	0.17	96%	90%	Init flag
3.49	3.02	9.28	0.45	53%	50%	Destroy flag
2.93	2.52	7.42	0.39	50%	46%	Mask bits in flag

Appendix B. Real-time characterization

3.58	3.12	9.38	0.46	50%	50%	Set bits in flag [no waiters]
7.48	7.22	12.90	0.35	96%	96%	Wait for flag [AND]
4.92	4.66	11.22	0.39	96%	96%	Wait for flag [OR]
4.58	4.30	11.66	0.44	96%	96%	Wait for flag [AND/CLR]
4.39	4.12	11.02	0.43	96%	96%	Wait for flag [OR/CLR]
0.06	0.00	1.40	0.11	87%	87%	Peek on flag
1.82	1.58	8.02	0.40	96%	96%	Create alarm
7.27	6.54	17.86	0.77	93%	87%	Initialize alarm
3.30	2.58	7.28	0.60	56%	71%	Disable alarm
7.60	5.82	14.72	0.84	81%	12%	Enable alarm
3.86	3.06	9.20	0.67	53%	65%	Delete alarm
4.03	3.90	7.18	0.23	96%	90%	Tick counter [1 alarm]
25.12	24.98	28.82	0.24	96%	93%	Tick counter [many alarms]
7.92	7.64	14.00	0.40	96%	96%	Tick & fire counter [1 alarm]
155.10	154.42	161.04	0.37	90%	6%	Tick & fire counters [>1 together]
29.27	29.02	35.48	0.42	96%	93%	Tick & fire counters [>1 separately]
17.87	17.32	49.30	0.56	98%	97%	Alarm latency [0 threads]
24.39	22.02	63.60	1.43	57%	19%	Alarm latency [2 threads]
55.33	52.72	62.44	1.11	67%	20%	Alarm latency [many threads]
37.98	36.54	170.56	2.17	97%	97%	Alarm -> thread resume latency
29	0	259	(main stack: 805)	Thread stack used (1120 total)		
All done, main stack :			stack used	805	size	2400
All done			: Interrupt stack used	163	size	4096
All done			: Idlethread stack used	239	size	2048
Timing complete - 28880 ms total						
PASS:<Basic timing OK>						
EXIT:<done>						

Appendix C. GNU General Public License

Copyright (C) 1989, 1991 Free Software Foundation, Inc.
59 Temple Place, Suite 330, Boston, MA 02111-1307 USA
Everyone is permitted to copy and distribute verbatim copies
of this license document, but changing it is not allowed.

Preamble

The licenses for most software are designed to take away your freedom to share and change it. By contrast, the GNU General Public License is intended to guarantee your freedom to share and change free software--to make sure the software is free for all its users. This General Public License applies to most of the Free Software Foundation's software and to any other program whose authors commit to using it. (Some other Free Software Foundation software is covered by the GNU Library General Public License instead.) You can apply it to your programs, too.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for this service if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs; and that you know you can do these things.

To protect your rights, we need to make restrictions that forbid anyone to deny you these rights or to ask you to surrender the rights. These restrictions translate to certain responsibilities for you if you distribute copies of the software, or if you modify it.

For example, if you distribute copies of such a program, whether gratis or for a fee, you must give the recipients all the rights that you have. You must make sure that they, too, receive or can get the source code. And you must show them these terms so they know their rights.

We protect your rights with two steps: (1) copyright the software, and (2) offer you this license which gives you legal permission to copy, distribute and/or modify the software.

Also, for each author's protection and ours, we want to make certain that everyone understands that there is no warranty for this free software. If the software is modified by someone else and passed on, we want its recipients to know that what they have is not the original, so that any problems introduced by others will not reflect on the original authors' reputations.

Finally, any free program is threatened constantly by software patents. We wish to avoid the danger that redistributors of a free program will individually obtain patent licenses, in effect making the program proprietary. To prevent this, we have made it clear that any patent must be licensed for everyone's free use or not licensed at all.

The precise terms and conditions for copying, distribution and modification follow.

GNU GENERAL PUBLIC LICENSE
TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

0. This License applies to any program or other work which contains a notice placed by the copyright holder saying it may be distributed under the terms of this General Public License. The "Program", below, refers to any such program or work, and a "work based on the Program" means either the Program or any derivative work under copyright law: that is to say, a work containing the Program or a portion of it, either verbatim or with modifications and/or translated into another language. (Hereinafter, translation is included without limitation in the term "modification".) Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not covered by this License; they are outside its scope. The act of running the Program is not restricted, and the output from the Program is covered only if its contents constitute a work based on the Program (independent of having been made by running the Program). Whether that is true depends on what the Program does.

1. You may copy and distribute verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice and disclaimer of warranty; keep intact all the notices that refer to this License and to the absence of any warranty; and give any other recipients of the Program a copy of this License along with the Program.

You may charge a fee for the physical act of transferring a copy, and you may at your option offer warranty protection in exchange for a fee.

2. You may modify your copy or copies of the Program or any portion of it, thus forming a work based on the Program, and copy and distribute such modifications or work under the terms of Section 1 above, provided that you also meet all of these conditions:

- a) You must cause the modified files to carry prominent notices stating that you changed the files and the date of any change.

b) You must cause any work that you distribute or publish, that in whole or in part contains or is derived from the Program or any part thereof, to be licensed as a whole at no charge to all third parties under the terms of this License.

c) If the modified program normally reads commands interactively when run, you must cause it, when started running for such interactive use in the most ordinary way, to print or display an announcement including an appropriate copyright notice and a notice that there is no warranty (or else, saying that you provide a warranty) and that users may redistribute the program under these conditions, and telling the user how to view a copy of this License. (Exception: if the Program itself is interactive but does not normally print such an announcement, your work based on the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole. If identifiable sections of that work are not derived from the Program, and can be reasonably considered independent and separate works in themselves, then this License, and its terms, do not apply to those sections when you distribute them as separate works. But when you distribute the same sections as part of a whole which is a work based on the Program, the distribution of the whole must be on the terms of this License, whose permissions for other licensees extend to the entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest your rights to work written entirely by you; rather, the intent is to exercise the right to control the distribution of derivative or collective works based on the Program.

In addition, mere aggregation of another work not based on the Program with the Program (or with a work based on the Program) on a volume of a storage or distribution medium does not bring the other work under the scope of this License.

3. You may copy and distribute the Program (or a work based on it, under Section 2) in object code or executable form under the terms of Sections 1 and 2 above provided that you also do one of the following:

a) Accompany it with the complete corresponding machine-readable source code, which must be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

b) Accompany it with a written offer, valid for at least three years, to give any third party, for a charge no more than your cost of physically performing source distribution, a complete

machine-readable copy of the corresponding source code, to be distributed under the terms of Sections 1 and 2 above on a medium customarily used for software interchange; or,

c) Accompany it with the information you received as to the offer to distribute corresponding source code. (This alternative is allowed only for noncommercial distribution and only if you received the program in object code or executable form with such an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for making modifications to it. For an executable work, complete source code means all the source code for all modules it contains, plus any associated interface definition files, plus the scripts used to control compilation and installation of the executable. However, as a special exception, the source code distributed need not include anything that is normally distributed (in either source or binary form) with the major components (compiler, kernel, and so on) of the operating system on which the executable runs, unless that component itself accompanies the executable.

If distribution of executable or object code is made by offering access to copy from a designated place, then offering equivalent access to copy the source code from the same place counts as distribution of the source code, even though third parties are not compelled to copy the source along with the object code.

4. You may not copy, modify, sublicense, or distribute the Program except as expressly provided under this License. Any attempt otherwise to copy, modify, sublicense or distribute the Program is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

5. You are not required to accept this License, since you have not signed it. However, nothing else grants you permission to modify or distribute the Program or its derivative works. These actions are prohibited by law if you do not accept this License. Therefore, by modifying or distributing the Program (or any work based on the Program), you indicate your acceptance of this License to do so, and all its terms and conditions for copying, distributing or modifying the Program or works based on it.

6. Each time you redistribute the Program (or any work based on the Program), the recipient automatically receives a license from the original licensor to copy, distribute or modify the Program subject to these terms and conditions. You may not impose any further

restrictions on the recipients' exercise of the rights granted herein. You are not responsible for enforcing compliance by third parties to this License.

7. If, as a consequence of a court judgment or allegation of patent infringement or for any other reason (not limited to patent issues), conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot distribute so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not distribute the Program at all. For example, if a patent license would not permit royalty-free redistribution of the Program by all those who receive copies directly or indirectly through you, then the only way you could satisfy both it and this License would be to refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under any particular circumstance, the balance of the section is intended to apply and the section as a whole is intended to apply in other circumstances.

It is not the purpose of this section to induce you to infringe any patents or other property right claims or to contest validity of any such claims; this section has the sole purpose of protecting the integrity of the free software distribution system, which is implemented by public license practices. Many people have made generous contributions to the wide range of software distributed through that system in reliance on consistent application of that system; it is up to the author/donor to decide if he or she is willing to distribute software through any other system and a licensee cannot impose that choice.

This section is intended to make thoroughly clear what is believed to be a consequence of the rest of this License.

8. If the distribution and/or use of the Program is restricted in certain countries either by patents or by copyrighted interfaces, the original copyright holder who places the Program under this License may add an explicit geographical distribution limitation excluding those countries, so that distribution is permitted only in or among countries not thus excluded. In such case, this License incorporates the limitation as if written in the body of this License.

9. The Free Software Foundation may publish revised and/or new versions of the General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies a version number of this License which applies to it and "any later version", you have the option of following the terms and conditions either of that version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of this License, you may choose any version ever published by the Free Software Foundation.

10. If you wish to incorporate parts of the Program into other free programs whose distribution conditions are different, write to the author to ask for permission. For software which is copyrighted by the Free Software Foundation, write to the Free Software Foundation; we sometimes make exceptions for this. Our decision will be guided by the two goals of preserving the free status of all derivatives of our free software and of promoting the sharing and reuse of software generally.

NO WARRANTY

11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

END OF TERMS AND CONDITIONS

How to Apply These Terms to Your New Programs

If you develop a new program, and you want it to be of the greatest possible use to the public, the best way to achieve this is to make it free software which everyone can redistribute and change under these terms.

To do so, attach the following notices to the program. It is safest

to attach them to the start of each source file to most effectively convey the exclusion of warranty; and each file should have at least the "copyright" line and a pointer to where the full notice is found.

<one line to give the program's name and a brief idea of what it does.>
Copyright (C) <year> <name of author>

This program is free software; you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation; either version 2 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with this program; if not, write to the Free Software Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307 USA

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this when it starts in an interactive mode:

Gnomovision version 69, Copyright (C) year name of author
Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type 'show w'.
This is free software, and you are welcome to redistribute it under certain conditions; type 'show c' for details.

The hypothetical commands 'show w' and 'show c' should show the appropriate parts of the General Public License. Of course, the commands you use may be called something other than 'show w' and 'show c'; they could even be mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your school, if any, to sign a "copyright disclaimer" for the program, if necessary. Here is a sample; alter the names:

Yoyodyne, Inc., hereby disclaims all copyright interest in the program
'Gnomovision' (which makes passes at compilers) written by James Hacker.

<signature of Ty Coon>, 1 April 1989
Ty Coon, President of Vice

This General Public License does not permit incorporating your program into

Appendix C. GNU General Public License

proprietary programs. If your program is a subroutine library, you may consider it more useful to permit linking proprietary applications with the library. If this is what you want to do, use the GNU Library General Public License instead of this License.